Red Hat Enterprise Linux 6

虚拟化管理指南

管理虚拟环境

Last Updated: 2023-02-26

Red Hat Enterprise Linux 6 虚拟化管理指南

管理虚拟环境

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

虚拟化管理指南涵盖了主机物理计算机、网络、存储、设备和客户机虚拟机管理和故障排除的管理。注：本文档的开发存在，可能进行大量更改，并且仅作为技术预览提供。包括的信息和说明不应被视为完成，应谨慎使用。 要扩展您的专业知识，您可能还对红帽 虚拟化(RH318)培训课程感兴趣。

目录

第 1 章 服务器最佳实践

以下任务和提示可帮助您 提高 Red Hat Enterprise Linux 主机的性能。有关其他提示，请参阅 *Red Hat Enterprise Linux Virtualization Tuning and Optimization Guide*

- 在 enforcing 模式下运行 SELinux。使用 **setenforce** 命令，将 SELinux 设置为在 enforcing 模式下运行。

```
# setenforce 1
```

- 删除或禁用任何不必要的服务，如 **AutoFS**、**NFS**、**FTP**、**HTTP**、**NIS**、**teld**、**sendmail** 等。
- 仅添加服务器上平台管理所需的最少用户帐户数量，并删除不必要的用户帐户。
- 避免在主机上运行任何正式的应用程序。在主机上运行应用程序可能会影响虚拟机性能，并可能会影响服务器稳定性。任何可能导致服务器崩溃的应用程序都将导致服务器中的所有虚拟机停机。
- 将中央位置用于虚拟机安装和镜像。虚拟机映像应存储在 **/var/lib/libvirt/images/** 下。如果您在虚拟机镜像中使用其他目录，请确保将目录 添加到您的 SELinux 策略中，并在开始安装前重新标记它。强烈建议在中央位置使用可共享的网络存储。

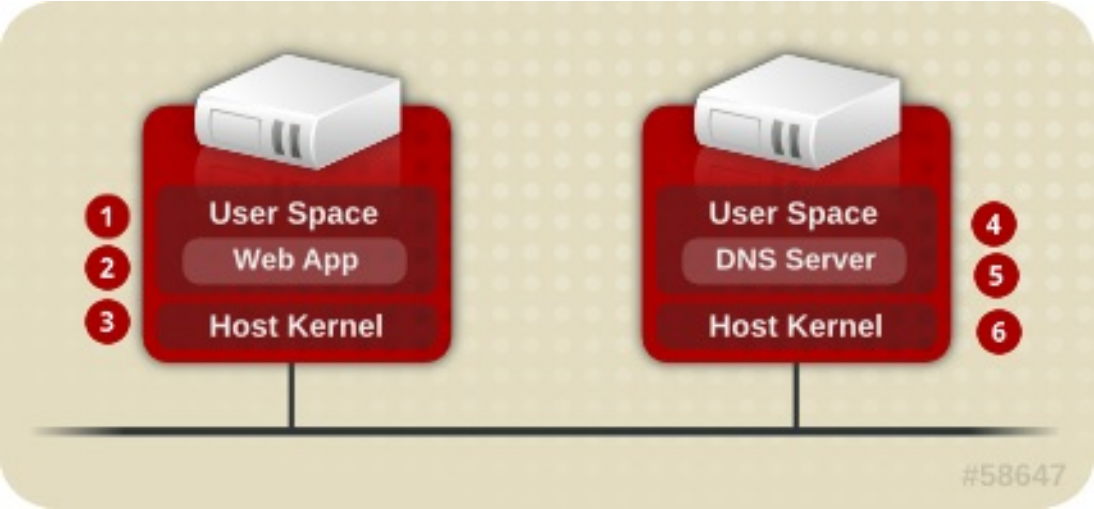
第 2 章 SVIRT

sVirt 是包含在 Red Hat Enterprise Linux 6 中集成 SELinux 和虚拟化的技术。sVirt 应用 Mandatory Access Control(MAC)来改进使用客户机虚拟机时的安全性。此集成技术提高了安全性并强化了系统，免受虚拟机监控程序中的漏洞。这对防止主机物理机器或其他客户机虚拟机进行攻击特别有用。

本章论述了 sVirt 如何与 Red Hat Enterprise Linux 6 中的虚拟化技术集成。

非虚拟化环境

在非虚拟化环境中，主机物理机器彼此分开，每个主机物理物理机器有一个自包含的环境，由 web 服务器或 DNS 服务器等服务组成。这些服务直接与自己的用户空间通信，托管物理机器的内核和物理硬件，从而直接向网络提供服务。下方的镜像代表非虚拟化环境：



??????
: 用户空间 - 所有用户模式应用程序和一些驱动程序执行的内存区域。

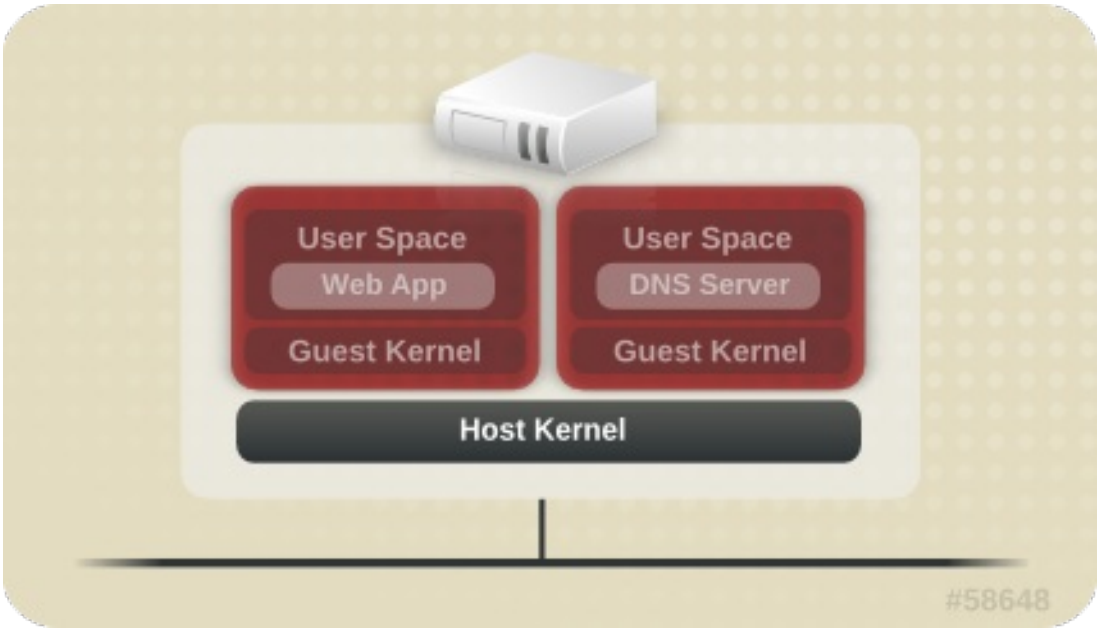
???
: Web App（网页应用服务器）- 提供可通过浏览器访问的 Web 内容。

??????
: Host Kernel - 严格保留用于运行主机物理机器的特权内核、内核扩展和大多数设备驱动程序。

???
: DNS 服务器 - 存储 DNS 记录，允许用户使用逻辑名称而不是 IP 地址访问网页。

虚拟化环境

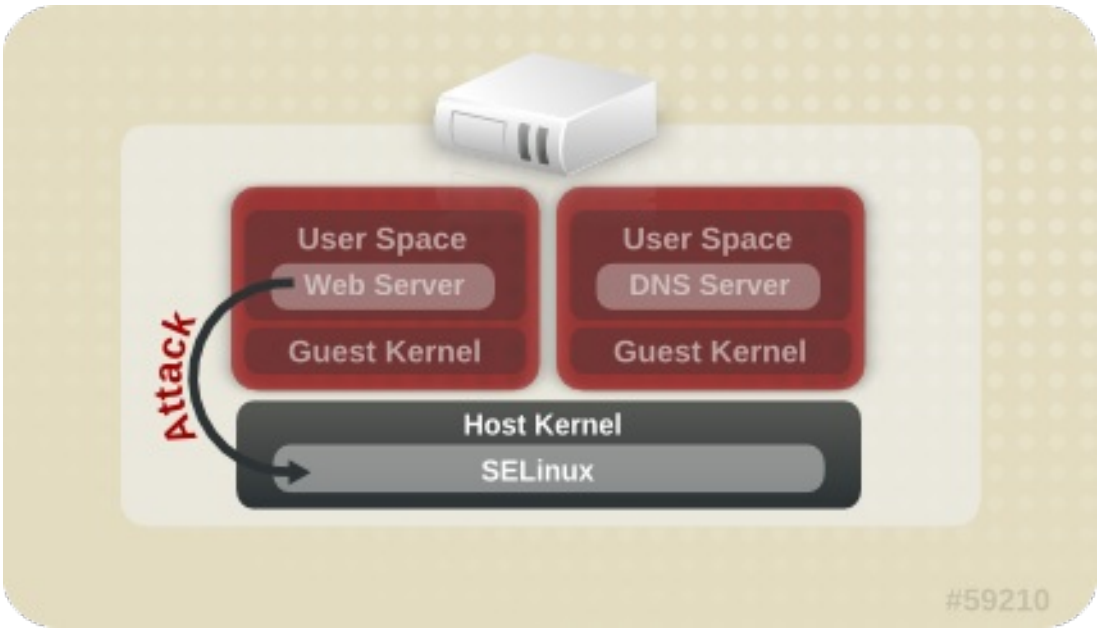
在虚拟环境中，多个虚拟操作系统可以在驻留于主机物理机器的单个内核中运行。下图代表虚拟化环境：



2.1. 安全和虚拟化

如果没有虚拟化服务，则机器会被物理分隔。任何漏洞通常都会包含在受影响的机器中，但存在网络攻击的明显例外。当服务分组到虚拟化环境中时，系统中出现了额外的漏洞。如果虚拟机监控程序中存在可由客户机虚拟机利用的安全漏洞，这种客户机虚拟机可能不仅可以攻击主机物理计算机，而且该虚拟机也可能不再受到该运行虚拟机的其他客户机虚拟机。这些攻击可以扩展到客户机虚拟机之外，也可公开其他 guest 虚拟机以攻击。

sVirt 是隔离客户机虚拟机并限制其在被利用时启动进一步攻击的能力。在以下镜像中演示这个情况，其中攻击无法断出 guest 虚拟机和其他客户机虚拟机：



SELinux 在实现强制访问控制(MAC)的过程中引入了虚拟化实例的可插拔安全框架。sVirt 框架允许 guest 虚拟机及其资源进行唯一标记。标记后，规则可以应用，以拒绝不同客户机虚拟机之间的访问。

2.2. SVIRT 标记

与 SELinux 保护的其他服务一样，sVirt 使用基于进程的机制和限制，为客户机虚拟机提供额外的安全层。在典型的用途下，您甚至不应注意到 sVirt 正在在后台工作。这部分论述了 sVirt 的标签功能。

如以下输出中所示，在使用 sVirt 时，每个虚拟化客户机虚拟机进程都会标签并使用动态生成的级别运行。每个进程都与其他具有不同级别的虚拟机隔离：

```
# ps -eZ | grep qemu

system_u:system_r:svirt_t:s0:c87,c520 27950 ？ 00:00:17 qemu-kvm
```

实际磁盘镜像会自动标记以匹配进程，如以下输出中所示：

```
# ls -lZ /var/lib/libvirt/images/*

  system_u:object_r:svirt_image_t:s0:c87,c520   image1
```

下表概述了在使用 sVirt 时可以分配的不同上下文标签：

表 2.1. sVirt 上下文标签

SELinux 上下文	键入 / Description
system_u:system_r:svirt_t:MCS1	客户机虚拟机进程.MCS1 是一个随机 MCS 字段。支持大约 500,000 个标签。
system_u:object_r:svirt_image_t:MCS1	客户机虚拟机镜像。只有具有相同 MCS 字段的*svirt_t* 进程才能读写这些镜像。
system_u:object_r:svirt_image_t:s0	共享读/写内容的 guest 虚拟机.所有 *svirt_t* 进程都可以写入 svirt_image_t:s0 文件。

在使用 sVirt 时，也可以执行静态标记。静态标签允许管理员选择特定的标签，包括 guest 虚拟机的 MCS/MLS 字段。运行静态标记的虚拟化客户机虚拟机的管理员负责在镜像文件中设置正确的标签。客户机虚拟机将始终使用该标签启动，sVirt 系统永远不会修改静态标记的虚拟机内容的标签。这允许 sVirt 组件在 MLS 环境中运行。您还可以根据您的要求，在系统上运行具有不同敏感度级别的多个客户机虚拟机。

第 3 章 克隆虚拟机

创建客户机副本时，有两种客户端虚拟机实例：

- *克隆* 是单个虚拟机的实例。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幅修改。这对于查看这些更改对系统的影响非常有用。

克隆和模板都是虚拟机实例。它们之间的区别在于如何使用它们。

要使创建的克隆正常工作，在克隆前通常必须删除要克隆的虚拟机的信息和配置。根据使用克隆方式，需要删除的信息有所不同。

要删除的信息和配置可能位于以下任意级别：

- *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)的数量及其 MAC 地址。
- *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。例如，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任何内容。示例包括激活代码和注册信息。

注意

本章不包括关于删除应用程序级别的信息，因为信息和方法特定于每个应用程序。

因此，必须将一些信息和配置从虚拟机中删除，而其他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。

3.1. 为关闭准备虚拟机

在克隆虚拟机前，必须先在其磁盘镜像上运行 virt-sysprep 实用程序，或使用以下步骤来准备它：

过程 3.1. 准备虚拟机以进行克隆

1. **设置虚拟机**
   a. 构建要用于克隆或模板的虚拟机。
      - 在克隆上安装任何所需的软件。
      - 为操作系统配置任何非唯一的设置。
      - 配置任何非唯一的应用设置。
2. **删除网络配置**
   a. 使用以下命令删除任何持久性的 udev 规则：

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

注意

如果没有删除 udev 规则，则第一个 NIC 的名称可以是 eth1 而不是 eth0。

b. 通过对 **/etc/sysconfig/network-scripts/ifcfg-eth[x]** 进行以下编辑，从 ifcfg 脚本中删除唯一的网络详情：

i. 删除 HWADDR 和 Static 行

注意

如果 HWADDR 与新 guest 的 MAC 地址不匹配，则将忽略 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. 确保该文件包括以下行：

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

c. 如果存在以下文件，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. 使用以下之一删除注册详情：

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

```
# 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，请运行以下命令：

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

  - 如果使用原始虚拟机，则只运行以下命令：

```
# subscription-manager clean
```

注意

原始 RHSM 配置集保留在门户网站中。

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时出现问题。

   b. 删除任何其它应用程序特定标识符或配置，如果在多个机器上运行时可能会导致冲突。

5. **配置虚拟机，使其在下次引导时运行配置向导**

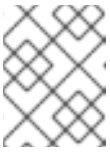
   a. 配置虚拟机，使其在下次引导时通过执行以下操作之一来运行相关配置向导：

   - 对于 Red Hat Enterprise Linux 6 及以下，使用以下命令在名为 .unconfigured 的 root 文件系统中创建一个空文件：

```
# 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从虚拟机中删除的配置。另外，在克隆第一次引导时，建议您更改主机名。

3.2. 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。

3.2.1. 使用 virt-clone 克隆客户机

您可以使用 **virt-clone** 来从命令行克隆虚拟机。

请注意，**virt-clone** 需要 root 权限才能成功完成。

virt-clone 命令提供多个可在命令行中传递的选项。它们包括常规选项、存储配置选项、网络配置选项和其它选项。仅需要 **--original**。要查看完整的选项列表，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 guest 虚拟机，并自动生成新名称和磁盘克隆路径。

例 3.1. 使用 virt-clone 克隆客户机

```
# 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 客户机虚拟机。

例 3.2. 使用 virt-clone 克隆客户机

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

3.2.2. 使用 virt-manager 克隆 Guest

这个步骤描述了使用 **virt-manager** 程序克隆客户机虚拟机。

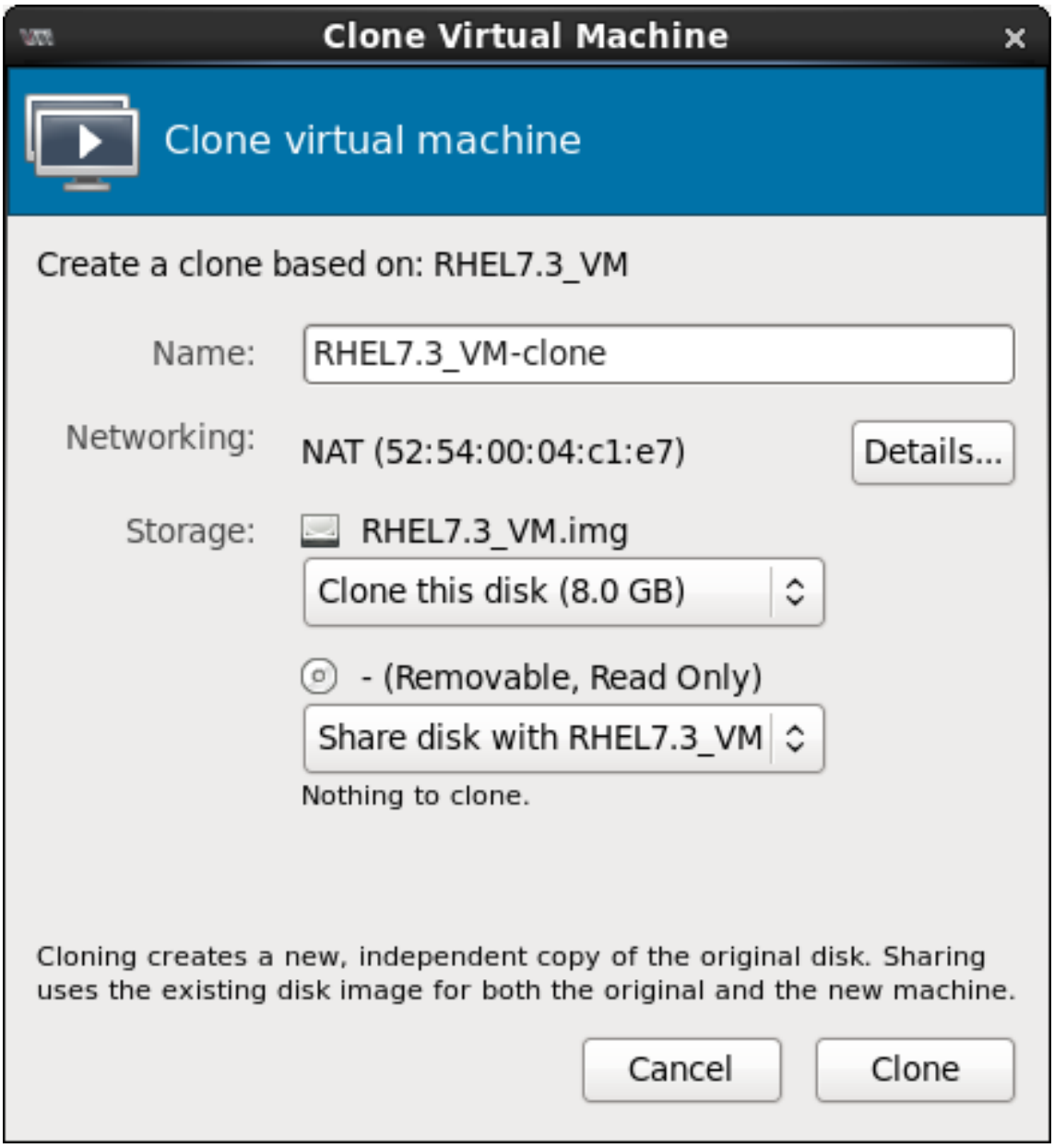
过程 3.2. 使用 virt-manager 克隆虚拟机

1. **打开 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 用户身份运行 virt-manager 命令。**

   从 **虚拟机管理器** 中的 guest 虚拟机列表中选择要克隆的 guest 虚拟机。

   右键单击要克隆的 guest 虚拟机，然后选择 **Clone**。此时会打开 Clone Virtual Machine 窗口。

图 3.1. 克隆虚拟机窗口



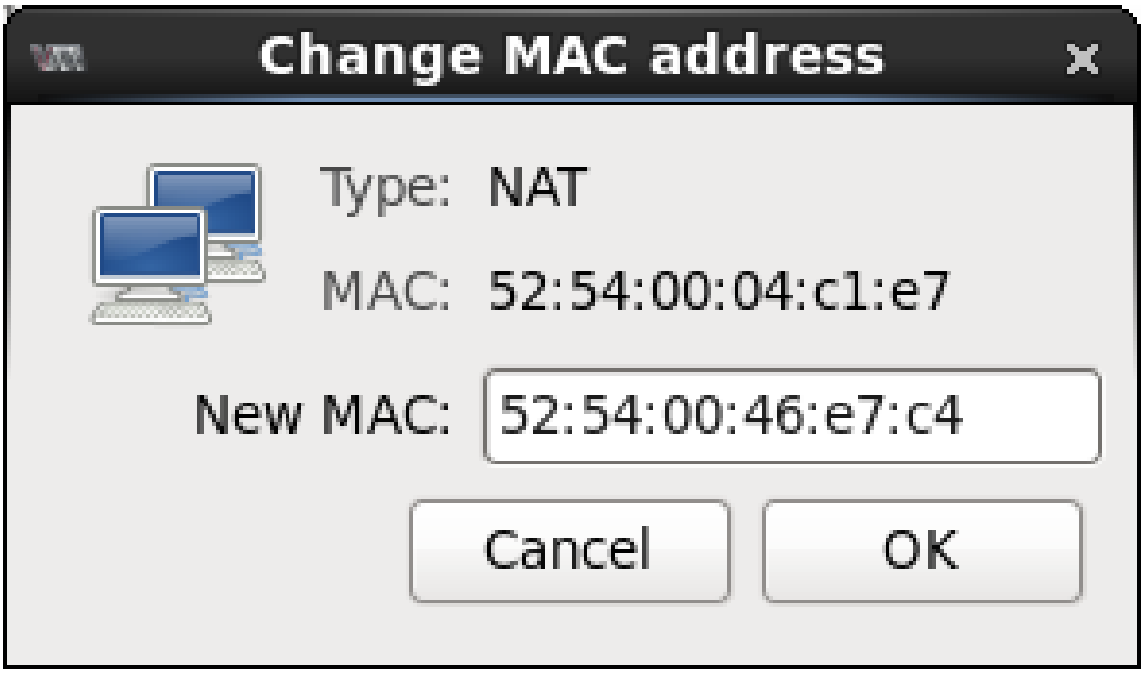
2. **配置克隆**

   - 要更改克隆的名称，请为克隆输入一个新名称。
   - 要更改网络配置，请单击 **Details**。

     为克隆输入新的 MAC 地址。

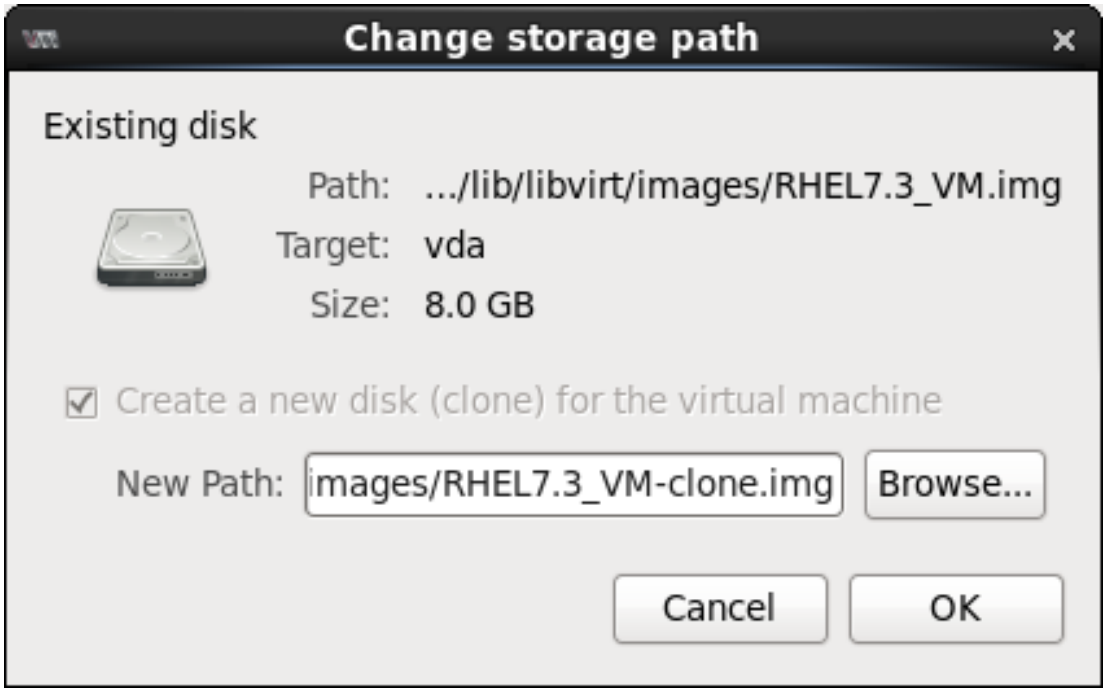
     点击 **确定**。

图 3.2. 更改 MAC Address 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** - 将为克隆的客户机虚拟机克隆磁盘
  - **与 *客户机虚拟机名称*共享磁盘 - 磁盘将由要克隆的 guest 虚拟机共享** 并克隆
  - **Details** - 打开 Change storage path 窗口，为磁盘选择新路径

    图 3.3. 更改 存储路径 窗口



3. **克隆客户端虚拟机**
   单击 **Clone**。

第 4 章 KVM 实时迁移

本章论述了将一台主机物理机器中运行的客户机虚拟机迁移到另一个主机。在两个实例中，主机物理计算机运行 KVM 管理程序。

迁移描述了将客户机虚拟机从一个主机物理机器移到另一个主机的过程。这是因为客户机虚拟机正在虚拟化环境中运行，而不是直接在硬件上运行。迁移适用于：

- 负载平衡 - 虚拟机可以在主机物理机器过载时迁移到主机物理机器，或者另一台主机物理机器处于使用不足时。
- 硬件独立性 - 当我们需要升级、添加或删除主机物理机器上的硬件设备时，我们可以安全地将 guest 虚拟机重新定位到其他主机物理机器。这意味着，guest 虚拟机不会因硬件改进而停机。
- 节能 - 可以将虚拟机重新分发到其他主机物理机器，因此可以关闭来节省能源并在低用量时降低成本。
- 地理迁移 - 虚拟机可以移动到低延迟或严重情况下的另一位置。

迁移工作：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。建议您使用共享、联网存储来存储要迁移的客户机虚拟机镜像。另外，还建议在迁移虚拟机时将 libvirt 管理的存储池用于共享存储。

迁移可以进行实时迁移，或者不执行。

在实时迁移中，客户机虚拟机可在源主机物理机器上继续运行，同时其内存页面会转移到目标主机物理机器。在迁移过程中，KVM 会监控其已传输的页面中任何更改的源，并在所有初始页面转移时开始传输这些更改。KVM 还估计迁移过程中传输速度，因此当要传输的剩余数据量将花费特定的配置期限（默认为 10 毫秒），KVM 会暂停原始 guest 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。

未执行实时迁移，暂停客户机虚拟机，然后将客户机虚拟机内存的镜像移动到目标主机物理机器。然后，客户端虚拟机会在目标主机物理机器上恢复，并释放源主机物理机器中使用的 guest 虚拟机。完成此类迁移所需的时间取决于网络带宽和延迟。如果网络使用过重或低带宽，迁移将花费更长的时间。

如果原始 guest 虚拟机修改的页面比 KVM 可以将其传送到目标主机物理机器的速度快，则必须使用离线迁移，因为实时迁移永远不会完成。

4.1. 实时迁移要求

迁移客户机虚拟机需要以下内容：

迁移要求

- 使用以下协议之一在共享存储上安装的客户机虚拟机：
  - 基于光纤通道的 LUN
  - iSCSI
  - FcoE
  - NFS
  - GFS2

  - SCSI RDMA 协议(SCSI RCP)：Infiniband 和 10GbE iWARP 适配器中使用的块导出协议

- 迁移平台和版本应根据表 表 4.1 “实时迁移兼容性” 检查。另请注意，Red Hat Enterprise Linux 6 支持在共享存储上使用 raw 和 qcow2 镜像的客户机虚拟机实时迁移。

- 两个系统都必须打开正确的 TCP/IP 端口。如果使用防火墙，请参考《 *Red Hat Enterprise Linux 虚拟化安全指南*》， https://access.redhat.com/site/documentation/ 该指南可通过以下网址获取详细的端口信息。

- 导出共享存储介质的独立系统。存储不应位于用于迁移的两个主机物理计算机上。

- 共享存储必须挂载到源和目标系统上的同一位置。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，但不建议这样做。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。但是，如果您想要使用 virsh 执行迁移，则可将不同的网络配置和挂载目录用于帮助 **--xml** 选项或进行迁移时预告。即使没有共享存储，迁移仍可以通过 **--copy-storage-all** 选项（已弃用）成功。有关 **prehook** 的更多信息，请参阅 libvirt.org，以及有关 XML 选项的更多信息，请参阅 第 20 章 *操作域 XML*。

- 当在公共 bridge+tap 网络中的现有客户端虚拟机上尝试迁移时，源和目标主机物理机器必须位于同一网络中。否则，客户机虚拟机网络在迁移后不会运行。

- 在 Red Hat Enterprise Linux 5 和 6 中，KVM 客户机虚拟机的默认缓存模式被设置为 ***none***，这可以防止磁盘状态不一致。将缓存选项设置为 ***none*** （例如使用 **virsh attach-disk 缓存 none** ），导致所有 guest 虚拟机的文件都使用 ***O_DIRECT*** 标记（在调用 **open** syscall）时打开，从而绕过主机物理机器的缓存，并只在客户端虚拟机上提供缓存。将缓存模式设置为 ***none*** 可防止潜在的不一致问题，并在使用时让虚拟机进行实时迁移。有关将缓存设置为 ***none*** 的详情，请参考 第 13.3 节 “在客户机中添加存储设备”。

确保已启用 **libvirtd** 服务(**# chkconfig libvirtd on**)并运行(**# service libvirtd start**)。还务必要注意，高效迁移的能力取决于 **/etc/libvirt/libvirtd.conf** 配置文件中的参数设置。

过程 4.1. Configuring libvirtd.conf

1. 打开 **libvirtd.conf** 需要以 root 用户身份运行命令：

```
# 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：

```
# service libvirtd restart
```

4.2. 实时迁移和 RED HAT ENTERPRISE LINUX 版本兼容性

实时迁移支持如表 表 4.1 “实时迁移兼容性” 所示：

表 4.1. 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	5.x → 6.y	不支持	

迁移方法	发行类型	示例	实时迁移支持	备注
向前	次发行版本	5.x → 5.y (y>x, x>=4)	完全支持	应报告任何问题
向前	次发行版本	6.x → 6.y (y>x, x>=0)	完全支持	应报告任何问题
向后	主发行版本	6.x → 5.y	不支持	
向后	次发行版本	5.x → 5.y (x>y,y>=4)	支持	有关已知问题，请参阅 迁移的故障排除
向后	次发行版本	6.x → 6.y (x>y, y>=0)	支持	有关已知问题，请参阅 迁移的故障排除

迁移的故障排除

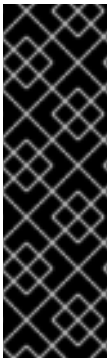
- **SPICE 的问题** - 我们发现，从 Red Hat Enterprise Linux 6.0 → 6.1 进行迁移时 SPICE 有不兼容的更改。在这种情况下，客户端可能会断开并重新连接，从而导致临时丢失音频和视频。这只是临时性，所有服务将恢复。
- **USB 的问题** - Red Hat Enterprise Linux 6.2 添加了 USB 功能，其中包括迁移支持，但不需要注意重置 USB 设备并导致在设备上运行的任何应用程序中止。这个问题已在 Red Hat Enterprise Linux 6.4 中解决，未来版本不应该发生这个问题。为防止这个问题在 6.4 之前的版本中发生，在使用 USB 设备时 abstain 被迁移。
- **迁移协议的问题** - 如果向后兼容未显示"未知部分错误"，请重复迁移过程可以解决这个问题，因为它可能是临时的错误。否则，请报告问题。

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 4.3 节 “共享存储示例：简单迁移的 NFS”

4.3. 共享存储示例：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。虽然大型安装并不实际，但会显示仅显示迁移技巧。不要使用这个示例来迁移或运行多个虚拟机。另外，还需要启用 ***sync*** 参数。这是正确导出 NFS 存储所必需的。另外，强烈建议您将 NFS 挂载到源主机物理机器上，并且需要在源主机物理机器的 NFS 挂载目录上创建 guest 虚拟机的镜像。另请注意，NFS 文件锁定 **不得** 被使用，因为 KVM 不支持它。

iSCSI 存储是大型部署的更好选择。有关配置详情请参考 第 12.5 节 “基于 iSCSI 的存储池”。

另请注意，本节中提供的说明并不会代替 *Red Hat Linux Storage Administration Guide* 中详述的说明。有关配置 NFS、打开 IP 表和配置防火墙的详情，请参考本指南。

1. **为磁盘镜像创建目录**
   此共享目录将包含 guest 虚拟机的磁盘映像。为此，可在与 **/var/lib/libvirt/images** 不同的位置创建一个目录。例如：

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

2. **为 NFS 配置文件添加新目录路径**
   NFS 配置文件是位于 **/etc/exports** 中的文本文件。打开该文件，并编辑在第 1 步中创建的新文件的路径。

   ```
   # echo "/var/lib/libvirt-img/images" >> /etc/exports/[NFS-Config-FILENAME.txt]
   ```

3. **启动 NFS**

   a. 确保打开了 **iptables** 中的 NFS 端口（例如，2049），并将 NFS 添加到 **/etc/hosts.allow** 文件中。

   b. 启动 NFS 服务：

   ```
   # service nfs start
   ```

4. **将共享存储挂载到源和目标上**
   在源和目标系统上挂载 **/var/lib/libvirt/images** 目录，运行以下命令两次。在源系统上，再次在目标系统上执行。

   ```
   # mount source_host:/var/lib/libvirt-img/images /var/lib/libvirt/images
   ```

警告

请确定使用这个流程创建的目录符合 第 4.1 节 “实时迁移要求” 中所述的要求。另外，可能需要使用正确的 SELinux 标签标记该目录。有关详情请参考 Red Hat Enterprise Linux Storage Administration Guide 中的 NFS 章节。

4.4. 使用 VIRSH 进行实时 KVM 迁移

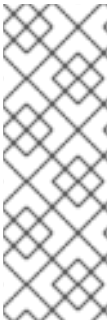
可以使用 **virsh** 命令将客户机虚拟机迁移到另一台主机物理机器。**migrate** 命令接受以下格式的参数：

```
# virsh migrate --live GuestName DestinationURL
```

请注意，如果不需要实时迁移，可以删除 **--live** 选项。其它选项在 第 4.4.2 节 “virsh migrate 命令的其它选项” 中列出。

GuestName 参数代表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。目标系统必须运行与 Red Hat Enterprise Linux 相同的版本，它使用相同的虚拟机监控程序，且在 **libvirt** 正在运行。

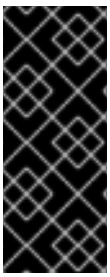


注意

用于正常迁移和对等迁移的 ***DestinationURL*** 参数有不同的语义：

- 常规迁移： ***DestinationURL*** 是目标主机物理机器的 URL，如源客户机虚拟机中看到。
- peer-to-peer migration： ***DestinationURL*** 是目标主机物理机器的 URL，如源主机物理机器所示。

输入完该命令后，会提示您输入目标系统的 root 密码。



重要

在源服务器上的 **/etc/hosts** 文件中，需要有目标主机物理机器的条目才能成功迁移。在此文件中输入目标主机物理机器的 IP 地址和主机名，如下例所示，替换您的目标主机物理机器的 IP 地址和主机名：

```
10.0.0.20 host2.example.com
```

示例：使用 virsh 进行实时迁移

本例从 **host1.example.com** 迁移到 **host2.example.com**。更改您的环境的主机物理机器名称。本例迁移一个名为 **guest1-rhel6-64** 的虚拟机。

这个示例假设您已完全配置共享存储并满足所有先决条件（在此列出： 迁移要求）。

1. **验证客户机虚拟机正在运行**
   从源系统 **host1.example.com**，验证 **guest1-rhel6-64** 是否正在运行：

   ```
   [root@host1 ~]# virsh list
   Id Name                 State
   ----------------------------------
    10 guest1-rhel6-64     running
   ```

2. **迁移客户机虚拟机**
   执行以下命令，将 guest 虚拟机实时迁移到目的地 **host2.example.com**。在目标 URL 的末尾附加 **/system**，以告知 libvirt 您需要完全访问。

   ```
   # virsh migrate --live guest1-rhel6-64 qemu+ssh://host2.example.com/system
   ```

   输入完该命令后，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的大小，迁移可能需要一些时间。**virsh** 仅报告错误.在完全迁移前，客户机虚拟机将继续在源主机物理机器中运行。

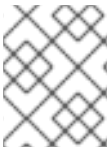
注意

在迁移过程中，完成百分比指示符数可能会在进程完成前减少多次。这是因为重新计算整个进度造成的，因为需要再次复制迁移后更改的源内存页面。因此，此行为是预期的，并不代表迁移中的任何问题。

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-rhel6-64** 是否正在运行：

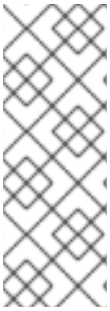
```
[root@host2 ~]# virsh list
Id Name                 State
----------------------------------
 10 guest1-rhel6-64     running
```

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 5 章 *客户机的远程管理*。



注意

无法使用 **virsh migrate** 命令迁移非运行 guest 虚拟机。要迁移非运行的客户机虚拟机，应使用以下脚本：

```
virsh dumpxml Guest1 > Guest1.xml
virsh -c qemu+ssh://<target-system-FQDN>  define Guest1.xml
virsh undefine Guest1
```

4.4.1. 使用 virsh 进行迁移的额外提示

每次迁移在单独的命令 shell 中运行时，可以执行多个并发实时迁移。但是，这要小心谨慎，并且应谨慎地进行计算，因为每个迁移实例各自使用一个 MAX_CLIENT（源和目标）。由于默认设置为 20，在不更改设置的情况下运行 10 个实例。如果您需要更改设置，请参阅 过程 4.1, “Configuring libvirtd.conf” 过程。

1. 打开 libvirtd.conf 文件，如 过程 4.1, “Configuring 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 = 20


# The minimum limit sets the number of workers to start up
# initially. If the number of active clients exceeds this,
```

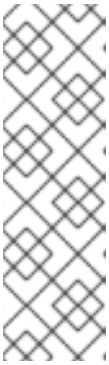
```
# then more threads are spawned, upto max_workers limit.
# Typically you'd want max_workers to equal maximum number
# of clients allowed
#min_workers = 5
#max_workers = 20


# 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是相同的。***max_clients*** 将每迁移（每个一侧）使用 2 个客户端（每个一侧），在目标执行阶段，并在完成阶段的 0 个 worker 中使用 1 个 worker。***max_workers***

重要

max_clients 和 ***max_workers*** 参数设置适用于所有向 libvirtd 服务的连接的客户机虚拟机。这意味着，使用同一客户机虚拟机的任何用户，并同时执行迁移将保留至 ***max_clients*** 和 ***max_workers*** 参数设置中设置的限值。这就是为什么在执行并发实时迁移前需要仔细考虑最大值。

4. 保存文件并重启该服务。



注意

有些情况下，迁移连接会丢弃，因为已启动但尚未进行身份验证的 ssh 会话太多。默认情况下，**sshd** 允许任何时候都允许 10 个会话处于"预先身份验证的状态"。此设置由 sshd 配置文件中的 **MaxStartups** 参数控制（在此位置：**/etc/ssh/sshd_config**），这可能需要进行一些调整。应谨慎调整这个参数，因为限制会被实施以防止 DoS 攻击（一般使用资源）。将此值设置为高时，将满足其用途。要更改此参数，请编辑文件 **/etc/ssh/sshd_config**，从 **MaxStartups** 行的开头删除 **#**，并将 **10**（默认值）改为一个更高的数字。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。

4.4.2. virsh migrate 命令的其它选项

除了 **--live** 之外，virsh migrate accept accept the options:

- **--direct** - 用于直接迁移
- **--p2p** - 用于对等的迁移
- **--tunnelled** - 用于隧道迁移
- **--persistent** - 将域保留为目标主机物理机器的持久性状态
- **--undefinesource** - 删除源主机物理机器上的客户机虚拟机
- **--suspend** - 在目标主机物理机器上使域处于暂停状态
- **--change-protection** - 强制实施在迁移时不对域进行不兼容的配置更改；在虚拟机监控程序支持时此选项会被隐式启用，但如果管理程序缺乏更改保护支持，则可以明确用于拒绝迁移。
- **--unsafe** - 强制迁移发生，忽略所有安全步骤。
- **--verbose** - 显示迁移的进度。
- **--abort-on-error** - 在迁移过程中发生软错误（如 I/O 错误）时取消迁移。
- **--migrateuri** - 通常被省略的迁移 URI。
- **--domain** *[string]*- 域名、id 或 uuid
- **--desturi** *[string]*- 目标主机物理机器的连接 URI（普通迁移）或 source(p2p migration)
- **--migrateuri** - 迁移 URI，通常可以省略
- **--timeout** *[seconds]*- 强制客户端虚拟机在实时迁移计数器超过 N 秒时暂停。它只能用于实时迁移。启动超时后，迁移便继续在暂停的客户机虚拟机上继续。
- **--dname** *[string]* - 在迁移过程中将客户机虚拟机的名称更改为新名称（如果支持）
- **--** XML - 表明的文件名可用于提供在目的地上使用的其他 XML 文件，为域 XML 的任何特定于主机的特定部分提供较大的更改，如访问底层存储时源和目的地之间的命名差异的核算。通常省略这个选项。

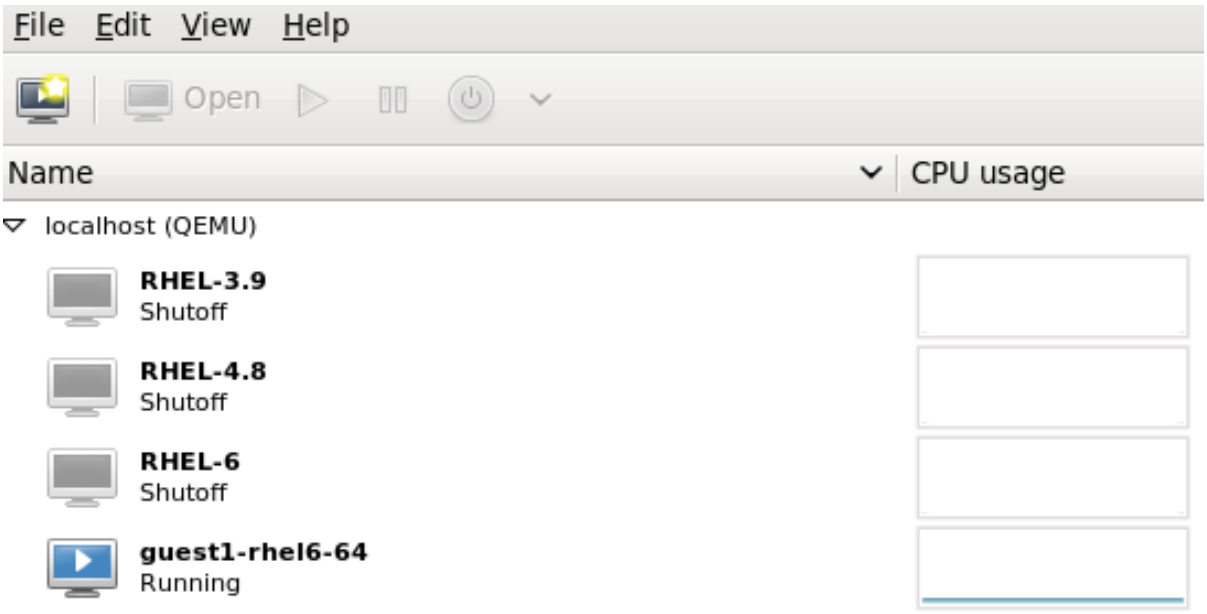
有关详细信息，请参阅 virsh man page。

4.5. 使用 VIRT-MANAGER 迁移

本节介绍使用 **virt-manager** 将 KVM 客户机虚拟机从一个主机物理机迁移到另一个主机。

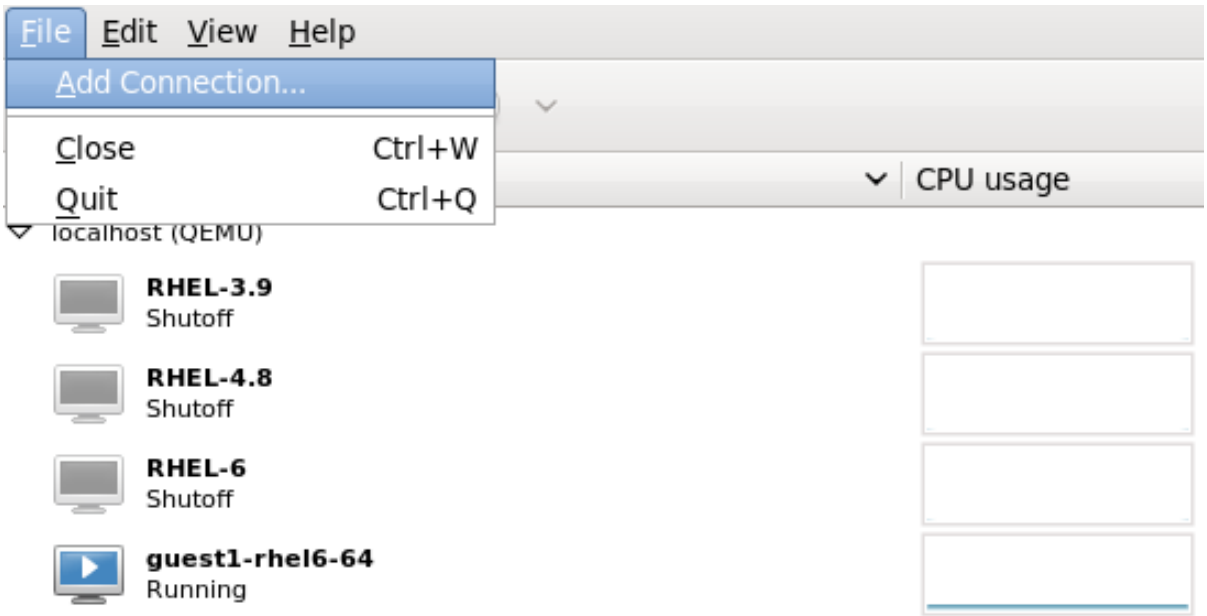
1. **打开 virt-manager**
   打开 **virt-manager**。从主菜单栏中选择" **应用程序 → ""系统工具"" → 虚拟机管理器** "，以启动 **virt-manager**。

图 4.1. virt-Manager 主菜单



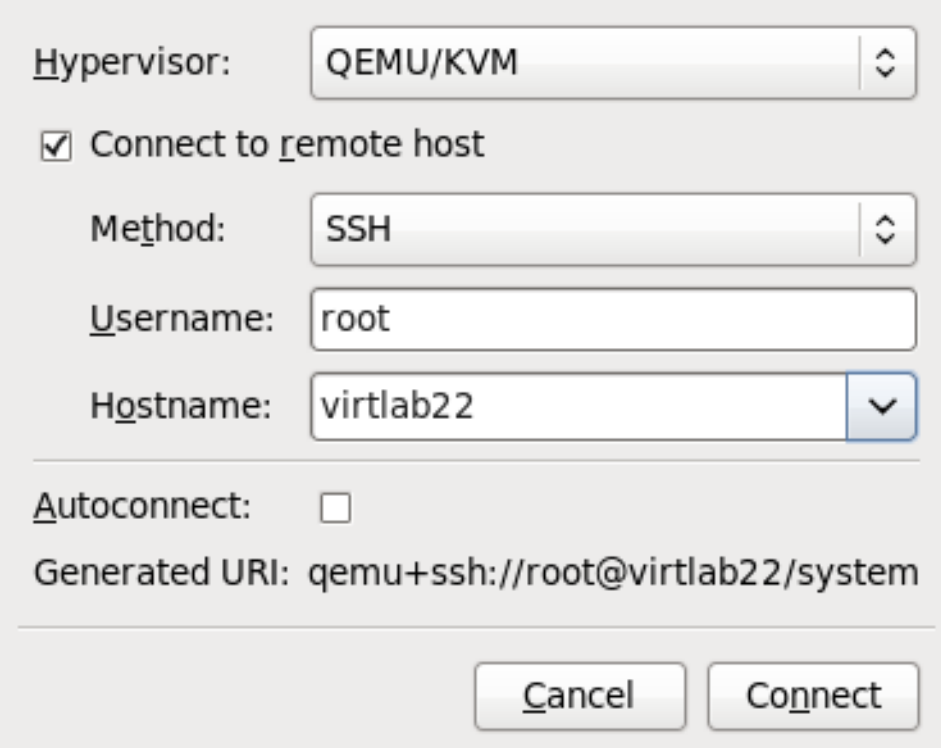
2. **连接到目标主机物理机器**
   单击 **File** 菜单，然后单击 **Add Connection**，以连接到目标主机物理机器。

图 4.2. 打开 Add Connection 窗口



3. **添加连接**
   此时会出现 **Add Connection** 窗口。

图 4.3. 添加与目标主机物理机器的连接

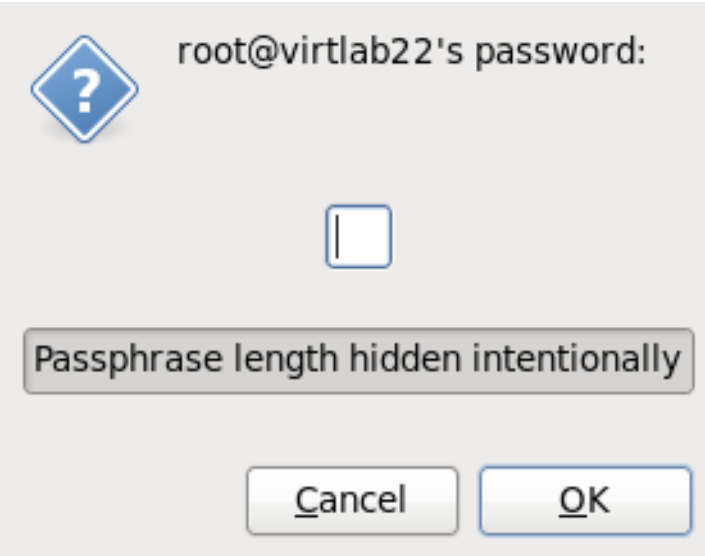


输入以下详情：

- **虚拟机监控程序** ：选择 QEMU/KVM。
- method ：选择连接方法。
- **用户名** ：输入远程主机物理机器的用户名。
- **hostname** ：输入远程主机物理机器的主机名。

点 **Connect** 按钮。本例中使用了 SSH 连接，因此下一步中必须输入指定用户的密码。

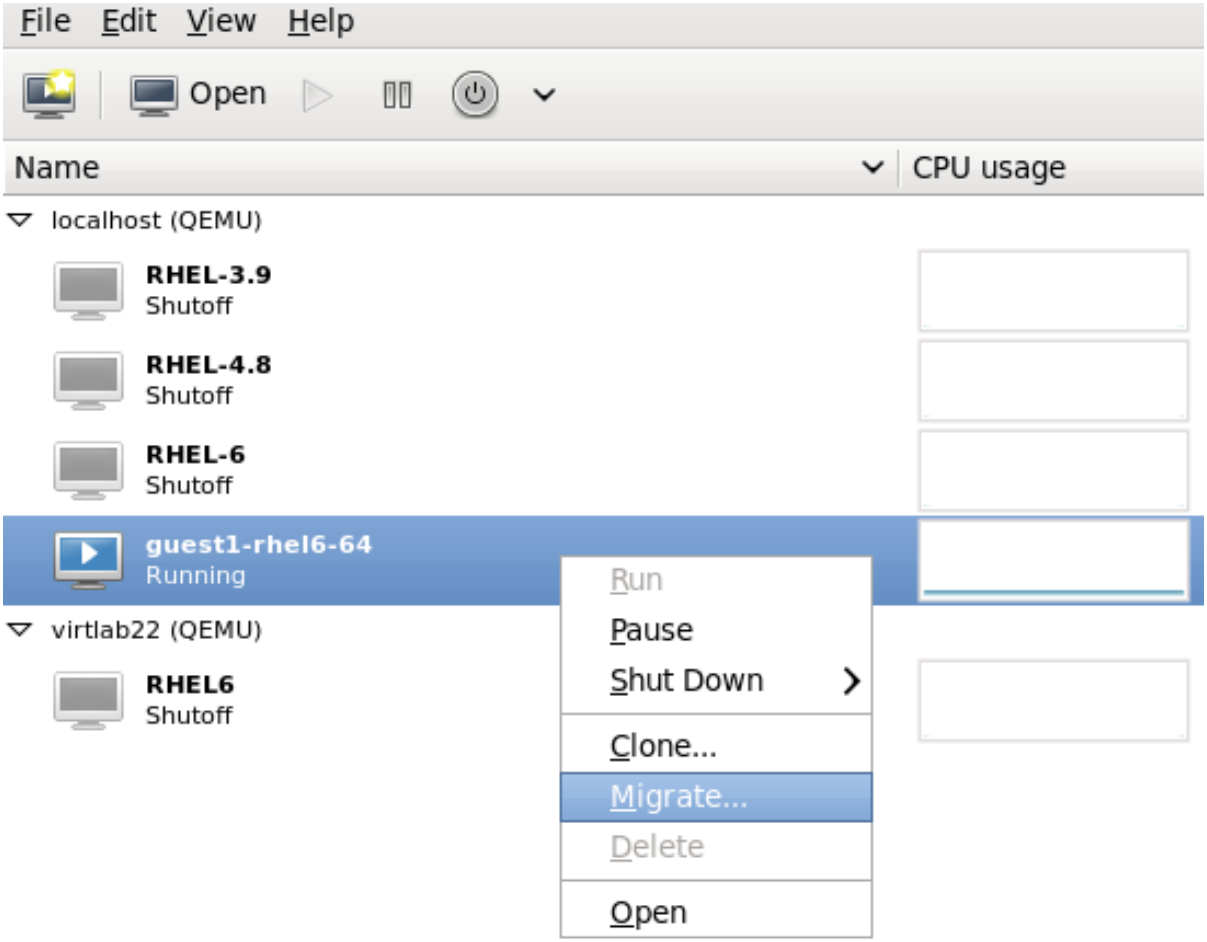
图 4.4. 输入密码



4. **迁移客户机虚拟机**

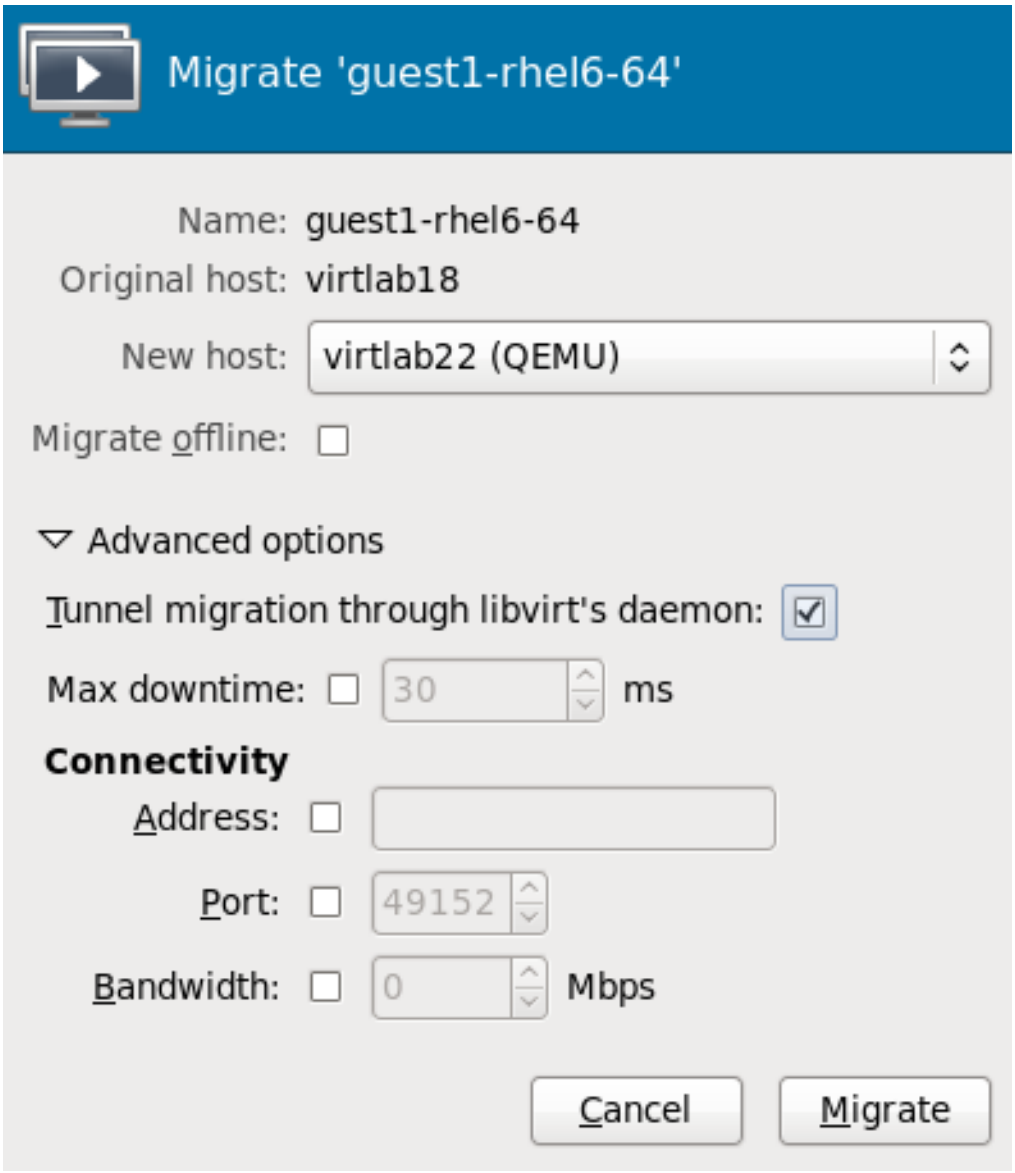
打开源主机物理机器中的客户机列表（单击主机名左侧的小三角），然后右键单击要迁移的 guest（本例中的**guest1-rhel6-** 64）并单击" **迁移** "。

图 4.5. 选择要迁移的客户端



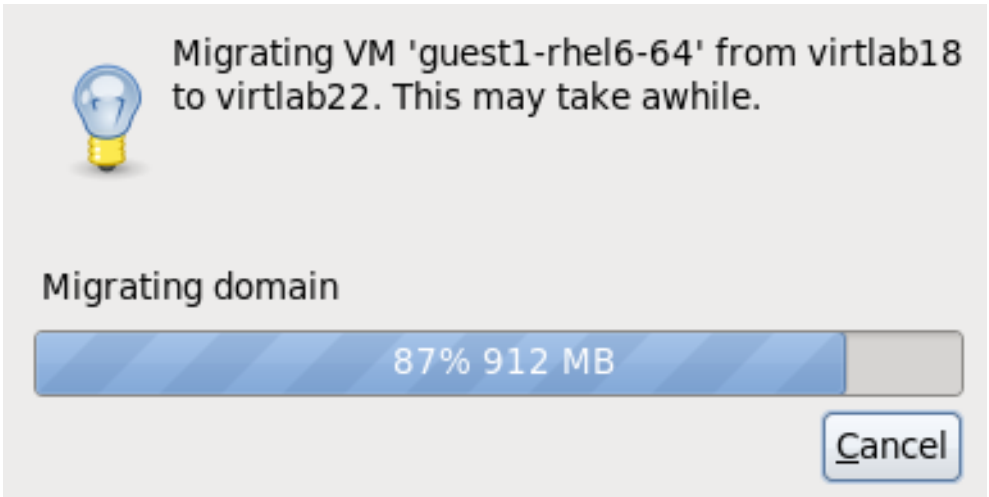
在 **New Host** 字段中，使用下拉列表选择要将 guest 虚拟机迁移到的主机物理机器，然后单击 **Migrate**。

图 4.6. 选择目标主机物理机器并启动迁移过程



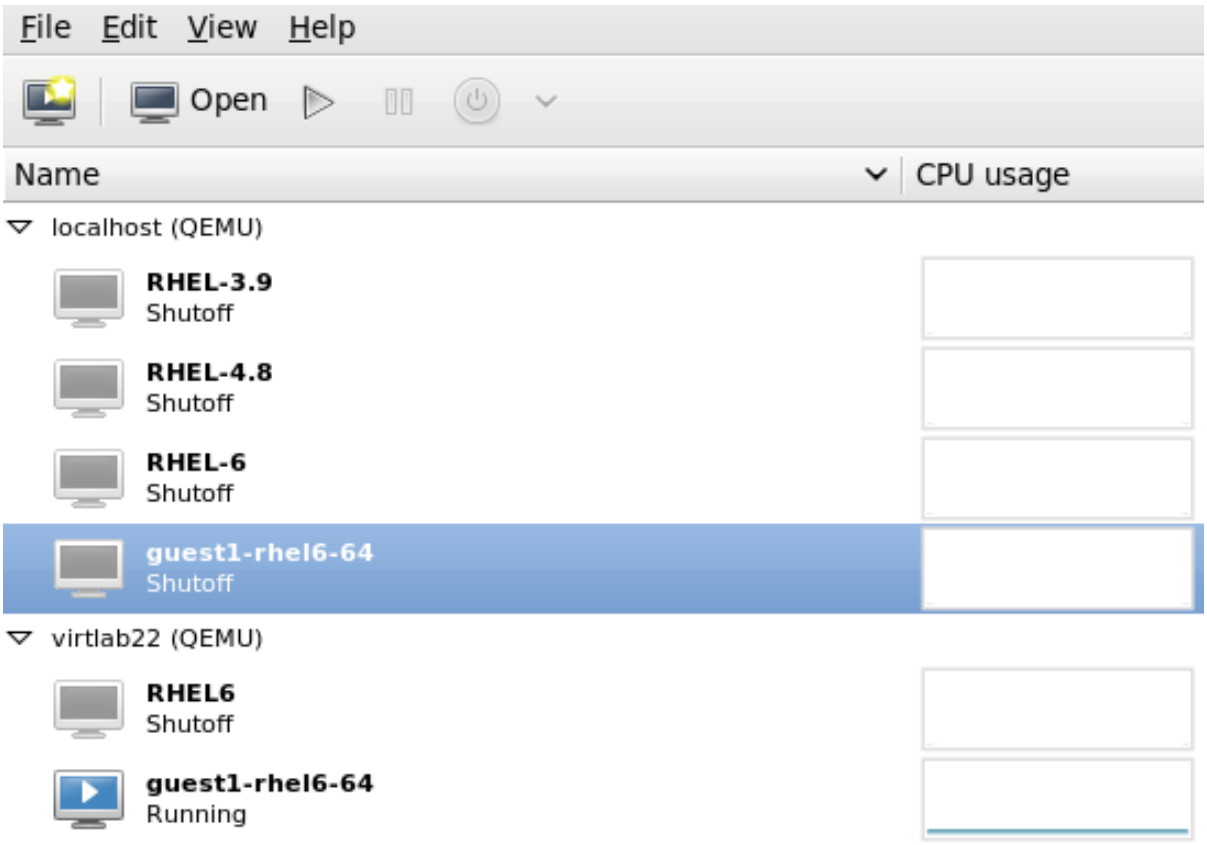
系统将显示进度窗口。

图 4.7. 进度窗口



virt-manager 现在显示目标主机上运行的新迁移的客户机虚拟机。在源主机物理机器中运行的客户端虚拟机现在被列为 Shutoff 状态。

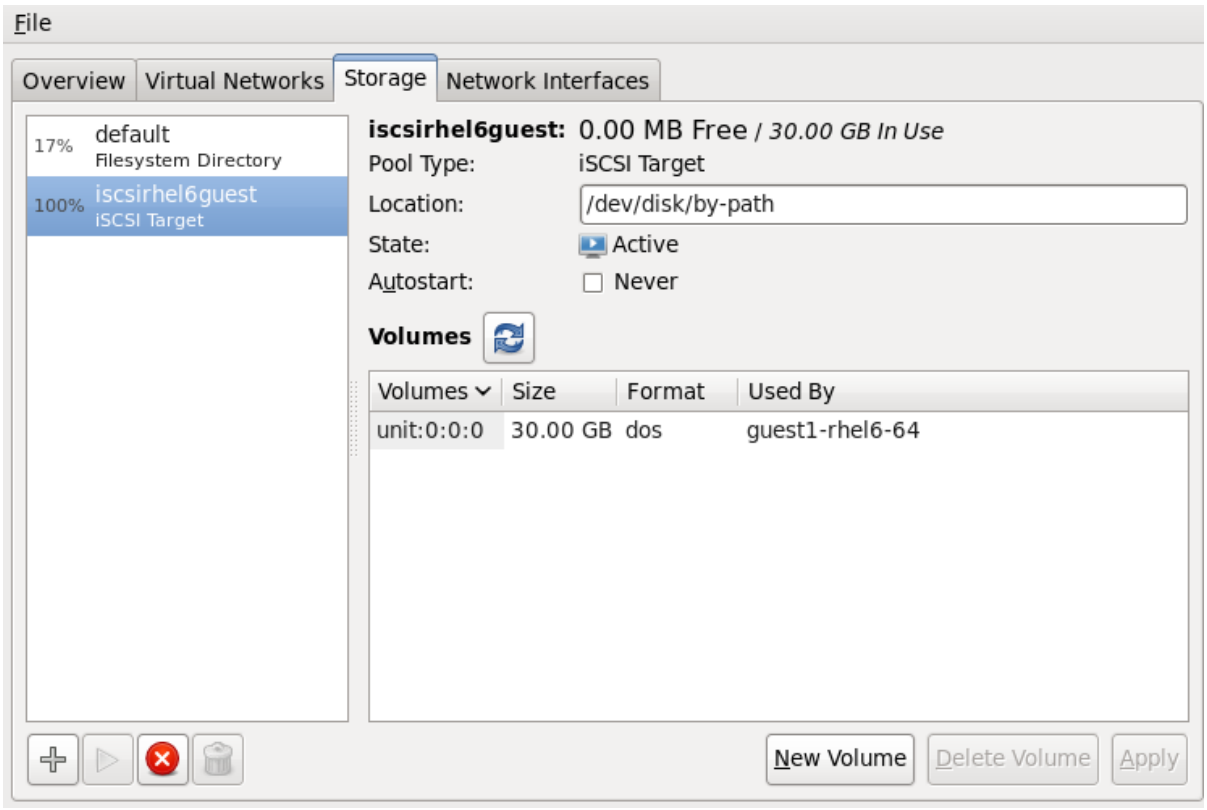
图 4.8. 迁移的客户机虚拟机在目标主机物理机器中运行



5. **可选 - 查看主机物理机器的存储详情**
在 **Edit** 菜单中，单击 **Connection Details**，显示 Connection Details 窗口。

点 **Storage** 选项卡。显示目标主机物理机器的 iSCSI 目标详情。请注意，迁移的客户机虚拟机被列为使用存储

图 4.9. 存储详情



此主机由以下 XML 配置定义：

图 4.10. 目标主机物理机器的 XML 配置

```
<pool type='iscsi'>
   <name>iscsirhel6guest</name>
   <source>
      <host name='virtlab22.example.com.'/>
      <device path='iqn.2001-05.com.iscsivendor:0-8a0906-fbab74a06-a700000017a4cc89-
rhevh'/>
   </source>
   <target>
      <path>/dev/disk/by-path</path>
   </target>
</pool>
  ...
```

第 5 章 客户机的远程管理

本节介绍如何使用 **ssh** 或 TLS 和 SSL 远程管理客户机。有关 SSH 的更多信息，请参见《 *Red Hat Enterprise Linux 部署指南*》。

5.1. 使用 SSH 进行远程管理

ssh 包提供了加密网络协议，可安全地向远程虚拟服务器发送管理功能。上述方法使用通过 **SSH** 连接安全地隧道化的 **libvirt** 管理连接来管理远程机器。所有验证都使用本地 **SSH** 代理收集的 **SSH** 公钥加密和密码或密码短语完成。此外，每个 guest 的 **VNC** 控制台也通过 **SSH** 进行隧道化。

请注意，使用 **SSH** 远程管理虚拟机时出现问题，包括：

- 您需要 root 登陆到远程机器来管理虚拟机，
- 初始连接设置过程可能很慢，
- 无法对所有主机或客户机撤销用户密钥的标准或简单方法，
- SSH 无法很好地扩展更大的远程机器。

注意

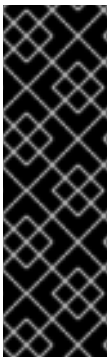
Red Hat Virtualization 允许远程管理大量虚拟机。详情请查看 Red Hat Virtualization 文档。

ssh 访问需要以下软件包：

- openssh
- openssh-askpass
- openssh-clients
- openssh-server

为 virt-manager配置无密码管理的 SSH 访问

以下说明假设您从头开始启动，并且尚未设置 **SSH** 密钥。如果您将 SSH 密钥设置并复制到其它系统中，您可以跳过这个过程。



重要

SSH 密钥依赖于用户，只能供其所有者使用。密钥的所有者是生成它的用户。键不能共享。

virt-manager 必须由拥有要连接到远程主机的密钥的用户运行。这意味着，如果远程系统由非 root 用户 **virt-manager** 管理，则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 所有并创建。

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。本示例使用本地 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**
使用机器 **virt-manager** 上的公钥对。本例使用 **~/.ssh/** 目录中的默认密钥位置。

```
# ssh-keygen -t rsa
```

3. **将密钥复制到远程主机**
不带密码或密码短语进行远程登录，需要向受管理的系统分发 SSH 密钥。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:
```

现在，尝试使用 **ssh root@host2.example.com** 命令登录到机器，并在 **.ssh/authorized_keys** 文件中检查以确保尚未添加意外密钥。

根据需要，对其他系统重复此操作。

4. **可选：在 ssh-agent 中添加密码短语**
下面的说明描述了如何在现有 ssh-agent 中添加密码短语。如果 ssh-agent 未在运行，它将无法运行。为了避免错误或冲突，请确保正确设置了 SSH 参数。如需更多信息，请参阅 *Red Hat Enterprise Linux 部署指南*。

如果需要，将 SSH 密钥的密语添加到 **ssh-agent**。在本地主机上，使用以下命令添加密码短语（如果有）来启用免密码登录。

```
# ssh-add ~/.ssh/id_rsa
```

SSH 密钥被添加到远程系统。

libvirt 守护进程(libvirtd)

libvirt 守护进程提供用于管理虚拟机的接口。您必须在需要管理的每个远程主机中安装并运行 **libvirtd** 守护进程。

```
$ ssh root@somehost
# chkconfig libvirtd on
# service libvirtd start
```

配置 **libvirtd** 和 **SSH** 后，您应能够远程访问和管理虚拟机。此时您应当能够使用 **VNC** 访问您的 guest。

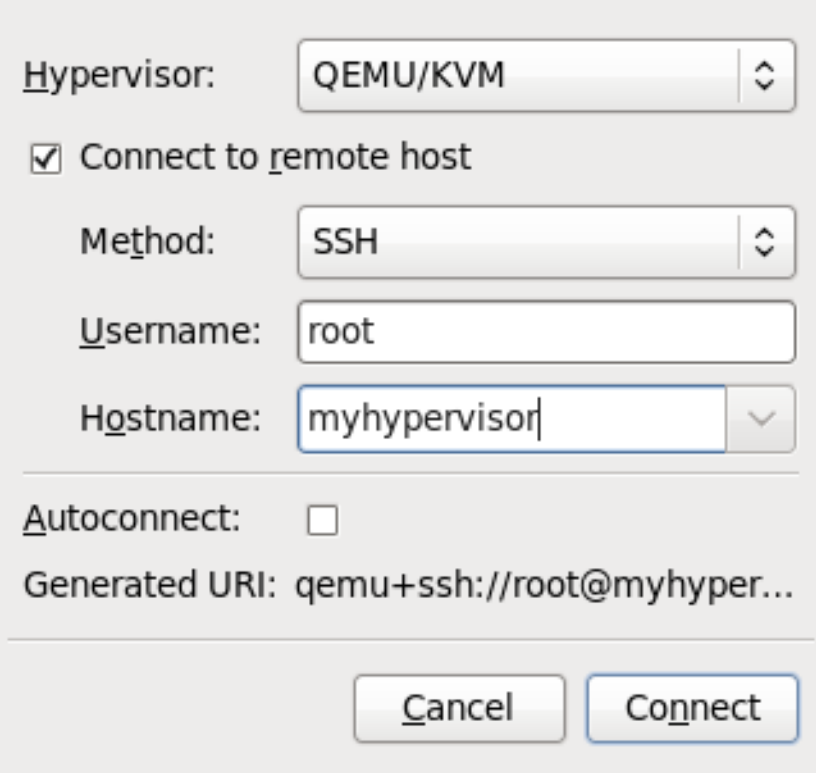
使用 virt-manager 访问远程主机

远程主机可以使用 virt-manager GUI 工具进行管理。SSH 密钥必须属于执行 virt-manager 的用户，以便免密码登录才能工作。

1. 启动 virt-manager。

2. 打开 **File->Add Connection** 菜单。

图 5.1. 添加连接菜单



3. 使用下拉菜单选择管理程序类型，然后单击"**连接到远程主机**"复选框以打开"连接方法"（在本例中为 Remote tunnel over SSH），然后输入所需的 **User name** 和 **Hostname**，然后单击 **Connect**。

5.2. 使用 TLS 和 SSL 进行远程管理

您可以使用 TLS 和 SSL 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 5.1 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技术。**libvirt** 管理连接打开传入连接的 TCP 端口，该端口根据 x509 证书安全地加密和验证。以下介绍了为 TLS 和 SSL 管理创建和部署身份验证证书的说明。

过程 5.1. 创建用于 TLS 管理的证书颁发机构(CA)密钥

1. 开始之前，请确认已安装 **certtool** 实用程序。如果没有：

```
# 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，下一步是创建一个签名文件，以便该密钥可以自我签名。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：

```
# vim ca.info
```

```
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。生成过程的结果名为 **cacert.pem**。此文件是公钥(certificate)。加载的文件 **cakey.pem** 是私钥。此文件不应保存在共享空间中。保留此密钥私钥。

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** 证书颁发机构证书文件，以使它们知道您的 CA 发布的证书可以被信任。要查看此文件的内容，请运行：

```
# certtool -i --infile cacert.pem
```

这是设置您的 CA 所需的全部内容。使 CA 的私钥保持安全，因为您需要为客户端和服务器发布证书。

过程 5.2. 发出服务器证书

此流程演示了如何为服务器的主机名发出 X.509 CommonName(CN)字段的证书。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 字段应该相同，ie 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 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. 使用以下命令创建证书：

```
# certtool --generate-certificate --load-privkey serverkey.pem --load-ca-certificate cacert.pem
--load-ca-privkey cakey.pem \ --template server.info --outfile servercert.pem
```

4. 这会导致生成的两个文件：

- ServerKey.pem - 服务器的私钥
- servercert.pem - 服务器的公钥

确保保留私钥 secret 的位置。要查看文件的内容，请执行以下命令：

```
# certtool -i --inifile servercert.pem
```

在打开此文件时，**CN=** 参数应该和之前设置的 CN 相同。例如， **mycommonname**。

5. 在以下位置安装这两个文件：

   - **ServerKey.pem** - 服务器的私钥。将此文件放在以下位置：
     /etc/pki/libvirt/private/serverkey.pem

   - **servercert.pem** - 服务器的证书。在服务器以下位置安装它：
     /etc/pki/libvirt/servercert.pem

过程 5.3. 发出客户端证书

1. 对于每个客户端（例如 virt-manager）链接到 libvirt 的任何程序，您需要向合适的名称(DN)发送带有 X.509 Distinguished Name(DN)的证书。这需要根据公司级别决定。

   例如，将使用以下信息：

   ```
   C=USA,ST=North Carolina,L=Raleigh,O=Red Hat,CN=name_of_client
   ```

   这个过程与 过程 5.2, “发出服务器证书” 类似，但请注意以下例外情况。

2. 使用以下命令生成私钥：

   ```
   # 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。该文件应当包含以下内容（应自定义字段来反映您的地区/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
   ```

5.3. 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)经验证并加密 TCP/IP 套接字，通常侦听公共端口号。要使用这个设置，您需要生成客户端和服务器证书。标准端口为 16514。

UNIX 套接字

UNIX 域套接字仅可在本地计算机上访问。套接字没有加密，并使用 UNIX 权限或 SELinux 进行身份验证。标准套接字名称为 **/var/run/libvirt/libvirt-sock** 和 **/var/run/libvirt/libvirt-sock-ro** （用于只读连接）。

SSH

通过安全 Shell 协议(SSH)连接传输。需要安装 Netcat（ nc 软件包）。libvirt 守护进程(**libvirtd**)必须在远程系统上运行。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 SSH 密钥管理（例如，**ssh-agent** 实用程序），否则将提示您输入密码。

ext

ext 参数用于任何可通过 libvirt 范围外连接到远程机器的外部程序。这个参数不被支持。

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，但管理员可以启用它进行测试或在可信网络上使用。默认端口为 16509。

默认传输（如果没有指定其他）为 TLS。

远程 URI

virsh 和 libvirt 使用统一资源标识符(URI)连接到远程主机。URI 也可与 **virsh** 命令的 **--connect** 参数一起使用，以在远程主机上执行单个命令或迁移。通过使用普通本地 URI 并添加主机名或传输名称，形成远程 URI。作为特殊情况，使用'remote' 的 URI 方案，将告知远程 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（用方括号为 "[]"，表示可选功能）：

```
driver[+transport]://[username@][hostname][:port]/path[?extraparameters]
```

请注意，如果虚拟机监控程序(driver)是 QEMU，则路径是必须的。如果是 XEN，则是可选的。

以下是有效远程 URI 的示例：

- qemu://hostname/
- xen://hostname/
- xen+ssh://hostname/

必须向外部位置提供传输方法或主机名。有关详情请参阅 http://libvirt.org/guide/html/Application_Development_Guide-Architecture-Remote_URIs.html。

远程管理参数示例

- 使用 SSH 传输和 SSH 用户名 **virtuser** 连接到一个名为 **host2** 的远程 KVM 主机。每个连接的命令是连接 [< **name>] [--readonly]**，其中 < **name&** gt; 是有效的 URI，如此处所述。有关 **virsh connect** 命令的详情请参考 第 14.1.5 节 “connect”

  ```
  qemu+ssh://virtuser@hot2/
  ```

- 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。本例中明确提供了到 UNIX 套接字的完整路径。

```
qemu+unix:///system?socket=/opt/libvirt/run/libvirt/libvirt-sock
```

- 使用未加密的 TCP/IP 连接到 libvirt 守护进程，使用 IP 地址 10.1.1.10 在端口 5000 上连接到服务器。这会将 test 驱动程序与默认设置搭配使用。

```
test+tcp://10.1.1.10:5000/default
```

附加 URI 参数

可以在远程 URI 中附加额外参数。下表 表 5.1 “附加 URI 参数” 涵盖了可识别的参数。其它参数将被忽略。请注意，参数值必须是 URI-escaped（即，在参数前附加一个问号(?)，并且特殊字符转换为 URI 格式）。

表 5.1. 附加 URI 参数

名称	传输模式	描述	用法示例
name	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、用户名和额外参数形成，但在某些情况下可以更好地提供名称。	name=qemu:///system
命令	SSH 和 ext	external 命令。对于 ext 传输，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，用于覆盖默认设置。对于 ssh 传输，这将传递给远程 netcat 命令（请参见 netcat）。	socket=/opt/libvirt/run/libvirt/libvirt-sock

名称	传输模式	描述	用法示例
netcat	ssh	**netcat** 命令可用于连接到远程系统。默认 netcat 参数使用 **nc** 命令。对于 SSH 传输，libvirt 使用以下表格构造 SSH 命令： ***command*** -p ***port*** [-l ***username***] ***hostname*** ***netcat*** -U socket ***port***、***username*** 和 ***hostname*** 参数可作为远程 URI 的一部分指定。***command***、***netcat*** 和 ***socket*** 来自其他的额外参数。	netcat=/opt/netcat/bin/nc
no_verify	tls	如果设置为非零值，则禁用对服务器证书的客户端检查。请注意，要禁用对客户端的证书或 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，则在无法自动登录到远程机器时，这将停止 ssh 询问密码（如果无法自动登录到远程机器）。当您无法访问终端时使用这个选项。	no_tty=1

第 6 章 使用 KVM 进行过量使用

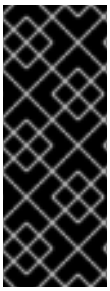
KVM 管理程序自动过量使用 CPU 和内存。这意味着可将更多虚拟化 CPU 和内存分配给虚拟机，超过系统中有物理资源。这是可行的，因为大多数进程都无法访问其分配资源的 100%。

因此，利用率低下的虚拟化服务器或桌面可在更少的主机上运行，这可节省大量系统资源，且在服务器硬件中影响较少、冷却和投资。

6.1. 过量使用内存

KVM 管理程序中运行的虚拟客户机没有为其分配物理 RAM 的专用块。相反，每个客户机虚拟机作为 Linux 进程，主机物理机器的 Linux 内核仅在请求时分配内存。此外，主机的内存管理器还可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 guest 虚拟机，并为主机物理机器的进程有足够的内存。作为基本规则，主机物理机器的操作系统需要最多 4GB 内存，最少有 4GB 的交换空间。有关为 swap 分区确定适当大小的高级说明，请参阅 红帽知识库基本。



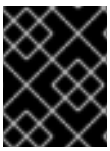
重要

对于常规内存问题，过量使用并不是一个理想的解决方案。建议使用内存短缺处理方法，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。

如果虚拟机被频繁交换，则虚拟机会较慢。另外，过量使用可能会导致系统内存不足(OOM)，这可能会导致 Linux 内核关闭重要系统进程。如果您决定过量使用内存，请确定执行足够的测试。若需提交，请联系红帽支持团队以获得帮助。

过量使用并不适用于所有客户机虚拟机，但已发现，可在桌面虚拟化设置中使用最小程度，或运行带有内核相同页面合并(KSM)的几台客户虚拟机。

有关 KSM 和过量使用的更多信息，请参阅 第 7 章 *KSM*。



重要

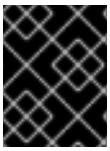
当使用 设备分配 时，所有虚拟机内存必须静态预先分配，才能启用具有分配设备的直接内存访问(DMA)。因此，设备分配不支持内存过量使用。

6.2. 过量使用虚拟 CPU

KVM 管理程序支持过量使用虚拟化 CPU。随着客户机虚拟机所允许的负载限制，可以过量使用虚拟化 CPU。当提交 VCPU 接近 100% 时，请小心操作可能会导致请求丢失或不可用。

当一台主机物理计算机拥有不共享相同的 vCPU 的客户机虚拟机时，虚拟化 CPU(vCPU)会被过度分配。KVM 应安全支持以 100% 负载的客户机虚拟机，其比率为五个 VCPU（5 台虚拟机）到一台主机物理计算机上的一个物理 CPU。KVM 将在所有机器间切换，确保负载处于平衡状态。

不要过量使用超过处理内核的物理数量的虚拟机。例如，具有四个 vCPU 的客户机虚拟机不应在具有双核处理器的主机中运行，而是在四核主机上。另外，建议不要为每个物理处理器内核分配超过 10 个 vCPU。



重要

在不进行大量测试的情况下，不要在生产环境中过量使用 CPU。使用 100% 处理资源的应用程序可能会在过量使用的环境中变得不稳定。在部署前进行测试。

有关如何从虚拟机获取最佳性能的更多信息，请参阅 Red Hat Enterprise Linux 6 虚拟化调整和优化指南。

第 7 章 KSM

共享内存的概念在现代操作系统中很常见。例如，当程序首次启动时，它会与父程序共享其所有内存。当子程序尝试修改此内存时，内核会分配一个新的内存区域，复制原始内容并允许程序修改这个新区域。这被称为 write（写时）的 copy。

KSM 是一种新的 Linux 功能，以相反地使用此概念。KSM 使内核能够检查两个或多个已运行程序并比较它们的内存。如果任何内存区域或页面相同，则 KSM 会将多个相同内存页减至单个页面。然后，这个页会在 write 上标记为副本。如果 guest 虚拟机修改了页面的内容，则会为该 guest 虚拟机创建一个新页面。

这对于使用 KVM 进行虚拟化非常有用。当客户机虚拟机启动时，它只会从父 **qemu-kvm** 进程中继承内存。客户机虚拟机运行后，可以在客户机运行相同操作系统或应用程序时，客户机虚拟机操作系统镜像的内容可以共享。

注意

页面重复数据删除技术（由 KSM 实施也使用）可能会引入侧信道，这些通道可能会用于泄漏多个客户机中的信息。如果出现这种情况，可以基于每个虚拟机禁用 KSM。

KSM 提供增强的内存速度和利用率。通过 KSM，通用进程数据存储在缓存或主内存中。这可减少 KVM 客户机的缓存丢失，从而可提高某些应用程序和操作系统的性能。其次，共享内存可减少客户机的总内存用量，从而可以增加密度并增加资源的利用率。

注意

从 Red Hat Enterprise Linux 6.5 开始，KSM 是 NUMA 感知功能。这样，它在合并页面时可以考虑 NUMA 本地性，从而避免与将页面移动到远程节点中的页面相关的性能下降。红帽建议在使用 KSM 时避免多节点内存合并。如果使用 KSM，将 **/sys/kernel/mm/ksm/merge_across_nodes** 可调项改为 ***0***，以避免在 NUMA 节点间合并页面。内核内存核算统计最终可在大量跨节点合并后相互冲突。因此，numad 在 KSM 守护进程合并大量内存后可能会变得混乱。如果您的系统有大量可用内存，可以通过关闭和禁用 KSM 守护进程来获得更高的性能。有关 NUMA 的更多信息，请参阅 *Red Hat Enterprise Linux 性能调优指南*。

Red Hat Enterprise Linux 使用两种单独的方法来控制 KSM：

- **ksm** 服务启动并停止 KSM 内核线程。
- **ksmtuned** 服务控制并调整 **ksm**，动态管理同一页面合并。**ksmtuned** 服务启动 **ksm**，并在不需要内存共享时停止 **ksm** 服务。**ksmtuned** 服务必须使用 ***retune*** 参数告知在创建或销毁新客户机时运行。

这两个服务都使用标准服务管理工具进行控制。

KSM 服务

ksm 服务包含在 qemu-kvm 软件包中。在 Red Hat Enterprise Linux 6 中默认禁用 KSM。但是，当使用 Red Hat Enterprise Linux 6 作为 KVM 主机物理机器时，它可能会被 **ksm/ksmtuned** 服务打开。

如果没有启动 **ksm** 服务，则 KSM 仅共享 2000 个页面。这个默认值较低，并提供有限的内存保存优势。

启动 **ksm** 服务后，KSM 将最多共享主机物理机器系统主内存的一半。启动 **ksm** 服务，使 KSM 能够共享更多内存。

```
# service ksm start
Starting ksm:                                [  OK  ]
```

ksm 服务可以添加到默认启动序列中。使用 chkconfig 命令使 **ksm** 服务持久。

```
# chkconfig ksm on
```

KSM 调整服务

ksmtuned 服务没有任何选项。**ksmtuned** 服务循环并调整 **ksm**。当客户机虚拟机被创建或销毁时，libvirt 会通知 **ksmtuned** 服务。

```
# service ksmtuned start
Starting ksmtuned:                           [  OK  ]
```

可以使用 ***retune*** 参数调整 **ksmtuned** 服务。***retune*** 参数指示 **ksmtuned** 手动运行调整功能。

在更改文件中的参数前，需要说明几个术语：

- **thres** - 激活阈值，单位为字节。当把 ***thres*** 值添加到所有 ***qemu-kvm*** 进程 RSZ 超过总系统内存时，会触发 KSM 周期。这个参数在 ***KSM_THRES_COEF*** 中定义的百分比的 kbytes 中相同。

/etc/ksmtuned.conf 文件是 **ksmtuned** 服务的配置文件。以下文件输出是默认的 **ksmtuned.conf** 文件。

```
# Configuration file for ksmtuned.

# How long ksmtuned should sleep between tuning adjustments
# KSM_MONITOR_INTERVAL=60

# Millisecond sleep between ksm scans for 16Gb server.
# Smaller servers sleep more, bigger sleep less.
# KSM_SLEEP_MSEC=10

# KSM_NPAGES_BOOST is added to the npages value, when free memory is less than thres.
# KSM_NPAGES_BOOST=300

# KSM_NPAGES_DECAY Value given is subtracted to the npages value, when free memory is
greater than thres.
# KSM_NPAGES_DECAY=-50

# KSM_NPAGES_MIN is the lower limit for the npages value.
# KSM_NPAGES_MIN=64

# KSM_NAGES_MAX is the upper limit for the npages value.
# KSM_NPAGES_MAX=1250

# KSM_TRES_COEF - is the RAM percentage to be calculated in parameter thres.
# KSM_THRES_COEF=20

# KSM_THRES_CONST - If this is a low memory system, and the thres value is less than
KSM_THRES_CONST, then reset thres value to KSM_THRES_CONST value.
# KSM_THRES_CONST=2048
```

```
# uncomment the following to enable ksmtuned debug information
# LOGFILE=/var/log/ksmtuned
# DEBUG=1
```

KSM 变量和监控

KSM 将监控数据存储在 **/sys/kernel/mm/ksm/** 目录中。这个目录中的文件由内核更新，是 KSM 使用和统计数据的准确记录。

以下列表中的变量也是 **/etc/ksmtuned.conf** 文件中的可配置变量，如下所示。

/sys/kernel/mm/ksm/ 文件

full_scans
完整扫描运行.

pages_shared
共享的总页面。

pages_sharing
现已共享的页面.

pages_to_scan
页面未扫描。

pages_unshared
页面不再共享。

pages_volatile
易失性页面的数量。

run
KSM 进程是否在运行。

sleep_millisecs
sleep 毫秒.

如果 ***DEBUG=1*** 行添加到 **/etc/ksmtuned.conf** 文件中，则 KSM 调整活动将存储在 **/var/log/ksmtuned** 日志文件中。日志文件位置可使用 ***LOGFILE*** 参数更改。不建议更改日志文件位置，可能需要特殊配置 SELinux 设置。

取消激活 KSM

KSM 具有性能开销，对于某些环境或主机物理机器系统来说可能太大。

通过停止 **ksmtuned** 和 **ksm** 服务可停用 KSM。停止服务会停用 KSM，但重启后不会保留。

```
# service ksmtuned stop
Stopping ksmtuned:                         [  OK  ]
# service ksm stop
```

```
Stopping ksm:                                [  OK  ]
```

使用 **chkconfig** 命令永久取消激活 KSM。要关闭服务，请运行以下命令：

```
# chkconfig ksm off
# chkconfig ksmtuned off
```

重要

确保交换大小足以用于提交的 RAM，即使是 KSM。KSM 可减少相同或类似虚拟机的 RAM 使用量。可能会在没有足够 swap 空间的情况下使用 KSM 过量使用客户机，但不建议使用，因为客户机虚拟机内存使用可能会导致页面变得不共享。

第 8 章 高级虚拟机管理

本章论述了高级管理工具，用于对客户机虚拟机进行微调和控制系统资源。

8.1. 控制组(CGROUPS)

Red Hat Enterprise Linux 6 提供了一个新的内核功能：*控制组，这些组* 通常称为 *cgroups*。Cgroups 允许您在系统上运行的用户定义的任务组之间分配资源，如 CPU 时间、系统内存、网络带宽或这些资源的组合。您可以监控您配置的 cgroups，拒绝 cgroups 对某些资源的访问，甚至可在运行中的系统上动态重新配置 cgroups。

libvirt 完全支持 cgroup 功能。默认情况下，**libvirt** 将每个 guest 放入各种控制器（如内存、cpu、blkio、device）的独立控制组中。

当 guest 启动时，它已在 cgroup 中。唯一可能需要的配置是 cgroups 的设置。有关 cgroups 的更多信息，请参阅 *Red Hat Enterprise Linux 资源管理指南*。

8.2. 巨页支持

这部分提供有关巨页支持的信息。

简介

x86 CPU 通常可在 4kB 页面中解决内存，但它们可以使用更大的页面，称为 *大页面*。KVM 客户机可以使用巨页内存支持进行部署，从而通过根据 transaction *Lookaside Buffer(TLB)* 增加 CPU 缓存点击来提高性能。巨页可能会显著提高性能，特别是大型内存和内存密集型工作负载。Red Hat Enterprise Linux 6 通过使用巨页增加页面大小，可以更有效地管理大量内存。

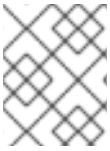
通过将大页面用于 KVM 客户机，减少了页表和 TLB 未命中的内存，因此显著降低性能，特别是用于内存密集型情况。

透明大内存页

透明大内存页 (THP)是一个内核功能，可减少应用程序所需的 TLB 条目。通过还允许所有可用内存用作缓存，性能也会提高。

要使用透明大内存页，需要在 **qemu.conf** 文件中不需要特殊配置。如果将 **/sys/kernel/mm/redhat_transparent_hugepage/enabled** 设置为 **always**，则默认使用巨页。

透明巨页不会阻止使用 **hugetlbfs** 功能。但是，如果没有使用 hugetlbfs，KVM 将使用透明的巨页，而不是常规的 4kB 页面大小。



注意

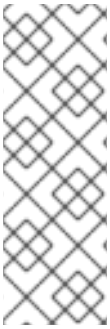
有关使用巨页 调整内存性能的步骤，请参阅 Red Hat Enterprise Linux 7 虚拟化调整和优化指南。

8.3. 在 HYPER-V HYPERVISOR 中将 RED HAT ENTERPRISE LINUX 作为虚拟机运行

可以在运行 Microsoft Windows Hyper-V hypervisor 的 Microsoft Windows 主机物理机器上运行 Red Hat Enterprise Linux 客户机虚拟机。特别是，已进行了以下改进，以便更轻松地部署和管理 Red Hat Enterprise Linux 客户机虚拟机：

- 升级的 VMBUS 协议 - VMBUS 协议已升级到 Windows 8 级别。作为这一工作的一部分，现在可以在客户端中的所有可用虚拟 CPU 上处理 VMBUS 中断。另外，Red Hat Enterprise Linux 客户机虚拟机和 Windows 主机物理机器之间的信号协议已被优化。
- 概要帧缓冲驱动程序 - 为 Red Hat Enterprise Linux 桌面用户提供增强的图形性能和出色的解决方案。
- 实时虚拟机备份支持 - 为实时 Red Hat Enterprise Linux 客户机虚拟机提供不间断备份支持。
- 固定大小 Linux VHD 的动态扩展 - 允许扩展实时挂载的固定大小 Red Hat Enterprise Linux VHD.

如需更多信息，请参阅以下文档：在 Windows Server 2012 R2 Hyper-V 上启用 Linux 支持 。



注意

如果最后一个分区后有可用空间，Hyper-V hypervisor 支持在 Red Hat Enterprise Linux 客户机中缩小 GPT 分区的磁盘，方法是允许用户丢弃磁盘的最后部分。但是，此操作将静默删除磁盘上的二级 GPT 标头，当客户机检查分区表时可能会生成错误消息（例如，使用 **parted**打印分区表时）。这是 Hyper-V 的已知限制。作为临时解决方案，可以使用 **gdisk** 和 **e** 命令中的专家菜单在缩小 GPT 磁盘后手动恢复二级 GPT 标头。另外，使用 Hyper-V Manager 中的"expand"选项还会将 GPT 次要标头放在磁盘末尾以外的位置，但可以使用 **parted** 移动。有关这些命令的更多信息，请参阅 **gdisk** 和 **parted** man page。

8.4. 客户机虚拟机内存分配

以下流程演示了如何为客户机虚拟机分配内存。这个分配和分配只在启动时自动启动，对任何内存值的任何更改都不会在下一次重启时生效。每个客户机可以分配的最大内存为 4 TiB，提供此内存分配不多于主机物理机器资源可以提供的。

有效的内存单元包括：

- ***b bytes*** 用于字节
- ***KB*** 对于千字节（10^3 或块 1000 字节）
- ***k*** 或 ***KiB*** 用于 kibibytes（2^{10} 或块 1024 字节）
- ***MB*** 兆字节（10 个^6 或块 1,000,000 字节）
- ***M*** 或者 ***MiB*** 用于兆字节（2^{20} 或块 1,048,576 字节）
- ***GB*** 千兆字节（10^9 或块 1,000,000,000 字节）
- ***G*** 或 ***GiB*** 用于千兆字节（230^个 或块为 1,073,741,824 字节）
- ***TB*** 太字节（10^{12} 或块 1,000,000,000 字节）
- ***T*** 或者 ***TiB*** 用于 tebibytes（2^{40} 或块 1,099,511,627,776 字节）

请注意，所有值将被 libvirt 舍入到最接近的基位字节，并可进一步舍入为管理程序支持的粒度。有些虚拟机监控程序还至少强制实施，如 4000KiB（或 4000 x 2^{10} 或 4,096,000 字节）。这个值的单位由可选属性 ***memory unit*** 决定，它默认为 kibibytes(KiB)作为测量结果单位，其中给出的值乘以 2^{10} 或 1024 字节的块。

如果客户机虚拟机崩溃的可选属性 ***dumpCore***，则可用来控制客户机虚拟机的内存是否应该包含在生成的 coredump(***dumpCore='on'***)中，或者不包含(***dumpCore='off'***)。请注意，默认设置为 ***on***，因此如果参数没有设置为 ***off***，则客户机虚拟机内存将包含在 coredump 文件中。

currentMemory 属性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，允许即时对客户机虚拟机内存进行膨胀。如果省略此项，则默认为与 memory 元素相同的值。unit 属性的行为与内存的行为相同。

在本节的所有情况下，需要更改域 XML，如下所示：

```
<domain>

  <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 coredump file -->
  <currentMemory unit='KiB'>524288</currentMemory>
  <!-- makes the current memory unit 524288 KiB -->
  ...
</domain>
```

8.5. 自动启动客户机虚拟机

本节介绍如何在主机物理机器系统的引导阶段自动启动 guest 虚拟机。

这个示例使用 **virsh** 设置客户机虚拟机 ***TestServer***，在主机物理机器引导时自动启动。

```
# virsh autostart TestServer
Domain TestServer marked as autostarted
```

现在，guest 虚拟机会从主机物理机器自动启动。

要停止客户机虚拟机，使用 ***--disable*** 参数

```
# virsh autostart --disable TestServer
Domain TestServer unmarked as autostarted
```

客户机虚拟机不再从主机物理机器自动启动。

8.6. 为 GUEST 虚拟机禁用 SMART 磁盘监控

SMART 磁盘监控可以安全地禁用为虚拟磁盘，物理存储设备则由主机物理计算机管理。

```
# service smartd stop
# chkconfig --del smartd
```

8.7. 配置 VNC 服务器

要配置 VNC 服务器，**在 System > Preferences 中使用 Remote Desktop** 应用程序。或者，您可以运行 **vino-preferences** 命令。

使用以下命令设置专用的 VNC 服务器会话：

如果需要，请创建，然后编辑 **~/.vnc/xstartup** 文件，以在每次 **vncserver** 启动时启动 GNOME 会话。第一次运行 **vncserver** 脚本时，它将询问您要用于 VNC 会话的密码。有关 vnc 服务器文件的更多信息，请参阅 *Red Hat Enterprise Linux 安装指南* 。

8.8. 生成新唯一 MAC 地址

在某些情况下，您需要为 guest 虚拟机生成一个新的和唯一的 MAC 地址。写入时没有可用的命令行工具生成新的 MAC 地址。以下提供的脚本可为您的 guest 虚拟机生成一个新的 MAC 地址。将脚本 在 guest 虚拟机上保存为 **macgen.py**。现在，您可以使用 **./macgen.py** 运行脚本，它会生成新的 MAC 地址。输出示例类似如下：

```
$ ./macgen.py
00:16:3e:20:b0:11
```

```
#!/usr/bin/python
# macgen.py script to generate a MAC address for guest virtual machines
#
import random
#
def randomMAC():
 mac = [ 0x00, 0x16, 0x3e,
  random.randint(0x00, 0x7f),
  random.randint(0x00, 0xff),
  random.randint(0x00, 0xff) ]
 return ':'.join(map(lambda x: "%02x" % x, mac))
#
print randomMAC()
```

8.8.1. 为 guest 虚拟机生成新 MAC 的另一个方法

您还可以使用 **python-virtinst** 的内置模块生成一个新的 MAC 地址和 **UUID**，以便在客户机虚拟机配置文件中使用：

```
# echo  'import virtinst.util ; print\
 virtinst.util.uuidToString(virtinst.util.randomUUID())' | python
# echo  'import virtinst.util ; print virtinst.util.randomMAC()' | python
```

以上脚本也可以作为脚本文件实施，如下所示。

```
#!/usr/bin/env python
#  -*- mode: python; -*-
print ""
print "New UUID:"
import virtinst.util ; print virtinst.util.uuidToString(virtinst.util.randomUUID())
print "New MAC:"
import virtinst.util ; print virtinst.util.randomMAC()
print ""
```

8.9. 改进客户机虚拟机响应时间

客户机虚拟机有时可能会慢慢，以响应某些工作负载和使用模式。可能导致缓慢或无响应的客户机虚拟机的示例：

- 严重过度分配的内存。
- 使用高处理器用量过量使用内存
- 其他（而非 **qemu-kvm** 进程）在主机物理机器上忙或者停止进程。

KVM 客户机虚拟机作为 Linux 进程运行。Linux 进程不会永久保存在主内存（物理 RAM）中，并且将被放入交换空间（虚拟内存）中，尤其是不使用它们。如果客户机虚拟机长时间不活跃，主机物理机器内核可将 guest 虚拟机移动到交换中。因为 swap 比物理内存慢，它可能会显示客户机不响应。当客户机加载到主内存后，这个更改。请注意，将客户机虚拟机从交换到主内存的过程可能需要几秒钟，每个分配给客户机虚拟机的 RAM 字节需要几秒钟，具体取决于用于交换的存储类型和组件的性能。

无论内存被过度分配还是总体内存用量，KVM 客户机虚拟机进程可能会被移动到交换中。

不建议使用不安全的过量使用级别，或在交换的情况下关闭客户机虚拟机进程或其他关键进程。在过量使用内存时，始终确保主机物理机器有足够的交换空间。

有关使用 KVM 过量使用的详情，请参考 第 6 章 *使用 KVM 进行过量使用*。

警告

虚拟内存允许 Linux 系统使用的内存超过系统中物理 RAM 的内存。在被使用的进程交换出来时，允许活动进程使用内存，从而提高了内存利用率。禁用交换可减少内存利用率，因为所有进程都存储在物理 RAM 中。

如果关闭交换，请不要过量使用客户机虚拟机。过量使用没有交换的客户机虚拟机可能会导致客户机虚拟机或主机物理机器系统崩溃。

8.10. 使用 LIBVIRT 管理虚拟机计时器

保证虚拟客户机的准确时间是虚拟化平台的关键挑战。不同的虚拟机监控程序试图以各种方式处理时间问题。libvirt 为时间管理提供管理程序独立的配置设置，使用域 XML 中的 <clock> 和 <timer> 元素。可以使用 **virsh edit** 命令编辑域 XML。详情请查看 第 14.6 节 “编辑客户机虚拟机的配置文件”。

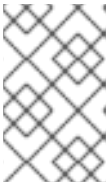
<clock> 元素用于确定客户端虚拟机时钟如何与主机物理机器时钟同步。clock 元素具有以下属性：

- **偏移** 决定了客户机虚拟机时钟在主机物理时钟中的偏移方式。offset 属性具有以下可能的值：

表 8.1. 偏移属性值

值	描述
utc	启动时，客户机虚拟机时钟将同步到 UTC。
localtime	客户端虚拟机时钟将在引导时同步到主机物理机器配置的时区（若有）。
timezone	客户机虚拟机时钟将同步到指定时区，由 ***timezone*** 属性指定。

值	描述
变量	客户机虚拟机时钟将与 UTC 中的任意偏移同步。增量相对于 UTC 的使用 ***adjustment*** 属性以秒为单位指定。客户机虚拟机可以自由地调整 Real Time Clock(RTC)，期望它在下次重启时将生效。这与 ***utc*** 模式不同，每次重启时会丢失任何 RTC 调整。



注意

默认情况下，**utc** 值被设置为虚拟机中的时钟偏移。但是，如果客户端虚拟机时钟使用 **localtime** 值运行，则需要将时钟偏移改为不同的值，以便使客户端虚拟机时钟与主机物理机器时钟同步。

- **timezone** 属性决定 guest 虚拟机时钟使用哪个时区。
- **adjustment** 属性提供客户机虚拟机时钟同步的 delta。以秒为单位，相对于 UTC。

例 8.1. 始终与 UTC 同步

```
<clock offset="utc" />
```

例 8.2. 始终与主机物理机器时区同步

```
<clock offset="localtime" />
```

例 8.3. 同步到任意时区

```
<clock offset="timezone" timezone="Europe/Paris" />
```

例 8.4. 同步到 UTC + 任意偏移

```
<clock offset="variable" adjustment="123456" />
```

8.10.1. 时钟的计时器子元素

clock 元素可以有零个或多个计时器元素作为子项。timer 元素指定用于客户端虚拟机时钟同步的时间源。timer 元素具有以下属性：只有 ***name*** 是必需的，所有其他属性都是可选的。

name 属性指定要使用的时间源类型，可以是以下之一：

表 8.2. 名称属性值

值	描述
pit	Programmable Interval Timer - 带有定期中断的计时器。
rtc	实时时钟 - 持续运行带有定期中断的计时器。
tsc	时间戳计数器 - 计算重置后空数，没有中断。
kvmclock	KVM 时钟 - KVM 客户机虚拟机的建议时钟源。KVM pvclock 或 kvm-clock 可让 guest 虚拟机读取主机物理机器的墙时钟时间。

8.10.2. track

trace 属性指定计时器跟踪的内容。仅对名称值 **rtc** 有效。

表 8.3. 跟踪属性值

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。
guest	RTC 始终跟踪 guest 虚拟机时间。
Wall	RTC 始终跟踪主机时间。

8.10.3. tickpolicy

tickpolicy 属性分配用于将 ticks 传递给 guest 虚拟机的策略。可接受以下值：

表 8.4. tickpolicy 属性值

值	描述
delay	继续以正常率提供（so ticks 延迟）。
catchup	以更高的速度来计算。
merge	勾选合并为一个单行.
discard	所有错过的点数都将被丢弃。

8.10.4. 频率、模式和存在

frequency 属性用于设置固定频率，以 Hz 为单位。只有 **name** 元素的值为 **tsc** 时，此属性才相关。所有其他计时器以固定频率（**pit**、**rtc**） 运行。

模式 决定了如何向客户机虚拟机公开时间源。此属性仅与 **tsc** 的 name 值相关。所有其他计时器都是始终模拟的。命令如下： &lt **;timer name='tsc' frequency='NN' mode='auto|native|emulate|smpsafe'/>**。模式定义在 表中指定。

表 8.5. 模式属性值

值	描述
auto	如果 TSC 不稳定，则原生允许本地 TSC 访问。
原生	始终允许原生 TSC 访问。
模拟	始终模拟 TSC。
smpsafe	始终模拟 TSC 和 Interlock SMP

present 用于覆盖对客户机虚拟机可见的默认计时器。

表 8.6. present 属性值

值	描述
是	使这个计时器强制到对 guest 虚拟机可见的。
否	强制此计时器对客户机虚拟机不可见。

8.10.5. 使用时钟同步示例

例 8.5. 时钟与 RTC 和 PIT 计时器同步

在这个示例中，时钟与 RTC 和 PIT 计时器同步到本地时间

```
<clock offset="localtime">
 <timer name="rtc" tickpolicy="catchup" track="guest virtual machine" />
 <timer name="pit" tickpolicy="delay" />

</clock>
```

注意

PIT 时钟源可以与在 64 位主机（无法使用 PIT）中运行的 32 位客户机一起使用，并满足以下条件：

- 客户机虚拟机只能有一个 CPU
- APIC 计时器必须被禁用（使用 "noapictimer" 命令行选项）
- 在 guest 中必须禁用 NoHZ 模式（使用 "nohz=off" 命令行选项）
- 在客户机中必须禁用高分辨率计时器模式（使用 "highres=off" 命令行选项）
- PIT 时钟源与高分辨率计时器模式或 NoHz 模式不兼容。

8.11. 使用 PMU 监控客户机虚拟机性能

在红帽企业 Linux 6.4 中，vPMU（虚拟 PMU）作为技术预览提供。vPMU 基于 Intel 的 PMU（性能监控单元），且只能在 Intel 机器上使用。PMU 允许跟踪指示 guest 虚拟机运行情况的统计信息。

通过使用性能监控，开发人员可以在使用性能工具进行性能分析时使用 CPU 的 PMU 计数器。虚拟性能监控单元功能让虚拟机用户可以识别客户机虚拟机中可能出现性能问题的来源，从而改进对 KVM 客户机虚拟机进行性能分析的能力。

要启用此功能，必须设置 **-cpu host** 标志。

这个功能只支持运行 Red Hat Enterprise Linux 6 的客户机虚拟机，且默认禁用。此功能只能使用 Linux perf 工具。使用以下命令确保安装了 perf 软件包：

```
# yum install perf.
```

有关 perf 命令的更多信息，请参阅 **perf** 的 man page。

8.12. 虚拟机电源管理

通过更改 Libvirt 的 Domain XML 中的以下参数，可以强制启用或禁用到客户端虚拟机的 BIOS 公告：

```
...
  <pm>
    <suspend-to-disk enabled='no'/>
    <suspend-to-mem enabled='yes'/>
  </pm>
  ...
```

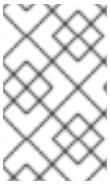
元素 ***pm*** 启用（是'yes）或禁用('no')对 S3（暂停到磁盘）和 S4（暂停到内存）的 BIOS 支持。如果未指定任何内容，则管理程序将保留为默认值。

第 9 章 客户机虚拟机设备配置

Red Hat Enterprise Linux 6 支持适用于客户机虚拟机的三类设备：

- *模拟设备是* 纯粹的虚拟设备，可模拟实际硬件，允许未经修改的虚拟机操作系统使用其标准驱动程序来配合它们。Red Hat Enterprise Linux 6 支持 216 virtio 设备。
- *VirtIO 设备* 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，但非 Linux 虚拟机不包括它们默认需要的驱动程序。虚拟机管理器(**virt-manager**)和 Red Hat Virtualization Hypervisor(RHV-H)等虚拟化管理软件自动为支持的非 Linux 客户机操作系统安装这些驱动程序。Red Hat Enterprise Linux 6 支持多达 700 个 scsi 磁盘。
- *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为一系列任务对 PCI 设备进行独占访问，并允许 PCI 设备出现，就像它们物理附加到客户端操作系统一样。Red Hat Enterprise Linux 6 支持每个虚拟机最多 32 个分配的设备。

设备分配在 PCIe 设备上受到支持，包括所选图形设备。现在，Red Hat Enterprise Linux 6 中的设备分配支持 Nvidia K-series Quadro、GRID 和 Tesla 图形卡 GPU 功能。并行 PCI 设备可能作为分配的设备被支持，但由于安全和系统配置冲突，这些设备有严重的限制。



注意

附加到虚拟机的设备数量取决于多个因素。一个因素是 QEMU 进程打开的文件数（在 **/etc/security/limits.conf** 中配置，该文件可以被 **/etc/libvirt/qemu.conf**覆盖）。其他限制因素包括虚拟总线上可用的插槽数，以及 sysctl 设置的打开文件的系统范围限制。

有关具体设备和限制的详情，请参考 第 20.16 节 “Devices”。

Red Hat Enterprise Linux 6 支持作为虚拟机单一功能插槽提供的 PCI 热插拔设备。可以将单一功能主机设备和多功能主机设备配置为启用此功能。建议您只针对非热拔应用程序将设备公开为多功能 PCI 插槽。

注意

需要平台支持中断重新映射，才能将客户机与主机分配的设备隔离开来。如果没有这种支持，主机可能会容易受到恶意虚拟机的注入攻击。在客户机被信任的环境中，管理员可能会选择使用 allow_unsafe_interrupts 选项为 ***vfio_iommu_type1*** 模块使用 **allow_unsafe_interrupts** 选项进行 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，包含以下内容：

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

或者动态使用 sysfs 条目进行相同的操作：

```
# echo 1 > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
```

9.1. PCI 设备

PCI 设备分配仅适用于支持 Intel VT-d 或 AMD IOMMU 的硬件平台。在 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 9.1. 为 PCI 设备分配准备 Intel 系统

1. **启用 Intel VT-d 规格**

Intel VT-d 规格为直接为虚拟机分配物理设备提供了硬件支持。这些规格需要使用 Red Hat Enterprise Linux 的 PCI 设备分配。

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。用于参考这些规格的条款在制造商之间有所不同；请参考您的系统厂商文档来获取适当的条款。

2. **在内核中激活 Intel VT-d**
在 **/etc/sysconfig/grub** 文件中，在 GRUB_CMDLINX_LINUX 行的末尾添加 ***intel_iommu=on*** 参数来激活 Intel VT-d。

以下示例是已激活 Intel VT-d 的修改 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"
```

3. **重新生成配置文件**

运行以下命令重新生成 **/etc/grub2.cfg**：

```
grub2-mkconfig -o /etc/grub2.cfg
```

请注意，如果您使用基于 **UEFI** 的主机，则目标文件应为 **/etc/grub2-efi.cfg**。

4. **准备好使用**

重启系统以启用更改。您的系统现在可以进行 **PCI** 设备分配。

过程 **9.2.** 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。这些规格必须在 BIOS 中启用。有些系统制造商默认禁用这些规格。

2. 启用 **IOMMU** 内核支持

将 *amd_iommu=on* 附加到 GRUB_CMDLINX_LINUX 行的末尾，在引号中附加，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，则目标文件应为 /etc/grub2-efi.cfg。

4. 准备好使用

重启系统以启用更改。您的系统现在可以进行 PCI 设备分配。

9.1.1. 使用 virsh 分配 PCI 设备

这些步骤涵盖将 PCI 设备分配给 KVM 管理程序上的虚拟机。

这个示例使用 PCI 网络控制器，其 PCI 标识符代码为 pci_0000_01_00_0，以及一个名为 *guest1-rhel6-64* 的完整虚拟化客户机计算机。

过程 9.3. 使用 virsh 将 PCI 设备分配给客户端虚拟机

1. 识别该设备

首先，识别为虚拟机分配设备指定的 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)
```

此以太网控制器显示短标识符 00:19.0。我们需要找出 virsh 使用的完整标识符，以便将此 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输出。

本例突出显示映射到以太网控制器的字符串，其 ID 为 00:19.0。在本例中，: 和 . 字符在完整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
```

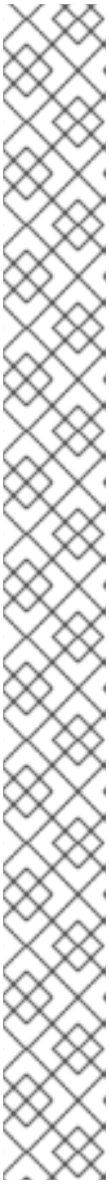
记录映射到您要使用的设备的 **PCI** 设备编号；在其他步骤中，这是必需的。

2. 查看设备信息

有关域、总线和功能的信息可在 **virsh nodedev-dumpxml** 命令的输出中获取：

```
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 从 IOMMU 的角度看设备的可见性和隔离决定。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 组内的所有端点才能分配给一个客户端。这可通过向客户机分配额外端点或使用 virsh nodedev-detach 从主机驱动程序分离来实现。单个组中包含的设备不能在多个虚拟机之间分割或分割在主机和客户机之间。非端点设备（如 PCIe 根端口、交换机端口和网桥）不应与主机驱动程序分离，且不会干扰端点的分配。

可以使用 virsh nodedev-dumpxml 输出的 iommuGroup 部分来确定 IOMMU 组中的设备。组的每个成员都通过单独的 "address" 字段提供。这些信息也可以在 sysfs 中使用：

```
$ ls /sys/bus/pci/devices/0000:01:00.0/iommu_group/devices/
```

输出的示例如下：

```
0000:01:00.0  0000:01:00.1
```

要只为客户机分配 0000.01.00.0，未使用的端点应当在启动客户机前从主机分离：

```
$ 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，请参阅 virsh nodedev-dumpxml pci_0000_00_19_0 命令的输出。

示例设备具有以下值： bus = 0, slot = 25 和 function = 0。十进制配置使用以下三个值：

```
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 **< source>** 部分中添加一个设备条目，以将 **PCI** 设备分配到客户端虚拟机。

```
# virsh edit guest1-rhel6-64
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0' bus='0' slot='25' function='0'/>
  </source>
</hostdev>
```

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-rhel6-64 file.xml
```

5. 启动虚拟机

```
# virsh start guest1-rhel6-64
```

PCI 设备现在应当成功分配给虚拟机，并可以被客户端操作系统访问。

9.1.2. 使用 virt-manager 分配 PCI 设备

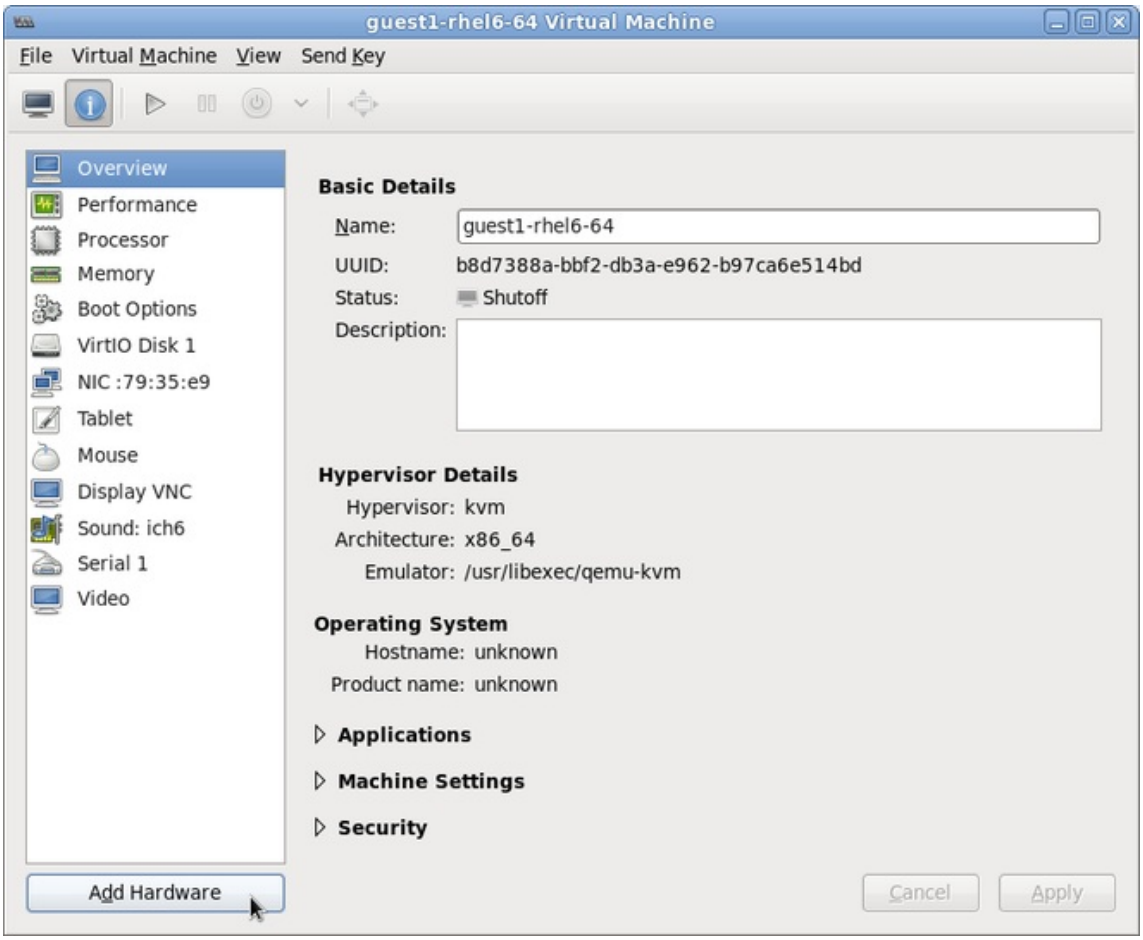
可以使用图形 **virt-manager** 工具将 **PCI** 设备添加到虚拟客户机中。以下流程将千兆位以太网控制器添加到客户机虚拟机。

过程 **9.4.** 使用 **virt-manager** 将 **PCI** 设备分配给客户端虚拟机

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 9.1. 虚拟机硬件信息窗口

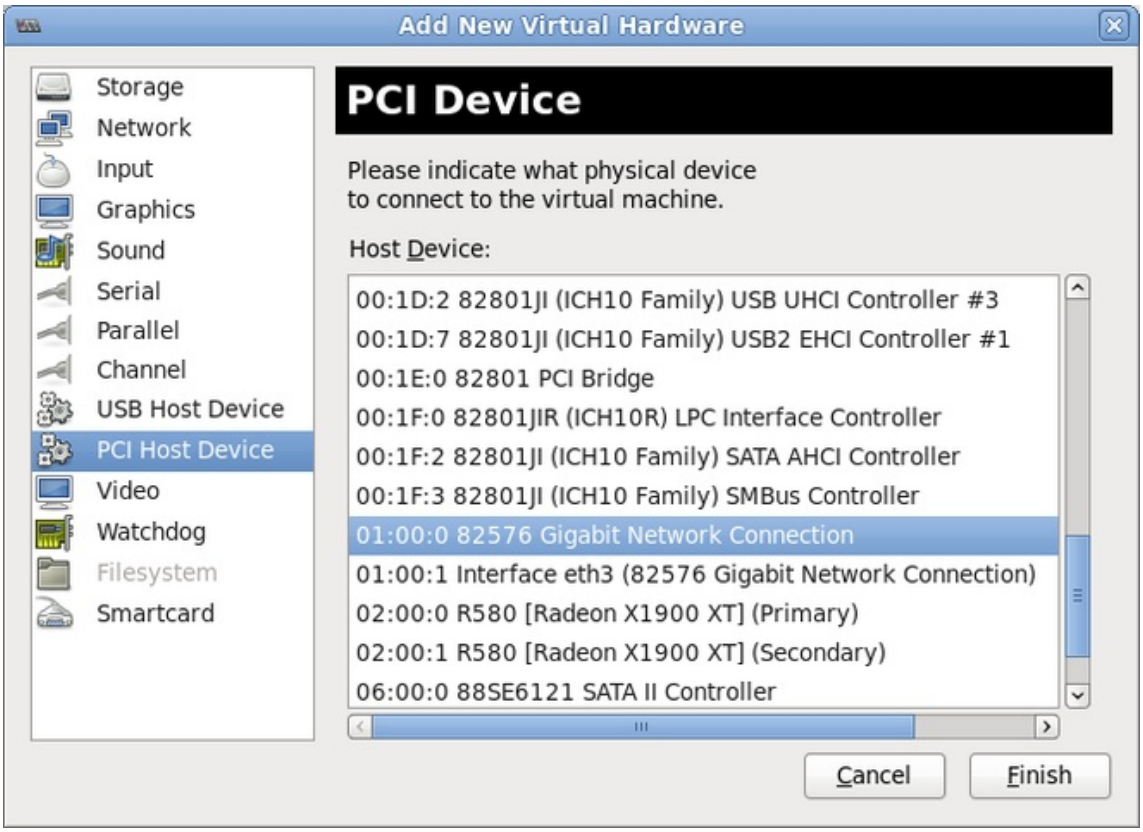


2. 选择 **PCI** 设备

从左侧的 **Hardware** 列表中选择 **PCI** 主机设备。

选择一个未使用的 **PCI** 设备。如果您选择了一个被另一个客户端使用的 **PCI** 设备，则可能会导致错误。在这个示例中使用备用 **82576** 网络设备。点 **Finish** 完成设置。

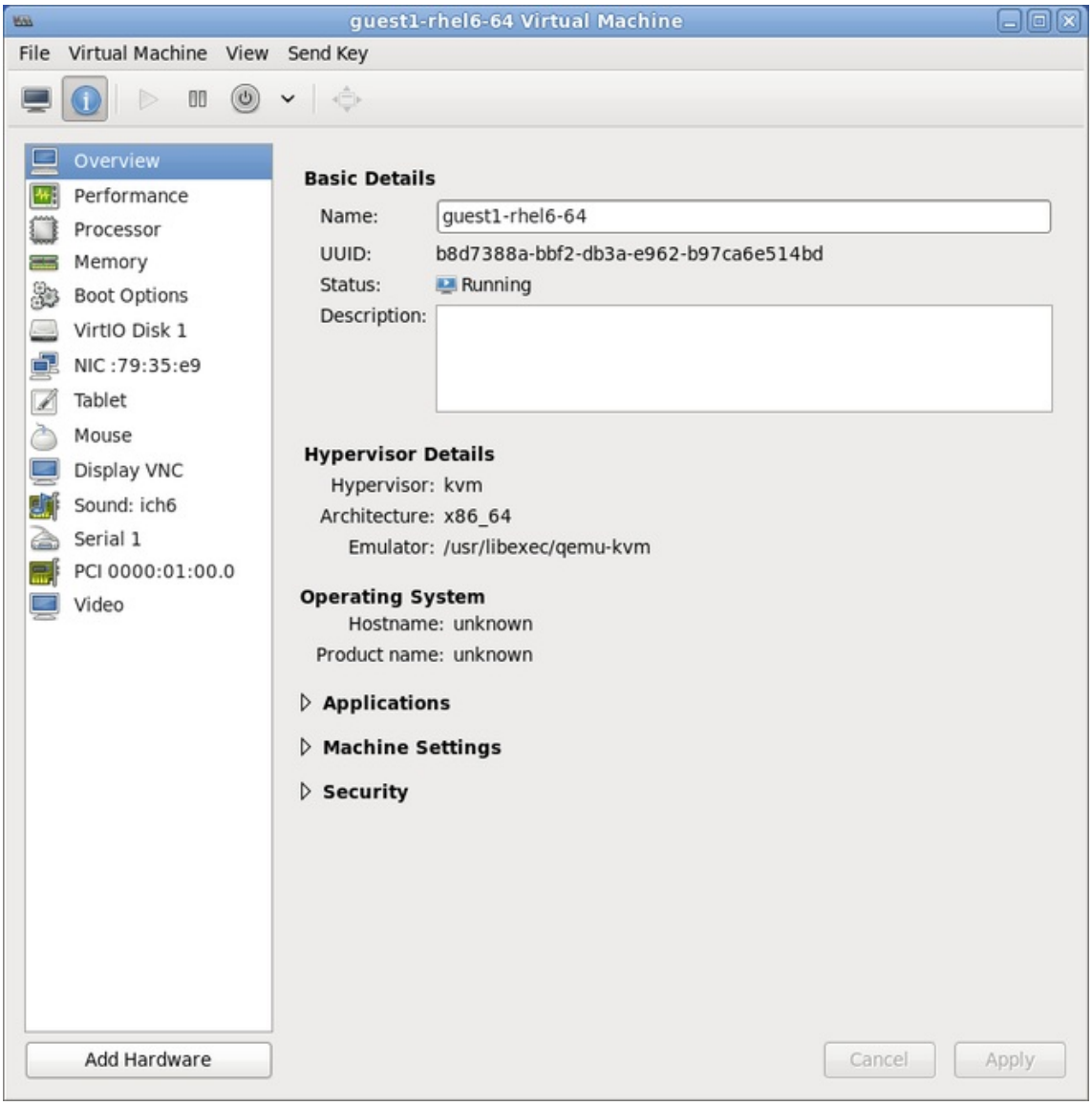
图 9.2. Add new virtual hardware 向导



3. **添加新设备**

   设置已完成，客户端虚拟机现在可以直接访问 PCI 设备。

图 9.3. 虚拟机硬件信息窗口



注意

如果设备分配失败，则仍附加到主机的同一 IOMMU 组中可能存在其他端点。无法利用 virt-manager 检索组信息，但 virsh 命令可以用来分析 IOMMU 组的绑定，以及必要的 sequester 设备。

有关 IOMMU 组以及如何使用 virsh分离端点设备的更多信息，请参阅 注意 中的 第 9.1.1 节 “使用 virsh 分配 PCI 设备”。

9.1.3. 使用 virt-install 的 PCI 设备分配

要使用 virt-install 分配 PCI 设备，使用 *--host-device* 参数。

过程 9.5. 使用 virt-install 为虚拟机分配 PCI 设备

1. 识别该设备

识别为客户端虚拟机分配设备指定的 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)
```

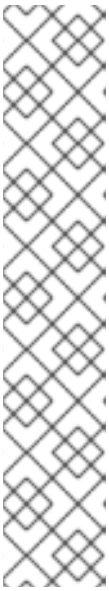
virsh nodedev-list 命令列出系统中附加的所有设备，并使用字符串识别每个 **PCI** 设备。要只将输出限制为 **PCI** 设备，请运行以下命令：

```
# 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；其他步骤中需要该数字。

有关域、总线和功能的信息可在 **virsh nodedev-dumpxml** 命令的输出中获取：

```
# virsh nodedev-dumpxml pci_0000_01_00_0
<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 **IOMMU** 组中有多个端点，而不是分配给客户机，则需要在启动客户端前手动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 9.1.1 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-rhel6-64 \
--disk path=/var/lib/libvirt/images/guest1-rhel6-64.img,size=8 \
--nonsparse --graphics spice \
--vcpus=2 --ram=2048 \
--location=http://example1.com/installation_tree/RHEL6.0-Server-x86_64/os \
--nonetworks \
```

```
--os-type=linux \
--os-variant=rhel6
--host-device=pci_0000_01_00_0
```

3. **完成安装**

   完成客户机安装。PCI 设备应当连接到客户端。

9.1.4. 分离分配的 PCI 设备

当为客户机机器分配了主机 PCI 设备时，主机将不再使用该设备。本部分论述了如何通过 virsh 或 virt-manager 从客户机中分离该设备，以便主机使用。

过程 9.6. 使用 virsh 从客户机中分离 PCI 设备

1. **分离该设备**

   使用以下命令，通过从客户机的 XML 文件中删除 PCI 设备来从客户机中分离它：

```
# virsh detach-device name_of_guest file.xml
```

2. **将设备重新关联到主机（可选）**

   如果设备处于 *managed* 模式，请跳过这一步。设备将自动返回到主机。

   如果该设备没有使用 *managed* 模式，使用以下命令将 PCI 设备重新附加到主机机器中：

```
# virsh nodedev-reattach device
```

   例如，要将 pci_0000_01_00_0 设备重新连接到主机：

```
virsh nodedev-reattach pci_0000_01_00_0
```

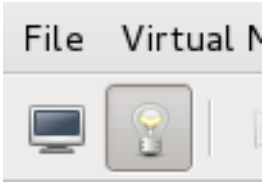
   现在，该设备可用于主机使用。

过程 9.7. 使用 virt-manager 从客户机分离 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，以显示虚拟硬件的列表。

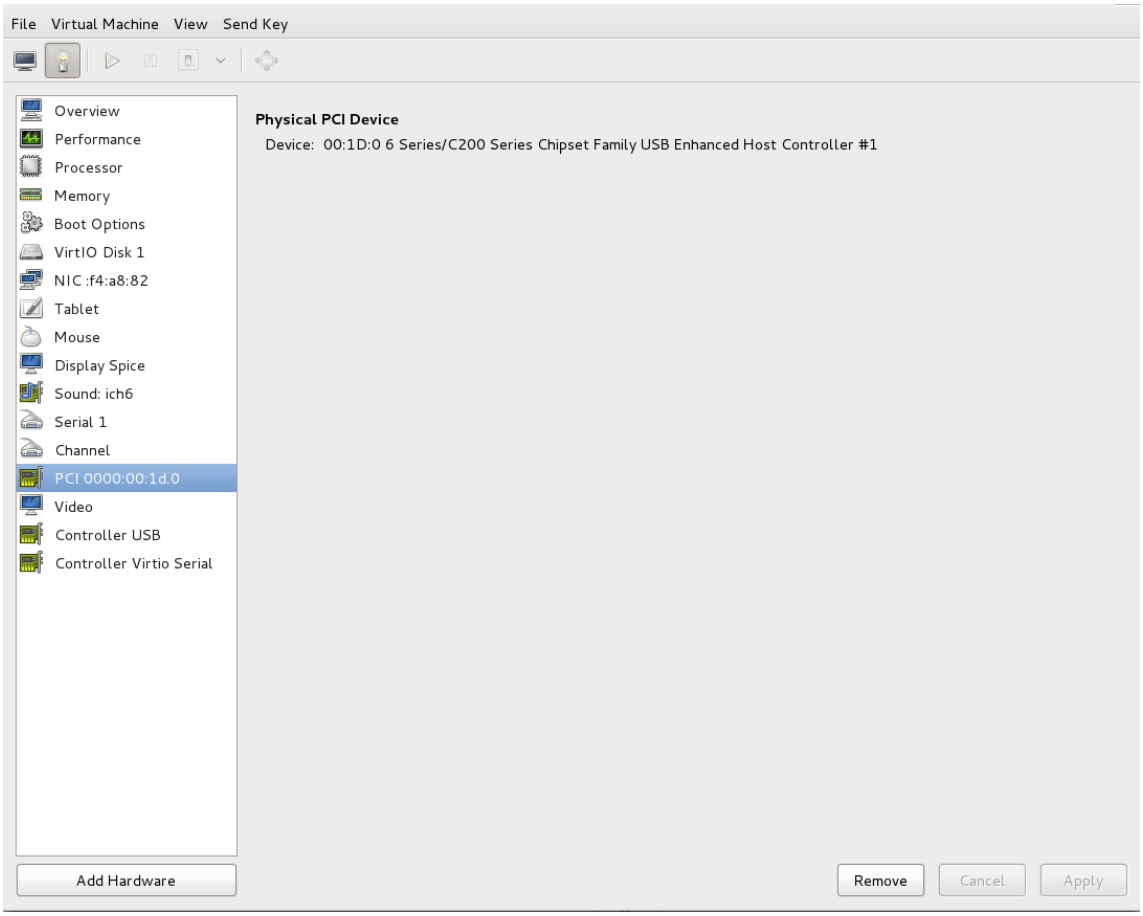
图 9.4. 虚拟硬件详细信息按钮



2. 选择并删除该设备

选择要从左侧面板中虚拟设备列表分离的 PCI 设备。

图 9.5. 选择要分离的 PCI 设备



单击删除按钮 以确认。现在，该设备可用于主机使用。

9.1.5. 创建 PCI 网桥

外围设备组件互连(PCI)网桥用于连接到设备，如网卡、模式和声音卡。正如其物理对应项一样，虚拟设备也可以附加到 PCI 网桥。过去，只能将 31 个 PCI 设备添加到任何客户端虚拟机。现在，当添加了 31st PCI 设备时，PCI 网桥将自动放在 31st 插槽中，将额外的 PCI 设备移到 PCI 网桥。每个 PCI 网桥具有 31 个插槽，用于 31 个附加设备，它们都可以桥接。这样，guest 虚拟机可以使用超过 900 个设备。

注意

当 guest 虚拟机正在运行时，无法执行该操作。您必须在要关闭的客户机虚拟机中添加 PCI 设备。

9.1.6. PCI Passthrough

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 guest，在第一个选择将设备的 MAC 地址设置为配置后，并使用可选的指定 <虚拟端口> 元素将设备与 802.1Qbh 相关联（请参阅上述为 type='direct' 网络设备的 virtualport 示例）。由于标准单端口 PCI 以太网卡驱动程序设计的限制 - 仅限 SR-IOV（Single Root I/O 虚拟化）虚拟功能(VF)设备，以此方式分配标准单端口 PCI 或 PCIe 以太网卡，请使用传统的 <hostdev> 设备定义。

要使用 VFIO 设备分配而不是传统的 KVM 设备分配(VFIO)是一种新的设备分配方法，与 UEFI 安全引导兼容，<type='hostdev'> 接口可以有一个可选的驱动程序子元素，并将 name 属性设置为 "vfio"。要使用旧的 KVM 设备分配，您可以将 name 设置为 "kvm"（或只省略 <驱动程序> 元素，因为 <driver='kvm'> 目前是默认值）。

注意

网络设备的智能透传与标准 <hostdev> 设备的功能相似，这种方法为通过设备指定 MAC 地址 <和虚拟端口>。如果不需要这些功能，或者您使用的是支持 SR-IOV 的标准单端口 PCI、PCIe 或 USB 网卡（因此，在分配到客户端域后，任何时候都会丢失配置的 MAC 地址），或者如果您使用比 0.9.11 旧的 libvirt 版本，您应该使用标准 <hostdev> 将设备分配给 guest，而不是 <接口 type=host/dev'>。

图 9.6. 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>
```

9.1.7. 使用 SR-IOV 设备配置 PCI 分配(Passthrough)

本节仅用于 SR-IOV 设备。SR-IOV 网卡提供多个 *虚拟功能* (VF)，可分别使用 PCI 设备分配来单独分

配给客户机虚拟机。分配之后，每个设备将作为完整的物理网络设备。这允许多个客户机虚拟机获得直接 PCI 设备分配的性能优势，同时仅在主机物理机器上使用单一插槽。

这些 VF 可使用元素 <hostdev> 为虚拟客户机分配，但因为 SR-IOV VF 网络设备没有永久唯一的 MAC 地址，从而导致客户端虚拟机的网络设置在每次重新引导主机物理机器时都必须重新配置问题。要补救这一点，您需要在将 VF 分配给主机物理机器前设置 MAC 地址，每次客户端虚拟机引导时都需要设置它。要分配这个 MAC 地址以及其他选项，请参阅 过程 9.8, “配置 MAC 地址、vLAN 和虚拟端口，以便在 SR-IOV 中分配 PCI 设备” 中描述的步骤。

过程 9.8. 配置 MAC 地址、vLAN 和虚拟端口，以便在 SR-IOV 中分配 PCI 设备

务必要注意，<hostdev> 元素不能用于特定于功能的项目，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> 元素不是 <hostdev> 的有效子项。当它们对 <接口> 有效时，增加了对新接口类型的支持(<interface type='hostdev'>)。这个新接口设备类型作为接口和 <hostdev> 的混合运行。<>因此，在为客户机虚拟机分配 PCI 设备前，libvirt 在虚拟机 XML 配置文件中初始化特定于网络的硬件/交换机（例如设置 MAC 地址、设置 vLAN 标签或与 802.1Qbh 交换机关联）。有关设置 vLAN 标签的详情，请参考 第 18.14 节 “设置 vLAN Tags”。

1. 关闭客户机虚拟机

   使用 virsh shutdown 命令（请参考 第 14.9.1 节 “关闭客户机虚拟机”），关闭名为 *guestVM* 的客户机虚拟机。

   ```
   # virsh shutdown guestVM
   ```

2. 收集信息

   要使用 <接口 type='hostdev'>，您必须有一个支持 SR-IOV 功能的网卡，主机物理机器硬件支持 Intel VT-d 或 AMD IOMMU 扩展，您必须知道您要分配的 VF 的 PCI 地址。

3. 打开 XML 文件进行编辑

   运行 # virsh save-image-edit 命令以打开 XML 文件进行编辑（详情请参阅 第 14.8.10 节 “编辑域 XML 配置文件”）。由于您要将 guest 虚拟机恢复到以前的运行状态，在这种情况下，--running 将用在内。本例中的配置文件的名称是 *guestVM.xml*，因为客户端虚拟机的名称是 *guestVM*。

   ```
   # virsh save-image-edit guestVM.xml --running
   ```

   在默认编辑器中打开 *guestVM.xml*。

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 9.7. 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>
```

请注意，如果您不提供 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，只有在您要连接到 802.11Qgh 硬件开关(802.11Qbg)(a.k.a)时，才会使用 <virtualport> 元素。"VEPA"）交换机当前不受支持。

5. 重新启动客户机虚拟机

运行 virsh start 命令，以重新启动您在第一步中关闭的 guest 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 14.8.1 节 “启动定义的域”。

```
# virsh start guestVM
```

当客户机虚拟机启动时，它会看到由物理主机的适配器（带有配置的 MAC 地址）提供的网络设备。此 MAC 地址在客户机虚拟机之间保持不变，主机物理机器重新引导。

9.1.8. 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限制：

- *指定的 VF 必须在每次启动客户机虚拟机时可用，这意味着管理员必须将每个 VF 永久分配给单个客户端虚拟机（或者修改每个 guest 虚拟机的配置文件，以指定目前未使用的 VF 的 PCI 地址）。*

- 如果 guest 虚拟机被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 总线上同一位置上有相同的硬件（或者，启动之前必须修改客户机虚拟机配置）。

通过创建一个包含 SR-IOV 设备所有 VF 的 libvirt 网络，可以避免这两个问题。完成后，您要将 guest 虚拟机配置为引用这个网络。每次启动 guest 时，将从池中分配一个 VF，并分配给客户机虚拟机。当客户机虚拟机停止时，VF 将返回给池，供其他虚拟客户机使用。

过程 9.9. 创建设备池

1. 关闭客户机虚拟机

   使用 virsh shutdown 命令（请参考 第 14.9 节 “关闭客户机虚拟机的关闭、重新启动和关闭”），关闭名为 guestVM 的客户机虚拟机。

   ```
   # virsh shutdown guestVM
   ```

2. 创建配置文件

   使用您选择的编辑器在 /tmp 目录中创建 XML 文件（例如：throughthrough .xml ）。确保将 pf dev='eth3' 替换为您自己的 SR-IOV 设备的 PF 的 netdev 名称

   以下是一个网络定义示例，它会在主机 物理机器上使用其物理功能(PF) 为 SR-IOV 适配器提供所有 VF 池：

   图 9.8. 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. 加载新的 XML 文件

   运行以下命令，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. 重新启动客户机虚拟机

运行 virsh start 命令，以重新启动您在第一步中关闭的 guest 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 14.8.1 节 “启动定义的域”。

```
# virsh start guestVM
```

6. 为设备启动 passthrough

虽然只显示单个设备，但 libvirt 会在其域 XML 中使用接口定义（如下所示）在 PF 首次启动时，自动获得与 PF 相关联的所有 VF 列表：

图 9.9. 接口网络定义的域 XML 示例

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

7. 验证

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；您可以获得类似如下的输出：

图 9.10. 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>
```

9.2. USB 设备

这部分提供了处理 USB 设备所需的命令。

9.2.1. 为客户机虚拟机分配 USB 设备

Web 相机、卡读器、键盘或 mice 等大多数设备都使用 USB 端口和电缆连接到计算机。可以通过两种方式将这样的设备传递给客户端虚拟机：

- 使用 USB 透传 - 这要求设备物理连接到托管客户机虚拟机的主机物理机器。在这种情况下不需要 SPICE。主机上的 USB 设备可以使用命令行或 virt-manager 传递给客户端。有关 virt 管理器 指示，请参阅 第 15.3.1 节 “将 USB 设备附加到虚拟机”。

注意

virt-manager 不应该用于热插拔或热拔设备。如果要热插拔/热拔 USB 设备，请参阅 过程 14.1, “热插 USB 设备供客户端虚拟机使用”。

- 使用 USB 重定向 - USB 在主机物理机器中运行时最好使用。用户从本地机器或瘦客户端连接到其/继承的客户机虚拟机。在该本地计算机上，有 SPICE 客户端。用户可以将任何 USB 设备附加到瘦客户机，而 SPICE 客户端会将设备重定向到数据中心的主机物理机器，以便其可供在瘦客户端上运行的客户机虚拟机使用。有关使用 virt-manager 进行 USB 重定向的说明，请参考 第 15.3.1 节 “将 USB 设备附加到虚拟机”，应该注意，使用 TCP 协议（推荐 BZ#1085318）无法进行 USB 重定向。

9.2.2. 在 USB 设备重定向上设置限制

要从重定向过滤某些设备，请将过滤器属性传递给 **-device usb-redir。filter 属性采用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。您可以将 **|** 用作分隔符，在同一命令行中使用多个规则。

重要

如果设备与任何规则过滤器都不匹配，则不会重定向它！

例 9.1. 使用 windows 客户机虚拟机限制重定向的示例

1. 准备 **Windows 7** 客户机虚拟机。

2. 将以下代码摘录添加到 **guest** 虚拟机的 **domain xml** 文件中：

```
<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. 启动客户端虚拟机并确认设置更改：

```
#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** 设备插入主机物理机器，并使用 **virt-manager** 连接到客户端虚拟机。

5. 在菜单中，单击 **Redirect USB Service**，这将生成以下消息：**"Some USB** 设备受主机

策略阻止"。单击确定以确认 并继续。

过滤器生效。

6. 为确保过滤器正确捕获了 USB 设备供应商和产品，然后在 guest 虚拟机的域 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 guest 虚拟机。USB 设备现在会将流量重定向到客户端虚拟机。

9.3. 配置设备控制器

根据客户机虚拟机架构，一些设备总线可能出现多次，并且有一组虚拟设备连接到虚拟控制器。通常，libvirt 可以自动推断此类控制器而无需明确的 XML 标记，但在某些情况下，最好显式设置虚拟控制器元素。

图 9.11. 自动化控制器的域 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*

<控制器> 元素具有强制属性 <控制器索引>，它是十进制整数，描述总线控制器被遇到的顺序（用于控制器 <地址> 元素）。当 <控制器类型 = 'virtio-serial'> 时，还有两个可选属性（名为 端口和 向量），它控制可以通过控制器连接多少个设备。请注意，Red Hat Enterprise Linux 6 不支持使用每个设备超过 32向量。使用更多的向量将导致迁移客户机虚拟机失败。

当 <控制器类型 = 'scsi'> 时，有一个可选的属性 模型，它可具有以下值：

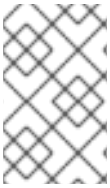
- *auto*
- *buslogic*
- *ibmvscsi*
- *lsilogic*

- *lsisas1068*
- *lsisas1078*
- *virtio-scsi*
- *vmpvscsi*

当 <控制器类型 = 'usb'> 时，有一个可选的属性 模型，它可具有以下值：

- *piix3-uhci*
- *piix4-uhci*
- *ehci*
- *ich9-ehci1*
- *ich9-uhci1*
- *ich9-uhci2*
- *ich9-uhci3*
- *vt82c686b-uhci*
- *pci-ohci*

- *nec-xhci*



注意

如果需要为 guest 虚拟机明确禁用 USB 总线，可以使用 <model='none'>。 .

对于在 PCI 或 USB 总线上本身设备的控制器，可选的子元素 <地址> 可以指定控制器与其主总线的确切关系，以及语义，如 第 9.4 节 “为设备设置地址” 所示。

可选的 sub-element <驱动程序> 可以指定驱动程序特定选项。目前它只支持属性队列，这指定了控制器的队列数量。为获得最佳性能，建议指定一个与 vCPU 数量匹配的值。

USB companion 控制器具有一个可选的子元素 <master>，用于指定与主控制器相配套的关系。companion 控制器与其 master 位于同一个总线上，因此相应的 索引 值应该相等。

可以使用的 XML 示例如下：

图 9.12. 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 控制器具有可选的 模型 属性，具有以下可能的值：

- *pci-root*

- *pcie-root*
- *pci-bridge*
- *dmi-to-pci-bridge*

root 控制器（pci-root 和 pcie-root）具有一个可选的 pcihole64 元素，用于指定 pcihole64 单元 属性指定的单位是 pcihole。有些客户机虚拟机（如 Windows Server 2003）可能会导致崩溃，除非禁用单位 （设置为 0 单元='0'）。

对于提供隐式 PCI 总线的机器类型，pci-root 控制器自动添加 index='0'，且需要使用 PCI 设备。pci-root 没有地址。如果已在由 model='pci-root' 提供的一个总线上适合一个由 model='pci-root' 或一个大于零的 PCI 总线号，则会自动添加 PCI 网桥。PCI 网桥也可以手动指定，但其地址应仅引用已经指定 PCI 控制器提供的 PCI 总线。PCI 控制器索引中的间隔可能会导致无效的配置。以下 XML 示例可添加到 <devices> 部分：

图 9.13. 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(PCIe)总线（例如，基于 Q35 芯片组的机器类型），带有 index='0' 的 pcie-root 控制器会自动添加到域配置中。pcie-root 也没有地址，但提供 31 个插槽（数字 1-31），且只能用于附加 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 控制器插入到 PCIe 插槽（由 pcie-root 提供），本身提供 31 个标准 PCI 插槽（非热插拔）。要在客户机系统中拥有热插拔 PCI 插槽，所有由 libvirt 自动终止的 pci-bridge 控制器也会自动创建并连接到自动创建的 dmi-to-pci-bridge 控制器的插槽之一。所有具有 PCI 地址且由 libvirt 自动终止的客户机设备都将放置在此 pci-bridge 设备中。

图 9.14. 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>
...
```

9.4. 为设备设置地址

许多设备具有一个可选的 <地址> 子元素，用于描述设备放置在提供给虚拟客户机的虚拟总线上。如果输入时省略了地址（或地址中的任何可选属性）时，libvirt 将生成一个适当的地址；不过，如果需要更多地控制布局，则需要一个明确的地址。如需包括 <地址> 元素在内的域 XML 设备示例，请参阅 图 9.6 “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。在设备和客户机虚拟机架构中限制在给定设备使用的地址的选择。例如：<磁盘设备> 使用 type='drive'，而 <控制台> 设备在 i686 或 x86_64 客户机虚拟机构架中使用 type='pci'。每个地址类型都具有更多可选属性，可控制该设备在总线上的位置，如表中所述：

表 9.1. 支持的设备地址类型

地址类型	描述
type='pci'	PCI 地址具有以下额外属性： ● 域（2 字节十六进制整数，当前不供 qemu 使用） ● 总线（0 到 0 到 0xff 之间的十六进制值，含） ● 插槽（0x0 和 0x1f 之间的十六进制值，含） ● 功能（0 到 7 之间的值） ● 默认情况下，多功能控制为 PCI 控制寄存器中的特定插槽/功能开启了多功能位。默认情况下，它设置为"off"，但应该设置为"on"用于插槽的功能 0，它将具有多个功能。

地址类型	描述
type='drive'	驱动器地址具有以下额外属性： ● 控制器（2 位控制器号） ● 总线（2 位总线号） ● 目标（2 位总线号） ● 单元（总线上的 2 位单元数）
type='virtio-serial'	每个 virtio-serial 地址都有以下附加属性： ● 控制器（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'	用于智能卡的 CCID 地址具有以下附加属性： ● 总线（2 位总线号） ● 插槽属性（总线中的 2 位插槽）
type='usb'	USB 地址有以下附加属性： ● 总线（0 到 0 到 0xfff 之间的十六进制值，含） ● 端口（最多 4 个八位字节，如 1.2 或 2.1.3.1）
type='isa'	ISA 地址有以下附加属性： ● iobase ● irq

9.5. 在虚拟机中管理存储控制器

从 Red Hat Enterprise Linux 6.4 开始，支持将 SCSI 和 virtio-SCSI 设备添加到运行 Red Hat Enterprise Linux 6.4 或更高版本的客户机虚拟机中。与 virtio 磁盘不同，SCSI 设备需要在客户机虚拟机中存在控制器。VirtIO-SCSI 提供了直接与 SCSI LUN 连接的功能，与 virtio-blk 相比显著提高可扩展性。virtio-SCSI 的优点是，与 virtio-blk 相比，可以处理数百个设备，它们只能处理 28 个设备并耗尽 PCI 插槽。现在，virtio-SCSI 能够继承目标设备的功能集，并可以：

- ***通过 virtio-scsi 控制器附加虚拟硬盘驱动器或 CD，***

- 通过 QEMU scsi-block 设备从主机传递物理 SCSI 设备，

- 和 允许为每个客户机使用数百个设备；从 28 设备限制 virtio-blk 中有所改进。

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）以及将 SCSI 存储添加到客户端虚拟机所需的步骤。

过程 9.10. 创建虚拟 SCSI 控制器

1. 显示客户机虚拟机的配置(Guest1)，并查找预先存在的 SCSI 控制器：

```
# virsh dumpxml Guest1 | grep controller.*scsi
```

如果存在设备控制器，命令会输出类似如下的一个或多个行：

```
<controller type='scsi' model='virtio-scsi' index='0'/>
```

2. 如果上一步没有显示设备控制器，使用以下步骤为某个新文件创建一个描述并将其添加到虚拟机中：

   a. 通过在新文件中写入 < controller& gt; 元素，并使用 XML 扩展保存文件来创建设备控制器。virtio-scsi-controller.xml，例如：

   ```
   <controller type='scsi' model='virtio-scsi'/>
   ```

   b. 将您刚刚在 virtio-scsi-controller.xml 中创建的设备控制器与您的客户机虚拟机（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 过程 13.2, “在客户机中添加物理块设备”。

3. 添加新的 SCSI 磁盘或 CD-ROM。可使用部分 第 13.3.1 节 “在 客户机中添加基于文件的存储” 和 第 13.3.2 节 “在客户机中添加硬盘和其他块设备” 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。

```
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img sdb --cache none
```

根据客户机虚拟机中驱动程序的版本，正在运行的 guest 虚拟机可能不会立即检测到新磁盘。按照 Red Hat Enterprise Linux Storage Administration Guide 中的步骤进行操作。

9.6. 随机数字生成器(RNG)设备

virtio-rng 是一个虚拟 RNG（随机数生成器）设备，该设备向客户机虚拟机的操作系统提供 RNG 数据，从而在请求时为客户机虚拟机提供全新的熵。

当使用 RNG 的设备（如键盘）时，鼠标和其他输入不足以在客户机虚拟机上生成足够的熵。virtio-rng 设备可用于 Red Hat Enterprise Linux 和 Windows 客户机虚拟机。有关安装 Windows 要求的步骤，请参阅 注意。除非另有说明，否则以下描述适用于 Red Hat Enterprise Linux 和 Windows 客户机虚拟机。

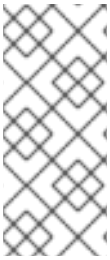
当在 Linux 客户机虚拟机上启用了 virtio-rng 时，会在客户端虚拟机中创建 chardev，其位置为 /dev/hwrng/。然后可以打开这个 chardev，并读取从主机物理机器获取熵。为了使客户机应用程序能够透明地使用 virtio-rng 设备的随机性，/dev/hwrng/ 中的输入必须转发到客户机虚拟机中的内核熵池。如果此位置中的信息与 rgnd 守护进程合并（位于 rng-tools 中），则可以实现这一点。

这耦合会导致将熵路由到 guest 虚拟机的 /dev/random 文件。这个过程是在 Red Hat Enterprise Linux 6 客户机虚拟机中手动完成的。

Red Hat Enterprise Linux 6 客户机虚拟机与运行以下命令相结合：

```
# rngd -b -r /dev/hwrng/ -o /dev/random/
```

如需更多帮助，请运行 man rngd 命令以获得此处所示的命令选项的说明。有关更多示例，请参考 过程 9.11, “使用命令行工具实施 virtio-rng” 来配置 virtio-rng 设备。



注意

Windows 客户机虚拟机需要安装驱动程序 viorng。安装后，虚拟 RNG 设备将使用 Microsoft 提供的 CNG（下一代）API 工作。安装完驱动程序后，virtrng 设备就会出现在 RNG 供应商列表中。

过程 9.11. 使用命令行工具实施 virtio-rng

1. 关闭客户机虚拟机。

2. 在终端窗口中，使用 virsh edit domain-name 命令打开所需 guest 虚拟机的 XML 文件。

3. 编辑 <devices> 元素使其包含以下内容：

```
...
<devices>
  <rng model='virtio'>
    <rate period="2000" bytes="1234"/>
    <backend model='random'>/dev/random</backend>
        <source mode='bind' service='1234'>
        <source mode='connect' host='192.0.2.1' service='1234'>
    </backend>
  </rng>
</devices>
...
```

第 10 章 QEMU-IMG 和 QEMU 客户机代理

本章包含将 qemu-img 软件包与客户机虚拟机搭配使用的有用提示和提示。如果您要查找有关 QEMU 跟踪事件和参数的信息，请参考此处的 README 文件： /usr/share/doc/qemu-*/README.systemtap。

10.1. 使用 QEMU-IMG

qemu-img 命令行工具用于格式化、修改和验证 KVM 使用的各种文件系统。下方列出了 QEMU-img 选项和用法。

检查

对磁盘映像 文件名 执行一致性检查。

```
#  qemu-img check -f qcow2 --output=qcow2 -r all filename-img.qcow2
```

注意

只有 qcow2 和 vdi 格式支持一致性检查。

使用 -r 尝试修复在检查过程中发现的所有不一致问题，但在修复 -r leaks 集群泄漏时，会修复并处理 -r all 各种错误被修复。请注意，这存在选择错误的修复或隐藏可能已经发生崩溃问题的风险。

Commit（提交）

使用 qemu-img commit 命令将指定文件（文件名）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（格式）。

```
# qemu-img commit [-f format] [-t cache] filename
```

convert

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。

命令格式：

```
# qemu-img convert [-c] [-p] [-f format] [-t cache] [-O output_format] [-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** 选项允许创建 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都类似标准文件，但物理块仅包含零（无）。当操作系统看到此文件时，它会将其视为存在，并占用实际磁盘空间，即使实际情况也不会采取任何情况。这在为客户机虚拟机创建磁盘时特别有用，因为这会造成磁盘所占用的磁盘空间比它多得多。例如，如果您在磁盘镜像上的 **-S** 设置为 **50Gb**，则您的 **10Gb** 磁盘空间的 **10Gb** 的大小将显示为 **60Gb**，即使实际使用了 **10Gb**。

使用格式 **filename** 将磁盘镜像 **output_filename** 转换为磁盘镜像 **output_format**。磁盘镜像可以选择使用 **-c** 选项压缩，或者通过设置 **-o** 来使用 **-o encryption** 选项加密。请注意，**-o** 参数可用的选项与所选格式不同。

只有 **qcow2** 格式支持加密或压缩。**qcow2** 加密使用 **AES** 格式，以及安全 **128** 位密钥。**qcow2** 压缩为只读模式，因此如果压缩的扇区从 **qcow2** 格式转换，它将被写成为未压缩数据的新格式。

在使用可增加的格式时（如 **qcow** 或 **cow** ）时，镜像转换也很有用。检测到空扇区并从目标镜像中禁止。

创建

创建新的磁盘大小 **size** 并格式化 **format**。

```
# qemu-img create [-f format] [-o options] filename [size][preallocation]
```

如果使用 **-o backing_file=filename** 指定基础镜像，则镜像只记录自身和基础镜像之间的区别。除非您使用 **commit** 命令，否则不会修改后备文件。在这种情况下不需要指定大小。

预分配是一个只能用于创建 **qcow2** 镜像的选项。接受的值包括 **-o preallocation=off|meta|full|falloc**。预分配元数据的镜像大于镜像，无需.但是，在镜像大小增加的情况下，随着镜像的增长，性能会提高。

请注意，使用 完整 分配可能需要较长时间的大型镜像。如果您想要实现完全分配且时间为准，使用 **falloc** 将为您节省时间。

info

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，其大小可以与显示的大小不同。如果快照存储在磁盘镜像中，也会显示它们。例如，该命令显示块设备上 qcow2 镜像采用的空间量。这可以通过运行 qemu-img 来完成。您可以使用 qemu-img check 命令来检查镜像是否为与 qemu-img info 命令的输出相符。请参阅 第 10.1 节 “使用 qemu-img”。

```
# 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
```

map

 # qemu-img map [-f format] [--output=output_format] filename 命令转储镜像文件名及其后备文件链的元数据。具体来说，此命令会转储指定文件的每个扇区的分配状态，以及在后备文件链中分配它的最顶层文件。例如，如果您有一个链，如 **c.qcow2 → b.qcow2 → a.qcow2**，则 **a.qcow** 是原始文件，**b.qcow2** 是对 **a.qcow2** 和 **c.qcow2** 的更改是来自 **b.qcow2** 的 **delta** 文件。创建此链时，镜像文件会存储普通镜像数据，以及关于什么文件中以及它们在文件所在的位置的信息。此信息称为镜像的元数据。**f format** 选项是指定镜像文件的格式。可以使用 **raw**、**qcow2**、**vhdx** 和 **vmdk** 等格式。可能有两个输出选项：**human** 和 **json**。

- **人** 为默认设置。它旨在更加易读的人类阅读，因此这种格式不应被解析。为了清晰和简单性，默认 **人** 格式仅转储文件的已知非零区域。文件的已知零部分将被完全省略，也同样适用于不在整个链中分配的部分。当执行命令时，**qemu-img** 输出将识别从其中读取数据的文件，以及 文件中的偏移。输出显示为包含四个列的表：前三个是十六进制数字。

```
# qemu-img map -f qcow2 --output=human /tmp/test.qcow2
Offset          Length          Mapped to       File
0               0x20000         0x50000         /tmp/test.qcow2
0x100000        0x80000         0x70000         /tmp/test.qcow2
0x200000        0x1f0000        0xf0000         /tmp/test.qcow2
0x3c00000       0x20000         0x2e0000        /tmp/test.qcow2
0x3fd0000       0x10000         0x300000        /tmp/test.qcow2
```

- **JSON 或 JSON（JavaScript** 对象表示法）可供人类读取，但由于编程语言，它也旨在解析。例如，要在解析器中解析 **"qemu-img map"** 的输出，则应使用 **--output=json** 选项。

```
# qemu-img map -f qcow2 --output=json /tmp/test.qcow2
[{ "start": 0, "length": 131072, "depth": 0, "zero": false, "data": true, "offset": 327680},
{ "start": 131072, "length": 917504, "depth": 0, "zero": true, "data": false},
```

如需有关 JSON 格式的更多信息，请参阅 qemu-img(1)手册页。

rebase

更改镜像的后端文件。

```
# qemu-img rebase [-f format] [-t cache] [-p] [-u] -b backing_file [-F backing_format] filename
```

后备文件更改为 **backing_file**，并（如果 文件名 格式支持该功能），则后备文件格式更改为 **backing_format**。

注意

只有 qcow2 格式支持更改后备文件(rebase)。

稳定可以操作的两种不同的模式： **Safe** 和 **Unsafe**。

默认使用 安全模式，并执行实际的 **rebase** 操作。新的后备文件可能与旧文件不同，**qemu-img rebase** 命令就会小心，确保 **guest** 虚拟机不可见 的文件名 内容保持不变。为实现此目的，在对 后备文件 和文件名的旧备份文件不同的集群都会合并到 文件名 中，然后对后备文件进行任何更改。

请注意，安全模式是一个昂贵操作，相当于转换镜像。需要旧后备文件才能成功完成。

如果将 **-u** 选项传递给 **qemu-img rebase**，则会使用不安全模式。在这个模式中，只更改 文件名 的后备文件名和格式，而不对文件内容进行任何检查。确保正确指定新后备文件，否则镜像可见的内容会损坏。

这个模式对重命名或移动后备文件很有用。它可以在不访问旧后备文件的情况下使用它。例如，它可以用于修复已移动或重命名后备文件的镜像。

调整大小

更改磁盘镜像 文件名，就像它创建 大小为。无论版本如何，只有采用原始格式的镜像才可以重新定义大小。**Red Hat Enterprise Linux 6.1** 及之后的版本添加了以 **qcow2** 格式增加（但不缩小）镜像的功能。

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
```

您还可以相对于磁盘镜像的当前大小来调整。要指定大小相对于当前大小，请为加上 + 的字节数加前缀，或 - 通过该字节数来减小磁盘镜像的大小。添加单元后缀允许您以千字节(K)、兆字节(M)、千兆字节(G)或 TB(T)为单位设置镜像大小。

```
# qemu-img resize filename [+|-]size[K|M|G|T]
```

警告

在使用此命令缩小磁盘镜像之前，您必须使用 虚拟机本身中的文件系统和分区工具来减少分配的文件系统和分区大小。否则会导致数据丢失。

在使用此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实际开始使用该设备中的新空间。

Snapshot

列出、应用、创建或删除映像的现有快照（快照）（文件名）。

```
# qemu-img snapshot [ -l | -a snapshot | -c snapshot | -d snapshot ] filename
```

- 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 可以创建映像的快照（快照）。-d 删除指定的快照。

支持的格式

QEMU-img 旨在将文件转换为以下格式之一：

raw

原始磁盘镜像格式（默认）。这可以是基于文件的最快格式。如果您的文件系统支持 holes（例如，在 Linux 上的 ext2 或 ext3 中，Windows 上的 NTFS），则只有写扇区将被保留空间。使用

qemu-img info 获取镜像或 ls -ls on Unix/Linux 上使用的实际大小。虽然 Raw 镜像提供了最佳性能，但只有 Raw 镜像具有非常基本的功能（例如，没有快照可用）。

qcow2

QEMU 镜像格式，包含最佳功能集的最通用格式。使用它来具有可选的 AES 加密，基于 zlib 的压缩，支持多个虚拟机快照和较小的镜像，这对于不支持 holes 的文件系统（ Windows 中的非NTFS 文件系统）非常有用。请注意，这种广泛的功能集以性能为代价。

虽然只有上述格式可用于在客户机虚拟机或主机物理计算机中运行，qemu-img 还识别并支持以下格式，以便将其转换为 raw 或 qcow2 格式。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，也可将其从 raw 或 qcow2 转换为原始格式。

Bochs

Bochs 磁盘镜像格式。

cloop

Linux 压缩的 Loop 镜像，仅适用于重复使用压缩的 CD-ROM 镜像，例如 Knoppix CD-ROM。

COW

user Mode Linux Copy On Write image format。cow 格式仅用于与之前版本兼容。它无法使用 Windows。

dmg

MAC 磁盘镜像格式。

nbd

网络块设备.

Parallels

并行虚拟化磁盘映像格式.

QCOW

旧的 QEMU 镜像格式。只适用于与旧版本兼容。

vdi

Oracle VM VirtualBox 硬盘映像格式.

vmdk

VMware 兼容镜像格式（对版本 1 和 2 的读写支持），对版本 3 具有只读访问权限。

vpc

Windows 虚拟 PC 磁盘映像格式.也称为 vhd 或 Microsoft 虚拟硬盘映像格式。

vvfat

虚拟 VFAT 磁盘镜像格式。

10.2. QEMU 客户机代理

QEMU 客户机代理在客户机中运行，并允许主机计算机使用 libvirt 向客户机操作系统发出命令。然后，客户机操作系统异步响应这些命令。本章论述了用于客户机代理的 libvirt 命令和选项。

重要

请注意，仅当由受信任的客户机运行时，它才能够安全地依赖客户机。不受信任客户机可能会恶意忽略或滥用客户机代理协议，尽管存在内置保护机制以防止拒绝服务攻击，但主机需要客户机协作才能按预期运行。

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)，以此调整 vCPU 数量，而无需使用热拔功能。如需更多信息，请参阅 第 14.13.6 节 “配置虚拟 CPU 数”。

10.2.1. 安装并启用客户机代理

使用 yum install qemu-guest-agent 命令，在 guest 虚拟机上安装 qemu-guest-agent，并使其在每次引导时都作为服务(qemu-guest-agent.service)自动运行。

10.2.2. 设置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计算机之间的 VirtIO 串行连接与客户机代理通信。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，并且客户机侦听此串行通道。以下流程演示了如何为客户机代理设置主机和虚拟机机器。

注意

有关如何在 Windows 客户端上设置 QEMU 客户机代理的说明，请参考 中的说明。http://msdn.microsoft.com/en-us/library/windows/desktop/bb968832%28v=vs.85%29.aspx

过程 10.1. 设置客户机代理和主机之间的通信

1. **打开客户机 XML**

   使用 QEMU 客户机代理配置打开客户机 XML。您将需要 guest 名称来打开文件。使用主机机器上的 # virsh list 命令列出它可以识别的客户机。在本例中，guest 的名称是 rhel6 ：

```
# virsh edit rhel6
```

2. **编辑客户机 XML 文件**

   将下列元素添加到 XML 文件并保存更改。

图 10.1. 编辑客户机 XML 以配置 QEMU 客户机代理

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/rhel6.agent'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

3. **在客户端中启动 QEMU 客户机代理**

   如果还没有这样做，请使用 yum install qemu-guest-agent 在客户机虚拟机中下载并安装客户机代理。安装后，按如下所示启动该服务：

```
# service start qemu-guest-agent
```

现在，您可以通过在已建立的字符设备驱动程序发送有效的 libvirt 命令与客户机通信。

10.2.3. 使用 QEMU 客户机代理

Red Hat Enterprise Linux 6.5 及更新版本完全支持 QEMU 客户机代理协议(QEMU GA)软件包 qemu-guest-agent。但是，对 isa-serial/virtio-serial 传输有以下限制：

- qemu-guest-agent 无法检测到客户端是否连接到频道。
- 客户端无法检测到 qemu-guest-agent 是否断开连接或重新连接后端。
- 如果 virtio-serial 设备重置和 qemu-guest-agent 没有连接到频道（由重启或热插件导致），则客户端中的数据将被丢弃。
- 如果 qemu-guest-agent 在通过 virtio-serial 设备重置后连接到频道，则客户端中的数据将被排队（如果可用缓冲区是否已耗尽），不管是否仍在运行或连接了 qemu-guest-agent。

10.2.4. 将 QEMU 客户机代理与 libvirt 搭配使用

安装 QEMU 客户机代理允许各种其他 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 - 此关闭方法比 virsh shutdown --mode=acpi 更为可靠，因为与 QEMU 客户机代理一起使用的 virsh shutdown 保证可保证以干净状态关闭合作客户机。如果没有代理，libvirt 需要依赖注入 ACPI 关闭事件，但有些客户机会忽略该事件，因此不会关闭。

  可用于 virsh reboot 的相同语法。

- virsh snapshot-create --quiesce - 允许 guest 在创建快照之前将其 I/O 刷新到稳定状态，这样就允许在不执行 fsck 或丢失部分数据库交易的情况下使用快照。客户机代理通过提供客户机协作，实现高水平的磁盘内容稳定性。
- virsh setvcpus --guest - 对客户机进行离线设置，使 CPU 离线。
- virsh dompmsuspend - 使用客户机操作系统的电源管理功能安全暂停正在运行的 guest。

10.2.5. 创建客户机虚拟机磁盘备份

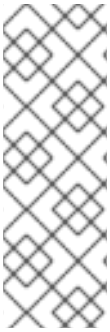
libvirt 可以与 qemu-ga 通信，以确保 guest 虚拟机文件系统的快照在内部一致，并可根据需要使用。Red Hat Enterprise Linux 6 的改进的目的是确保文件和应用程序级别同步（同步）都已完成。客户机操作系统管理员可以编写和安装特定于应用程序的 freeze/thaw hook 脚本。在释放文件系统之前，qemu-ga 调用主 hook 脚本（在 qemu-ga 软件包中包括）。freezing 进程会临时取消激活所有 guest 虚拟机应用程序。

只有在文件系统被冻结前，才发生以下操作：

- *文件系统应用程序 / 数据库冲刷到虚拟磁盘的工作缓冲区，并停止接受客户端连接*
- *应用程序将其数据文件置于一致状态*
- *主 hook 脚本返回*
- *qemu- gazes 文件系统和管理堆栈采用快照*
- *已确认快照*
- *文件系统功能恢复*

Thawing 会以相反的顺序进行。

使用 snapshot-create-as 命令创建客户机磁盘的快照。有关这个命令的详情请参考 第 14.15.2.2 节 “为当前域创建快照”。



注意

特定于应用程序的 hook 脚本可能需要各种 SELinux 权限才能正确运行，因为当脚本需要连接到套接字时，就可以与数据库进行通信。通常，出于此类目的，应开发并安装本地 SELinux 策略。访问文件系统节点后，在标记为 /etc/qemu-ga/fsfreeze-hook.d/ 的表行中发出 restorecon -FvvR 命令后，应立即工作。表 10.1 “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 10.1. QEMU 客户机代理软件包内容

文件名	描述
/etc/rc.d/init.d/qemu-ga	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 **/etc/rc.d/init.d/qemu-ga** 控制脚本读取。设置记录在 文件中，并包含 shell 脚本注释。
/usr/bin/qemu-ga	QEMU 客户机代理二进制文件。
/usr/libexec/qemu-ga/	hook 脚本的根目录。
/usr/libexec/qemu-ga/fsfreeze-hook	主 hook 脚本.这里不需要修改。
/usr/libexec/qemu-ga/fsfreeze-hook.d/	单独、特定于应用程序的 hook 脚本的目录。客户机系统管理员应将 hook 脚本手动复制到此目录，确保它们的正确文件模式位，然后在此目录上运行 **restorecon -FvvR**。
/usr/share/qemu-kvm/qemu-ga/	带有示例脚本的目录（例如，仅用于）。此处包含的脚本未执行。

主 hook 脚本 /usr/libexec/qemu-ga/fsfreeze-hook 会记录其自身的消息，以及应用程序特定脚本的标准输出和错误消息，在以下日志文件中：/var/log/qemu-ga/fsfreeze-hook.log。有关更多信息，请参阅 wiki.qemu.org 或 libvirt.org 的 qemu-guest-agent wiki 页面。

10.3. 在 WINDOWS 虚拟客户机中运行 QEMU 客户机代理

Red Hat Enterprise Linux 主机机器可以通过在客户机中运行 QEMU 客户机代理向 Windows 客户机发出命令。它支持运行 Red Hat Enterprise Linux 6.5 及更新版本的主机，并在以下 Windows 客户机操作系统中受支持：

- *Windows XP Service Pack 3（不支持VSS）*
- *Windows Server 2003 R2 - x86 和 AMD64（不支持VSS）*

- *Windows Server 2008*
- *Windows Server 2008 R2*
- *Windows 7 - x86 和 AMD64*
- *Windows Server 2012*
- *Windows Server 2012 R2*
- *Windows 8 - x86 和 AMD64*
- *Windows 8.1 - x86 和 AMD64*

注意

Windows 客户机虚拟机需要 QEMU 客户机代理软件包用于 Windows，qemu-guest-agent-win。对于在 Red Hat Enterprise Linux 上运行的 Windows 客户机虚拟机，VSS(Volume Shadow Copy Service)需要此代理。有关更多信息，请参见 http://msdn.microsoft.com/en-us/library/windows/desktop/bb968832%28v=vs.85%29.aspx。

过程 10.2. 在 Windows 客户端中配置 QEMU 客户机代理

针对在 Red Hat Enterprise Linux 主机机器中运行的 Windows 客户机，请按照以下步骤操作。

1. *准备 Red Hat Enterprise Linux 主机机器*

   确保在 Red Hat Enterprise Linux 主机物理机器上安装了以下软件包：

   - *virtio-win，位于 /usr/share/virtio-win/*

   *要在 Windows 客户端中复制驱动程序，请使用以下命令为 qxl 驱动程序生成 *.iso 文件：*

```
# mkisofs -o /var/lib/libvirt/images/virtiowin.iso /usr/share/virtio-win/drivers
```

2. **准备 Windows 客户机**

   通过将 *.iso 挂载到 Windows guest 以更新驱动程序，在客户机中安装 virtio-serial driver。启动 guest，然后将驱动程序 .iso 文件连接到 guest（使用名为 hdb的磁盘）：

```
# virsh attach-disk guest /var/lib/libvirt/images/virtiowin.iso hdb
```

   要使用 Windows Control Panel 来安装驱动程序，请导航到以下菜单：

   - **要安装 virtio-win 驱动程序 - Select Hardware and Sound > Device manager > virtio-serial driver。**

3. **更新 Windows 客户机 XML 配置文件**

   Windows 客户机的客户机 XML 文件位于 Red Hat Enterprise Linux 主机中。要获取这个文件的访问权限，您需要 Windows 虚拟客户机名称。在主机机器上使用 # virsh list 命令，列出它可识别的客户机。在本例中，guest 的名称是 win7x86。

   使用 # virsh edit win7x86 命令在 XML 文件中添加以下元素并保存更改。请注意，源套接字名称在主机中必须是唯一的，本例中为 win7x86.agent ：

图 10.2. 编辑 Windows 客户机 XML 以配置 QEMU 客户机代理

```
...
<channel type='unix'>
   <source mode='bind' path='/var/lib/libvirt/qemu/win7x86.agent'/>
   <target type='virtio' name='org.qemu.guest_agent.0'/>
   <address type='virtio-serial' controller='0' bus='0' port='1'/>
</channel>
<channel type='spicevmc'>
   <target type='virtio' name='com.redhat.spice.0'/>
   <address type='virtio-serial' controller='0' bus='0' port='2'/>
</channel>
...
```

4. **重启 Windows 客户机**

   重启 Windows 客户机以应用更改：

```
# virsh reboot win7x86
```

5. *在 Windows 客户端中准备 QEMU 客户机代理*

   在 Windows 客户端中准备客户机代理：

   a. *安装最新的 virtio-win 软件包*

      在 Red Hat Enterprise Linux 主机物理机器终端窗口中运行以下命令，以查找要安装的文件。请注意， 下面显示的 文件可能与系统找到的文件完全相同，但应为最新的官方版本。

```
# rpm -qa|grep virtio-win
virtio-win-1.6.8-5.el6.noarch

# rpm -iv virtio-win-1.6.8-5.el6.noarch
```

   b. *确认安装已完成*

      在 virtio-win 软件包完成安装后，检查 /usr/share/virtio-win/guest-agent/ 文件夹，并找到名为 qemu-ga-x64.msi 的文件或 qemu-ga-x86.msi，如下所示：

```
# ls -l /usr/share/virtio-win/guest-agent/

total 1544

-rw-r--r--. 1 root root 856064 Oct 23 04:58 qemu-ga-x64.msi

-rw-r--r--. 1 root root 724992 Oct 23 04:58 qemu-ga-x86.msi
```

   c. *安装 .msi 文件*

      从 Windows 客户机（例如，win7x86）通过双击 文件来安装 qemu-ga-x64.msi 或 qemu-ga-x86.msi。安装后，它将在 System Manager 中的 Windows guest 中显示为 qemu-ga 服务。此管理器可用于监控服务的状态。

10.3.1. 在 Windows Guests 上使用带有 QEMU 客户机代理的 libvirt 命令

QEMU 客户机代理可在 Windows 客户机中使用以下 virsh 命令：

- *virsh shutdown --mode=agent - 此关闭方法比 virsh shutdown --mode=acpi 更为可靠，因为与 QEMU 客户机代理一起使用的 virsh shutdown 保证可保证以干净状态关闭合作客户机。如果没有代理，libvirt 需要依赖注入 ACPI 关闭事件，但有些客户机会忽略该事件，因此不会关闭。*

  可用于 virsh reboot 的相同语法。

- **virsh snapshot-create --quiesce - 允许 guest 在创建快照之前将其 I/O 刷新到稳定状态，这样就允许在不执行 fsck 或丢失部分数据库交易的情况下使用快照。客户机代理通过提供客户机协作，实现高水平的磁盘内容稳定性。**

- **virsh dompmsuspend - 使用客户机操作系统的电源管理功能安全暂停正在运行的 guest。**

10.4. 在设备重定向上设置限制

要从重定向过滤某些设备，请将过滤器属性传递给 -device usb-redir。filter 属性采用由过滤规则组成的字符串。规则的格式为：

```
<class>:<vendor>:<product>:<version>:<allow>
```

使用 -1 值指定它接受特定字段的任何值。您可以将 | 用作分隔符，在同一命令行中使用多个规则。请注意，如果设备没有匹配任何过滤器规则，则不会允许重定向。

例 10.1. 使用 Windows 客户机虚拟机限制重定向

1. **准备 Windows 7 客户机虚拟机。**

2. **在 guest 虚拟机的 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. **启动客户端虚拟机并确认设置更改：**

```
# 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 设备插入主机物理计算机中，并使用 virt-viewer 连接到客户机虚拟机。

5. 点击菜单中的 USB 设备选择，这将生成以下信息："Some USB 设备受主机策略阻止"。单击确定以确认 并继续。

过滤器生效。

6. 为确保过滤器正确捕获了 USB 设备供应商和产品，然后在主机物理机器的域 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 guest 虚拟机。USB 设备现在会将流量重定向到客户端虚拟机。

10.5. 动态更改附加到虚拟 NIC 的主机物理机器或网桥

本节演示如何将客户机虚拟机的 vNIC 从一个网桥移到另一个网桥，而 guest 虚拟机在不影响客户机虚拟机的情况下运行

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 是否连接到您指示的网桥。

```
# 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 guest 虚拟机网络：

```
# virsh update-device test1 br1.xml

Device updated successfully
```

5. 在 guest 虚拟机上，运行 service network restart。客户机虚拟机获得 virbr1 的新 IP 地址。检查 guest 虚拟机的 vnet0 是否连接到新网桥(virbr1)

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.5254007da9f2       yes             virbr0-nic
virbr1          8000.525400682996       yes             virbr1-nic      vnet0
```

第 11 章 存储概念

本章介绍了用于描述和管理存储设备的概念。存储池和卷等术语在后续小节中阐述。

11.1. 存储池

存储池 是由 libvirt 管理的文件、目录或存储设备，用于向客户机虚拟机提供存储。存储池可以是本地的，也可以通过网络共享。存储池是管理员设置的存储数量（通常是专用存储管理员）供客户机虚拟机使用。存储池由存储管理员或系统管理员划分到存储卷，卷则作为块设备分配到客户机虚拟机。在简短的存储卷中，需要对什么是存储池进行分区。虽然存储池是一个虚拟容器，但有两个因素限制： qemu-kvm 允许的最大值，以及主机物理机器上的磁盘大小。存储池不能超过主机物理机器上的磁盘大小。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 块大小）
- XFS = ~8 Exabytes
- 在尝试非常大的镜像大小时，qcow2 和主机文件系统会保留自己的元数据和可扩展性。使用原始磁盘意味着可能会影响可扩展性或最大大小的层数。

libvirt 使用基于目录的存储池 /var/lib/libvirt/images/ 目录作为默认存储池。可以将默认存储池改为另一个存储池。

- 本地存储池 - 本地存储池直接附加到主机物理机器服务器。本地存储池包括：本地目录、直接附加磁盘、物理分区和 LVM 卷组。这些存储卷存储客户机虚拟机镜像，或作为额外存储附加到客户机虚拟机。由于本地存储池直接附加到主机物理服务器，它们在开发、测试和小型部署非常有用，不需要迁移或大量客户机虚拟机。本地存储池不适用于许多生产环境，因为本地存储池不支持实时迁移。
- 网络（共享）存储池 - 网络的存储池包括使用标准协议通过网络共享的存储设备。使用 virt-manager 在主机物理机之间迁移虚拟机时需要网络存储，但在使用 virsh 迁移时是可选的。网络的存储池由 libvirt 管理。网络存储池支持的协议包括：

- 基于光纤通道的 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。

注意

不应创建或使用多路径存储池，因为它们没有被完全支持。

11.2. 卷

存储池被分成多个存储卷。存储卷是物理分区、LVM 逻辑卷、基于文件的磁盘镜像以及 libvirt 处理的其他存储类型的抽象。无论底层硬件是什么，存储卷都作为本地存储设备向虚拟客户机呈现。

引用卷

要引用特定卷，可以使用三种方法：

卷和存储池的名称

卷可以通过名称来指代，以及它所属的存储池的标识符。在 virsh 命令行中，格式为 **--pool storage_pool volume_name**。

例如，在 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

virsh #
```

主机物理机器系统中存储的的完整路径

卷也可能由文件系统中的完整路径来引用。使用此方法时，不需要包含池标识符。

例如，名为 **secondimage.img** 的卷，对主机物理机器系统作为 **/images/secondimage.img** 可见。该镜像可以指代为 **/images/secondimage.img**。

```
# virsh vol-info /images/secondimage.img
Name:           secondimage.img
Type:           file
Capacity:       20.00 GB
Allocation:     136.00 kB
```

唯一卷密钥

当卷首次在虚拟化系统中创建时，将生成唯一标识符并为其分配它。唯一标识符术语 卷键。此卷密钥的格式因所使用的存储而异。

与基于块的存储（如 **LVM**）一同使用时，卷密钥可能会采用以下格式：

```
c3pKz4-qPVc-Xf7M-7WNM-WJc8-qSiz-mtvpGn
```

与基于文件的存储一起使用时，卷密钥可能是卷存储的完整路径的副本。

```
/images/secondimage.img
```

例如，卷键为 **WIvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr:**

```
# virsh vol-info WIvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
Name:           firstimage
Type:           block
Capacity:       20.00 GB
Allocation:     20.00 GB
```

virsh 提供在卷名称、卷路径或卷密钥间进行转换的命令：

vol-name

当提供卷路径或卷密钥时，返回卷名称。

```
# virsh vol-name /dev/guest_images/firstimage
firstimage
# virsh vol-name Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

vol-path

当提供卷密钥或存储池标识符和卷名称时，返回卷路径。

```
# virsh vol-path Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
/dev/guest_images/firstimage
# virsh vol-path --pool guest_images firstimage
/dev/guest_images/firstimage
```

vol-key 命令

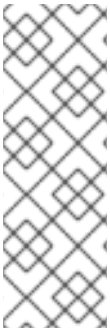
当提供卷路径或存储池标识符和卷名称时，返回卷密钥。

```
# virsh vol-key /dev/guest_images/firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
# virsh vol-key --pool guest_images firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

第 12 章 存储池

本章包含创建分类类型的存储池的说明。存储池是 管理员设置的存储数量（通常是专用存储管理员）供虚拟机使用。存储池通常由存储管理员或系统管理员划分到存储卷，卷则作为块设备分配给客户机虚拟机。

> **例 12.1. NFS 存储池**
>
> 假设负责 NFS 服务器的存储管理员创建了共享来存储客户机虚拟机的数据。系统管理员将在主机物理计算机上定义具有共享详细信息的池(nfs.example.com:/path/to/share should mounted on /vm_data)。当池启动时，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。如果池配置为自动启动，libvirt 可确保 NFS 共享挂载到 libvirt 启动时指定的目录中。
>
> 池启动后，NFS 共享的文件会被报告为卷，然后使用 libvirt API 查询存储卷的路径。然后可将卷的路径复制到客户机虚拟机 XML 定义文件的部分，该文件描述了客户机虚拟机块设备的源存储。使用 NFS 时，使用 libvirt API 的应用程序可以在池中创建和删除卷（NFS 共享中的文件）到池大小的限制（共享的最大存储容量）。并非所有池类型都支持创建和删除卷。在这种情况下，停止池需要启动操作，卸载 NFS 共享。销毁操作不会修改共享中的数据，尽管名称也是如此。详情请查看 man virsh。



注意

正确操作客户机虚拟机不需要存储池和卷。池和卷为 libvirt 提供了一种方式，可确保特定的存储可供虚拟机使用，但有些管理员更喜欢管理自己的存储和客户机虚拟机，无需定义的任何池或卷即可正确运行。在不使用池的系统上，系统管理员必须确保 guest 虚拟机存储的可用性使用自己喜欢的任何工具（例如，将 NFS 共享添加到主机物理计算机的 fstab 中），以便在启动时挂载共享。

警告

在客户机上创建存储池时，请务必遵循安全性注意事项。Red Hat Enterprise Linux 虚拟化安全指南 中会更加详细地探讨此信息，网址为：https://access.redhat.com/site/documentation/

12.1. 基于磁盘的存储池

本节介绍为客户机虚拟机创建基于磁盘存储设备。

警告

不应该向客户机授予对整个磁盘或块设备的写入权限（例如：/dev/sdb）。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机，客户机可能会对其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。

12.1.1. 使用 virsh 创建基于磁盘的存储池

这个过程使用 virsh 命令的磁盘设备创建新存储池。

警告

将磁盘专用于存储池将重新格式化并清除在磁盘设备上存储的所有数据。强烈建议您在以以下步骤开始前备份存储设备。

1. 在磁盘上创建 GPT 磁盘标签

   磁盘必须使用 GUID 分区表 (GPT)磁盘标签重新标记。GPT 磁盘标签允许在每个设备中创建大量分区（最多 128 个分区）。GPT 分区表可以存储比 MS-DOS 分区表更多的分区数据。

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

2. 创建存储池配置文件

创建包含新设备所需的存储池信息的临时 XML 文本文件。

该文件必须采用如下所示的格式，并包含以下字段：

<name>guest_images_disk</name>

name 参数决定存储池的名称。本例在示例中使用名称 guest_images_disk。

<device path='/dev/sdb'/>

带有 device 属性的 path 参数指定存储设备的设备路径。这个示例使用 /dev/sdb 设备。

<target> <path>/dev</path></target>

带有 target 子参数的文件系统 path 参数决定主机物理机器文件系统中的位置，以附加使用此存储池创建的卷。

例如：sdb1、sdb2、sdb3。使用 /dev/，如以下示例所示，从这个存储池创建的卷可以作为 /dev /sdb1、/dev/sdb2、/ dev/sdb3 进行访问。

<format type='gpt'/>

format 参数指定分区表类型。这个示例使用以下示例中的 gpt 与上一步中创建的 GPT 磁盘标签类型匹配。

使用文本编辑器为存储池设备创建 XML 文件。

例 12.2. 基于磁盘的存储设备存储池

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

```
-
```

3. **附加该设备**

   使用 virsh pool-define 命令和上一步中创建的 XML 配置文件添加存储池定义。

```
# virsh pool-define ~/guest_images_disk.xml
Pool guest_images_disk defined from /root/guest_images_disk.xml
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_disk    inactive   no
```

4. **启动存储池**

   使用 virsh pool-start 命令启动存储池。验证已使用 virsh pool-list --all 命令启动池。

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

5. **打开自动启动**

   为存储池打开 autostart。autostart 将 libvirtd 服务配置为在服务启动时启动存储池。

```
# virsh pool-autostart guest_images_disk
Pool guest_images_disk marked as autostarted
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_disk    active     yes
```

6. **验证存储池配置**

   验证存储池是否已正确创建，报告的大小是否正确，以及状态报告 正在运行。

```
# virsh pool-info guest_images_disk
Name:           guest_images_disk
UUID:           551a67c8-5f2a-012c-3844-df29b167431c
State:          running
Capacity:       465.76 GB
Allocation:     0.00
Available:      465.76 GB
# ls -la /dev/sdb
```

```
brw-rw----. 1 root disk 8, 16 May 30 14:08 /dev/sdb
# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
```

7. **可选：删除临时配置文件**

   如果不需要，请删除临时存储池 XML 配置文件。

```
# rm ~/guest_images_disk.xml
```

基于磁盘的存储池现在可用。

12.1.2. 使用 virsh 删除存储池

以下命令演示了如何使用 virsh 删除存储池：

1. **为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。**

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

2. **删除存储池的定义**

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

12.2. 基于分区的存储池

本节论述使用预格式化的块设备（分区）作为存储池。

在以下示例中，主机物理计算机将 500GB 硬盘驱动器(/dev/sdc)分区到一个 500GB、ext4 格式化分区(/dev/sdc1)。我们使用以下步骤为其设置一个存储池。

12.2.1. 使用 virt-manager 创建基于分区的存储池

这个过程使用存储设备的分区创建新存储池。

过程 12.1. 使用 virt-manager 创建基于分区的存储池

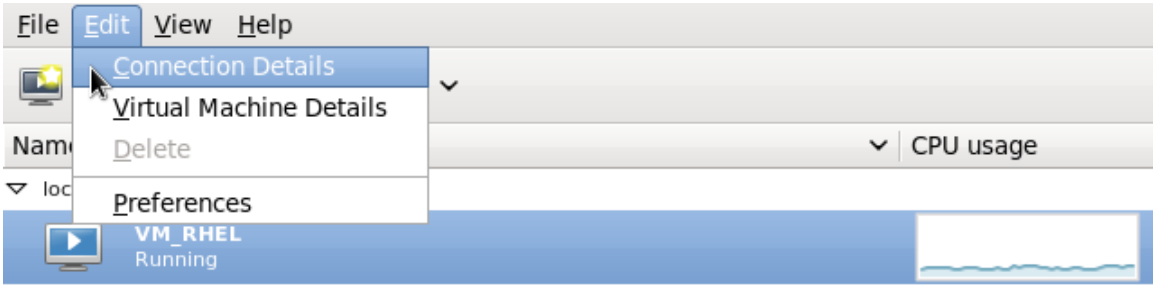
1. *打开存储池设置*

   a.

   在 virt-manager 图形界面，从主窗口中选择主机物理计算机。

   打开 Edit 菜单，然后选择 Connection Details

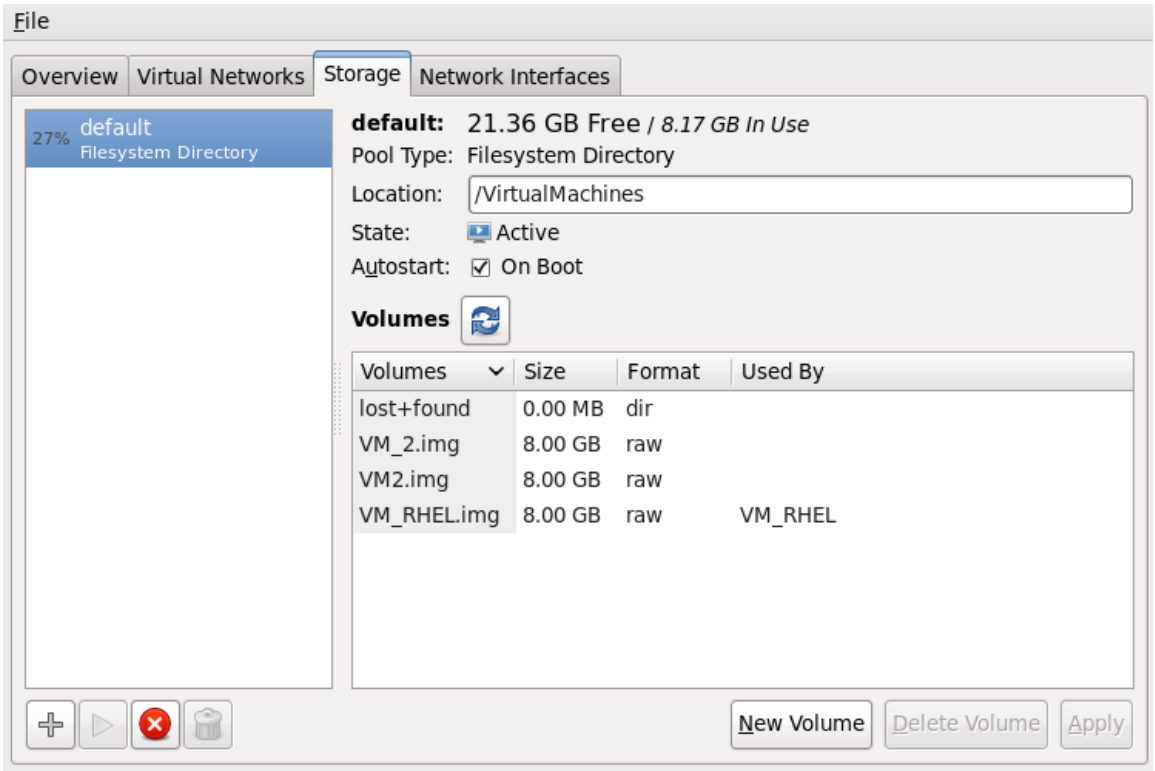
   图 12.1. 连接详情



   b.

   单击 Connection Details 窗口中的 Storage 选项卡。

   图 12.2. 存储标签



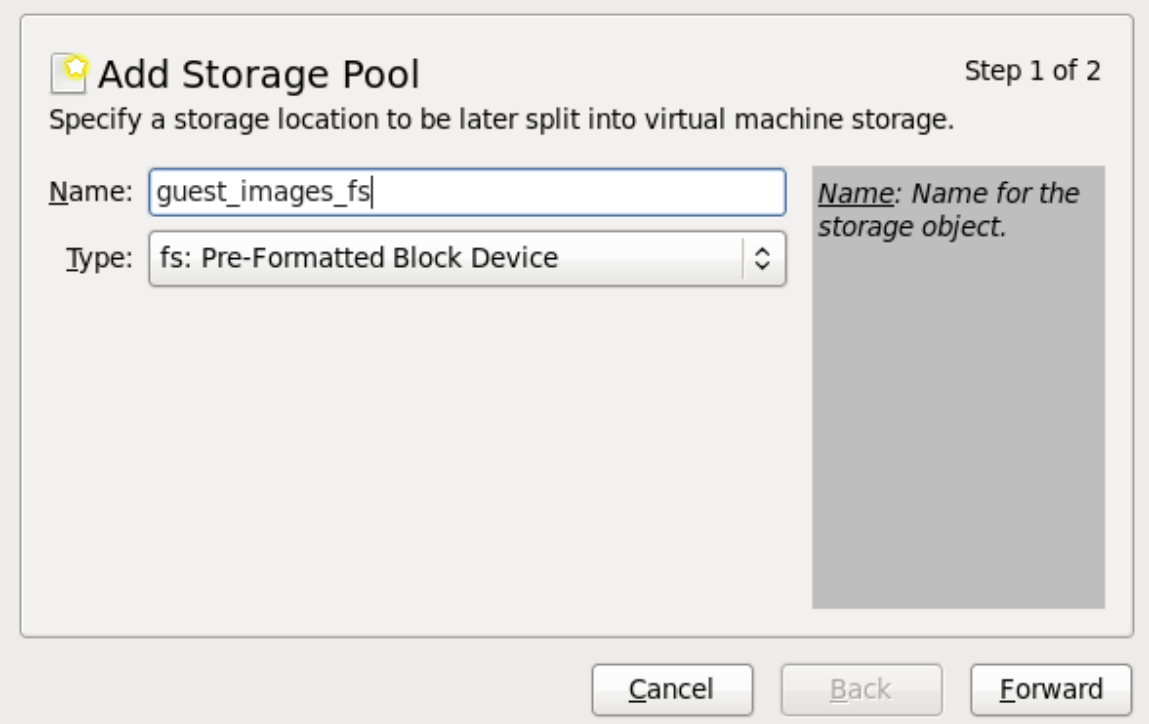
2. *创建新存储池*

   a. *添加新池（第 1 部分）*

按 + 按钮（添加池按钮）。此时会出现 Add a New Storage Pool 向导。

为存储池选择一个 名称。这个示例使用名称 guest_images_fs。将 Type 更改为 fs: PreFormatted Block Device。

图 12.3. 存储池名称和类型

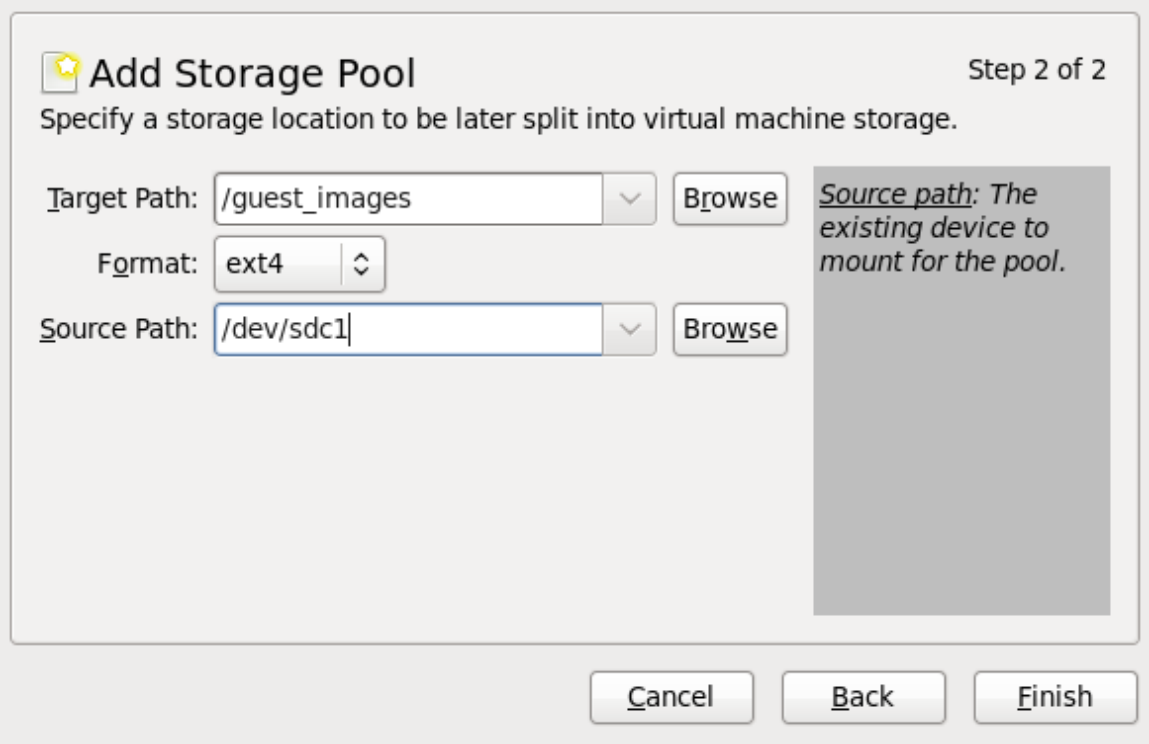


按 转发 按钮继续。

b. *添加新池（第 2 部分）*

更改 Target Path、Format 和 Source Path 字段。

图 12.4. 存储池路径和格式



目标路径

在 Target Path 字段中，为存储池输入挂载源设备的位置。如果该位置尚不存在，virt-manager 将创建目录。

格式

从 Format 列表中选择一个格式。设备使用所选格式进行格式化。

这个示例使用 ext4 文件系统，它是默认的 Red Hat Enterprise Linux 文件系统。

源路径

在 Source Path 字段中输入设备。

这个示例使用 /dev/sdc1 设备。

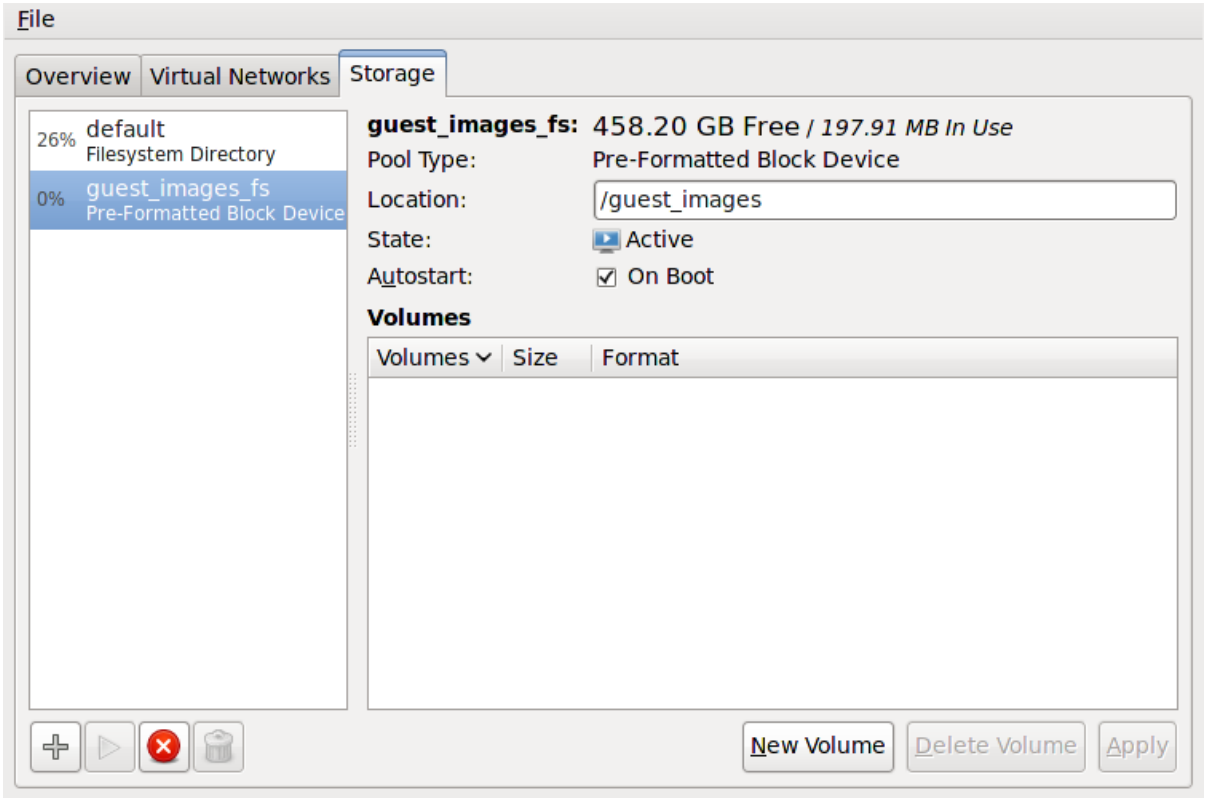
验证详细信息，然后按 "完成 "按钮创建存储池。

3. *验证新存储池*

在几秒钟后，新存储池会出现在左侧的存储列表中。验证大小已如预期报告，本例中为 458.20 GB 空闲。验证 State 字段将新存储池报告为 Active。

选择存储池。在 Autostart 字段中，单击 On Boot 复选框。这将确保无论 libvirtd 服务启动时都能启动存储设备。

图 12.5. 存储列表确认



现在创建了存储池，关闭 Connection Details 窗口。

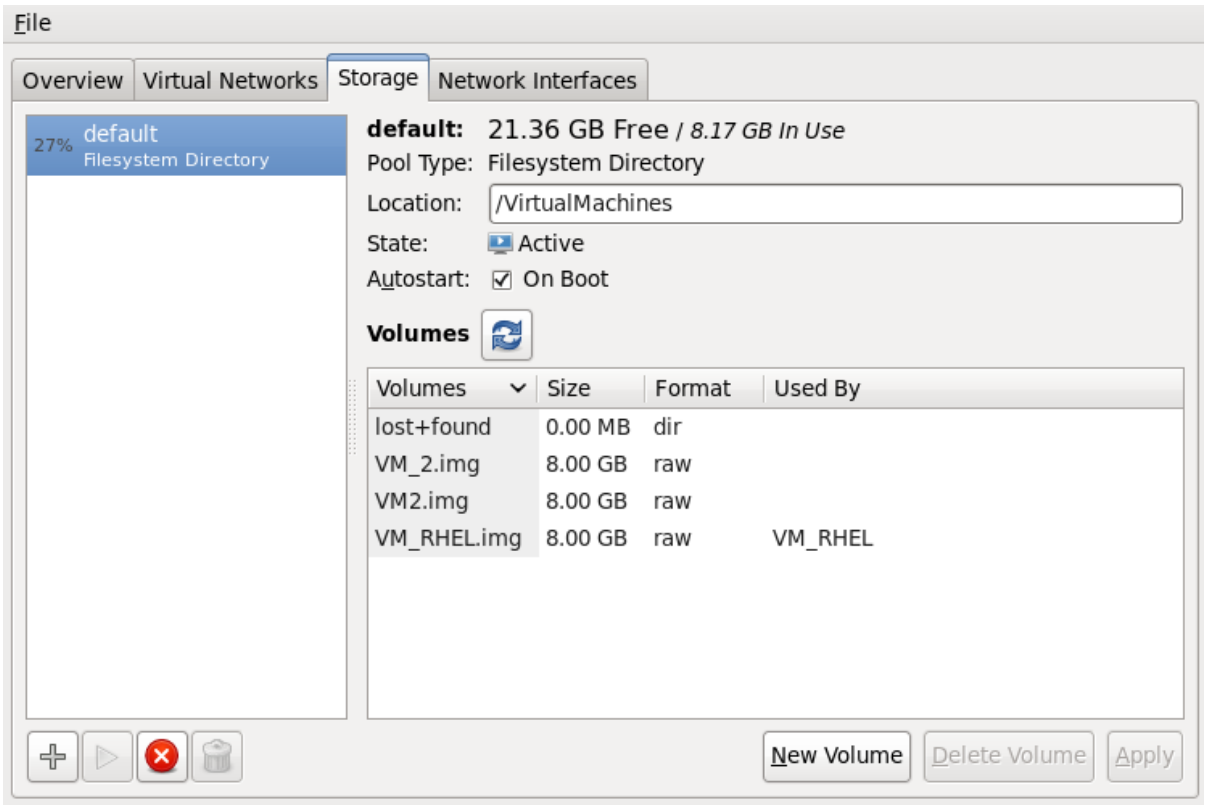
12.2.2. 使用 virt-manager 删除存储池

此流程演示了如何删除存储池。

1.

为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。要做到这一点，选择您要停止的存储池，并点击 Storage 窗口底部的红色 X 图标。

图 12.6. 停止图标



2. *点 Trash can 图标删除存储池。只有您首先停止存储池时才会启用此图标。*

12.2.3. 使用 virsh 创建基于分区的存储池

这部分论述了使用 virsh 命令创建基于分区的存储池。

> *警告*
>
> *不要使用此流程将整个磁盘分配为一个存储池（例如：/dev/sdb）。不应该对整个磁盘或块设备进行写入访问权限。仅使用此方法将分区（例如 /dev/sdb1）分配给存储池。*

过程 12.2. 使用 virsh 创建预格式化的块设备存储池

1. *创建存储池定义*

   使用 virsh pool-define-as 命令创建一个新的存储池定义。必须提供三个选项来定义预格式化的磁盘作为存储池：

分区名称

name 参数决定存储池的名称。本例使用以下示例中的 guest_images_fs 名称。

device

带有 device 属性的 path 参数指定存储设备的设备路径。这个示例使用分区 /dev/sdc1。

mountpoint

挂载格式化设备的本地文件系统中的 mountpoint。如果挂载点目录不存在，则 virsh 命令可以创建该目录。

本例中使用了 /guest_images 目录。

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

新的池和挂载点现已创建。

2. *验证新池*

列出 present 存储池。

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

3. *创建挂载点*

使用 virsh pool-build 命令为预格式化的文件系统存储池创建挂载点。

```
# 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
```

4. 启动存储池

   使用 virsh pool-start 命令将文件系统挂载到挂载点，并让池可用。

```
# 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
```

5. 打开自动启动

   默认情况下，使用 virsh 定义的存储池不会设置为在每次 libvirtd 启动时自动启动。要补救这一点，请使用 virsh pool-autostart 命令启用自动启动。现在，每次 libvirtd 启动时都会自动启动存储池。

```
# 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
```

6. 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，并且报告为运行 状态。验证文件系统的挂载点中存在"lost+found"目录，表示挂载该设备。

```
# 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
```

12.2.4. 使用 virsh 删除存储池

1. 为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。

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

2. 另外，如果要删除存储池所在的目录，请使用以下命令：

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

3. 删除存储池的定义

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

12.3. 基于目录的存储池

本节介绍将客户机虚拟机存储在主机物理机器的目录中。

可以通过 virt-manager 或 virsh 命令行工具创建基于目录的存储池。

12.3.1. 使用 virt-manager 创建基于目录的存储池

1. 创建本地目录

   a. 可选：为存储池创建新目录

      在主机物理计算机中为存储池创建 目录。这个示例使用名为 /guest virtual_images 的目录。

```
# mkdir /guest_images
```

   b. 设置目录所有权

      更改 目录的用户和组所有权。目录必须由 root 用户所有。

```
# chown root:root /guest_images
```

c. *设置目录权限*

更改 目录的文件权限。

```
# chmod 700 /guest_images
```

d. *验证更改*

验证权限已被修改。输出显示正确配置了空目录。

```
# ls -la /guest_images
total 8
drwx------.  2 root root 4096 May 28 13:57 .
dr-xr-xr-x. 26 root root 4096 May 28 13:57 ..
```

2. ***配置 SELinux 文件上下文***

为新目录配置正确的 SELinux 上下文。请注意，池的名称和 目录不需要匹配。但是，当您关闭客户端虚拟机时，libvirt 必须把上下文设置为默认值。目录的上下文决定了这个默认值。值得显式标记目录 virt_image_t，因此，当客户机虚拟机关闭时，镜像会标记为"virt_image_t"，因此与在主机物理机器上运行的其他进程隔离开来。

```
# semanage fcontext -a -t virt_image_t '/guest_images(/.*)?'
# restorecon -R /guest_images
```

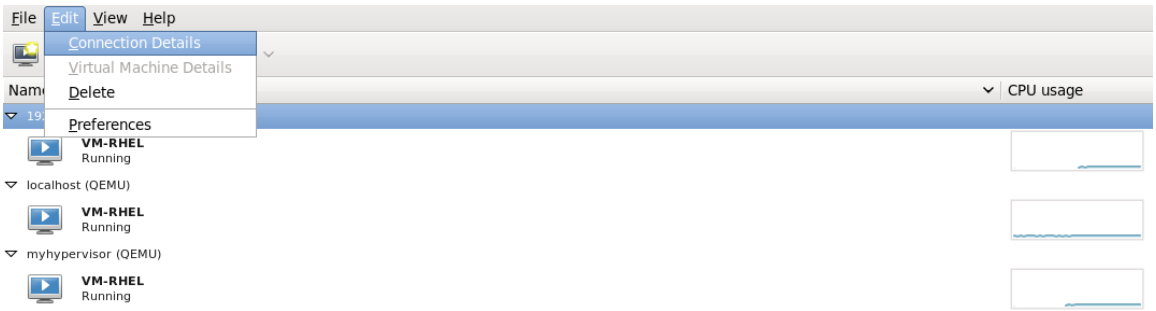
3. ***打开存储池设置***

a.

在 virt-manager 图形界面，从主窗口中选择主机物理计算机。

打开 Edit 菜单，然后选择 Connection Details

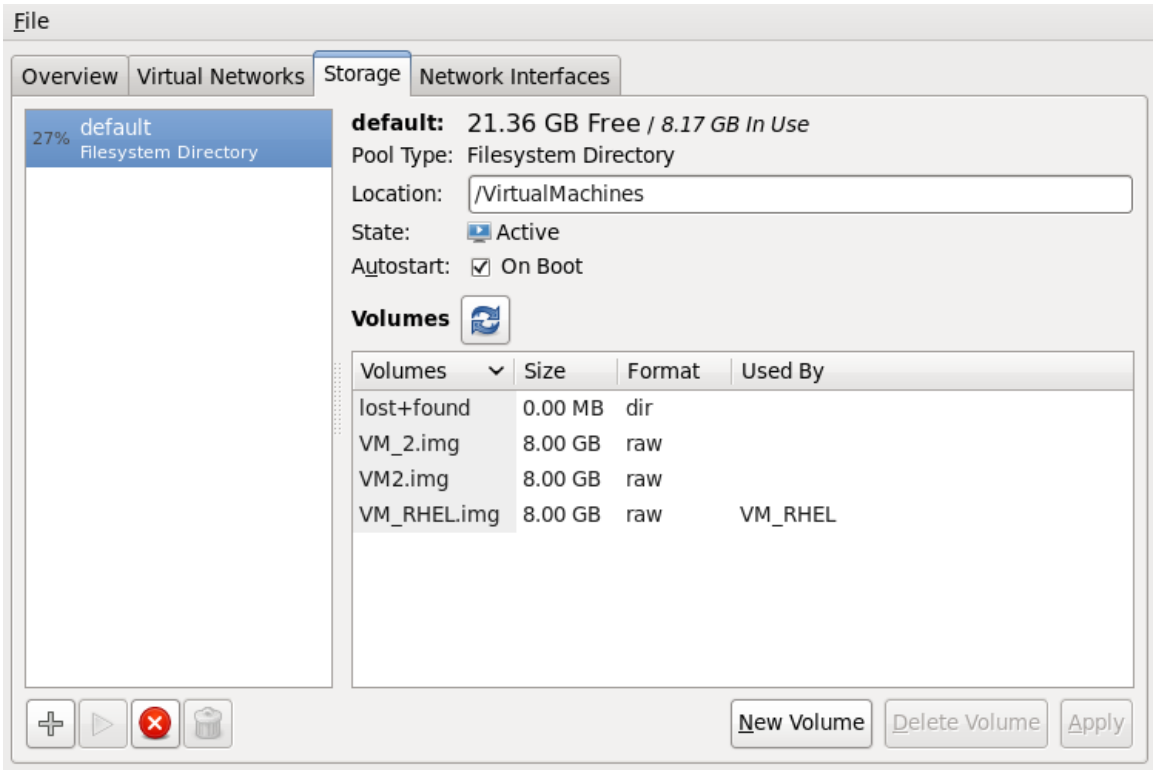
图 12.7. 连接详情窗口



b.

*单击 **Connection Details** 窗口中的 **Storage** 选项卡。*

图 12.8. 存储标签



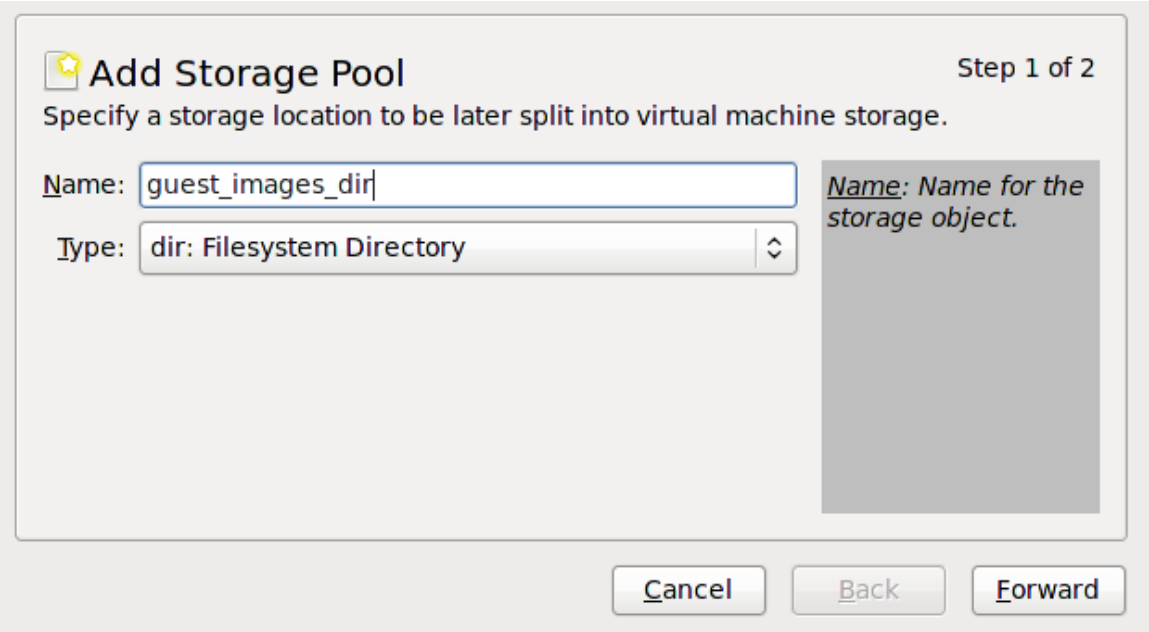
4. *创建新存储池*

a. ***添加新池（第 1 部分）***

*按 + 按钮（添加池按钮）。此时会出现 **Add a New Storage Pool** 向导。*

*为存储池选择一个 **名称**。这个示例使用名称 **guest_images**。将 **Type** 更改为 **dir: Filesystem Directory**。*

图 12.9. 将存储池命名为



按 转发 按钮继续。

b. *添加新池（第 2 部分）*

更改 Target Path 字段。例如： /guest_images。

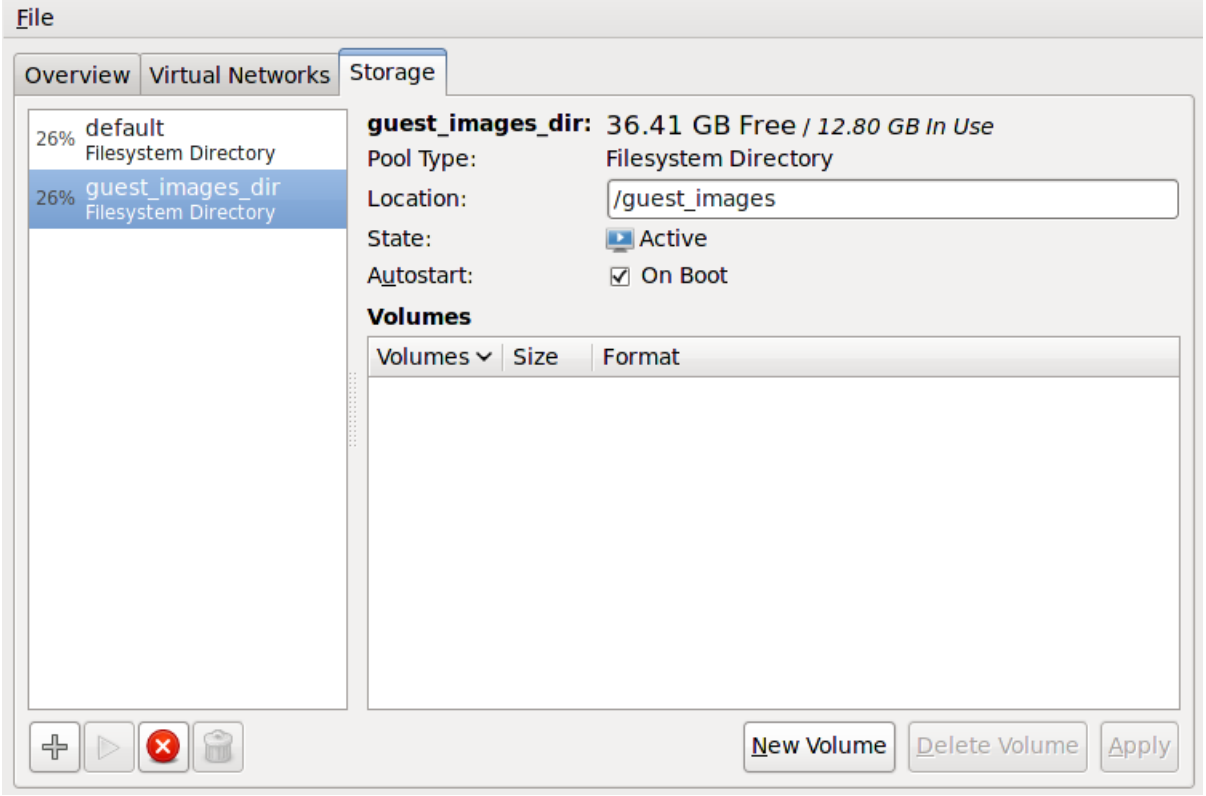
验证详细信息，然后按 "完成 "按钮创建存储池。

5. *验证新存储池*

在几秒钟后，新存储池会出现在左侧的存储列表中。验证大小已如预期报告，本例中为 36.41 GB 空闲。验证 State 字段将新存储池报告为 Active。

选择存储池。在 Autostart 字段中，确认选中 On Boot 复选框。这将确保无论 libvirtd 服务启动时都启动存储池。

图 12.10. 验证存储池信息



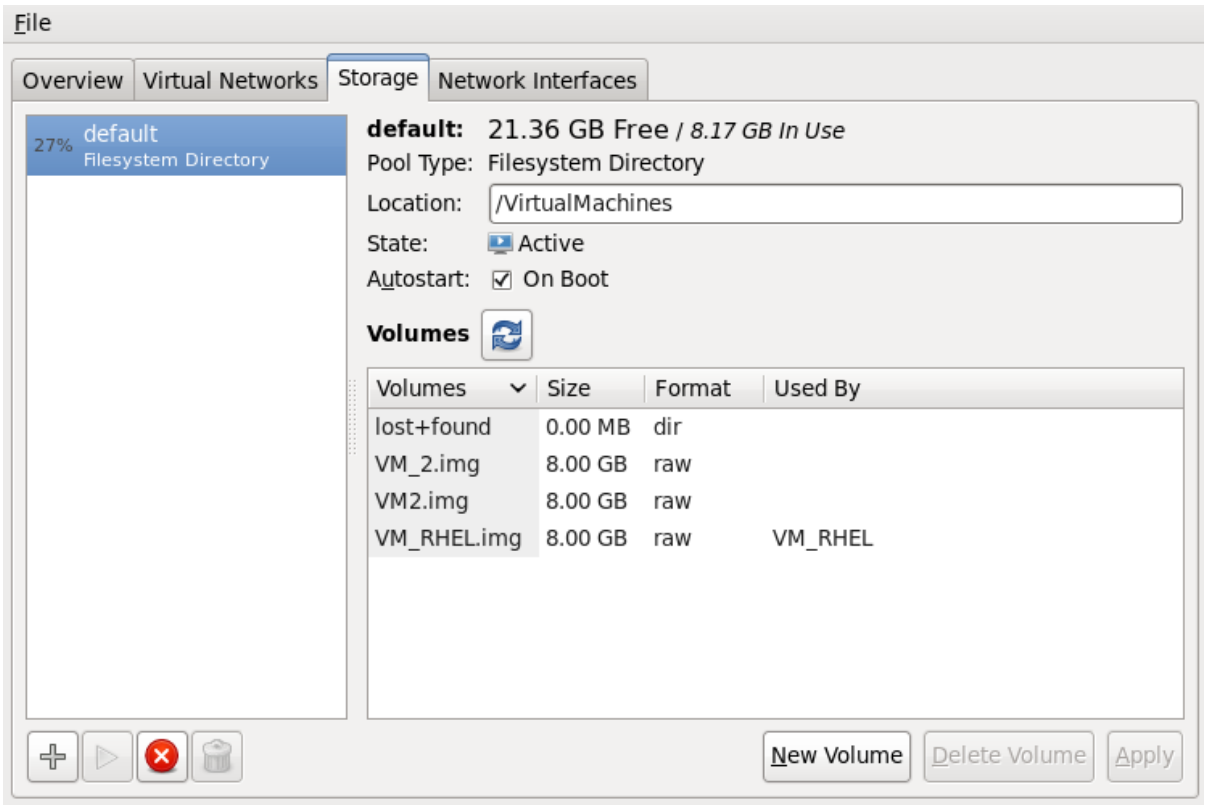
*现在创建了存储池，关闭 **Connection Details** 窗口。*

12.3.2. 使用 virt-manager 删除存储池

此流程演示了如何删除存储池。

1. *为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。要做到这一点，选择您要停止的存储池，并点击 **Storage** 窗口底部的红色 **X** 图标。*

图 12.11. 停止图标



2. *点 Trash can 图标删除存储池。只有您首先停止存储池时才会启用此图标。*

12.3.3. 使用 virsh 创建基于目录的存储池

1. *创建存储池定义*

   使用 virsh pool-define-as 命令定义新的存储池。创建基于目录的存储池需要两个选项：

   - *存储池的名称。*

     这个示例使用名称 guest_images。本例中使用的所有进一步 virsh 命令使用此名称。

   - *用于存储客户机镜像文件的文件系统目录的路径。如果该目录不存在，virsh 将创建该目录。*

     这个示例使用 /guest_images 目录。

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

2. 验证是否列出了存储池

   验证存储池对象是否已正确创建，状态则报告为 不活动。

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

3. 创建本地目录

   使用 virsh pool-build 命令为目录 guest_images （例如，如下所示）构建基于目录的存储池：

```
# virsh pool-build guest_images
Pool guest_images built
# ls -la /guest_images
total 8
drwx------.  2 root root 4096 May 30 02:44 .
dr-xr-xr-x. 26 root root 4096 May 30 02:44 ..
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images     inactive   no
```

4. 启动存储池

   使用 virsh 命令 pool-start 启用目录存储池，从而允许池的卷用作客户机磁盘镜像。

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

5. 打开自动启动

   为存储池打开 autostart。autostart 将 libvirtd 服务配置为在服务启动时启动存储池。

```
# virsh pool-autostart guest_images
Pool guest_images marked as autostarted
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images         active     yes
```

6. 验证存储池配置

   验证存储池是否已正确创建，其大小会被正确报告，并且报告为运行 状态。如果希望池可以被访问，即使客户机虚拟机没有运行，请确保将 永久 报告为 **yes**。如果您希望池在服务启动时自动启动，请确保将 **Autostart** 报告为 **yes**。

```
# virsh pool-info guest_images
Name:           guest_images
UUID:           779081bf-7a82-107b-2874-a19a9c51d24c
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       49.22 GB
Allocation:     12.80 GB
Available:      36.41 GB

# ls -la /guest_images
total 8
drwx------.  2 root root 4096 May 30 02:44 .
dr-xr-xr-x. 26 root root 4096 May 30 02:44 ..
#
```

现在可以使用基于目录的存储池。

12.3.4. 使用 virsh 删除存储池

以下命令演示了如何使用 **virsh** 删除存储池：

1. 为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。

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

2. 另外，如果要删除存储池所在的目录，请使用以下命令：

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

3. 删除存储池的定义

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

12.4. 基于 LVM 的存储池

本章论述了将 LVM 卷组用作存储池。

基于 LVM 的存储组提供了 LVM 的完整灵活性。

注意

目前，基于 LVM 的存储池无法进行精简配置。

注意

有关 LVM 的详情，请参考 Red Hat Enterprise Linux Storage Administration Guide。

警告

基于 LVM 的存储池需要一个完整磁盘分区。如果使用这些步骤激活新分区/设备，分区将被格式化并清除所有数据。如果使用主机的现有卷组(VG)，则不会删除任何内容。建议您在开始以下流程前备份存储设备。

12.4.1. 使用 virt-manager 创建基于 LVM 的存储池

基于 LVM 的存储池可以使用现有的 LVM 卷组，或者在空白分区中创建新 LVM 卷组。

1. 可选：为 LVM 卷创建新分区

   这些步骤描述了如何在新硬盘中创建新分区和 LVM 卷组。

警告

这个过程将从所选存储设备中删除所有数据。

a. **创建新分区**

使用 fdisk 命令从命令行创建新磁盘分区。以下示例创建一个使用存储设备 /dev/sdb 上的整个磁盘的新分区。

```
# fdisk /dev/sdb
Command (m for help):
```

为新分区按 n。

b.

在主分区中按 p。

```
Command action
   e   extended
   p   primary partition (1-4)
```

c.

选择可用分区号。在这个示例中，通过输入 1 来选择第一个分区。

```
Partition number (1-4): 1
```

d.

按 Enter 输入默认柱面。

```
First cylinder (1-400, default 1):
```

e.

选择分区的大小。在这个示例中，通过按 Enter 来分配整个磁盘。

```
Last cylinder or +size or +sizeM or +sizeK (2-400, default 400):
```

f.

按 t 设置分区类型。

-

```
Command (m for help): t
```

g. 选择您在前面的步骤中创建的分区。在这个示例中，分区号是 1。

```
Partition number (1-4): 1
```

h. 为 Linux LVM 分区输入 8e。

```
Hex code (type L to list codes): 8e
```

i. 将更改写入磁盘并退出。

```
Command (m for help): w
Command (m for help): q
```

j. 创建新 LVM 卷组

使用 vgcreate 命令创建一个新的 LVM 卷组。这个示例创建名为 guest_images_lvm 的卷组。

```
# vgcreate guest_images_lvm /dev/sdb1
  Physical volume "/dev/vdb1" successfully created
  Volume group "guest_images_lvm" successfully created
```

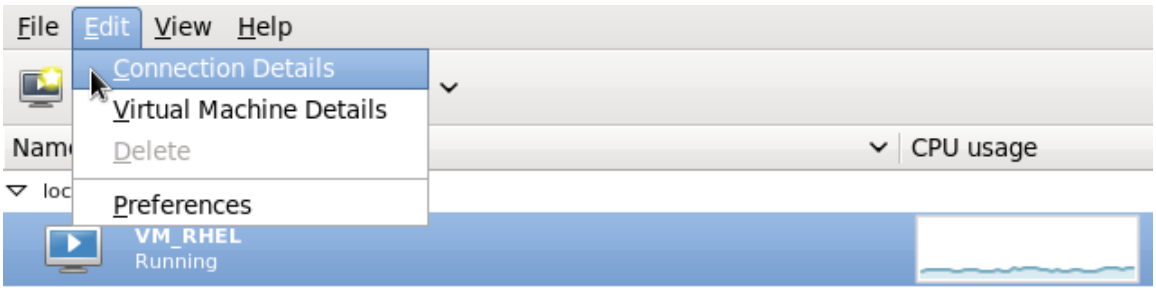
新的 LVM 卷组 guest_images_lvm 现在可以用于基于 LVM 的存储池。

2. 打开存储池设置

a. 在 virt-manager 图形界面中，从主窗口中选择主机。

打开 Edit 菜单，然后选择 Connection Details

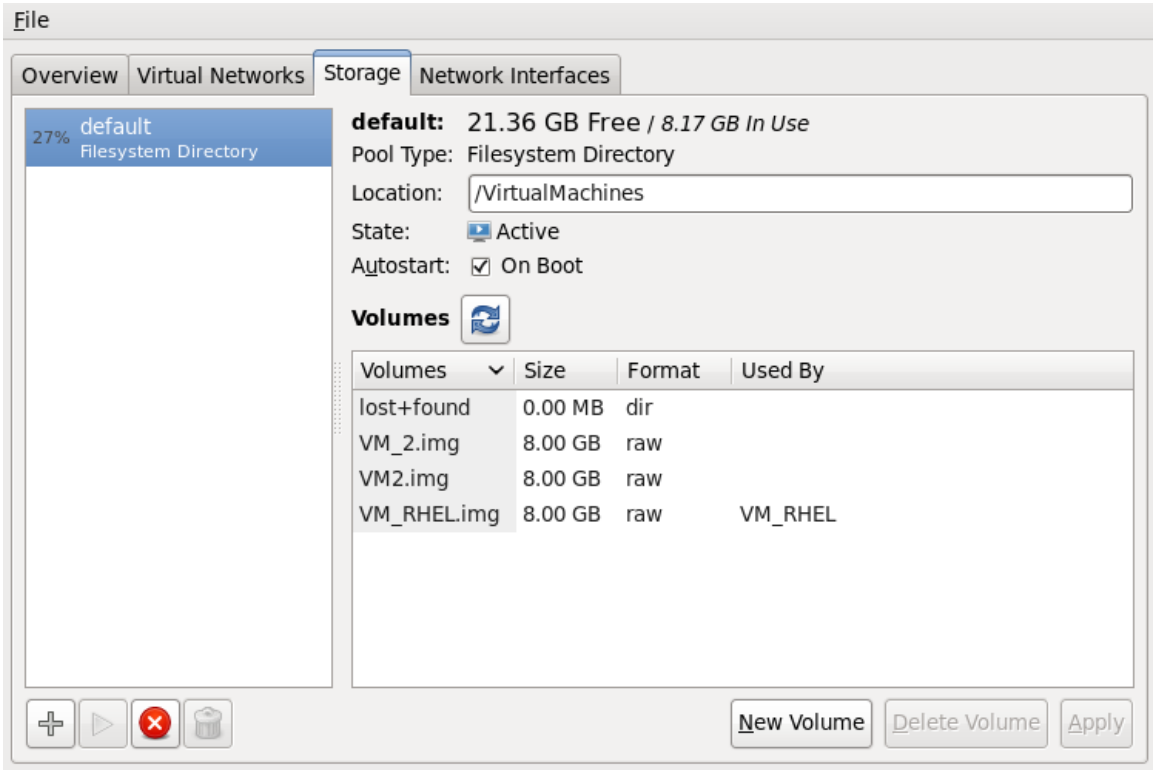
图 12.12. 连接详情



b.

点 Storage 选项卡。

图 12.13. 存储标签



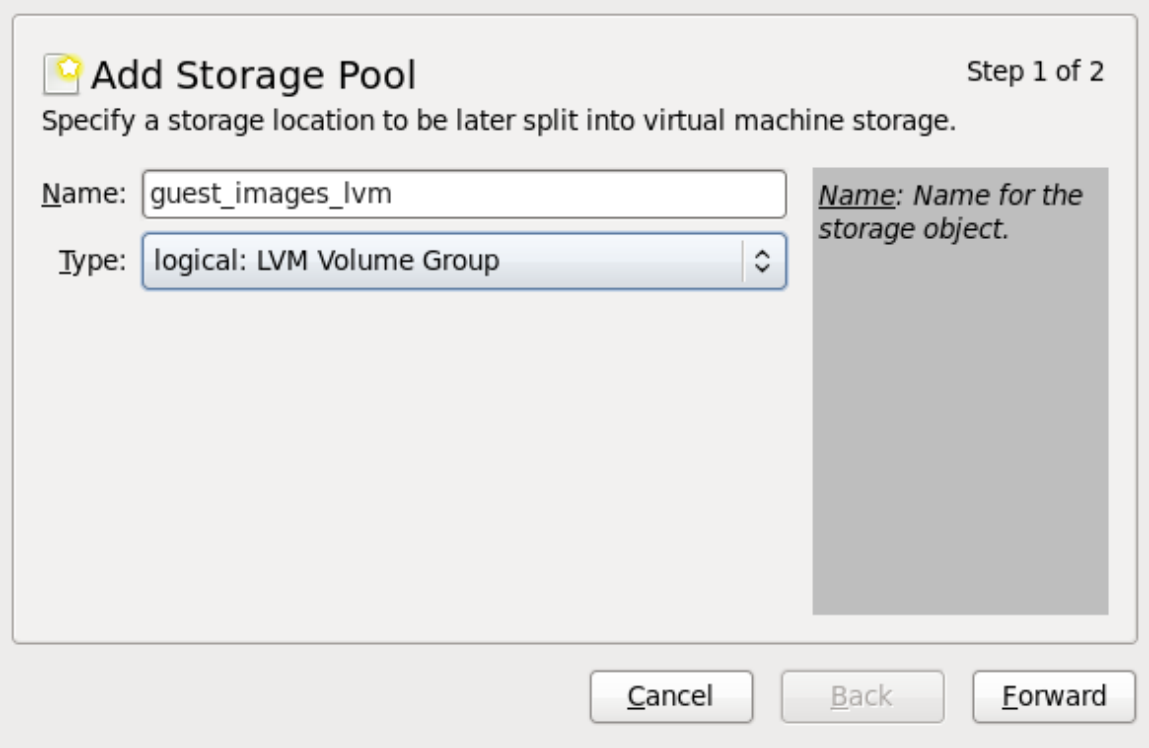
3. *创建新存储池*

a. *启动向导*

按 + 按钮（添加池按钮）。此时会出现 Add a New Storage Pool 向导。

为存储池选择一个 名称。在本例中，我们使用 guest_images_lvm。然后将类型更改为 逻辑：LVM 卷组，然后

图 12.14. 添加 LVM 存储池



按 转发 按钮继续。

b. *添加新池（第 2 部分）*

更改 Target Path 字段。这个示例使用 /guest_images。

现在填写 Target Path 和 Source Path 字段，然后选中 Build Pool 复选框。

- *使用 Target Path 字段选择现有 LVM 卷组或新卷组的名称。默认格式为 /dev/storage_pool_name。*

  本例使用名为 /dev/guest_images_lvm 的新卷组。

- *如果在 目标路径中使用了现有 LVM 卷组，则 Source Path 字段是可选的。*

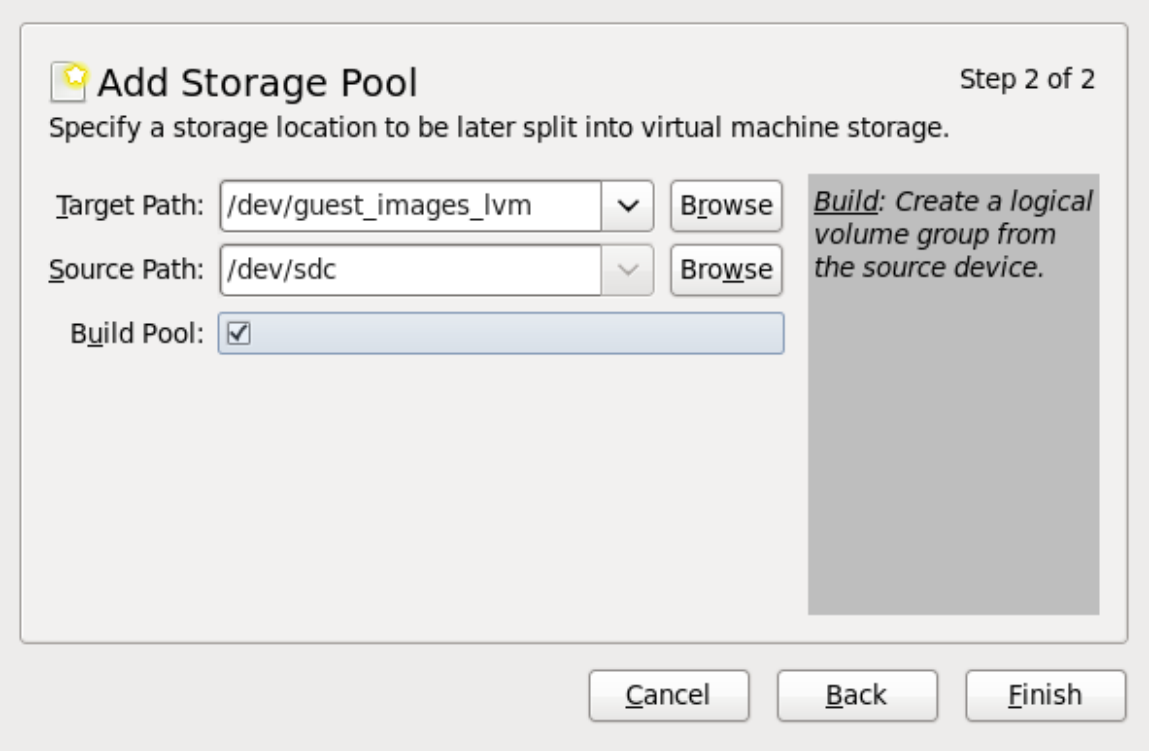
  对于新的 LVM 卷组，在 Source Path 字段中输入存储设备的位置。这个示例使用空白分区 /dev/sdc。

- *Build Pool 复选框指示 virt-manager 创建新的 LVM 卷组。如果您使用现有的卷*

*组，则不应选择 **Build Pool** 复选框。*

*本示例使用空分区来创建新卷组，因此必须选择 **Build Pool** 复选框。*

图 12.15. 添加目标和源

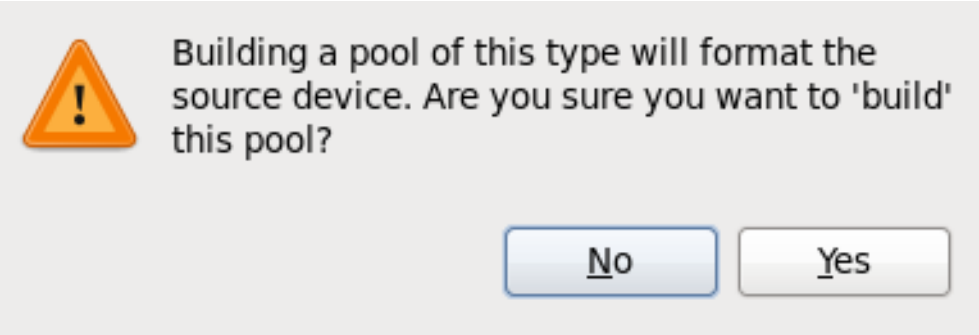


验证详细信息并按 "完成 "按钮格式化 LVM 卷组并创建存储池。

c. *确认要格式化的设备*

此时会出现一个警告信息。

图 12.16. 警告信息



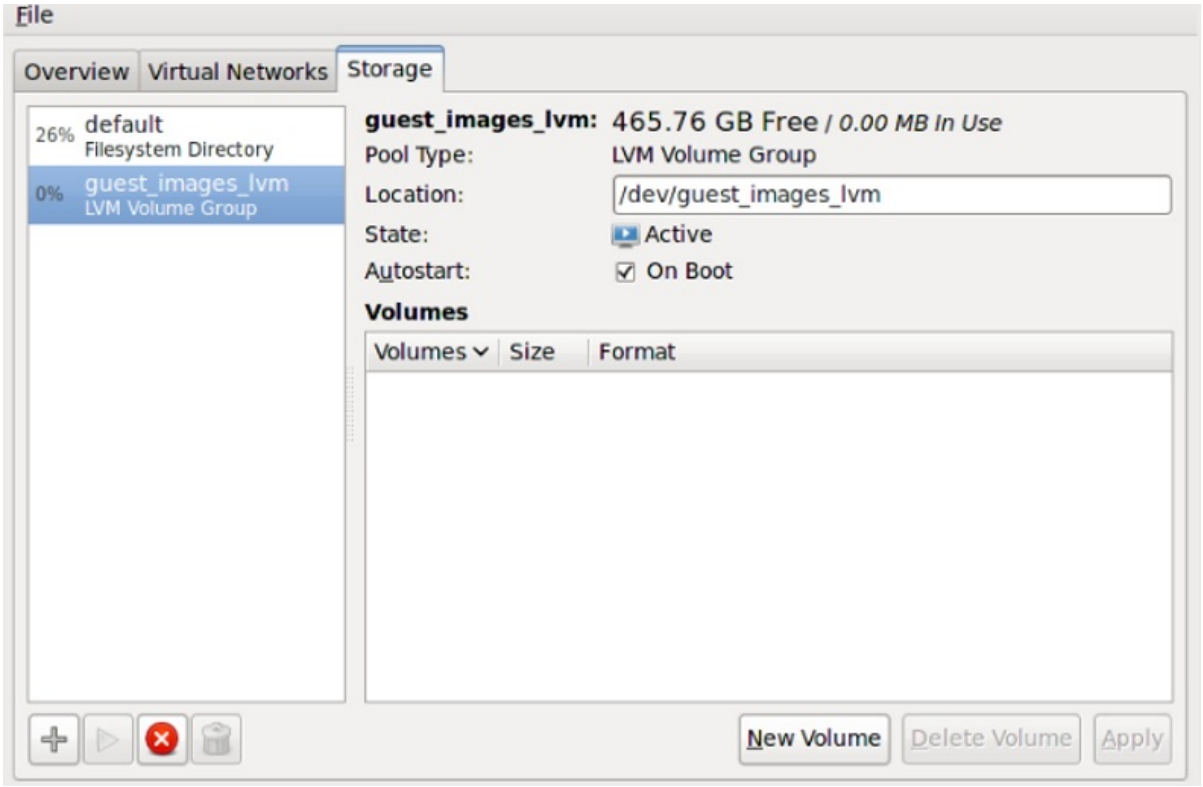
按" 是 "按钮继续清除存储设备上的所有数据并创建存储池。

4. *验证新存储池*

新存储池将在几秒钟后左侧的列表中显示。验证您期望的详细信息，如示例中 465.76 GB 可用。另外，验证 State 字段会报告新的存储池为 Active。

通常最好启用了 Autostart 复选框，以确保存储池在 libvirtd 自动启动。

图 12.17. 确认 LVM 存储池详情



关闭 Host Details 对话框，因为任务现已完成。

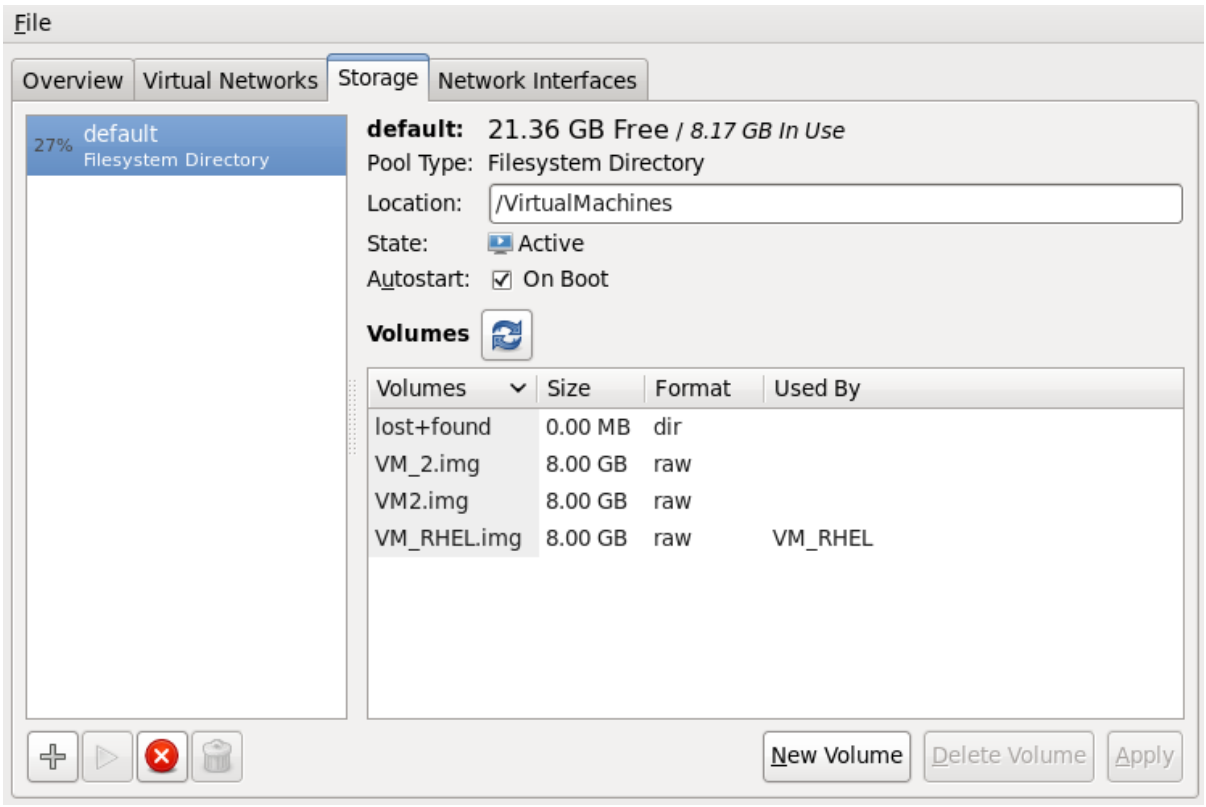
12.4.2. 使用 virt-manager 删除存储池

此流程演示了如何删除存储池。

1.

为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。要做到这一点，选择您要停止的存储池，并点击 Storage 窗口底部的红色 X 图标。

图 12.18. 停止图标



2. *点 Trash can 图标删除存储池。只有您首先停止存储池时才会启用此图标。*

12.4.3. 使用 virsh 创建基于 LVM 的存储池

本节概述了使用 virsh 命令创建基于 LVM 的存储池所需的步骤。它使用一个名为 guest_images_lvm 的池的示例，该池来自单个驱动器(/dev/sdc)。它只是一个示例，您的设置应根据情况替代。

过程 12.3. 使用 virsh 创建基于 LVM 的存储池

1. *定义池名称 guest_images_lvm。*

```
# virsh pool-define-as guest_images_lvm logical - - /dev/sdc libvirt_lvm \ /dev/libvirt_lvm
Pool guest_images_lvm defined
```

2. *根据指定名称构建池。如果您使用已存在的卷组，请跳过这一步。*

```
# virsh pool-build guest_images_lvm

Pool guest_images_lvm built
```

3.

初始化新池。

```
# virsh pool-start guest_images_lvm

Pool guest_images_lvm started
```

4.

使用 vgs 命令显示卷组信息。

```
# vgs
VG          #PV #LV #SN Attr   VSize   VFree
libvirt_lvm   1   0   0 wz--n- 465.76g 465.76g
```

5.

将池设置为自动启动。

```
# virsh pool-autostart guest_images_lvm
Pool guest_images_lvm marked as autostarted
```

6.

使用 virsh 命令列出可用的池。

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

7.

以下命令演示了在此池中创建三个卷（volume1、 volume2 和 volume3）。

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

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

# virsh vol-create-as guest_images_lvm volume3 8G
Vol volume3 created
```

8.

使用 virsh 命令，列出这个池中的可用卷。

```
# virsh vol-list guest_images_lvm
Name                 Path
-----------------------------------------
```

```
volume1              /dev/libvirt_lvm/volume1
volume2              /dev/libvirt_lvm/volume2
volume3              /dev/libvirt_lvm/volume3
```

9. 以下两个命令（**lvscan** 和 **lvs**）显示有关新创建的卷的更多信息。

```
# lvscan
ACTIVE            '/dev/libvirt_lvm/volume1' [8.00 GiB] inherit
ACTIVE            '/dev/libvirt_lvm/volume2' [8.00 GiB] inherit
ACTIVE            '/dev/libvirt_lvm/volume3' [8.00 GiB] inherit

# lvs
LV      VG          Attr     LSize   Pool Origin Data%  Move Log Copy%  Convert
volume1 libvirt_lvm -wi-a-   8.00g
volume2 libvirt_lvm -wi-a-   8.00g
volume3 libvirt_lvm -wi-a-   8.00g
```

12.4.4. 使用 virsh 删除存储池

以下命令演示了如何使用 **virsh** 删除存储池：

1. 为避免使用同一池的其他客户机出现任何问题，最好停止存储池并释放其使用中的任何资源。

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

2. 另外，如果要删除存储池所在的目录，请使用以下命令：

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

3. 删除存储池的定义

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

12.5. 基于 ISCSI 的存储池

本节介绍使用基于 **iSCSI** 的设备来存储客户机虚拟机。

iSCSI（互联网小型计算机系统接口）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）。

12.5.1. 配置软件 iSCSI 目标

scsi-target-utils 软件包提供了用于创建软件支持的 **iSCSI** 目标的工具。

过程 **12.4.** 创建 **iSCSI** 目标

1. **安装所需软件包**

   安装 **scsi-target-utils** 软件包及所有依赖项

```
# yum install scsi-target-utils
```

2. 启动 **tgtd** 服务

   tgtd 服务托管物理机器 **SCSI** 目标，并使用 **iSCSI** 协议托管物理机器目标。启动 **tgtd** 服务，并使用 **chkconfig** 命令重新启动后使服务持久。

```
# service tgtd start
# chkconfig tgtd on
```

3. **可选：创建 LVM 卷**

   LVM 卷对于 **iSCSI** 后备镜像很有用。对于 **guest** 虚拟机，**LVM** 快照和大小大小会很有用。这个示例在 **RAID5** 阵列中创建一个名为 **virtimage1** 的 **LVM** 镜像，用于托管 **iSCSI** 的客户机虚拟机。

   a. 创建 **RAID** 阵列

      Red Hat Enterprise Linux 部署指南 介绍了如何创建软件 **RAID5** 阵列。

   b. 创建 **LVM** 卷组

      使用 **vgcreate** 命令创建名为 **virtstore** 的卷组。

```
# vgcreate virtstore /dev/md1
```

   c. 创建 **LVM** 逻辑卷

      使用 **lvcreate** 命令，在 **virtstore** 卷组中创建名为 **virtimage1** 的逻辑卷组，大小为 **20GB**。

```
# lvcreate --size 20G -n virtimage1 virtstore
```

新逻辑卷 virtimage1 已准备好用于 iSCSI。

4. 可选：创建基于文件的镜像

基于文件的存储足以进行测试，但不建议用于生产环境或任何重要的 I/O 活动。此可选步骤为 iSCSI 目标创建名为 virtimage2.img 的文件。

a. 为镜像创建新目录

创建新目录来存储镜像。目录必须具有正确的 SELinux 上下文。

```
# mkdir -p /var/lib/tgtd/virtualization
```

b. 创建镜像文件

创建名为 virtimage2.img、大小为 10GB 的镜像。

```
# dd if=/dev/zero of=/var/lib/tgtd/virtualization/virtimage2.img bs=1M seek=10000 count=0
```

c. 配置 SELinux 文件上下文

为新镜像和目录配置正确的 SELinux 上下文。

```
# restorecon -R /var/lib/tgtd
```

基于文件的新映像 virtimage2.img 已准备好用于 iSCSI。

5. 创建目标

通过在 /etc/tgt/targets.conf 文件中添加 XML 条目，即可创建目标。target 属性需要 iSCSI 限定名称(IQN)。IQN 采用以下格式：

```
iqn.yyyy-mm.reversed domain name:optional identifier text
```

其中：

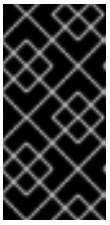
- YYYY-mm 表示设备已启动的年和月（例如 ：2010-05）；
- reversed 域名 是反向主机物理机器域名（例如 IQN 中的 server1.example.com ）是 com.example.server1；以及

- 可选标识符文本 是任何文本字符串，没有空格，可协助管理员识别设备或硬件。

本例为在 server1.example.com 上可选步骤创建的两类镜像创建 iSCSI 目标（可选标识符 test ）。将以下内容添加到 /etc/tgt/targets.conf 文件中。

```
<target iqn.2010-05.com.example.server1:iscsirhel6guest>
  backing-store /dev/virtstore/virtimage1  #LUN 1
  backing-store /var/lib/tgtd/virtualization/virtimage2.img  #LUN 2
  write-cache off
</target>
```

确保 /etc/tgt/targets.conf 文件包含 default-driver iscsi 行，以将驱动程序类型设置为 iSCSI。驱动程序默认使用 iSCSI。



重要

这个示例创建了一个全局可访问的目标，且无访问权限控制。有关实现安全访问的信息，请参阅 scsi-target-utils。

6. 重启 tgtd 服务

重启 tgtd 服务以重新载入配置更改。

```
# service tgtd restart
```

7. iptables 配置

通过 iptables 打开端口 3260，以进行 iSCSI 访问。

```
# iptables -I INPUT -p tcp -m tcp --dport 3260 -j ACCEPT
# service iptables save
# service iptables restart
```

8. 验证新目标

查看新目标，以确保设置成功，使用 tgt-admin --show 命令。

```
# tgt-admin --show
Target 1: iqn.2010-05.com.example.server1:iscsirhel6guest
System information:
```

```
Driver: iscsi
State: ready
I_T nexus information:
LUN information:
LUN: 0
    Type: controller
    SCSI ID: IET     00010000
    SCSI SN: beaf10
    Size: 0 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: None
LUN: 1
    Type: disk
    SCSI ID: IET     00010001
    SCSI SN: beaf11
    Size: 20000 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: /dev/virtstore/virtimage1
LUN: 2
    Type: disk
    SCSI ID: IET     00010002
    SCSI SN: beaf12
    Size: 10000 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: /var/lib/tgtd/virtualization/virtimage2.img
Account information:
ACL information:
ALL
```

警告

ACL 列表设置为 all。这允许本地网络上的所有系统访问这个设备。建议在生产环境中设置主机物理机器访问 ACL。

9. **可选：测试发现**

   测试新 iSCSI 设备是否可以发现。

```
# iscsiadm --mode discovery --type sendtargets --portal server1.example.com
127.0.0.1:3260,1 iqn.2010-05.com.example.server1:iscsirhel6guest
```

10. *可选：测试附加该设备*

    附加新设备(iqn.2010-05.com.example.server1:iscsirhel6guest)，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:iscsirhel6guest, portal:
10.0.0.1,3260]
Login to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260] successful.
```

    分离该设备。

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

Logging out of session [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel6guest,
portal: 10.0.0.1,3260
Logout of [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260] successful.
```

iSCSI 设备现在可以用于虚拟化。

12.5.2. 在 virt-manager 中添加 iSCSI 目标

此流程论述了在 virt-manager 中创建带有 iSCSI 目标的存储池。

过程 12.5. 在 virt-manager 中添加 iSCSI 设备

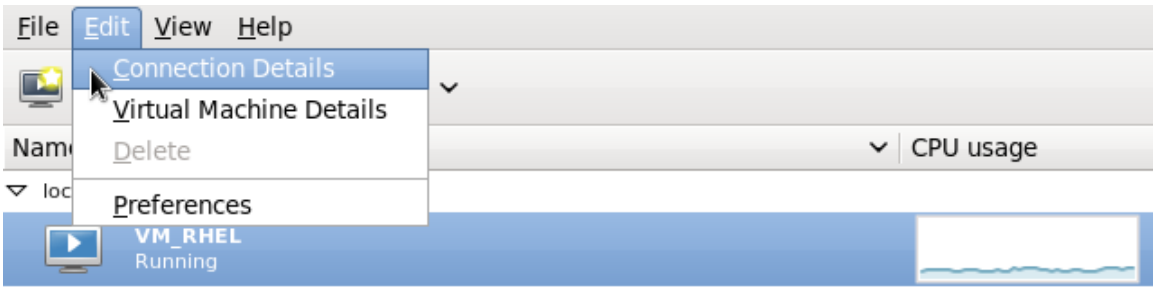
1. *打开主机物理机器的存储标签页*

   在 Connection Details 窗口中，打开 Storage 选项卡。

   a. *打开 virt-manager。*

   b. *从主 virt-manager 窗口中选择主机物理计算机。单击 Edit 菜单，然后选择 Connection Details。*

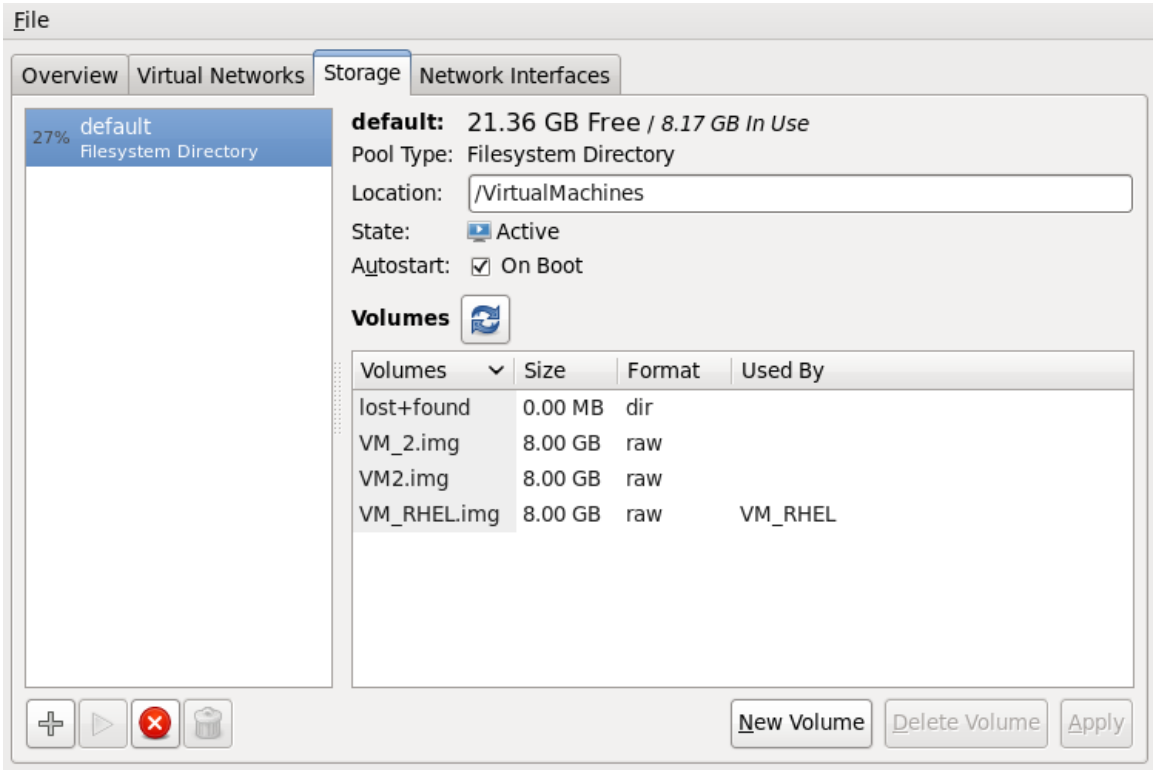
图 12.19. 连接详情



c.

点 Storage 选项卡。

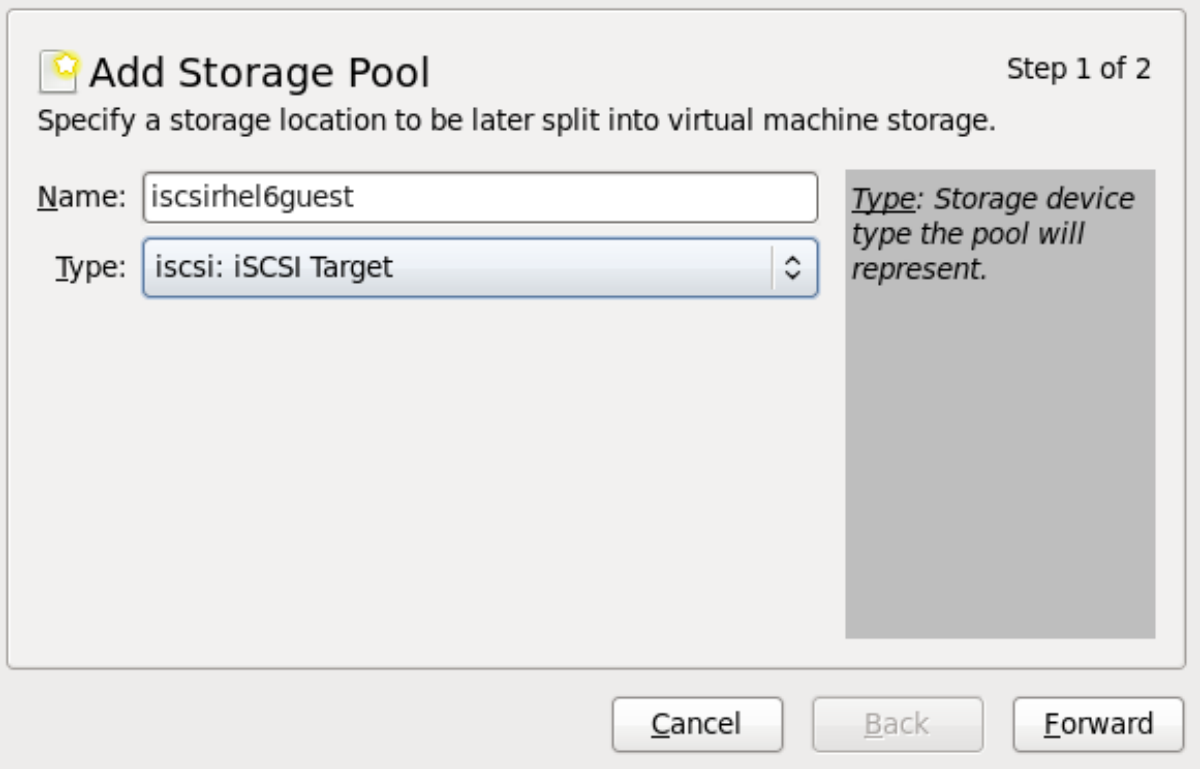
图 12.20. 存储菜单



2. ***添加新池（第 1 部分）***

按 + 按钮（添加池按钮）。此时会出现 Add a New Storage Pool 向导。

图 12.21. 添加 iscsi 存储池名称并键入



为存储池选择一个名称，将"类型"更改为 iscsi，然后按 "下一步" 以继续。

3. *添加新池（第 2 部分）*

   您需要您在 第 12.5 节 “基于 iSCSI 的存储池” 和 过程 12.4, “创建 iSCSI 目标” 中使用的信息完成此菜单中的字段。

   a. *输入 iSCSI 源和目标。Format 选项不作为格式化提供，由客户机虚拟机处理。不建议编辑 目标路径。默认目标路径值 /dev/disk/by-path/ 会向该目录添加驱动器路径。所有主机物理机器上的目标路径应该相同，以迁移。*

   b. *输入 iSCSI 目标的主机名或 IP 地址。这个示例使用 host1.example.com。*

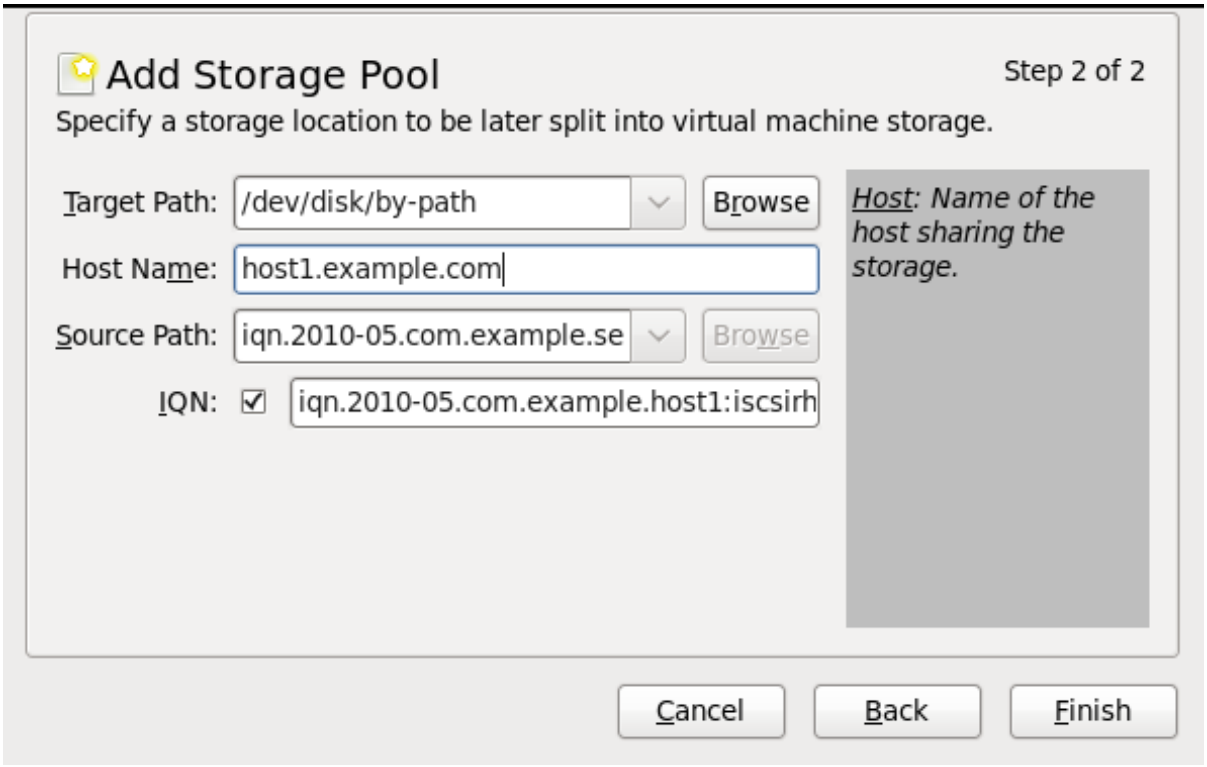
   c. *在 Source Path字段中，输入 iSCSI 目标 IQN。如果您在 过程 12.4, “创建 iSCSI 目标” 中查看 第 12.5 节 “基于 iSCSI 的存储池”，这是您在 /etc/tgt/targets.conf 文件中添加的信息。这个示例使用 iqn.2010-05.com.example.server1:iscsirhel6guest。*

   d. *选中 IQN 复选框，以输入 initiator 的 IQN。这个示例使用 iqn.2010-05.com.example.host1:iscsirhel6。*

e.

单击 **Finish** 以创建新存储池。

图 12.22. 创建 iscsi 存储池



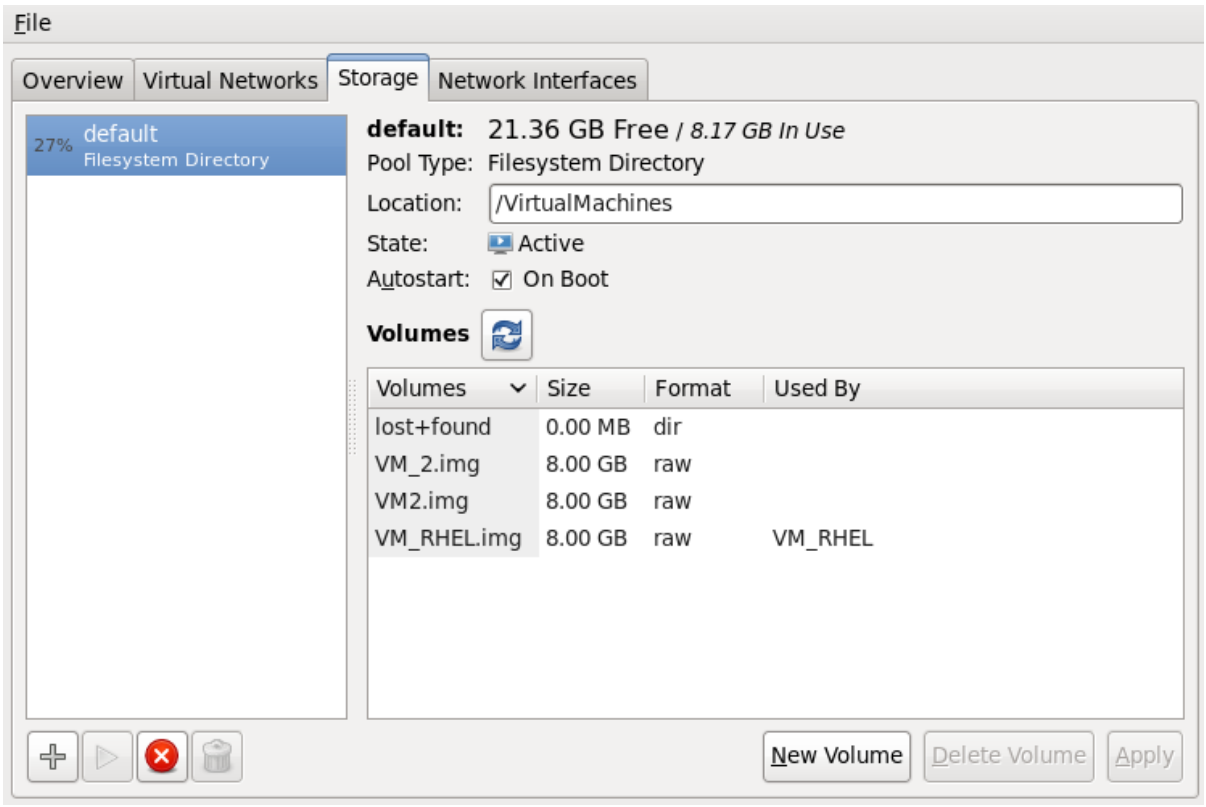
12.5.3. 使用 virt-manager 删除存储池

此流程演示了如何删除存储池。

1.

为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。要做到这一点，选择您要停止的存储池，并点击 **Storage** 窗口底部的红色 **X** 图标。

图 12.23. 停止图标



2. *点 Trash can 图标删除存储池。只有您首先停止存储池时才会启用此图标。*

12.5.4. 使用 virsh 创建基于 iSCSI 的存储池

1. *使用 pool-define-as 从命令行定义池*

   可以使用 virsh 命令行工具创建存储池定义。使用 virsh 创建存储池对系统管理员使用脚本创建多个存储池非常有用。

   virsh pool-define-as 命令具有多个参数，它们以以下格式接受：

   ```
   virsh pool-define-as name type source-host source-path source-dev source-name target
   ```

   这些参数按如下方式解释：

   type

   将此池定义为特定类型的 iscsi，例如

   name

必须是唯一的，并设置存储池的名称

source-host 和 source-path

分别是主机名和 iSCSI IQN

source-dev 和 source-name

基于 iSCSI 的池不需要这些参数，使用 - 字符将字段留空。

目标

定义在主机物理机器上挂载 iSCSI 设备的位置

下面的示例创建了与上一步骤相同的基于 iSCSI 的存储池。

```
#   virsh pool-define-as --name scsirhel6guest --type iscsi \
    --source-host server1.example.com \
    --source-dev iqn.2010-05.com.example.server1:iscsirhel6guest
    --target /dev/disk/by-path
Pool iscsirhel6guest defined
```

2. 验证是否列出了存储池

验证存储池对象是否已正确创建，并且状态报告为 不活动。

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
iscsirhel6guest      inactive   no
```

3. 启动存储池

对此使用 virsh 命令 pool-start。pool-start 启用目录存储池，允许它用于卷和客户机虚拟机。

```
# virsh pool-start guest_images_disk
Pool guest_images_disk started
# virsh pool-list --all
Name                 State      Autostart
```

```
----------------------------------------
default              active     yes
iscsirhel6guest      active     no
```

4. **打开自动启动**

   为存储池打开 **autostart。autostart 将 libvirtd 服务配置为在服务启动时启动存储池。**

```
# virsh pool-autostart iscsirhel6guest
Pool iscsirhel6guest marked as autostarted
```

   验证 **iscsirhel6guest 池是否设置了 autostart:**

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
iscsirhel6guest      active     yes
```

5. **验证存储池配置**

   验证存储池是否已正确创建，报告的大小是否正确，以及状态报告 正在运行。

```
# virsh pool-info iscsirhel6guest
Name:           iscsirhel6guest
UUID:           afcc5367-6770-e151-bcb3-847bc36c5e28
State:          running
Persistent:     unknown
Autostart:      yes
Capacity:       100.31 GB
Allocation:     0.00
Available:      100.31 GB
```

现在提供了基于 iSCSI 的存储池。

12.5.5. 使用 virsh 删除存储池

以下命令演示了如何使用 virsh 删除存储池：

1. 为了避免同一池的其他客户机虚拟机出现任何问题，最好停止存储池并释放其使用中的任何资源。

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

2. 删除存储池的定义

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

12.6. 基于 NFS 的存储池

此流程论述了在 virt-manager 中创建带有 NFS 挂载点的存储池。

12.6.1. 使用 virt-manager 创建基于 NFS 的存储池

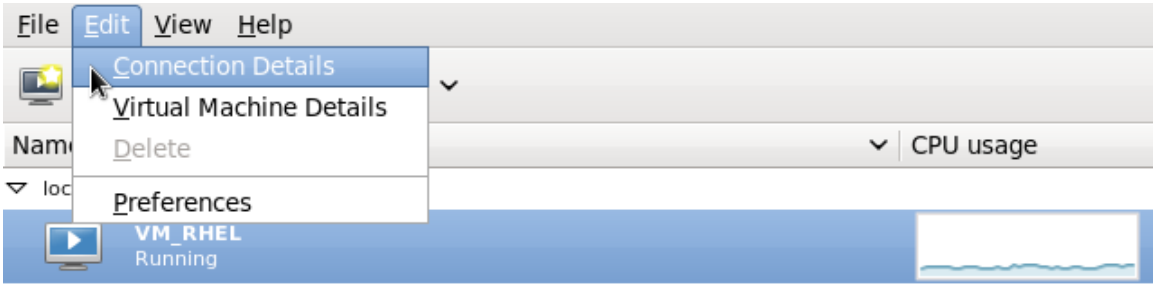
1. 打开主机物理机器的存储标签页

   在 Host Details 窗口中，打开 Storage 选项卡。

   a. 打开 virt-manager。

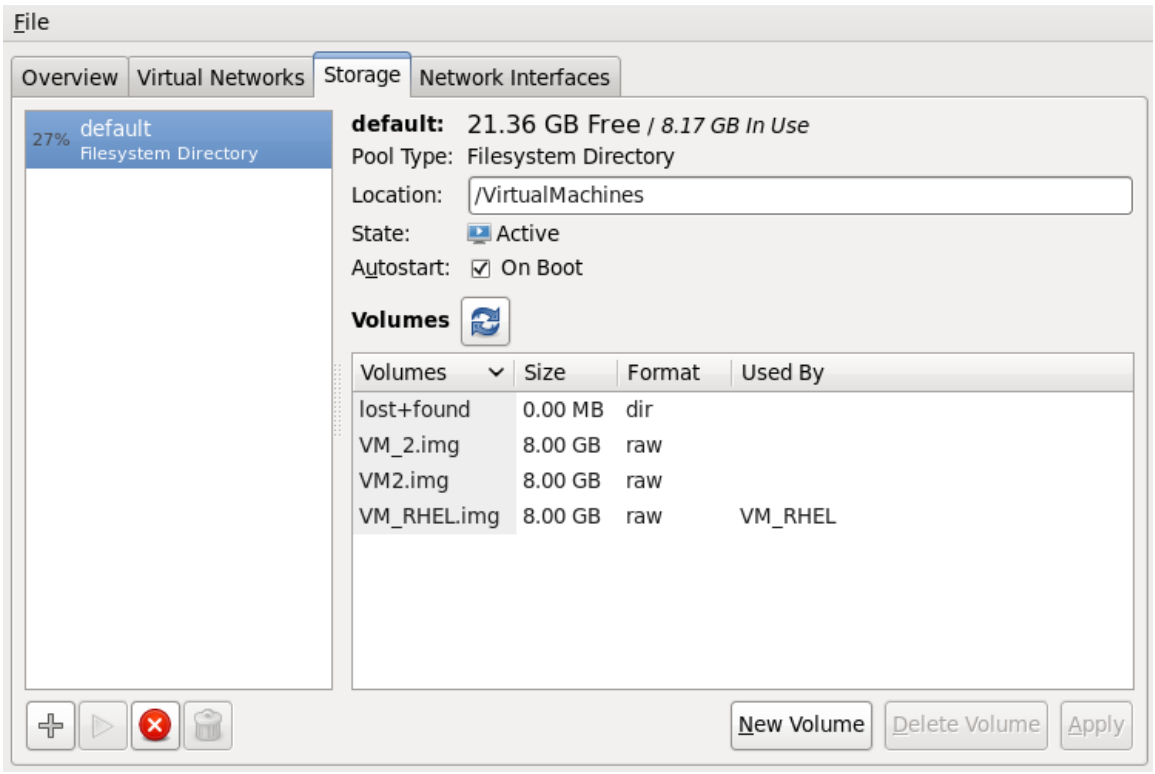
   b. 从主 virt-manager 窗口中选择主机物理计算机。单击 Edit 菜单，然后选择 Connection Details。

   图 12.24. 连接详情



   c. 点 Storage 选项卡。

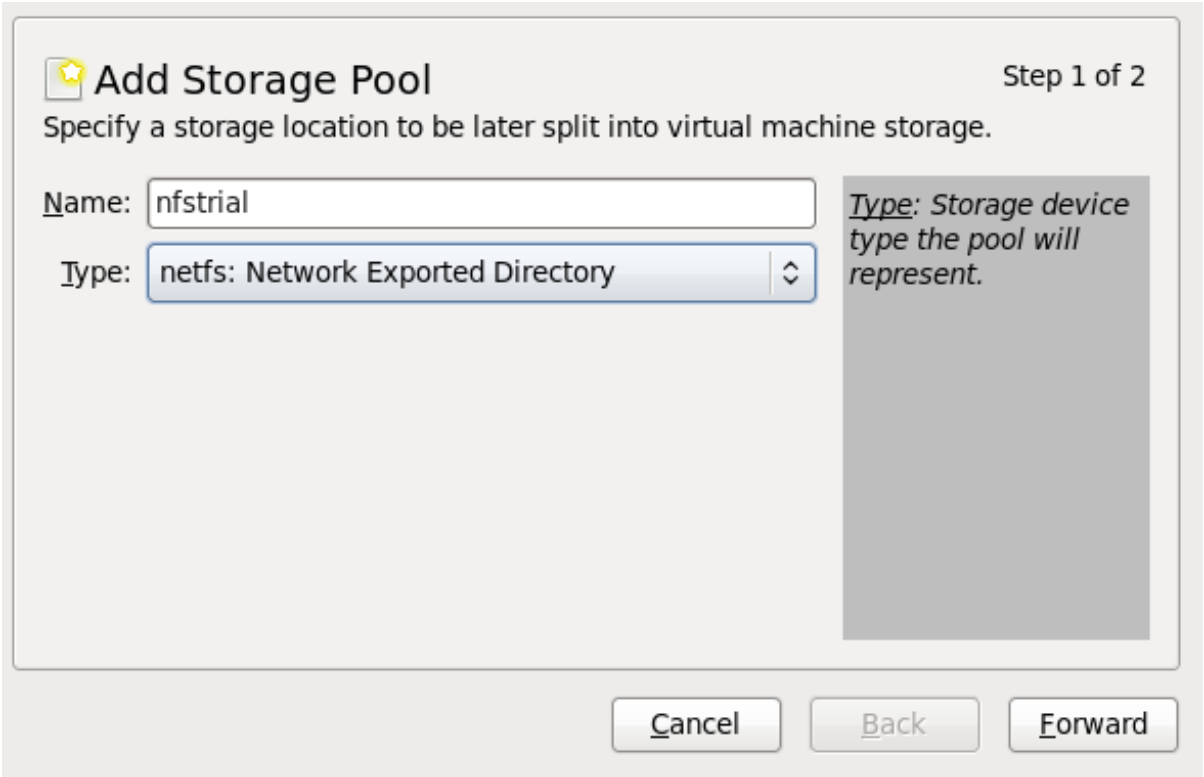
图 12.25. 存储标签



2. *创建新池（第 1 部分）*

   按 + 按钮（添加池按钮）。此时会出现 Add a New Storage Pool 向导。

图 12.26. 添加 NFS 名称和类型



为存储池选择一个名称，然后按 "下一步" 以继续。

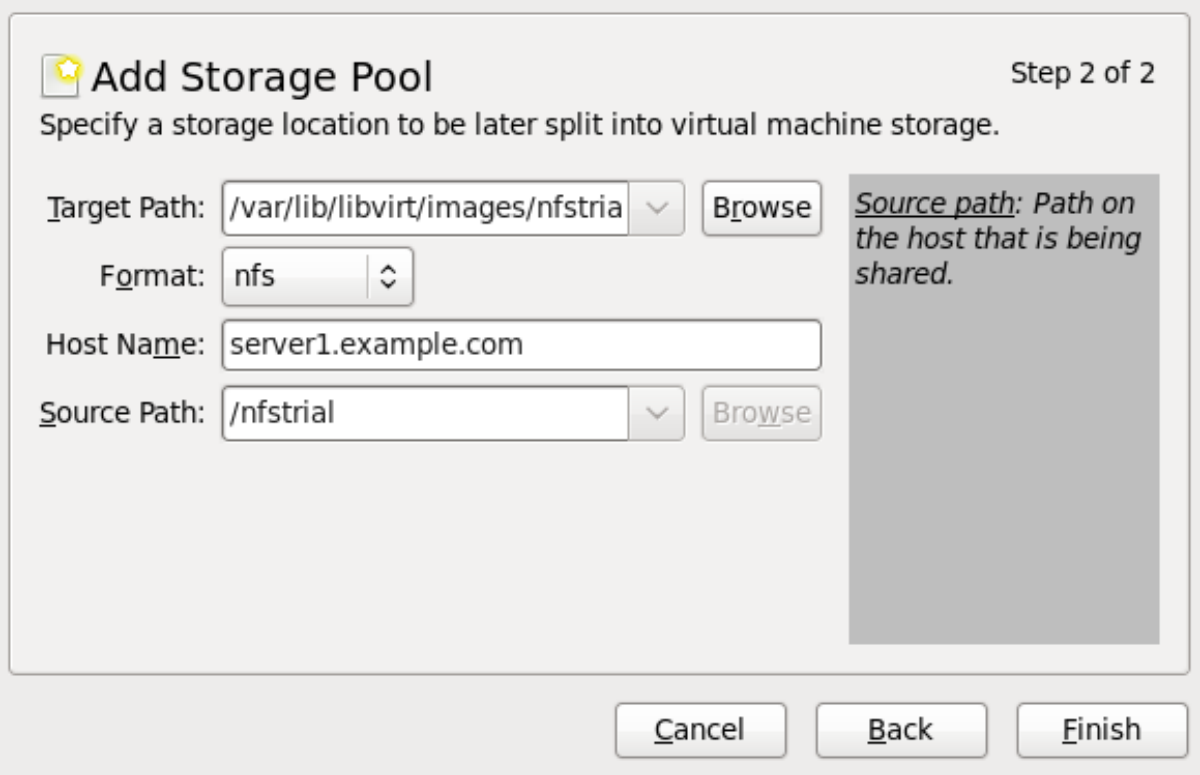
3. *创建新池（第 2 部分）*

输入该设备的目标路径、主机名和 NFS 共享路径。将 Format 选项设置为 NFS 或 auto （以检测类型）。所有主机物理机器上的目标路径必须相同，才能进行迁移。

输入 NFS 服务器的主机名或 IP 地址。这个示例使用 server1.example.com。

输入 NFS 路径。这个示例使用 /nfstrial。

图 12.27. 创建 NFS 存储池



按 Finish 以创建新存储池。

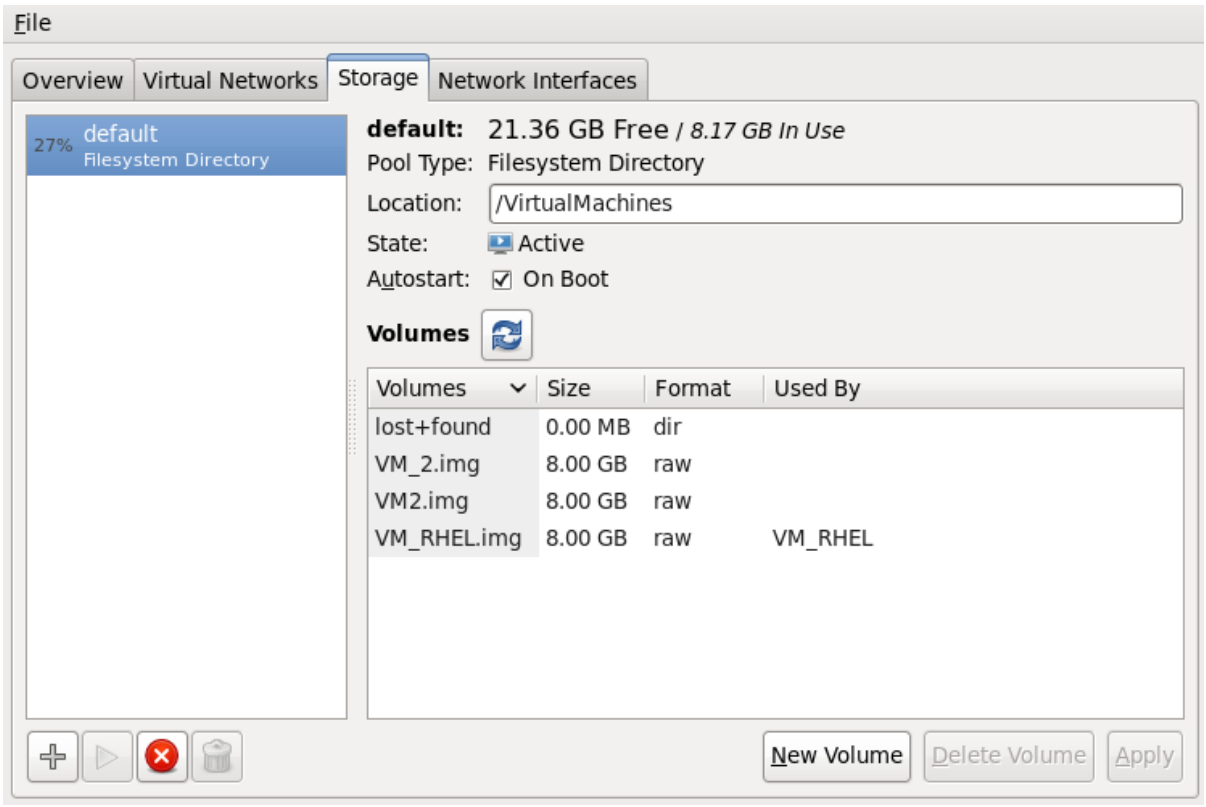
12.6.2. 使用 virt-manager 删除存储池

此流程演示了如何删除存储池。

1.

为避免使用同一池的其他客户机出现任何问题，最好停止存储池并释放其使用中的任何资源。要做到这一点，选择您要停止的存储池，并点击 Storage 窗口底部的红色 X 图标。

图 12.28. 停止图标



2. *点 Trash can 图标删除存储池。只有您首先停止存储池时才会启用此图标。*

12.7. GLUSTERFS 存储池

GlusterFS 是一个使用 FUSE 的用户空间文件系统。当在客户机虚拟机中启用后，KVM 主机物理机器可以从一个或多个 GlusterFS 存储卷引导客户机虚拟机镜像，并使用 GlusterFS 存储卷中的镜像作为客户机虚拟机的数据磁盘。

重要

Red Hat Enterprise Linux 6 不支持将 GlusterFS 与存储池搭配使用。但是，Red Hat Enterprise Linux 6.5 及之后的版本包括对使用 GlusterFS 使用 libgfapi 库创建虚拟机的原生支持。

12.8. 使用带有 SCSI 设备的 NPIV 虚拟适配器(VHBA)

NPIV(N_Port ID Virtualization)是一个软件技术，允许共享单一物理光纤通道主机总线适配器(HBA)。

这允许多个虚拟机从多个物理主机查看相同的存储，从而可以更轻松地进行存储的迁移路径。因此，只

要指定正确的存储路径，不需要迁移来创建或复制存储。

在虚拟化中，虚拟主机总线适配器 （或 vHBA ）控制虚拟机的 LUN。每个 vHBA 都由其自身的 WWNN(World Wide Node Name)和 WWPN(World Wide Port Name)标识。存储的路径由 WWNN 和 WWPN 值决定。

这部分提供了在虚拟机上配置 vHBA 的说明。请注意，Red Hat Enterprise Linux 6 不支持在主机重启后持久性 vHBA 配置；验证主机重启后任何与 vHBA 相关的设置。

12.8.1. 创建 vHBA

过程 12.6. 创建 vHBA

1. 在主机系统中找到 HBA

   要在主机系统中定位 HBA，请检查主机系统中的 SCSI 设备，以查找带有 vport 功能的 scsi_host。

   运行以下命令以检索 scsi_host 列表：

```
# virsh nodedev-list --cap scsi_host
scsi_host0
scsi_host1
scsi_host2
scsi_host3
scsi_host4
```

   对于每个 scsi_host，运行以下命令检查 <capability type='vport_ops'> 行的设备 XML，它表示 scsi_host 具有 vport 功能。

```
# virsh nodedev-dumpxml scsi_hostN
```

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 命令的 XML 输出将列出用于创建 vHBA 的字段 <name>、<wwnn> 和 <wwpn>。<max_vports> 值显示支持的 vHBA 的最大数量。

```
 # virsh nodedev-dumpxml scsi_host3
<device>
  <name>scsi_host3</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
  <parent>pci_0000_10_00_0</parent>
  <capability type='scsi_host'>
    <host>3</host>
    <capability 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> 值显示在 HBA 配置中可以使用总计 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 主机创建一个名为 vhba_host3.xml的 XML 文件。

```
# cat vhba_host3.xml
  <device>
    <parent>scsi_host3</parent>
    <capability type='scsi_host'>
      <capability type='fc_host'>
      </capability>
    </capability>
  </device>
```

<parent> 字段指定要与这个 vHBA 设备关联的 HBA 设备。 <device> 标签的详情在下一步中使用为主机创建新 vHBA 设备。有关 http://libvirt.org/formatnode.html nodedev XML 格式的更多信息，请参阅。

4. *在 vHBA 主机设备中创建一个新的 vHBA*

   要在 vhba_host3 上创建 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>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
  </capability>
</device>
```

12.8.2. 使用 vHBA 创建存储池

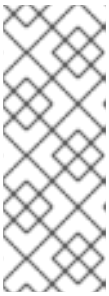
建议您基于 vHBA 定义 libvirt 存储池来保留 vHBA 配置。

使用存储池有两个主要优点：

- libvirt 代码可以使用 virsh 命令输出轻松查找 LUN 的路径，并
- 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，在虚拟机的 XML 配置中必须指定 vHBA LUN、libvirt LUN、libvirt 存储池和卷名称。如需示例，请参阅 第 12.8.3 节 “将虚拟机配置为使用 vHBA LUN”。

1. 创建 SCSI 存储池

要创建 vHBA 配置，首先请创建一个 libvirt 'scsi' 存储池 XML 文件，该文件基于 vHBA 使用以下格式。



注意

确定使用在 过程 12.6, “创建 vHBA” 中创建的 vHBA 作为主机名，修改 vHBA 名称 scsi_ hostN 为存储池配置。在本例中，vHBA 名为 scsi_host5，它在 Red Hat Enterprise Linux 6 libvirt 存储池中指定为 &lt ;adapter name='host5' />。

建议您为 <path> 值使用稳定位置，如系统中的 /dev/disk/by-{path|id|uuid|label} 位置。有关 <path> 和 <target> 中的元素的更多信息，请访问 http://libvirt.org/formatstorage.html。

在这个示例中，'scsi' 存储池名为 vhbapool_host3.xml ：

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <uuid>e9392370-2917-565e-692b-d057f46512d6</uuid>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <adapter name='host5'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0700</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>
```

2. 定义池

要定义存储池（在这个示例中名为 vhbapool_host3 ），请使用 virsh pool-define 命令：

```
# virsh pool-define vhbapool_host3.xml
Pool vhbapool_host3 defined from vhbapool_host3.xml
```

3. 启动池

使用以下命令启动存储池：

```
# virsh pool-start vhbapool_host3
Pool vhbapool_host3 started
```

4. 启用自动启动

最后，要确保后续主机重启将自动定义虚拟机中使用的 vHBA，设置存储池自动启动功能（在本例中，对于名为 vhbapool_host3的池）：

```
# virsh pool-autostart vhbapool_host3
```

12.8.3. 将虚拟机配置为使用 vHBA LUN

为 vHBA 创建存储池后，将 vHBA LUN 添加到虚拟机配置中。

1. 查找可用的 LUN

   首先，使用 virsh vol-list 命令在 vHBA 上生成可用 LUN 的列表。例如：

```
# virsh vol-list vhbapool_host3
 Name                 Path
-----------------------------------------------------------------------------
 unit:0:4:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016844602198-lun-0
 unit:0:5:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0
```

   显示的 LUN 名称列表将用作虚拟机配置中的磁盘卷。

2. 在虚拟机中添加 vHBA LUN

   通过在虚拟机的 XML 中指定，将 vHBA LUN 添加到虚拟机 ：

   - 在 lun 参数中作为 disk 或 <disk> 设备类型，以及
   - <source> 参数中的源设备。请注意，这可作为 /dev/sdaN 输入，或者作为 /dev/disk/by-path|by-id|by-uuid|by-uuid|by-label 命令生成的符号链接，该文件可通过运行 virsh vol-list 池 命令找到。

   例如：

```
<disk type='block' device='lun'>
  <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'/>
</disk>
```

12.8.4. 销毁 vHBA 存储池

virsh pool-destroy 命令可以销毁 vHBA 存储池：

```
# virsh pool-destroy vhbapool_host3
```

使用以下命令删除 vHBA

```
# virsh nodedev-destroy scsi_host5
```

要验证池和 vHBA 已被销毁，请运行：

```
# virsh nodedev-list --cap scsi_host
```

scsi_host5 将不再显示在结果列表中。

第 13 章 卷

13.1. 创建卷

本节演示了如何在基于块的存储池中创建磁盘卷。在以下示例中，**virsh vol-create-as** 命令在 **guest_images_disk** 存储池中创建一个具有特定大小的存储卷。因为这个命令会根据卷重复，因此会创建三个卷，如示例所示。

```
# 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              /dev/sdb1
volume2              /dev/sdb2
volume3              /dev/sdb3

# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start   End     Size    File system  Name     Flags
2      17.4kB  8590MB  8590MB               primary
3      8590MB  17.2GB  8590MB               primary
1      21.5GB  30.1GB  8590MB               primary
```

13.2. 克隆卷

新卷将从与要克隆的卷相同的存储池中分配。**virsh vol-clone** 必须具有 **--pool** 参数，该参数指明要克隆包含卷的存储池的名称。命令的其余部分命名要克隆的卷(**volume3**)和已克隆的新卷的名称(**clone1**)。**virsh vol-list** 命令列出存储池中存在的卷(**guest_images_disk**)。

```
# virsh vol-clone --pool guest_images_disk volume3 clone1
Vol clone1 cloned from volume3

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
volume1              /dev/sdb1
```

```
volume2             /dev/sdb2
volume3             /dev/sdb3
clone1             /dev/sdb4


# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    File system  Name     Flags
1      4211MB  12.8GB  8595MB  primary
2      12.8GB  21.4GB  8595MB  primary
3      21.4GB  30.0GB  8595MB  primary
4      30.0GB  38.6GB  8595MB  primary
```

13.3. 在客户机中添加存储设备

本节介绍在客户机中添加存储设备。可以根据需要添加额外的存储。

13.3.1. 在 客户机中添加基于文件的存储

基于文件的存储是保存在主机物理机器文件系统中的文件集合，充当客户机的虚拟化硬盘驱动器。要添加基于文件的存储，请执行以下步骤：

过程 13.1. 添加基于文件的存储

1. 创建存储文件或使用现有文件（如 IMG 文件）。请注意，以下两个命令都创建一个 4GB 文件，该文件可用作客户机的额外存储：

   - 建议为基于文件的存储镜像使用预分配文件。使用以下 dd 命令创建预分配文件：

     ```
     # dd if=/dev/zero of=/var/lib/libvirt/images/FileName.img bs=1M count=4096
     ```

   - 或者，创建稀疏文件，而不是预分配的文件。稀疏文件创建速度更快，并可用于测试，但由于数据完整性和性能问题，不建议在生产环境中使用。

     ```
     # dd if=/dev/zero of=/var/lib/libvirt/images/FileName.img bs=1M seek=4096 count=0
     ```

2. 通过在新文件中写入 < disk> 元素来创建额外的存储。在本例中，该文件将被称为

NewStorage.xml。

< disk > 元素描述了磁盘源，以及虚拟块设备的设备名称。设备名称应该在客户机中所有设备间唯一，并确定客户机将在其上查找虚拟块设备的总线。以下示例定义了 virtio 块设备，其源是一个基于文件的存储容器，名为 FileName.img ：

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

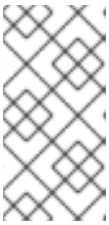
设备名称也以"hd"或"sd"开头，分别标识 IDE 和 SCSI 磁盘。配置文件也可以包含 < address& gt; 子元素，用于指定新设备的总线上的位置。如果是 virtio 块设备，则应该是 PCI 地址。省略 & lt;address > 子元素可让 libvirt 查找并分配下一个可用的 PCI 插槽。

3. *按如下方式附加 CD-ROM ：*

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

4. *使用您的 guest(Guest1)添加 NewStorage.xml 中定义的设备：*

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



注意

这个更改只有在客户机被销毁并重启后才会应用。另外，永久性设备只能添加到持久域中，这是使用 virsh define 命令保存的配置的域。

如果客户机正在运行，并且您希望临时添加新设备，直到销毁客户端为止，省略 --config 选项：

```
# virsh attach-device Guest1 ~/NewStorage.xml
```

注意

virsh 命令允许 attach-disk 命令使用更简单的语法来设置有限数量的参数，而无需创建 XML 文件。attach-disk 命令使用与前面提到的 attach-device 命令类似的方法，如下所示：

```
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img vdb --cache
none --driver qemu --subdriver raw
```

请注意，virsh attach-disk 命令也接受 --config 选项。

5. 启动 **guest 机器（如果当前尚未运行）：**

```
# virsh start Guest1
```

注意

以下步骤特定于 Linux 客户机。其他操作系统以不同的方式处理新存储设备。有关其他系统，请参考该操作系统的文档。

6. *对磁盘驱动器进行分区*

现在，客户机具有名为 /dev/vdb 的硬盘设备。如果需要，对这个磁盘驱动器进行分区并格式化分区。如果没有看到添加的设备，这表示您的客户端操作系统中存在磁盘热插问题。

a. 为**新设备启动 fdisk ：**

```
# fdisk /dev/vdb
Command (m for help):
```

b. 为**新分区输入 n。**

c. **此时会出现以下内容：**

```
Command action
e   extended
p   primary partition (1-4)
```

为主分区输入 **p**。

d.

选择可用分区号。在这个示例中，通过输入 **1** 来选择第一个分区。

```
Partition number (1-4): 1
```

e.

按 **Enter** 输入默认柱面。

```
First cylinder (1-400, default 1):
```

f.

选择分区的大小。在本例中，通过按 **Enter** 来分配整个磁盘。

```
Last cylinder or +size or +sizeM or +sizeK (2-400, default 400):
```

g.

输入 **t** 来配置分区类型。

```
Command (m for help): t
```

h.

选择您在前面的步骤中创建的分区。在这个示例中，分区号是 **1**，因为只有一个分区创建，**fdisk** 会自动选择分区 **1**。

```
Partition number (1-4): 1
```

i.

为 **Linux** 分区输入 **83**。

```
Hex code (type L to list codes): 83
```

j.

输入 **w** 写入更改并退出。

```
Command (m for help): w
```

k.

将新分区格式化为 **ext3** 文件系统。

```
# mke2fs -j /dev/vdb1
```

7. 创建挂载目录，并在客户端上挂载磁盘。在本例中，目录位于 **myfiles**。

```
# mkdir /myfiles
# mount /dev/vdb1 /myfiles
```

客户机现在有一个额外的基于文件的虚拟化存储设备。但请注意，除非在 **guest** 的 **/etc/fstab** 文件中定义，否则此存储不会在系统重启后持久挂载：

```
/dev/vdb1    /myfiles    ext3     defaults    0 0
```

13.3.2. 在客户机中添加硬盘和其他块设备

系统管理员可以选择使用其他硬盘驱动器来增加客户机的存储空间，或者将系统数据与用户数据分开。

过程 **13.2.** 在客户机中添加物理块设备

1. 这个步骤描述了如何在主机物理机器中添加硬盘。它适用于所有物理块设备，包括 **CD-ROM**、**DVD** 和软盘设备。

   将硬盘设备物理附加到主机物理机器。如果默认无法访问驱动器，请配置主机物理机器。

2. 执行以下操作之一：

   a. 通过在新文件中 写入磁盘 元素来创建额外的存储。在本例中，该文件将被称为 **NewStorage.xml**。以下示例是配置文件部分，其中包含主机物理机器分区 **/dev/sr0** 的额外基于设备的存储容器：

   ```
   <disk type='block' device='disk'>
       <driver name='qemu' type='raw' cache='none'/>
       <source dev='/dev/sr0'/>
       <target dev='vdc' bus='virtio'/>
   </disk>
   ```

   b. 按照上一节中的 指令，将设备连接到 **guest** 虚拟机。另外，您可以使用 **virsh attach-disk** 命令，如下所示：

   ```
   # virsh attach-disk Guest1 /dev/sr0 vdc
   ```

-

请注意，以下选项可用：

- virsh attach-disk 命令也接受 --config、--type 和 --mode 选项，如下所示：

```
# virsh attach-disk Guest1 /dev/sr0 vdc --config --type cdrom --mode readonly
```

- 另外， --type 在设备是硬盘时接受 --type disk。

3. 现在，客户机虚拟机在 Linux 上有一个名为 /dev/vdc 的新硬盘设备（或者类似它，具体取决于虚拟机操作系统选择的内容）或 D: 驱动器 （例如 Windows）。现在，您可以按照客户端虚拟机的标准步骤从客户端虚拟机初始化磁盘。如需示例，请参阅 过程 13.1, “添加基于文件的存储”。

警告

向客户机添加块设备时，请务必遵循安全性注意事项。Red Hat Enterprise Linux 虚拟化安全指南 中会更加详细地探讨此信息，网址为：https://access.redhat.com/site/documentation/

重要

不应该向客户机虚拟机提供对整个磁盘或块设备的写入权限（例如：/dev/sdb）。具有访问整个块设备的虚拟客户机可能需要修改卷标签，这可用于破坏主机物理机器系统。使用分区（例如 /dev/sdb1）或 LVM 卷来防止此问题。

13.4. 删除和删除卷

本节介绍如何使用 virsh vol-delete 命令从基于块的存储池中删除数据卷。在本例中，卷是 卷 1，存储池是 guest_images。

```
# virsh vol-delete --pool guest_images volume1
Vol volume1 deleted
```

第 14 章 使用 VIRSH 管理 GUEST 虚拟机

virsh 是用于管理 guest 虚拟机和 hypervisor 的命令行工具。virsh 命令行工具在 libvirt 管理 API 上构建，并作为 qemu-kvm 命令和图形 virt-manager 应用程序的替代操作。virsh 命令可在只读模式下供非特权用户使用，也可以使用 root 访问权限、完整的管理功能。virsh 命令是编写虚拟化管理脚本的理想选择。

14.1. 通用命令

本节中的命令是通用的，因为它们不特定于任何域。

14.1.1. 帮助

$ virsh help [command|group] help 命令可与 或不使用选项一起使用。当在没有选项的情况下使用时，将列出所有命令，每行一个。与选项一起使用时，它将分组为不同的类别，显示每个组的关键字。

要显示只针对特定选项的命令，您需要为该组提供关键字作为选项。例如：

```
$ virsh help pool
 Storage Pool (help keyword 'pool'):
    find-storage-pool-sources-as   find potential storage pool sources
    find-storage-pool-sources      discover potential storage pool sources
    pool-autostart              autostart a pool
    pool-build                  build a pool
    pool-create-as              create a pool from a set of args
    pool-create                 create a pool from an XML file
    pool-define-as              define a pool from a set of args
    pool-define                 define (but don't start) a pool from an XML file
    pool-delete                 delete a pool
    pool-destroy                destroy (stop) a pool
    pool-dumpxml                pool information in XML
    pool-edit                   edit XML configuration for a storage pool
    pool-info                   storage pool information
    pool-list                   list pools
    pool-name                   convert a pool UUID to pool name
    pool-refresh                refresh a pool
    pool-start                  start a (previously defined) inactive pool
    pool-undefine               undefine an inactive pool
    pool-uuid                   convert a pool name to pool UUID
```

使用与命令选项相同的命令，提供该特定命令的帮助信息。例如：

```
$ virsh help vol-path
  NAME
```

```
  vol-path - returns the volume path for a given volume name or key

SYNOPSIS
  vol-path <vol> [--pool <string>]

OPTIONS
  [--vol] <string>  volume name or key
  --pool <string>  pool name or uuid
```

14.1.2. 退出并退出

quit 命令和 exit 命令将关闭终端。例如：

```
$ virsh exit
```

```
$ virsh quit
```

14.1.3. version

version 命令显示当前 libvirt 版本，并显示有关构建位置的信息。例如：

```
$ virsh version
Compiled against library: libvirt 1.1.1
Using library: libvirt 1.1.1
Using API: QEMU 1.1.1
Running hypervisor: QEMU 1.5.3
```

14.1.4. 参数显示

virsh echo [--shell][--xml][arg] 命令回显或显示指定的参数。调用的每个参数都将一个空格分开。通过使用 --shell 选项，输出将根据需要进行单引号，以便其适合在 shell 命令中重复使用。如果使用 --xml 选项，则输出将适合用于 XML 文件。例如，命令 virsh echo --shell "hello world" 将发送输出 'hello world'。

14.1.5. connect

连接到管理程序会话.当 shell 首次启动时，该命令会在 -c 命令请求 URI 参数时自动运行。URI 指定如何连接到虚拟机监控程序。最常用的 URI 是：

- **Xen:/// - 连接到本地 Xen 管理程序.**

- QEMU :///system - 以 root 身份从本地连接守护进程，可打开 QEMU 和 KVM 域。
- Xen:///session - 以用户身份本地连接到用户一组 QEMU 和 KVM 域。
- lxc:/// - 连接到本地 Linux 容器。

libvirt 的网站 http://libvirt.org/uri.html 提供了其他值。

该命令可以按如下方式运行：

```
$ virsh connect {name|URI}
```

其中 {name} 是 hypervisor 的机器名称（主机名）或 URL（ virsh uri 命令的输出）。要启动只读连接，请使用 --readonly 附加上述命令。如需有关 URI 的更多信息，请参阅 远程 URI。如果您不确定 URI，则 virsh uri 命令将显示它：

```
$ virsh uri
qemu:///session
```

14.1.6. 显示基本信息

以下命令可以用来显示基本信息：

- $ hostname - 显示虚拟机监控程序的主机名
- $ sysinfo - 显示管理程序系统信息的 XML 表（如果可用）

14.1.7. 注入 NMI

$ virsh inject-nmi [domain] 将 NMI（不可屏蔽中断）信息注入给客户机虚拟机。这在响应时间至关重要时使用，如不可恢复的硬件错误。要运行这个命令：

```
$ virsh inject-nmi guest-1
```

14.2. 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 13.3.1 节 “在 客户机中添加基于文件的存储”

过程 14.1. 热插 USB 设备供客户端虚拟机使用

以下步骤演示了如何将 USB 设备连接到客户端虚拟机。当客户机虚拟机作为热插拔程序运行时，可以完成此操作，也可以在客户端关闭时完成。要模拟的设备需要附加到主机物理机器。

1. 使用以下命令找到您要连接的 USB 设备：

```
# lsusb -v

idVendor           0x17ef Lenovo
idProduct          0x480f Integrated Webcam [R5U877]
```

2. 创建一个 XML 文件，并为它指定逻辑名称（例如usb_device.xml ）。请确定您复制厂商和产品 ID，如搜索中所示。

图 14.1. USB 设备 XML 片段

```
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x17ef'/>
    <product id='0x480f'/>
  </source>
</hostdev>
...
```

3. 使用以下命令附加该设备：

```
# virsh attach-device rhel6 --file usb_device.xml --config
```

在本例中，[rhel6] 是客户机虚拟机的名称，[usb_device.xml] 是您在上一步中创建的文件。如果要使更改在下次重启时生效，请使用 --config 选项。如果您希望此更改具有持久性，请使用 --persistent 选项。如果您希望更改对当前域生效，请使用 --current 选项。详情请查看 Virsh man page。

4.

如果要分离设备(hot unplug)，请执行以下命令：

```
# virsh detach-device rhel6 --file usb_device.xml
```

在这个示例中 [rhel6] 是客户机虚拟机的名称，[usb_device.xml] 是您在上一步中附加的文件

14.3. 附加接口设备

virsh attach-interface 域类型 source 命令可使用以下选项：

- --live - 从正在运行的域中获取值
- --config - 获取在下次引导时使用的值
- --current - 根据当前域状态获取值
- --persistent - 行为类似于 --config 表示离线域，如 --live 用于正在运行的域。
- --target - 表示客户端虚拟机中的目标设备。
- -- MAC - 使用它来指定网络接口的 MAC 地址
- --script - 使用它来指定脚本文件处理网桥的路径，而不是默认路径。
- --model - 使用它指定模型类型。
- --inbound - 控制接口的入站带宽。可接受的值为 、 peak 和 burst。
- --outbound - 控制接口的出站带宽。可接受的值为 、 peak 和 burst。

类型可以是 网络 （表示物理网络设备），或 网桥 指明到设备的网桥。source 是该设备的源。要删除连接的设备，请使用 virsh detach-device。

14.4. 更改 CDROM 的介质

将 CDROM 介质改为其他源或格式

```
# change-media domain path source --eject --insert --update --current --live --config --force
```

- --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 - Eject the media
- --insert - Insert the media
- --update - 更新介质
- --current - 可以是 --live 和 --config，它取决于虚拟机监控程序驱动程序的实现
- --live - 更改运行域的实时配置
- --config - 更改持久配置，在下一次引导时观察到效果
- --force - 强制媒体更改

14.5. 域命令

大多数命令都需要一个域名，因为它们直接操作指定的域。域可以指定为简短整数(0,1,2....)，一个名称或完整 UUID。

14.5.1. 将域配置为在引导时自动启动

$ virsh autostart [--disable] 域 将在启动时自动启动指定的域。使用 **--disable** 选项可禁用自动启动。

```
# virsh autostart rhel6
```

在上例中，**rhel6** 客户机虚拟机将在主机物理机引导时自动启动

```
# virsh autostart rhel6 --disable
```

在上面的示例中，自动启动功能被禁用，在主机物理机引导时 **guest** 虚拟机将不再自动启动。

14.5.2. 为 guest 虚拟机连接 Serial Console

$ virsh console <domain> [--devname <string>] [--force] [--safe] 命令连接客户机虚拟机的虚拟串行控制台。可选的 **--devname <string>** 参数是指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，则会打开主控制台。**force** 选项将强制控制台连接，或者与断开连接一起使用时，将断开连接。使用 **--safe** 选项将只允许客户端连接是否支持安全控制台处理。

```
$ virsh console virtual_machine --safe
```

14.5.3. 使用 XML 文件定义域

定义 **<FILE >** 命令从 **XML** 文件定义域。在这种情况下，域定义已注册但未启动。如果域已在运行，则更改将在下次启动时生效。

14.5.4. 编辑和显示域的描述和标题

以下命令用于显示或修改域的描述和标题，但不配置它：

```
# virsh desc [domain-name] [[--live] [--config] | [--current]] [--title] [--edit] [--new-desc New description or title message]
```

这些值是用户字段，允许存储任意文本数据以便易于识别域。理想情况下，标题应该很短，尽管 **libvirt** 不强制实施。

options --live 或 **--config** 选择此命令是否可以用于域的实时或永久定义。如果指定了 **--live** 和 **--**

config，则会首先实施 --config 选项，其中在命令中输入的描述成为新的配置设置，该设置应用于实时配置和持久配置设置。current 选项将修改或获取当前状态配置，并且不具有持久性。如果未指定 --live 和 --config，则 --current 选项将使用 --current。edit 选项指定包含当前描述或标题内容的编辑器应打开，之后再保存的内容。使用 --title 选项将显示或修改域的标题字段，而不包括其描述。另外，如果命令中没有使用 --edit 和 --new-desc，则仅显示描述且无法修改。

例如，以下命令可将 guest 虚拟机的标题从 testvm 更改为 TestVM-4F，并将描述更改为 guest 虚拟机在第四四个阶段 ：

```
$ virsh desc testvm --current --title TestVM-4F --new-desc Guest VM on fourth floor
```

14.5.5. 显示设备块统计信息

此命令将显示正在运行的域的块统计信息。您需要具有域名和设备名称（使用 virsh domblklist 列出设备）。在这种情况下，块设备是唯一目标名称(<target dev='name'/>)或源文件(< source file ='name'/>)。请注意，并非每个虚拟机监控程序都可以显示每个字段。要确保输出在最方便的表单中显示，请使用 --human 选项，如下所示：

```
# virsh domblklist rhel6
Target     Source
------------------------------------------------
vda        /VirtualMachines/rhel6.img
hdc        -

# virsh domblkstat --human rhel6 vda
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
```

14.5.6. 检索网络统计信息

domnetstat [domain][interface-device] 命令显示给定域上运行的指定设备的网络接口统计信息。

```
# domifstat rhel6 eth0
```

14.5.7. 修改域虚拟接口的链路状态

以下命令可以将指定的接口配置为 up 或 down：

```
# domif-setlink [domain][interface-device][state]{--config}
```

使用此项可修改指定域指定接口的状态。请注意，如果您只想修改域的持久配置，您需要使用 --config选项。另请注意，出于兼容性的原因，-- persistent 是 --config 的别名。"接口设备"可以是接口的目标名称或 MAC 地址。

```
# domif-setlink rhel6 eth0 up
```

14.5.8. 列出域虚拟接口的链路状态

此命令可用于查询给定域中指定接口的状态。请注意，如果您只想修改域的持久配置，您需要使用 --config选项。另请注意，出于兼容性的原因，-- persistent 是 --config 的别名。"接口设备"可以是接口的目标名称或 MAC 地址。

```
# domif-getlink rhel6 eth0 up
```

14.5.9. 设置网络接口带宽参数

domiftune 设置 guest 虚拟机的网络接口带宽参数。使用以下格式：

```
#virsh domiftune domain interface-device [[--config] [--live] | [--current]] [--inbound
average,peak,burst] [--outbound average,peak,burst]
```

唯一需要的参数是客户机虚拟机的域名和接口设备、--config、--live 和 --current 功能与 第 14.19 节 “设置调度参数” 相同。如果没有指定限制，它将查询当前的网络接口设置。否则，使用以下选项更改限制：

- <interface-device> 这是强制性的，它将设置或查询域的带宽参数。interface-device 可以是接口的目标名称(<target dev='name'/>)，也可以是 MAC 地址。
- 如果没有指定 --inbound 或 --outbound，此命令将查询并显示带宽设置。否则，它将设置入站或出站带宽。平均，peak，burst 与 attach-interface 命令中的相同。请参阅 第 14.3 节 “附加接口设备”

14.5.10. 检索正在运行的域的内存统计信息

这个命令可能会返回变体结果，具体取决于您使用的虚拟机监控程序。

dommemstat [domain] [--period(sec)][[--config][-live]|[--current]] 显示正在运行的域的内存统计信息。使用 --period 选项需要以秒为单位。将此选项设置为大于 0 的值将允许 balloon 驱动程序返回之后的 domemstat 命令将显示的其他统计信息。将 --period 选项设置为 0，将停止 balloon 驱动程序集合，但不会清除 balloon 驱动程序中的统计数据。您不能在不设置 --period 选项的情况下使用 --live、--config 或 --current 选项来设置 balloon 驱动程序的收集周期。如果指定了 --live 选项，则只有正在运行的客户端的集合周期会受到影响。如果使用 --config 选项，它将影响下一次持久 guest 的启动。如果使用 --current 选项，它将影响当前的客户机状态

可以使用 --live 和 --config 选项，但 --current 是独占的。如果没有指定选项，其行为将因客户机的状态而异。

```
#virsh domemstat rhel6 --current
```

14.5.11. 在块设备中显示错误

该命令最好按照 domstate 报告因为 I/O 错误而暂停域。domblkerror 域 显示给定域上处于错误状态的所有块设备，它会显示设备正在报告的错误消息。

```
# virsh domblkerror rhel6
```

14.5.12. 显示块设备大小

在这种情况下，块设备是唯一的目标名称(<target dev='name'/>)或源文件(< source file ='name'/>)。若要检索列表，您可以运行 domblklist。这个 domblkinfo 需要 域名。

```
# virsh domblkinfo rhel6
```

14.5.13. 显示与某个域关联的块设备

domblklist domain --inactive --details 显示所有与指定域关联的块设备表。

如果指定了 --inactive，则结果将显示在下次启动时要使用的设备，且不会显示当前运行的域正在使用的设备。如果指定了 --details，则磁盘类型和设备值将包含在表中。此表中显示的信息可与 domblkinfo 和 snapshot-create 一起使用。

```
#domblklist rhel6 --details
```

14.5.14. 显示与某个域关联的虚拟接口

运行 domiflist 命令会生成表，显示与指定域关联的所有虚拟接口的信息。domiflist 需要 域名，并可

选择性地使用 --inactive 选项。

如果指定了 --inactive，则结果将显示在下次启动时要使用的设备，且不会显示当前运行的域正在使用的设备。

需要虚拟接口的 MAC 地址的命令（如 detach-interface 或 domif-setlink）将接受此命令显示的输出。

14.5.15. 使用 blockcommit 短性链

本节介绍如何使用 virsh blockcommit 缩短后备链。有关后备链的更多背景信息，请参阅 第 14.5.18 节 “使用实时块复制进行磁盘镜像管理”。

blockcommit 将链中的一个部分的数据复制到支持文件中，从而可以放弃链的其余部分，从而绕过提交的部分。例如，假设这是当前状态：

```
base ← snap1 ← snap2 ← active.
```

使用 blockcommit 将 snap2 的内容移动到 snap1，以便您从链中删除 snap2，从而加快备份的速度。

过程 14.2. virsh blockcommit

- 运行以下命令：

```
# virsh blockcommit $dom $disk -base snap1 -top snap2 -wait -verbose
```

snap2 的内容将移到 snap1 中，导致：

base MOTD snap1 InventoryService active.Snap2 不再有效，可以删除

警告

blockcommit 将破坏依赖于 -base 选项的任何文件（除了依赖于 -top 选项的文件，因为这些文件现在指向这个基础）。要防止这种情况，请不要将更改提交到多个虚拟客户机共享的文件。-verbose 选项允许在屏幕中打印进度。

14.5.16. 使用 blockpull 进行反转链

blockpull 可以在以下应用程序中使用：

- *通过填充其后备映像链中的数据来扁平化镜像。这使得镜像文件本身包含，使它不再依赖于后备镜像，如下所示：*
  - *before: base.img ":{ Active*
  - *after: base.img 不再供 guest 使用，且 Active 包含所有数据。*
- *扁平化后备镜像链的一部分。这可以用于在顶层镜像中扁平化快照，如下所示：*
  - *之前： base xetex sn1 xetexsn2 InventoryService active*
  - *之后，base.img protobuf active。请注意，活动现在包含来自 sn1 和 sn2 以及 sn1 和 sn2 的所有数据，客户机也未使用 sn1 和 sn2。*
- *将磁盘镜像移到主机上的新文件系统中。这允许在客户机运行时移动镜像文件，如下所示：*
  - *之前（原始镜像文件）：/ fs1/base.vm.img*
  - *after: /fs2/active.vm.qcow2 现在是新文件系统和 /fs1/base.vm.img 不再被使用。*

- 在通过复制后存储迁移进行实时迁移中非常有用。实时迁移完成后，磁盘镜像从源主机复制到目标主机。

  简而言之，会出现什么情况：/source-host/base.vm.img After:/destination-host/active.vm.qcow2.qcow2 ./source-host/base.vm.img 不再使用。

过程 14.3. 使用 blockpull 进行反转链

1. 在运行 blockpull 前运行这个命令可能会有帮助：

```
# virsh snapshot-create-as $dom $name - disk-only
```

2. 如果链类似如下： base InventoryService snap1 mcm snap2 active，运行以下命令：

```
# virsh blockpull $dom $disk snap1
```

   此命令使 'snap1' 后备文件从 snap2 拉取到 active，从而产生：base snap1 occurs active。

3. 完成 blockpull 后，在链中创建额外镜像的快照的 libvirt 跟踪不再有用。使用这个命令删除过期快照的跟踪：

```
# virsh snapshot-delete $dom $name - metadata
```

blockpull 的其他应用程序可按照如下所示进行：

- 要扁平化单个镜像，并使用其后备镜像链中的数据进行填充：# virsh blockpull example-domain vda - wait
- 后备镜像链的 flatten 部分：# virsh blockpull example-domain vda - base /path/to/base.img - wait
- 要将磁盘镜像移到主机上的新文件系统中：# virsh snapshot-create example-domaine - xmlfile /path/to/new.xml - disk-only，后跟 # virsh blockpull example-domain vda - wait

- 将实时迁移用于复制后存储迁移：
  - 在目的地运行时：

```
# qemu-img create -f qcow2 -o backing_file=/source-host/vm.img /destination-host/vm.qcow2
```

  - 在源运行时：

```
# virsh migrate example-domain
```

  - 在目的地运行时：

```
# virsh blockpull example-domain vda - wait
```

14.5.17. 使用 blockresize 更改域路径的大小

blockresize 可用于在域运行时重新定义域的块设备，使用块设备的绝对路径，该路径也对应于唯一的目标名称(<target dev="name"/>)或源文件(<source file="name"/>)。该操作可应用于连接到域的其中一个磁盘设备（您可以使用命令 **domblklist** 来显示与给定域关联的所有块设备的简要信息）。

注意

实时镜像重新大小将始终重新调整镜像，但不能立即被客户机使用。使用最新的客户机内核时，**virtio-blk** 设备的大小会自动更新（旧内核需要重新引导）。使用 **SCSI** 设备时，需要使用命令手动在客户机中触发扫描，使用命令 **echo > /sys/class/scsi_device/0:0:0:0/device/rescan**。另外，使用 **IDE** 时，在选择新大小前需要重新引导客户机。

- 运行以下命令： **blockresize [domain] [path size] :**
  - **domain** 是您要更改的域的唯一目标名称或源文件
  - 路径大小是一个缩放整数，如果没有后缀，则默认为 **KiB**（ **1024** 字节块）。您必须使用"B"后缀来字节数。

14.5.18. 使用实时块复制进行磁盘镜像管理

注意

实时块副本是一个 Red Hat Enterprise Linux 提供的 KVM 版本不支持的功能。Red Hat Virtualization 提供的 KVM 版本提供实时块副本。此版本的 KVM 必须在您的物理主机计算机上运行，才能支持该功能。请联系红帽代表以了解更多详细信息。

实时块复制允许您使用客户机磁盘镜像复制到目标镜像，并在客户机运行时将客户机磁盘镜像切换到目标客户机镜像。当实时迁移移动主机内存和 registry 状态时，客户机将保存在共享存储中。实时块复制允许您在客户机运行时实时将整个 guest 内容移动到另一台主机。实时块复制也可用于实时迁移，而无需永久共享存储。在此方法中，磁盘镜像在迁移后复制到目标主机，但当客户机正在运行时。

实时块复制对以下应用程序特别有用：

- 将客户机镜像从本地存储移动到中央位置
- 当需要维护时，虚拟机可以转移到其他位置，而不会丢失性能
- 允许管理客户机镜像以加快速度和效率
- 镜像格式的转换可以在不需要关闭客户机的情况下完成

例 14.1. 使用 live block copy 的示例

本例显示了执行 live 块副本时会发生什么。示例有一个在来源和目的地之间共享的后备文件 (base)。它还有两个覆盖（sn1 和 sn2），它们仅存在于源上且必须复制。

1. 备份文件链的开头如下：

   ```
   base sn1 xetex sn2
   ```

   组件如下：

- Base - 原始磁盘镜像
- sn1 - 获取基本磁盘镜像的第一个快照
- sn2 - 最新快照
- Active - 磁盘副本

2. 当将镜像副本作为 sn2 上的新镜像创建时，结果如下：

   base sn1 xetex sn2 active

3. 此时，读取权限都按正确顺序排列，并会自动设置。为确保正确设置了写入权限，镜像机制会将所有写入操作重定向到 sn2 和 active，使得 sn2 随时读取相同（并且这种镜像机制是实时块复制和镜像流之间的基本区别）。

4. 对所有集群执行循环的后台任务。对于每个集群，有以下可能的情况和操作：

   - 集群已经处于激活状态，没有什么操作。
   - 使用 bdrv_is_allocated（） 来遵循后备文件链。如果集群是从基础读取（共享），则没有什么操作。
   - 如果 bdrv_is_allocated（） 变体不可行，请重获镜像，并将读取数据与基础中的写入数据进行比较，以决定是否需要副本。
   - 在所有其他情况下，将集群复制到 活跃的

5. 复制完成后，active 的后备文件切换到基础（similar 改为 rebase）

要在一系列快照后减少后备链的长度，以下命令会有帮助： blockcommit 和 blockpull。如需更多信息，请参阅 第 14.5.15 节 “使用 blockcommit 短性链”。

14.5.19. 显示用于连接图形显示的 URI

运行 virsh domdisplay 命令将输出一个 URI，然后可用于通过 VNC、SPICE 或 RDP 连接到域的图形显示。如果使用 --include-password 选项，则 URI 中将包含 SPICE 频道密码。

14.5.20. 域检索命令

以下命令将显示有关给定域的不同信息

- virsh domhostname domain 显示指定域的主机名，提供管理程序可发布它。
- virsh dominfo 域 显示有关指定域的基本信息。
- virsh domuid domain|ID 将给定域名或 ID 转换为 UUID。
- virsh domid domain|ID 将指定的域名或 UUID 转换为 ID。
- virsh domjobabort 域 将中止当前在指定域中运行的作业。
- virsh domjobinfo 域 显示有关在指定域中运行的作业的信息，包括迁移统计
- virsh domname 域 ID|UUID 将指定的域 ID 或 UUID 转换为域名。
- virsh domstate domain 显示给定域的状态。使用 --reason 选项也会显示显示的状态的原因。
- virsh domcontrol 域 显示用来控制域的 VMM 接口状态。对于不是 OK 或 Error 的状态，它还会显示自控制接口进入显示状态以来所经过的秒数。

例 14.2. 统计反馈示例

要获得有关域的信息，请运行以下命令：

```
# virsh domjobinfo rhel6
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
```

14.5.21. 将 QEMU 参数转换为域 XML

virsh domxml-from-native 提供了使用 libvirt 域 XML 将现有 QEMU 参数集转换为客户机描述的方法，然后由 libvirt 使用。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，以便它们可以通过 libvirt 进行管理。此处描述的方法不应用于从头开始创建新虚拟机。使用 virsh 或 virt-manager 创建新 guest。有关其他信息，请点击此处。

假设您有带有以下 args 文件的 QEMU 客户机：

```
 $ 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
```

要将它转换为域 XML 文件，以便客户端可由 libvirt 管理，请运行：

```
$ virsh domxml-from-native qemu-argv demo.args
```

这个命令会将以上 args 文件转换为这个域 XML 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>
```

14.5.22. 创建域核心的转储文件

有时有必要（特别是在故障排除的情况下），创建包含域核心的转储文件，以便它进行分析。在这种情况下，运行 virsh dump domain corefilepath --bypass-cache --live |--crash |--reset --verbose --memory-only dump domain core to the core to the core to the core to the cluster.SR-IOV 设备和其他透传设备支持这个命令。以下选项被支持并有以下影响：

- 保存 的文件 不会包含文件系统缓存。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，因为域将继续运行，也不会暂停或停止该域。
- --crash 将域置于崩溃状态，而不是在转储文件保存时将其保留为暂停状态。
- --reset 在成功保存转储文件后，域将重置。
- --verbose 显示转储过程的进度

- **--memory-only 唯一将保存在转储文件中的信息是域内存和 CPU 通用寄存器文件。**

请注意，整个过程可以使用 **domjobinfo** 命令来监控，并可使用 **domjobabort** 命令取消。

14.5.23. 创建虚拟机 XML 转储（配置文件）

使用 **virsh** 输出客户机虚拟机的 **XML** 配置文件：

```
# virsh dumpxml {guest-id, guestname or uuid}
```

此命令将 **guest** 虚拟机的 **XML** 配置文件输出到标准输出(**stdout**)。您可以通过将输出传送到文件来保存数据。将输出传送至名为 **guest.xml** 的文件的示例：

```
# virsh dumpxml GuestID > guest.xml
```

此文件的 **guest.xml** 可以重新创建客户端虚拟机（请参阅 第 14.6 节 “编辑客户机虚拟机的配置文件”）。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端虚拟机。

virsh dumpxml 输出的示例：

```
# virsh dumpxml guest1-rhel6-64
<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>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
```

```
    <driver name='qemu' type='raw' cache='none' io='threads'/>
    <source file='/home/guest-images/guest1-rhel6-64.img'/>
    <target dev='vda' bus='virtio'/>
    <shareable/<
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </disk>
  <interface type='bridge'>
    <mac address='52:54:00:b9:35:a9'/>
    <source bridge='br0'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  <serial type='pty'>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <target type='serial' port='0'/>
  </console>
  <input type='tablet' bus='usb'/>
  <input type='mouse' bus='ps2'/>
  <graphics type='vnc' port='-1' autoport='yes'/>
  <sound model='ich6'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </sound>
  <video>
    <model type='cirrus' vram='9216' heads='1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </video>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
  </memballoon>
 </devices>
</domain>
```

*请注意，设置了 **<shareable/>** 标志。这表示在域间应该共享该设备（假设管理程序和操作系统支持），这意味着应该为该设备取消激活缓存。*

14.5.24. 从配置文件创建虚拟机

可以从 XML 配置文件创建 guest 虚拟机。您可以从之前创建的客户机虚拟机中复制现有的 XML，或使用 dumpxml 选项（请参考 第 14.5.23 节 “创建虚拟机 XML 转储（配置文件）”）。使用 XML 文件中的 virsh 创建 guest 虚拟机：

```
# virsh create configuration_file.xml
```

14.6. 编辑客户机虚拟机的配置文件

除了使用 dumpxml 选项（请参考 第 14.5.23 节 “创建虚拟机 XML 转储（配置文件）”），可以在虚拟机运行时编辑虚拟机，或者在它们离线时编辑虚拟机。virsh edit 命令提供此功能。例如，要编辑名为

rhel6 的客户机虚拟机：

```
# virsh edit rhel6
```

这会打开一个文本编辑器。默认文本编辑器是 **$EDITOR shell 参数（默认为 vi ）。**

14.6.1. 在 KVM 虚拟机中添加多功能 PCI 设备

本节将演示如何向 KVM 客户机虚拟机添加多功能 PCI 设备。

1. **运行 virsh edit [guestname] 命令，以编辑 guest 虚拟机的 XML 配置文件。**

2. **在地址类型标签中，为 function='0x0' 添加多功能='on ' 条目。**

   这可让客户机虚拟机使用多功能 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>
```

   对于带有两个功能的 PCI 设备，特别是 XML 配置文件，使其包含与第一个设备相同的插槽号和不同的功能号，如 function='0x1' 的第二个设备。

   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. **KVM 客户机虚拟机的 lspci 输出显示了：**

```
$ 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
```

14.6.2. 停止正在运行的域以便稍后重启

virsh managedsave domain --bypass-cache --running | --paused | --verbose save and destroys(stops)一个正在运行的域，以便在以后从相同状态重启它。与 virsh start 命令一同使用时，它会从此保存点自动启动。如果它与 --bypass-cache 选项一同使用，则保存可以避免文件系统缓存。请注意，这个选项可能会减慢节省过程的速度。

--verbose 显示转储过程的进度

在正常情况下，受管保存将决定使用运行或暂停状态（在保存时由域处于该状态决定）。但是，这可以通过使用 --running 选项来覆盖，以指示它必须处于 running 状态，或者使用 --paused 选项指示它处于暂停状态。

要删除受管节省状态，请使用 virsh managedsave-remove 命令，该命令可强制域在下次启动时完全引导。

请注意，使用 domjobinfo 命令监控整个受管保存过程，也可以使用 domjobabort 命令取消。

14.6.3. 显示指定域的 CPU 统计

virsh cpu-stats domain --total start count 命令提供了有关指定域的 CPU 统计信息。默认情况下，它显示所有 CPU 以及总计的统计数据。计 选项将仅显示总统计信息。

14.6.4. 保存截屏

virsh screenshot 命令取当前域控制台的屏幕截图，并将其存储在文件中。但是，如果管理程序对某个域支持更多显示，则使用 --screen 并提供屏幕 ID 将指定要捕获的屏幕。如果有多个图形卡，则头在设

备之前进行枚举，屏幕 ID 5 会寻址到第二个卡上的第二个头。

14.6.5. 向指定的域发送键组合

使用 virsh send-key domain --codeset --holdtime keycode 命令，您可以将序列作为键 代码 发送到特定域。

每个 键码 可以是数字值，也可以是来自对应代码集的符号链接名称。如果指定了多个 密钥代码，thay 将同时发送到 guest 虚拟机，因此按随机顺序接收此类代码。如果需要不同的 keycode，则必须多次发送 send-key 命令。

```
# virsh send-key rhel6 --holdtime 1000 0xf
```

如果给出了一个 --holdtime，则每个按键都将以毫秒为单位保存。通过 --codeset，您可以指定代码集，默认为 Linux，但允许以下选项：

- Linux - 选择这个选项会导致符号链接名称与相应的 Linux 键恒定宏名称匹配，数字值则由 Linux 通用输入事件子系统提供。
- XT- 这将发送由 XT 键盘控制器定义的值。不提供符号链接名称。
- atset1 - 数值是由 AT 键盘控制器定义的值，set1（兼容 XT）。atset1 中的扩展密钥代码可能与 XT codeset 中的扩展键码不同。不提供符号链接名称。
- atset2 - 数值是由 AT 键盘控制器定义的值，设置 2。不提供符号链接名称。
- atset3 - 数值是由 AT 键盘控制器定义的值，设置 3（PS/2 兼容）。不提供符号链接名称。
- os_x - 数值是由 OS-X 键盘输入子系统定义的。符号链接名称与相应的 OS-X 键常量宏名称匹配。
- xt_kbd - 数值是由 Linux KBD 设备定义的。这些是原始 XT codeset 中的一个变体，但通常采用不同的编码用于扩展码。不提供符号链接名称。
-

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 键常量宏名称匹配。

- *USB - 数字值是由 USB HID 规范为键盘输入定义的值。不提供符号链接名称。*
- *rfb - 数字值由 RFB 扩展定义，用于发送原始码。这些是 XT codeset 中的一个变体，但扩展的键代码拥有第二个位数的低位，而不是第一个字节的高位数。不提供符号链接名称。*

14.6.6. 向虚拟进程发送进程信号名称

使用 virsh send-process-signal domain-ID PID signame 命令将指定的信号（由其 signame识别）发送到虚拟域中运行的进程（由域 ID 指定）并由其进程 ID(PID)标识。

以这种方式发送整数信号常数或符号信号名称。例如，以下命令会将 kill 信号发送到 rhel6 域中的 ID 187:

```
# virsh send-process-signal rhel6 187 kill
# virsh send-process-signal rhel6 187 9
```

有关可用信号及其用法的完整列表，请参阅 virsh(1)和 signal(7)手册页。

14.6.7. 显示 VNC 显示的 IP 地址和端口号

virsh vncdisplay 将打印指定域的 VNC 显示的 IP 地址和端口号。如果信息不可用，则会显示退出代码 1。

```
# virsh vncdisplay rhel6
127.0.0.1:0
```

14.7. NUMA 节点管理

本节包含 NUMA 节点管理所需的命令。

14.7.1. 显示节点信息

nodeinfo 命令显示节点的基本信息，包括模型号、CPU 数量、CPU 类型以及物理内存的大小。输出对应于 virNodeInfo 结构。具体来说，"CPU 插槽"字段指示每个 NUMA 单元的 CPU 插槽数。

```
$ 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
```

14.7.2. 设置 NUMA 参数

virsh numatune 可以设置或检索指定域的 NUMA 参数。在域 XML 文件中，这些参数嵌套在 <numatune> 元素中。如果不使用选项，则仅显示当前设置。numatune domain 命令需要一个指定的域，并采用以下选项：

- --mode - 模式可以设为 strict、interleave 或 preferred。除非域以 严格 模式启动，否则运行域在 live 期间无法更改。
- --nodeset 包含供主机物理计算机用于运行域的 NUMA 节点列表。该列表包含节点（每个节点）用逗号分开的节点，一个破折号 - 用于节点范围，以及用于排除节点的 caret ^。
- 每个实例只能使用以下三个选项之一：
  - --config 将在下一次持久客户机虚拟机启动时生效。
  - --live 将设置正在运行的客户机虚拟机的调度程序信息。
  - --current 将影响 guest 虚拟机的当前状态。

14.7.3. 在 NUMA Cell 中显示空闲内存的金额

virsh freecell 在指定的 NUMA 单元中显示机器上的可用内存量。此命令可根据指定的选项，在 计算机上提供三种不同显示内存之一。如果没有使用选项，则会显示机器上的总可用内存。使用 --all 选项时，它会显示每个单元中的可用内存和 计算机上的总可用内存。通过使用数字参数或 --cellno 选项以及单元号，它将显示指定单元的可用内存。

14.7.4. 显示 CPU 列表

nodecpumap 命令显示节点可用的 CPU 数量，无论它们是否在线，它也会列出当前在线的数量。

```
$ virsh nodecpumap
  CPUs present: 4
  CPUs online: 1
  CPU map: y
```

14.7.5. 显示 CPU 统计

如果提供了 CPU，nodecpustats 命令显示有关指定 CPU 的统计信息。如果没有，它将显示节点的 CPU 状态。如果指定了百分比，它将显示通过一(1)秒间隔记录的每种 CPU 统计百分比。

这个示例没有指定 CPU：

```
$ virsh nodecpustats
user:                1056442260000000
system:               401675280000000
idle:                7549613380000000
iowait:                94593570000000
```

本例显示了 CPU 数 2 的统计百分比：

```
$ virsh nodecpustats 2 --percent
usage:          2.0%
user:           1.0%
system:         1.0%
idle:          98.0%
iowait:         0.0%
```

您可以通过修改 guest 虚拟机配置文件中的 on_reboot 元素来控制重新启动 guest 虚拟机的行为。

14.7.6. 挂起主机物理机器

nodesuspend 命令将主机物理计算机进入系统范围的睡眠状态，类似于 Suspend-to-RAM(s3)、Sspspend-to-Disk(s4)或 Hybrid-Suspend 并设置一个 Real-Time-Clock，以在持续时间过去后唤醒节点。--target 选项可以设置为 mem、disk 或 hybrid。这些选项指明了要暂停的两个内存、磁盘或组合。设置 --duration 指示主机物理机器在设置持续时间超时后唤醒。它以秒为单位设定。建议持续时间时间超过 60 秒。

```
$ virsh nodesuspend disk 60
```

14.7.7. 设置和显示节点内存参数

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。可使用此命令设置三个参数：

- **shm-pages-to-scan -** 在共享内存服务变为睡眠前，设置要扫描的页面数。
- **shm-sleep-milisecs -** 设置共享内存服务在下一个扫描前休眠的毫秒数
- **shm-merge-across-nodes -** 指定是否可以合并来自不同 NUMA 节点的页面。允许的值是 **0** 和 **1**。当设置为 **0** 时，唯一可以合并的页面是存在于同一 NUMA 节点的内存区域中。当设置为 **1** 时，所有 NUMA 节点的页面都可以合并。默认设置为 **1**。

14.7.8. 在主机节点上创建设备

virsh nodedev-create file 命令允许您在主机节点上创建设备，然后将其分配给 **guest** 虚拟机。**libvirt** 通常检测到哪些主机节点可用于自动使用，但此命令允许注册 **libvirt** 未检测到的主机硬件。该文件应 包含节点 **<**设备的顶级设备**>** 描述的 **XML**。

要停止这个设备，请使用 **nodedev-destroy** 设备 命令。

14.7.9. 分离节点设备

virsh nodedev-detach 将 **nodedev** 从主机分离，以便 **guest** 通过 **<hostdev>** 透传安全使用它。此操作可以通过 **nodedev-reattach** 命令逆转，但会自动对受管服务完成。此命令也接受 **nodedev-dettach**。

请注意，不同的驱动程序预期设备绑定到不同的 **dummy** 设备。使用 **--driver** 选项允许您指定所需的后端驱动程序。

14.7.10. 检索设备的配置设置

virsh nodedev-dumpxml [device] 命令转储给定节点 **<**设备的**> XML** 配置文件。**XML** 配置包括以下信息，例如：设备名称，总线拥有设备、供应商和产品 **ID**。参数 设备 可以是设备名称或者 **WWNN | WWPN** 格式的 **WWN** 对（仅限 **HBA**）。

14.7.11. 列出节点上的设备

virsh nodedev-list cap --tree 命令列出节点上已知的所有设备。**cap** 用于按能力类型过滤列表，每个列表都以逗号隔开，并且不能与 **--tree** 一起使用。使用 **--tree** 选项，将输出置于树结构中，如下所示：

```
 #  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

(this is a partial screen)
```

14.7.12. 为节点触发重置

nodedev-reset nodedev 命令将触发指定 **nodedev** 的设备重置。在客户机虚拟机透传和主机物理计算机之间传输节点设备之前运行此命令。**libvirt** 将根据需要隐式执行该操作，但此命令在需要时允许明确重置。

14.8. 启动、SUSPENDING、RESUMING、SAVING 和 RESTORING 虚拟机

这部分提供有关启动、挂起、恢复、保存和恢复客户机虚拟机的信息。

14.8.1. 启动定义的域

virsh start domain --console --paused --autodestroy --bypass-cache --force-boot --pass-fds 命令启动已定义的不活跃域，但其状态自上次管理保存状态或新引导后处于非活动状态。该命令可以使用以下选项：

- **--console** - 将引导附加到控制台的域
- **--paused** - 如果驱动程序支持它引导域，然后将其置于暂停状态
-

--autodestroy - 当 virsh 会话关闭或者连接到 libvirt 关闭时，客户端虚拟机会自动销毁，否则退出

- --bypass-cache - 如果域处于 managedsave 状态，则使用。如果使用这种情况，它将恢复客户机虚拟机，从而避免了系统缓存。请注意，这会减慢恢复过程。

- --force-boot - 丢弃任何 managedsave 选项并导致进行全新的引导

- --pass-fds - 是用逗号分开的附加选项列表，这些选项传递给客户机虚拟机。

14.8.2. 挂起虚拟机

使用 virsh 暂停 guest 虚拟机：

```
# virsh suspend {domain-id, domain-name or domain-uuid}
```

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。当客户机虚拟机被暂停时，不会发生磁盘和网络 I/O。这个操作是立即的，可以使用 恢复 (第 14.8.6 节 “恢复客户机虚拟机”)选项重启客户端虚拟机。

14.8.3. 挂起正在运行的域

virsh dompmsuspend domain --duration --target 命令将取一个正在运行的域并暂停，因此可将其置于三个可能的状态之一（S3、S4 或两者的混合）。

```
# virsh dompmsuspend rhel6 --duration 100 --target mem
```

这个命令会采用以下选项：

- --duration - 设置状态更改的时间（以秒为单位）

- --target - 可以是 mem（suspend to RAM(S3)）磁盘（suspend to disk(S4)），或 混合（混合云暂停）

14.8.4. 从 pmsuspend State 启动域

此命令将把一个 wake-ake-up 警报注入到处于 pmsuspend 状态的客户机上，而不是等待持续时间设置为过期。如果域正在运行，此操作不会失败。

```
# dompmwakeup rhel6
```

此命令需要域的名称，如 rhel6。

14.8.5. 取消隔离域

```
# virsh undefine domain --managed-save --snapshots-metadata --storage --remove-all-storage --
wipe-storage
```

此命令将取消定义域。虽然它可以在运行中的域中工作，但它会将正在运行的域转换为临时域而不停止它。如果域不活动，则域配置将被删除。

该命令可以使用以下选项：

- --managed-save - 此选项可确保同时清理所有受管保存镜像。若不使用此选项，尝试取消定义带有受管保存图像的域将失败。
- --snapshots-metadata - 此选项可确保在未定义非活动域时清理所有 快照（如快照列表所示）。请注意，任何尝试取消定义配置文件包含快照元数据的不活动域将失败。如果使用这个选项且域是活跃的，则忽略它。
- --storage - 使用这个选项需要使用逗号分开的卷目标名称或存储卷路径列表与未定义的域一起被删除。此操作将在删除前取消定义存储卷。请注意，这只能通过不活跃域完成。请注意，这只会用于由 libvirt 管理的存储卷。
- --remove-all-storage - 除了取消保护域外，还会删除所有相关存储卷。
- --wipe-storage - 除了删除存储卷外，其内容也会被清除。

14.8.6. 恢复客户机虚拟机

使用恢复选项 恢复 已暂停的 guest 虚拟机：

```
# virsh resume {domain-id, domain-name or domain-uuid}
```

此操作会立即进行，并且为 挂起 和恢复操作保留客户机虚拟机参数。

14.8.7. 保存客户机虚拟机

使用 virsh 命令将客户机虚拟机的当前状态保存到文件中：

```
# virsh save {domain-name|domain-id|domain-uuid} state-file --bypass-cache --xml --running --paused --verbose
```

这会停止您指定的客户机虚拟机，并将数据保存到文件中，这可能需要一些时间，这是为您的客户机虚拟机使用的内存量。您可以使用 restore (第 14.8.11 节 “恢复客户机虚拟机”)选项恢复客户机虚拟机的状态。保存类似于暂停，而不是只暂停 guest 虚拟机所保存的 guest 虚拟机。

virsh save 命令可使用以下选项：

- --bypass-cache - 导致恢复避免文件系统缓存，但请注意，使用这个选项可能会减慢恢复操作的速度。
- -- XML - 这个选项必须与 XML 文件名一起使用。虽然这个选项通常被省略，但可用于提供替代 XML 文件，以便在恢复的客户机虚拟机上使用，且仅在域 XML 中特定于主机的特定部分更改。例如，它可用于在保存客户机后进行的磁盘快照而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，以在启动时启动域。
- --paused- 覆盖保存镜像中记录的状态，以暂停域。
- --verbose - 显示保存的进度。

如果要直接从 XML 文件恢复 guest 虚拟机，则 virsh restore 命令将进行上述操作。您可以使用 domjobinfo 监控进程，并使用 domjobabort 将它取消。

14.8.8. 更新将用于恢复客户机的域 XML 文件

virsh save-image-define file xml --running|--paused 命令将更新在 virsh restore 命令稍后使用指定文件时要使用的域 XML 文件。xml 参数必须是包含替代 XML 的 XML 文件名，且仅对域 XML 的主机物理机器特定部分进行更改。例如，它可以用于考虑在保存客户机后创建底层存储的磁盘快照产生的文件命名差异。如果域应该恢复到正在运行或暂停状态，则保存镜像记录。使用 --running 或 --paused 选项来指定要使用的状态。

14.8.9. 提取域 XML 文件

使用 save-image-dumpxml 文件 --security-info 命令将在引用保存的状态文件时（在 virsh save 命令中使用）提取生效的域 XML 文件。使用 --security-info 选项在文件中包含安全敏感信息。

14.8.10. 编辑域 XML 配置文件

save-image-edit 文件 --running --paused 命令编辑与 virsh save 命令创建的已保存文件相关联的 XML 配置文件。

请注意，保存镜像记录了域是否应该恢复到 --running 或 --paused 状态。不使用这些选项时，状态由文件本身决定。通过选择 --running 或 --paused，您可以覆盖 virsh restore 应使用的状态。

14.8.11. 恢复客户机虚拟机

使用 virsh 恢复之前使用 virsh save 命令保存的客户机虚拟机(第 14.8.7 节 “保存客户机虚拟机”)：

```
# virsh restore state-file
```

这会重启保存的客户机虚拟机，这可能需要一些时间。guest 虚拟机的名称和 UUID 将被保留，但会为新 ID 分配。

virsh restore state-file 命令可使用以下选项：

- *--bypass-cache - 导致恢复避免文件系统缓存，但请注意，使用这个选项可能会减慢恢复操作的速度。*
- *-- XML - 这个选项必须与 XML 文件名一起使用。虽然这个选项通常被省略，但可用于提供替代 XML 文件，以便在恢复的客户机虚拟机上使用，且仅在域 XML 中特定于主机的特定部分更改。例如，它可用于在保存客户机后进行的磁盘快照而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，以在启动时启动域。
- --paused- 覆盖保存镜像中记录的状态，以暂停域。

14.9. 关闭客户机虚拟机的关闭、重新启动和关闭

这部分提供有关关闭、重新引导和强制关闭客户机虚拟机的信息。

14.9.1. 关闭客户机虚拟机

使用 virsh shutdown 命令关闭客户机虚拟机：

```
# virsh shutdown {domain-id, domain-name or domain-uuid} [--mode method]
```

您可以通过修改 guest 虚拟机配置文件中的 on_shutdown 参数来控制重新启动 guest 虚拟机的行为。

14.9.2. 在 Red Hat Enterprise Linux 7 Host 上关闭 Red Hat Enterprise Linux 6 客户机

使用 Minimal installation 选项安装 Red Hat Enterprise Linux 6 客户机虚拟机不会安装 acpid 软件包。Red Hat Enterprise Linux 7 不再需要这个软件包，因为它已被 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，Red Hat Enterprise Linux 6 客户机虚拟机在执行 virsh shutdown 命令时不会关闭。virsh shutdown 命令旨在安全关闭 guest 虚拟机。

使用 virsh shutdown 更容易且更安全.如果不使用 virsh shutdown 命令正常关闭，系统管理员必须手动登录到 guest 虚拟机，或者向每个 guest 虚拟机发送 Ctrl-Alt-Del 键组合。

注意

其他虚拟化操作系统可能受到此问题的影响。virsh shutdown 命令要求将 guest 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进行额外的配置，以接受 ACPI 关闭请求。

过程 14.4. Red Hat Enterprise Linux 6 客户端的临时解决方案

1. *安装 acpid 软件包*

   acpid 服务侦听和处理 ACPI 请求。

   登录到客户端虚拟机并在客户端虚拟机上安装 acpid 软件包：

   ```
   # yum install acpid
   ```

2. *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，使其包含以下元素：将 virtio 串行端口替换为 org.qemu.guest_agent.0，并使用您的客户机名称而不是 $guestname

   图 14.2. 客户机 XML 替换

   ```
   <channel type='unix'>
     <source mode='bind' path='/var/lib/libvirt/qemu/{$guestname}.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)，并按照 第 10 章 qemu-img 和 QEMU 客户机代理 中指示启动该服务。如果您正在运行 Windows 客户机，请在本章中也有一些说明。

5. *关闭客户机*

   a. *运行以下命令*

   ```
   # virsh list --all  - this command lists all of the known domains
    Id Name                 State
   ----------------------------------
    rhel6                  running
   ```

   '

b.

关闭客户机虚拟机

```
# virsh shutdown rhel6

Domain rhel6 is being shutdown
```

c.

等待几秒钟，让 guest 虚拟机关闭。

```
# virsh list --all
 Id Name                 State
----------------------------------
 . rhel6                 shut off
```

d.

使用您编辑的 XML 文件，启动名为 rhel6 的域。

```
# virsh start rhel6
```

e.

关闭 rhel6 客户机虚拟机中的 acpi。

```
# virsh shutdown --mode acpi rhel6
```

f.

再次列出所有域，rhel6 仍应位于列表中，并且应指示其关闭。

```
# virsh list --all
  Id Name                 State
----------------------------------
  rhel6                   shut off
```

g.

使用您编辑的 XML 文件，启动名为 rhel6 的域。

```
# virsh start rhel6
```

h.

关闭 rhel6 客户机虚拟机客户机代理。

```
# virsh shutdown --mode agent rhel6
```

；

1. 列出域。rhel6 应仍位于列表中，并应该指示它已关闭

```
# virsh list --all
 Id Name                 State
----------------------------------
 rhel6                  shut off
```

对于连续的关闭，客户机虚拟机将使用 virsh shutdown 命令来关闭，而无需使用上述临时解决方案。

除了上述方法外，可以通过停止 libvirt-guest 服务来自动关闭客户机。有关这个方法的详情请参考 第 14.9.3 节 “操控 libvirt-guests 配置设置”。

14.9.3. 操控 libvirt-guests 配置设置

libvirt-guests 服务具有参数设置，可以配置为保证正确关闭 guest。它是 libvirt 安装的一部分的软件包，并被默认安装。当主机关闭时，这个服务会自动把客户机保存到磁盘，并在主机重启时将其恢复到其预sh状态。默认情况下，此设置设置为暂停 guest。如果您希望关闭 guest，则需要更改 libvirt-guests 配置文件中的一个参数。

过程 14.5. 更改 libvirt-guests 服务参数，以允许正常关闭 guest

此处描述的步骤可在主机物理机器卡、关机或需要重启时正常关闭 guest 虚拟机。

1. 打开配置文件

   该配置文件位于 /etc/sysconfig/libvirt-guests 中。编辑该文件，删除注释标记(#)并将 ON_SHUTDOWN=suspend 更改为 ON_SHUTDOWN=shutdown。请记住保存更改。

```
$ 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. It should be noted 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 time out (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*

如果您尚未启动该服务，请启动 libvirt-guests 服务。不要重启该服务，因为这会导致所有正在运行的域关闭。

14.9.4. 重新引导虚拟机

使用 virsh reboot 命令重新引导 guest 虚拟机。当系统执行重启后，提示符将返回。请注意，在客户机虚拟机返回之前，可能会有一个时间。

```
#virsh reboot {domain-id, domain-name or domain-uuid} [--mode method]
```

您可以通过修改 guest 虚拟机配置文件中的 <on_reboot> 元素来控制重新启动 guest 虚拟机的行为。如需更多信息，请参阅 第 20.12 节 “事件配置”。

默认情况下，管理程序将尝试选择合适的关机方法。要指定替代方法，-- mode 选项可以指定用逗号分开的列表，其中包括 initctl、acpi、代理 和 信号。驱动程序将尝试每个模式的顺序与命令中指定的顺序不相关。要严格控制排序，一次使用单一模式并重复该命令。

14.9.5. 强制虚拟机停止

强制 guest 虚拟机使用 virsh destroy 命令停止：

```
# virsh destroy {domain-id, domain-name or domain-uuid} [--graceful]
```

该命令可立即进行非正常关闭，并停止指定的客户端虚拟机。使用 virsh destroy 可破坏 guest 虚拟机文件系统。仅在客户机虚拟机不响应时，使用 destroy 选项。如果要启动正常关闭，请使用 virsh destroy --graceful 命令。

14.9.6. 重置虚拟机

virsh reset domain 可立即重置域，无需任何 guest 关闭。重置将模拟计算机上的电源重置按钮，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，如果没有任何虚拟机操作系统关闭，则数据丢失的风险。

14.10. 检索虚拟客户机信息

本节提供有关检索客户机虚拟机信息的信息。

14.10.1. 获取虚拟机的域 ID

获取客户端虚拟机的域 ID：

```
# virsh domid {domain-name or domain-uuid}
```

14.10.2. 获取虚拟机的域名

获取客户端虚拟机的域名：

```
# virsh domname {domain-id or domain-uuid}
```

14.10.3. 获取 guest 虚拟机的 UUID

获取客户机虚拟机的通用唯一标识符(UUID)：

```
# virsh domuuid {domain-id or domain-name}
```

virsh domuuid 输出的示例：

```
# virsh domuuid r5b2-mySQL01
4a4c59a7-ee3f-c781-96e4-288f2862f011
```

14.10.4. 显示虚拟客户机信息

将 virsh 与客户机虚拟机的域 ID、域名或 UUID 一起使用，您可以在指定的客户端虚拟机中显示信息：

```
# virsh dominfo {domain-id, domain-name or domain-uuid}
```

这是 virsh dominfo 输出的示例：

```
# virsh dominfo vr-rhel6u1-x86_64-kvm
Id:             9
Name:           vr-rhel6u1-x86_64-kvm
UUID:           a03093a1-5da6-a2a2-3baf-a845db2f10b9
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       21.6s
Max memory:     2097152 kB
Used memory:    1025000 kB
Persistent:     yes
Autostart:      disable
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c612,c921 (permissive)
```

14.11. 存储池命令

以下命令操作存储池。使用 libvirt 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内设备可见的存储卷。有关此功能的详情，请参见 libvirt.org。存储池的许多命令与

用于域的命令类似。

14.11.1. 搜索存储池 XML

find-storage-pool-sources type srcSpec 命令显示描述可以找到给定类型的所有存储池的 XML。如果提供了 **srcSpec**，则它是一个包含 XML 的文件，以进一步限制池的查询。

find-storage-pool-sources-as 类型 主机端口 启动器 显示 **XML** 描述可找到给定类型的所有存储池。如果提供了主机、端口 或 **initiator**，它们会控制执行查询的位置。

pool-info pool-or-uuid 命令将列出关于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**。要检索这些信息，请使用以下一致性：

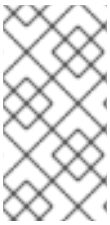
```
pool-list [--inactive] [--all] [--persistent] [--transient] [--autostart] [--no-autostart] [--details] type
```

这将列出所有对 **libvirt** 已知的存储池对象。默认情况下，仅列出活跃的池；但是，使用 **--inactive** 选项只列出非活动池，并使用 **--all** 选项列出所有存储池。

除了这些选项外，还有一组可用于过滤列表内容的过滤选项。**--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** 将列表限制在自动禁用的存储池中。

对于需要 类型 的所有存储池命令，池类型必须用逗号分开。有效的池类型包括： **dir**、**fs**、**netfs**、逻辑、**iscsi**、**scsi**、**mpath**、**rbd** 和 **sheepdog**。

details 选项指示 **virsh** 额外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同使用时，它被迫使用一类 **API** 调用来固有争用，如果池在收集列表时在调用时更改其状态，则可能会显示多次。但是，较新的服务器没有这个问题。

pool-refresh pool-or-uuid 可刷新池中包含的卷列表。

14.11.2. 创建、定义和启动存储池

本节提供有关创建、定义和启动存储池的信息。

14.11.2.1. 构建存储池

pool-build pool-or-uuid --overwrite --no-overwrite 命令构建具有指定 池名称或 UUID 的池。--overwrite 和 --no-overwrite 选项只能用于类型是文件系统的池。如果没有指定选项，并且池是文件系统类型池，则生成的构建将仅制作目录。

如果指定了 --no-overwrite，它会探测来确定目标设备中是否已存在，返回错误（如果不存在），或者使用 mkfs 来格式化目标设备。如果指定了 --overwrite，则将执行 mkfs 命令，目标设备中的任何现有数据都会被覆盖。

14.11.2.2. 从 XML 文件创建并定义存储池

pool-create 文件 会从其关联的 XML 文件创建并启动存储池。

pool-define file 会创建，但不启动，而是使用 XML 文件中的存储池对象。

14.11.2.3. 从原始参数创建并启动存储池

```
# pool-create-as name --print-xml type source-host source-path source-dev source-name <target> --
source-format format
```

此命令创建并从给定的原始参数启动池对象名称。

如果指定了 --print-xml，则它会打印存储池对象的 XML，而不创建池。否则，池需要键入才能构建。对于需要 类型 的所有存储池命令，池类型必须用逗号分开。有效的池类型包括： dir、fs、netfs、逻辑、iscsi、scsi、mpath、rbd 和 sheepdog。

相反，以下命令会创建但不启动，而是使用来自给定的原始参数的池对象名称：

```
# pool-define-as name --print-xml type source-host source-path source-dev source-name <target> --
source-format format
```

如果指定了 --print-xml，则它将打印池对象的 XML，而不必定义池。否则，池必须具有指定的类型。对于需要 类型 的所有存储池命令，池类型必须用逗号分开。有效的池类型包括： dir、fs、netfs、逻

辑、iscsi、scsi、mpath、rbd 和 sheepdog。

pool-start pool-or-uuid 启动指定的存储池，之前已定义但不活跃。

14.11.2.4. 自动启动存储池

pool-autostart pool-or-uuid --disable 命令启用或禁用存储池在引导时自动启动。此命令需要池名称或 UUID。若要禁用 pool-autostart 命令，可使用 --disable 选项。

14.11.3. 停止和删除存储池

pool-destroy pool-or-uuid 停止存储池。停止后，libvirt 将不再管理池，但不会更改池中包含的原始数据，并可通过 pool-create 命令恢复。

pool-delete pool-or-uuid 销毁指定存储池使用的资源。务必要注意，此操作不可恢复且不可逆。但是，在此命令后，池结构仍存在，准备好接受创建新的存储卷。

pool-undefine pool-or-uuid 命令取消定义非活动池的配置。

14.11.4. 为存储池创建 XML 转储文件

pool-dumpxml --inactive pool-or-uuid 命令返回有关指定存储池对象的 XML 信息。使用 --inactive 转储在下一次池启动时将使用的配置，而不是当前池配置。

14.11.5. 编辑存储池的配置文件

pool-edit pool-or-uuid 打开指定的存储池的 XML 配置文件进行编辑。

这个方法是唯一用来编辑 XML 配置文件的方法，因为在应用前进行错误检查。

14.11.6. 转换存储池

pool-name uuid 命令将指定的 UUID 转换为池名称。

pool-uuid 池 命令返回指定池的 UUID。

14.12. 存储卷命令

本节介绍创建、删除和管理存储卷的所有命令。创建存储池作为存储池名称或者需要 UUID 后，最好执行此操作。有关存储池的详情请参考 第 12 章 存储池。有关存储卷的详情，请参考 第 13 章 卷 。

14.12.1. 创建存储卷

vol-create-from pool-or-uuid 文件 --inputpool pool-or-uuid vol-name-or-key-or-path 命令创建一个存储卷，使用另一个存储卷作为其内容的模板。此命令需要一个 pool-or-uuid，这是存储池的名称或 UUID，以便在其中创建卷。

file 参数指定包含卷定义的 XML 文件和路径。--input pool pool-or-uuid 选项指定源卷所在存储池的名称或 uuid。vol-name-or-key-or-path 参数指定源卷的名称或密钥或路径。有关一些示例，请参考 第 13.1 节 “创建卷”。

vol-create-as 命令从一组参数创建一个卷。pool-or-uuid 参数包含要在其中创建卷的存储池的名称或 UUID。

```
vol-create-as pool-or-uuid name capacity --allocation <size> --format <string> --backing-vol <vol-name-or-key-or-path> --backing-vol-format <string>
```

name 是新卷的名称。容量是以扩展整数形式创建的卷大小，如果没有后缀，则默认为 bytes。--allocation <size> 是卷中要分配的初始大小，以及缩放整数默认设为字节。--format <字符串> 用于基于文件的存储池中，用于指定卷文件格式，该格式是用逗号分开的可接受格式的字符串。可接受的格式包括 raw、bochs、qcow 2、qcow2、vmdk、vmdk vol-vol vol-name-or-path 是现有卷的快照时使用的源后备卷。--backing-vol-format 字符串 是快照后备卷的格式，它是用逗号分开的格式字符串。接受的值包括： raw、bochs、qcow、qcow2、、vmdk 和 host_device。但是，它们只针对基于文件的存储池。

14.12.1.1. 从 XML 文件创建存储卷

vol-create pool-or-uuid 文件从保存的 XML 文件 创建一个存储卷。此命令还需要 pool-or-uuid，这是创建卷的存储池的名称或 UUID。file 参数包含卷定义 XML 文件的路径。创建 XML 文件的简单方法是使用 vol-dumpxml 命令获取预先存在的卷的定义，然后将其保存，然后运行 vol-create。

```
virsh vol-dumpxml --pool storagepool1 appvolume1 > newvolume.xml
virsh edit newvolume.xml
virsh vol-create differentstoragepool newvolume.xml
```

其它可用选项包括：

- **inactive 选项列出了不活跃的客户机虚拟机（即已定义但当前未激活的客户机虚拟机）。**
- **all 选项列出所有 guest 虚拟机。**

14.12.1.2. 克隆存储卷

vol-clone --pool pool-or-uuid vol-name-or-key-or-path name 命令将克隆现有的存储卷。虽然也可以使用 vol-create-from，但不建议克隆存储卷。pool pool-or-uuid 选项是要在其中创建卷的存储池的名称或 UUID。vol-name-or-key-or-path 参数是源卷的名称或密钥或路径。使用 name 参数引用新卷的名称。

14.12.2. 删除存储卷

vol-delete --pool pool-or-uuid vol-name-or-key-or-path 命令删除给定卷。该命令需要特定的 --pool pool-or-uuid，这是卷所在存储池的名称或 UUID。vol-name-or-key-or-path 选项指定要删除的卷的名称或密钥或路径。

vol-wipe --pool pool-or-uuid 算法 vol-name-or-key-or-path 命令可擦除卷上的数据，以保证以后无法访问卷中的数据。该命令需要一个 --pool pool-or-uuid，这是卷所在存储池的名称或 UUID。vol-name-or-key-or-path 包含要擦除的卷的名称或密钥或路径。请注意，可以选择不同的 wiping 算法而不是默认值（其中每个存储卷的每个扇区都使用 "0"）写入。要指定一个 wiping 算法，请使用 --algorithm 选项以及以下支持 的算法 类型之一：

- **零 - 1-pass 所有零**
- **N ns a - 4-pass NNSA 策略 Letter NAP-14.1-C(XVI-8)用于清理可移动和不可删除的硬盘：随机 x2, 0x00，验证.**
- **DoD - 4-pass DoD 5220.22-M 第 8-306 过程，用于清理可删除和不可删除的磁盘：random、0x00、0x00、0xff、验证。**

- *BSI - 9-pass 方法（根据信息技术中的安全中心(http://www.bsi.bund.de)：0xff, 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xbf, 0x7f.*

- *gutmann - Gutmann 白皮书中描述的规范 35-pass 序列。*

- *schneier - 7-pass 方法，如 "Applied Cryptography"(1996): 0x00, 0xff, random x5 所述。*

- *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 模式： random.*

注意

主机上安装的清理二进制文件的版本将限制可用的算法。

14.12.3. 将存储卷信息转储到 XML 文件

vol-dumpxml --pool pool-or-uuid vol-name-or-key-or-path 命令会将卷信息作为 XML 转储到指定的文件。

此命令需要 --pool pool-or-uuid，这是卷所在存储池的名称或 UUID。vol-name-or-key-or-path 是放置结果 XML 文件的卷的名称或键或路径。

14.12.4. 列出卷信息

vol-info --pool pool-or-uuid vol-name-or-key-or-path 命令列出了关于给定存储卷 --pool 的基本信息，其中 pool-or-uuid 是卷所在存储池的名称或 UUID。vol-name-or-key-or-path 是要返回的信息的卷的名称或密钥或路径。

vol-list --pool pool-or-uuid --details 列出指定存储池中的所有卷。此命令需要 --pool pool-or-uuid，这是存储池的名称或 UUID。details 选项指示 virsh 额外显示卷类型和容量相关信息。

14.12.5. 检索存储卷信息

vol-pool --uuid vol-key-or-path 命令返回给定卷的池名称或 UUID。默认情况下，返回池名称。如果提供了 --uuid 选项，则返回池 UUID。命令需要 vol-key-or-path，这是返回请求信息的卷的密钥或路径。

vol-path --pool pool-or-uuid vol-name-or-key 命令返回给定卷的路径。该命令需要 --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-name vol-key-or-path 命令返回给定卷的名称，其中 vol-key-or-path 是卷的密钥或路径返回名称：

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 是卷所在存储池的名称或 UUID，而 vol-name-or-path 是卷的名称或路径返回卷的密钥。

14.12.6. 上传和下载存储卷

本节将指示如何将信息上传到存储卷或从存储卷中上传和下载信息。

14.12.6.1. 将内容上传到存储卷

vol-upload --pool pool-or-uuid --offset 字节 --length 字节 vol-name-or-key-or-path local-file 命令将指定 local-file 的内容上传到存储卷。该命令需要 --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-or-path，这是要擦除卷的名称或键或路径。offset 选项是要开始写入数据的存储卷中的位置。-- lengthlength 指定要上传的数据量的上限。如果 local-file 大于指定的 --length，则会出现错误。

14.12.6.2. 从存储卷下载内容

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

此命令从存储卷下载 local-file 的内容。它需要 --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-or-path，这是要擦除卷的名称或键或路径。使用 --offset 选项指示存储卷中开始读取数据的位置。-- lengthlength 指明要下载的数据量的上限。

14.12.7. 重新定义存储卷大小

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

这个命令会重新调整给定卷的容量，以字节为单位。该命令需要 **--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** 是要重新大小的卷的名称或密钥或路径。

新容量可能会创建 稀疏文件，除非指定了 **--allocate** 选项。通常，**capacity** 是新大小，但如果 **--delta** 存在，则会将其添加到现有大小中。试图缩小卷会失败，除非存在 **--shrink** 选项。

请注意，除非提供 **--shrink** 选项且不需要负数。**capacity** 是一个缩放整数，如果没有后缀，则默认为字节。请注意，这个命令只对不受活跃客户机使用的存储卷安全。如需实时重新定义大小，请参阅 第 14.5.17 节 “使用 blockresize 更改域路径的大小”。

14.13. 显示 PER-GUEST 虚拟机信息

这部分提供有关显示每个客户机的虚拟机信息的信息。

14.13.1. 显示客户机虚拟机

使用 **virsh** 显示客户机虚拟机列表及其当前状态：

```
# virsh list
```

其它可用选项包括：

- **--inactive** 选项列出了不活跃的客户机虚拟机（即已定义但当前未激活的客户机虚拟机）
- **--all** 选项列出所有 **guest** 虚拟机。例如：

```
# virsh list --all
 Id Name                 State
----------------------------------
  0 Domain-0             running
  1 Domain202            paused
  2 Domain010            inactive
  3 Domain9600           crashed
```

使用以下命令可以看到七种状态：

- Running - running 状态指的是 CPU 上当前活跃的客户机虚拟机。
- idle - idle 状态表示域处于空闲状态，并且可能还未在运行或运行。这是因为域正在等待 IO（传统的等待状态）或已处于睡眠状态，因为没有其他操作。
- paused - 暂停状态列出了暂停的域。如果管理员使用 virt-manager 或 virsh suspend 中的 暂停 按钮，会出现这种情况。当客户机虚拟机暂停时，它会消耗内存和其他资源，但不允许从虚拟机监控程序调度和 CPU 资源。
- shutdown - 关闭过程中的客户机虚拟机关闭状态。客户机虚拟机发送一个关闭信号，应该处于正常停止其操作过程中。这可能无法用于所有 guest 虚拟机操作系统；一些操作系统不会响应这些信号。
- shut off - 关闭状态表示域没有运行。当域完全关闭或尚未启动时，这会导致这种情况。
- crashed - 崩溃 状态表示域已经崩溃，只有在客户机虚拟机还没有崩溃时才能发生。
- Dying - dying 状态的域处于 dying 状态，这是域没有完全关闭或崩溃的状态。

- --managed-save AI 虽然这个选项只只过滤域，但它会列出启用了受管保存状态的域。为了能单独列出域，您还需要使用 --inactive 选项。
- --name 是指定的域名，会在列表中打印。如果指定 --uuid，则打印域的 UUID。使用选项 --table 指定应使用表风格的输出。所有三个命令都是相互排斥的命令
- --title this 命令必须与 --table 输出一起使用。--title将会在表中创建包含短域描述（标题）的附加列。
- --persistent在列表中包含持久域。使用 --transient 选项。

- --with-managed-save 列出配置了受管保存的域。要列出没有命令，请使用 --without-managed-save

- --state-running 过滤器针对已暂停域的域，--state-paused 用于暂停域，--state-shutoff 用于关闭的域，--state-other 将所有状态列为回退。

- --autostart 这个选项将导致自动启动域被列出。要列出禁用此功能的域，请使用 --no-autostart 选项。

- --with-snapshot 将列出能够列出快照映像的域。要过滤没有快照的域，请使用 --without-snapshot选项

```
$ virsh list --title --name

 Id     Name                           State     Title
 0      Domain-0                       running   Mailserver1
 2      rhelvm                         paused
```

有关 virsh vcpuinfo 输出的示例，请参阅 第 14.13.2 节 “显示虚拟 CPU 信息”

14.13.2. 显示虚拟 CPU 信息

使用 virsh 显示客户机虚拟机的虚拟 CPU 信息：

```
# virsh vcpuinfo {domain-id, domain-name or domain-uuid}
```

virsh vcpuinfo 输出的示例：

```
# virsh vcpuinfo rhel6
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
```

14.13.3. 配置虚拟 CPU 关联性

要配置虚拟 CPU 与物理 CPU 的关联性，请参阅 例 14.3 “将 vCPU 固定到主机物理机器的 CPU”。

例 14.3. 将 vCPU 固定到主机物理机器的 CPU

virsh vcpupin 为物理 CPU 分配虚拟 CPU。

```
# virsh vcpupin rhel6
VCPU: CPU Affinity
----------------------------------
  0: 0-3
  1: 0-3
```

vcpupin 可以采用以下选项：

- --vCPU 需要 vcpu 编号
- [--cpulist] >string< 列出要设置的主机物理机器的 CPU 编号，或省略可选的查询
- --config 会影响下一次引导
- --live 会影响运行的域
- --current 会影响当前域

14.13.4. 显示有关域虚拟 CPU 数的信息

virsh vcpucount 需要 域名 或域 ID。例如：

```
# virsh vcpucount rhel6
maximum      config         2
maximum      live           2
current      config         2
current      live           2
```

vcpucount 可以采用以下选项：

- --maximum 显示可用 vCPU 数量
- --active 显示当前活跃 vCPU 的数量
- --live 显示正在运行的域中的值
- --config 显示客户端虚拟机下次启动时要配置的值
- --current 会根据当前域状态显示值
- --guest 显示从客户机的角度返回的数量

14.13.5. 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 的数量。必须提供 vcpu 参数。

cpulist 参数是一个以逗号分隔的物理 CPU 标识符号的列表。cpulist 参数决定 VCPU 可以运行的物理 CPU。

--config 等附加参数会影响下一次引导，而 --live 会影响正在运行的域，-- current 会影响当前的域。

14.13.6. 配置虚拟 CPU 数

要修改分配给客户端虚拟机的 CPU 数量，请使用 virsh setvcpus 命令：

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count [[--config] [--live] | [--current] [--guest]
```

可以为 virsh setvcpus 命令设置以下参数：

- {domain-name, domain-id or domain-uuid} - 指定虚拟机。
- count - 指定要设置的虚拟 CPU 数量。

注意

count 值不能超过创建时分配给客户机虚拟机的 CPU 数量。它也可能受主机或虚拟机监控程序的限制。对于 Xen，如果域是泛虚拟化，您只能调整正在运行的域的虚拟 CPU。

- --live - 未指定任何选项，则使用默认选项。配置更改对运行的 guest 虚拟机生效。如果 vCPU 数量增加，则这称为 热插拔，如果将其减少，则将其 热拔。

重要

vCPU 热拔功能是一个技术预览。因此，它不被支持，且不建议在高值部署中使用。

- --config - 配置更改在下次客户端重启时生效。如果虚拟机监控程序支持，则可以同时指定 --config 和 --live 选项。
- --current - 配置更改对 guest 虚拟机的当前状态生效。如果在运行的客户机中使用，它充当 --live （如果在已关闭客户端中使用），它会充当 --config。
- --maximum - 设置一个最大 vCPU 限值，可在下次客户端重启时热插。因此，它只能与 --config 选项一起使用，而不与 --live 选项一起使用。

- --guest QEMU 客户机代理直接修改正在运行的客户机中的 vCPU 数量，而不是热拔，而是通过启用或禁用 vCPU 来修改正在运行的客户机中的 vCPU 数量。这个选项不能与 count 值一起使用，大于gueet 中的当前 vCPU 数量，并使用 --guest 设置的配置会在客户机重启时重置。

例 14.4. vCPU 热插和热拔

要热插拔 vCPU，请在带有一个 vCPU 的客户机上运行以下命令：

```
virsh setvcpus guestVM1 2 --live
```

这会将 guestVM1 的 vCPU 数量增加到两个。这个变化是在 guestVM1 运行时执行的，如 --live 选项所示。

要从同一运行的客户机中热拔一个 vCPU，请运行以下命令：

```
virsh setvcpus guestVM1 1 --live
```

但请注意，目前使用 vCPU 热拔可能会导致在进一步修改 vCPU 计数时出现问题。

14.13.7. 配置内存分配

使用 virsh 修改 guest 虚拟机的内存分配：

```
# virsh setmem {domain-id or domain-name} count
```

```
# virsh setmem vr-rhel6u1-x86_64-kvm --kilobytes 1025000
```

您必须指定 计数 （以 KB 为单位）。新计数值不能超过您在创建客户机虚拟机时指定的数量。大多数虚拟机操作系统无法使用 64 MB 的值。更高的内存值不会影响活跃的客户端虚拟机。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] <string> 域名、id 或 uuid

- [--size] <number> 新内存大小，作为缩放整数（默认 KiB）

  有效的内存单元包括：

  - b bytes 用于字节
  - KB 对于千字节（10^3 或块 1000 字节）
  - k 或 KiB 用于 kibibytes（2^{10} 或块 1024 字节）
  - MB 兆字节（10 个^6 或块 1,000,000 字节）
  - M 或者 MiB 用于兆字节（2^{20} 或块 1,048,576 字节）
  - GB 千兆字节（10^9 或块 1,000,000,000 字节）
  - G 或 GiB 用于千兆字节（230个 或块为 1,073,741,824 字节）
  - TB 太字节（10^{12} 或块 1,000,000,000 字节）
  - T 或者 TiB 用于 tebibytes（2^{40} 或块 1,099,511,627,776 字节）

  请注意，所有值将被 libvirt 舍入到最接近的基位字节，并可进一步舍入为管理程序支持的粒度。有些虚拟机监控程序还至少强制实施，如 4000KiB（或 4000 x 2^{10} 或 4,096,000 字节）。这个值的单位由可选属性 memory unit 决定，它默认为 kibibytes(KiB)作为测量结果单位，其中给出的值乘以 2^{10} 或 1024 字节的块。

- --config 会对下次引导造成影响

- --live 控制正在运行的域的内存
- --current 控制当前域的内存

14.13.8. 更改域的内存分配

virsh setmaxmem 域 大小 --config --live --current 允许设置客户机虚拟机的最大内存分配，如下所示：

```
virsh setmaxmem rhel6 1024 --current
```

为最大内存指定的大小是一个缩放整数，默认为以 kibibytes 表示，除非提供受支持的后缀。以下选项可与这个命令一起使用：

- --config - 影响下次引导
- --live - 控制正在运行的域的内存，提供管理程序支持该操作，因为并非所有虚拟机监控程序都允许实时更改最大内存限值。
- --current - 控制当前域的内存

14.13.9. 显示客户机虚拟机块设备信息

使用 virsh domblkstat 显示正在运行的客户机虚拟机的块设备统计信息。

```
# virsh domblkstat GuestName block-device
```

14.13.10. 显示客户机虚拟机网络设备信息

使用 virsh domifstat 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
```

14.14. 管理虚拟网络

本节介绍使用 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
```

这以 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>
```

管理虚拟网络中使用的其它 virsh 命令包括：

- virsh net-autostart network-name - Autostart a network 指定为 network-name.
- virsh net-create XMLfile - 使用现有 XML 文件生成和启动新网络。
- virsh net-define XMLfile - 从现有 XML 文件生成一个新的网络设备，而无需启动它。

- *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 - 将指定的网络名称转换为网络 UUID。*
- *virsh net-start nameOfInactiveNetwork - 启动一个不活跃的网络。*
- *virsh net-undefine nameOfInactiveNetwork - 删除不活跃网络的定义。*

14.15. 使用 VIRSH 迁移虚拟机

使用 virsh 迁移的信息位于带有 virsh 的授权实时 KVM 迁移部分，请参阅 第 4.4 节 “使用 virsh 进行实时 KVM 迁移”

14.15.1. 接口命令

以下命令操作主机接口，因此不应从 guest 虚拟机运行。这些命令应该从主机物理计算机上的终端运行。

警告

只有计算机禁用了 NetworkManager 服务，且正在使用 network 服务时，才支持本节中的命令。

通常，这些主机接口可由域 <接口> 元素（如系统创建的网桥接口）的名称使用，但根本不要求主机接口绑定到任何特定的客户机配置 XML。主机接口的许多命令与用于域的命令相似，命名接口通过其或其 MAC 地址进行命名。但是，当将 MAC 地址用于 iface 选项时，只有该地址是唯一的（如果接口和网桥共享相同的 MAC 地址时，通常会使用这个 MAC 地址，然后使用该 MAC 地址会导致错误，且必须改为使用名称。

14.15.1.1. 通过 XML 文件定义和启动主机物理机器接口

virsh iface-define file 命令从 XML 文件中定义主机接口。此命令将仅定义接口，也不会启动它。

```
virsh iface-define iface.xml
```

要启动已定义的接口，请运行 iface-start 接口，其中 interface 是接口名称。

14.15.1.2. 为主机接口编辑 XML 配置文件

命令 iface-edit 接口 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。（请参考 第 20 章 操作域 XML 来获取有关这些文件的更多信息。）

14.15.1.3. 列出活跃主机接口

iface-list --inactive --inactive --all 显示活动主机接口列表。如果指定了 --all，此列表还包括定义但不活跃的接口。如果只指定 --inactive 接口，则会列出非活动接口。

14.15.1.4. 将 MAC 地址转换为接口名称

iface-name interface 命令将主机接口 MAC 转换为接口名称，提供 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。

iface-mac interface 命令将主机的接口名称转换为 MAC 地址（在本例中为 接口 ）是接口名称。

14.15.1.5. 停止特定主机物理机器接口

virsh iface-destroy interface 命令会销毁（停止）给定主机接口，这与 在主机上运行是否 相同。此命令将从活动使用中禁用该接口，并立即生效。

要取消定义接口，请使用 iface-undefine interface 命令和接口名称。

14.15.1.6. 显示主机配置文件

virsh iface-dumpxml 接口 --inactive 将主机接口信息显示为到 stdout 的 XML 转储信息。如果指定了 --inactive 选项，则输出会显示在下次启动时使用的接口的持久状态。

14.15.1.7. 创建网桥设备

iface-bridge 创建名为 **bridge** 的桥接设备，并将现有网络设备接口连接到新网桥，该网桥可立即启动，**STP** 启用并延迟 **0**。

```
# virsh iface-bridge interface bridge --no-stp delay --no-start
```

请注意，这些设置可以使用 **--no-stp**、**--no-start** 和整数来延迟。接口的所有 **IP** 地址配置将移到新的网桥设备。有关停止网桥的信息，请参阅 第 14.15.1.8 节 “中断桥接设备”。

14.15.1.8. 中断桥接设备

iface-un bridge bridge --no-start 命令停止名为 **bridge** 的指定桥接设备，将其底层接口回滚到正常使用，并将所有 **IP** 地址配置从网桥设备移到底层设备。除非使用 **--no-start** 选项，但通常不建议重启底层接口。如需用于创建网桥的命令，请参阅 第 14.15.1.7 节 “创建网桥设备”。

14.15.1.9. 操控接口快照

iface-begin 命令可创建当前主机接口设置的快照，稍后可以提交（使用 **iface-commit**）或恢复 (**iface-rollback**)。如果快照已存在，则此命令将失败，直到之前的快照被提交或恢复为止。如果在创建快照及其最终提交或回滚之间对 **libvirt API** 之外的主机接口做任何外部更改，则未定义的行为将会导致。

使用 **iface-commit** 命令声明自上一次 **iface-begin** 所做的所有更改，然后删除回滚点。如果没有通过 **iface-begin** 启动接口快照，则此命令将失败。

使用 **iface-rollback** 将所有主机接口设置恢复到执行 **iface-begin** 命令最后一次时间的状态。如果之前未执行 **iface-begin** 命令，则 **iface-rollback** 将失败。请注意，重新引导主机物理机器也充当隐式回滚点。

14.15.2. 管理快照

以下小节描述了可以执行的操作，以便操作域快照。快照 在指定时间点获取域的磁盘、内存和设备状态，并保存它供以后使用。快照具有许多用途，从保存操作系统镜像的"干净"副本，以便在可能出现破坏性操作之前保存域的状态。快照与唯一名称进行标识。有关用于表示快照属性的 **XML** 格式的文档，请参阅 **libvirt 网站**。

14.15.2.1. 创建快照

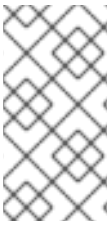
virsh snapshot-create 命令使用域 **XML** 文件中指定的属性（如 **<name>** 和 **<description>** 元素以

及 <disks>）为域创建快照。

要创建快照，请运行：

```
# snapshot-create <domain> <xmlfile> [--redefine] [--current] [--no-metadata] [--reuse-external]
```

域名、ID 或 UID 可用作域要求。XML 要求是一个字符串，必须包含 <name>、<description> 和 <disks> 元素。



注意

Red Hat Enterprise Linux 不支持实时快照。virsh snapshot-create 命令还有附加选项，可用于 libvirt 中可见但 Red Hat Enterprise Linux 6 中不支持的实时快照。

Red Hat Enterprise Linux 中可用的选项包括：

- --redefine 指定，如果 snapshot-dumpxml 生成的所有 XML 元素均有效；它可以用于将快照层次结构从一个机器迁移到另一台机器，以便为稍后使用相同名称和 UUID 的临时域重新创建，或可在快照元数据中更改更改（例如，特定于主机域在快照中的某些方面）。当提供这个选项时，必须使用 xmlfile 参数，并且也不会更改域的当前快照，除非也提供了 --current 选项。
- --no-metadata 创建快照，但任何元数据都被立即丢弃（即，libvirt 不会将快照视为当前的快照，除非 --redefine 稍后被用来再次教授 libvirt 关于元数据）。
- 如果使用 --reuse-external，则此选项指定要使用的现有外部 XML 快照的位置。如果现有外部快照尚不存在，命令将无法执行快照，以避免丢失现有文件的内容。

14.15.2.2. 为当前域创建快照

virsh snapshot-create-as domain 命令使用域 XML 文件中指定的属性（如 <name> 和 <description> 元素）为域创建快照。如果 XML 字符串中没有包括这些值，libvirt 将选择一个值。要创建快照运行，请执行以下操作：

```
# virsh snapshot-create-as domain {[--print-xml] | [--no-metadata] [--reuse-external]} [name]
[description] [--diskspec] diskspec]
```

其余选项如下：

- --print-xml 会为 snapshot-create 创建适当的 XML 作为输出，而不是实际创建快照。

- --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。这个选项可以根据域 XML 中的 <disk> 元素数量多次发生。每个 <diskspec> 都格式为 disk[,snapshot=type][,driver=type][,file=name]。要在磁盘或者 file=name 中包括字面逗号，请使用第二个逗号进行转义。除非还存在三个 <domain>、<name> 和 <description>，否则每个 diskspec 都需要有一个字面上的 --diskspec。例如，diskspec of vda,snapshot=external,file=/path/to,new 会导致以下 XML：

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new'/>
</disk>
```

- --reuse-external 会利用现有文件作为目标创建一个外部快照（文件名会被覆盖）。如果此目标不存在，则快照请求将被拒绝，以避免丢失现有文件的内容。

- --no-metadata 会创建快照数据，但任何元数据都被立即丢弃（即，libvirt 不会将快照视为当前的快照，除非快照创建稍后被用来再次教授 libvirt 的相关元数据）。这个选项与 --print-xml 不兼容。

14.15.2.3. 为当前域生成快照

此命令用于查询当前正在使用的快照。要使用，请运行：

```
# virsh snapshot-current domain {[--name] | [--security-info] | [snapshotname]}
```

如果没有使用 snapshotname，则域当前快照的快照 XML（如果有）将显示为输出。如果指定了 --name，则只有当前快照名称而不是完整的 XML 作为输出发送。如果提供了 --security-info，则 XML 中将包含安全敏感信息。使用 snapshotname，libvirt 生成一个请求，使现有命名快照成为当前快照，而不将其恢复为域。

14.15.2.4. snapshot-edit-domain

此命令用于编辑当前正在使用的快照。要使用，请运行：

```
#virsh snapshot-edit domain [snapshotname] [--current] {[--rename] [--clone]}
```

如果指定了 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，才能编辑当前的快照。

这等同于以下命令序列，但它还包括一些错误检查：

```
# 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，则生成的编辑的文件将保存在其他文件名称中。如果指定了 --clone，则更改快照名称将创建一个快照元数据的克隆。如果没有指定，则编辑不会更改快照名称。请注意，更改快照名称必须小心完成，因为某些快照的内容（例如单个 qcow2 文件中的内部快照）只能从原始快照文件名访问。

14.15.2.5. snapshot-info-domain

snapshot-info-domain 显示有关快照的信息。要使用，请运行：

```
# snapshot-info domain {snapshot | --current}
```

输出关于指定 快照 的基本信息，或使用 --current 的当前快照。

14.15.2.6. snapshot-list-domain

列出给定域的所有可用快照，默认为显示快照名称、创建时间和域状态的列。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] [--internal] [--
external]
```

剩余的可选选项如下：

- --parent 在输出表中添加一个列，提供每个快照的父级名称。此选项不能与 --roots 或 --tree 一起使用。
-

--roots 过滤列表，以仅显示没有父快照的快照。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。这三个选项是相互排斥的选项。此选项不能与 --roots 或 --parent 一起使用。

- --from 将列表过滤到作为给定快照的子项的快照；或者，如果提供了 --current，则会导致列表从当前快照开始。在隔离或使用 --parent 结合使用时，列表仅限于直接的子项，除非也存在 --descendants。与 --tree 一起使用时，对 --descendants 的使用会被简化。这个选项与 --roots 不兼容。请注意，-- from 或 --current 的起点不包含在列表中，除非也存在 --tree 选项。

- 指定了 --leaves，该列表将只过滤为没有子级的快照。同样，如果指定了 --no-leaves，则该列表将只过滤为带有子项的快照。（请注意，省略这两个选项时，如果提供这两个选项都会生成相同的列表或错误，具体取决于服务器是否识别这些选项，而过滤选项与 --tree 不兼容。

- 指定了 --metadata，该列表将过滤为仅包含 libvirt 元数据的快照，从而防止意外域意外，或者丢失了临时域的销毁。同样，如果指定了 --no-metadata，则该列表将过滤为仅过滤为存在的快照，而无需 libvirt 元数据。

- 指定了 --inactive，该列表将过滤为在域关闭时所执行的快照。如果指定了 --active，则列表将过滤为在域运行时所执行的快照，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，则列表将过滤为在域运行时所执行的快照，但快照只包括磁盘状态。

- 指定了 --internal，该列表将过滤成使用现有磁盘镜像的内部存储的快照。如果指定了 --external，则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。

14.15.2.7. snapshot-dumpxml domain snapshot

virsh snapshot-dumpxml 域 快照输出域的名为 snapshot 的快照 XML。要使用，请运行：

```
# virsh snapshot-dumpxml domain snapshot [--security-info]
```

security-info 选项也将包含安全敏感信息。使用 snapshot-current 轻松访问当前快照的 XML。

14.15.2.8. snapshot-parent 域

输出父快照的名称（如果有），或者针对给定快照的当前快照，或使用 --current 输出当前快照的名

称。要使用，请运行：

```
#virsh snapshot-parent domain {snapshot | --current}
```

14.15.2.9. snapshot-revert 域

将给定域恢复到快照指定的 快照，或使用 --current 恢复到当前快照。

警告

请注意，这是破坏性操作；自上次执行快照以来，任何更改都将丢失。另请注意，在 snapshot-revert 完成后域的状态将是生成原始快照时域的状态。

要恢复快照，请运行

```
# snapshot-revert domain {snapshot | --current} [{--running | --paused}] [--force]
```

通常，恢复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不活动状态。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运行的域）。由于临时域无法激活，因此当恢复到临时域的磁盘快照时，需要使用这些选项之一。

快照恢复 涉及额外的风险的两个情况下，需要使用 --force 才能继续操作。快照缺少用于恢复配置的完整域信息的一个情况；因为 libvirt 无法证明当前配置在快照时使用的内容匹配，请提供 --force s libvirt，与当前配置兼容（如果不是，则域将无法运行）。另一个情况是从正在运行的域恢复到活跃的状态，其中必须创建新的管理程序而不是重复使用现有虚拟机监控程序，因为它代表了破坏任何现有 VNC 或 Spice 连接的缺陷；这种条件发生于使用合理的不兼容的配置的活跃快照，以及 --start 或 --pause 选项组合的不活跃快照。

14.15.2.10. snapshot-delete 域

snapshot-delete 域 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 删除当前快照。如果此快照有子快照，

则来自此快照的更改将合并到子项中。如果使用 --children 选项，它将删除此快照以及此快照的任何子项。如果使用 --children-only，则它将删除此快照的任何子项，但此快照保持不变。这两个选项是相互排斥的选项。

使用 --metadata，它将删除 libvirt 维护的快照元数据，而将快照内容保留给外部工具的访问；否则删除快照也会从该时间点删除其数据内容。

14.16. 客户机虚拟机 CPU 型号配置

这部分提供有关客户机虚拟机 CPU 模型配置的信息。

14.16.1. 简介

每个虚拟机监控程序都拥有自己的策略，适用于客户机虚拟机默认查看其 CPU。有些虚拟机监控程序决定了 guest 虚拟机可以使用哪些 CPU 主机物理机器功能，QEMU/KVM 则呈现客户机虚拟机，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 分为几个组，并为虚拟客户机呈现的每个组都有一个基准 CPU 模型。这样的行为使在主机物理计算机之间安全迁移虚拟客户机，只要它们都有将物理 CPU 分为同一组。libvirt 通常不会强制实施策略本身，而是提供较高层定义自己所需的策略的机制。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，以确保客户机虚拟机在主机物理计算机之间成功迁移至关重要。请注意，管理程序只能模拟它了解的功能以及在虚拟机监控程序发布的后创建的功能可能无法模拟。

14.16.2. 了解主机物理机器 CPU 模型

virsh capabilities 命令显示描述管理程序连接和主机物理机器的 XML 文档。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 模型的一个最大挑战是，每个架构都有不同的方法来公开其功能。在 x86 上，现代 CPU 的功能通过 CPUID 指令公开。本质上，这分为一组 32 位整数，每个位都给出一个具体含义。幸运的是，AMD 和 Intel 同意这些位的通用语义。其他虚拟机监控程序以客户机虚拟机配置格式直接公开 CPUID 掩码的概念。然而，QEMU/KVM 不仅支持 x86 架构，因此 CPUID 显然不适合规范配置格式。QEMU 使用结合 CPU 型号名称字符串的方案以及一组指定选项的方案结束。在 x86 上，CPU 模型映射到基准 CPUID 掩码，并可使用选项在掩码上切换或关闭位。libvirt 决定按照此潜在客户并使用模型名称和选项的组合。

数据库列表不是所有已知的 CPU 型号，因此 libvirt 具有少量的基准 CPU 模型名称列表。它选择与实际主机物理机器 CPU 共享最大数量的 CPUID 位，然后按照指定功能列出剩余的位。请注意，libvirt 不显示基准 CPU 包含的功能。这可能与第一个漏洞类似，但本节将对此进行说明，实际上并不需要了解此信息。

14.16.3. 确定兼容的 CPU 型号以 Suit a Pool of Host Physical Machines

现在，可以找出单个主机物理机器拥有的 CPU 功能，下一步是确定哪些 CPU 功能最适合公开给客户机虚拟机。如果已知 guest 虚拟机绝不需要迁移到另一台主机物理机器，则主机物理机器 CPU 模型可以

直接通过未经修改的方式进行传递。虚拟化数据中心可能具有一组配置，可保证所有服务器具有 100% 相同的 CPU。再次通过未修改的方式传递主机物理机器 CPU 模型。尽管如此，但 CPU 在主机物理机器之间有变化的情况。在这种混合 CPU 环境中，必须确定最低共用的 denominator CPU。这不是完全直接的，因此 libvirt 提供一个 API 来执行此任务。如果 libvirt 提供了 XML 文档列表，每个描述主机物理机器的 CPU 模型，libvirt 将在内部将它们转换为 CPUID 掩码，计算它们的交集，并将 CPUID 掩码结果重新转换为 XML CPU 描述。

以下是执行 virsh 功能时，作为基本工作站功能的libvirt 报告的示例：

图 14.3. 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，与任何随机服务器比较，使用相同的 virsh capabilities 命令：

图 14.4. 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 描述是否与先前工作站 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
```

如此输出中所示，libvirt 正确报告 CPU 不严格兼容。这是因为客户端 CPU 中缺少了服务器 CPU 中的一些功能。为了能够在客户端和服务器之间迁移，需要打开 XML 文件并注释掉某些功能。要确定需要删除哪些功能，请在包含两台机器的 CPU 信息的 both-cpus.xml 上运行 virsh cpu-baseline 命令。运行 # virsh cpu-baseline both-cpus.xml，结果如下：

图 14.5. 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。并非常见内容应该被注释掉。

14.17. 配置客户机虚拟机 CPU 型号

对于简单默认值，guest 虚拟机 CPU 配置接受与主机物理机器功能 XML 公开相同的基本 XML 表示。换句话说，cpu-baseline virsh 命令的 XML 现在可以直接复制到 <domain> 元素的顶级的客户机虚拟机 XML 中。 在前面的 XML 片段中，在描述客户机虚拟机 XML 中的 CPU 时，有几个额外的属性可用。这些主要可以忽略，但是对他们有什么作用是非常快速的描述。顶级 <cpu> 元素具有名为 match 的属性，可能的值如下：

- *match='minimum' - 主机物理机器 CPU 必须至少有 guest 虚拟机 XML 中描述的 CPU 功能。如果主机物理计算机除客户机虚拟机配置外的其他功能，则也会向客户机虚拟机公开这些功能。*
- *match='exact' - 主机物理机器 CPU 必须至少有 guest 虚拟机 XML 中描述的 CPU 功能。如果主机物理计算机除客户机虚拟机配置之外的其他功能，则这些功能将从 guest 虚拟机中屏蔽。*
- *match='strict' - 主机物理机器 CPU 必须具有客户机虚拟机 XML 中描述的 CPU 功能完全相同。*

下一个改进是 <feature> 元素可以有一个额外的 'policy' 属性，可能的值如下：

- *policy='force' - 即使主机物理机器没有它，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使用。*

- policy='require' - 将功能公开给客户机虚拟机，如果主机物理计算机未提供，则失败。这是允许的默认值。
- policy='optional' - 如果出现支持它，则向客户机虚拟机公开功能。
- policy='disable' - 如果主机物理机器有此功能，则从客户机虚拟机中隐藏它。
- policy='forbid' - 如果主机物理计算机拥有此功能，则会失败并拒绝启动客户机虚拟机。

'forbid' 策略适用于一个有机率的场景，错误运行的应用程序会尝试使用功能，即使它不在 CPUID 掩码中，并且您希望防止在具有该功能的主机物理机器上意外运行 guest 虚拟机。'optional' 策略对迁移有特殊行为。当 guest 虚拟机最初启动该参数为可选时，但当客户机虚拟机实时迁移时，此策略会变为 'require'，因为您在迁移之间无法消失功能。

14.18. 管理客户机虚拟机的资源

virsh 允许基于每个客户机虚拟机对资源进行分组和分配。这由 libvirt 守护进程管理，它代表客户机虚拟机创建 cgroups 并管理它们。系统管理员唯一剩下可以查询或设置对指定客户机虚拟机的可调项。可使用以下可调项：

- Memory - 内存控制器允许对 RAM 和 swap 使用量设置限制，并查询组中所有进程的累积用量
- Cpuset - CPU 设置控制器将组中的进程绑定到一组 CPU 并控制 CPU 之间的迁移。
- cpuacct - CPU 记帐控制器为一组进程提供 CPU 使用量的信息。
- cpu - CPU 调度程序控制器控制组中的进程的优先级。这类似于授予 nice 级别特权。
- devices - 设备控制器在字符和块设备上授予访问控制列表。
- 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。

在创建组层次结构 cgroup 会将挂载点和目录设置完全保留为管理员的自由裁量，而不只是向 /etc/fstab 添加一些挂载点。需要设置目录层次结构，并且决定进程如何放入其中。这可以通过以下 virsh 命令完成：

- schedinfo - 所述 第 14.19 节 “设置调度参数”
- blkiotune- 如下所述 第 14.20 节 “显示或设置块 I/O 参数”
- domiftune- 所述 第 14.5.9 节 “设置网络接口带宽参数”
- memtune - 所述 第 14.21 节 “配置内存调整”

14.19. 设置调度参数

schedinfo 允许将调度程序参数传递给客户机虚拟机。应使用以下命令格式：

```
#virsh schedinfo domain --set --weight --cap --current --config --live
```

以下是每个参数的信息：

- 域 - 这是客户机虚拟机域
- --set - 此处放置的字符串是要调用的控制器或操作。如果需要，还应添加其他参数或值。
- --current - 与 --set 一起使用时，将使用指定的 set 字符串作为当前的调度程序信息。当在没有 的情况下使用时，将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 集合 字符串。当在没有 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，将在当前运行的虚拟客户机上使用指定 的集合 字符串。当在没有 的情况下使用时，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。

例 14.5. schedinfo show

本例显示了 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 14.6. schedinfo set

在本例中， cpu_shares 更改为 2046。这会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
```

14.20. 显示或设置块 I/O 参数

blkiotune 设置和显示指定 guest 虚拟机的 I/O 参数.使用以下格式：

```
# virsh blkiotune domain [--weight weight] [--device-weights device-weights] [[--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

14.21. 配置内存调整

在 Virtualization Tuning and Optimization Guide 中介绍了 virsh memtune virtual_machine --

parameter 大小。

14.22. 虚拟网络命令

以下命令操作虚拟网络。libvirt 具有用于定义虚拟网络的功能，然后可由域使用并链接到实际的网络设备。有关此功能的详情，请查看 libvirt 网站 的文档。虚拟网络的许多命令与用于域的命令相似，但将虚拟网络命名为 或 UUID。

14.22.1. 自动启动虚拟网络

此命令将将虚拟网络配置为在 guest 虚拟机启动时自动启动。要运行这个命令：

```
# virsh net-autostart network [--disable]
```

这个命令接受 --disable 选项，该选项禁用 autostart 命令。

14.22.2. 从 XML 文件创建虚拟网络

此命令从 XML 文件创建虚拟网络。请参阅 libvirt 的网站 以获取 libvirt 使用的 XML 网络格式的描述。在这个命令 文件中，这是 XML 文件的路径。要从 XML 文件创建虚拟网络，请运行：

```
# virsh net-create file
```

14.22.3. 从 XML 文件定义虚拟网络

此命令从 XML 文件定义虚拟网络，仅定义网络，而不实例化。要定义虚拟网络，请运行：

```
# net-define file
```

14.22.4. 停止虚拟网络

此命令销毁（停止）由其名称或 UUID 指定的特定虚拟网络。这会立即生效。要停止指定的网络，则需要停止指定的 网络。

```
# net-destroy network
```

14.22.5. 创建转储文件

此命令输出虚拟网络信息，作为指定虚拟网络的 XML 转储到 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。要创建转储文件，请运行：

```
# virsh net-dumpxml network [--inactive]
```

14.22.6. 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。

14.22.7. 获取有关虚拟网络的信息

此命令返回有关 网络对象 的基本信息。要获取网络信息，请运行：

```
# virsh net-info network
```

14.22.8. 列出有关虚拟网络的信息

如果指定了 --all，则返回活动网络列表，它将包括已定义但 不活跃的 网络（如果仅指定了非活动网络）。您可能还希望通过 --persistent 过滤返回的网络，以列出由 --transient 来列出临时的临时网络，--autostart 列出那些启用自动启动的功能，--no-autostart 可以列出自动启动禁用状态。

注意：当与旧服务器进行通信时，此命令会被强制使用一类 API 调用来固有竞争，如果池在收集列表时在调用时可能会显示，也可能显示一次。较新的服务器没有这个问题。

要列出虚拟网络，请运行：

```
# net-list [--inactive | --all] [--persistent] [<--transient>] [--autostart] [<--no-autostart>]
```

14.22.9. 将网络 UUID 转换为网络名称

此命令将网络 UUID 转换为网络名称。为此，请运行：

```
# virsh net-name network-UUID
```

14.22.10. 启动（之前定义的） inactive Network

此命令启动（之前定义的）非活动网络。为此，请运行：

```
# virsh net-start network
```

14.22.11. 取消定义非主动网络的配置

此命令取消定义不活跃网络的配置。为此，请运行：

```
# net-undefine network
```

14.22.12. 将网络名称转换为网络 UUID

此命令将网络名称转换为网络 UUID。为此，请运行：

```
# virsh net-uuid network-name
```

14.22.13. 更新现有网络定义文件

此命令将更新现有网络定义的给定部分，并立即生效，而无需销毁和重新启动网络。这个命令是 "add-first", "add-last", "add"（一个用于 add-last）、"delete"或"modify"。部分是""bridge", "domain", "ip-dhcp-host", "ip-dhcp-host", "ip-dhcp-range", "forward", "forward-interface", "forward-interface", "forward-pf", "portgroup", "dns-host", "dns-txt" 或 "dns-srv"，每个部分都由 xml 元素层次结构的串联命名，从而导致更改元素。例如，" <ip> -dhcp-host"将更改包含在 network. 元素内的 <dhcp> 元素内的 <主机> 元素。xml 是所更改类型的完整 xml 元素的文本（例如 "<host mac="00:11:33:33:44:55' ip='192.0.2.1'）或包含完整 xml 元素的文件的名称。不清清是通过查看提供的文本的第一个字符 - 如果第一个字符为"<"，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。--parent-index 选项指定所请求元素（基于 0）的多个父元素。例如，dhcp <主机> 元素可以位于网络中多个 <ip> 元素之一；如果未提供父索引，则最合适的"最合适的" <ip> 元素将被选择（通常是已具有 <dhcp> 元素的一个选项），但如果提供了 --parent-index，<ip> 的特定实例将获得修改。如果指定了 --live，则会影响正在运行的网络。如果指定了 --config，则会影响下一次持久性卷启动。如果指定了 -- current，则会影响当前的网络状态。可以同时提供 --live 和 --config 选项，但 --current 是独占的。不指定任何选项与指定 --current 相同。

要更新配置文件，请运行：

```
# virsh net-update network command section xml [--parent-index index] [[--live] [--config] | [--current]]
```

第 15 章 使用虚拟机管理器(VIRT-MANAGER)管理 GUEST.

这部分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系统上虚拟机监控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：

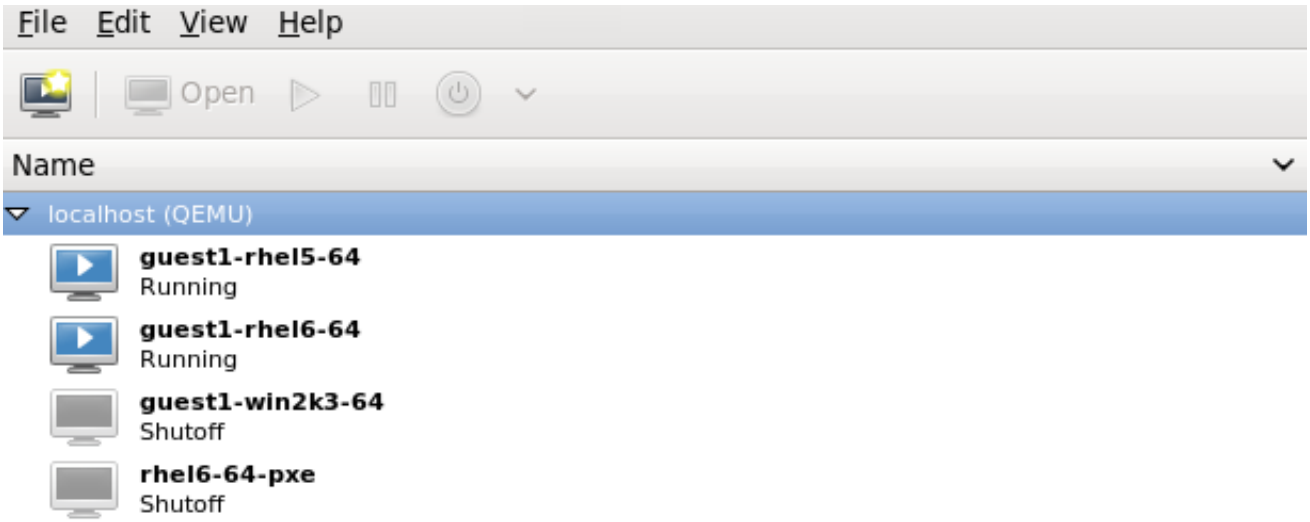
- 定义和创建客户机、
- 分配内存，
- 分配虚拟 CPU、
- 监控操作性能、
- 保存和恢复、暂停和恢复以及关闭和启动客户机、
- 链接到文本和图形控制台，以及
- 实时和离线迁移。

15.1. 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 15.1. 启动 virt-manager



或者，您可以使用 ssh 远程启动 virt-manager，如下命令所示：

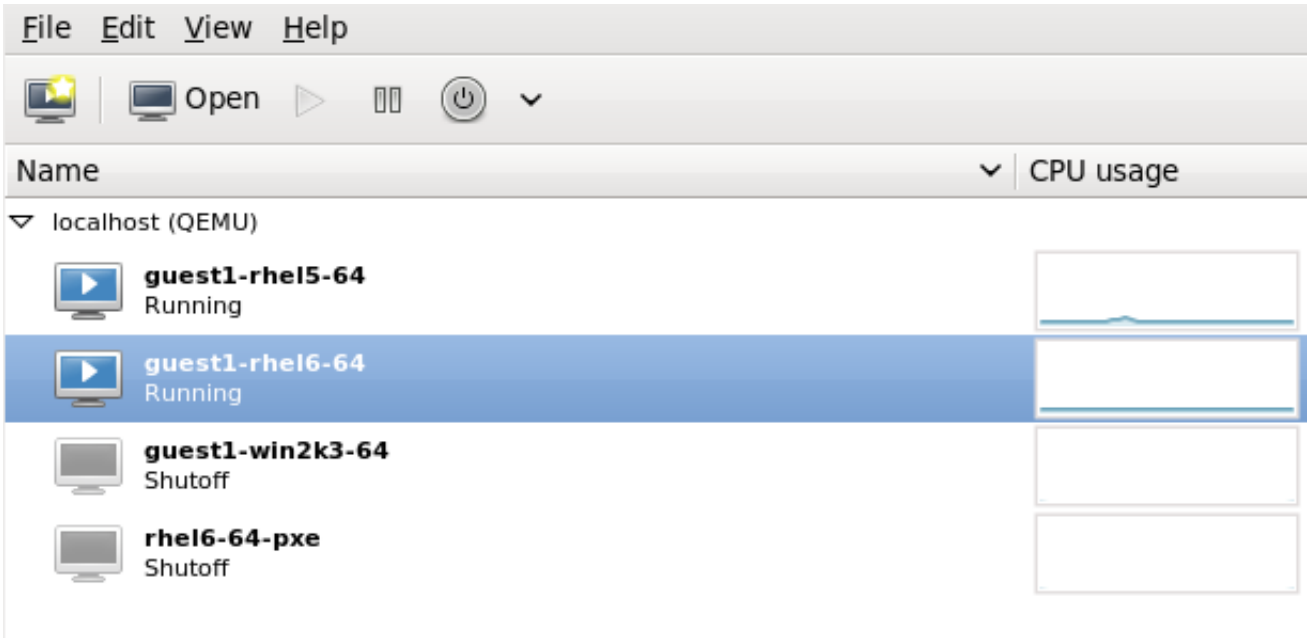
```
ssh -X host's address
[remotehost]# virt-manager
```

在 第 5.1 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。

15.2. VIRTUAL MACHINE MANAGER MAIN 窗口

此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 的名称来选择 guest。

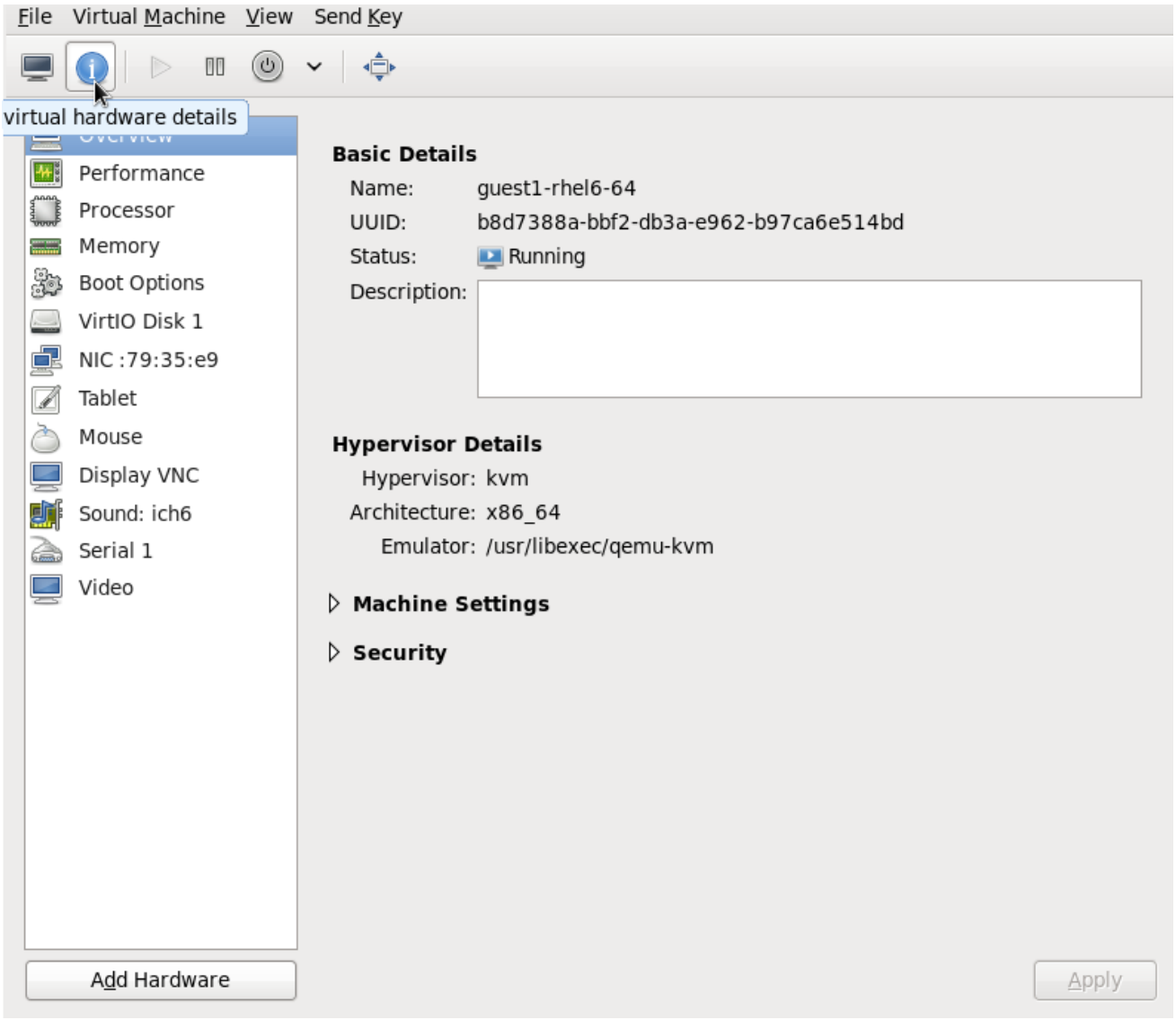
图 15.2. 虚拟机管理器主窗口



15.3. VIRTUAL HARDWARE DETAILS 窗口

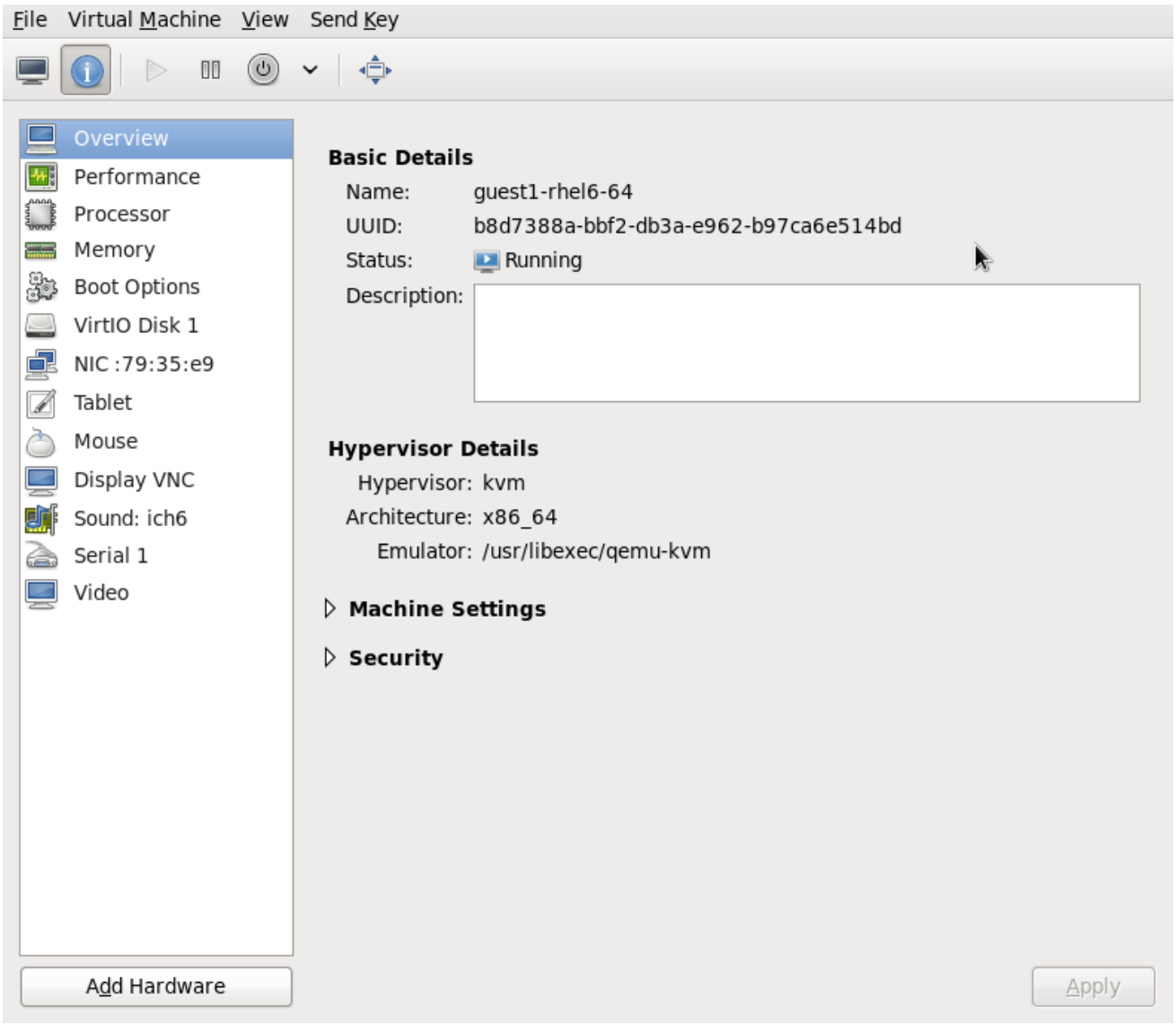
虚拟硬件详细信息窗口显示有关为客户机配置的虚拟硬件的信息。在此窗口中，可以添加、删除和修改虚拟硬件资源。要访问虚拟硬件详细信息窗口，请点击工具栏中的图标。

图 15.3. 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 15.4. 虚拟硬件详情窗口



15.3.1. 将 USB 设备附加到虚拟机

注意

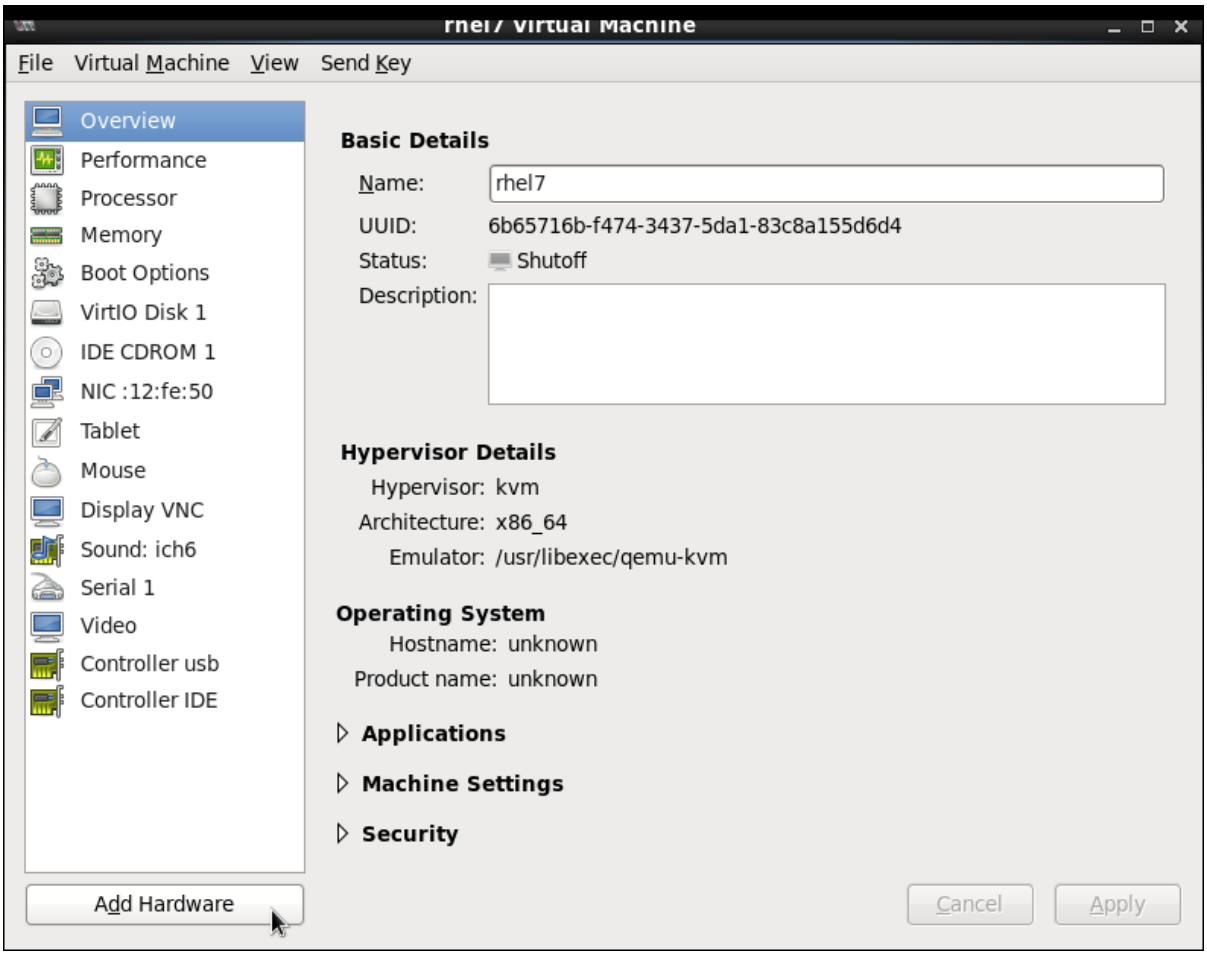
要将 USB 设备附加到虚拟机，首先必须将其附加到主机物理机器，并确认该设备正常工作。如果 guest 正在运行，则需要在继续之前将其关闭。

过程 15.1. 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

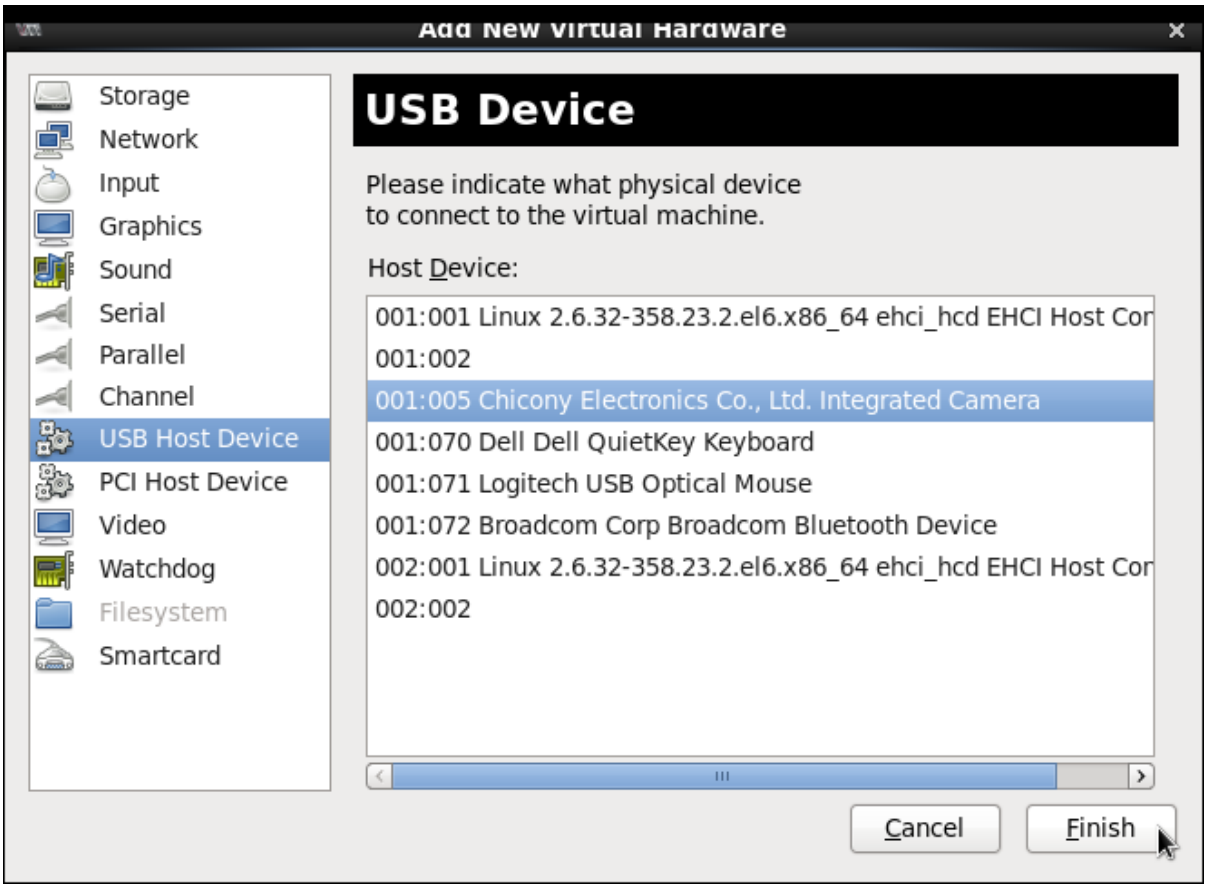
图 15.5. 添加硬件按钮



3.

在 Add New Virtual Hardware 弹出窗口中，选择 USB Host Device，从列表中选择您要附加的设备并点击 Finish。

图 15.6. 添加 USB 设备

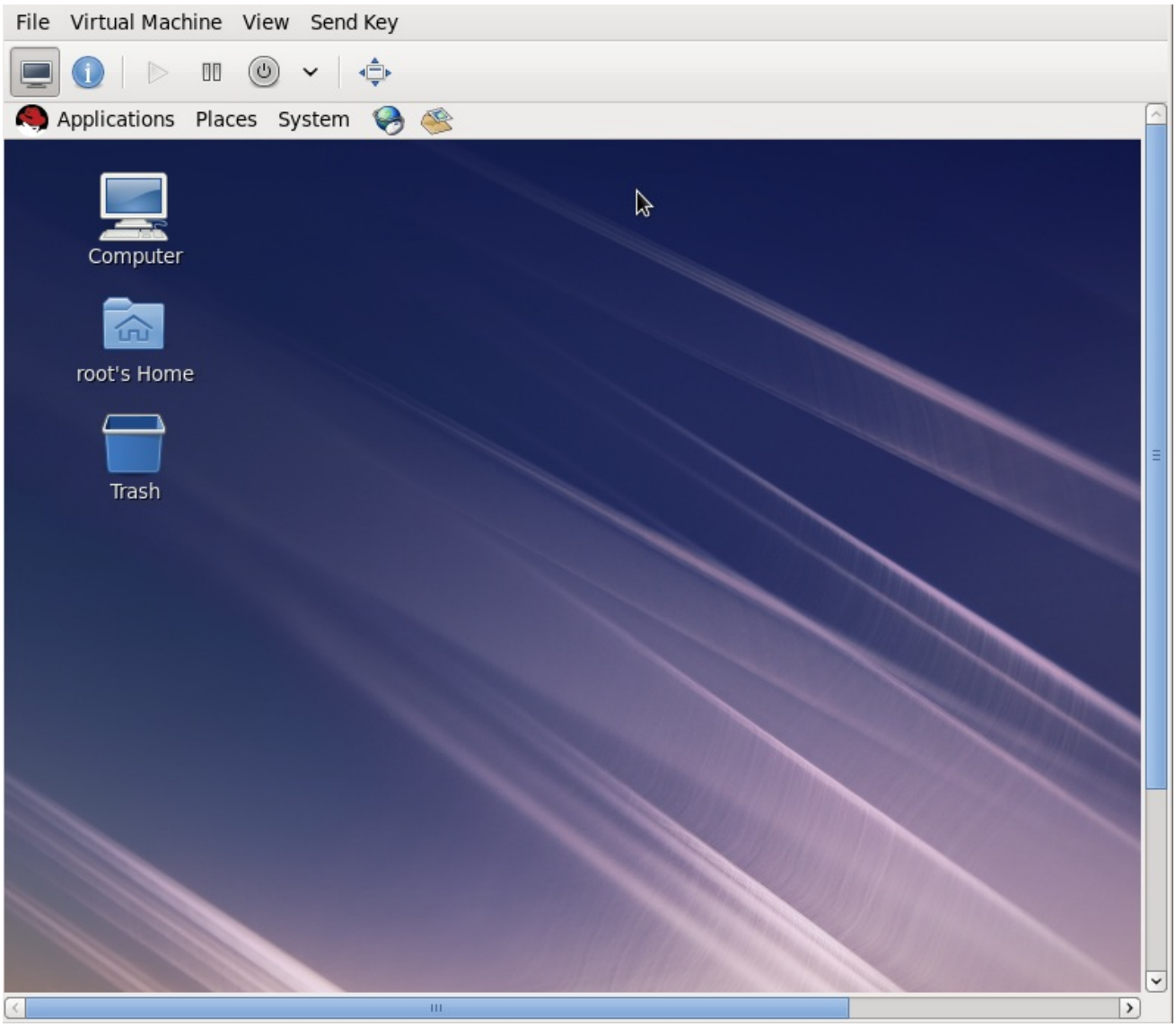


4. *要使用客户端虚拟机中的 USB 设备，请启动 guest 虚拟机。*

15.4. 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： 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 图形控制台会提示您输入密码，然后显示显示。

图 15.7. 图形控制台窗口



注意

VNC 被视为不受许多安全专家的不安全，但已进行了一些更改，以在 Red Hat Enterprise Linux 上为虚拟化启用 VNC 安全使用。客户机计算机仅侦听本地主机的回送地址(127.0.0.1)。这样可保证只有具有 shell 权限的那些可以通过 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替代方法，但不推荐这样做。

远程管理可以通过 SSH 来进行，这可加密流量。虽然 VNC 可以配置为在不通过 SSH 进行隧道的情况下远程访问，但出于安全原因，我们不建议这样做。要远程管理客户机，请按照以下说明进行：第 5 章 客户机的远程管理TLS 可以为客户机和主机系统提供企业级安全性。

您的本地桌面可截获组合键（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 键菜单选项来发送这些序列。在客户机机器窗口中，单击 Send key 菜单，然后选择要发送的密钥

序列。另外，您还可以从此菜单中捕获屏幕输出。

SPICE 是 Red Hat Enterprise Linux 可使用的 VNC 的替代方案。

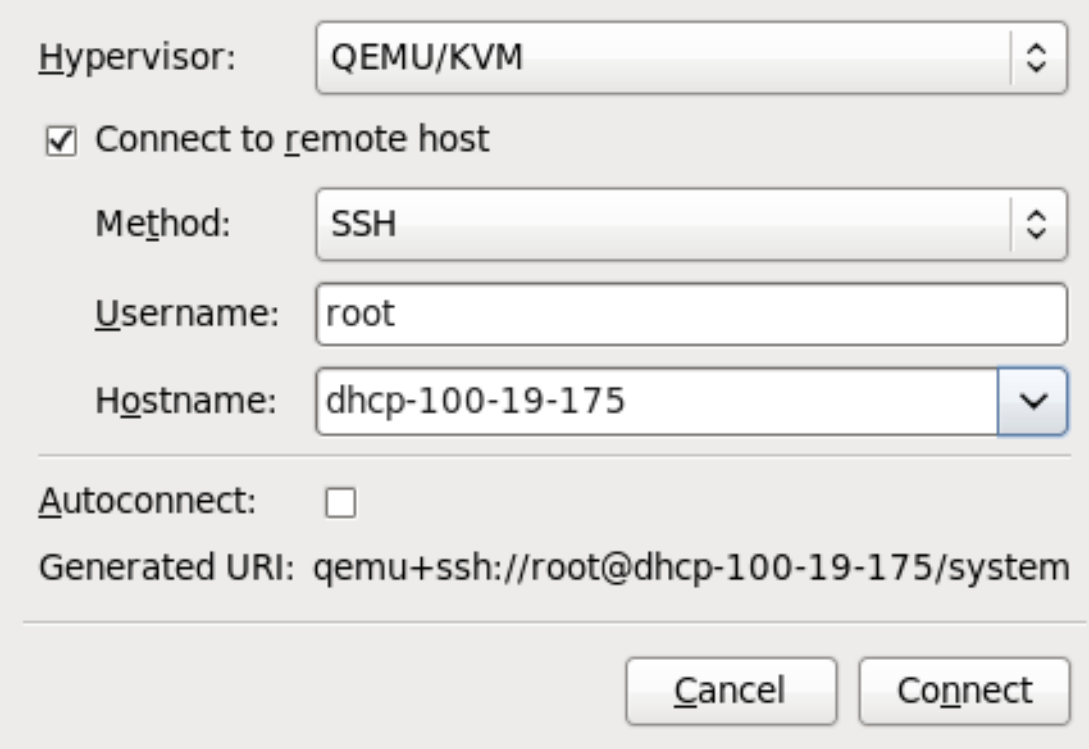
15.5. 添加远程连接

此流程介绍了如何使用 virt-manager 设置到远程系统的连接。

1. 要创建新连接，打开 File 菜单，然后选择 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 6 系统，选择 QEMU/KVM。为本地系统或其中一个远程连接选项选择 Local，然后单击 连接。这个示例通过 SSH 使用远程隧道，这适用于默认安装。有关配置远程连接的详情，请参考 第 5 章 客户机的远程管理

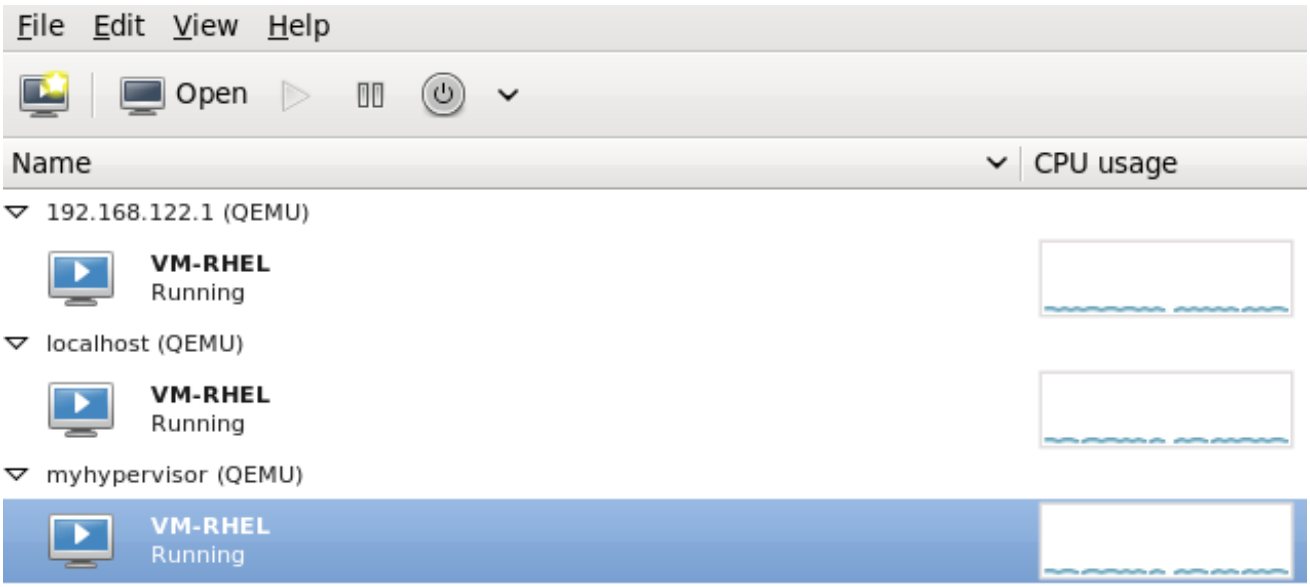
   图 15.8. 添加连接



3. 提示时为所选主机输入 root 密码。

远程主机现在已连接并出现在主 virt-manager 窗口中。

图 15.9. 在主 virt-manager 窗口中的远程主机



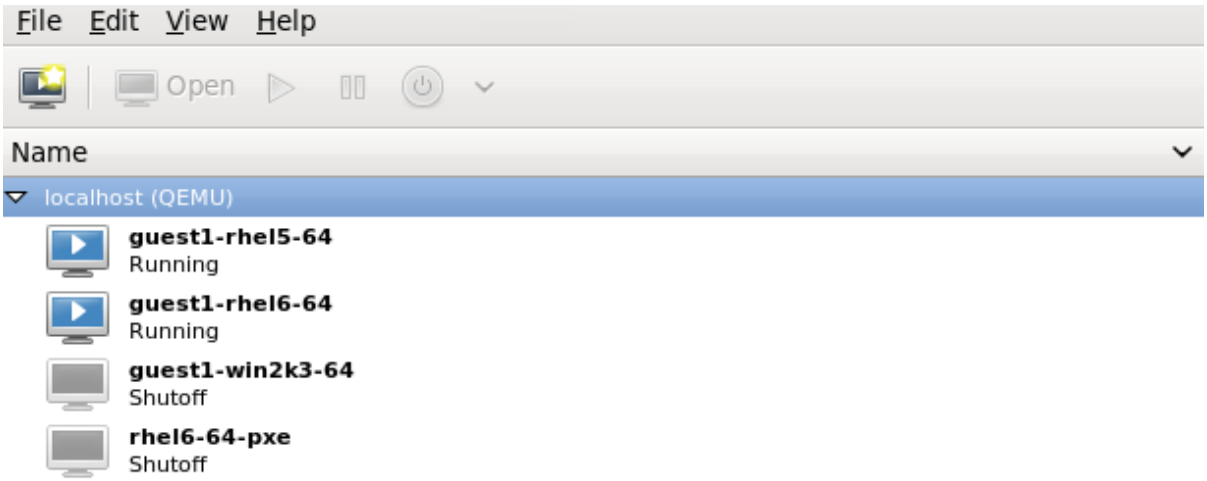
15.6. 显示客户机详情

您可以使用 Virtual Machine Monitor 来查看系统上任何虚拟机的活动信息。

查看虚拟系统的详情：

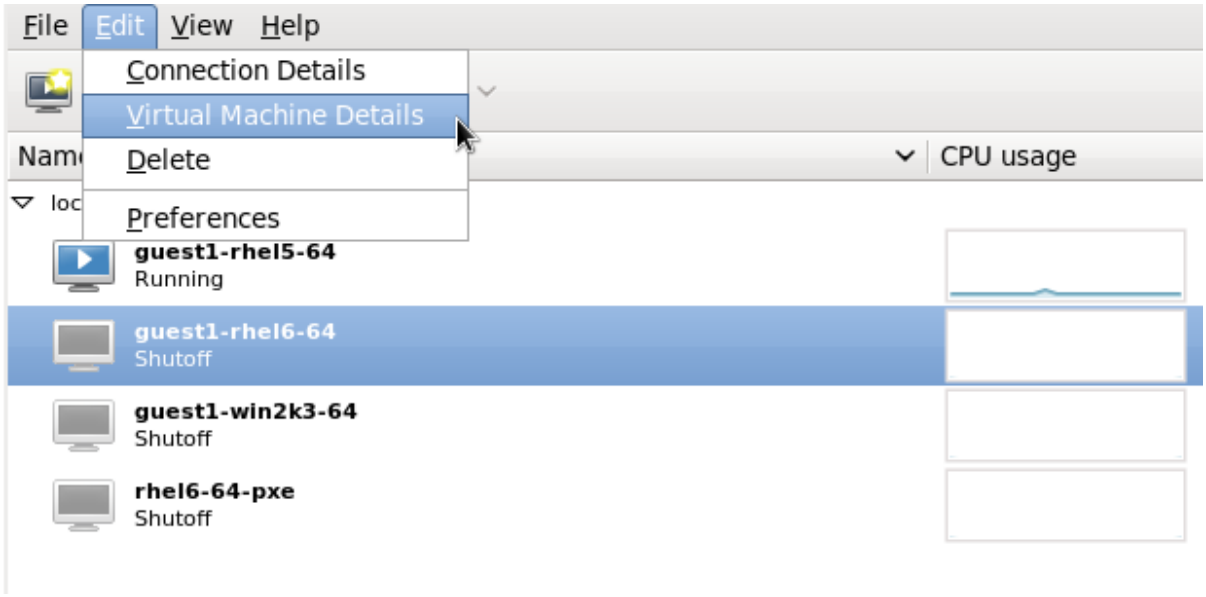
1. *在 Virtual Machine Manager 主窗口中，突出显示您要查看的虚拟机。*

   图 15.10. 选择要显示的虚拟机

2. *在 Virtual Machine Manager Edit 菜单中，选择 Virtual Machine Details。*

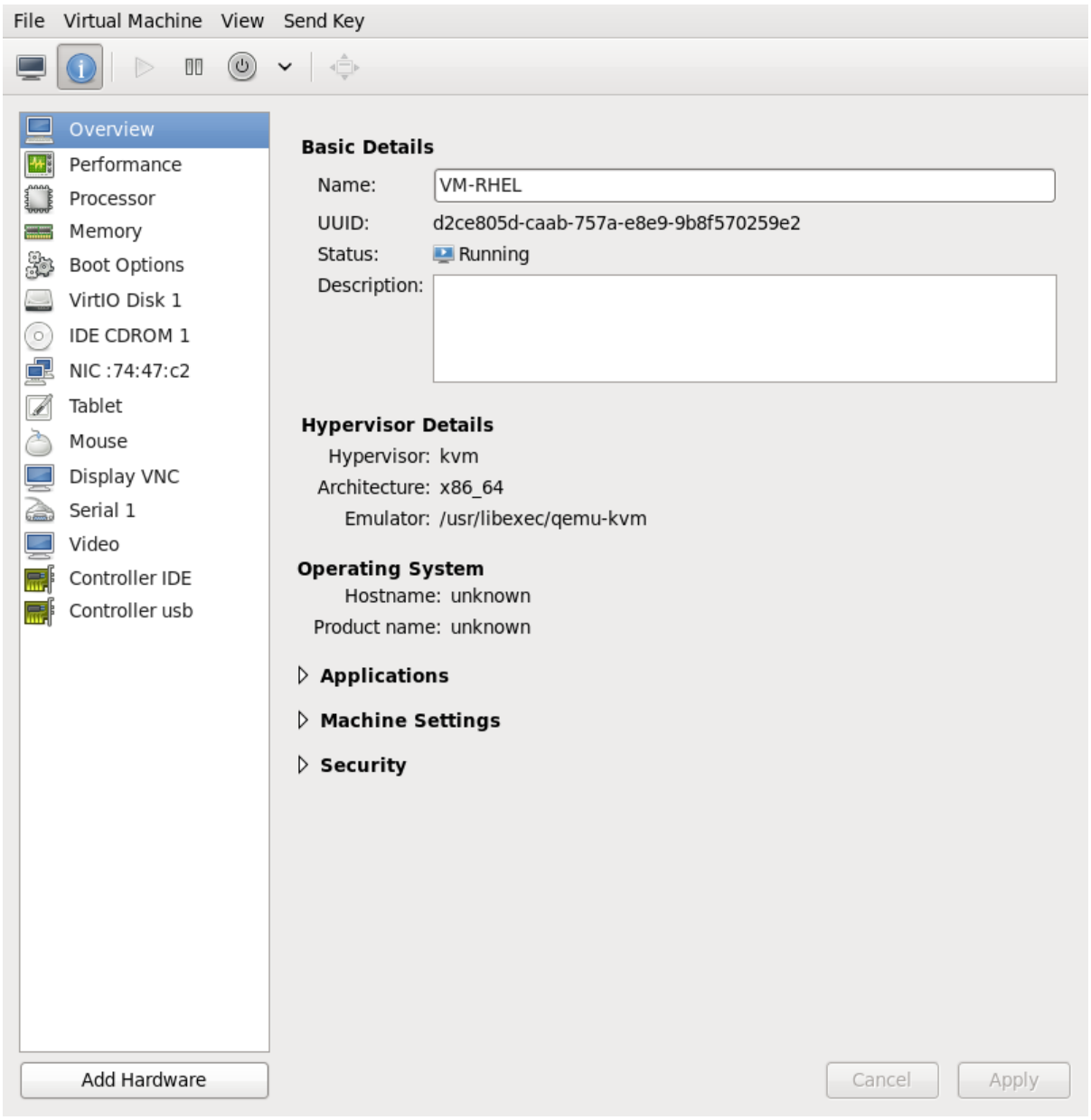
图 15.11. 显示虚拟机详情



当 Virtual Machine 详情窗口打开时，可能会显示控制台。如果发生这种情况，请单击" 查看 "，然后选择" 详细信息 "。默认会首先打开 Overview 窗口。要返回这个窗口，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 15.12. 显示客户机详情概述

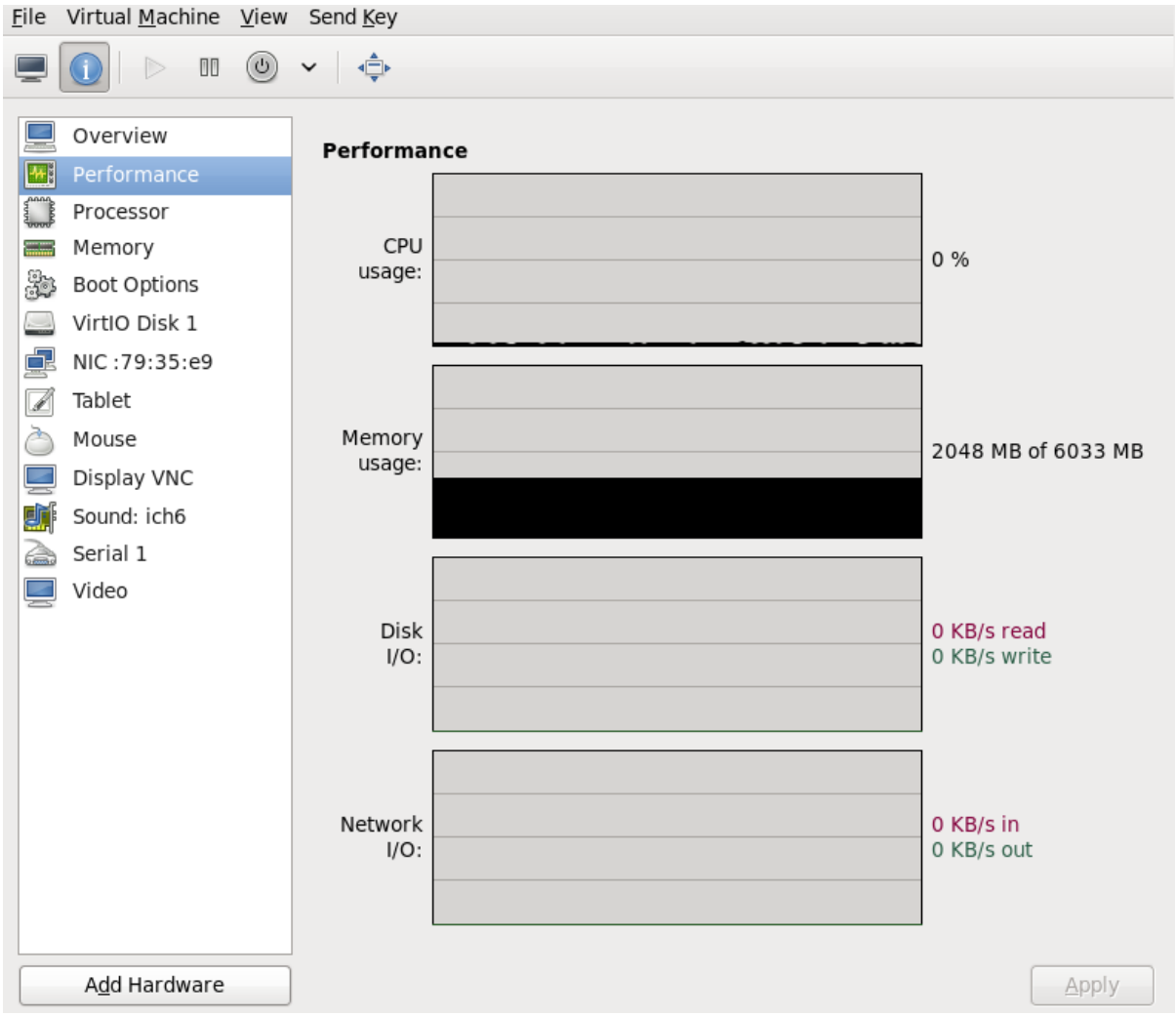


3.

从左侧的导航窗格中选择 Performance。

性能 视图显示客户机性能的摘要，包括 CPU 和内存使用情况。

图 15.13. 显示客户机性能详情



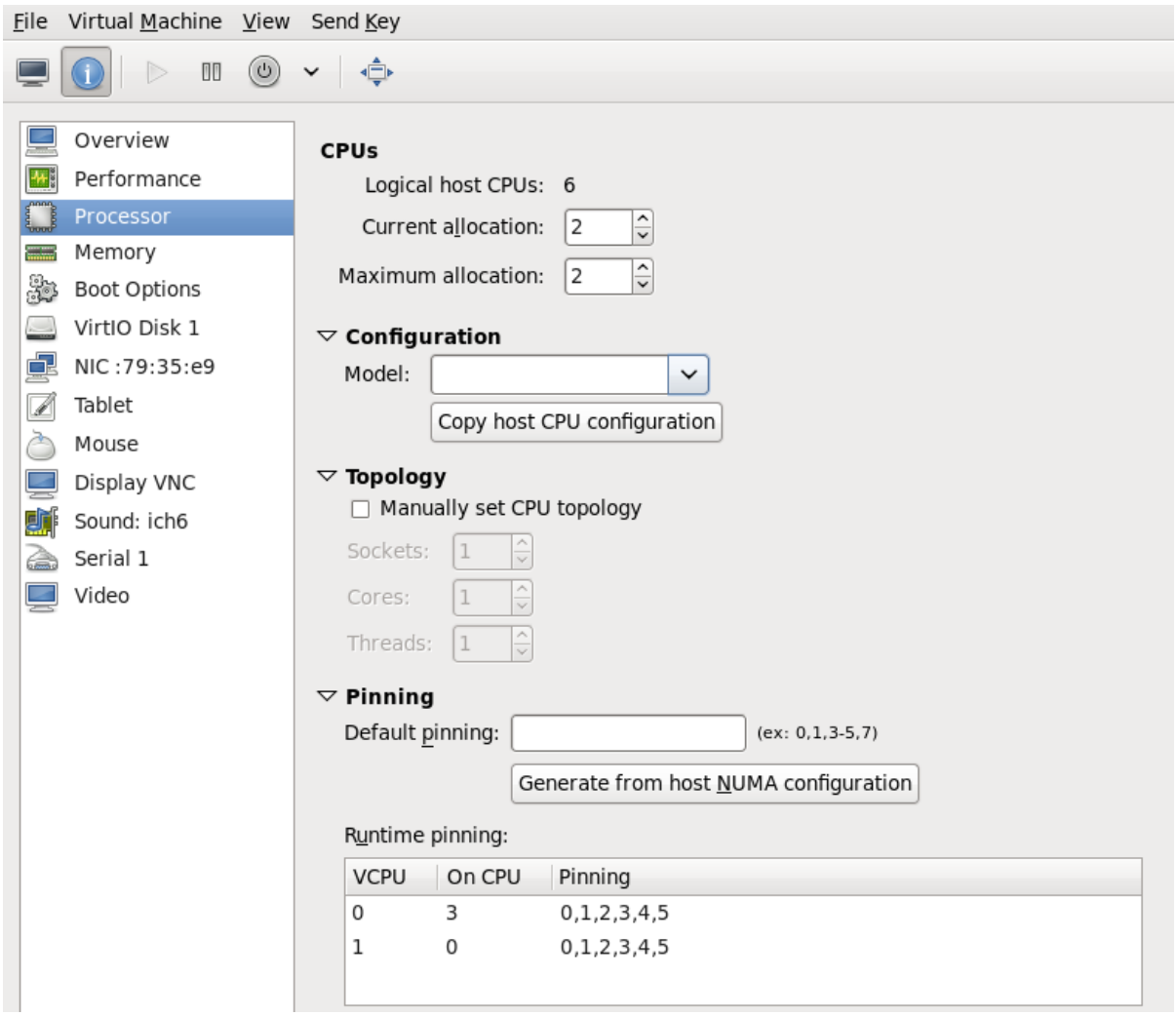
4. *从左侧的导航窗格中选择 Processor。Processor 视图允许您查看当前的处理器分配，以及更改它。*

虚拟机正在运行时，也可以更改虚拟 CPU(vCPU)的数量，这称为 热插拔 和热拔。

重要

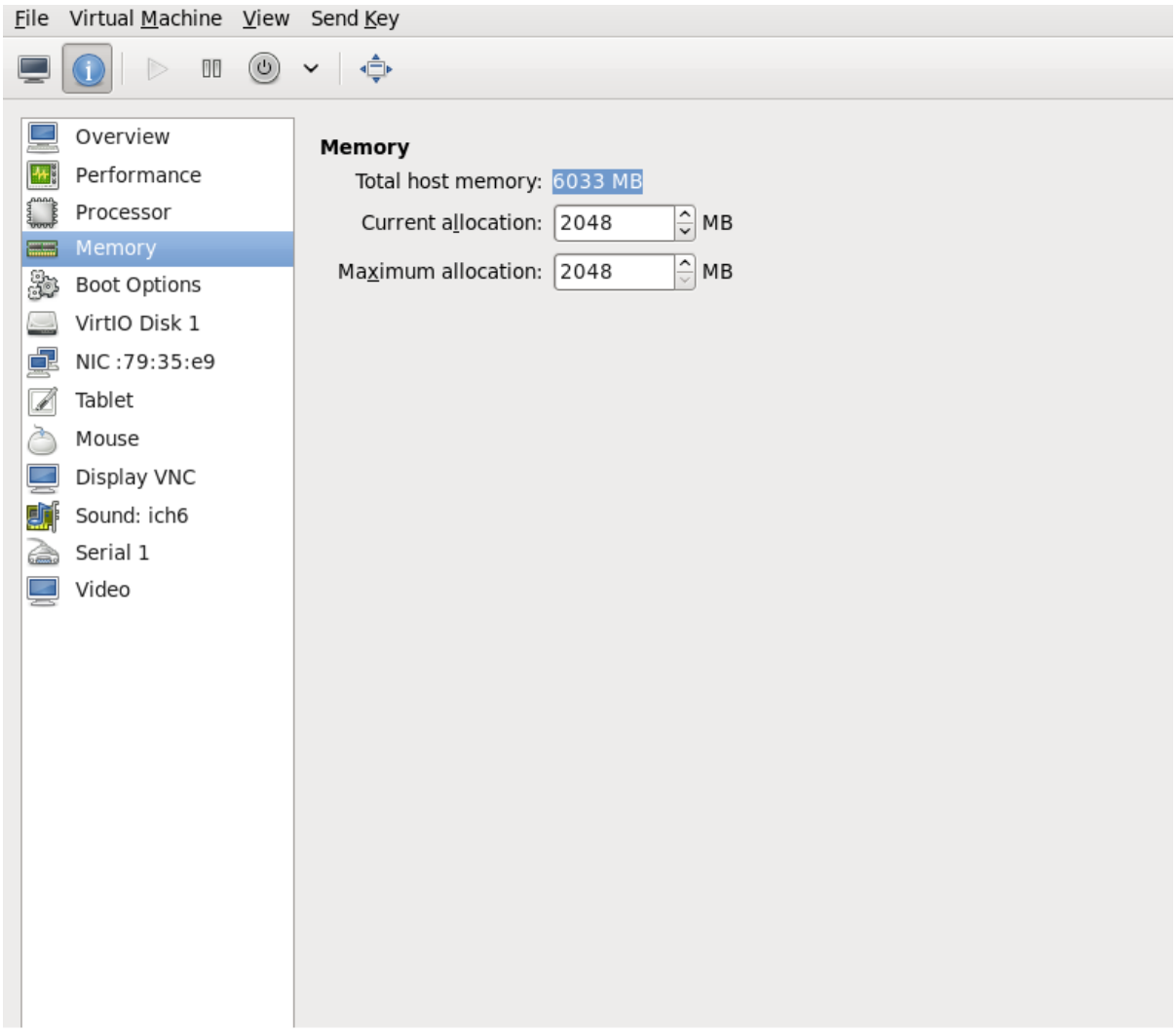
热拔功能仅作为技术预览提供。因此，它不被支持，且不建议在高值部署中使用。

图 15.14. 处理器分配面板



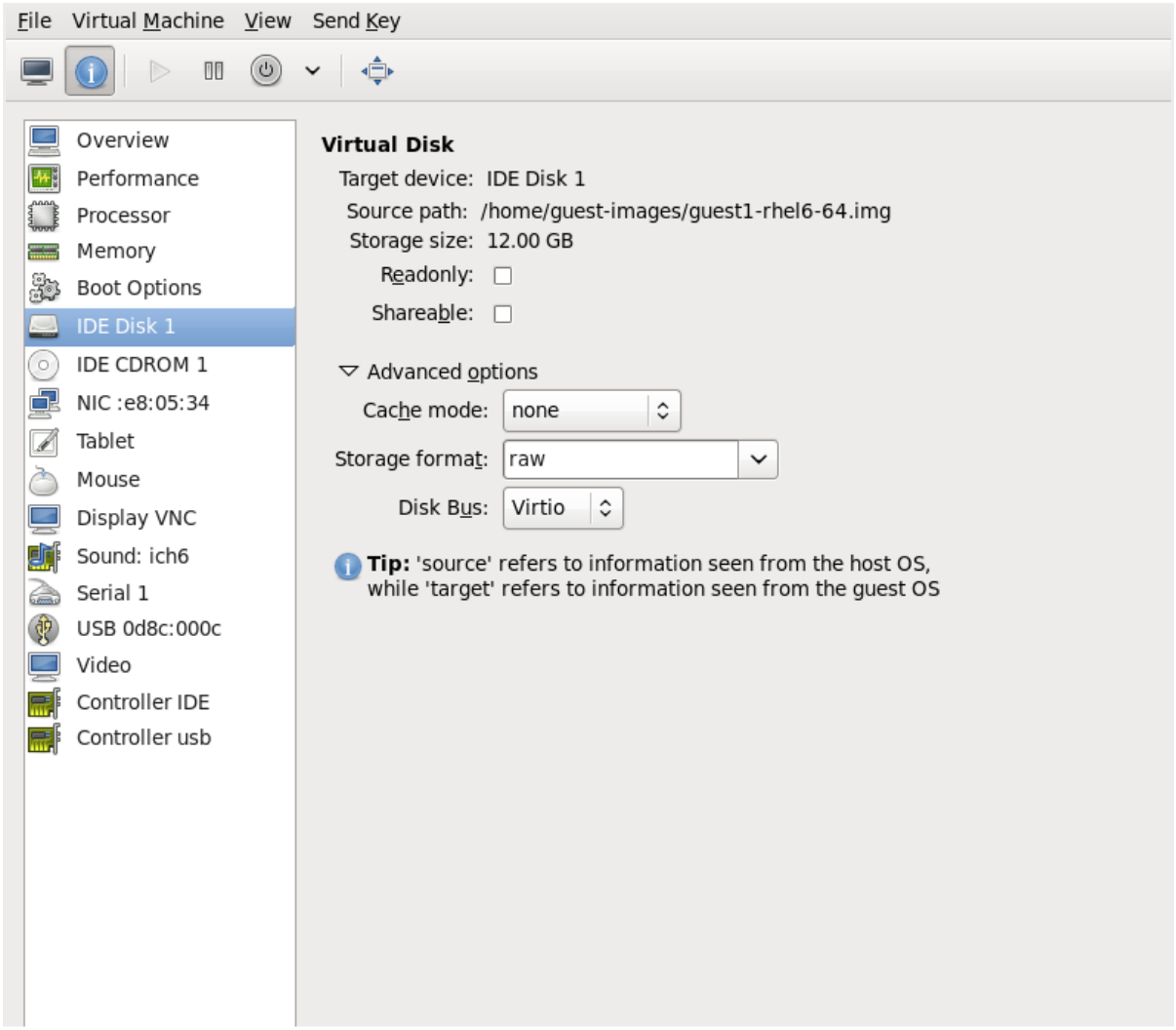
5. *从左侧的导航窗格中选择 **Memory**。**Memory** 视图允许您查看或更改当前的内存分配。*

图 15.15. 显示内存分配



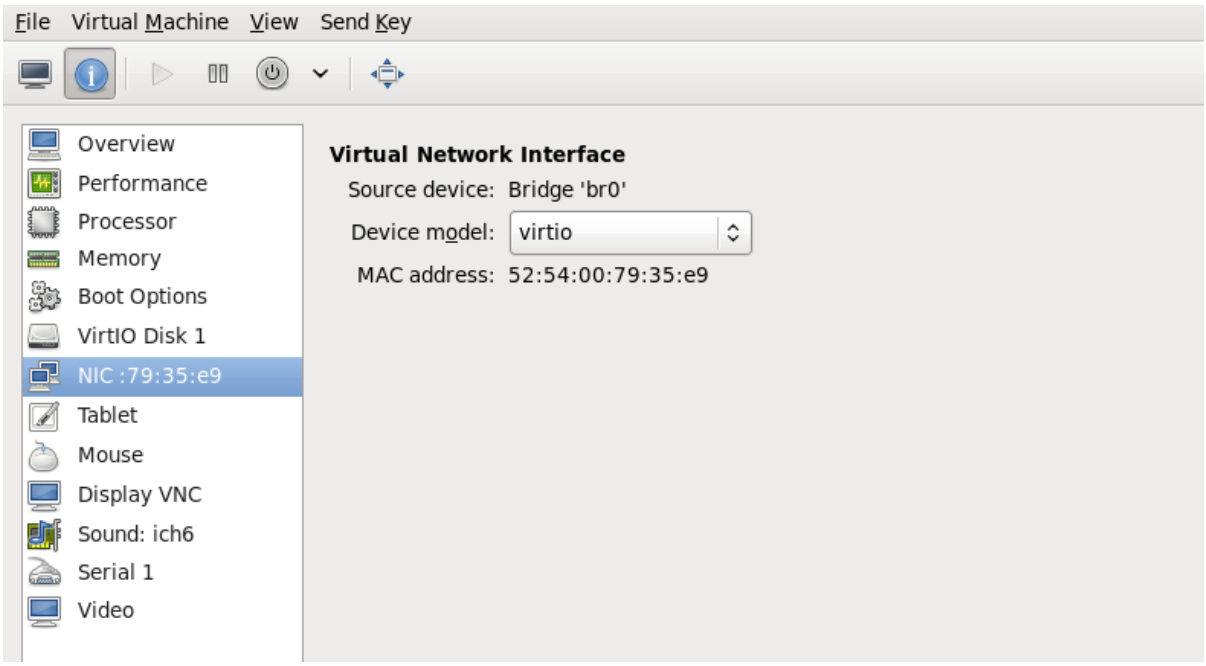
6. *附加到虚拟机的每个虚拟磁盘都会在导航窗格中显示。点击虚拟磁盘进行修改或删除它。*

图 15.16. 显示磁盘配置



7. *附加到虚拟机的每个虚拟网络接口会显示在导航窗格中。点击虚拟网络接口进行修改或删除它。*

图 15.17. 显示网络配置



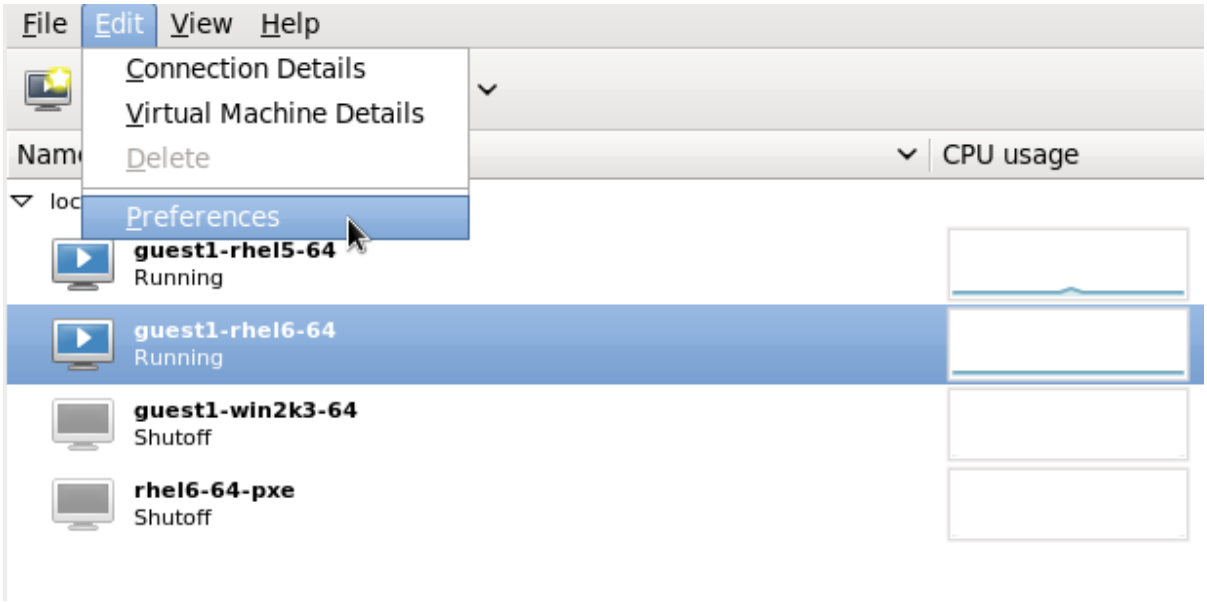
15.7. 性能监控

可以通过 virt-manager 的首选项修改性能监控首选项。

配置性能监控：

1. *在" 编辑 "菜单中，选择 "首选项 "。*

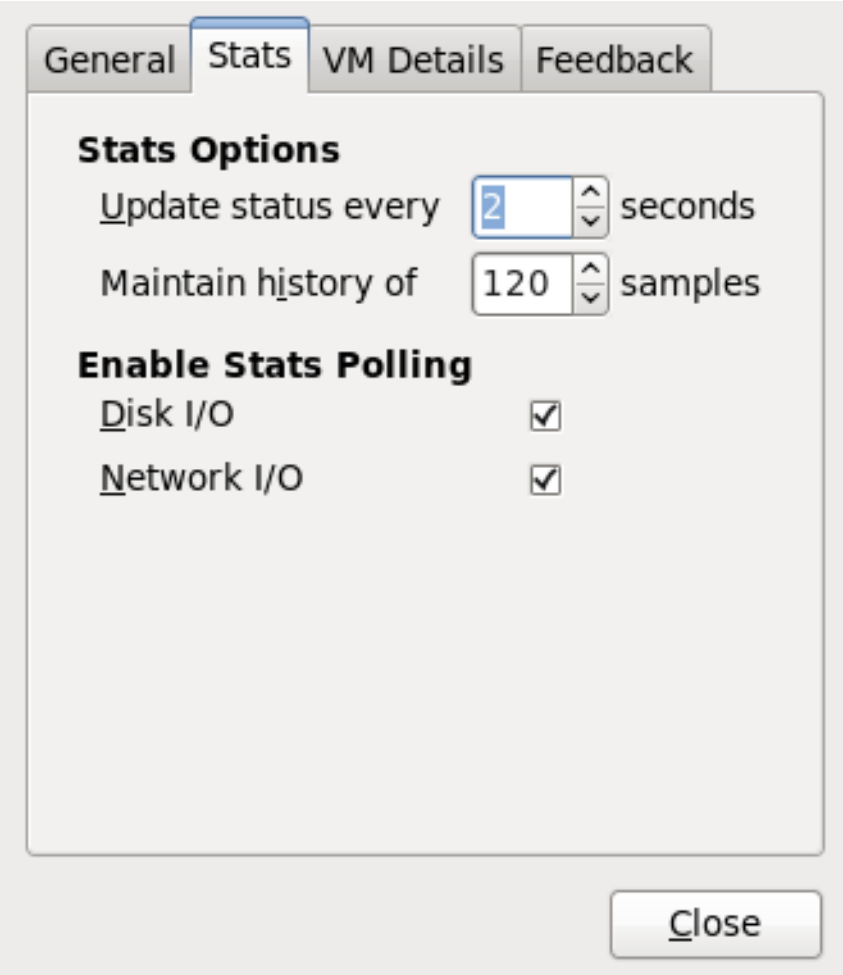
图 15.18. 修改客户机首选项



此时会出现 Preferences 窗口。

2. 从 Stats 选项卡指定时间（以秒为单位）或 stats 轮询选项。

图 15.19. 配置性能监控

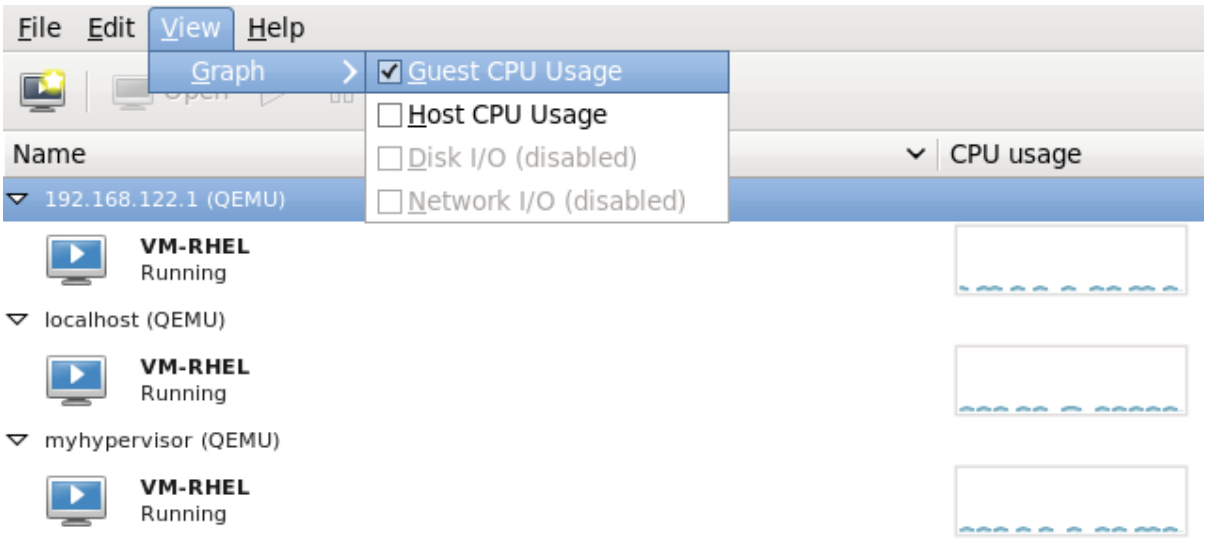


15.8. 显示客户机的 CPU 用量

查看系统中所有客户端的 CPU 使用量：

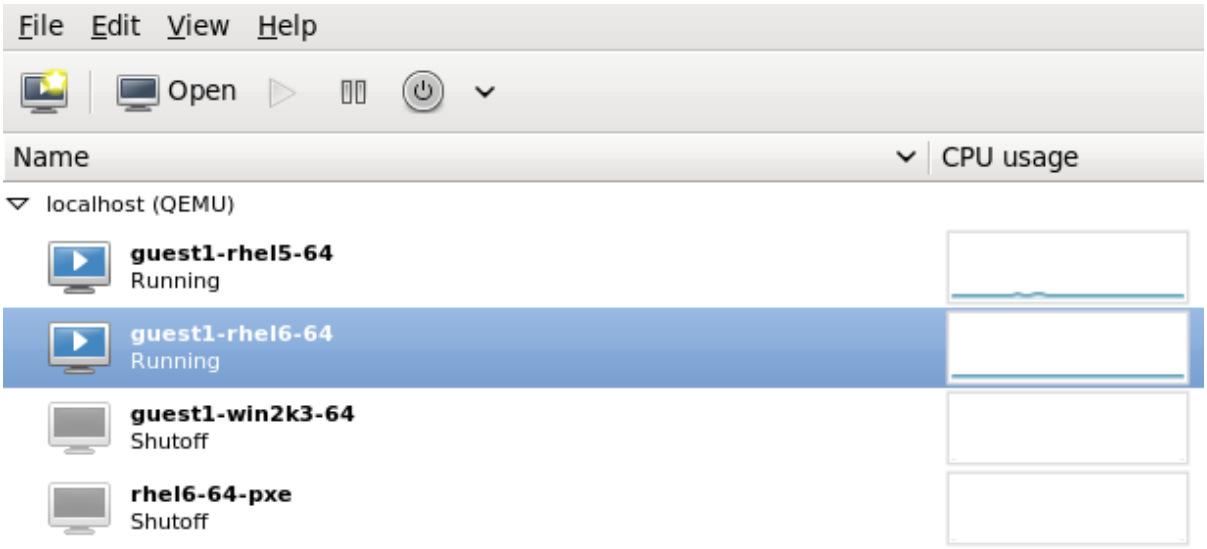
1. 在 View 菜单中，选择 Graph，然后选择 Guest CPU Usage 复选框。

图 15.20. 启用客户机 CPU 用量统计图



2. ***Virtual Machine Manager** 显示系统上所有虚拟机的 CPU 使用量图。*

图 15.21. 客户机 CPU 用量图

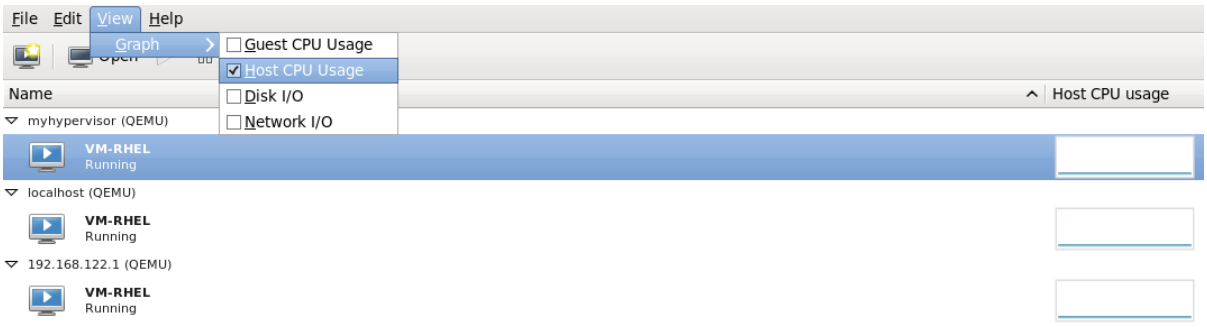


15.9. 显示主机的 CPU 用量

查看系统中所有主机的 CPU 使用量：

1. *在 **View** 菜单中，选择 **Graph**，然后选择 **Host CPU Usage** 复选框。*

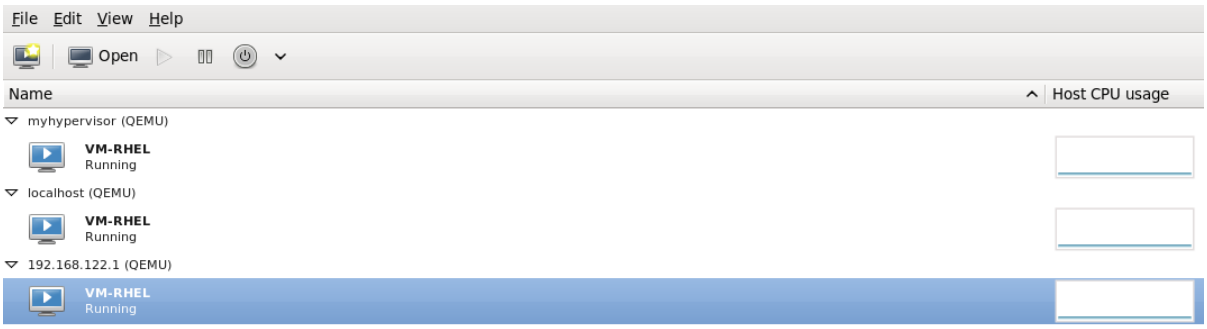
图 15.22. 启用主机 CPU 用量统计图



2.

虚拟机管理器显示系统中主机 CPU 用量图表。

图 15.23. 主机 CPU 用量图



15.10. 显示磁盘 I/O

查看系统中所有虚拟机的磁盘 I/O：

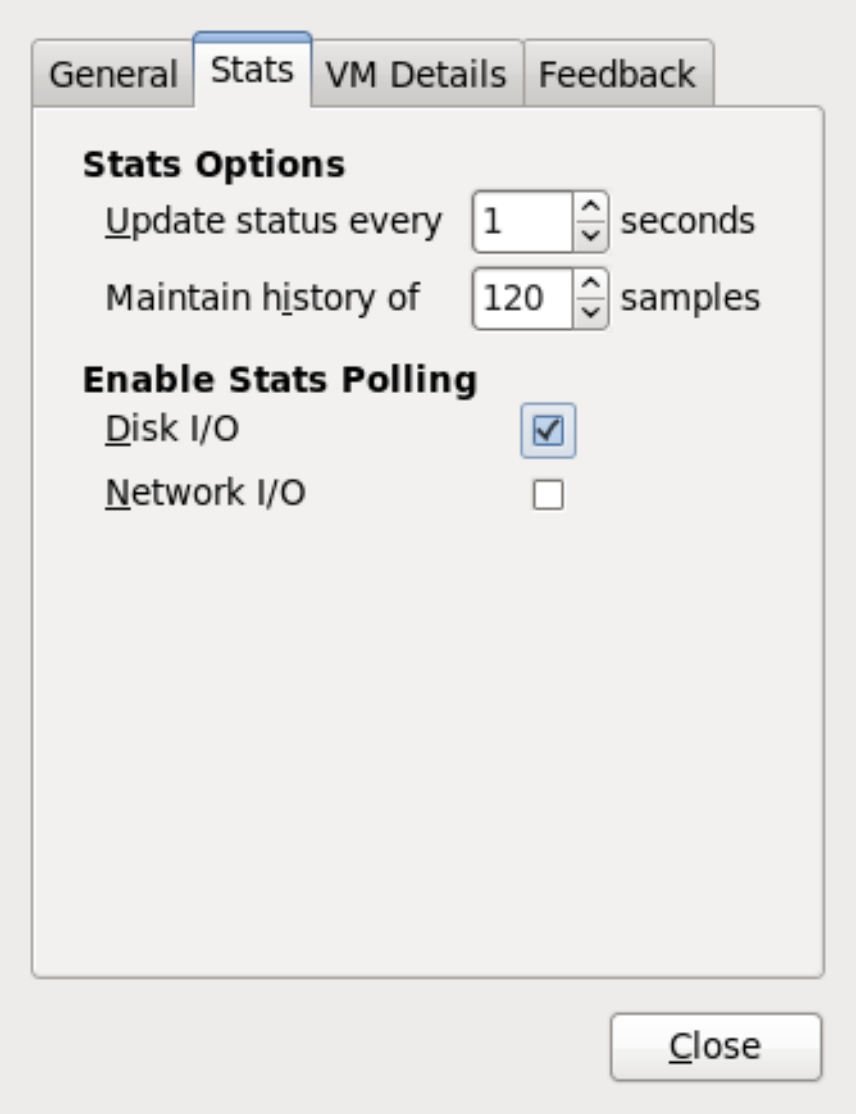
1.

确保启用了磁盘 I/O 统计数据集合。为此，请从" 编辑 "菜单中选择" 首选项" 并单击"设置 "选项卡。

2.

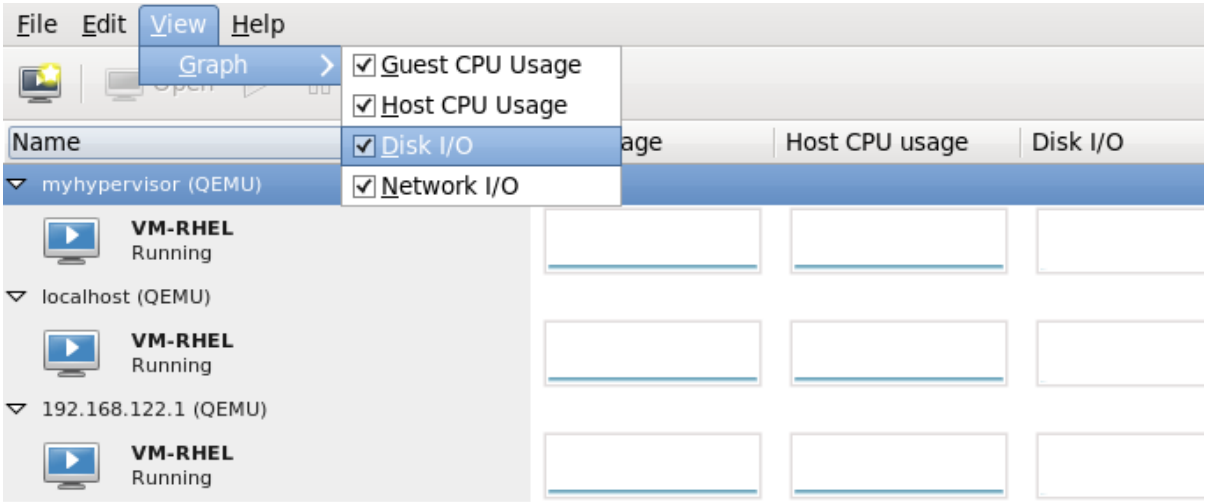
选择 Disk I/O 复选框。

图 15.24. 启用磁盘 I/O



3. *要启用 Disk I.O 显示，从 View 菜单中选择 Graph，然后选择 Disk I/O 复选框。*

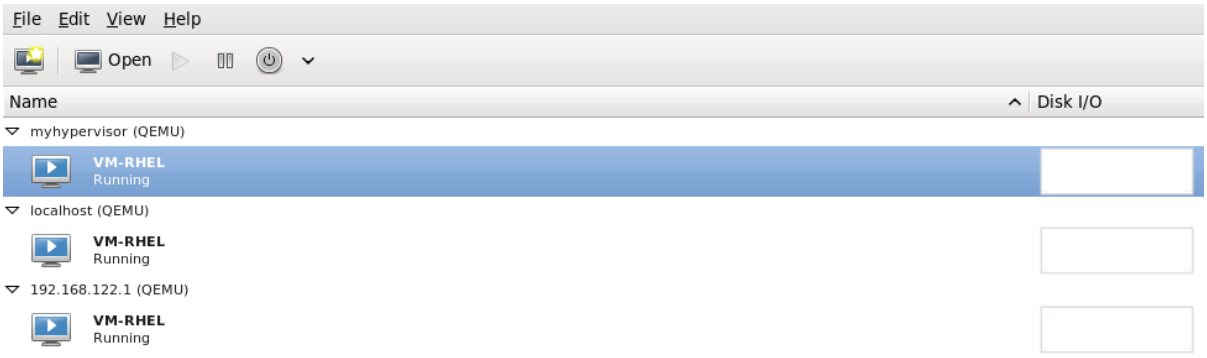
图 15.25. 选择磁盘 I/O



4.

虚拟机管理器显示系统上所有虚拟机的磁盘 I/O 图表。

图 15.26. 显示磁盘 I/O



15.11. 显示网络 I/O

查看系统中所有虚拟机的网络 I/O：

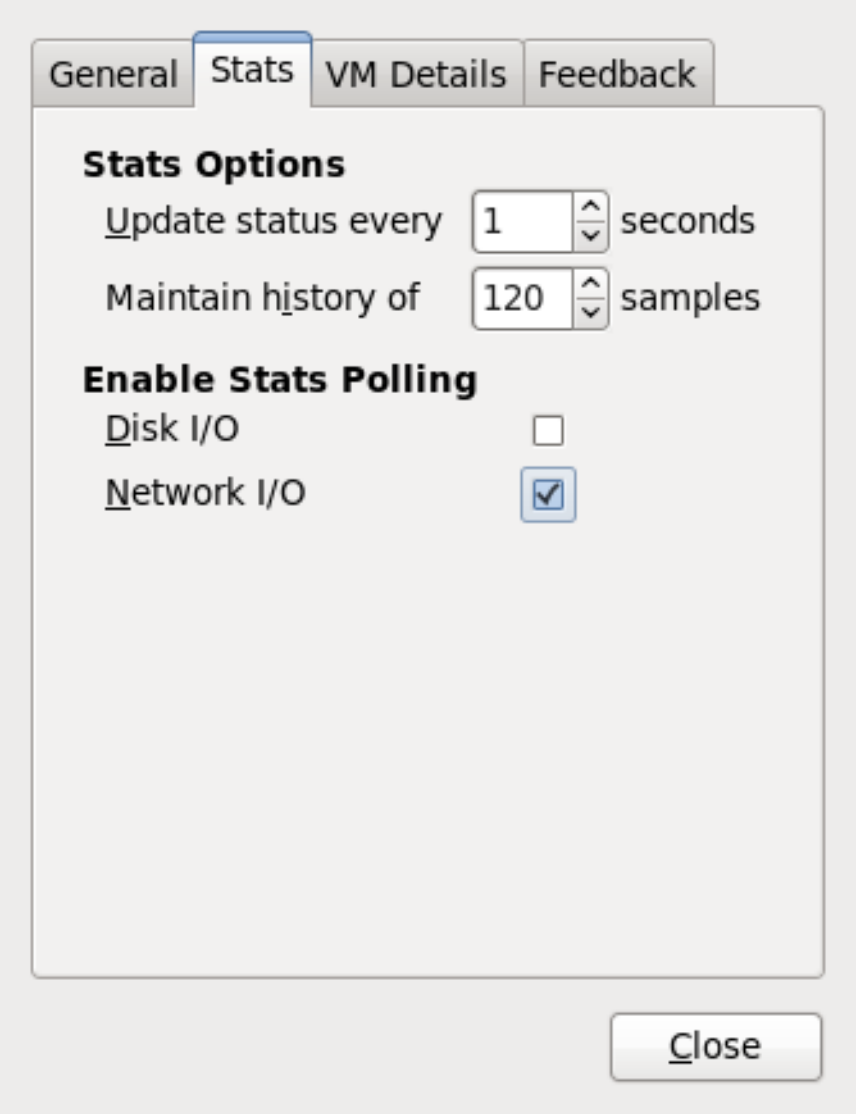
1.

确保已启用网络 I/O 统计数据集合。为此，请从" 编辑 "菜单中选择" 首选项" 并单击"设置 "选项卡。

2.

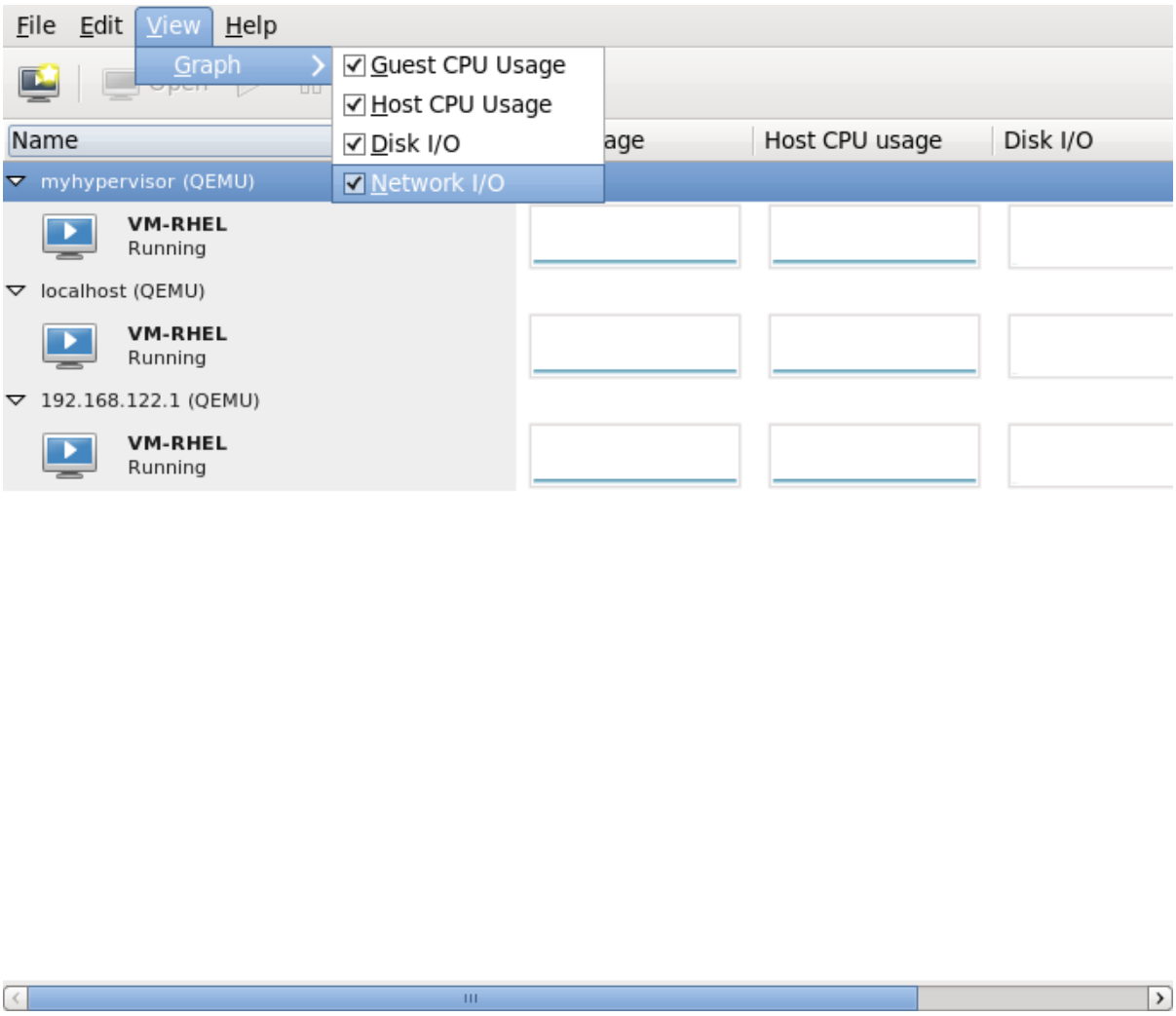
选中 Network I/O 复选框。

图 15.27. 启用网络 I/O



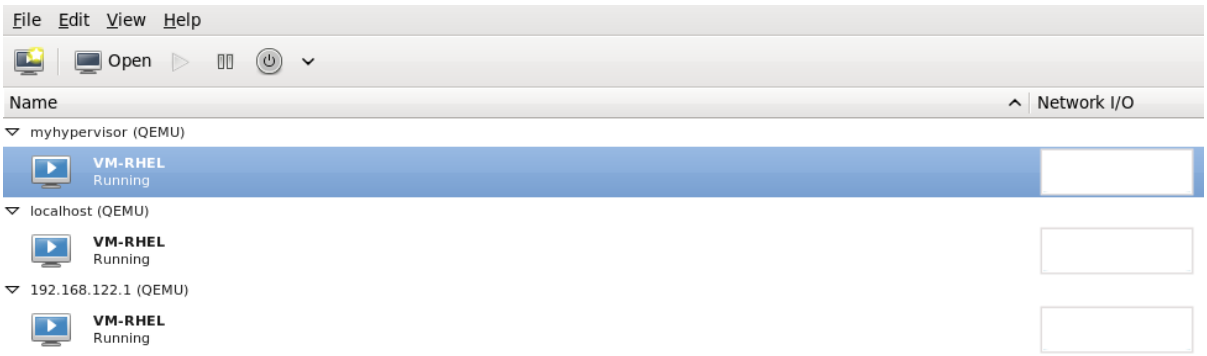
3. *要显示 Network I/O 统计信息，请从 View 菜单中选择 Graph，然后选择 Network I/O 复选框。*

图 15.28. 选择网络 I/O



4. *虚拟机管理器显示系统上所有虚拟机的网络 I/O 图表。*

图 15.29. 显示网络 I/O



第 16 章 使用离线工具访问客户端虚拟机磁盘

16.1. 简介

Red Hat Enterprise Linux 6 附带可访问、编辑和创建主机物理磁盘或其他磁盘镜像的工具。这些工具有几个用途，包括：

- 查看或下载位于主机物理机器磁盘上的文件。
- 编辑或将文件上传到主机物理磁盘中。
- 读取或编写主机物理计算机配置.
- 在 Windows 主机物理机器中读取或编写 Windows 注册表。
- 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。
- 修复无法引导的主机物理机器或需要更改引导的主机物理机器。
- 监控主机物理机器的磁盘用量。
- 审核主机物理机的合规性，例如到组织安全标准。
- 通过克隆和修改模板部署主机物理计算机。
- 读取 CD 和 DVD ISO 和软盘磁盘映像.

警告

您 绝不 能使用这些工具写入主机物理计算机或磁盘镜像（附加到正在运行的虚拟机上，甚至以写入模式打开此类磁盘镜像）。这样做会导致客户端虚拟机的磁盘损坏。那些尝试阻止您执行此操作的工具，但并不捕获所有情况。如果 guest 虚拟机可能正在运行任何可疑，则强烈建议使用的工具，或者至少 总是 以只读模式使用这些工具。

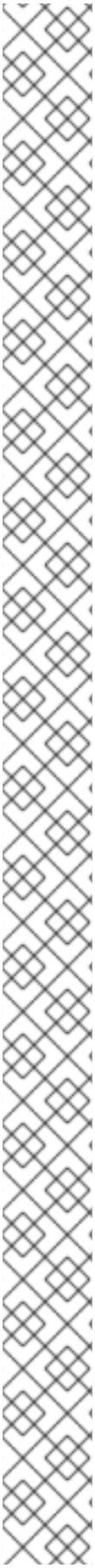
注意

Red Hat Enterprise Linux 6 中的一些虚拟化命令允许您指定远程 libvirt 连接。例如：

```
virt-df -c qemu://remote/system -d Guest
```

但是，Red Hat Enterprise Linux 6 中的 libguestfs 无法访问远程客户端，使用远程 URL 的命令无法按预期工作。这会影响以下 Red Hat Enterprise Linux 6 命令：

- *guestfish*
- *guestmount*
- *virt-alignment-scan*
- *virt-cat*
- *virt-copy-in*
- *virt-copy-out*
- *virt-df*



- ***virt-edit***
- ***virt-filesystems***
- ***virt-inspector***
- ***virt-inspector2***
- ***virt-list-filesystems***
- ***virt-list-partitions***
- ***virt-ls***
- ***virt-rescue***
- ***virt-sysprep***
- ***virt-tar***
- ***virt-tar-in***
- ***virt-tar-out***
- ***virt-win-reg***

***16.2.** 术语*

本节介绍本章中使用的术语。

- libguestfs（Guest 文件系统库）- 底层 C 库提供了打开磁盘镜像、读写文件等的基本功能。您可以直接将 C 程序写入这个 API，但它非常低。
- guestfish（Guest 文件系统交互式 shell）是一个交互式 shell，您可以从命令行或从 shell 脚本使用。它公开 libguestfs API 的所有功能。
- 不同的 virt 工具是在 libguestfs 之上构建的，它们提供了一种从命令行执行特定单任务的方法。工具包括 virt-df、virt-rescue、virt-resize 和 virt-edit。
- hivex 和 Augeas 是分别编辑 Windows 注册表和 Linux 配置文件的库。虽然它们与 libguestfs 分开，但 libguestfs 的大部分值都来自于这些工具的组合。
- 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。这个功能不是必需的，但可能会很有用。

16.3. 安装

要安装 libguestfs、guestfish、guestmount 和 Windows 客户机虚拟机支持，请订阅 Red Hat Enterprise Linux V2WIN 频道，访问 Red Hat Website 并运行以下命令：

```
# yum install libguestfs guestfish libguestfs-tools libguestfs-winsupport
```

要安装所有与 libguestfs 相关的软件包，包括语言绑定，请运行以下命令：

```
# yum install '*guestf*'
```

16.4. GUESTFISH SHELL

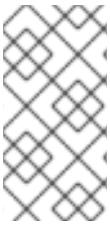
guestfish 是一个交互式 shell，您可以从命令行使用或从 shell 脚本访问 guest 虚拟机文件系统。libguestfs API 的所有功能都可从 shell 访问。

要开始查看或编辑虚拟机磁盘镜像，请运行以下命令，替换所需磁盘镜像的路径：

```
guestfish --ro -a /path/to/disk/image
```

--ro 表示磁盘镜像是以只读方式打开。这个模式始终安全，但不允许写入访问。只有当您 确定 guest 虚拟机未在运行时，或者磁盘镜像未附加到实时客户端虚拟机时，才省略这个选项。无法使用 libguestfs 编辑 live 客户机虚拟机，并尝试造成不可逆向的磁盘损坏。

/path/to/disk/image 是磁盘的路径。这可以是文件、主机物理机器逻辑卷（如 /dev/VG/LV）、主机物理设备(/dev/cdrom)或 SAN LUN(/dev/sdf3)。



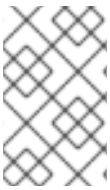
注意

libguestfs 和 guestfish 不需要 root 权限。如果被访问的磁盘镜像需要 root 读取或写入，则只需要以 root 身份运行它们。

当您以交互方式启动 guestfish 时，它将显示这个提示：

```
guestfish --ro -a /path/to/disk/image

Welcome to guestfish, the libguestfs filesystem interactive shell for editing virtual machine
filesystems.

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 来发起库并附加磁盘镜像。在第一次完成后最多可能需要 30 秒。随后启动将更快地完成。



注意

libguestfs 将使用硬件虚拟化加速，如 KVM（如果可用）来加快此过程。

输入了 run 命令后，可以使用其他命令，如以下部分所示。

16.4.1. 使用 guestfish 查看文件系统

这部分提供有关使用 guestfish 查看文件的信息。

16.4.1.1. 手动列表和查看

list-filesystems 命令将列出由 libguestfs 找到的文件系统。此输出显示了 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
```

此输出显示 Windows 磁盘镜像：

```
><fs> run
><fs> list-filesystems
/dev/vda1: ntfs
/dev/vda2: ntfs
```

其他有用的命令有 list-devices、list-partitions、lvs、pvs、vfs-type 和 文件。您可以通过键入 help 命令获取有关任何命令的更多信息和帮助，如下所示：

```
><fs> help vfs-type
 NAME
    vfs-type - get the Linux VFS type corresponding to a mounted device

 SYNOPSIS
     vfs-type device

 DESCRIPTION
    This command gets the file system type corresponding to the file system on
    "device".

    For most file systems, the result is the name of the Linux VFS module
    which would be used to mount this file system if you mounted it without
    specifying the file system type. For example a string such as "ext3" or
    "ntfs".
```

要查看文件系统的实际内容，必须首先挂载它。这个示例使用前面输出中显示的 Windows 分区之一 (/dev/vda2)，在这种情况下，已知与 C:\ 驱动器对应：

```
><fs> mount-ro /dev/vda2 /
><fs> ll /
```

```
total 1834753
 drwxrwxrwx  1 root root       4096 Nov  1 11:40 .
 drwxr-xr-x 21 root root       4096 Nov 16 21:45 ..
 lrwxrwxrwx  2 root root         60 Jul 14  2009 Documents and Settings
 drwxrwxrwx  1 root root       4096 Nov 15 18:00 Program Files
 drwxrwxrwx  1 root root       4096 Sep 19 10:34 Users
 drwxrwxrwx  1 root root      16384 Sep 19 10:34 Windows
```

您可以使用 ls、ll ll、cat、下载 和 tar 等 guestfish 命令来查看和下载文件和目录。

注意

此 shell 中当前工作目录没有概念。与普通 shell 不同，您无法使用 cd 命令来更改目录。所有路径都必须使用正斜杠(/)字符在顶部开始。使用 Tab 键完成路径。

要退出 guestfish shell，请按 exit 或按 Ctrl+d。

16.4.1.2. 使用 guestfish 检查

若要让 guestfish 本身检查映像并挂载文件系统，而不必手动列出和挂载文件系统。要做到这一点，请在命令行中添加 -i 选项：

```
guestfish --ro -a /path/to/disk/image -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 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
 [etc]
```

由于 guestfish 需要启动 libguestfs 后端才能执行检查和挂载，因此使用 -i 选项时不需要 运行命令。i 选项适用于许多通用的 Linux 和 Windows 客户机虚拟机。

16.4.1.3. 按名称访问客户机虚拟机

当您指定名称给 libvirt（换句话说，如 virsh list --all）中时，可以从命令行访问 guest 虚拟机。使用 -d 选项按名称访问客户端虚拟机，并使用 -i 选项或不使用 -i 选项：

```
guestfish --ro -d GuestName -i
```

16.4.2. 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，首先在本节的开头显示警告： 您的 guest 虚拟机必须关闭。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。确定 guest 虚拟机已关闭时，您可以省略 --ro 选项以便通过命令获得写入访问权限，例如：

```
guestfish -d RHEL3 -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 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
```

编辑文件的命令包括 编辑、 vi s 和 emacs。还存在用于创建文件和目录的许多命令，如 write、 mkdir、 upload 和 tar-in。

16.4.3. 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，以及更多命令，例如 mkfs、part-add、lvresize、lvresize、lvcreate 和 pvcreate。

16.4.4. 使用 guestfish 进行 shell 脚本

熟悉以互动方式使用 guestfish 后，将 shell 脚本编写成非常有用。以下是一个简单的 shell 脚本，用

于向客户机添加新的 MOTD（一天的消息）：

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$guestname" -i <<'EOF'
   write /etc/motd "Welcome to Acme Incorporated."
   chmod 0644 /etc/motd
 EOF
```

16.4.5. augeas 和 libguestfs 脚本

将 libguestfs 与 Augeas 结合使用可以帮助编写脚本来操作 Linux 客户机虚拟机配置。例如，以下脚本使用 Augeas 解析 guest 虚拟机的键盘配置，并打印出布局。请注意，这个示例只适用于运行 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 也可以用于修改配置文件。您可以修改以上脚本以 更改键盘布局 ：

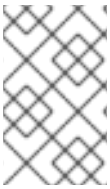
```
#!/bin/bash -
 set -e
 guestname="$1"

 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，从而能够写入 guest 虚拟机。

2. aug-get 命令已被改为 aug-set 来修改值，而不是获取它。新值为 "gb"（包括引号）。

3. aug-save 命令在此处使用，因此 Augeas 会将更改写入到磁盘。



注意

有关 Augeas 的更多信息，请访问网站 http://augeas.net。

guestfish 可以在此简介文档中介绍更多工作。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，请参阅 man page guestfish(1)。

16.5. 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的简单工具。

- virt-cat 与 guestfish 下载 命令类似。它将下载并向客户端虚拟机显示一个文件。例如：

```
# virt-cat RHEL3 /etc/ntp.conf | grep ^server
server      127.127.1.0       # local clock
```

- virt-edit 与 guestfish edit 命令类似。它可用于交互式地编辑客户机虚拟机中的一个文件。例如，您可能需要在无法引导的基于 Linux 的客户机虚拟机中编辑 grub.conf 文件：

```
# virt-edit LinuxGuest /boot/grub/grub.conf
```

virt-edit 还有另一种模式，可用于对单个文件进行简单的非交互式更改。为此，使用了 -e 选项。例如，这个命令将 Linux 客户机虚拟机中的 root 密码改为没有密码：

```
# virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/'
```

- virt-ls 与 guestfish ls、ll 和 find 命令相似。它用于列出目录或目录（递归）。例如，以下命令会在 Linux 客户机虚拟机中递归列出 /home 下的文件和目录：

```
# virt-ls -R LinuxGuest /home/ | less
```

16.6. VIRT-RESCUE: RESCUE SHELL

这部分提供有关使用 rescue shell 的信息。

16.6.1. 简介

这部分论述了 virt-rescue，它可以看作用于虚拟机的救援 CD。它将客户机虚拟机启动到救援 shell 中，以便执行维护以更正错误，并且可以修复 guest 虚拟机。

virt-rescue 和 guestfish 之间存在一些重叠。务必要区分不同的用途。virt-rescue 是使用普通 Linux 文件系统工具进行交互式的临时更改。它特别适用于更正已失败的 guest 虚拟机。virt-rescue 无法脚本化。

相反，guestfish 对通过一组正式命令(libguestfs API)进行脚本化的结构化更改特别有用，但它也可以以交互方式使用。

16.6.2. 运行 virt-rescue

在您对客户机虚拟机使用 virt-rescue 之前，请确保 guest 虚拟机没有运行，否则将发生磁盘损坏。确定 guest 虚拟机未处于活动状态时，请输入：

```
virt-rescue GuestName
```

（其中 GuestName 是 libvirt 已知的客户端名称），或：

```
virt-rescue /path/to/disk/image
```

（其中该路径可以是任意文件、任何逻辑卷、LUN 等等），其中包含客户机虚拟机磁盘。

您将首先看到输出滚动，因为 virt-rescue 引导救援虚拟机。在结束时，您会看到：

```
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 shell 提示符是一个普通 bash shell，并提供了一组减少的普通 Red Hat Enterprise Linux 命令。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，建议将其挂载到 /sysroot 下，这是救援机器上的一个空目录，供用户挂载您喜欢的任何内容。请注意，/ 下的文件是 rescue 虚拟机本身的文件：

```
><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
[...]
```

完成客户端虚拟机清理后，输入 exit 或 Ctrl+d 退出 shell。

virt-rescue 有很多命令行选项。最常使用的选项有：

- --ro ：在 guest 虚拟机上以只读模式确定。不会保存任何更改。您可以使用它来试验客户端虚拟机。从 shell 退出后，所有更改将被丢弃。
- --network ：启用 rescue shell 中的网络访问。如果您需要将 RPM 或其他文件下载到客户端虚拟机，请使用它。

16.7. VIRT-DF: 监控磁盘使用情况

本节介绍了使用 virt-df 监控磁盘使用情况。

16.7.1. 简介

这部分论述了 virt-df，它从磁盘镜像或客户机虚拟机显示文件系统的使用。它类似于 Linux df 命令，但对于虚拟机。

16.7.2. 运行 virt-df

要显示磁盘镜像中所有文件系统的文件系统使用情况，请输入以下内容：

```
# virt-df /dev/vg_guests/RHEL6
 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/RHEL6 是 Red Hat Enterprise Linux 6 客户机虚拟机磁盘映像。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 来列出所有 guest 虚拟机（例如对 libvirt 已知的信息）。virt-df 命令可识别一些与标准 df （可读）和 - i （显示索引节点而不是块）相同的选项。

virt-df 也可以在 Windows 客户机虚拟机上工作：

```
# virt-df -h
 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%
 Win7x32:/dev/sda1             100.0M      24.1M      75.9M   25%
 Win7x32:/dev/sda2      19.9G      7.4G      12.5G   38%
```

注意

您可以在 live guest 虚拟机上使用 virt-df，因为它只需要只读访问。但是，您不应该预期数字与在 guest 虚拟机内运行的 df 命令中完全一样。这是因为磁盘上的内容与实时客户机虚拟机的状态稍有同步。然而，它应该足够多地进行分析和监视。

virt-df 旨在允许您将统计数据集成到监控工具、数据库等中。这样，系统管理员可以生成关于磁盘使用情况趋势的报告，并在客户机虚拟机耗尽磁盘空间时警报。要做到这一点，您应该使用 --csv 选项来生成 machine- readable Comma-Separated-Values(CSV)输出。CSV 输出可由大多数数据库、电子表格软件以及各种其他工具和编程语言读取。原始 CSV 类似如下：

```
# virt-df --csv WindowsGuest
Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
Win7x32,/dev/sda1,102396,24712,77684,24.1%
Win7x32,/dev/sda2,20866940,7786652,13080288,37.3%
```

有关如何处理此输出以产生趋势和警报的资源和理念，请参考以下 URL： http://libguestfs.org/virt-df.1.html

16.8. VIRT-RESIZE：重新定义虚拟机离线大小

这部分提供有关重新定义离线客户端虚拟机大小的信息。

16.8.1. 简介

这部分论述了 virt-resize，这是扩展或缩小客户机虚拟机的工具。它只适用于离线的虚拟客户机（关闭）。它的工作原理是复制客户机虚拟机镜像并省略原始磁盘镜像。这是理想情况，因为您可以使用原始镜像作为备份，但您需要达到两倍的磁盘空间量。

16.8.2. 扩展磁盘镜像

本节演示了扩展磁盘镜像的简单情况：

1. *找到要调整大小的磁盘镜像。您可以为 libvirt guest 虚拟机使用 virsh dumpxml GuestName 命令。*

2. *决定您需要扩展 guest 虚拟机的方式。在 guest 虚拟机磁盘上运行 virt-df -h 和 virt-list-partitions -lh，如下所示：*

```
# virt-df -h /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-list-partitions -lh /dev/vg_guests/RHEL6
/dev/sda1 ext3 101.9M
/dev/sda2 pv 7.9G
```

本示例将演示如何：

- 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充剩余的空间。
- 展开 /dev/VolGroup00/LogVol00 以填写第二个分区中的新空间。

1. 确保 guest 虚拟机已关闭。
2. 将原始磁盘重命名为备份。如何这样做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（在此示例中所示），使用 lvrename ：

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. 创建新磁盘。本例中的要求是将磁盘总大小扩展至 16GB。因为这里使用了逻辑卷，使用以下命令：

```
# lvcreate -L 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. 此命令表示第 2 步中的要求：

```
# 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 将第一个分区的大小调整为

500MB。--expand /dev/sda2 扩展第二个分区以填充所有剩余空间。--LV-expand /dev/VolGroup00/LogVol00 扩展客户机虚拟机逻辑卷，以填补第二个分区中的额外空间。

virt-resize 描述了它在输出中要执行的操作：

```
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。如果它可以正常工作（在测试完后），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-list-partitions 显示新大小：

```
# virt-df -h /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%
```

重新定义 guest 虚拟机大小并非精确的科学程度。如果 virt-resize 失败，您可以在 virt-resize(1)man page 中查看和尝试。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。

16.9. VIRT-INSPECTOR：检查客户机虚拟机

这部分提供有关使用 virt-inspector 检查虚拟客户机的信息。

16.9.1. 简介

virt-inspector 是一个检查磁盘镜像的工具，用于查找它所包含的操作系统。

注意

Red Hat Enterprise Linux 6.2 提供了这个程序的两个变体： virt-inspector 是 Red Hat Enterprise Linux 6.0 中的原始程序，现已被弃用的上游。virt-inspector2 与新的上游 virt-inspector 程序相同。

16.9.2. 安装

要安装 virt-inspector 和文档，请输入以下命令：

```
# yum install libguestfs-tools libguestfs-devel
```

要处理 Windows 客户机虚拟机，还必须安装 libguestfs-winsupport。详情请查看 第 16.10.2 节 “安装”。输出中包括示例 XML 输出和 Relax-NG 模式的文档将安装在 /usr/share/doc/libguestfs-devel-*/ 中，其中 "*" 被 libguestfs 的版本号替换。

16.9.3. 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 --xml disk.img > report.xml
```

如下所示：

```
virt-inspector --xml GuestName > report.xml
```

结果为 XML 报告(report.xml)。XML 文件的主要组件是一个顶层 < operatingsystems> 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>
```

处理这些报告最好使用 W3C 标准 XPath 查询来完成。Red Hat Enterprise Linux 6 附带了一个可用于简单实例的命令行程序(xpath)；但是，出于长期和高级用途，您应该考虑使用 XPath 库以及您最喜欢的编程语言。

例如，您可以使用以下 XPath 查询列出所有文件系统设备：

```
virt-inspector --xml 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安装的所有应用程序的名称：

```
virt-inspector --xml GuestName | xpath //application/name
[...long list...]
```

16.10. VIRT-WIN-REG：阅读并编辑 WINDOWS REGISTRY

16.10.1. 简介

virt-win-reg 是一个在 Windows 客户机虚拟机中操作 Registry 的工具。它可用于读取 registry 密钥。您还可以使用它来更改 Registry，但 您永远不会 试图为 live/ running guest 虚拟机执行此操作，因为它会导致磁盘损坏。

16.10.2. 安装

要使用 virt-win-reg，您必须运行以下命令：

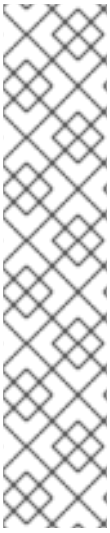
```
# yum install /usr/bin/virt-win-reg
```

16.10.3. 使用 virt-win-reg

要读取 Registry 密钥，请指定客户端虚拟机的名称（或其磁盘镜像）和 Registry 密钥的名称。您必须使用单引号括起所需密钥的名称：

```
# virt-win-reg WindowsGuest \
  'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall' \
  | less
```

输出采用 Windows 上 .REG 文件的标准文本格式。



注意

hex-quoting 用于字符串，因为格式没有正确为字符串定义可移植编码方法。这是确保在将 .REG 文件从一个计算机传输到另一台计算机时所采用的唯一方法。

您可以通过此简单的 Perl 脚本来传送 virt-win-reg 的输出可打印的 hex-quoted 字符串：

```
perl -MEncode -pe's?hex\((\d+)\):(\S+)?
$t=$1;$_=$2;s,\,,,g;"str($t):\"".decode(utf16le=>pack("H*",$_))."\""?eg'
```

要将更改合并到离线客户端虚拟机的 Windows 注册表中，您必须首先准备 .REG 文件。这里提供了关于执行此操作的大量文档。准备好 .REG 文件后，输入以下内容：

```
# virt-win-reg --merge WindowsGuest input.reg
```

这将更新客户机虚拟机中的 registry。

16.11. 使用编程语言的 API

红帽企业 Linux 6.2 中的以下语言可直接使用 libguestfs API：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
```

每个语言的绑定基本上相同，但具有细微变化。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 ()
```

本节仅包括 C 中的 API。

在 C 和 C++ 绑定中，您必须手动检查是否有错误。在其他绑定中，错误会被转换为例外。以下示例中显示的其他错误检查不需要其他语言，但您可能希望添加代码来捕获异常。有关 libguestfs API 架构的一些关注点，请参考以下列表：

- libguestfs API 是同步的。每个调用块直到完成为止。如果要异步调用，您必须创建线程。
- libguestfs API 不是线程安全：每个句柄应该只从一个线程使用，或者您希望在线程之间共享一个进程，您应该实施自己的 mutex，以确保两个线程无法同时在一个进程上执行命令。
- 您不应该在同一磁盘镜像中打开多个句柄。如果所有句柄都是只读的，但仍然不建议这样做。
- 如果其他部分可能正在使用该磁盘镜像（例如，实时虚拟机），您不应该添加磁盘镜像来写入。执行此操作将导致磁盘损坏。
- 在当前使用（例如，实时虚拟机）的磁盘镜像上打开只读句柄；但是，如果磁盘映像在读取时大量写入该磁盘镜像时，结果可能会无法预测或不一致。

16.11.1. 通过 C 程序与 API 交互

您的 C 程序应该首先包括 {s.> 标头文件并创建一个句柄：

```
#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);
 }
```

将此程序保存到文件(test.c)。编译这个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，应该打印没有输出。本节的其余部分演示了演示了如何扩展此程序以创建新磁盘镜像、对它进行分区，将其格式化为 ext4 文件系统，并在文件系统中创建一些文件。磁盘映像将命名为 disk.img，并在当前目录中创建。

这个程序概述如下：

- 创建句柄。
- 将磁盘添加到句柄。
- 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，您应该使用名为 disk.img 的磁盘镜像（可以使用 guestfish 检查）来保留它：

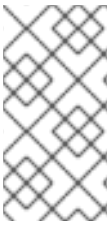
```
guestfish --ro -a disk.img -m /dev/sda1
><fs> ll /
><fs> cat /foo/resolv.conf
```

默认情况下（仅用于 C 和 C++ 绑定），libguestfs 会将错误打印到 stderr。您可以通过设置错误处理程序来更改此行为。guestfs(3)man page 详细阐述此问题。

16.12. 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，以便克隆可以从中创建克隆。此过程涉及删除 SSH 主机密钥、永久网络 MAC 配置和用户帐户。virt-sysprep 还可以自定义虚拟机，例如通过添加 SSH 密钥、用户或徽标。根据需要，可以启用或禁用每个步骤。

术语"sysprep"来源于与 Microsoft Windows 系统一起使用的系统准备工具(sysprep.exe)。尽管如此，这些工具目前无法在 Windows 客户机上工作。



注意

libguestfs 和 guestfish 不需要 root 权限。如果被访问的磁盘镜像需要 root 访问权限才能读取或写入，您只需要以 root 身份运行它们。

virt-sysprep 工具是 libguestfs-tools-c 软件包的一部分，该软件包使用以下命令安装：

```
$ yum install libguestfs-tools-c
```

或者，使用以下命令只能安装 virt-sysprep 工具：

```
$ yum install /usr/bin/virt-sysprep
```

重要

virt-sysprep 修改客户机或磁盘镜像就位。要使用 virt-sysprep，guest 虚拟机必须离线，因此您必须在运行命令前关闭它。要保留客户机虚拟机的现有内容，必须首先进行快照、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。

下列命令可与 virt-sysprep 一起使用：

表 16.1. virt-sysprep 命令

命令	描述	示例
--help	显示关于特定命令或整个软件包的简短帮助条目。有关其他帮助信息，请查看 virt-sysprep man page。	$ **virt-sysprep --help**
-a [*file*] 或 --add [*file*]	添加 *指定的文件*，它应该是来自客户机虚拟机的磁盘映像。磁盘镜像的格式是自动检测的。要覆盖此并强制使用特定格式，请使用 **--format** 选项。	$ **virt-sysprep --add /dev/vms/disk.img**
-c [*URI*] 或 --connect [*URI*]	如果使用 libvirt，连接到给定的 URI。如果省略，它将通过 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**)，则根本未使用 libvirt。	$ **virt-sysprep -c qemu:///system**

命令	描述	示例
-d [*guest*] 或 --domain [*guest*]	添加来自指定 guest 虚拟机的所有磁盘。可以使用域 UUID 而不使用域名。	$ **virt-sysprep --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-n 或 --dry-run 或 --dryrun	在 guest 虚拟机上执行只读"dry run" sysprep 操作。它会运行 sysprep 操作，但会丢弃对磁盘所做的任何更改。	$ **virt-sysprep -n**
--enable [*operations*]	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-hotkeys,udev-persistent-net**
--format [**raw**|**qcow2**|**auto**]	-a 选项的默认值是自动检测磁盘镜像的格式。这样做会强制在命令行中使用 -a 选项的磁盘格式。使用 --format 自动切换回后续 -a 选项（请参阅上面的 -a 命令）。	$ **virt-sysprep --format raw -a *disk.img*** 会针对 disk.img 强制原始格式（无自动探测），但 **virt-sysprep --format raw -a *disk.img* --format auto -a *another.img*** *会强制对 disk **.img** 强制实施原始格式（无自动探测）。如果您有不受信任的原始格式的客户机磁盘镜像，您应该使用这个选项指定磁盘格式。这可避免出现恶意客户机的可能出现安全问题。*
--list-operations	列出 virt-sysprep 程序支持的操作。它们每行列出，带有一个或多个以空格分隔的字段。输出中的第一个字段是操作名称，可提供给 **--enable** 标志。如果默认操作被启用，则第二个字段是 * 字符，如果没有启用，则为空。同一行中的附加字段包含操作的描述。	$ **virt-sysprep --list-operations**
--mount-options	为客户端虚拟机中的每个挂载点设置挂载选项。使用由分号分隔的 mountpoint:options 对列表。您可能需要在此列表旁边放置引号，以防止它被 shell。	$ **virt-sysprep --mount-options "/:notime"** 将以 **notime** 操作挂载根目录。

命令	描述	示例
--SELinux-relabel 和 --no-selinux-relabel	virt-sysprep 不总是在 guest 第一次引导时调度 SELinux 重新标记。在某些情况下，将执行重新标记（例如，virt-sysprep 已修改文件时）。但是，当所有操作都只删除文件时（例如，在使用 **--enable delete --delete /some/file**时）不会调度重新标记。使用 **--selinux-relabel** 选项总是强制 SELinux 重新标记，而使用 **--no-selinux-relabel** 设置时，不会调度重新标记。建议使用 **--selinux-relabel** 来确保文件具有正确的 SELinux 标签。	$ **virt-sysprep --selinux-relabel**
-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。可用于明确指定新密码，或者使用所选文件的第一行中的字符串 更为安全。	$ **virt-sysprep --root-password password:***123456* **-a** *guest.img* 或者 $ **virt-sysprep --root-password file:***SOURCE_FILE_PATH* **-a** *guest.img*

如需更多信息，请参阅 libguestfs 文档。

16.13. 故障排除

可以使用测试工具检查 libguestfs 是否正常工作。在安装 libguestfs（不需要 root 访问权限）后运行以下命令，以测试正常操作：

```
$ libguestfs-test-tool
```

此工具打印大量文本，以测试 libguestfs 的操作。如果测试成功，则在输出末尾附近将出现以下文本：

```
===== TEST FINISHED OK =====
```

-

16.14. 在哪里可以找到 FURTHER 文档

libguestfs 和工具文档的主要来源是 Unix man page。API 记录在 guestfs(3)中。guestfish 记录在 guestfish(1)中。virt 工具在自己的 man page（如 virt-df(1)）中进行。

第 17 章 虚拟机管理的图形用户界面工具

除了 virt-manager 之外，Red Hat Enterprise Linux 6 还提供了以下工具来访问您的 guest 虚拟机控制台。

17.1. 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 协议进行访问。可以通过名称、ID 或 UUID 来引用 guest。如果 guest 尚未运行，可以在尝试连接到控制台之前将查看器设置为等待等待。viewer 可以连接到远程主机以获取控制台信息，然后使用相同的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，但资源需要较少。另外，与 virt-manager 不同，多数情况下virt-viewer 不需要对 libvirt 具有读写权限。因此，未授权用户可以连接和显示客户机但不允许配置它们，从而可以使用它。

要安装 virt-viewer 工具，请运行：

```
# sudo yum install virt-viewer
```

语法

基本 virt-viewer 命令行语法如下：

```
# virt-viewer [OPTIONS] {guest-name|id|uuid}
```

基本 virt-viewer 命令行语法如下：

连接到客户端虚拟机

如果不带任何选项使用，virt-viewer 将列出它可以在本地系统的默认系统管理程序上连接到的客户机。

要连接到使用默认 hypervisor 的客户机虚拟机：

```
# virt-viewer guest-name-or-UUID
```

要连接到使用 KVM-QEMU 管理程序的客户机虚拟机：

```
# virt-viewer --connect qemu:///system guest-name-or-UUID
```

使用 TLS 连接到远程控制台：

```
# virt-viewer --connect xen://example.org/ guest-name-or-UUID
```

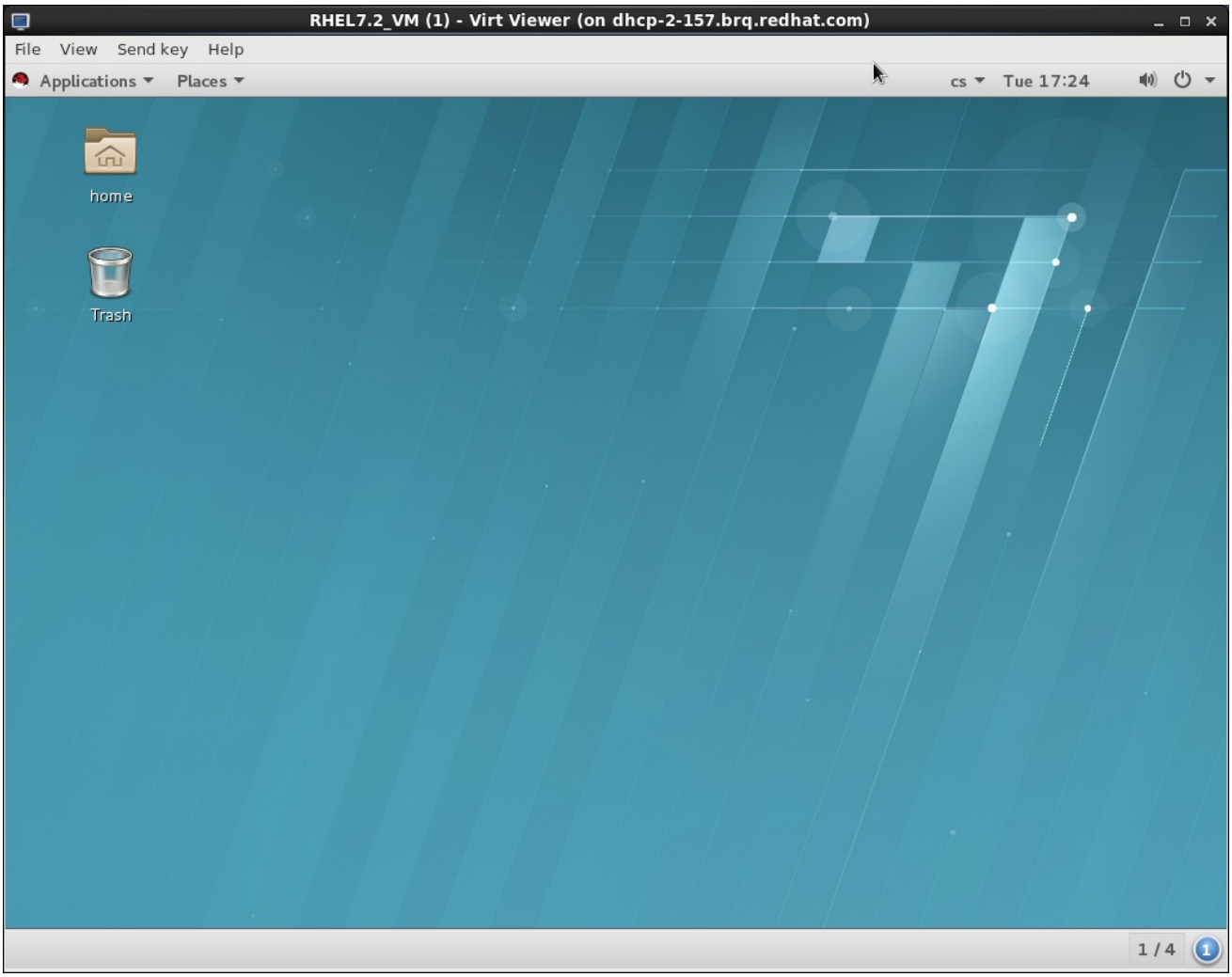
要使用 SSH 连接到远程主机上的控制台，请查找客户机配置，然后建立到控制台的直接非隧道连接：

```
# virt-viewer --direct --connect xen+ssh://root@example.org/ guest-name-or-UUID
```

Interface

默认情况下，virt-viewer 接口只提供与客户端交互的基本工具：

图 17.1. 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-or-UUID
```

可将以下操作分配给热键：

- ***切换-fullscreen***
- ***release-cursor***
- ***smartcard-insert***

- *smartcard-remove*

key-name 组合热键不区分大小写。请注意，热密钥设置不会接管将来的 virt-viewer 会话。

> *例 17.1. 设置 virt-viewer 热键*
>
> *当连接到名为 testguest 的 KVM-QEMU 客户端时，添加热键以切换到完整屏幕模式：*
>
> ```
> # 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 已关闭。当管理员希望限制用户的范围操作到指定客户机时，这很有用。

要使用 kiosk 模式，请使用 -k 或 --kiosk 选项连接到 guest。

> *例 17.2. 在 kiosk 模式中使用 virt-viewer*
>
> *要以 kiosk 模式连接到 KVM-QEMU 虚拟机，在机器关闭后终止，请使用以下命令：*
>
> ```
> # virt-viewer --connect qemu:///system guest-name-or-UUID --kiosk --kiosk-quit on-disconnect
> ```

但请注意，kiosk 模式本身不能确保用户在关闭后与主机系统或客户机设置交互。这要求进一步的安全措施，如禁用主机上的窗口管理器。

17.2. REMOTE-VIEWER

remote-viewer 是支持 SPICE 和 VNC 的简单远程桌面显示客户端。它通过 virt-viewer 共享大多数功能和限制。

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：

```
# sudo yum install virt-viewer
```

语法

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，请使用 man remote-viewer。

连接到客户端虚拟机

如果不带任何选项使用，remote-viewer 将列出它可以连接到本地系统的默认 URI 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 14.5.19 节 “显示用于连接图形显示的 URI”。

例 17.3. 使用 SPICE 连接到客户端显示

使用以下内容连接到名为"testguest"的机器上的 SPICE 服务器，该服务器使用端口 5900 进行 SPICE 通信：

```
# remote-viewer spice://testguest:5900
```

例 17.4. 使用 VNC 连接到客户端显示

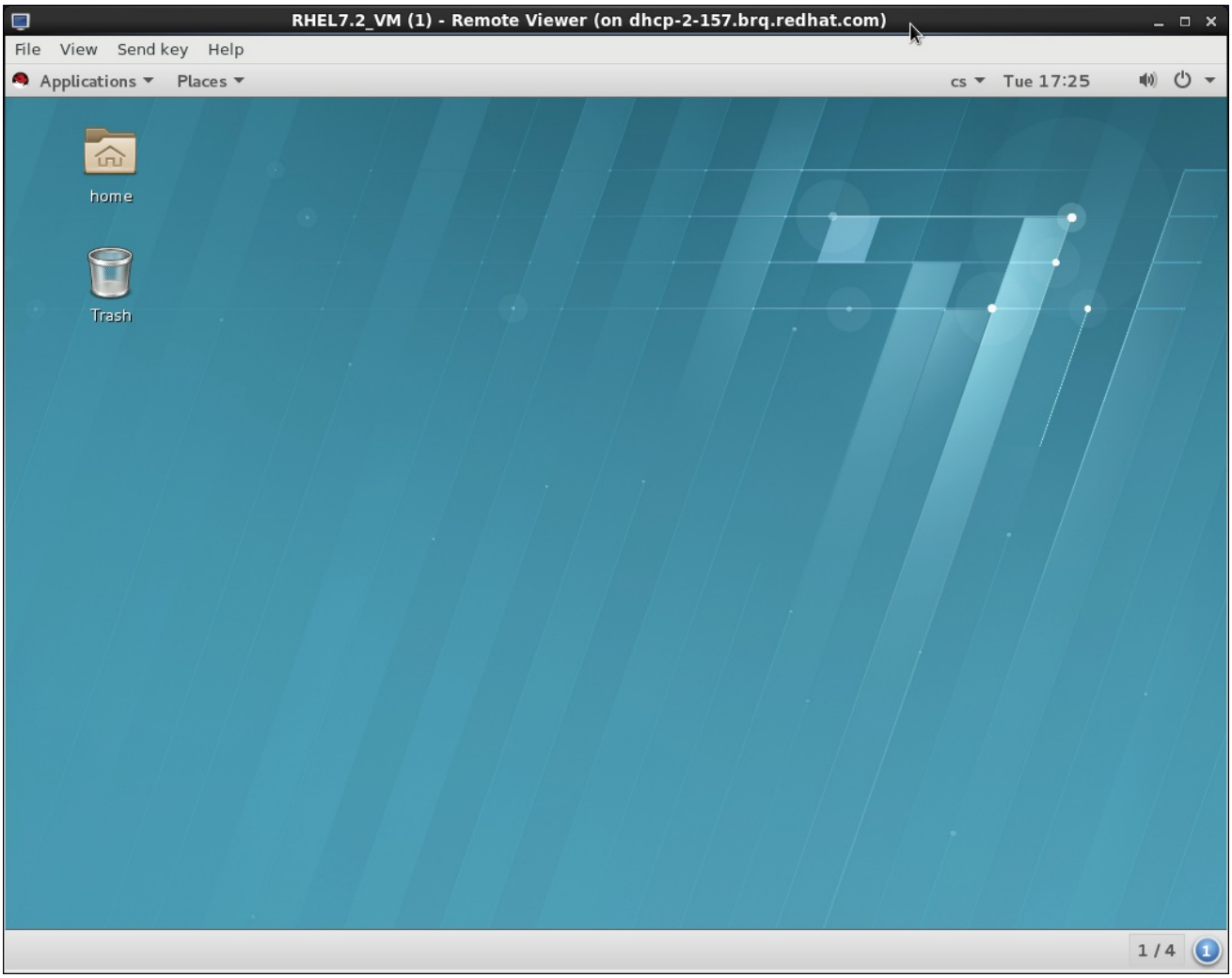
使用以下内容连接到名为 testguest2 的机器上的 VNC 服务器，它使用端口 5900 进行 VNC 通信：

```
# remote-viewer vnc://testguest2:5900
```

Interface

默认情况下，remote-viewer 接口仅提供与客户机交互的基本工具：

图 17.2. remote-viewer 接口示例



第 18 章 虚拟网络

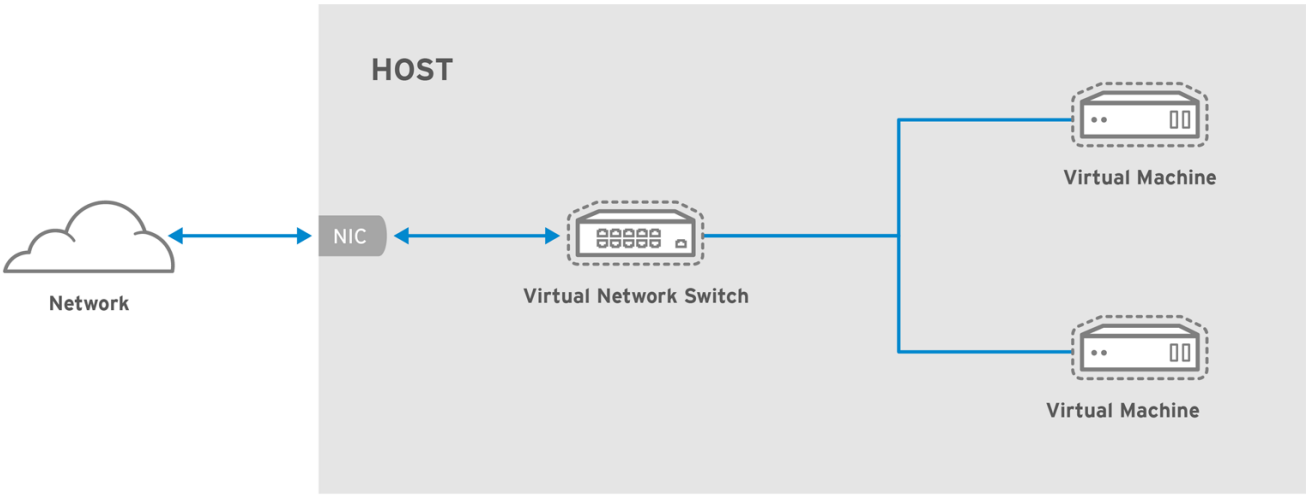
本章介绍了使用 libvirt 创建、启动、停止、删除和修改虚拟网络所需的概念。

libvirt 参考章节中可以找到其他信息

18.1. 虚拟网络切换

libvirt 虚拟网络使用了虚拟网络交换机 的概念。虚拟网络交换机是在主机物理计算机服务器上运行的软件构造，虚拟机(guests)连接。客户机的网络流量通过这个交换机定向：

图 18.1. 使用两个客户机的虚拟网络交换机



Linux 主机物理机器服务器，代表作为网络接口的虚拟网络交换机。当 libvirtd 守护进程(libvirtd)首次安装并启动时，代表虚拟网络交换机的默认网络接口为 virbr0。

与任何其他接口一样，可以使用 ip 命令查看这个 virbr0 接口：

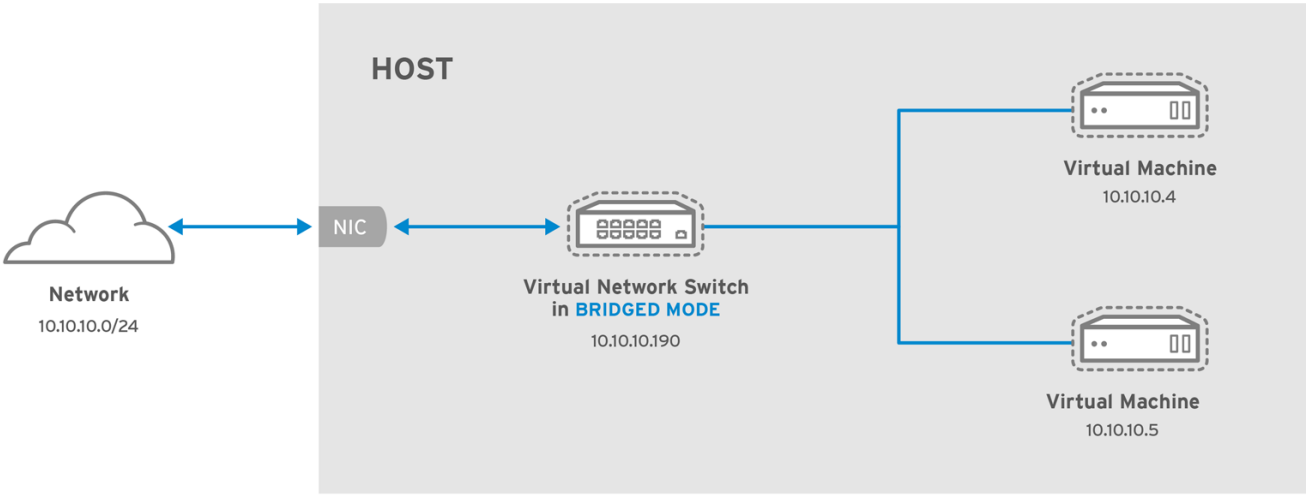
```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

18.2. 网桥模式

使用 Bridged 模式 时，所有 guest 虚拟机都会显示在与主机物理机器相同的子网中。同一物理网络中的所有其他物理机器都了解虚拟机，并可以访问虚拟机。桥接在 OSI 网络模型的第 2 层操作。

通过将多个物理接口与绑定一起加入，可以在管理程序中使用多个物理接口。然后，绑定会添加到桥接中，然后将客户机虚拟机添加到网桥中。但是，绑定驱动程序有多种操作模式，只有少数一种模式可用于虚拟机正在使用的网桥。

图 18.2. 使用桥接模式的虚拟网络交换机



警告

应该用于 guest 虚拟机的唯一绑定模式是模式 1、模式 2 和模式 4。在任何情况下，不应使用模式 0、3、5 或 6。另请注意，mii-monitoring 应该用来监控绑定模式，因为 arp-monitoring 无法工作。

有关绑定模式的详情，请参考有关 绑定模式、或 Red Hat Enterprise Linux 6 部署指南 的知识库文章。

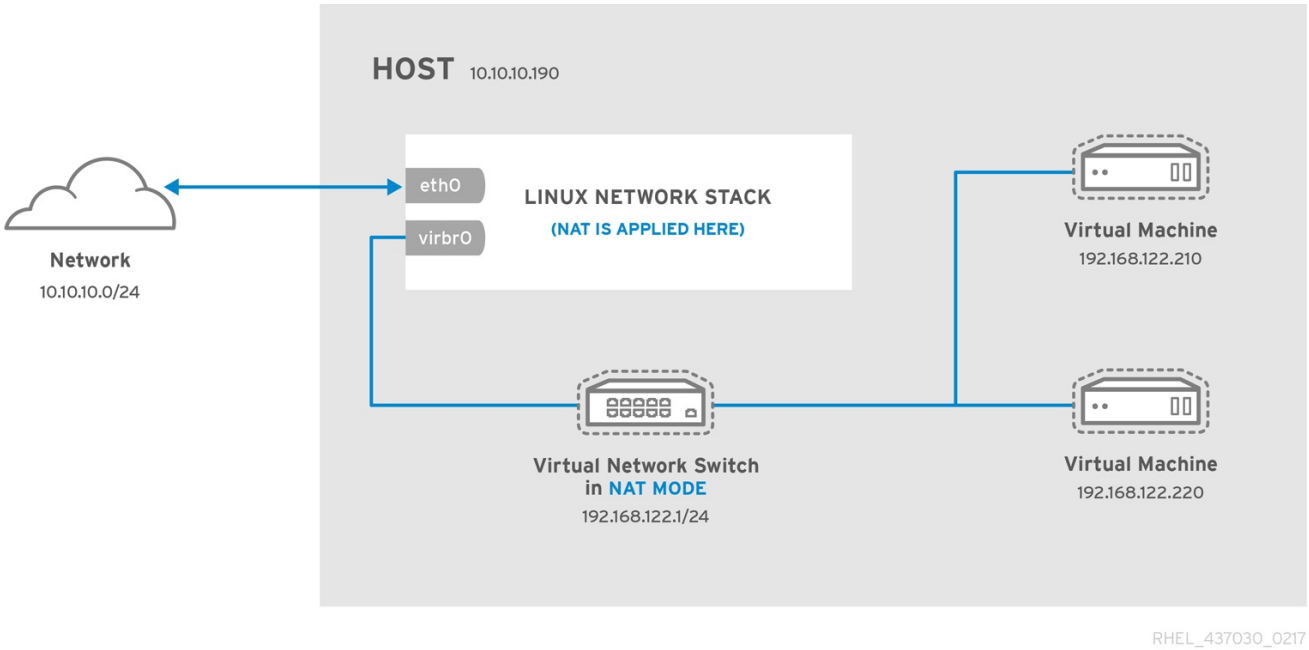
有关 bridge_opts 参数的详细说明，请参阅 Red Hat Virtualization 管理指南。

18.3. 网络地址转换模式

默认情况下，虚拟网络交换机在 NAT 模式下操作。它们使用 IP 伪装而不是 SNAT(Source-NAT)或 DNAT(Destination-NAT)。IP 伪装可让连接的虚拟机使用主机物理机器 IP 地址与任何外部网络通信。默

认情况下，当虚拟网络交换机以 NAT 模式运行时，将外部放在主机物理机器的计算机不能与客户机通信，如下图所示：

图 18.3. 使用带两个客户机的 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会导致交换机无法进行通信。

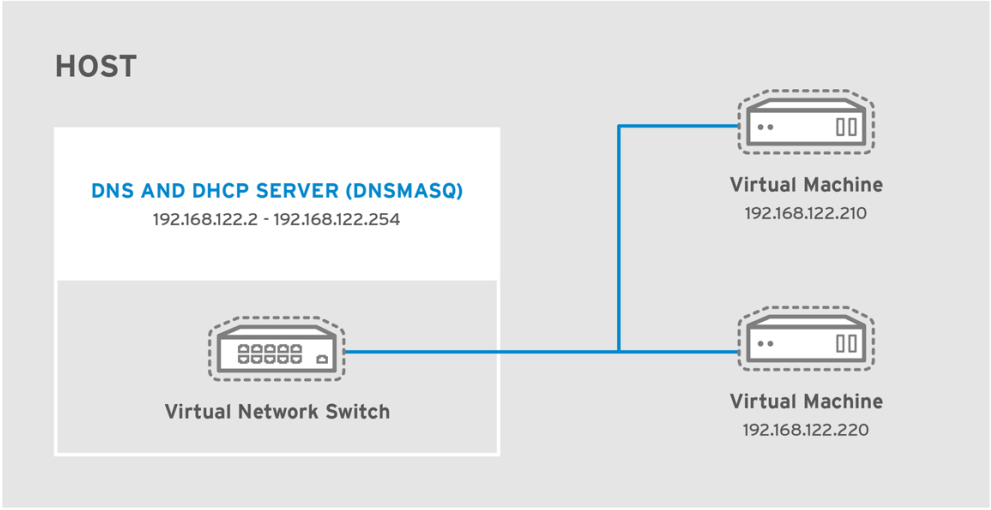
如果交换机没有运行，您可以为转发模式 NAT 设置 th 公共 IP 范围，以便通过运行以下命令创建端口伪装范围：

```
# iptables -j SNAT --to-source [start]-[end]
```

18.3.1. DNS 和 DHCP

IP 信息可以通过 DHCP 分配给客户机。为此，可以为虚拟网络交换机分配地址池。libvirt 使用 dnsmasq 程序来执行此操作。libvirt 自动配置并启动 dnsmasq 实例，用于需要它的每个虚拟网络交换机。

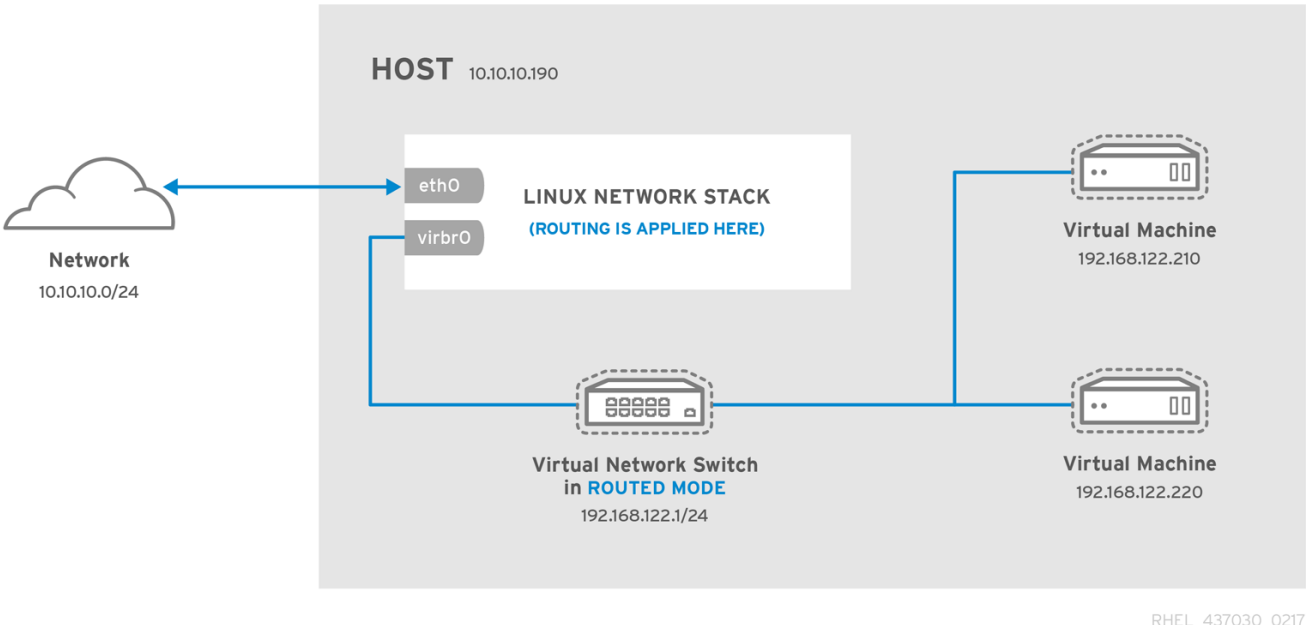
图 18.4. 运行 dnsmasq 的虚拟网络交换机



18.4. 路由模式

当使用 Routed 模式时，虚拟交换机连接到连接到主机物理 LAN 的物理 LAN，并在不使用 NAT 的情况下传递流量。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。使用这种模式时，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。这一情形并非始终如物理网络上的其他主机物理计算机知道虚拟机，无需手动物理路由器配置，而且无法访问虚拟机。路由模式在 OSI 网络模型的第 3 层运行。

图 18.5. 路由模式下的虚拟网络交换机

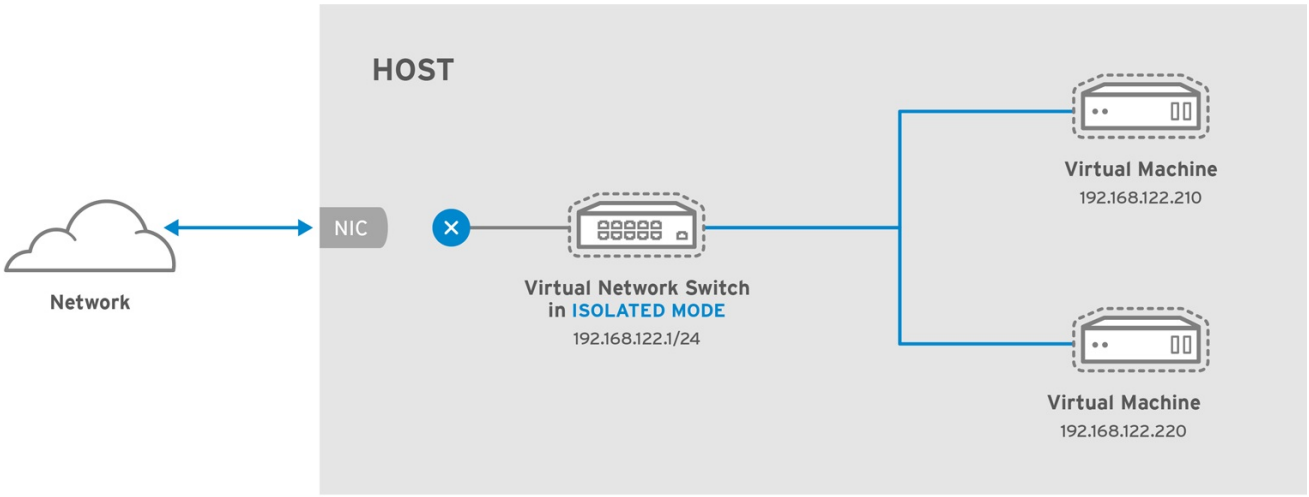


18.5. 隔离模式

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，而且主机物理计算机，但其流量不会在主机物理计算机之外传递，也不能从主机物理机器接收流量。在这个模式中需要使用 dnsmasq 的基本

功能，如 DHCP。但是，即使这个网络与任何物理网络隔离，DNS 名称仍会解决。因此，当 DNS 名称解析但 ICMP echo request(ping)命令失败时可能会出现这种情况。

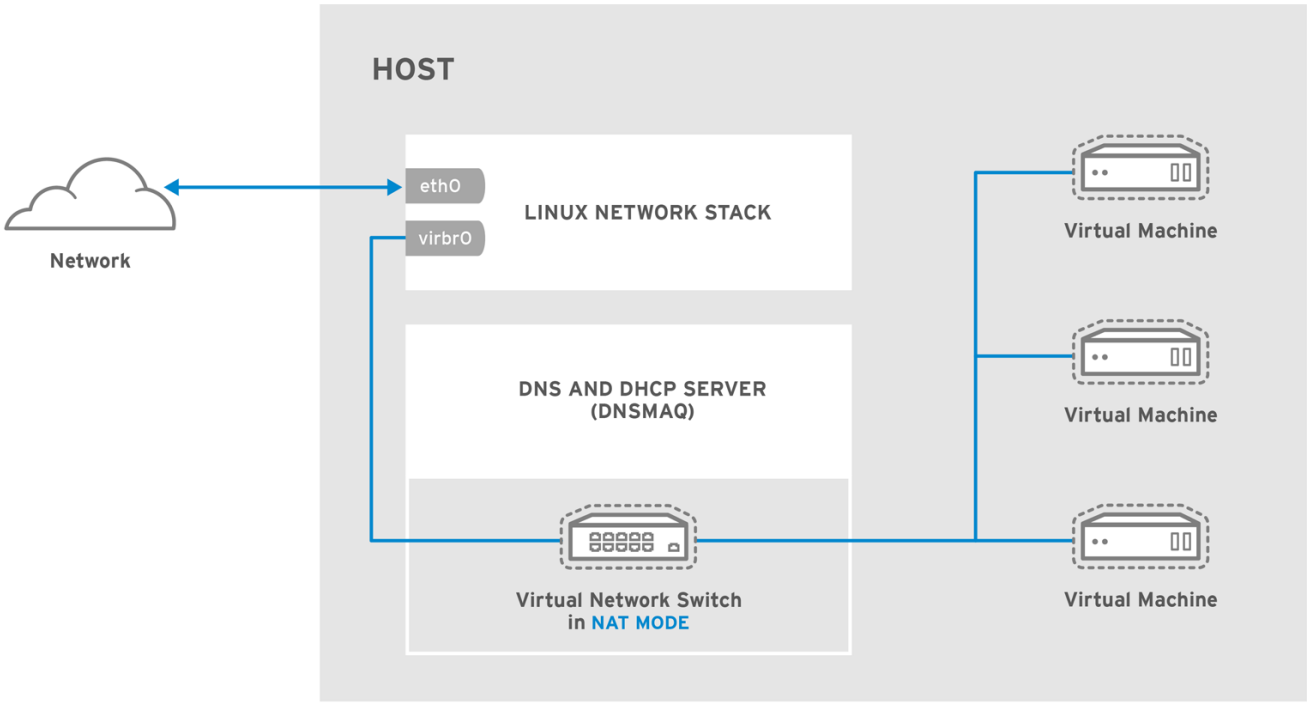
图 18.6. 以隔离模式的虚拟网络交换机

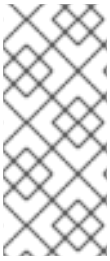


18.6. 默认配置

首次安装 libvirtd 守护进程(libvirtd)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，以便安装的客户机能够通过主机物理机器与外部网络通信。下图演示了 libvirtd 的这个默认配置：

图 18.7. 默认 libvirt 网络配置





注意

虚拟网络可以限制到特定的物理接口。这对具有多个接口的物理系统（例如 eth 0、eth1 和 eth2）可能很有用。这只适用于路由和 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。

18.7. 通用场景示例

本节演示了不同的虚拟网络模式，并提供了一些示例场景。

18.7.1. 网桥模式

网桥模式在 OSI 模型的第 2 层操作。使用时，所有 guest 虚拟机将会显示在与主机物理机器相同的子网中。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，使虚拟机和物理机之间的差别对最终用户透明。
- 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。
- 部署必须可以被现有物理网络轻松访问的客户机虚拟机。将客户机虚拟机放置到必须在现有广播域（如 DHCP）中访问服务的物理网络中。
- 将客户机虚拟机连接到使用 VLAN 的现有网络。

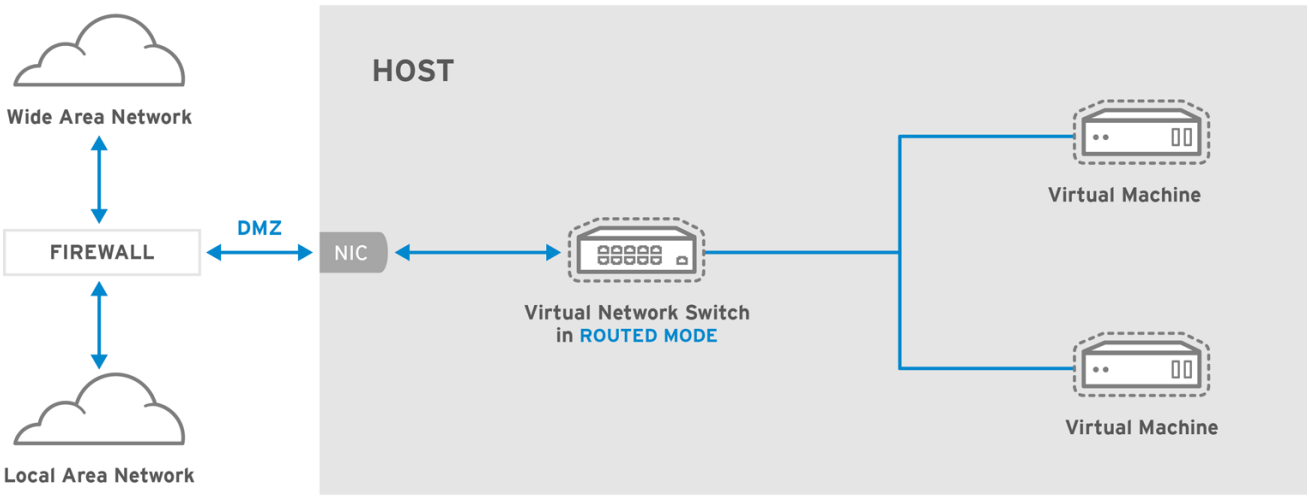
18.7.2. 路由模式

本节提供有关路由模式的信息。

DMZ

出于安全原因，请考虑一个或多个节点放置在受控子网中的网络。部署特殊的子网（如 ）是一个常见做法，且子网被称为 DMZ。有关此布局的详情，请参考下图：

图 18.8. DMZ 配置示例

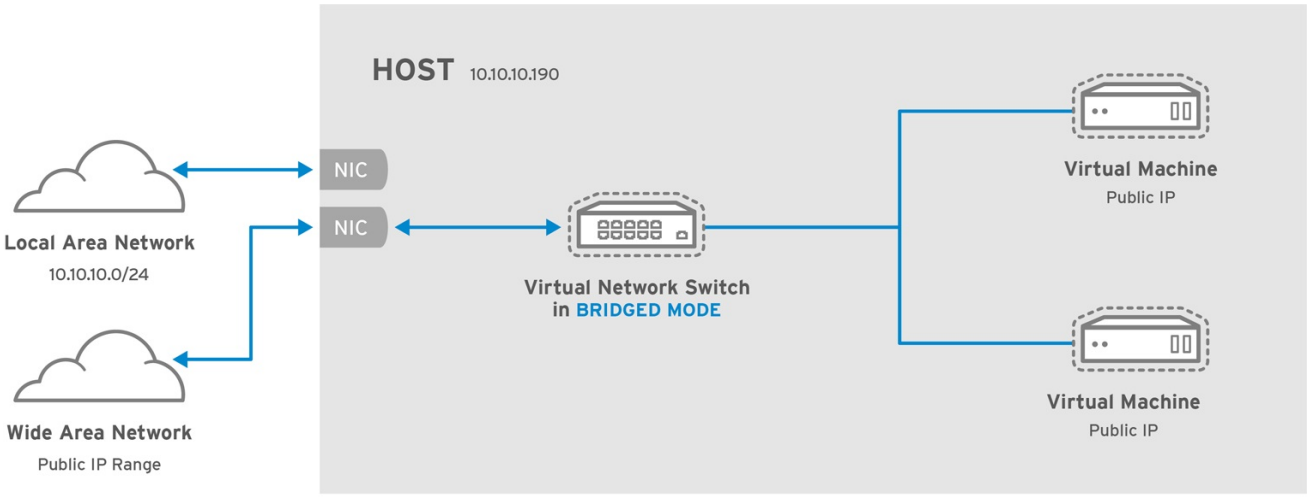


DMZ 中的主机物理计算机通常为 WAN（外部）主机物理计算机以及 LAN（内部）主机物理计算机提供服务。由于这需要它们可以被多个位置访问，并考虑这些位置可根据其安全性和信任级别以不同的方式控制和操作，路由模式是此环境的最佳配置。

虚拟服务器托管

考虑托管具有多个主机物理计算机的虚拟服务器，每个主机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 拥有自己的公共 IP 地址，但主机物理计算机使用专用 IP 地址作为 guest 管理仅可以由内部管理员执行。请参阅下图以了解这种情况：

图 18.9. 托管示例配置的虚拟服务器



18.7.3. NAT 模式

NAT（网络地址转换）模式是默认模式。当不需要直接网络可见性时，它可用于测试。

18.7.4. 隔离模式

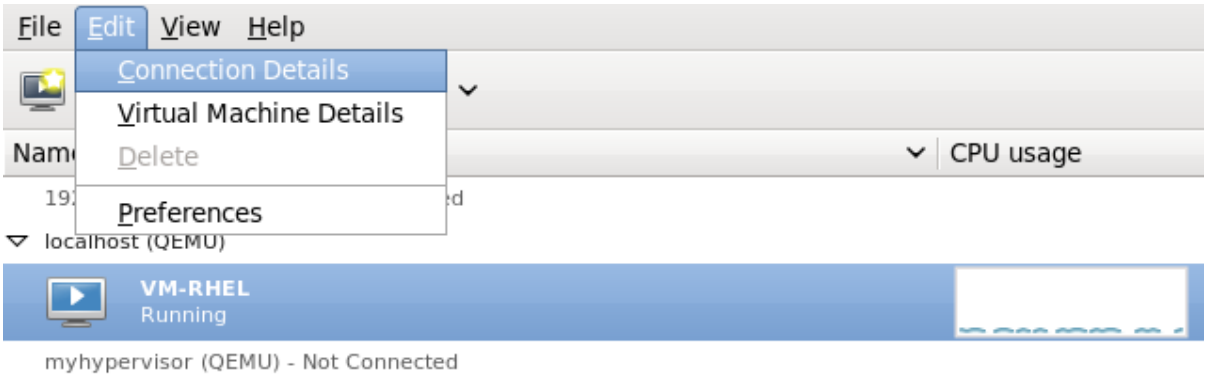
隔离模式允许虚拟机仅相互通信。它们无法与物理网络交互。

18.8. 管理虚拟网络

在您的系统中配置虚拟网络：

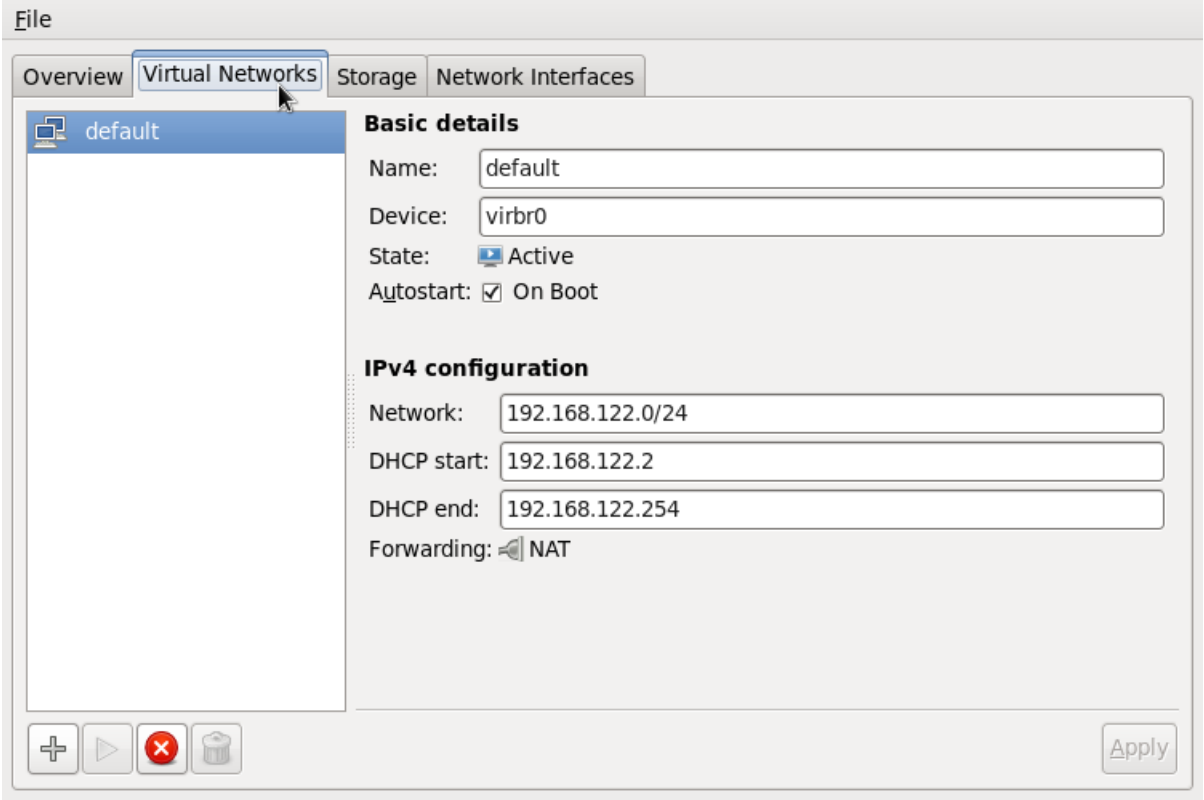
1. 在 Edit 菜单中，选择 Connection Details。

图 18.10. 选择主机物理机器详情



2. 这将打开 Connection Details 菜单。点 Virtual Networks 标签页。

图 18.11. 虚拟网络配置



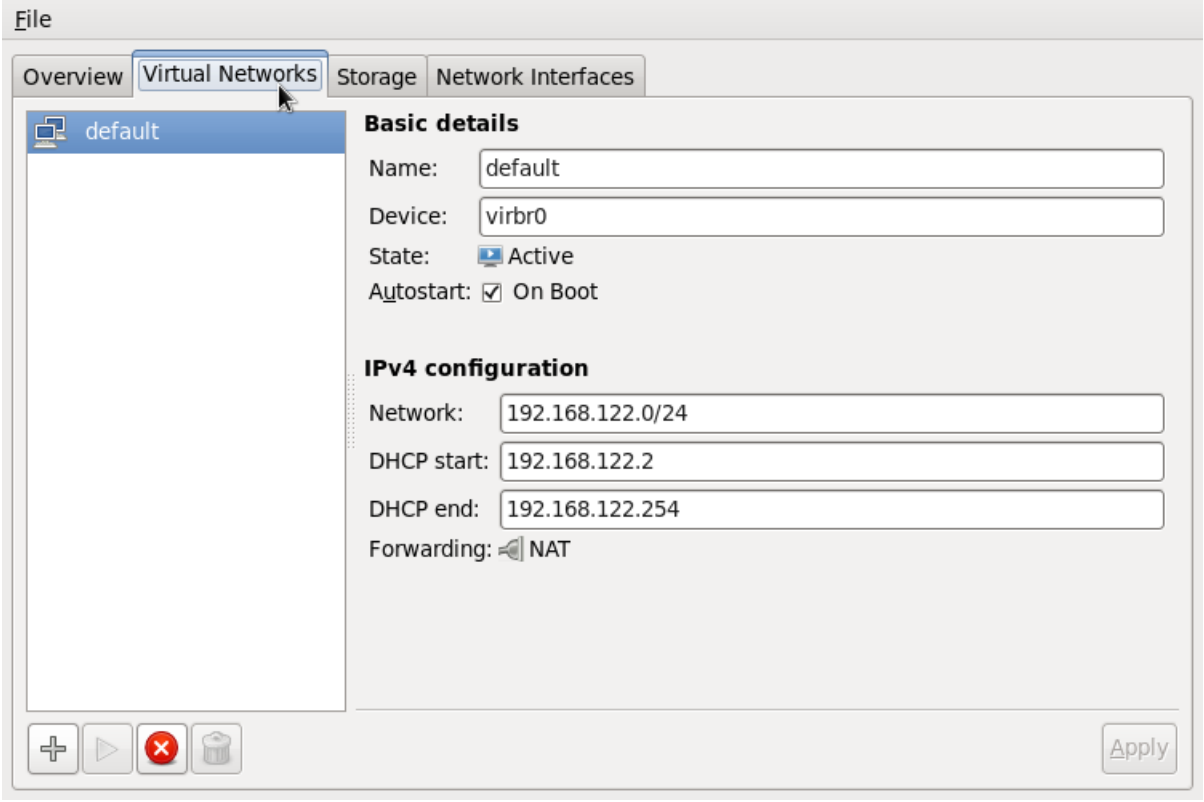
3. *所有可用的虚拟网络都列在菜单的左侧框中。您可以通过从此框中选择并编辑虚拟网络，以编辑虚拟网络的配置。*

18.9. 创建虚拟网络

在您的系统中创建虚拟网络：

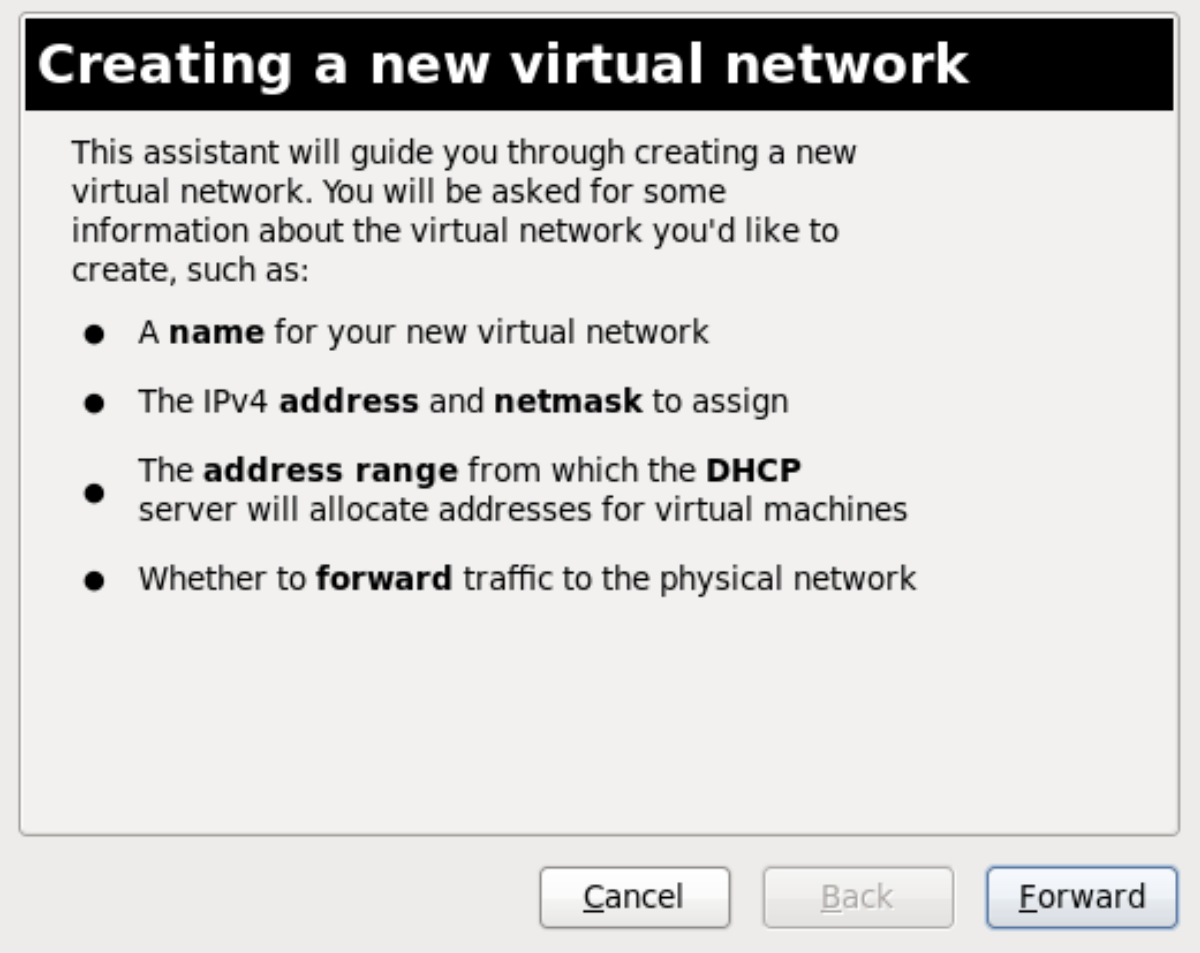
1. *从" 连接详细信息 "菜单内打开" 虚拟网络 "选项卡。单击 Add Network 按钮，具体操作为加号(+)图标。有关详情请参阅 第 18.8 节 “管理虚拟网络”。*

图 18.12. 虚拟网络配置



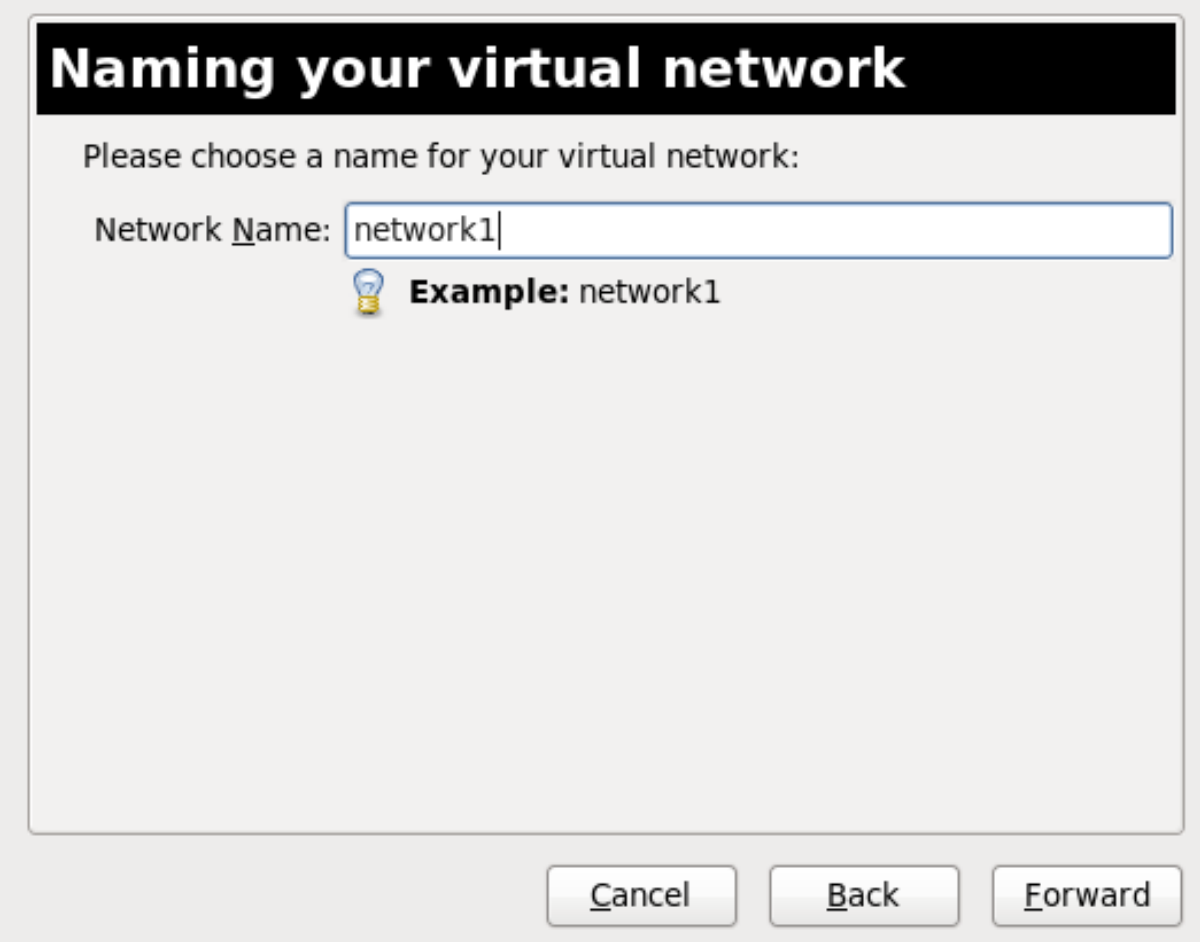
*这将打开 **Create a new virtual network** 窗口。单击 **"下一步"** 以继续。*

图 18.13. 创建新的虚拟网络



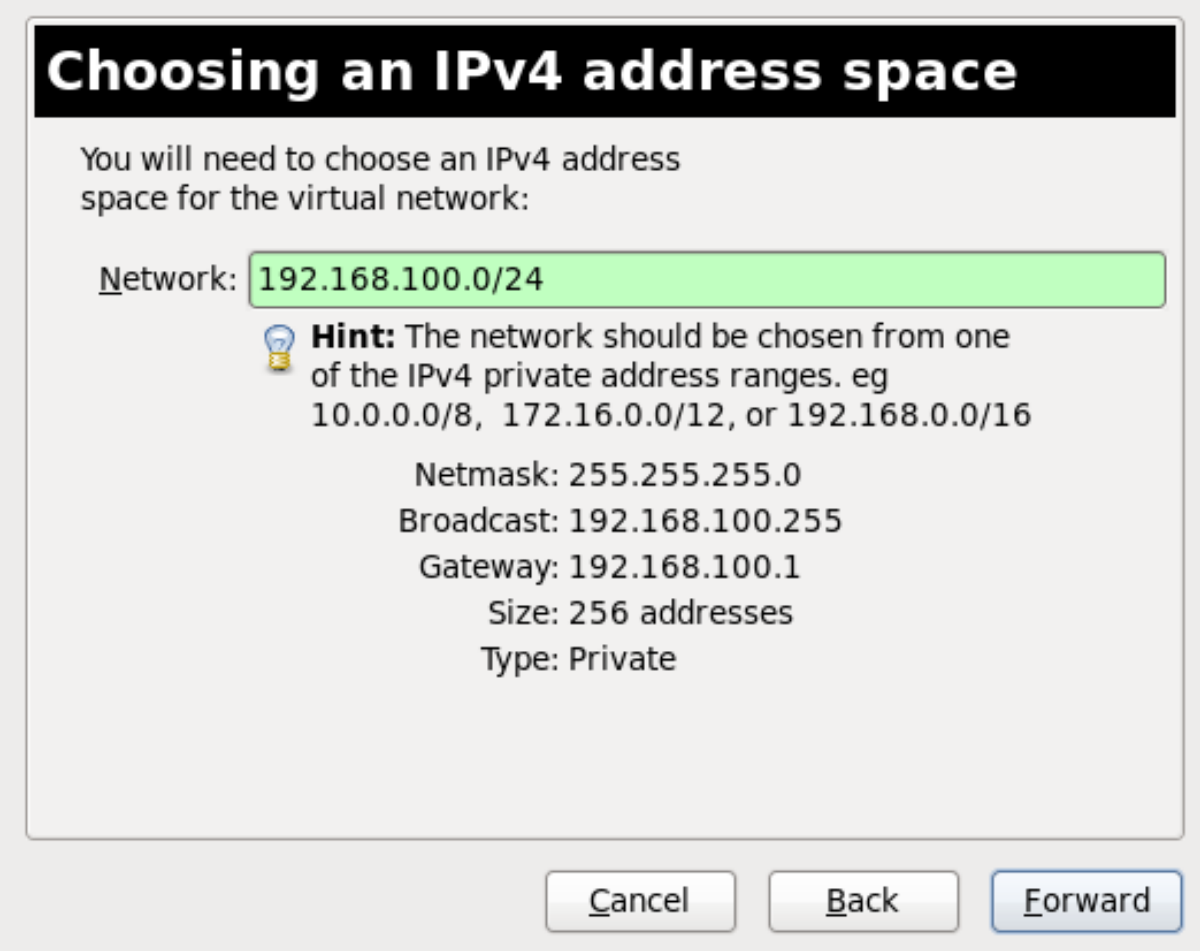
2. *为您的虚拟网络输入合适的名称并单击 转发。*

图 18.14. 命名您的虚拟网络



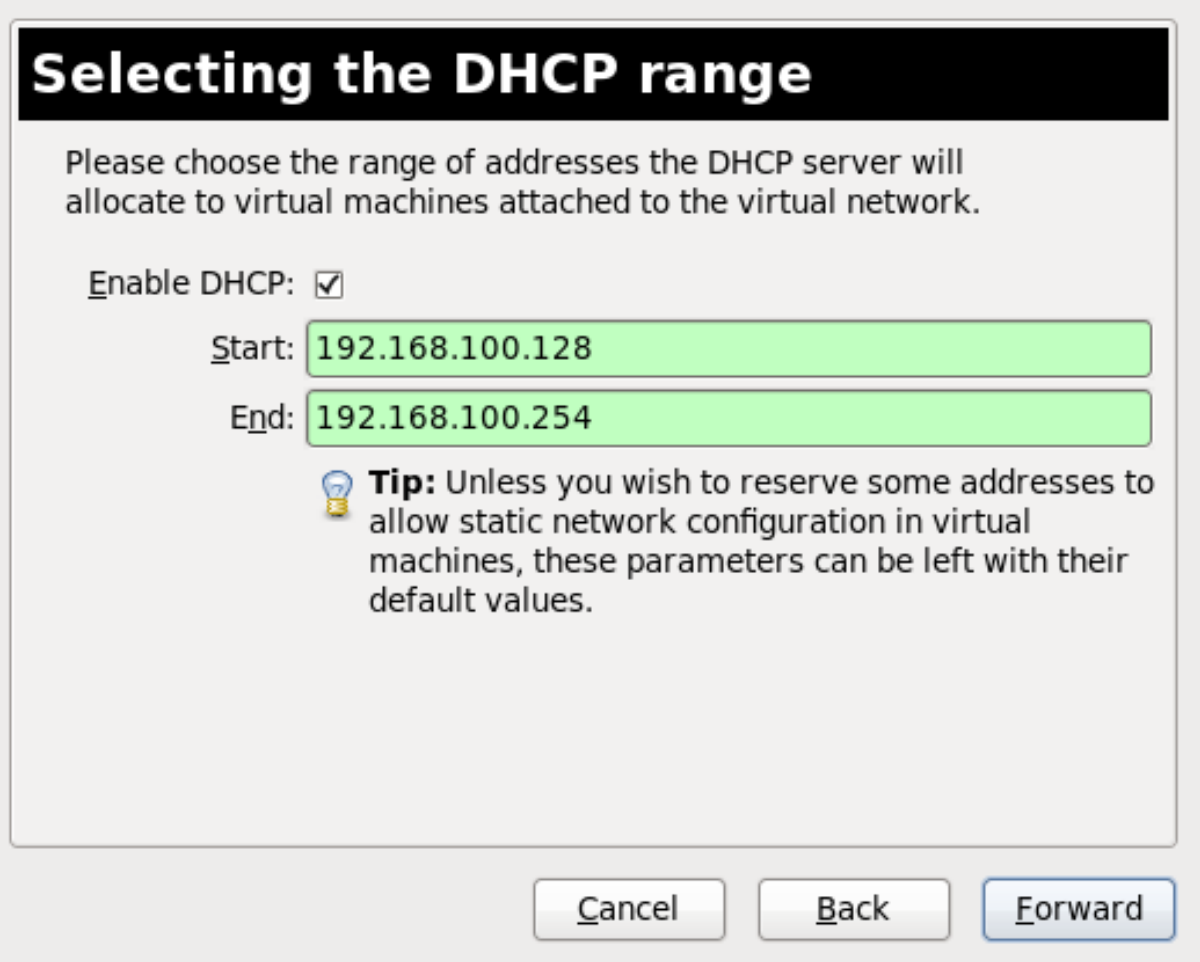
3. *为您的虚拟网络输入 IPv4 地址空间 并单击下一步。*

图 18.15. 选择 IPv4 地址空间



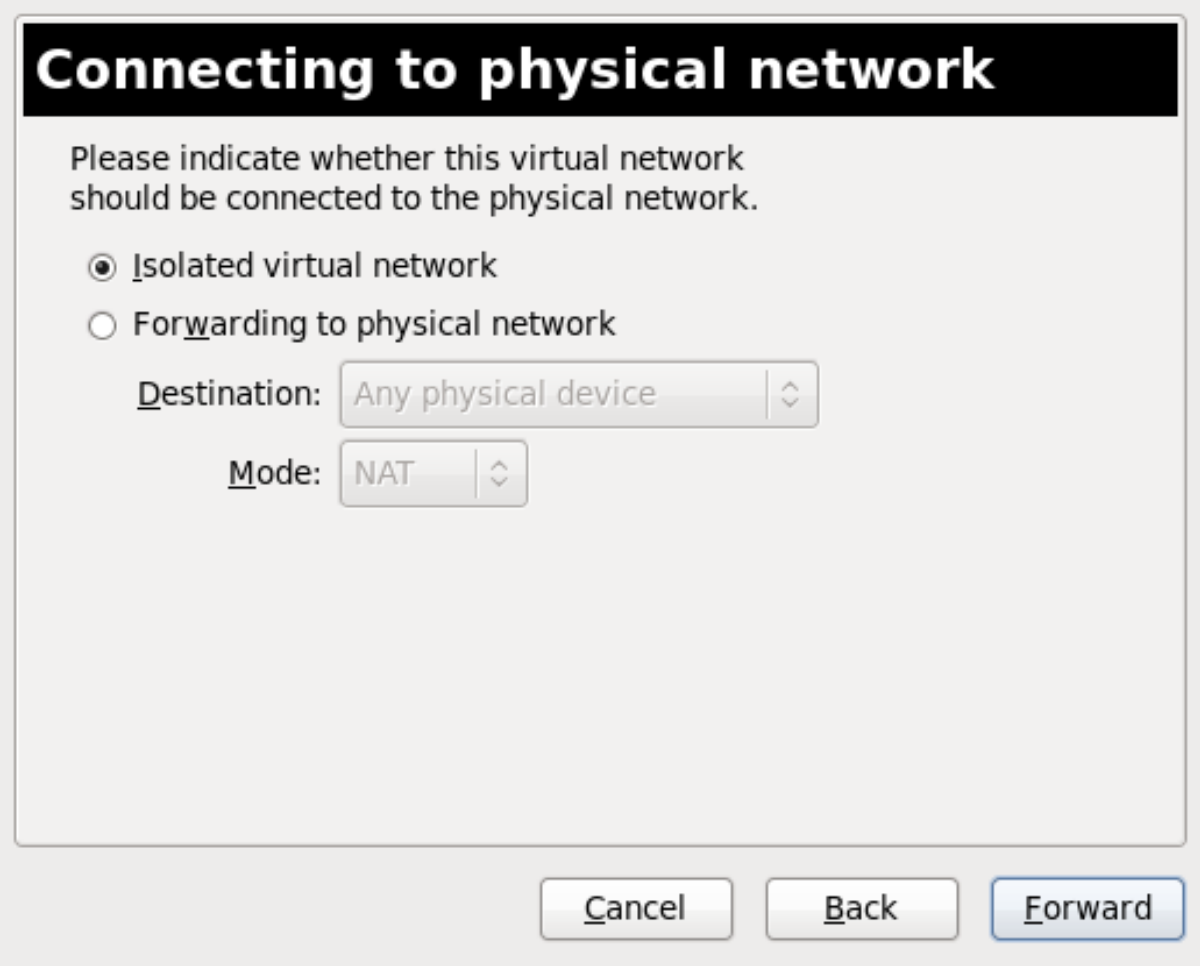
4. *通过指定开始和结束 IP 地址范围，为您的虚拟网络定义 DHCP 范围。 单击 "下一步" 以继续。*

图 18.16. 选择 DHCP 范围



5. *选择虚拟网络应如何连接到物理网络。*

图 18.17. 连接到物理网络

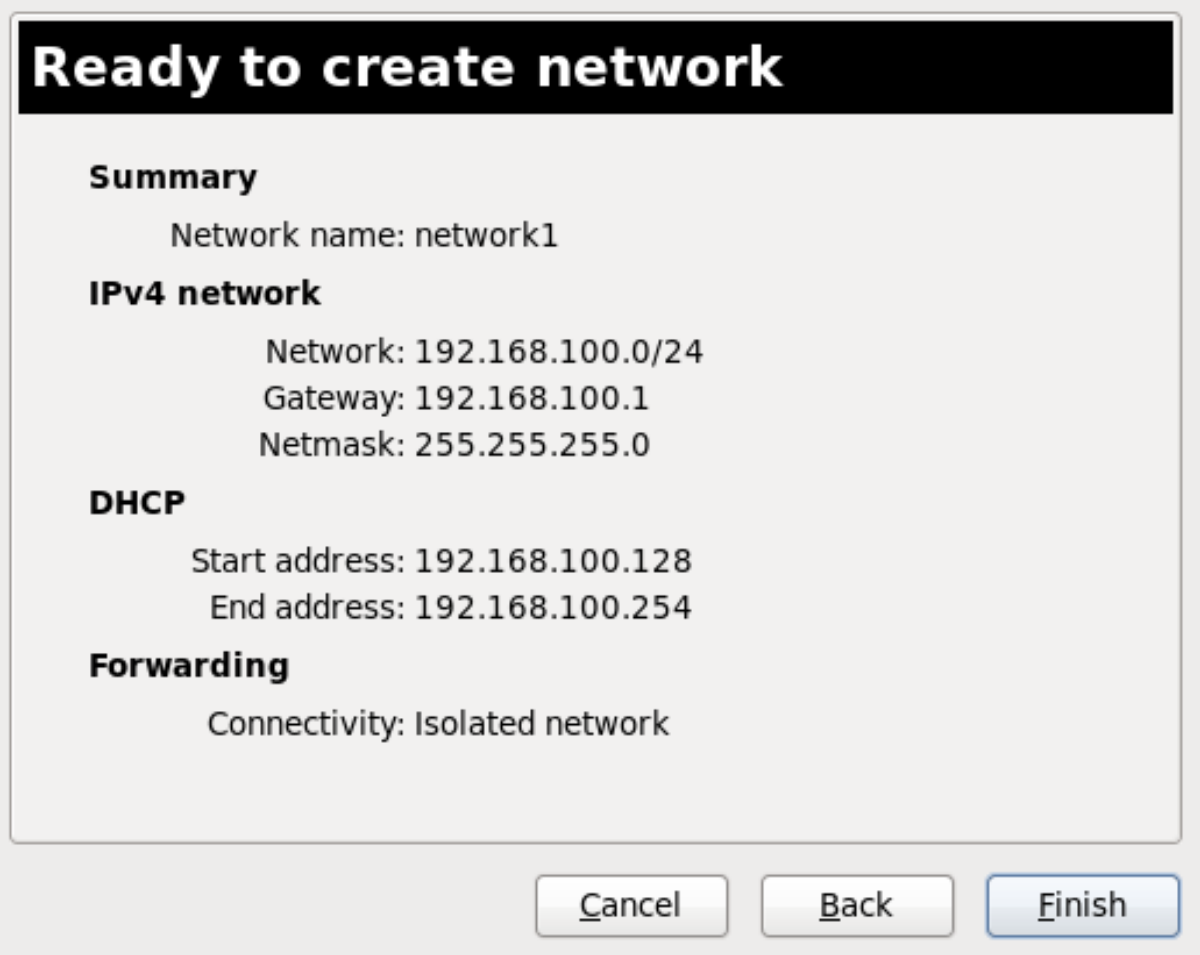


如果您选择 转发到物理网络，请选择 目标 应该是 任何物理设备 还是特定物理设备。另外，选择 Mode 应为 NAT 还是 Routed。

单击 "下一步" 以继续。

6. *您现在已准备好创建网络。检查您网络的配置并单击" 完成"。*

图 18.18. 准备好创建网络



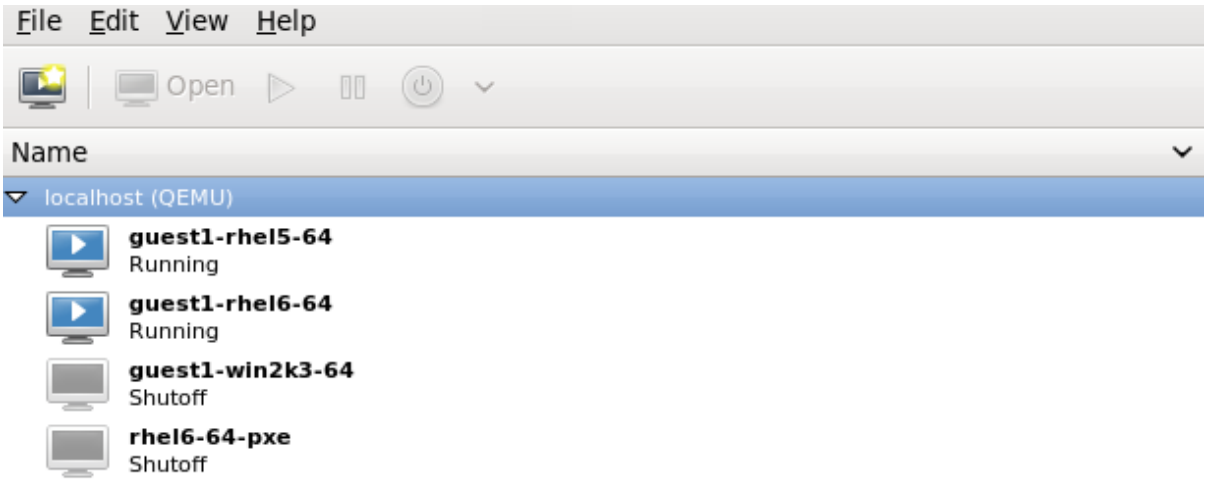
7. *新的虚拟网络现在包括在 Connection Details 窗口的 Virtual Networks 选项卡中。*

18.10. 将虚拟网络附加到虚拟机

将虚拟网络附加到客户端：

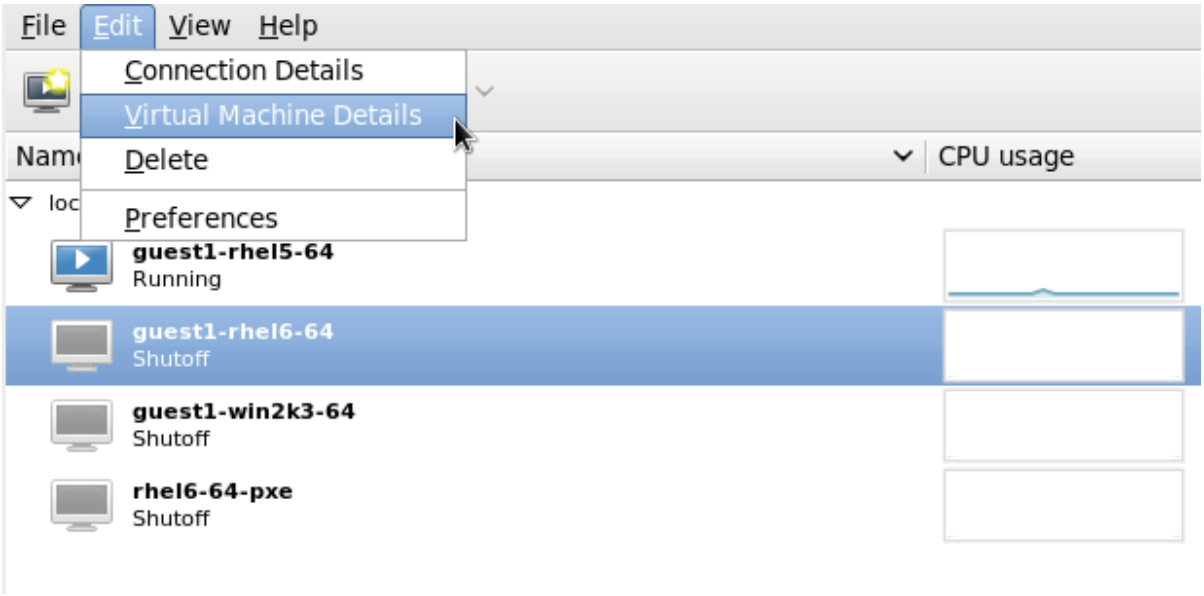
1. *在 Virtual Machine Manager 窗口中，突出显示将分配网络的 guest。*

图 18.19. 选择要显示的虚拟机



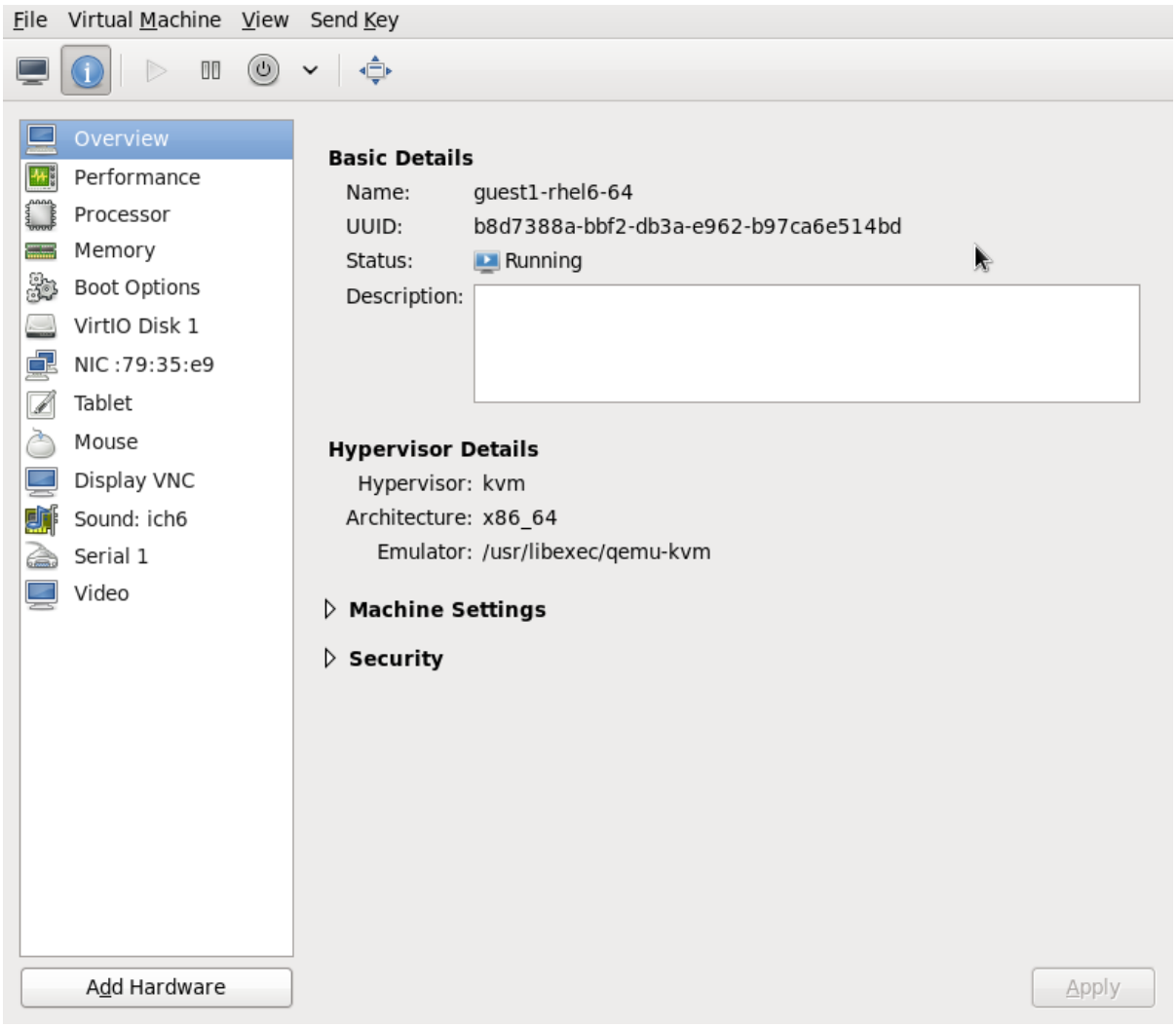
2. ***在 Virtual Machine Manager Edit 菜单中，选择 Virtual Machine Details。***

图 18.20. 显示虚拟机详情



3. ***点 Virtual Machine Details 窗口中的 Add Hardware 按钮。***

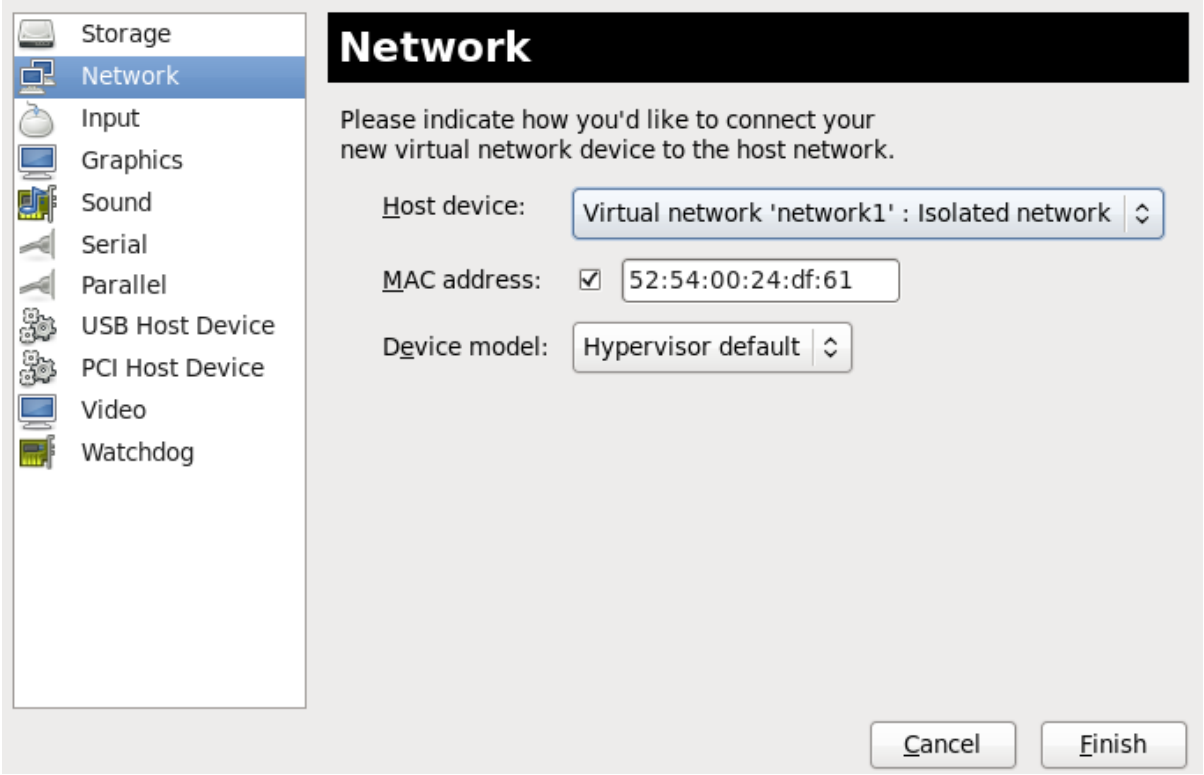
图 18.21. Virtual Machine Details 窗口



4.

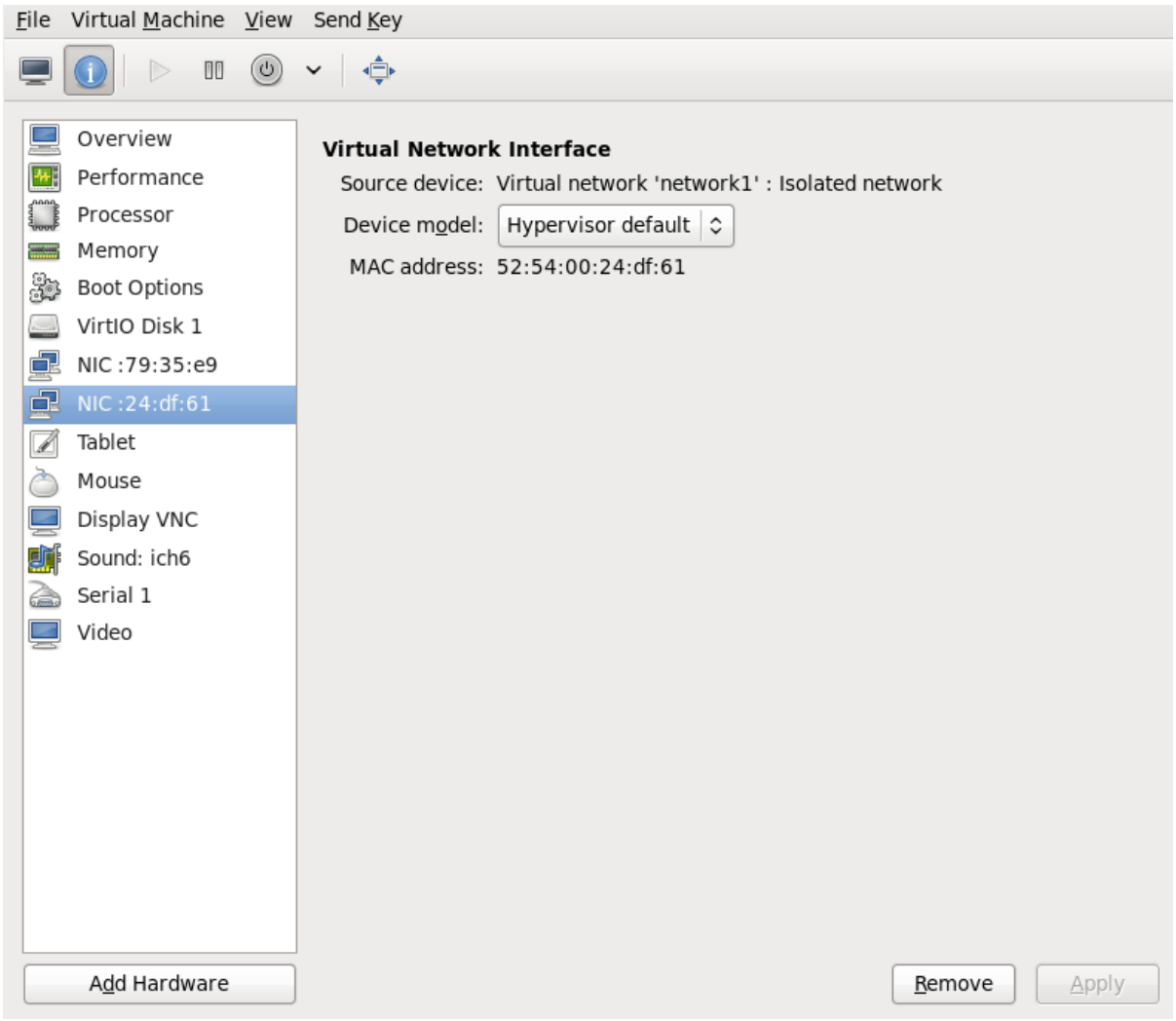
在 Add new virtual hardware 窗口中，从左侧窗格中选择 Network，然后从 主机设备 菜单中选择您的网络名称（本例中为network1 ），然后单击" 完成"。

图 18.22. 从 Add new virtual hardware 窗口中选择您的网络



5. *现在，新网络显示为虚拟网络接口，在启动时将显示为 guest。*

图 18.23. guest 硬件列表中显示的新网络



18.11. 将虚拟 NIC 直接附加到物理接口

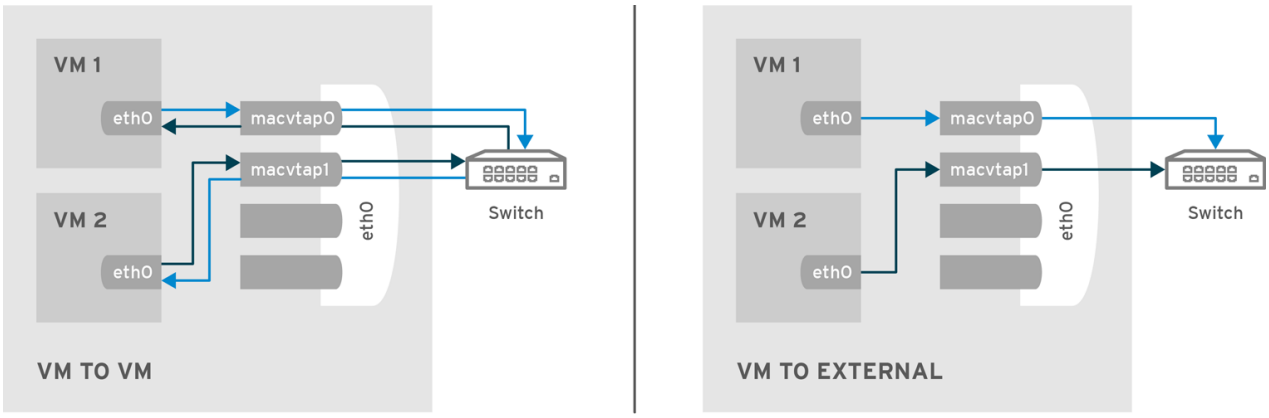
作为默认 NAT 连接的替代方案，您可以使用 macvtap 驱动程序将 guest 的 NIC 直接附加到主机的指定物理接口。这并不是与 设备分配 （也称为 passthrough）混淆。macvtap 连接具有以下模式，每种模式都有不同的优点和用例：

物理接口交付模式

VEPA

在虚拟以太网端口聚合器(VEPA)模式中，来自客户机的所有数据包都发送到外部交换机。这可让用户通过交换机强制进行客户机流量。要使 VEPA 模式正常工作，外部交换机还必须支持 hairpin 模式，这样可确保目标作为虚拟客户机在同一主机计算机上的数据包通过外部交换机发回到主机。

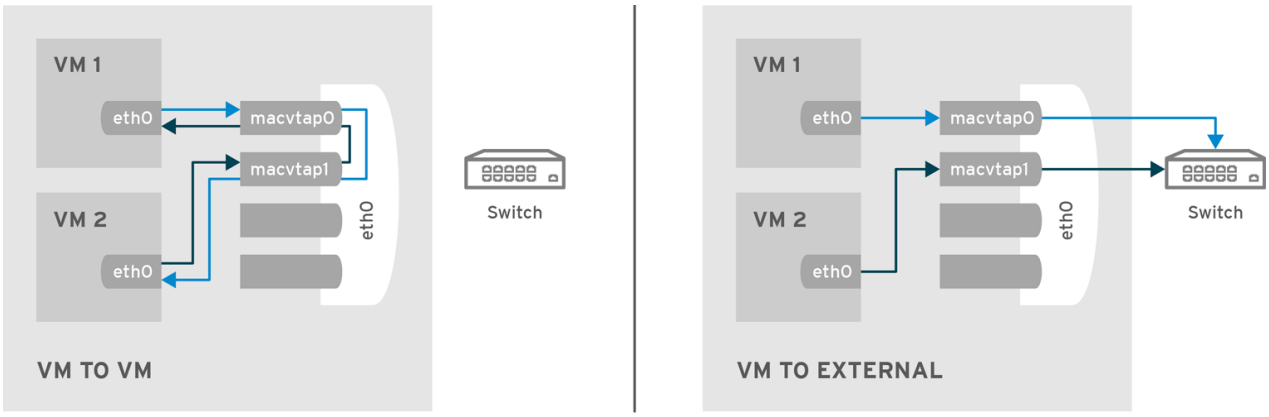
图 18.24. VEPA 模式



bridge

目的地在与源客户机相同的主机机器上直接传送到目标 macvtap 设备。源设备和目的地设备都需要处于网桥模式才能成功进行直接传输。如果其中任何一个设备处于 VEPA 模式，则需要使用具有功能性功能的外部交换机。

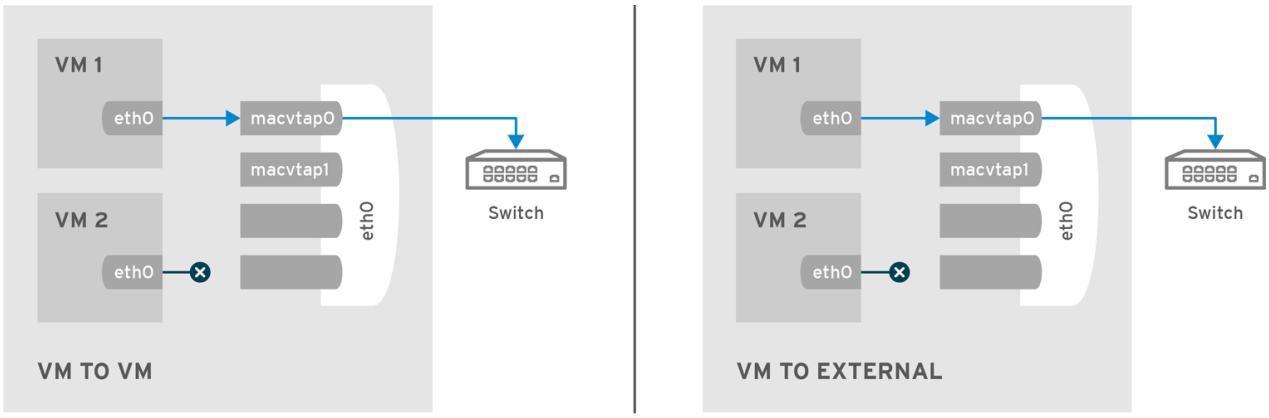
图 18.25. 网桥模式



private

所有数据包都发送到外部交换机，只有在通过外部路由器或网关发送时，它们才会传送到同一主机上的目标 guest，并将它们发回到主机。私有模式可用于防止单一主机上的各个客户机相互通信。如果源或目标设备处于私有模式，则执行此步骤。

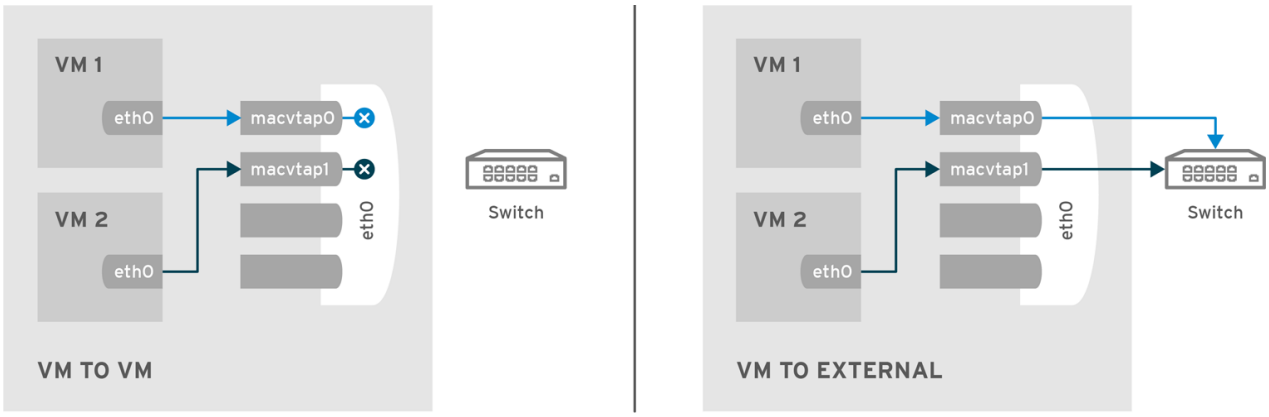
图 18.26. 私有模式



passthrough

此功能将物理接口设备或 SR-IOV 虚拟功能(VF)直接附加到客户机，而不会丢失迁移功能。所有数据包直接发送到指定的网络设备。请注意，一个网络设备只能被传递给一个客户端，因为网络设备无法在 passthrough 模式中的客户机之间共享。

图 18.27. Passthrough 模式



通过更改域 xml 文件来配置四个模式。打开该文件后，按如下所示更改模式设置：

```
<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以由主机物理机器所连接到的物理接口来管理。

如果交换机符合 IEEE 802.1Qbg 标准，接口可以还有其他参数。virtualport 元素的参数在 IEEE

802.1Qbg 标准中更详细地阐述。值特定于网络，应当由网络管理员提供。在 802.1Qbg 术语中，虚拟工作站(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。另外，如果交换机符合 IEEE 802.1Qbh 标准，则该值特定于网络，并且应由网络管理员提供。

虚拟工作站类型

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，赋予值 0。

typeid

VSI Type ID 标识 VSI 类型特征，以优化网络访问。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

在创建 VSI 实例（这是虚拟机虚拟接口）时，将生成 VSI 实例 ID 标识符。这是全局唯一标识符。

profileid

配置集 ID 包含要应用于此接口的端口配置集的名称。此名称由端口 profile 数据库解析为来自端口配置集的网络参数，这些网络参数将应用到此接口。

通过更改 domain xml 文件来配置这四种类型。打开该文件后，按如下所示更改模式设置：

```
<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 显示在此处：

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private'/>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
</devices>
...
```

18.12. 应用网络过滤

本节介绍了 libvirt 的网络过滤器、目标、概念和 XML 格式。

18.12.1. 简介

网络过滤的目的是使虚拟化系统的管理员能够在虚拟机上配置并强制实施网络流量过滤规则，并管理虚拟机允许发送或接收的网络流量参数。当虚拟机启动时，网络流量过滤规则会应用到主机物理机器上。由于过滤规则不能从虚拟机内部绕过，因此它从虚拟机用户的角度看必须要。

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单独配置。这些规则在虚拟机启动时应用到主机物理机器，并可在虚拟机运行时进行修改。后者可通过修改网络过滤器的 XML 描述来实现。

多个虚拟机可以使用相同的通用网络过滤器。修改此类过滤器时，将更新引用此过滤器的所有运行中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，可以对为某些类型网络配置配置的个别网络接口上应用网络流量过滤规则。支持的网络类型包括：

- network
- Ethernet -- 必须用于桥接模式

- **bridge**

例 18.1. 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，可以包含对其他过滤器的引用、用于流量过滤的规则或两者的组合。以上引用的过滤器 clean-traffic 是仅包含对其他过滤器的引用且没有实际过滤规则的过滤器。由于可以使用对其他过滤器的引用，因此可以构建过滤器的树。使用 命令来查看 clean-traffic 过滤器： # virsh nwfilter-dumpxml clean-traffic。

如前文所述，一个网络过滤器可以被多个虚拟机引用。由于接口通常关联有各自的流量过滤规则，因此可以使用变量对过滤器的 XML 中描述的规则进行规范化。在本例中，变量名称在过滤器 XML 中使用，名称和值则位于引用过滤器的位置。

例 18.2. 扩展描述

在以下示例中，接口描述已使用参数 IP 和点号的 IP 地址作为值进行扩展。

```
<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>
```

具体示例中，clean-traffic 网络流量过滤器将用 IP 地址参数 10.0.0.1 表示，根据规则规定，来自此接口的所有流量将始终使用 10.0.0.1 作为源 IP 地址，这是此特定过滤器的一个目的。

18.12.2. 过滤链

过滤规则以过滤链的形式组织。这些链可以看作树结构，并将数据包过滤规则作为单个链中的条目 (branches)。

数据包在根链中启动其过滤器评估，然后在其他链中继续评估，从这些链返回，或被其中一个遍历链的过滤规则中丢弃或接受。

libvirt 的网络过滤系统会自动为每个虚拟机的网络接口创建单独的 root 链，供用户选择来激活流量过滤。用户可以编写在根链中直接实例化的过滤规则，也可以创建特定于协议的过滤链，以有效地评估特定于协议的规则。

存在以下链：

- root
- mac
- STP（跨度树协议）
- vlan
- ARP 和 rarp
- ipv4
- ipv6

可以使用协议名称创建评估 mac、stp、vlan、rp、ipv4 或 ipv6 协议的多个链。

> **例 18.3. ARP 流量过滤**
>
> 此示例允许指定名称 arp-xyz 或 arp-test 链，并在这些链中评估其 ARP 协议数据包。

以下过滤器 XML 显示了在 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rp 链中放置特定于 ARP 的规则（而非在根链中），即 ARP 以外的数据包协议不需要由 ARP 协议特定的规则进行评估。这提高了流量过滤的效率。但是，必须注意，只有给定协议的过滤规则放入链中，因为不会评估其他规则。例如，IPv4 规则不会在 ARP 链中评估，因为 IPv4 协议数据包不会遍历 ARP 链。

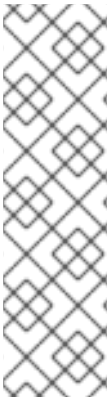
18.12.3. 过滤链优先级

如前文所述，在创建过滤规则时，所有链都连接到 root 链。这些链被链的顺序受链的优先级的影响。下表显示了可分配优先级及其默认优先级的链。

表 18.1. 过滤链默认优先级值

链（前缀）	默认优先级
stp	-810
mac	-800

链（前缀）	默认优先级
vlan	-750
ipv4	-700
ipv6	-600
arp	-500
rarp	-400



注意

在具有更高值值之前，访问具有较低优先级值的链。

还可以通过将 [-1000 到 1000] 范围内的值写入过滤器节点中的优先级(XML)属性，为表 18.1 “过滤链默认优先级值” 中列出的链分配自定义优先级。第 18.12.2 节 “过滤链”过滤器显示 -500 用于 arp 链的默认优先级，例如：

18.12.4. 在过滤器中使用变量

网络流量过滤子系统（MAC 和 IP）保留给使用的两个变量。

MAC 为网络接口的 MAC 地址指定。引用此变量的过滤规则将自动替换为接口的 MAC 地址。这在没有用户必须显式提供 MAC 参数的情况下可以正常工作。虽然可以指定与上述 IP 参数类似的 MAC 参数，但不建议这样做，因为 libvirt 知道将要使用的 MAC 地址。

参数 IP 代表虚拟机内部操作系统的 IP 地址应在给定接口上使用。目前为止，IP 参数是特殊的，因为 libvirt 守护进程将尝试确定在接口中使用的 IP 地址（因此，如果未明确提供该参数但引用）。有关 IP 地址检测的当前限制，请参考有关如何使用这个功能的限制 第 18.12.12 节 “限制” 部分。第 18.12.2 节 “过滤链” 中显示的 XML 文件包含过滤器 no-arp-spoofing，它是一个使用网络过滤器 XML 来引用 MAC 和 IP 变量的示例。

请注意，引用的变量始终使用字符 $ 作为前缀。变量值的格式必须是 XML 中标识的 filter 属性所预期的类型。在上例中，IP 参数必须以标准格式保存法律 IP 地址。如果未提供正确的结构，则过滤器变量将不会替换为值，并且阻止虚拟机启动或防止在使用热插拔时连接接口。各个 XML 属性预期的一些类型显示在 例 18.4 “变量类型示例” 示例中。

例 18.4. 变量类型示例

由于变量可以包含元素列表，例如，变量 IP 可以包含特定接口上有效的多个 IP 地址，例如，为 IP 变量提供多个元素的标记如下：

```
<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>
```

此 XML 文件创建了过滤器，以启用每个接口的多个 IP 地址。每个 IP 地址都将导致单独的过滤规则。因此，使用上面的 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>
```

由于可以访问保存元素列表的变量的各个元素，因此下面的过滤规则访问变量 DSTPORTS 的第二代元素。

```
<rule action='accept' direction='in' priority='500'>
  <udp dstportstart='$DSTPORTS[1]'/>
</rule>
```

例 18.5. 使用各种变量

因为可以创建过滤规则，使用表示法 $VARIABLE[@<iterator id="x">] 表示不同列表的规则组合。以下规则允许虚拟机接收一组在 DSTPORTS 中指定的端口上的流量，这些端口来自 SRCIPADDRESSES 中指定的源 IP 地址集合。该规则使用两个独立的迭代器生成变量 DSTPORTS 的所有元素和 SRCIPADDRESSES 的不同元素。

```
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]'/>
</rule>
```

将 concrete 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] 将值分配给变量，然后导致创建的所有地址和端口组合，如下所示：

- 10.0.0.1, 80
- 10.0.0.1, 8080
- 11.1.2.3, 80
- 11.1.2.3, 8080

使用单个迭代器访问同一变量，例如使用表示法 $SRCIPADDRESSES[@1] 和 $DSTPORTS[@1]，从而可以并行访问这两个列表并产生以下组合：

- 10.0.0.1, 80
- 11.1.2.3, 8080

注意

$VARIABLE 是 $VARIABLE[@0] 的速记。前者表示法总是使用迭代器 id="0" 假定其角色（如本节顶部的"打开段落所示）。

18.12.5. 自动 IP 地址检测和 DHCP Snooping

本节介绍自动 IP 地址检测和 DHCP 侦听的信息。

18.12.5.1. 简介

如果引用了变量 IP，但没有为其分配值，则虚拟机的接口上使用的 IP 地址的检测会被自动激活。变量 CTRL_IP_LEARNING 可以用来指定要使用的 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，如果变量 TRL_IP_LEARNING 没有设置，这是默认设置。这个方法只会检测每个接口的一个 IP 地址。检测到客户机虚拟机的 IP 地址后，其 IP 网络流量将锁定到那个地址（例如，IP 地址欺骗会被其过滤器之一阻止。在这种情况下，虚拟机的用户将无法更改客户机虚拟机内部接口的 IP 地址，这被视为 IP 地址欺骗。当客户机虚拟机迁移到另一台主机物理机器或暂停操作后恢复时，客户机虚拟机发送的第一个数据包将再次确定客户机虚拟机可在特定接口上使用的 IP 地址。

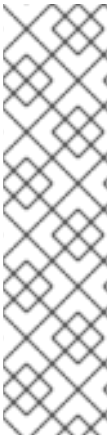
dhcp 指示 libvirt 仅允许具有有效租期的 DHCP 服务器分配地址。此方法支持每个接口检测和使用多个 IP 地址。当客户机虚拟机在暂停操作后恢复时，任何有效的 IP 地址租用都会应用到其过滤器。否则，客户机虚拟机预期使用 DHCP 来获取新的 IP 地址。当客户机虚拟机迁移到另一台物理主机物理虚拟机时，需要客户机虚拟机重新运行 DHCP 协议。

如果 CTRL_IP_LEARNING 设为 none，libvirt 不会进行 IP 地址学习并引用 IP 地址，而不为其分配显式值就出现错误。

18.12.5.2. DHCP Snooping

CTRL_IP_LEARNING=dhcp (DHCP snooping)提供额外的反欺骗安全性，特别是在组合使用过滤器时，仅允许可信 DHCP 服务器来分配 IP 地址。要启用此功能，将变量 DHCPSERVER 设置为有效的 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 snooping 被启用且 DHCP 租期过期时，客户机虚拟机将不再能够使用 IP 地址，直到从 DHCP 服务器获取新的有效租期。如果迁移了客户机虚拟机，它必须获取新的有效 DHCP 租期才能使用 IP 地址（例如，通过使虚拟机接口停止并重新启动）。



注意

自动 DHCP 检测侦听客户端虚拟机交换的 DHCP 流量，与基础架构的 DHCP 服务器交换。为了避免对 libvirt 的拒绝服务攻击，对这些数据包的评估具有速率限制，这意味着客户机虚拟机每秒发送过多的 DHCP 数据包不会评估所有这些数据包，因此过滤器可能无法适应。假设假定一般 DHCP 客户端行为每秒发送少量的 DHCP 数据包。此外，务必要在基础架构中的所有虚拟客户机上设置适当的过滤器，以避免它们能够发送 DHCP 数据包。因此，guest 虚拟机必须被阻止，将 UDP 和 TCP 流量从端口 67 发送到端口 68，或所有 guest 虚拟机上应使用 DHCPSERVER 变量来限制仅来自于可信 DHCP 服务器的信息。同时，必须在 子网中的所有客户机虚拟机上启用反欺骗措施。

例 18.6. 激活 DHCP 侦听的 IP

以下 XML 提供了一种使用 DHCP 侦听方法激活 IP 地址学习的示例：

```
<interface type='bridge'>
```

```
  <source bridge='virbr0'/>
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp'/>
  </filterref>
</interface>
```

18.12.6. 保留变量

表 18.2 “保留变量” 显示 libvirt 被视为保留和使用的变量：

表 18.2. 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	当前没有实施：接口使用的 IPV6 地址列表
DHCPSERVER	可信 DHCP 服务器的 IP 地址列表
DHCPSERVERV6	当前未实施：可信 DHCP 服务器的 IPv6 地址列表
CTRL_IP_LEARNING	选择 IP 地址检测模式

18.12.7. 元素和属性概述

所有网络过滤器所需的根元素都命名为 **<filter>**，其中有两个可能的属性。**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，从而更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**、**ipv 6**、**nrp** 和 **rarp**。

18.12.8. 其他过滤器的引用

任何过滤器都可以保留对其他过滤器的引用。在过滤器树中多次引用各个过滤器，但过滤器之间的引用不能引入循环。

例 18.7. 一个干净的流量过滤器示例

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 filterref。此节点必须具有其值包含要引用的过滤器名称的属性过滤器。

可以随时定义新的网络过滤器，可以包含对 libvirt 未知的网络过滤器的引用。但是，一旦启动虚拟机或引用过滤器的网络接口将会被热插，则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，否则网络接口无法附加。

18.12.9. 过滤规则

以下 XML 显示了一个网络流量过滤器的简单示例，当传出 IP 数据包中的 IP 地址（通过变量 IP）提供时丢弃流量的规则并非预期，从而防止 IP 地址被虚拟机欺骗。

例 18.8. 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。此节点可包含以下属性中最多三个属性：

- 操作是必需的：
  - drop（匹配规则会静默丢弃数据包而不进一步分析）
  - reject（匹配规则会生成 ICMP 拒绝消息，无进一步分析）

- 接受（与规则匹配，无进一步分析）
- 返回（与规则通过此过滤器匹配，但返回到调用过滤器以进行进一步分析）
- 继续（匹配该规则进入下一规则以便进一步分析）

- 方向是必需的可具有以下值：
  - 对于传入的流量
  - 传出流量
  - 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实例化的顺序。具有较低值的规则将在具有更高值的规则之前实例化。有效值为 -1000 到 1000 范围。如果没有提供此属性，则默认分配优先级为 500。请注意，根链中过滤规则按照优先级后与 root 链连接的过滤器进行排序。这允许通过 访问过滤链来交集过滤规则。如需更多信息，请参阅 第 18.12.3 节 “过滤链优先级”。

- statematch 是可选的。可能的值有 '0' 或 'false' 来关闭与连接匹配的底层连接状态。默认设置为"true"或 1

更多信息请参阅 第 18.12.11 节 “高级过滤器配置主题”。

上面的 例 18.7 “一个干净的流量过滤器示例” 示例表示 类型为 ip 的流量将 与链 ipv4 关联，规则将具有 priority=500。例如，如果引用另一个过滤器，类型为 ip 的流量也与链 ipv4 关联，则过滤的规则将相对于显示规则的 priority=500 进行排序。

规则可以包含用于过滤流量的单一规则。上例显示过滤了 ip 类型的流量。

18.12.10. 支持的协议

以下小节列出了网络过滤子系统支持的协议的一些详情。规则节点中以嵌套节点提供这种类型的流量规则。根据流量类型规则过滤，属性不同。上例显示了在 ip 流量过滤节点中有效的单一属性 srcipaddr。以下小节显示了有效属性以及它们期望的数据类型。可用的 datatypes 如下：

- UINT8 : 8 位整数；范围 0-255
- UINT16: 16 位整数；范围 0-65535
- MAC_ADDR: MAC 地址采用点十进制格式，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: 采用点十进制格式的 IP 地址，如 10.1.2.3
- ip_MASK：以点十进制格式(255.255.248.0)或 CIDR 掩码(0-32)的 IP 地址掩码。
- IPV6_ADDR: IPv6 地址采用数字格式，如 FFFF::1
- IPV6_MASK：数字格式的 IPv6 掩码(FFFF:FF:FC00::)或 CIDR mask(0-128)
- STRING: 字符串
- BOOLEAN: 'true', 'yes', '1' 或 'false', 'no', '0'
- IPSETFLAGS：由最多 6 种 'src' 或 'dst' 元素描述的 ipset 的源和目的地标记，从数据包标头的源或目标部分选择功能；example: src， src,dst。此处提供的"selectors"的数量取决于引用的 ipset 类型

除类型为 IP_MASK 或 IPV6_MASK 的属性外的每个属性都可使用 match 属性而不是值 来 划分。多个多组属性可以分组在一起。以下 XML 片段显示了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>
[...]
```

规则的行为评估规则，并在给定的协议属性的边界内查看规则。因此，如果单个属性的值与规则中的值不匹配，则整个规则将在评估过程中跳过。因此，只有在上面的例子中，如果协议属性 attribute1 与 value1 不匹配，且协议属性 attribute2 不匹配 value2，且协议属性 attribute3 与 value3 匹配，则传入的流量才会被丢弃。

18.12.10.1. mac(Ethernet)

协议 ID：mac

此类型的规则应当转至 root 链。

表 18.3. 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	掩码应用到发送方的 MAC 地址
dstmacaddr	MAC_ADDR	目标的 MAC 地址
dstmacmask	MAC_MASK	掩码应用到目的地的 MAC 地址
protocolid	UINT16(0x600-0xffff), STRING	第 3 层协议 ID。有效字符串包括 [arp, rarp, ipv4, ipv6]
注释	字符串	文本字符串最多 256 个字符

过滤器可以编写如下：

```
[...]
<mac match='no' srcmacaddr='$MAC'/>
[...]
```

18.12.10.2. VLAN (802.1Q)

*协议 **ID：vlan***

此类型的规则应当转至根或 vlan 链。

表 18.4. 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	掩码应用到发送方的 MAC 地址
dstmacaddr	MAC_ADDR	目标的 MAC 地址
dstmacmask	MAC_MASK	掩码应用到目的地的 MAC 地址
vlan-id	UINT16 (0x0-0xfff, 0 - 4095)	VLAN ID
encap-protocol	UINT16(0x03c-0xfff)、String	封装的第 3 层协议 ID，有效的字符串为rp, ipv4 ipv6
注释	字符串	文本字符串最多 256 个字符

18.12.10.3. STP(Spanning Tree Protocol)

*协议 **ID： stp***

此类型的规则应当转至根或 stp 链。

表 18.5. 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	掩码应用到发送方的 MAC 地址
type	UINT8	网桥协议数据单元(BPDU)类型
标记	UINT8	BPDU 标记的macmask
root-priority	UINT16	root 优先级范围开始

属性名称	datatype	定义
root-priority-hi	UINT16 (0x0-0xfff, 0 - 4095)	root 优先级范围结束
root-address	MAC _ADDRESS	Root MAC 地址
root-address-mask	MAC _MASK	Root MAC 地址掩码
roor-cost	UINT32	根路径成本（大量开始）
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
port	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	消息期限计时器（范围启动）
msg-age-hi	UINT16	消息年龄计时器范围结束
max-age-hi	UINT16	最长期限时间范围结束
hello-time	UINT16	hello 时间计时器（范围启动）
hello-time-hi	UINT16	hello 计时器范围结束
forward-delay	UINT16	转发延迟（范围启动）
forward-delay-hi	UINT16	转发延迟范围结束
注释	字符串	文本字符串最多 256 个字符

18.12.10.4. ARP/RARP

协议 ID: arp 或 rarp

此类型的规则应当进入 root 或 arp 链。

表 18.6. 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	掩码应用到发送方的 MAC 地址
dstmacaddr	MAC_ADDR	目标的 MAC 地址
dstmacmask	MAC_MASK	掩码应用到目的地的 MAC 地址
hwtype	UINT16	硬件类型
protocoltype	UINT16	协议类型
opcode	UINT16, 字符串	opcode 有效字符串有：Request、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
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	布尔值指示是否检查 gettuitous ARP 数据包
注释	字符串	文本字符串最多 256 个字符

18.12.10.5. IPv4

协议 ID：ip

此类型的规则应当转至 root 或 ipv4 链。

表 18.7. 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	掩码应用到发送方的 MAC 地址
dstmacaddr	MAC_ADDR	目标的 MAC 地址
dstmacmask	MAC_MASK	掩码应用到目的地的 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	掩码应用到目的地 IP 地址
protocol	UINT8, 字符串	第 4 层协议标识符。协议的有效字符串有： tcp、udp、udp、udplite、esp、icmp、igmp、sctp
srcportstart	UINT16	开始有效源端口 ; 需要协议
srcportend	UINT16	有效源端口的结束 ; 需要协议
dstportstart	UNIT16	有效目标端口的范围 ; 需要协议
dstportend	UNIT16	有效目标端口的结束 ; 需要协议
注释	字符串	文本字符串最多 256 个字符

18.12.10.6. IPv6

协议 ID： ipv6

此类型的规则应当转至 root 或 ipv6 链。

表 18.8. IPv6 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址

属性名称	datatype	定义
srcmacmask	MAC_MASK	掩码应用到发送方的 MAC 地址
dstmacaddr	MAC_ADDR	目标的 MAC 地址
dstmacmask	MAC_MASK	掩码应用到目的地的 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	掩码应用到目的地 IP 地址
protocol	UINT8, 字符串	第 4 层协议标识符。协议的有效字符串有： tcp、udp、udp、udplite、esp、sh、icmpv6、sctp
scrportstart	UNIT16	开始有效源端口 ; 需要协议
srcportend	UINT16	有效源端口的结束 ; 需要协议
dstportstart	UNIT16	有效目标端口的范围 ; 需要协议
dstportend	UNIT16	有效目标端口的结束；需要协议
注释	字符串	文本字符串最多 256 个字符

18.12.10.7. TCP/UDP/SCTP

协议 ID：tcp、udp、sctp

此类型的流量会忽略 chain 参数，并应省略或设置为 root。

表 18.9. 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

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	掩码应用到目的地 IP 地址
scripto	IP_ADDR	源 IP 地址范围开始
srcipfrom	IP_ADDR	源 IP 地址结束
dstipfrom	IP_ADDR	目标 IP 地址范围的开头
dstipto	IP_ADDR	目标 IP 地址范围结束
scrportstart	UNIT16	开始有效源端口；需要协议
srcportend	UINT16	有效源端口的结束；需要协议
dstportstart	UNIT16	有效目标端口的范围；需要协议
dstportend	UNIT16	有效目标端口的结束；需要协议
注释	字符串	文本字符串最多 256 个字符
state	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 分隔的列表
标记	字符串	仅 TCP-only：使用掩码和标志格式的掩码/标签格式是 SYN、ACK、URG、PSH、FIN、RST 或 NONE 或 ALL 的逗号分隔列表
ipset	字符串	libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

18.12.10.8. ICMP

协议 ***ID：icmp***

注意：对于这种类型的流量，链参数将被忽略，并应省略或设置为 root。

表 18.10. 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	掩码应用到目的地的 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	掩码应用到目的地 IP 地址
srcipfrom	IP_ADDR	源 IP 地址范围开始
scripto	IP_ADDR	源 IP 地址结束
dstipfrom	IP_ADDR	目标 IP 地址范围的开头
dstipto	IP_ADDR	目标 IP 地址范围结束
type	UNIT16	ICMP 类型
code	UNIT16	ICMP 代码
注释	字符串	文本字符串最多 256 个字符
state	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 分隔的列表
ipset	字符串	libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

18.12.10.9. IGMP、ESP、AH、UDPLITE、"ALL"

协议 ID：igmp、esp、ah、udplite、所有

此类型的流量会忽略 chain 参数，并应省略或设置为 root。

表 18.11. 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	掩码应用到目的地的 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	掩码应用到目的地 IP 地址
srcipfrom	IP_ADDR	源 IP 地址范围开始
scripto	IP_ADDR	源 IP 地址结束
dstipfrom	IP_ADDR	目标 IP 地址范围的开头
dstipto	IP_ADDR	目标 IP 地址范围结束
注释	字符串	文本字符串最多 256 个字符
state	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 分隔的列表
ipset	字符串	libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

18.12.10.10. IPV6 上的 TCP/UDP/SCTP

协议 ID：tcp-ipv6、udp-ipv6、sctp-ipv6

此类型的流量会忽略 chain 参数，并应省略或设置为 root。

表 18.12. TCP, UDP, SCTP 通过 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
dstipmask	IP_MASK	掩码应用到目的地 IP 地址
srcipfrom	IP_ADDR	源 IP 地址范围开始
scripto	IP_ADDR	源 IP 地址结束
dstipfrom	IP_ADDR	目标 IP 地址范围的开头
dstipto	IP_ADDR	目标 IP 地址范围结束
srcportstart	UINT16	有效源端口的范围
srcportend	UINT16	有效源端口的范围结束
dstportstart	UINT16	有效目标端口的范围
dstportend	UINT16	有效目标端口的范围结束
注释	字符串	文本字符串最多 256 个字符
state	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 分隔的列表
ipset	字符串	libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

18.12.10.11. ICMPv6

协议 ID：icmpv6

此类型的流量会忽略 chain 参数，并应省略或设置为 root。

表 18.13. 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
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	掩码应用到目的地 IP 地址
srcipfrom	IP_ADDR	源 IP 地址范围开始
scripto	IP_ADDR	源 IP 地址结束
dstipfrom	IP_ADDR	目标 IP 地址范围的开头
dstipto	IP_ADDR	目标 IP 地址范围结束
type	UINT16	ICMPv6 类型
code	UINT16	ICMPv6 代码
注释	字符串	文本字符串最多 256 个字符
state	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 分隔的列表
ipset	字符串	libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

18.12.10.12. IGMP、ESP、AH、UDPLITE、'ALL' over IPv6

协议 ID： igmp-ipv6, esp-ipv6, ah-ipv6, udplite-ipv6, all-ipv6

此类型的流量会忽略 chain 参数，并应省略或设置为 root。

表 18.14. IGMP、ESP、AH、UDPLITE、'ALL' over IPv 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	掩码应用到目的地 IP 地址
srcipfrom	IP_ADDR	源 IP 地址范围开始
scripto	IP_ADDR	源 IP 地址结束
dstipfrom	IP_ADDR	目标 IP 地址范围的开头
dstipto	IP_ADDR	目标 IP 地址范围结束
注释	字符串	文本字符串最多 256 个字符
state	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 分隔的列表
ipset	字符串	libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

18.12.11. 高级过滤器配置主题

以下部分讨论高级过滤配置主题。

18.12.11.1. 连接跟踪

网络过滤子系统（Linux 上）利用连接跟踪支持 IP 表。这有助于强制网络流量的双向（匹配）以及计算和限制客户机同时连接的数量。例如，如果客户机虚拟机具有 TCP 端口 8080 作为服务器打开，客户端可以在端口 8080 上连接到 guest 虚拟机。然后，连接跟踪和执行可能会阻止客户机虚拟机启动从（TCP 客户端）端口 8080 到主机物理机器的连接。更为重要的是，跟踪有助于防止远程攻击者建立到客户机虚拟机的连接。例如，如果客户机虚拟机中的用户已建立到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 发起到客户机虚拟机的连接。默认情况下，连接状态匹配，可启用连接跟踪，然后打开流量的双向连接。

-

例 18.9. 关闭到 TCP 端口连接的 XML 示例

以下显示了一个 XML 片段示例，为到 TCP 端口 12345 的传入连接关闭这个功能。

```
[...]
 <rule direction='in' action='accept' statematch='false'>
   <cp dstportstart='12345'/>
 </rule>
[...]
```

这现在允许传入流量到 TCP 端口 12345，但也可启用从虚拟机中的启动（客户端）TCP 端口 12345，它们可能或可能并不需要。

18.12.11.2. 限制连接数

要限制客户机虚拟机可以建立的连接数量，必须提供一个规则来设定指定类型的流量的连接限制。例如，如果应当允许某一虚拟机一次仅 ping 个其他 IP 地址，并且应一次只有一个活动传入的 ssh 连接。

例 18.10. 将限制用于连接的 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

在接受流量的规则前，必须在 XML 中列出的限制规则。根据 例 18.10 “将限制用于连接的 XML 示例文件” 中的 XML 文件，允许发送到端口 22 的 DNS 流量进入客户机虚拟机的额外规则，以避免因为 ssh 守护进程无法建立与 DNS 查找失败相关的 ssh 会话。离开此规则可能会导致 ssh 客户端在尝试连接时意外挂起。应格外小心，以处理与跟踪流量相关的超时。用户可能在虚拟机内终止的 ICMP ping 在主机物理机的连接跟踪系统中可能具有较长的超时，因此不允许其他 ICMP ping 到达。

最佳解决方案是使用以下命令在主机物理机器的 sysfs 中调整超时：# echo 3 > /proc/sys/netfilter/nf_conntrack_icmp_timeout。此命令将 ICMP 连接跟踪超时设置为 3 秒。这样做的影响是一旦一个 ping 终止，另一个 ping 将在 3 秒后启动。

如果出于某种原因客户机虚拟机没有正确关闭其 TCP 连接，则需要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，任何闲置连接可能会导致连接跟踪系统中超时，在交换数据包后可以重新激活。

但是，如果设置限制过低，则新发起的连接可能会强制进入 TCP backoff 的闲置连接。因此，应该设置连接限制而不是高，以便在新的 TCP 连接中造成与空闲连接相关的异常流量行为。

18.12.11.3. 命令行工具

virsh 已延长了网络过滤器的生命周期支持。与网络过滤子系统相关的所有命令都以前缀 nwfilter 开始。以下命令可用：

- nwfilter-list ：列出所有网络过滤器的 UUID 和名称
- nwfilter-define：定义新的网络过滤器或更新现有网络过滤器（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）

18.12.11.4. 预先存在的网络过滤器

以下是使用 libvirt 自动安装的网络过滤器示例列表：

表 18.15. ICMPv6 协议类型

命令名称	描述
no-arp-spoofing	防止客户机虚拟机欺骗 ARP 流量；此过滤器仅允许 ARP 请求和回复消息，并强制这些数据包包含客户机虚拟机的 MAC 和 IP 地址。
allow-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）
allow-dhcp-server	允许 guest 虚拟机从指定的 DHCP 服务器请求 IP 地址。DHCP 服务器的点十进制 IP 地址必须在引用此过滤器时提供。变量名称必须是 *DHCPSERVER*。
no-ip-spoofing	防止客户机虚拟机使用不同于数据包中的源 IP 地址发送 IP 数据包。
no-ip-multicast	防止客户机虚拟机发送 IP 多播数据包。
clean-traffic	防止 MAC、IP 和 ARP 欺骗。此过滤器引用其他几个过滤器作为构建块。

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。以上列表中最常用的是 clean-traffic 过滤器。例如，此过滤器本身可以与 no-ip-multicast 过滤器结合使用，以防止虚拟机在阻止数据包欺骗之上发送 IP 多播流量。

18.12.11.5. 编写您自己的过滤器

由于 libvirt 仅提供了几个网络过滤器，因此您可能会考虑自行编写。当计划这样做时，您可能需要了解网络过滤子系统及其在内部工作方式。当然，您还必须了解和理解您需要过滤的协议，使其不再比您想要通过的内容做进一步的通信，事实上要想让流量通过。

网络过滤子系统目前仅适用于 Linux 主机物理计算机，仅适用于 Qemu 和 KVM 类型的虚拟机。在 Linux 上，它基于对 ebtables、iptables 和 ip6tables 的支持，并利用了其功能。考虑在 第 18.12.10 节 “支持的协议” 中找到的列表，可以使用 ebtables 实施以下协议：

- mac

- STP（跨度树协议）
- vlan (802.1Q)
- ARP, rarp
- ipv4
- ipv6

任何通过 IPv4 运行的协议均支持使用 iptables，通过 IPv6 使用 ip6 实施它们。

使用 Linux 主机物理机器，由 libvirt 的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，且仅在通过 iptables 或 ip6tables 过滤器之后进行。如果过滤器树有带有协议的规则，如 mac、stp、vlan arp、ipv4 或 ipv6；会首先自动使用列出的 ebtable 规则和值。

可以创建多个同一协议链。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，可以指定一个名称 arp-test 的链，例如：

例如，可以使用 IP 协议过滤器通过源和目标端口过滤 UDP 流量，并为要接受的 UDP 数据包指定协议、源和目标 IP 地址和端口的属性。这允许使用 ebtables 早期过滤 UDP 流量。但是，一旦 IP 或 IPv6 数据包（如 UDP 数据包）传递了 ebtables 层，并且一个过滤器树中至少有一个规则实例化 iptables 或 ip6tables 规则，那么还需要为这些过滤层提供 UDP 数据包通过的规则。这可以通过包含适当的 udp 或 udp-ipv6 流量过滤节点的规则来实现。

> 例 18.11. 创建自定义过滤器
>
> 假设需要一个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，但不会让虚拟机在接口上 ping

- 允许虚拟机进行 DNS 查找（UDP 给端口 53）

防止在现有 clean-traffic 网络过滤器实现欺骗的要求，因此从自定义过滤器引用它的方法。

要启用 TCP 端口 22 和 80 的流量，添加了两条规则来启用此类流量。允许 guest 虚拟机发送 ping 流量以进行 ICMP 流量。为了简单起见，一般的 ICMP 流量可以从虚拟客户机启动，不会指定 ICMP 回显请求和响应消息。其它流量都无法被客户端虚拟机到达或启动。为此，将添加丢弃所有其他流量的规则。假设 guest 虚拟机名为 test，并且关联我们的过滤器的接口称为 eth0，则创建名为 test-eth0 的过滤器。

这些注意事项的结果是以下网络过滤器 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>
```

18.12.11.6. 自定义过滤器示例

虽然上述 XML 中的其中一个规则包含 guest 虚拟机的 IP 地址作为源或目标地址，但流量的过滤可以正常工作。原因在于，规则的评估会基于每个接口在内部进行，而规则将根据已发送的(tap)接口发送或接收数据包，而不是根据哪个源或目标 IP 地址进行评估。

例 18.12. 网络接口描述的 XML 示例

测试客户机虚拟机的域 XML 中可能的网络接口描述的 XML 片段如下所示：

```
[...]
 <interface type='bridge'>
   <source bridge='mybridge'/>
   <filterref filter='test-eth0'/>
 </interface>
[...]
```

要更严格地控制 ICMP 流量并强制只能从客户机虚拟机发送 ICMP 回显请求，并且仅由 guest 虚拟机接收 ICMP 回显响应，可使用以下两个规则替换上述 ICMP 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 18.13. 第二个自定义过滤器示例

本例演示如何构建类似的过滤器，但使用位于 guest 虚拟机上的 ftp 服务器扩展要求列表。此过滤器的要求有：

- 防止客户机虚拟机接口来自 MAC、IP 和 ARP 欺骗
- 在客户机虚拟机接口中只打开 TCP 端口 22 和 80

- *允许 guest 虚拟机从接口发送 ping 流量，但不允许在接口上 ping guest 虚拟机*
- *允许客户机虚拟机执行 DNS 查找（UDP 给端口 53）*
- *启用 ftp 服务器（在活动模式下），以便它可以在 guest 虚拟机内运行*

要求允许在客户机虚拟机内运行 FTP 服务器到需要，允许端口 21 能够访问 FTP 控制流量，并让客户机虚拟机能够建立源自虚拟客户机的 TCP 端口 20 到 FTP 客户端的传出 TCP 连接（FTP 活动模式）。本例中提供了几种方式编写此过滤器，以及两个可能的解决方案。

第一个解决方案利用 TCP 协议的 state 属性，在 Linux 主机物理机的连接跟踪框架中提供 hook。对于客户机虚拟机发起的 FTP 数据连接（FTP 活跃模式）的 RELATED 状态，用于检测 guest 虚拟机发起的 FTP 数据连接后果（或与 的 的关系）现有的 FTP 控制连接，从而使其允许通过防火墙传递数据包。然而，RELATED 状态仅适用于 FTP 数据路径传出 TCP 连接的第一个数据包。之后，其状态为 ESTABLISHED，然后适用于传入和传出方向。所有这些都与源自虚拟客户机的 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>
```

在使用 RELATED 状态尝试过滤器前，您必须确定适当的连接跟踪模块已加载到主机物理机器的内核。根据内核的版本，您必须在建立与客户端虚拟机的 FTP 连接前运行以下两个命令之一：

- **#modprobe nf_conntrack_ftp - where available OR**
- **#modprobe ip_conntrack_ftp （如果以上不可用）**

如果 FTP 以外的协议与 RELATED 状态结合使用，则必须加载其相应的模块。模块可用于协议：ftp、tftp、irc、sip、sctp 和 amanda。

第二个解决方案使用之前解决方案的连接的状态标志。此解决方案利用了事实：当检测到流量流的第一个数据包时，连接的 NEW 状态是有效的。因此，如果接受流的第一个数据包，则流将变为连接，因此进入 ESTABLISHED 状态。因此，可以编写常规规则，以允许 ESTABLISHED 连接到达 guest 虚拟机或由 guest 虚拟机发送。这是为由 NEW 状态标识的最第一个数据包编写特定规则，并指示数据可被接受的端口。所有数据包都用于未明确接受的端口，从而不会到达 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>
```

18.12.12. 限制

下表列出了网络过滤子系统当前已知的限制：

- 只有在目标主机物理机器上也提供了由客户机虚拟机顶层过滤器引用的整个过滤器树时，才支持虚拟机迁移。网络过滤器 **clean-traffic** 应该在所有 **libvirt** 安装中可用，因此可以迁移引用此过滤器的客户机虚拟机。为了保证版本兼容性不是一个问题，请确保通过定期更新软件包来确保正在使用 **libvirt** 的最新版本。

- 在 **libvirt** 安装版本 **0.8.1** 或更高版本之间必须进行迁移，才能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）无法过滤，其规则用于协议 ID arp、rarp、ipv4 和 ipv6。它们只能通过协议 ID、MAC 和 VLAN 进行过滤。因此，过滤器 clean-traffic 例 18.1 “网络过滤示例” 示例将无法按预期工作。

18.13. 创建 TUNNELS

本节将演示如何实施不同的隧道场景。

18.13.1. 创建多播 Tunnels

多播组设置为表示虚拟网络。任何网络设备在同一多播组中的虚拟机都可以相互通信，即使是在主机物理计算机之间。此模式也可供非特权用户使用。没有默认的 DNS 或 DHCP 支持，且没有传出网络访问。为了提供传出网络访问，其中一个 guest 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。多播协议与客户机虚拟机用户模式兼容。请注意，您提供的源地址必须是来自用于多播地址块的地址。

要创建多播隧道，在 <devices> 元素中指定以下 XML 详情：

图 18.28. 多播隧道 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>
...
```

18.13.2. 创建 TCP Tunnels

TCP 客户端/服务器架构提供虚拟网络。在此配置中，一个客户机虚拟机提供网络的服务器终止，而所有其他客户机虚拟机都配置为客户端。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。此模式也可用于非特权用户。请注意，这个模式不提供默认的 DNS 或 DHCP 支持，也不提供传出网络访问。为了提供传出网络访问，其中一个 guest 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 18.29. 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>
  ...
```

18.14. 设置 VLAN TAGS

使用 virsh net-edit 命令添加虚拟局域网(vLAN) 标签。此标签也可以用于带有 SR-IOV 设备的 PCI 设备分配。有关详情请参阅 第 9.1.7 节 “使用 SR-IOV 设备配置 PCI 分配(Passthrough)”。

图 18.30. vSetting VLAN 标签（仅在支持的网络类型中）

```
<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 if）网络类型支持对客户机透明的 vlan 标记，则可选的 <vlan> 元素可以指定一个或多个 vlan 标签，以应用到使用这个网络的所有客户端的流量。（openvswitch 和 type='hostdev 的 SR-IOV 网络支持透明的 VLAN 标记；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），将客户机流量标记到特定的 vlans。） 如预期所示，tag 属性指定要使用的 vlan 标签。如果网络定义了多个 <vlan> 元素，则假设用户希望使

用所有指定的标签进行 VLAN 中继。如果需要使用单个标签的 VLAN 中继，可选属性 trunk='yes' 可以添加到 VLAN 元素中。

对于使用 openvswitch 的网络连接，可以配置 'native-tagged' 和 'native-untagged' VLAN 模式。这使用 <tag> 元素上的可选的 nativeMode 属性： nativeMode 可以设置为 'tagged' 或 'untagged'。元素的 id 属性设置原生 vlan。

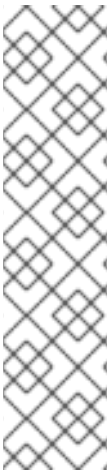
VLAN 元素也可以在 <portgroup> 元素中指定，也可以直接在域的 <interface> 元素中指定。<>如果在多个位置中指定 vlan 标签，则 <接口> 中的设置具有优先权，后面跟上由接口配置选择的 <portgroup> 中的设置。只有在 <端口组或> <接口> 中提供没有时，<网络中的> <vlan> 才会被选择。

18.15. 将 QOS 应用到您的虚拟网络

质量服务(QoS) 是指资源控制系统，可保证网络上所有用户最佳体验，确保没有延迟、判断或数据包丢失。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 20.16.9.14 节 “服务质量”。

第 19 章 QEMU-KVM 命令、FLAGS 和 ARGUMENTS

19.1. 简介



注意

本章的主要目标是提供 qemu-kvm 实用程序命令、标志和参数的列表，用作仿真程序和 Red Hat Enterprise Linux 6 中的虚拟机监控程序。这是已知运行但需要自行使用的选项的完整概述。Red Hat Enterprise Linux 6 将 KVM 用作底层的虚拟化技术。使用的机器模拟器和虚拟机监控程序是经过修改的 QEMU 版本，称为 qemu-kvm。此版本不支持原始 QEMU 的所有配置选项，还会添加一些附加选项。

不应在 此处列出的选项。

白名单格式

- <name > - 在语法描述中使用时，此字符串应替换为用户定义的值。
- [a|b|c] - 在语法描述中使用时，只使用 | 分隔的字符串之一。
- 如果没有注释，则支持所有可能值的选项。

19.2. 基本选项

这部分提供有关基本选项的信息。

模拟机器

-m & lt;machine-type>

-machine <machine-type>[,<property>[=<value>][,..]]

处理器类型

-cpu <model>[,<FEATURE>][...]

通过运行 -cpu ? 命令来查看其他模型。

- *Opteron_G5 - AMD Opteron 63xx 类 CPU*
- *Opteron_G4 - AMD Opteron 62xx 类 CPU*
- *Opteron_G3 - AMD Opteron 23xx(AMD Opteron Gen 3)*
- *Opteron_G2 - AMD Opteron 22xx(AMD Opteron Gen 2)*
- *Opteron_G1 - AMD Opteron 240(AMD Opteron Gen 1)*
- *Westmere - Westmere E56xx/L56xx/X56xx(Nehalem-C)*
- *Haswell - Intel Core Processor(Haswell)*
- *SandyBridge - Intel Xeon E312xx(Sandy Bridge)*
- *Nehalem - Intel Core i7 9xx(Nehalem Class Core i7)*
- *Penryn - Intel Core 2 Duo P9xxx(Penryn Class Core 2)*
- *Conroe - Intel Celeron_4x0(Conroe/Merom Class Core 2)*
- *cpu64-rhel5 - Red Hat Enterprise Linux 5 支持 QEMU Virtual CPU 版本*
- *cpu64-rhel6 - Red Hat Enterprise Linux 6 支持 QEMU Virtual CPU 版本*

- *默认值 - 特殊选项使用上述中的默认选项。*

处理器拓扑

-smp <n>[,cores=<ncores>][,threads=<nthreads>][,sockets=<nsocks>][,maxcpus=<maxcpus>]

在处理器拓扑上应用虚拟机监控程序和客户机操作系统限制。

NUMA System

-numa <nodes>[,mem=<size>][,cpus=<cpu[-cpu>]][,nodeid=<node>]

在处理器拓扑上应用虚拟机监控程序和客户机操作系统限制。

内存大小

-m <megs>

支持的值受客户机最少和最大值和虚拟机监控程序限制。

键盘布局

-k <language>

虚拟客户机名称

-name <name>

客户机 UUID

-UUID & lt;uuid>

19.3. 磁盘选项

这部分提供有关磁盘选项的信息。

通用驱动器

-drive <option>[,<option>[,<option>[,...]]]

以下选项支持：

- *ReadOnly[on|off]*
- *werror[enospc|report|stop|ignore]*
- *rerror[report|stop|ignore]*
- *id=<id>*

  对于 if=none，驱动器的 ID 有以下限制：

  - *IDE 磁盘必须采用 <id> 格式：驱动器-ide0-<BUS>-<UNIT>*

    正确格式示例：

    -drive if=none,id=drive-ide0-<BUS>-<UNIT>,.. -device ide-drive,drive=drive-ide0-<BUS>-<UNIT>,bus=ide.<BUS>,unit=<UNIT>
- *文件=<file>*

  <file> 的值使用以下规则解析：

  - *不支持将软盘设备作为 <file> 传递。*
  - *仅将 cd-rom 设备作为 <file> 传递为 cdrom 介质类型(media=cdrom)，并且仅作为 IDE 驱动器（如果 =ide 或 if=none + -device ide-drive）。*

- *如果 <file> 不是 block 或 字符设备，它不得包含 ':'。*

- *如果=<interface>*

  支持以下接口：none、ide、virtio、软盘。

- *index=<index>*

- *media=<media>*

- *缓存=<cache>*

  支持的值有：none、writeback 或 writethrough。

- *copy-on-read=[on|off]*

- *snapshot=[yes|no]*

- *serial=<serial>*

- *AIO =< aio>*

- *格式=<format>*

  此选项不是必需的，可以忽略。但是，不建议在原始镜像中使用此功能，因为它代表了安全风险。支持的格式有：

  - *qcow2*

- raw

引导选项

-boot [order=<drives>][,menu=[on|off]]

快照模式

-snapshot

19.4. 显示选项

这部分提供有关显示选项的信息。

禁用图形

-nographic

VGA 卡 Emulation

-vga <type>

支持的类型：

- C irrus - Cirrus Logic GD5446 视频卡.
- std - 带有 Bochs VBE 扩展的标准 VGA 卡。
- QXL - Spice 泛虚拟化虚拟卡.
- none - 禁用 VGA 卡。

VNC 显示

-vnc <display>[,<option>[,<option>[,...]]]

支持的显示值：

- *[<host>]:<port>*
- *UNIX:<path>*
- *share[allow-exclusive|force-shared|ignore]*
- *无 - 未指定其他选项时支持。*

支持的选项有：

- *to=<port>*
- *reverse*
- *password*
- *tls*
- *x509=</path/to/certificate/dir> - 指定 tls 时支持。*
- *x509verify=</path/to/certificate/dir> - 指定 tls 时支持。*
- *sasl*
- *acl*

SPICE Desktop

-spice option[,option[,...]]

支持的选项有：

- *端口=<number>*
- *addr=<addr>*
- *ipv4*

  ipv6
- *密码=<secret>*
- *disable-ticketing*
- *disable-copy-paste*
- *tls-port=<number>*
- *x509-dir=</path/to/certificate/dir>*
- *x509-key-file=<file>*

  x509-key-password=<file>

  x509-cert-file=<file>

x509-cacert-file=<file>

x509-dh-key-file=<file>

- *tls-cipher=<list>*
- *tls-channel[main|display|cursor|inputs|record|playback]*

  plaintext-channel[main|display|cursor|inputs|record|playback]
- *image-compression=<compress>*
- *jpeg-wan-compression=<value>*

  zlib-glz-wan-compression=<value>
- *streaming-video=[off|all|filter]*
- *agent-mouse=[on|off]*
- *playback-compression=[on|off]*
- *seamless-migratio=[on|off]*

19.5. 网络选项

这部分提供有关网络选项的信息。

TAP 网络

-netdev tap,id=<id>[,<options>...]

支持以下选项（全部使用 name=value 格式）：

- *ifname*
- *fd*
- *script*
- *downscript*
- *sndbuf*
- *vnet_hdr*
- *vhost*
- *vhostfd*
- *vhostforce*

19.6. 设备选项

这部分提供有关设备选项的信息。

常规设备

-device <driver>[,<prop>[=<value>][,...]]

所有驱动程序都支持以下属性

- *id*
- *总线*

支持以下驱动程序（具有可用属性）：

- *pci-assign*
  - *主机*
  - *bootindex*
  - *configfd*
  - *addr*
  - *rombar*
  - *romfile*
  - *多功能*

  如果设备具有多个功能，则需要把所有这些功能分配给同一客户端。

- *rtl8139*

- *mac*
- *netdev*
- *bootindex*
- *addr*

- *e1000*
  - *mac*
  - *netdev*
  - *bootindex*
  - *addr*

- *virtio-net-pci*
  - *ioeventfd*
  - *向量*
  - *indirect*
  - *event_idx*

- ***csum***
- ***guest_csum***
- ***gso***
- ***guest_tso4***
- ***guest_tso6***
- ***guest_ecn***
- ***guest_ufo***
- ***host_tso4***
- ***host_tso6***
- ***host_ecn***
- ***host_ufo***
- ***mrg_rxbuf***
- ***status***
- ***ctrl_vq***

- ***ctrl_rx***
- ***ctrl_vlan***
- ***ctrl_rx_extra***
- ***mac***
- ***netdev***
- ***bootindex***
- ***x-txtimer***
- ***x-txburst***
- ***tx***
- ***addr***

- ***qxl***
  - ***ram_size***
  - ***vram_size***
  - ***修订***

- *cmdlog*
- *addr*

- *ide-drive*
  - *unit*
  - *驱动器*
  - *physical_block_size*
  - *bootindex*
  - *ver*
  - *wwn*

- *virtio-blk-pci*
  - *class*
  - *驱动器*
  - *logical_block_size*
  - *physical_block_size*

- ***min_io_size***
- ***opt_io_size***
- ***bootindex***
- ***ioeventfd***
- ***向量***
- ***indirect_desc***
- ***event_idx***
- ***scsi***
- ***addr***

- ***virtio-scsi-pci - 6.3 中的技术预览，自 6.4 开始支持。***

  对于 Windows 客户机，Windows Server 2003 一直是技术预览，自 6.5 起不再受支持。但是，自 6.5 开始，Windows Server 2008 和 2012, Windows 桌面 7 和 8 被完全支持。

  - ***向量***
  - ***indirect_desc***
  - ***event_idx***

- *num_queues*
- *addr*

- *isa-debugcon*
- *ISA-serial*
  - *index*
  - *iobase*
  - *irq*
  - *chardev*
- *virtserialport*
  - *nr*
  - *chardev*
  - *name*
- *virtconsole*
  - *nr*

- *chardev*
- *name*

- *virtio-serial-pci*
  - *向量*
  - *class*
  - *indirect_desc*
  - *event_idx*
  - *max_ports*
  - *flow_control*
  - *addr*

- *ES1370*
  - *addr*

- *AC97*
  - *addr*

- *intel-hda*
  - *addr*
- *hda-duplex*
  - *cad*
- *hda-micro*
  - *cad*
- *hda-output*
  - *cad*
- *i6300esb*
  - *addr*
- *ib700 - 无属性*
- *sga - 无属性*
- *virtio-balloon-pci*
  - *indirect_desc*

    - *event_idx*
    - *addr*
- *USB-tablet*
    - *migrate*
    - *port*
- *usb-kbd*
    - *migrate*
    - *port*
- *USB-mouse*
    - *migrate*
    - *port*
- *USB-ccid - 自 6.2 起受支持.*
    - *port*
    - *插槽*

- *USB-host - 自 6.2 起技术预览.*
  - *hostbus*
  - *hostaddr*
  - *hostport*
  - *vendorid*
  - *productid*
  - *isobufs*
  - *port*
- *USB-hub - 自 6.2 起支持.*
  - *port*
- *USB-ehci - 自 6.2 起技术预览.*
  - *freq*
  - *maxframes*
  - *port*

- ***USB-storage - 自 6.2 起技术预览.***
  - *驱动器*
  - ***bootindex***
  - ***serial***
  - ***removable***
  - ***port***
- ***USB-redir - 6.3 技术预览，自 6.4 开始支持***
  - ***chardev***
  - ***filter***
- ***SCSI-cd - 6.3 技术预览，自 6.4 开始支持***
  - *驱动器*
  - ***logical_block_size***
  - ***physical_block_size***
  - ***min_io_size***

- ***opt_io_size***
- ***bootindex***
- ***ver***
- ***serial***
- ***scsi-id***
- ***LUN***
- ***channel-scsi***
- ***wwn***

- ***SCSI-hd - 自 6.4 起支持 6.3 技术预览***
  - *驱动器*
  - ***logical_block_size***
  - ***physical_block_size***
  - ***min_io_size***
  - ***opt_io_size***

- ***bootindex***
- ***ver***
- ***serial***
- ***scsi-id***
- ***LUN***
- ***channel-scsi***
- ***wwn***

- ***SCSI-block - 自 6.4 起支持 6.3 技术预览***
  - *驱动器*
  - ***bootindex***

- ***SCSI-disk - 对 6.3 技术预览***
  - ***drive=drive***
  - ***logical_block_size***
  - ***physical_block_size***

- ***min_io_size***
- ***opt_io_size***
- ***bootindex***
- ***ver***
- ***serial***
- ***scsi-id***
- ***LUN***
- ***channel-scsi***
- ***wwn***

- ***piix3-usb-uhci***
- ***piix4-usb-uhci***
- ***ccid-card-passthru***

全局设备设置

-global <device>.<property>=<value>

这些附加设备的 "General device" 部分支持的设备和属性：

- *isa-fdc*
  - *driveA*
  - *driveB*
  - *bootindexA*
  - *bootindexB*
- *qxl-vga*
  - *ram_size*
  - *vram_size*
  - *修订*
  - *cmdlog*
  - *addr*

字符设备

-chardev 后端,id=<id>[,<options>]

支持的后端有：

- *null,id=<id> - null device*

- *socket,id=<id>,port=<port>[,host=<host>][,to=<to>][,ipv4][,ipv6][,nodelay][,server][,nowait][,telnet] - tcp socket*

- *socket,id=<id>,path=<path>[,server][,nowait][,telnet] - unix socket*

- *file,id=<id>,path=<path> - trafit to file。*

- *stdio,id=<id> - 标准 i/o*

- *spicevmc,id=<id>,name=<name> - spice 频道*

启用 USB

-usb

19.7. LINUX/多引导

这部分提供有关 Linux 和多引导引导的信息。

内核文件

-kernel <bzImage>

注意： 不支持多引导镜像

RAM 磁盘

-initrd <file>

命令行参数

-append <cmdline>

19.8. 专家选项

这部分提供有关专家选项的信息。

KVM 虚拟化

-enable-kvm

QEMU-KVM 只支持 KVM 虚拟化，并在可用的情况下默认使用它。如果使用 -enable-kvm，且 KVM 不可用，qemu-kvm 将失败。但是，如果没有使用 -enable-kvm，且 KVM 不可用，qemu-kvm 在 TCG 模式下运行，这不被支持。

禁用内核模式 PIT 重新注入

-no-kvm-pit-reinjection

没有关闭

-no-shutdown

没有重启

-no-reboot

serial Port, Monitor, QMP

-serial <dev>

-monitor <dev>

-qmp <dev>

支持的设备有：

- *stdio - 标准输入/输出*
- *null - null 设备*

- *file:<filename> - 输出到 file.*
- *TCP :[<host>]:<port>[,server][,nowait][,nodelay] - TCP Net console.*
- *UNIX:<path>[,server][,nowait] - Unix 域套接字。*
- *MON:<dev_string> - 以上所有设备，也用于多x monitor。*
- *none - disable，仅对 -serial 有效。*
- *chardev:<id> - 使用 -chardev 创建的字符设备。*

监控重定向

-mon <chardev_id>[,mode=[readline|control]][,default=[on|off]]

手动 CPU 启动

-S

RTC

-rtc [base=utc|localtime|date][,clock=host|vm][,driftfix=none|slew]

Watchdog

-watchdog 模型

watchdog Reaction

-watchdog-action <action>

客户机内存备份

-mem-prealloc -mem-path /dev/hugepages

SMBIOS Entry

-smbios type=0[,vendor=<str>][,<version=str>][,date=<str>][,release=%d.%d]

-SMBIOS type=1[, manufacturer=<str>][,product=<str>][,version=<str>][,serial=<str>][,uuid=<uuid>][,sku=<str>][,family=<str>][,family=<str>][,version=<str>][,serial=<str>][,uuid=<uuid>][,sku=<str>][,family=<str>][

19.9. 帮助和信息选项

这部分提供有关帮助和信息选项的信息。

Help

-h

-help

版本

-version

音频帮助

-audio-help

19.10. 其它选项

这部分提供有关其它选项的信息。

Migration（迁移）

-incoming

没有默认配置

-nodefconfig

-nodefaults

不支持在没有 -nodefaults 的情况下运行

设备配置文件

-readconfig <file>

-writeconfig <file>

Loaded Saved State

-loadvm <file>

第 20 章 操作域 XML

这部分论述了用于代表域的 XML 格式。此处术语 域 指所有 guest 虚拟机所需的根域元素。<>域 XML 有两个属性： type 指定用于运行域的管理程序。允许的值是特定驱动程序，但包括 KVM 和其他值。ID 是正在运行的客户机虚拟机的唯一整数标识符。不活跃的机器没有 id 值。本章中的部分将解决域 XML 的组件。在处理域 XML 时，本手册中的其他章节可能会参考本章。

注意

本章基于 libvirt 上游文档。

20.1. 常规信息和元数据

这些信息位于域 XML 中：

图 20.1. 域 XML 元数据

```
<domain type='xen' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <title>A short description - title - of the domain</title>
  <description>Some 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 中本节的组件如下：

表 20.1. 常规元数据元素

元素	描述
<name>	为虚拟机指定名称。此名称应仅包含字母数字字符，且需要在单一主机物理计算机范围内唯一。它通常用于创建用于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。格式必须是 RFC 4122-compliant, eg **3e3fce45-4f53-4fa7-bb32-11f34168b82b**。如果在定义/生成新机器时省略，则生成一个随机 UUID。还可以使用 sysinfo 规范来提供 UUID。

元素	描述
<title>	**标题** 会为域的简短描述而创建空间。标题不应包含任何换行符。
<description>	与标题不同，libvirt 不使用这些数据，它可以包含用户想要显示的任何信息。
<metadata>	应用程序可以使用以 XML 节点/树的形式存储自定义元数据。应用必须在其 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层元素（如果应用需要结构，它们应当有子元素到其命名空间元素）

20.2. 操作系统启动

有许多不同的方法可以分别使用各自的方法引导虚拟机。它们分别在后续子部分进行说明，并包含：BIOS 引导装载程序、主机物理机器引导装载程序以及直接内核引导。

20.2.1. BIOS 引导装载程序

对于支持完全虚拟化的虚拟机监控程序，可以通过 BIOS 启动。在这种情况下，BIOS 具有引导顺序优先级（软盘、硬磁盘、cdrom、网络）确定在哪里获取/查找引导镜像。域 XML 的 OS 部分包含如下信息：

图 20.2. BIOS 引导装载程序域 XML

```
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <boot dev='hd'/>
  <boot dev='cdrom'/>
  <bootmenu enable='yes'/>
  <smbios mode='sysinfo'/>
  <bios useserial='yes' rebootTimeout='0'/>
</os>
...
```

域 XML 中本节的组件如下：

表 20.2. BIOS 引导装载程序元素

元素	描述
<type>	指定要在客户端虚拟机上引导的操作系统类型。**HVM** 表示操作系统是设计在裸机上运行，因此需要完全虚拟化。**Linux** 是指支持 Xen 3 管理程序 guest ABI 的操作系统。另外还有两个可选属性，**arch** 指定到虚拟化的 CPU 架构，以及引用 **机器类型的** 机器。如需更多信息，*请参阅驱动程序功能*。
<loader>	指的是用于协助域创建过程的一个固件。仅在使用 Xen 完全虚拟化域时才需要。
<boot>	取一个值：**fd**、**hd**、**cdrom** 或 **网络**，用于指定要考虑的下一个引导设备。boot 元素可以重复多次重复，以设置引导设备的优先级列表，以便依次尝试。同一类型的多个设备按照其目标进行排序，同时保留总线的顺序。定义域后，libvirt 返回它的 XML 配置（通过 virDomainGetXMLDesc）以排序顺序列出设备。且排序，第一个设备将标记为可引导。如需更多信息，请参阅 *BIOS 引导装载程序*。
<引导菜单>	确定是否在 guest 虚拟机启动时启用交互式引导菜单提示。**enable** 属性可以是 **yes** 或 **no**。如果没有指定，将使用管理程序默认
<smbios>	决定如何在客户机虚拟机中显示 SMBIOS 信息。必须指定 **mode** 属性，作为模拟（允许虚拟机监控程序生成所有值）、**host**（所有 Block 0 和块 1，但 UUID 除外）来自于主机物理机器的 SMBIOS 值；virConnectGetSysinfo 调用可用于查看复制的值）或 **sysinfo**（使用 sysinfo 元素中的值）。**如果没有指定，则使用系统管理程序默认设置。**
<BIOS>	此元素具有属性 **useserial**，可能的值是 **yes** 或 **no**。属性启用或禁用 Serial Graphics Adapter，允许用户在串行端口上查看 BIOS 信息。因此，一个需要定义串行端口。请注意，有另一个属性 **rebootTimeout**，它控制在启动失败时 guest 虚拟机应重新启动的时间（根据 BIOS）应重新启动的时间。该值以毫秒为单位，最大为 **65535**，特殊值 **-1** 可禁用重启。

20.2.2. 主机物理 Machine Boot Loader

使用半虚拟化的虚拟机监控程序通常不会模拟 BIOS，而是主机物理机器负责操作系统启动。这可能会在主机物理机器中使用伪引导加载器提供接口来为客户机虚拟机选择内核。示例是带有 Xen 的 pygrub。

图 20.3. 主机物理机器引导装载程序域 XML

```
...
 <bootloader>/usr/bin/pygrub</bootloader>
 <bootloader_args>--append single</bootloader_args>
 ...
```

域 XML 中本节的组件如下：

表 20.3. BIOS 引导装载程序元素

元素	描述
<bootloader>	在主机物理机器操作系统中提供引导装载程序的完全限定路径。这个引导装载程序会选择引导哪个内核。引导装载程序所需的输出依赖于使用的虚拟机监控程序。
<bootloader_args>	允许将命令行参数传递给引导装载程序（可选命令）

20.2.3. 直接内核引导

安装新的 guest 虚拟机操作系统时，直接从主机物理机器操作系统中存储的 initrd 启动通常很有用，允许命令行参数直接传递给安装程序。对于半虚拟化和完全虚拟化的客户机虚拟机，这个能力通常都可用。

图 20.4. 直接内核引导

```
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <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 中本节的组件如下：

表 20.4. 直接内核引导元素

元素	描述
<type>	与 BIOS 引导部分所述
<loader>	与 BIOS 引导部分所述
<内核>	指定主机物理机器操作系统中的内核镜像的完全限定路径
<initrd>	指定主机物理机器操作系统中（可选） ramdisk 镜像的完全限定域名。
<cmdline>	指定在引导时传递给内核（或安装程序）的参数。这通常用来指定替代的主控制台（如串行端口）或安装介质源 / kickstart 文件

20.3. SMBIOS 系统信息

通过一些虚拟机监控程序，可以控制向客户机虚拟机显示哪些系统信息（例如，SMBIOS 字段可由虚拟机监控程序进行填充，并使用 guest 虚拟机中的 midecode 命令来检查）。可选的 sysinfo 元素涵盖此类信息类别。

图 20.5. 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，它决定了子元素的布局，并定义如下：

- *SMBIOS - Sub-elements 调用特定 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 的每个子元素都使用 SMBIOS 块，在这些元素中，可以是描述块内字段的条目元素列表。可识别以下块和条目：*

- BIOS - 这是 SMBIOS 的 block 0，条目名从 供应商、版本、date 和 release 中提取。

- <系统> - 这是 SMBIOS 的块 1，条目名从 制造商、产品、版本、serial、uuid、sku 和 family 中提取。如果 uuid 条目与顶级 uuid 元素一同提供，则两个值必须匹配。

20.4. CPU 分配

图 20.6. CPU 分配

```
<domain>
  ...
  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
  ...
</domain>
```

<cpu> 元素定义为 guest 虚拟机操作系统分配的虚拟 CPU(vCPU)的最大数量，这些 CPU 必须在 1 之间，且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，它是以逗号分隔的物理 CPU 编号列表，默认情况下可将域进程和虚拟 CPU 固定到此元素。

请注意，可以使用 cputune 属性来单独指定域进程和虚拟 CPU 的固定策略。如果在 <cputune> 中指定 模拟器 属性，则 <vcpu> 指定的 cpuset 值将被忽略。

同样，为 vcpupin 设定了值的虚拟 CPU 会导致 cpuset 设置被忽略。未指定 vcpupin 的虚拟 CPU 将固定到 cpuset 指定的物理 CPU。C puset 列表中的每一元素是单个 CPU 编号、CPU 编号的范围，或者用 caret(^)，后面接一个 CPU 编号，在上一个范围内排除在内。可以使用 当前 属性来指定是否应启用虚拟 CPU 的最大数量。

可以使用可选属性 放置 来指定域进程的 CPU 放置模式。放置 可以设置为 static 或 auto。如果设置 <vcpu placement='auto'>，系统将查询 numad 并使用 <numatune> 标签中指定的设置，并忽略 <vcpu> 中的任何其他设置。如果设置 <vcpu placement='static'>，系统将使用 <vcpu 放置> 标签中指定的设置，而不是 <numatune> 中的设置。

20.5. CPU TUNING

图 20.7. CPU 调整

```
<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 中的这个部分的组件如下：

表 20.5. CPU 调整元素

元素	描述
<cputune>	提供有关域的 CPU 可调项参数的详细信息。这是可选的。
<vcpupin>	指定域 VCPU 将要固定到的主机物理 CPU。如果省略此项，并且未指定 **<vcpu>** 的属性 **cpuset**，则 vCPU 会默认固定到所有物理 CPU。它包含两个必要属性，属性 **vcpu** 指定 **id**，而 attribute **cpuset** 与 element **<vcpu>** 的属性相同。
<模拟器兼容性>	指定主机物理机器 CPU、"emulator"（包括 vcpu 的域的子集）将固定到：如果省略，则不指定 element **<vcpu>** 的属性 **cpuset**，则"emulator"默认固定到所有物理 CPU。它包含一个必需的属性 **cpuset**，指定要固定到哪个物理 CPU。如果元素 **<vcpu>** 的属性 **放置** 是自动的，则不允许使用 **仿真程序**。
<共享>	为域指定按比例加权共享。如果省略此项，则默认为操作系统固有的默认值。如果没有值的单元，它会相对于其他 guest 虚拟机的设置进行计算。例如，如果 guest 虚拟机配置了 2048 值，它将达到两倍的处理时间，客户机虚拟机配置了值为 1024 的客户机虚拟机。

元素	描述
<周期>	以微秒为单位指定强制间隔。通过使用 **period**，允许每个域的 vcpu 消耗超过其分配的配额的运行时间。这个值应该在以下范围内： **1000-1000000**。一个 **period** > 值为 **0** 表示没有值。
<quota>	指定以微秒为单位允许的最大带宽。**配额** 为任何负值的域表示域具有无限带宽，这意味着它不控制带宽。该值应该在以下范围内：**1000 - 18446744073709551** 或小于 **0。**值为 **0 的配额** 意味着无值。您可以使用此功能确保所有 vcpus 都以相同的速度运行。
<emulator_period>	以微秒为单位指定强制间隔。在 **<paper_period>** 中，域的仿真程序线程（不含 vcpus 除外）将不允许消耗在运行时超过 **<emulator_quota>**。**<模拟器_period>** 值应位于以下范围： **1000 - 1000000**。值为 **0** 的 **<仿真程序_period>** 表示无值。
<emulator_quota>	指定域仿真程序线程（不包括 vcpus）的最大允许带宽（以微秒为单位）。**<模拟器_quota>** 作为负值的域表示域具有仿真程序线程的无限带宽（不包括 vcpus），这意味着它不受带宽控制。该值应位于以下范围： **1000 - 18446744073709551**，或小于 **0。**值为 **0** 的 **<仿真程序_quota>** 表示无值。

20.6. 内存备份

内存后备允许管理程序在客户机虚拟机中正确管理大型页面。

可选的 <memoryBacking> 元素可能会设置 <hugepages> 元素。这告知虚拟机监控程序，客户机虚拟机应使用大页而不是普通原生页面大小来分配内存。

*图 **20.8. 内存后备***

```
<domain>
  ...
  <memoryBacking>
    <hugepages/>
  </memoryBacking>
  ...
</domain>
```

20.7. 内存调整

图 20.9. 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>
```

虽然所有都是可选的，但域 XML 中的这个部分的组件如下：

表 20.6. 内存调优元素

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，则默认为提供的 OS。因此，在设置限制时，参数会作为整体应用，因此需要添加 guest 虚拟机 RAM、guest 虚拟机视频 RAM 并允许一些内存开销。最后一个部分很难判断，因此使用试用和错误。对于每个可调项，可以使用与 **<内存>** 相同的值来指定输入中哪个单位。为了向后兼容，输出总是以 KiB 为单位。
<hard_limit>	这是客户机虚拟机可以使用的最大内存。这个值 **的单位** 以 **kibibytes** （1024 字节的块）表示。
<soft_limit>	这是在内存争用过程中强制执行的内存限值。这个值 **的单位** 以 kibibytes（1024 字节的块）表示。
<swap_hard_limit>	这是客户端虚拟机可以使用的最大内存加上交换。这个值 **的单位** 以 kibibytes（1024 字节的块）表示。这必须大于所提供的 **<hard_limit>** 值
<min_guarantee>	这是保证客户机虚拟机的最小内存分配。这个值的单位以 kibibytes（1024 字节的块）表示。

20.8. NUMA 节点调整

使用传统管理工具完成 NUMA 节点调整后，生效了以下域 XML 参数：

图 20.10. 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 中的这个部分的组件如下：

表 20.7. NUMA 节点调整元素

元素	描述
<numatune>	通过控制域进程的 NUMA 策略，提供有关如何调整 NUMA 主机物理机器的性能的详细信息。
<内存>	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。属性 **模式是 交集、strict** 或 **首选的**。如果没有给定值，则默认为 **strict**。属性 **nodeset** 指定 NUMA 节点，其语法与 element **<vcpu>** 的属性 **cpuset** 相同。属性 **放置** 可用于指示域进程的内存放置模式。其值可以 **是静态的**，也可以是 **自动**。如果指定了属性 **<nodeset>**，则默认为 **<vcpu>** 或 **static <的放置>**。**auto** 表示域进程将仅从 query numad 返回的公告 nodeset 分配内存，如果指定，则忽略属性 nodeset 的值。如果 **vcpu** 的属性 **放置** 是 **auto**，且没有指定属性 **<numatune>**，则使用 **<放置> auto** 和 mode **strict** 的默认 numatune 将被隐式添加。

20.9. 块 I/O 调整

图 20.11. 块 I/O 调优

```
<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 中的这个部分的组件如下：

表 20.8. 块 I/O 调整元素

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，则默认为提供的 OS。
<weight>	此可选的 weight 元素是客户机虚拟机的整体 I/O 权重。该值应该在范围 100 - 1000 之间。
<device>	域可能有多个 **<设备>** 元素，它们进一步调整域正在使用的每个主机物理块设备的权重。请注意，多个客户端虚拟机磁盘可以共享一个主机物理机器块设备。另外，当它们由同一主机物理机器文件系统中的文件支持时，这个调优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（将它们绑定到单个 **<磁盘>**）。**<>**每个设备元素有两个强制子元素，即描述设备的绝对路径的路径，**<权重>** 为该设备的相对权重，它的相对权重为 100 - 1000。**<>**

20.10. 资源分区

虚拟机监控程序(hypervisor)可能允许在资源分区中将虚拟机放在资源分区中，并有可能嵌套这些分区。<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，其内容定义了放置域的资源分区的路径。如果没有列出分区，则域将被置于默认分区中。app/admin 的职责是确保启动客户机虚拟机之前存在分区。默认只假定（特定于管理程序）默认分区存在。

图 20.12. 资源分区

```
<resource>
  <partition>/virtualmachines/production</partition>
</resource>
```

QEMU 和 LXC 驱动程序目前支持资源分区，该驱动程序可将分区路径映射到所有挂载的控制器中的 cgroups 目录。

20.11. CPU 型号和拓扑

本节涵盖了 CPU 模型的要求。请注意，每个虚拟机监控程序都有自己的策略，因此 guest 将默认查看其 CPU 功能。QEMU/KVM 呈现的 CPU 功能集合取决于客户机虚拟机配置中的 CPU 模型。qemu32 和 qemu64 是基本的 CPU 型号，但还有其他模型（具有额外的功能）。每个模型及其拓扑都使用域 XML 中的以下元素来指定：

图 20.13. CPU 模型和拓扑示例 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 20.14. 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 20.15. CPU 型号和拓扑示例 3

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

如果没有限制，则无法将限制放在 CPU 模型或其功能上，则可以使用如下 CPU 元素：

图 20.16. CPU 型号和拓扑示例 4

```
<cpu>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

域 XML 中本节的组件如下：

表 20.9. CPU 型号和拓扑元素

元素	描述
<cpu>	此元素包含 vCPU 功能集的所有参数。
<匹配>	指定 **<cpu>** 元素中显示的功能必须与可用的 vCPU 匹配。如果 **<拓扑>** 是 **<cpu>** 元素中嵌套的唯一元素，则可以省略 **match** 属性。**match** 属性的可能值有： ● **最小值** - 列出的功能是最低要求。然后，vCPU 中可能会提供更多功能，但这是接受的最小值。如果没有满足最低要求，则此值将失败。 ● **exact** - 为客户机虚拟机提供的虚拟 CPU 必须完全与指定的功能完全匹配。如果未找到匹配项，则会导致错误。 ● **严格** - 除非主机物理机器 CPU 与规格完全匹配，否则不会创建 guest 虚拟机。 如果 **<cpu>** 元素中省略了 **match** 属性，则使用默认设置 **match='exact'**。

元素	描述
<模式>	此可选属性可用于使客户端虚拟机 CPU 更容易配置为尽可能与主机物理机器 CPU 关闭。mode 属性的可能值有： • **Custom** - 描述如何将 CPU 出现在客户端虚拟机中。如果没有指定 **mode** 属性，这是默认设置。这个模式会使持久的客户机虚拟机能够看到同一硬件，无论在其上启动 guest 虚拟机的主机是什么。 • **host-model** - 这基本上是将主机物理机器 CPU 定义从功能 XML 复制到域 XML 的快捷方式。因为在启动域前复制 CPU 定义，可以在不同的主机物理机器上使用相同的 XML，同时仍可提供最佳的客户机虚拟机 CPU 每个主机物理机器支持。此模式中无法使用 **match** 属性或任何功能元素。如需更多信息，请参阅 libvirt 域 XML CPU 型号 • 通过 **此模式**，guest 虚拟机可见的 CPU 与主机物理机器 CPU（包括 libvirt 中导致错误的元素）完全相同。这种模式明显的缺点是，客户端虚拟机环境无法在不同的硬件上重现，因此建议不要再现此模式。这个模式中不允许使用 **模型** 或功能元素。 • **请注意，在 host-model 和 host-passthrough 模式中，当调用 virDomainGetXMLDesc API 时，通过指定 VIR_DOMAIN_XML_UPDATE_CPU 标志时，可以确定在当前主机物理机器上使用的 CPU 定义。当运行客户机虚拟机时，在显示不同的硬件时容易对操作系统进行响应，并在具有不同功能的主机物理机器间迁移这些虚拟机，您可以使用此输出将 XML 重写到自定义模式，以实现更强大的迁移。**
<model>	指定客户机虚拟机请求的 CPU 模型。可用 CPU 模型列表及其定义可在 libvirt 数据目录中安装的 **cpu_map.xml** 文件中找到。如果虚拟机监控程序无法使用确切的 CPU 模型，libvirt 会在维护 CPU 功能列表的同时自动回退到虚拟机监控程序支持的最接近模型。可选的 **fallback** 属性可用于避免此行为，在这种情况下，尝试启动请求不支持的 CPU 模型的域将失败。支持 fallback 属性的值有： **allow**（这是默认值）和 **forbid**。可选的 **vendor_id** 属性可用于设置客户机虚拟机看到的供应商 ID。它的长度必须为 12 个字符。如果没有设置，则使用主机物理机器的厂商 id。典型的可能值包括 **AuthenticAMD** 和 **GenuineIntel**。

元素	描述
<vendor>	指定客户机虚拟机请求的 CPU 供应商。如果缺少此元素，客户机虚拟机会在与给定功能匹配的 CPU 上运行，无论其供应商是什么。受支持的供应商列表可在 **cpu_map.xml** 中找到。
<topology>	指定提供给客户端虚拟机的虚拟 CPU 请求的拓扑。必须为套接字、内核和线程指定三个非零值：CPU 插槽总数、每个插槽的内核数和每个内核的线程数。
<功能>	可以包含零个或更多元素，用于对所选 CPU 模型提供的功能进行微调。已知功能名称列表可在与 CPU 型号相同的文件中找到。每个特性元素的含义取决于其策略属性，它必须设置为以下值之一： • **force** - 强制虚拟机 CPU 支持，无论主机物理 CPU 是否真正支持它。 • **require** - 指定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 支持这个功能，但仅在主机物理机器 CPU 支持时才支持该功能。 • **disable** - 虚拟 CPU 不支持它。 • **forbid** - 如果主机物理机器 CPU 支持该功能，客户机虚拟机创建将失败。

20.11.1. 客户机虚拟机 NUMA 拓扑

可以使用 <numa> 元素和域 XML 中的以下内容来指定虚拟机 NUMA 拓扑：

图 20.17. 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>
...
```

每个单元单元指定了一个 NUMA 单元或 NUMA 节点。CPU 指定作为节点一部分的 CPU 或 CPU 范围。memory 指定节点内存（以 kibibytes 为单位）（1024 字节的块）。从 0 开始，为每个单元或节点被分配 cellid 或 nodeid。

20.12. 事件配置

使用域 XML 的以下部分可以覆盖各种事件中采取的默认操作。

图 20.18. 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客户端虚拟机操作系统触发生命周期操作时指定操作。常见的用例是在进行初始操作系统安装时强制将重启视为 **poweroff**。这允许为第一次安装后重新配置虚拟机。

域 XML 中本节的组件如下：

表 20.10. 事件配置元素

状态	描述
<on_poweroff>	指定在 guest 虚拟机请求 poweroff 时要执行的操作。可能有四个可能的参数： • **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，并使用相同的配置重启它 • **保留** - 此操作将完全终止域，但保留其资源以允许未来分析。 • **rename-restart** - 此操作将完全终止域，然后使用新名称重启它
<on_reboot>	指定在客户端虚拟机请求重启时要执行的操作。可能有四个可能的参数： • **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，并使用相同的配置重启它 • **保留** - 此操作将完全终止域，但保留其资源以允许未来分析。 • **rename-restart** - 此操作将完全终止域，然后使用新名称重启它

状态	描述
<on_crash>	指定客户端虚拟机崩溃时要执行的操作。另外，它还支持这些附加操作： • **coredump-destroy** - 崩溃的域内核被转储，域会被完全终止，并释放所有资源。 • **coredump-restart** - 崩溃的域内核被转储，且域使用相同的配置设置重启 可能有四个可能的参数： • **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，并使用相同的配置重启它 • **保留** - 此操作将完全终止域，但保留其资源以允许未来分析。 • **rename-restart** - 此操作将完全终止域，然后使用新名称重启它
<on_lockfailure>	指定锁定管理器丢失资源锁定时应采取什么操作。下列操作由 libvirt 识别，尽管并不是单个锁定管理器需要支持所有这些操作。如果没有指定操作，则每个锁定管理器都会执行其默认操作。以下参数可以： • **poweroff** - 强制关闭域 • **restart** - 重启域以重新静止其锁定。 • **pause** - 暂停域，以便在解决锁定问题时手动恢复。 • **ignore** - 如果没有发生，使域保持运行。

20.13. 电源管理

可使用传统管理工具对客户机虚拟机操作系统进行强制启用或禁用 BIOS 公告，这将影响域 XML 的以下部分：

图 20.19. 电源管理

```
...
<pm>
  <suspend-to-disk enabled='no'/>
  <suspend-to-mem enabled='yes'/>
</pm>
...
```

<pm> 元素可以使用 arguement yes 或 disabled 参数启用。可使用参数 suspend-to-disk 和 S4（使用参数 suspend-to-mem ACPI 休眠状态）为 S3 实施 BIOS 支持。如果未指定任何内容，管理程序将保留其默认值。

20.14. 管理程序功能

虚拟机监控程序可以允许某些 CPU / 机器功能启用(state='on') 或禁用(state='off')。

图 20.20. 管理程序特性

```
...
<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
  </hyperv>
</features>
...
```

如果未指定 <状态>，则所有功能均列在 <features> 元素中。可以通过调用 功能 XML 来查找可用的功能，但为完全虚拟化域有一个通用的设置：

表 20.11. 管理程序特性

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。
<acpi>	对于电源管理非常有用，例如，在进行 KVM 客户机虚拟机时，安全关闭才能工作。
<apic>	允许使用可编程 IRQ 管理。对于此元素，有一个可选的属性 **eoi**，**它的值** 设置为 guest 虚拟机的 EOI（中断中断）的可用性。
<hap>	如果硬件中有可用的硬件使用，则可使用硬件。
hyperv	支持各种功能来改进运行 Microsoft Windows 的客户机虚拟机的行为。使用可选的属性 **relaxed 的值** 来启用或禁用计时器的 relax 约束

状态	描述

20.15. TIMEKEEPING

客户机虚拟机时钟通常从主机物理机器时钟中初始化。大多数操作系统预期硬件时钟保持在 UTC 中，这是默认设置。请注意，对于 Windows 客户机虚拟机，必须在 localtime 中设置 guest 虚拟机。

图 20.21. timekeeping

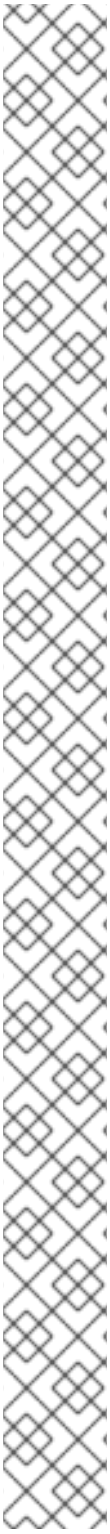
```
...
<clock offset='localtime'>
  <timer name='rtc' tickpolicy='catchup' track='guest'>
    <catchup threshold='123' slew='120' limit='10000'/>
  </timer>
  <timer name='pit' tickpolicy='delay'/>
</clock>
...
```

域 XML 中本节的组件如下：

表 20.12. 时间保持元素

状态	描述

状态	描述
<clock>	**偏移** 属性取 4 个可能的值，允许对 guest 虚拟机时钟与主机物理计算机的同步方式进行精细的控制。请注意，管理程序并不需要在所有时间源中支持所有策略 • **UTC** - 在引导时将时钟同步到 UTC。**UTC** 模式可以转换为 **变量** 模式，可以通过使用 adjustment 属性来控制。如果 **重置了** 值，则不会执行转换。数字值强制转换为 **变量** 模式，并将值用作初始调整。默认调整是特定于系统管理程序的。 • **localtime** - 在启动时将客户机虚拟机时钟与主机物理机器配置的时区进行同步。调整属性的行为与在"utc"模式中相同。 • **timezone** - 使用 timezone 属性将客户机虚拟机时钟同步到请求的时区。 • **变量** - 根据基础属性，在客户机虚拟机时钟中使用相对于 UTC 或 localtime 的任意偏移值。使用 **adjustment** 属性来指定相对于 UTC（或本地时间）的增量（或本地时间）。客户机虚拟机可以自由调整 RTC，期望它在下次重启时将生效。这与 **utc** 和 **localtime** 模式（使用可选属性 **adjustment='reset'**）不同，其中 RTC 调整会在每次重启时丢失。此外，**基础** 属性可以是 **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
<频率>	这是一个未签名的整数，用于指定运行 **name="tsc"** 的频率。
<模式>	**mode** 属性控制如何管理 **name="tsc" <timer>**，并可以设置为：**auto**、**原生**、**模拟**、**paravirt** 或 **smpsafe**。其他计时器始终模拟。
<存在>	指定特定计时器是否可用于 guest 虚拟机。可以设置为 **yes** 或 **no**



注意

每个 <计时器> 元素必须包含 name 属性，并且可能具有以下属性，具体取决于指定的名称：

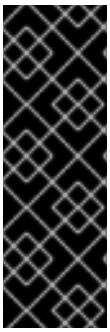
- <name> - 选择修改哪个 计时器。以下值可以接受：kvmclock (QEMU-KVM)、pit(QEMU-KVM)或 rtc(QEMU-KVM)或 tsc（仅限libxl）。请注意，当前不支持 的平台。
- trace - 指定计时器跟踪。可接受以下值： 引导、guest 或 wall。trace 仅对名称="rtc" 有效。
- tickpolicy - 决定错过向客户机虚拟机注入循环的截止时间会发生什么。可分配以下值：
  - delay -will 继续以正常速率提供数量。客户端虚拟机时间会因为空后出现延迟
  - catchup - 提供点高，以便与缺失的 tick 相乘以更高的速度。捕获完成后不会显示 guest 虚拟机时间。另外，还可有三个可选属性，每个正整数，如下所示： threshold、slew 和 limit。
  - 合并 - 将丢失的勾号合并到一个循环中，并注入它们。根据合并如何，客户机虚拟机时间可能会延迟。
  - discard - 丢弃丢失的勾号，并在默认的间隔设置中继续注入。客户机虚拟机时间可能会延迟，除非有明确声明处理丢失的 ticks

20.16. DEVICES

这一组 XML 元素都用于描述为虚拟客户机域提供的设备。以下所有设备都显示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-9p-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 总线虚拟随机数字生成器设备



重要

如果创建了 virtio 设备，其中向量数量被设置为大于 32 的值，则设备的行为就像在 Red Hat Enterprise Linux 6 中被设置为零个值，但没有在 Enterprise Linux 7 中。如果平台上的任何 virtio 设备中向量数量设置为 33 或更高，则生成的向量设置不匹配会导致迁移错误。因此，不会试图将 向量 值设置为大于 32。除 virtio-balloon-pci 和 virtio-rng-pci 外，所有 virtio 设备都接受 向量 参数。

图 20.22. devices - 子元素

```
...
<devices>
  <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
</devices>
...
```

<模拟器> 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 指定为每个特定域类型或架构组合使用的建议默认模拟器。

20.16.1. 硬盘驱动器, Floppy Disks, CDROMs

域 XML 的这一部分指定了类似于磁盘的任何设备，它是软盘、硬盘、cdrom 或半虚拟化驱动程序通过 disk 元素指定。

图 20.23. devices - 硬盘、软盘、CDROM

```
...
 <devices>
   <disk type='file' snapshot='external'>
     <driver name="tap" type="aio" cache="default"/>
     <source file='/var/lib/xen/images/fv0' startupPolicy='optional'>
       <seclabel relabel='no'/>
     </source>
     <target dev='hda' bus='ide'/>
     <iotune>
       <total_bytes_sec>10000000</total_bytes_sec>
       <read_iops_sec>400000</read_iops_sec>
       <write_iops_sec>100000</write_iops_sec>
     </iotune>
     <boot order='2'/>
     <encryption type='...'>
       ...
     </encryption>
     <shareable/>
     <serial>
       ...
     </serial>
   </disk>
     ...
   <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>
   <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>
   <disk type='block' device='cdrom'>
     <driver name='qemu' type='raw'/>
     <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>
   <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>
  </devices>
  ...
```

20.16.1.1. 磁盘元素

<disk> 元素是用于描述磁盘的主要容器。属性 类型 可以与 <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 磁盘元素

20.16.1.2. Source 元素

如果磁盘 <type='file' 的>，则 file 属性指定保存磁盘的文件的完全限定域名。如果 <disk type='block'>，则 dev 属性指定要用作磁盘的主机物理设备的路径。使用 文件和 块 （一个或多个可选子元素 seclabel ）可用于只为源文件覆盖域安全标签策略。如果磁盘类型是 dir，则 dir 属性指定要用作磁盘的目录的完全限定路径。如果磁盘类型是 网络，则 protocol 属性指定要访问所请求镜像的协议；可能的值是 nbd, rbd ,rbd,sheepdog 或 gluster。

如果 protocol 属性是 rbd、heepdog 或 gluster，则需要一个额外的属性名称来指定将使用哪个卷和镜像。当磁盘类型是 网络 时，源 可能具有零个或多个 主机 子元素，用于指定要连接的主机物理机器，包括 type='dir' 和 type='network'。对于代表 cdrom 或 floppy（设备属性）的文件内容类型，可以在无法访问源文件时定义对磁盘执行的操作。这可以通过操作 startupPolicy 属性和以下值实现：

- 如果由于任何原因丢失，则 强制 会导致失败。这是默认设置。
- 如果引导时缺少，则会导致失败，如果缺少 migration/restore/revert
- 可选，如果任何开始尝试都缺少，则会丢弃

20.16.1.3. mirror 元素

如果虚拟机监控程序已启动了 BlockCopy 操作，则该文件中的镜像位置最终会与源的内容相同，且文件格式的格式为属性（源的格式可能会不同）。<>如果存在属性 ready，则已知磁盘已准备好 pivot；否则，磁盘可能仍进行复制。现在，此元素仅在输出中有效，它会被忽略。

20.16.1.4. 目标元素

<target> 元素控制磁盘公开给客户端虚拟机 OS 的总线 / 设备。dev 属性表示逻辑设备名称。指定的实际设备名称不能映射到客户端虚拟机 OS 中的设备名称。可选总线属性指定要模拟的磁盘设备类型；可能的值是特定驱动程序的驱动值，其中典型的值是 ide、scsi、virtio、xen、usb 或 sata。如果省略，总线类型从设备名称的样式推断出来。例如，名为 "sda" 的设备通常使用 SCSI 总线导出。可选属性 托盘 表示可移动磁盘的遍历状态（如 CD-ROM 或 Floppy 磁盘），可以 打开或关闭 该值。默认设置为 关闭。如需更多信息，请参阅 目标元素

20.16.1.5. iotune

可选的 <iotune> 元素提供额外的每个设备 I/O 调整功能，每个设备的值可能会有所不同（与 blkiotune 元素相对应的 blkiotune 元素进行全局应用到域）。此元素具有下列可选子元素：请注意，任何未指定的子元素或指定了值为 0 的子元素表示没有限制。

- <total_bytes_sec> - 总吞吐量限值（以字节/秒为单位）。此元素不能与 <read_bytes_sec> 或 <write_bytes_sec> 一起使用。
- <read_bytes_sec> - 每秒的读取吞吐量限制。

- <write_bytes_sec> - 每秒写入吞吐量限制（以字节为单位）。
- <total_iops_sec> - 每秒总 I/O 操作数。此元素不能与 <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。

20.16.1.6. driver

可选 <驱动程序> 元素允许指定与用来提供磁盘的虚拟机监控程序驱动程序相关的更多详情。可使用以下选项：

- 如果虚拟机监控程序支持多个后端驱动程序，则 name 属性选择主要后端驱动程序名称，而可选 type 属性则提供子类型。如需可能类型的列表，请参阅 驱动程序元素
- 可选 的缓存 属性控制缓存机制，可能的值有：默认、无、写回、直接同步 （类似于 writethrough，但会绕过主机物理机器页面缓存）和 不安全 （主机物理计算机可能会缓存所有磁盘 io，以及来自 guest 虚拟机虚拟机的同步请求）。
- 可选的 error_policy 属性控制虚拟机监控程序在磁盘读取或写入错误上的行为方式，可能的值 将停止、report、ignore 和 enospace。error_policy 的默认设置为 报告。还有一个可选的 rerror_policy，它只控制读取错误的行为。如果没有给出 rerror_policy，则 error_policy 会同时用于读写错误。如果给出了 rerror_policy，它将覆盖 error_policy 读取错误。另请注意，enospace 不是读取错误的有效策略，因此，如果 error_policy 被设置为 enospace 且未提供 rerror_policy，则会使用默认设置的读取错误，报告 会被使用。
- 可选的 io 属性控制 I/O 上的特定策略； qemu guest 虚拟机支持 线程 和 原生。可选的 ioeventfd 属性允许用户为磁盘设备设置域 I/O 异步处理。默认设置可以自由裁量使用虚拟机监控程序。接受的值为 on，关闭。启用此功能可让在单独的线程处理 I/O 时执行 guest 虚拟机。通常，在 I/O 期间具有高系统 CPU 利用率的客户机虚拟机将从此中受益。另一方面，过载主机物理机器可提高客户机虚拟机 I/O 延迟。除非您绝对认证了需要操作 io，否则强烈建议您不要更改默认设置并允许管理程序指定设置。
- 可选的 event_idx 属性控制设备事件处理的一些方面，并可以设置为 on 或 off - 如果它存在，它将减少中断数量并为客户机虚拟机退出。默认设置由管理程序确定，默认设置则位于 上。

在这种情况下，这个行为微不足道，此属性提供了一种强制关闭功能的方法 。除非是需要操作 event_idx 的绝对证书，否则强烈建议您不要更改默认设置并允许管理程序指定设置。

- 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 可避免重复访问同一后备文件扇区，当后备文件超过较慢的网络时很有用。默认 copy-on-read 为 。

20.16.1.7. 其他设备元素

以下属性可在 设备 元素中使用：

- <boot> - 指定磁盘可引导。

  其他引导值

  - <order> - 确定启动序列过程中将尝试的设备的顺序。
  - 在 BIOS 引导装载程序部分中无法与常规引导元素一起使用 <每个设备> 引导元素

- <加密> - 指定卷加密方式。如需更多信息，请参阅存储加密页面。

- <ReadOnly> - 表示客户机虚拟机无法修改该设备。此设置是具有 属性 device='cdrom' 的磁盘的默认设置。

- 可共享 设备预期在域间共享（只要虚拟机监控程序和操作系统支持）。如果使用 shareable，则 cache='no' 应该为该设备使用。

- <瞬态>- 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 会阻止域参与迁移或快照。

- <serial>- 指定客户机虚拟机的序列号。例如： <serial>WD-WMAP9A966149</serial>。

- WWN - 指定虚拟硬盘或 CD-ROM 驱动器的 WWN(World Wide Name)。<>它必须由 16 位十六进制数字组成。

- <vendor> - 指定虚拟硬盘或 CD-ROM 设备的厂商。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 4 属性： viz、名称、端口、传输和 套接字，分别指定主机名、端口号、传输类型和路径。此元素的含义和元素的数量取决于 协议 属性，如下所示：

  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

  - RBD - 监控 RBD 类型的服务器，并可用于一个或多个主机物理机器

  - sheepdog - 指定其中一个 sheepdog 服务器（默认为 localhost:700）,且可以使用一台或任何主机物理计算机

  - Gluster - 指定运行 glusterd 守护进程的服务器，只能用于一台主机物理机器。transport 属性的有效值为 tcp、rdma 或 unix。如果未指定任何内容，则假设 tcp。如果传输为 unix，则 socket 属性指定到 unix 套接字的路径。

- <address> - 指向控制器给定插槽的磁盘。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 总线、插槽 和 功能的其他属性，以及可选的 域和 多功能功能。multifun ction 默认为 off。对于 驱动器 控制器，还有额外的属性 控制器、总线、目标 和 单元，各自具有默认设置 0。

- auth - 提供访问源所需的身份验证凭证。它包括一个强制属性 username，用于标识身份验证期间要使用的用户名，以及带有强制属性 类型 的子元素 secret。此处可参见 设备元素的更多信息

- geometry - 提供覆盖 geometry 设置的能力。这在 S390 DASD-disks 或旧的 DOS-disks 中非常有用。

- 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-Modus，并具有以下值：none、lba 或 auto
- blockio - 允许使用以下列出的任意块设备属性覆盖块设备：

blockio 选项

- logical_block_size- 向客户机虚拟机虚拟机操作系统报告，并描述磁盘 I/O 的最小单元。
- physical_block_size - 向客户机虚拟机虚拟机操作系统报告，并描述磁盘的硬件扇区大小，它们可以与磁盘数据协调相关。

20.16.2. 文件系统

主机物理机器中的文件系统目录，可从 guest 虚拟机直接访问

图 20.24. 设备 - 文件系统

```
...
 <devices>
  <filesystem type='template'>
    <source name='my-vm-template'/>
    <target dir='/'/>
  </filesystem>
  <filesystem type='mount' accessmode='passthrough'>
    <driver type='path' wrpolicy='immediate'/>
    <source dir='/export/to/guest'/>
    <target dir='/import/from/host'/>
    <readonly/>
  </filesystem>
  ...
 </devices>
 ...
```

filesystem 属性具有以下可能的值：

- *type='mount' - 指定要在 guest 虚拟机中挂载的主机物理机器目录。如果没有指定默认类型。此模式也具有可选的子元素 驱动程序，其属性 type='path' 或 type='handle'。驱动程序块有一个可选属性 wrpolicy，可进一步控制与主机物理机器页面缓存的交互；省略属性会恢复到默认设置，而指定值会立即意味着主机物理机器写回会在客户机虚拟机文件写入操作中立即触发。*

- *type='template' - 指定 OpenVZ 文件系统模板，且仅对 OpenVZ 驱动程序使用。*

- *type='file' - 指定主机物理机器文件将被视为镜像并挂载到客户机虚拟机中。此文件系统格式将被自动检测，并且仅由 LXC 驱动程序使用。*

- *type='block' - 指定要在客户机虚拟机中挂载的主机物理机器块设备。文件系统格式将被自动检测，仅由 LXC 驱动程序使用。*

- *type='ram' - 使用主机物理机器操作系统中的内存指定内存文件系统。source 元素具有单一属性 使用量，它以 kibibytes 提供内存用量限制，并且仅由 LXC 驱动程序使用。*

- *type='bind' - 指定客户端虚拟机中的一个目录，它将绑定到客户机虚拟机中的其他目录。此元素仅由 LXC 驱动程序使用。*

- *access Mode 指定访问源的安全模式。目前，这只适用于 QEMU/KVM 驱动程序的 type='mount'。可能的值有：*

  - *Passthrough - 指定源可通过从客户机虚拟机内部设置的用户权限设置进行访问。如果没有指定默认访问模式，则这是默认访问模式。*

  - *mapping - 指定源可通过虚拟机监控程序的权限设置进行访问。*

  - *squash - Similar 到 "passthrough"，例外情况是忽略 chown 等特权操作失败。这使得类似直通的模式可用于将管理程序作为非 root 运行的用户。*

- *<source> - 指定在客户机虚拟机中要访问的主机物理机器上的资源。name 属性必须与 <type='template'> 一起使用，并且 dir 属性必须与 <type='mount'> 一起使用。usage 属性与*

<type='ram'> 一起使用，以在 KB 中设置内存限制。

- 目标 - 划分可以在 guest 虚拟机中访问源驱动程序的位置。对于大多数驱动程序来说，这是自动挂载点，但对于 QEMU-KVM 而言，这只是在 guest 虚拟机上导出为要挂载的任意字符串标签。

- ReadOnly - 启用将 sydtem 文件导出为 guest 虚拟机 的只读挂载，默认给定读写访问权限。

- space_hard_limit - 指定此客户机虚拟机文件系统可用的最大空间

- space_soft_limit - 指定此 guest 虚拟机文件系统中可用的最大空间。在宽限期内，容器可超过其软限制。之后会强制使用硬限制。

20.16.3. 设备地址

许多设备具有一个可选的 <地址> 子元素，用于描述在虚拟机上放置于虚拟总线上的设备的位置。如果输入时省略了地址（或地址中的任何可选属性）时，libvirt 将生成一个适当的地址；不过，如果需要更多地控制布局，则需要一个明确的地址。有关地址元素在内的设备示例，请参见以下设备。

每个地址都有一个强制属性 type，用于描述该设备所在的总线。在设备和客户机虚拟机架构中限制在给定设备使用的地址的选择。例如：磁盘设备使用 type='disk'，而控制台设备在 32 位 AMD 和 Intel 构架或者 AMD64 及 Intel 64 客户机虚拟机上使用 type='papr-vio'，或在 PowerPC64 台虚拟机中使用 type='spapr-vio'。每一地址 <类型> 具有额外的可选属性，可控制该设备在总线上的位置。其他属性如下：

- type='pci' - PCI 地址有以下附加属性：
  - 域 （2 字节十六进制整数，当前不供 qemu 使用）
  - 总线 （0 到 0 到 0xff 之间的十六进制值，含）
  - 插槽 （0x0 和 0x1f 之间的十六进制值，含）

- *功能 （0 到 7 之间的值）*
- *也可使用 多功能 属性，它控制在 PCI 控制寄存器中特定插槽/功能开启多个功能。这个多功能属性默认为 'off'，但应该设置为 'on' for a function 0，它使用多个功能的插槽中 0。*

- *type='drive - 驱动器地址具有以下额外属性：*
  - *controller- （2 位控制器号）*
  - *总线 - （2 位总线号）*
  - *Target - （2 位总线号）*
  - *unit - （总线中 2 位的单元数）*
- *type='virtio-serial' - 每个 virtio-serial 地址都有以下附加属性：*
  - *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' - 用于智能卡的 CCID 地址，具有以下附加属性：*
  - *总线 - （2 位总线号）*
  - *插槽 属性 - （总线中的 2 位插槽）*

- type='usb' - USB 地址有以下附加属性：
  - 总线 -（0 到 0 到 0xfff 之间的十六进制值，含）
  - 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，设备可以分配给 SPAPR-VIO 总线。它具有扁平 64 位地址空间；根据惯例，设备通常在零个 0x1000 的零次分配，但其他地址由 libvirt 有效并允许。额外属性：reg（开始寄存器的十六进制值地址）可以分配给此属性。

20.16.4. controllers

根据客户机虚拟机架构，可以为单个总线分配多个虚拟设备。在正常情况下，libvirt 可以自动推断控制器用于总线的情况。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 20.25. 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" 的强制属性 索引，它是强制属性索引，描述总线控制器被遇到的十进制整数（用于控制器元素的控制器属性）。"virtio-serial" 控制器具有两个额外的可选属性，即端口和 向量，它控制能够通过控制器连接多少个设备。

<控制器 type='scsi'> 有一个可选的属性 模型，它是 "auto"、"buslogic"、"ibmvscsi"、"lsilogic"、"lsias1068"、"virtio-scsi 或 "vmpvscsi "。应注意，virtio-scsi 控制器和驱动程序将在 KVM 和 Windows 客户机虚拟机上均工作。<控制器 type='scsi'> 也具有属性 num_queues，它为指定的队列数量启用多队列支持。

"usb" 控制器具有可选属性 模型，它是 "piix3-uhci", "piix4-uhci", "ehci", "ich9-ehci1", "ich9-uhci1", "ich9-uhci2", "ich9-uhci3", "vt82c686b-uhci", "pci-ohci " 或 "nec-xhci "。另外，如果需要为 guest 虚拟机明确禁用 USB 总线，则可以使用 model='none'。PowerPC64 "spapr-vio" 地址没有关联的控制器。

对于 PCI 或 USB 总线上的设备，可选的子元素 地址 可以使用上面给出的语义指定控制器与其主总线的确切关系。

USB companion 控制器具有一个可选的子元素 master，用于指定与主控制器相配套的关系。companion 控制器与其 master 位于同一个总线上，因此相应的索引值应该相等。

图 20.26. 设备 - 控制器 - 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>
...
```

20.16.5. 设备租用

当使用锁定管理器时，您可以选择记录针对客户机虚拟机的设备租期。锁定管理器将确保客户机虚拟机不会启动，除非可以获取租赁。使用传统管理工具进行配置时，xml 的以下部分将生效：

图 20.27. devices - 设备租期

```
...
<devices>
  ...
  <lease>
    <lockspace>somearea</lockspace>
    <key>somekey</key>
    <target path='/some/lease/path' offset='1024'/>
  </lease>
  ...
</devices>
...
```

lease 部分可以具有以下参数：

- lockspace - 标识保存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的限制。
- key - 一个任意字符串，可唯一标识要获取租期。锁定管理器可能会对密钥的格式或长度施加额外的限制。
- Target - 与锁定空间相关联的文件的完全限定路径。偏移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，请将此值设置为 0。

20.16.6. 主机物理机器设备分配

本节介绍主机物理机器设备分配的信息。

20.16.6.1. USB/ PCI 设备

主机物理机器的 USB 和 PCI 设备可以通过 hostdev 元素传递给客户机虚拟机，方法是使用域 xml 文件的以下管理工具修改主机物理机器：

图 20.28. 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>
...
```

或者，也可以执行以下操作：

图 20.29. 设备 - 主机物理机器设备分配替代设备

```
...
<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>
...
```

域 XML 中本节的组件如下：

表 20.13. 主机物理机器设备分配元素

参数	描述
hostdev	这是描述主机物理机器设备的主要容器。对于 USB 设备透传 **模式** 总是 **子系统**，对于 USB 设备，**类型为 usb**，对于 PCI 设备，则为 **pci**。当对 PCI 设备 **托管** 是 **yes** 时，它会在主机物理机器中分离开来，然后再传递至客户端虚拟机，并在虚拟机退出后重新附加到主机物理机器。如果省略或 **没有** PCI 设备，则用户负责在热拔或停止客户端虚拟机后使用 **virNodeDeviceDettach**（或 **virsh nodedev-dettach**）的参数 **virNodeDeviceReAttach**（或 **virsh nodedev-reattach**）。

参数	描述
source	描述主机物理机器中看到的设备。可以通过供应商/产品 ID 使用供应商和 **产品元素或主机** 物理机器上的设备地址或主机物理机器上的设备地址来解决 USB 设备。 **另一方面，PCI 设备仅可通过其地址进行描述。请注意，USB 设备的源元素可能包含 start Policy 属性，可用于在未找到指定主机物理机器 USB 设备时定义规则。该属性接受以下值：** • **必需的 - 如果因任何原因（默认）而失败。** • **requisite - 如果引导中缺少，则会在 migrate/restore/revert 上缺少 requisite - 失败** • **可选 - 如果任何开始尝试都缺少，则丢弃**
vendor, 产品	这些元素都有一个 **id** 属性，用于指定 USB 供应商和产品 ID。ID 可使用十进制、十六进制（以 0x 开始）或八进制（以 0 开始）形式。
boot	指定设备可引导。属性的顺序决定了在启动序列期间将尝试设备的顺序。在 BIOS 引导装载程序部分中无法与常规引导元素一起使用每个设备引导元素。
rom	用于改变如何将 PCI 设备的 ROM 呈现给客户端虚拟机。可选 **条** 属性可以设置为 **on** 或 **off**，并确定该设备的 ROM 是否在客户机虚拟机的内存映射中可见。（在 PCI 文档中，**rombar** 设置控制 ROM Base Address Register 的存在。如果没有指定 rom bar，则将使用默认设置。可选的 **file** 属性用于指向作为设备 ROM BIOS 向虚拟客户机呈现的二进制文件。例如，这对支持 sr-iov 功能的虚拟功能（VF 没有引导 ROM）的虚拟功能提供 PXE 引导 ROM 非常有用。

参数	描述
address	另外，也有一个 **总线** 和设备属性，用于指定 **设备** 出现在主机物理机器上的 USB 总线和设备号码。这些属性的值可以用十进制、十六进制（以 0x 开始）或八进制（以 0 开始）形式给出。对于 PCI 设备，元素执行 3 属性，允许将设备指定为 **lspci** 或 **virsh nodedev-list**

20.16.6.2. 块/字符设备

主机物理机器的块设备可使用管理工具传递给 guest 虚拟机，以修改域 xml hostdev 元素。请注意，这只适用于基于容器的虚拟化。

图 20.30. 设备 - 主机物理机器设备分配块设备

```
...
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
```

另一种方法是：

图 20.31. 设备 - 主机物理机器设备分配块设备替代 1

```
...
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
```

另一种替代方法是：

图 20.32. 设备 - 主机物理机器设备分配块设备替代 2

```
...
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
```

域 XML 中本节的组件如下：

表 20.14. 块/字符设备元素

参数	描述
hostdev	这是描述主机物理机器设备的主要容器。对于块/字符设备透传 **模式**，其类型始终为块设备和 **字符设备** 的收费。
source	这描述了在主机物理机器中看到的设备。对于块设备，主机物理机器 OS 中的块设备路径在嵌套 **块** 元素中提供，而对于使用 **char** 元素的字符设备

20.16.7. 重定向设备

支持通过字符设备进行 USB 设备重定向，可通过管理工具进行配置，该工具修改域 xml 的以下部分：

图 20.33. 设备 - 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 中本节的组件如下：

表 20.15. 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。**总线** 必须是 **usb** for a USB 设备。需要额外属性类型，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）。redirdev 元素具有一个可选的子元素 **地址**，可将设备绑定到特定的控制器。此外，可以根据给定 **类型** 的要求子元素（尽管不需要**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 guest 虚拟机中可见的设备）。
boot	指定设备可引导。order 属性决定在引导序列中尝试的设备的顺序。在 BIOS 引导装载程序部分中无法与常规引导元素一起使用每个设备引导元素。
Redirfilter	这用于创建过滤规则从重定向过滤某些设备。它使用子元素 **usbdev** 来定义各个过滤规则。**class** 属性是 USB 类代码。

20.16.8. 智能卡设备

可以通过智能卡元素向客户机虚拟机提供虚拟 智能卡 设备。主机机器上的 USB 智能卡读取器设备无法在具有简单设备透传的 guest 上使用，因为它不能同时提供给主机和客户机，并在从客户机中删除主机计算机时锁定主机计算机。因此，一些虚拟机监控程序提供特殊的虚拟设备，它可向客户机虚拟机提供智能卡接口，具有几种模式来描述如何从主机或第三方智能卡供应商创建的频道获取凭证。要通过字符设备设置 USB 设备重定向的参数，请编辑域 XML 的以下部分：

图 20.34. 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。支持以下模式；在各个模式下，客户机虚拟机在其 USB 总线上看到设备，其行为类似于物理 USB CCID(Chip/Smart Card Interface Device)卡。

模式属性如下：

表 20.16. SmartCard 模式元素

参数	描述
mode='host'	在这个模式中，管理程序通过 NSS 将客户机虚拟机的所有直接访问请求转发到主机物理机器的智能卡。不需要其他属性或子元素。请参阅以下关于使用可选 **地址** 子元素的信息。
mode='host-certificates'	这个模式允许您提供位于主机物理机器的数据库中的三个 NSS 证书名称，而不需要插入到主机物理机器中的智能卡。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**，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（在创建证书时与 **certutil** 命令的 **-d** 选项匹配）；如果不存在，则默认为 **/etc/pki/nssdb**。

参数	描述
mode='passthrough'	此模式允许您通过二级字符设备将所有请求传输到第三方提供程序（而后者又与智能卡或使用三个证书文件）进行传输，而不是让虚拟机监控程序直接与主机物理计算机通信。在这个模式中，需要一个额外的属性 **类型**，与受支持的串行设备类型之一匹配，以描述隧道的主机物理机器侧；**type='tcp'** 或 **type='spicevmc'**（它使用 SPICE 图形设备的智能卡频道）。此外，可以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，而不是虚拟客户机中可见的设备）。

每个模式支持可选的子元素 地址，它微调智能卡与 ccid 总线控制器之间的相关性（推荐至 第 20.16.3 节 “设备地址”）。

20.16.9. 网络接口

使用管理工具修改网络接口设备，该工具将配置 Domain XML 的以下部分：

图 20.35. 设备 - 网络接口

```
...
<devices>
  <interface type='bridge'>
    <source bridge='xenbr0'/>
    <mac address='00:16:3e:5d:c7:9e'/>
    <script path='vif-bridge'/>
    <boot order='1'/>
    <rom bar='off'/>
  </interface>
</devices>
...
```

指定对客户机虚拟机可见的网络接口有几种可能。以下每个小节都提供有关常见设置选项的更多详情。另外，每个 <interface> 元素具有一个可选 <地址> 子元素，它可以将接口绑定到特定的 pci 插槽，带属性 type='pci' (Refer to 第 20.16.3 节 “设备地址”)。

20.16.9.1. 虚拟网络

这是在主机带有动态/无线网络配置（或多主机物理机器环境（主机物理机器硬件详细信息中单独描述）上常规的客户机虚拟机连接的建议配置。<>另外，它提供了一个连接，其详情由指定的网络定义描述。根据虚拟网络的 转发模式 配置，网络可以完全隔离（未声明 <转发> 元素）、NAT 到显式网络设备

或默认路由（转发模式='nat'），路由没有 NAT（转发 mode='route'/），或者直接连接到其中一个主机物理机器的网络接口（使用 macvtap）或桥接设备（转发模式='private| 网桥）

对于具有转发模式 bridge、private、vepa 和 passthrough 的网络，假定主机物理机器在 libvirt 范围内已经有必要的 DNS 和 DHCP 服务。如果是隔离、nat 和路由网络，DHCP 和 DNS 在虚拟网络上由 libvirt 提供，可以通过使用 virsh net-dumpxml [networkname] 检查虚拟网络配置来确定。有一个虚拟网络为 'default' 设置，它针对默认路由进行 NAT' 并有一个 IP 范围 192.168.122.0/255.255.255.0。每个客户机虚拟机都将使用名称 vnetN 创建关联的 tun 设备，该设备也可通过 <target> 元素覆盖（请参考 第 20.16.9.11 节 “覆盖 target 元素”）。

当接口源是网络时，可以指定端口组以及网络的名称；一个网络可以定义多个端口组，每个 portgroup 含有不同类网络连接的稍有不同的配置信息。另外，类似 <直接> 网络连接（如下所示），类型为 network 的连接也可以指定 <virtualport> 元素，并将配置数据转发到 vepa(802.1Qbg)或 802.1Qbh 兼容交换机，或者 Open vSwitch 虚拟交换机。

由于实际类型的交换机可能因主机物理机器上网络的配置而不同，因此可以接受省略 <virtualport> 类型属性，并从多个不同的虚拟端口类型（以及离开某些属性）指定属性；在域启动时，完整的虚拟端口元素将由网络中定义的类型和属性合并，从而构成接口所引用的类型和属性。<>新结构化的虚拟端口是两者的组合。低虚拟端口中的属性无法对以更高的虚拟端口定义的项进行更改。接口具有最高优先级，端口组最低优先级。

例如，若要创建具有 802.1Qbh 交换机和 Open vSwitch 交换机的网络，您可以选择不指定类型，但必须要提供 profileid 和 interfaceid。要从虚拟端口填充的其他属性（如 managerid、typeid 或 profileid ）是可选的。

如果要将客户机虚拟机限制为仅连接到某些类型交换机，您可以指定虚拟端口类型，并且只有使用指定端口类型的交换机才会连接。您还可以通过指定附加参数来进一步限制交换机连接。因此，如果指定端口并且主机物理机器的网络具有不同类型的虚拟端口，接口的连接将失败。虚拟网络参数使用管理工具来定义，这些文件修改域 XML 的以下部分：

图 20.36. 设备 - 网络接口虚拟网络

```
...
<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>
...
```

20.16.9.2. 桥接到 LAN

请注意，这是在带有静态有线网络配置的主机物理机器上常规客户机虚拟机连接的建议配置设置。

bridge 到 LAN 提供了从客户机虚拟机直接位于 LAN 的桥接。这假设主机物理机器上有一个桥接设备，它包含一个或多个主机物理 NIC。客户机虚拟机将使用名称 <vnetN> 创建关联的 tun 设备，该设备也可通过 <target> 元素覆盖（请参考 第 20.16.9.11 节 “覆盖 target 元素”）。<tun> 设备将从属于网桥。IP 范围 / 网络配置是在 LAN 中使用的任何情况。这为虚拟客户机提供完全传入和传出的网络访问，就像物理机一样。

在 Linux 系统中，桥接设备通常是标准 Linux 主机物理机器桥接。在支持 Open vSwitch 的物理机上，也可以通过向接口定义中添加 virtualport type='openvswitch'/ 来连接到 Open vSwitch 网桥设备。Open vSwitch 类型 virtualport 接受 其参数 元素中的两个参数 - interfaceid 是一个标准 uuid，用于为 Open vSwitch 唯一标识此特定接口（如果您不指定任何指定，那么当您首次定义接口时，将生成一个随机的 interfaceid，以及发送到 Open vSwitch 的接口的可选 < profile id> 作为接口）。要将网桥设置为 LAN 设置，请使用可配置以下域 XML 部分的管理工具：

图 20.37. 设备 - 网络接口桥接到 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>
```

20.16.9.3. 设置端口伪装范围

如果要设置端口伪装范围，可以按照如下所示设置端口：

图 20.38. 端口伪装范围

```
<forward mode='nat'>
   <address start='192.0.2.1' end='192.0.2.10'/>
</forward>  ...
```

这些值应使用 iptables 命令设置，如下所示 第 18.3 节 “网络地址转换模式”

20.16.9.4. 用户空间 SLIRP 堆栈

设置用户空间 SLIRP 堆栈参数可提供虚拟 LAN 以及外部世界的 NAT。虚拟网络具有 DHCP 和 DNS 服务，并将从 10.0.2.15 开始给客户机虚拟机提供 IP 地址。默认路由器为 10.0.2.2，DNS 服务器将是 10.0.2.3。此联网是需要客户机虚拟机具有传出访问权限的非特权用户的唯一选项。

用户空间 SLIP 堆栈参数在域 XML 的以下部分定义：

图 20.39. 设备 - 网络接口空间 SLIRP 堆栈

```
...
<devices>
  <interface type='user'/>
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55"/>
  </interface>
</devices>
...
```

20.16.9.5. 通用以太网连接

提供管理员执行任意脚本以将客户机虚拟机网络连接到 LAN 的方法。客户机虚拟机将创建一个使用名称 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，会运行一个 shell 脚本，该脚本应该执行任何主机物理机器网络集成。默认情况下，此脚本名为 /etc/qemu-ifup，但可以被覆盖（请参阅 第 20.16.9.11 节 “覆盖 target 元素”）。

通用以太网连接参数在域 XML 的以下部分定义：

图 20.40. 设备 - 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>
...
```

20.16.9.6. 直接附加到物理接口

使用 <interface type='direct'> 将虚拟机的 NIC 附加到主机上的指定物理接口。

这个设置需要可用的 Linux macvtap 驱动程序。对于 macvtap 设备的操作模式，可以选择下列模式之一： vepa （虚拟以太网端口聚合器"），它是默认的模式，即 网桥或专用。

要设置到物理接口的直接附加，在域 XML 中使用以下参数：

图 20.41. 设备 - 网络接口直接附加到物理接口

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
...
```

单独的模式会导致传输数据包的行为，如 表 20.17 “直接附加到物理接口元素” 所示：

表 20.17. 直接附加到物理接口元素

元素	描述
vepa	所有虚拟客户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数据包，其数据包源自于 VEPA 功能通过 VEPA（日常网桥）向主机物理机器发送回主机物理机器。
bridge	其目的地位于同一主机物理机器上的数据包，其源自于目的地 macvtap 设备直接发送到目标 macvtap 设备。原始设备和目的地设备都需要处于网桥模式，才能直接传送。如果其中任何一个处于 **vepa** 模式，则需要一个 VEPA 功能网桥。
private	所有数据包都发送到外部网桥，只有当它们通过外部路由器或网关发送回主机物理机器时，它们才会传送到同一主机物理机器上的目标虚拟机。如果源或目标设备处于私有模式，则执行此步骤。
passthrough	此功能将 SRIOV 能力 NIC 的虚拟功能直接附加到客户机虚拟机，而不会丢失迁移功能。所有数据包都发送到配置的网络设备的 VF/IF。根据设备的功能额外先决条件或限制，例如，这需要内核 2.6.38 或更高版本。

直接附加虚拟机的网络访问可以由主机物理计算机连接到的物理接口来管理。

如果交换机符合 IEEE 802.1Qbg 标准，接口可以还有其他参数。virtualport 元素的参数在 IEEE 802.1Qbg 标准中更详细地阐述。值特定于网络，应当由网络管理员提供。在 802.1Qbg 术语中，虚拟工作站(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。

表 20.18 “直接附加到物理接口的其他元素” 中描述了可操作的其他元素：

表 20.18. 直接附加到物理接口的其他元素

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，赋予值 **0**。
typeid	VSI Type ID 标识 VSI 类型特征，以优化网络访问。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	在创建 VSI 实例（这是虚拟机虚拟接口）时，将生成 VSI 实例 ID 标识符。这是全局唯一标识符。
profileid	配置集 ID 包含要应用于此接口的端口配置集的名称。此名称由端口 profile 数据库解析为来自端口配置集的网络参数，这些网络参数将应用到此接口。

域 XML 中的其他参数包括：

图 20.42. 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，接口可以有其他参数，如下所示。值特定于网络，应当由网络管理员提供。

域 XML 中的其他参数包括：

图 20.43. devices - 网络接口直接附加到物理接口的其他参数

```
...
<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。此名称由端口 profile 数据库解析为来自端口配置集的网络参数，这些网络参数将应用到此接口。

20.16.9.7. PCI 透传

PCI 网络设备（由 源 元素指定）直接分配至使用通用设备透传的客户机虚拟机，然后首先将设备的 MAC 地址设置为配置的值，并将设备与 802.1Qbh 进行交换机关联，并使用可选的指定 虚拟端口 元素（请参阅上述为 type='direct 的网络设备提供的虚拟端口的示例）。请注意 - 由于标准单端口 PCI 以太网卡驱动程序设计的限制 - 只有 SR-IOV（Single Root I/O 虚拟化）虚拟功能(VF)设备可以采用这种方式分配；若要为客户机虚拟机分配一个标准单端口 PCI 或 PCIe 以太网卡，请使用传统的 hostdev 设备定义

请注意，网络设备的这种"智能直通"与标准 hostdev 设备的功能非常相似，这种方法的区别在于，这种方法允许为通过设备指定 MAC 地址 和虚拟端口。如果没有需要这些功能，如果您有一个支持 SR-IOV 的标准单端口 PCI、PCIe 或 USB 网卡（因此，在分配到客户机虚拟机域后，任何时候都会丢失配置的 MAC 地址），或者如果您使用比 0.9.11 旧版本的 libvirt.9.11，您应该使用标准 hostdev 将设备分配给 guest 虚拟机而非 host/dev。

图 20.44. 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>
...
```

20.16.9.8. 多播隧道

多播组可用于表示虚拟网络。任何网络设备位于相同多播组的客户机虚拟机都将相互通信，即使它们位于多个物理主机物理机器中。此模式可以作为一个非特权用户使用。没有默认的 DNS 或 DHCP 支持，且没有传出网络访问。要提供传出网络访问，其中一个 guest 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，使用的源地址必须是来自多播地址块。使用管理工具处理接口类型来创建多播隧道，并将它设置为 mcast，并提供 mac 和源地址。结果在域 XML 的更改中显示：

图 20.45. 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>
...
```

20.16.9.9. 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法，其中一台 guest 虚拟机提供网络的服务器结尾，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虚拟客户机路由。此模型也可用于非特权用户。没有默认的 DNS 或 DHCP 支持，且没有传出网络访问。为了提供传出网络访问，其中一个 guest 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。通过操作接口类型创建 TCP 隧道，并将它设置为服务器或客户端，并提供 mac 和源地址。 结果在域 XML 的更改中显示：

图 20.46. 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>
...
```

20.16.9.10. 设置特定于 NIC 驱动程序的选项

有些 NIC 可能会有特定于驱动程序的可调整选项。这些选项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配置域 XML 的以下部分来设置：

图 20.47. 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>
...
```

目前，以下属性可用于 "virtio" NIC 驱动程序：

表 20.19. virtio NIC 驱动程序元素

参数	描述
name	可选 **name** 属性强制使用哪些后端驱动程序。该值可以是 **qemu**（用户空间后端）或 **vhost**（内核后端），它需要由内核提供 vhost 模块；在没有内核支持内核时，会尝试拒绝 vhost 驱动程序。如果 **vhost** 驱动程序存在，则默认设置为 vhost，但如果不存在，则会静默返回到 **qemu**。

参数	描述
txmode	指定在传输缓冲区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**，则会在驱动程序底部一半的 iothread 中完成数据包 tx（此选项转换为将 **"tx=bh"** 添加到 **qemu** 命令行 -device virtio-net-pci 选项）。如果设置为 **timer**，tx 工作在 qemu 中完成；如果存在比当时发送的更多 tx 数据，则会在 qemu 移动进行其他操作前设置计时器；当计时器触发时，将进行另一个尝试来发送更多数据。通常情况下，您应该只保留这个选项，除非您特别需要修改它。
ioeventfd	允许用户设置接口设备的域 I/O 异步处理。默认设置可以自由裁量使用虚拟机监控程序。接受的值为 **on**，**关闭** 。启用此选项可让 qemu 在单独的线程处理 I/O 时执行客户机虚拟机。通常，在 I/O 期间具有高系统 CPU 使用率的虚拟机将从此中受益。另一方面，加载物理主机物理机器也可能会增加客户机虚拟机 I/O 延迟。因此，您应该只保留这个选项，除非您特别需要修改它。
event_idx	event_idx 属性控制设备事件处理的一些方面。该值可以是 **on** 或 **off**。在 **上** 选择，减少 guest 虚拟机的中断数量并退出。默认为在 **上**。如果出现这种行为低效的情况，此属性提供了一种强制关闭功能的方法。除非您只需要修改它，否则您应该只保留这个选项。

20.16.9.11. 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 20.48. devices - 网络接口覆盖目标元素

```
...
<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。可以手动指定此名称，但是该名称不能以 '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。

20.16.9.12. 指定引导顺序

要指定引导顺序，使用管理工具对域 XML 进行以下更改：

图 20.49. 指定引导顺序

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <boot order='1'/>
  </interface>
</devices>
...
```

对于支持它的虚拟机监控程序，您可以设置要用于网络引导的特定 NIC。属性顺序决定了在引导序列过程中尝试的设备的顺序。请注意，在 BIOS 引导装载程序部分中无法与常规引导元素一起使用每个设备引导元素。

20.16.9.13. 接口 ROM BIOS 配置

要指定 ROM BIOS 配置设置，使用管理工具对域 XML 进行以下更改：

图 20.50. 接口 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，您可以更改如何将 PCI 网络设备的 ROM 呈现给客户端虚拟机。bar 属性可以设置为 on 或 off，并确定该设备的 ROM 是否在客户机虚拟机的内存映射中可见。（在 PCI 文档中，"rombar"设置控制 ROM Base Address Register 的存在。如果没有指定 rom bar，则将使用 qemu 默认版本（旧的 qemu 版本默认使用 off，而较新的 qemus 则默认使用）。可选的 file 属性用于指向作为设备 ROM BIOS 向虚拟客户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。

20.16.9.14. 服务质量

域 XML 的这一部分提供了设置服务质量。传入和传出流量可以独立形成。带宽 元素最多可以有一个入站和最多一个出站子元素。将任何子项保留掉，不会在该流量方向应用 QoS。因此，当您只想形成域的传入流量时，请仅使用入站流量，反之亦然。

每个元素都有一个必需的属性 平均值 （ 或按 下面描述）。平均 指定所组成接口的平均位率。然后有两个可选属性： peak 指定接口可以发送数据的最大速率，而 burst 则指定在高峰速度可突发的字节数。属性的接受值为整数。

平均 和 峰值 属性的单位是每秒千字节的 KB，而 burst 仅以 KB 为单位设置。此外，入站流量可选择性地具有 floor 属性。这可保证所组成接口的最小吞吐量。使用 floor 要求所有流量穿过 QoS 决策制定的一点。因此，它仅可在 interface type='network'/ 带有 转发 类型 的路由、nat 或根本不转发的情况下使用。应注意，在虚拟网络内，所有连接的接口都需要至少具有入站 QoS 设置（至少至少），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 只能应用于入站和出站流量。

要指定 QoS 配置设置，使用管理工具对域 XML 进行以下更改：

图 20.51. 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>
...
```

20.16.9.15. 设置 VLAN 标签（仅在支持的网络类型中）

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 20.52. 设置 VLAN 标签（仅在支持的网络类型中）

```
...
<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（仅当）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，则可选的 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（openvswitch 和 type='hostdev' 的 SR-IOV 接口支持客户机虚拟机流量的透明 vlan 标记；否则，包括标准的 Linux 网桥和 libvirt 自身的虚拟网络）不支持透明的 vlan 标记。802.1Qbh(vn-link)和 802.1Qbg(VEPA)交换机提供自己的方法（在 libvirt 外），将客户机虚拟机流量标记到特定 vlans。）要允许指定多个标签（如果是 vlan 中继），一个子元素标签，指定要使用的 vlan 标签（例如： tag id='42'/ ）。如果接口定义了多个 vlan 元素，则假设用户希望使用所有指定的标签进行 VLAN 中继。如果需要具有单个标签的 vlan 中继，可选属性 trunk='yes' 可以添加到顶级 vlan 元素中。

20.16.9.16. 修改虚拟链接状态

此元素提供了设置虚拟网络链接状态的方法。属性 状态 的可能值为 up 和 down。如果将 down 指定为值，接口的行为就如同网络连接了网络电缆。如果此元素未指定，则默认行为为具有链接状态。

要指定虚拟链接状态配置设置，使用管理工具对域 XML 进行以下更改：

图 20.53. 修改虚拟链接状态

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <link state='down'/>
  </interface>
<devices>
...
```

20.16.10. 输入设备

输入设备允许与客户机虚拟机中的图形帧缓冲器交互。当启用帧缓冲时，会自动提供一个输入设备。可以显式添加其他设备，例如，为绝对光标移动提供图形表格。

要指定输入设备配置设置，使用管理工具对域 XML 进行以下更改：

图 20.54. 输入设备

```
...
<devices>
  <input type='mouse' bus='usb'/>
</devices>
...
```

<输入> 元素具有一个强制属性： type，可以将其设置为： mouse 或 tablet。后者提供绝对光标移动，而前者使用相对移动。可选 总线 属性可用于优化确切的设备类型，并可以设置为： xen （稀疏）、ps2 和 usb。

输入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，如上文所述。

20.16.11. hub Devices

hub 是一个将单个端口扩展到多个设备，以便有更多端口可用于将设备连接到主机物理机器系统。

要指定 hub 设备配置设置，使用管理工具对域 XML 进行以下更改：

图 20.55. hub 设备

```
...
<devices>
  <hub type='usb'/>
</devices>
...
```

hub 元素具有一个强制属性，其值只能是 usb 的类型。hub 元素具有一个带有 type='usb'的可选子元素 地址，可将设备绑定到特定的控制器。

20.16.12. 图形帧缓冲

图形设备允许与客户端虚拟机操作系统进行图形化交互。客户机虚拟机通常具有帧缓冲器或配置为允许与管理员交互的文本控制台。

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 20.56. 图形框架

```
...
<devices>
  <graphics type='sdl' display=':0.0'/>
  <graphics type='vnc' port='5904'>
    <listen type='address' address='192.0.2.1'/>
  </graphics>
  <graphics type='rdp' autoport='yes' multiUser='yes' />
  <graphics type='desktop' fullscreen='yes'/>
  <graphics type='spice'>
    <listen type='network' network='rednet'/>
  </graphics>
</devices>
...
```

图形元素具有强制 类型 属性，它取 sdl、vnc、rdp 或 desktop 的值，如下所示：

表 20.20. 图形帧缓冲元素

参数	描述
sdl	这将在主机物理机器桌面上显示一个窗口，它可使用 3 个可选参数：显示要使用的 **display** 属性、身份验证标识符的 **xauth** 属性以及可选的 **全屏** 属性接受值 **yes** 或 **no**
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 VNC 密码。**keymap** 属性指定要使用的 keymap。可能会设置密码有效期的限制，给出了 **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 接受 **仅保留** 值，请注意，所有管理程序可能都不支持它。QEMU 支持在 unix 域套接字路径中侦听的 socket 属性，而不使用 listen/port。

参数	描述
spice	启动 SPICE 服务器。**port** 属性指定 TCP 端口号（使用 -1 作为旧语法，表示它应该被自动分配），而 **tlsPort** 则提供了一个替代的安全端口号。**autoport** 属性是指示两个端口号的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** 属性以明文中提供 SPICE 密码。**keymap** 属性指定要使用的 keymap。可能会设置密码有效期的限制，给出了 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 接受了保持客户端连接、断开与客户端断开连接并且无法更改密码。请注意，它并不被所有虚拟机监控程序支持。**defaultMode** 属性设置默认频道安全策略，有效的值 为**安全**、**不安全和** 默认值（如果可能，这比较**安全**，但没有安全的路径，则不会出错）。

当 SPICE 同时配置了正常和 TLS 保护的 TCP 端口时，可能需要限制可在每个端口上运行哪些频道。这可以通过在主图形元素中添加一个或多个 频道 元素 来实现。有效的频道名称包括 主、显示、输入、光标、回放、记录、智能卡、和 usbredir。

要指定 SPICE 配置设置，使用 mangement 工具对域 XML 进行以下更改：

图 20.57. SPICE 配置

```
<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 支持音频、镜像和流处理的变量压缩设置。这些设置可通过以下所有元素中的压缩方式来访问：用于设置 映像 压缩（接受 auto_glz、auto_lz、fast、gz、Llz、Lz、off）、jpeg for JPEG 压缩镜像通过 wan（接受自动、始终为接受）和 playback 配置 wan 镜像压缩（接受，始终为行）。

streaming 模式由 streaming 元素设置，将其 mode 属性设为 过滤器 之一，即全部 或关闭。

此外，复制和粘贴功能（使用 SPICE 代理）由 剪贴板 元素进行设置。它默认是启用的，可以通过将 copypaste 属性设置为 no 来禁用。

鼠标模式由 鼠标 元素设置，将其 模式 属性设置为其中一个 服务器或客户端 。如果没有指定模式，则将使用 qemu 默认（客户端 模式）。

其他元素包括：

表 20.21. 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 端口自动分配的新首选语法。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。当一个新客户端以单一连接模式连接时，无论是否必须丢弃现有连接，并且 VRDP 服务器都需要建立新的连接。
desktop	这个值目前为 VirtualBox 域保留。它在主机物理机器桌面上显示一个窗口，类似于"sdl"，但使用 VirtualBox 查看器。正如"sdl"一样，它接受可选的属性显示和全屏。

参数	描述
listen	可在图形、**listen** 属性、名为 listen 的独立子元素（称为 **listen** ）中的用于为 **图形** 类型 vnc 和 spice 设置监听套接字的地址信息（请参阅上面的示例）。**侦听 以下属性：** • **Type - 设置为地址或网络。这告知此监听元素是指定要直接使用的地址，或通过命名网络（然后用来确定要侦听的相应地址）。** • **address - 此属性将包含 IP 地址或主机名（通过 DNS 查询解析为 IP 地址）以侦听。在运行域的"实时" XML 中，此属性将设置为用于侦听的 IP 地址，即使 type='network'。** • **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。将检查指定的网络配置来确定适当的侦听地址。例如，如果网络在其配置中有 IPv4 地址（例如，如果网络具有转发类型路由、nat 或 no forward type（隔离），则会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机器桥接，将使用与该网桥设备关联的第一个 IPv4 地址，如果网络描述其中一个"直接"(macvtap)模式，则将使用第一个转发 dev 的第一个 IPv4 地址。**

20.16.13. 视频设备

视频设备。

要指定视频设备配置设置，使用管理工具对域 XML 进行以下更改：

图 20.58. 视频设备

```
...
<devices>
  <video>
    <model type='vga' vram='8192' heads='1'>
      <acceleration accel3d='yes' accel2d='yes'/>
    </model>
  </video>
</devices>
...
```

图形元素具有强制 类型 属性，它采用 "sdl", "vnc", "rdp" 或 "desktop" 的值，如下所示：

表 20.22. 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 中没有图形元素，libvirt 将根据虚拟客户机类型添加默认 **视频。如果没有提供默认值，则使用"ram"或"vram"。**
model	这有一个强制 **类型** 属性，它取值 **vga**、**cirrus**、**vmvga**、**xen**、**vbox** 或 **qxl**，具体取决于可用的管理程序功能。您还可以使用 vram 以及数据数以 kibibytes（1024 字节块）提供视频内存量。
acceleration	如果支持使用 **accel3d** 和 **accel2d** 属性启用 **加速** 功能。
address	可选地址子元素可用于将视频设备绑定到特定的 PCI 插槽。

20.16.14. 控制台、Serial、Parallel 和 Channel Devices

字符设备提供了一种与虚拟机交互的方法。半虚拟化控制台、串行端口、并行端口和通道均以字符设备分类，因此使用相同的语法表示。

要指定 consols、channels 和其它设备配置设置，使用管理工具对域 XML 进行以下更改：

图 20.59. 控制台、串行、并行和通道设备

```
...
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2'/>
    <target port='0'/>
  </parallel>
  <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（parallel、串行、控制台、频道）描述如何向客户机虚拟机显示该设备。客户机虚拟机接口由 target 元素配置。提供给主机物理计算机的接口在顶级元素的 type 属性中提供。主机物理机器接口由源元素配置。source 元素可能包含可选的 seclabel，用于覆盖在套接字路径中标记完成的方式。如果没有此元素，则安全标签从每个域设置中继承。每一字符设备元素具有一个可选的子元素 地址，可将设备绑定到特定的控制器或 PCI 插槽。

20.16.15. 客户机虚拟机接口

字符设备将自身作为以下类型之一向虚拟客户机呈现：

要设置并行端口，请使用管理工具对域 XML 进行以下更改

图 20.60. 客户机虚拟机接口 Parallel 端口

```
...
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2'/>
    <target port='0'/>
  </parallel>
</devices>
...
```

<目标> 可以有一个 port 属性，用于指定端口号。端口从 0 开始。通常有 0、1 或 2 个并行端口。

要设置串行端口，请使用管理工具对域 XML 进行以下更改：

图 20.61. 客户机虚拟机接口串口

```
...
<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。通常有 0、1 或 2 串行端口。还有一个可选的 type 属性，它的值有两个选择，一个是 a isa-serial，另一个是 usb-serial。如果缺少 type，则将默认使用 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的控制器，如上面所述。

<console> 元素用于表示交互式控制台。根据所使用的客户机虚拟机类型，控制台可能是半虚拟化设备，或者根据以下规则，这些控制台可能是串行设备的克隆：

- *如果没有设置 targetType 属性，则默认 设备类型 取决于虚拟机监控程序的规则。当重新查询到 libvirt 中的 XML 时，将添加默认类型。对于完全虚拟化的 guest 虚拟机，默认设备类型通常是串行端口。*
- *如果 targetType 属性是 serial 的，如果不存在 <serial> 元素，则 console 元素将复制到 <serial> 元素中。如果 <serial> 元素已存在，则 console 元素将被忽略。*
- *如果 targetType 属性不是 串行 的，它将被正常处理。*
- *只有第一个 <控制台> 元素可以使用 serial 的 targetType。辅助控制台必须全部为泛虚拟化。*
- *在 s390 中，控制台元素可以使用 sclp 或 sclplm（在线模式）的 targetType。SCLP 是 s390 的原生控制台类型。SCLP 控制台没有关联控制器。*

在以下示例中，guest 虚拟机中会公开一个 virtio 控制台设备，作为 /dev/hvc[0-7]（详情请参阅 http://fedoraproject.org/wiki/Features/VirtioSerial）：

图 20.62. 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>
...
```

如果控制台以串行端口显示，则 <目标> 元素具有与串行端口相同的属性。通常只有一个控制台。

20.16.16. Channel

这代表了主机物理计算机与客户机虚拟机之间的专用通信通道，并通过使用管理工具更改 guest 虚拟机来操作，从而对域 xml 的以下部分所做的更改

图 20.63. 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/qemu/f16x86_64.agent'/>
    <target type='virtio' name='org.qemu.guest_agent.0'/>
  </channel>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0'/>
  </channel>
</devices>
...
```

这可以以各种方式实施。<目标> 元素的 type 属性中提供了特定类型的 <频道>。不同的频道类型有不同的目标属性，如下所示：

- guestfwd - 诊断客户机虚拟机发送到给定 IP 地址和端口的 TCP 流量，将转发到主机物理机器上的通道设备。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* 下的虚拟客户机中公开，如果指定了可选元素 名称，/dev/virtio-ports/$name （如需更多信息，请参阅 http://fedoraproject.org/wiki/Features/VirtioSerial）。可选的元素 地址 可将频道绑定到特定的 type='virtio-serial' 控制器，如上面所述。使用 QEMU 时，如果名称为 "org.qemu.guest_agent.0"，libvirt 可以与 guest 虚拟机安装的虚拟机代理交互，以获得 guest 虚拟机关闭或文件系统等操作。

- spicevmc - 半虚拟化 SPICE 频道.域还必须将 SPICE 服务器作为图形设备，在该设备指向主机物理机器 piggy-backs 信息。必须存在 target 元素，其中属性 type='virtio'；可选属性名称控制客户机虚拟机有权访问该通道的方式，默认为 name ='com.redhat.spice.0'。可选的 <address> 元素可将频道绑定到特定 type='virtio-serial' 控制器。

20.16.17. 主机物理机器接口

字符设备作为以下类型之一向主机物理机器表示自己：

表 20.23. 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 logfile	`<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将写入该文件。	`<devices>`
		`<serial type="file">`
		`<source path="/var/log/vm/vm-serial.log"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

虚拟控制台	将字符设备连接到虚拟控制台中的图形帧缓冲器。这通常通过特殊的热键序列访问，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.未向输入提供数据。所有写入的数据都会被丢弃。	`<devices>`
		`<serial type='null'>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

参数	描述	XML 片断
伪 TTY	使用 **/dev/ptmx** 分配 Pseudo TTY。**virsh 控制台** 等适合的客户端可以连接本地的串行端口交互。	<devices> <serial type="pty"> <source path="/dev/pts/3"/> <target port="1"/> </serial> </devices>
NB 特殊问题单	NB 特殊情况（**<如果控制台 type='pty'>**），则 TTY 路径也作为顶级 **<控制台>** 标签上的 **tty='/dev/pts/3'** 属性重复。这为 **<console>** 标签提供了与现有语法兼容。	
主机物理机器设备代理	字符设备通过 传递给底层物理字符设备。设备类型必须匹配，将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	字符设备会将输出写入命名管道。如需更多信息，请参阅 pipe(7)手册页。	<devices> <serial type="pipe"> <source path="/tmp/mypipe"/> <target port="1"/> </serial> </devices>
TCP 客户端/服务器	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"/>

参数	描述	XML 片断
		<target port="1"/> </serial> </devices> 或 作为等待客户端连接的 TCP 服务器。 <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 替代原始 TCP。另外，您还可以使用 telnets（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。这是一个丢失的服务。	<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，接受来自本地客户端的连接。	<devices> <serial type="unix"> <source mode="bind" path="/tmp/foo"/> <target port="1"/> </serial> </devices>

20.17. 声音设备

可以使用声音元素将虚拟声卡附加到主机物理机器。

图 20.64. 虚拟声卡

```
...
<devices>
  <sound model='es1370'/>
</devices>
...
```

sound 元素具有一个强制属性 model，它指定了模拟的实际声设备。有效值特定于底层虚拟机监控程序，但典型的选择是 'es1370'、'sb16'、'ac97' 和 'ich6'。此外，带有 ich6 模型的声音元素可以具有可选的子元素代码c，以将各种音频代码c附加到音频设备。如果没有指定，则会附加默认的 codec 来允许回放和记录。有效值为 'duplex'（反对行和换行符）和 "micro"（退出发言人和微手机）。

图 20.65. 声音设备

```
...
<devices>
  <sound model='ich6'>
    <codec type='micro'/>
  <sound/>
</devices>
...
```

每个声音元素具有一个可选的子元素 <地址>，可将设备绑定到特定的 PCI 插槽，如上方所述。

20.18. WATCHDOG 设备

可以通过 watchdog 元素将虚拟硬件 <watchdog> 设备添加到客户端虚拟机。watchdog 设备在客户机虚拟机中需要额外的驱动程序和管理守护进程。就像在 libvirt 配置中启用 watchdog 时，对其自身都非常有用。目前，当 watchdog 触发时没有支持通知。

图 20.66. watchdog 设备

```
...
<devices>
  <watchdog model='i6300esb'/>
</devices>
...

...
<devices>
  <watchdog model='i6300esb' action='poweroff'/>
</devices>
</domain>
```

此 XML 中声明了以下属性：

- Model - 所需的 model 属性指定模拟的实际 watchdog 设备。有效值特定于底层的虚拟机监控程序。
- model 属性可能会使用以下值：
  - i6300esb - 建议设备，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。有效值特定于底层的虚拟机监控程序。action 属性可以具有以下值：
  - reset - 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 - 强制关闭客户端虚拟机
  - pause - 暂停 guest 虚拟机
  - 无 - 不执行任何操作
  - 转储 - 自动转储客户机虚拟机。

请注意，"shutdown"操作要求 guest 虚拟机响应 ACPI 信号。在 watchdog 已过期的情况下，guest 虚拟机通常无法响应 ACPI 信号。因此，不建议使用 'shutdown'。另外，可通过在 /etc/libvirt/qemu.conf 文件中的 auto_dump_path 来配置转储文件的目录。

20.19. 内存 BALLOON 设备

虚拟内存 Balloon 设备添加到所有 Xen 和 KVM/QEMU 客户机虚拟机。它将显示为 <memballoon> 元素。它会在适当的时自动添加，因此不需要在客户机虚拟机 XML 中显式添加该元素，除非需要分配特定的 PCI 插槽。请注意，如果需要明确禁用 memballoon 设备，则可以使用 model='none'。

以下示例使用 KVM 自动添加设备

图 20.67. 内存 balloon 设备

```
...
<devices>
  <memballoon model='virtio'/>
</devices>
...
```

以下是使用静态 PCI 插槽 2 手动添加设备的示例

图 20.68. 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>
</domain>
```

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有： 'virtio'，这是使用 KVM 管理程序或 "xen"（使用 Xen 管理程序的默认设置）的默认设置。

20.20. 安全标签

<seclabel> 元素允许对安全驱动程序的操作进行控制。操作的基本模式是 'dynamic'，其中 libvirt 会自动生成唯一安全标签，即应用程序/管理员选择标签，或 'none' 被禁用。使用动态标签生成时，libvirt 始终会自动重新标记与虚拟机关联的任何资源。默认情况下，使用静态标签分配时，管理员或应用必须确保在任何资源上正确设置标签。但是，如果需要，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，每个驱动程序都有一个，并且每个标签引用的安全驱动程序可以使用属性 模型 Valid input XML 配置来定义：

图 20.69. 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'/>
```

如果在输入 XML 中不提供 "type" 属性，则将使用安全驱动程序默认设置，该设置可以是 "none" 或 "dynamic"。如果设置了 <baselabel>，但没有设置 'type'，则类型被假定为 'dynamic'。当查看正在运行的客户机虚拟机的 XML 时，将包括自动资源重新标记活跃状态的 XML 元素（一个 XML 元素 imagelabel ）。这是仅限输出的元素，在用户提供的 XML 文档中会忽略它。

可使用以下值处理以下元素：

- *Type - Either static,dynamic 或 none 来确定 libvirt 是否自动生成唯一的安全标签。*
- *Model - 有效的安全模型名称，与当前激活的安全模型匹配*
- *重新标记 - Either yes 或 no.如果使用了动态标签分配，则必须始终是 yes。如果分配静态标签，它将默认为 无。*
- *<label> - 如果使用静态标记，则必须指定要分配给虚拟域的完整安全标签。内容格式取决于使用中的安全驱动程序：*
  - *SELinux ：SELinux 上下文.*
  - *AppArmor ：AppArmor 配置集。*

- DAC ：所有者和组，以冒号分隔。它们可以定义为用户/组名称或 uid/gid。驱动程序首先会尝试将这些值作为名称解析，但前导加号用于强制驱动程序将它们解析为 uid 或 gid。

- <baselabel> - 如果使用动态标记，则可以选择性地指定基础安全标签。内容的格式取决于使用中的安全驱动程序。

- <imagelabel> - 这仅输出元素，显示与虚拟域关联的资源所使用的安全标签。内容格式取决于使用何时重新标记时的安全驱动程序，或者对特定源文件名进行的标签进行微调，方法是禁用标签（如果文件在 NFS 上实时或其他文件系统（如果缺少安全标签）或请求备用标签（在管理应用程序创建特殊标签时，不允许使用）来微调标签。当 seclabel 元素附加到特定路径而不是顶层域分配时，只支持属性重新标记或子元素标签。

20.21. 域 XML 配置示例

AMD64 和 Intel 上的 QEMU 模拟客户机虚拟机

图 20.70. 域 XML 配置示例

```
<domain type='qemu'>
  <name>QEmu-fedora-i686</name>
  <uuid>c7a5fdbd-cdaf-9455-926a-d65c16db1809</uuid>
  <memory>219200</memory>
  <currentMemory>219200</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='cdrom'>
      <source file='/home/user/boot.iso'/>
      <target dev='hdc'/>
      <readonly/>
    </disk>
    <disk type='file' device='disk'>
      <source file='/home/user/fedora.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <graphics type='vnc' port='-1'/>
  </devices>
</domain>
```

KVM 硬件加速 i686 上的客户机虚拟机

图 20.71. 域 XML 配置示例

```
<domain type='kvm'>
  <name>demo2</name>
  <uuid>4dea24b3-1d52-d8f3-2516-782e98a23fa0</uuid>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
  </os>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/demo2.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <mac address='24:42:53:21:52:45'/>
    </interface>
    <graphics type='vnc' port='-1' keymap='de'/>
  </devices>
</domain>
```

第 21 章 故障排除

本章论述了 Red Hat Enterprise Linux 6 虚拟化问题的常见问题和解决方案。

阅读本章，以开发与虚拟化技术相关的一些常见问题。故障排除需要从一本书学习困难的实践和体验。建议您在 Red Hat Enterprise Linux 6 上试验和测试虚拟化，以培养您的故障排除技能。

如果您无法找到本文档中的答案，则可以从虚拟化社区在线获得答案。有关 Linux 虚拟化网站列表，请参阅 第 B.1 节 “在线资源”。

21.1. 调试和故障排除工具

本节总结了系统管理员应用程序、网络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故障排除：

- kvm_stat - 参考 第 21.4 节 “kvm_stat”
- trace-cmd
- ftrace 请参考 Red Hat Enterprise Linux 开发者指南
- vmstat
- iostat
- lsof
- systemtap
- crash

- ***sysrq***

- ***sysrq t***

- ***sysrq w***

这些网络工具可以协助对虚拟化网络进行故障排除：

- ***ifconfig***

- ***tcpdump***

  tcpdump 命令的网络数据包。tcpdump 在查找网络异常和网络身份验证问题时很有用。有一个名为 wireshark 的 tcpdump 的图形版本。

- ***brctl***

  brctl 是检查并配置 Linux 内核中的以太网网桥配置的联网工具。在执行这些示例命令前，您必须有 root 访问权限：

```
# brctl show
bridge-name    bridge-id          STP  enabled  interfaces
-----------------------------------------------------------------------------
virtbr0             8000.feffffff       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
```

下面列出了用于对虚拟化进行故障排除的一些其他有用命令。

- strace 是一个跟踪系统调用和被另一个进程使用的事件的命令。
- vncviewer ：连接到在您的服务器或虚拟机上运行的 VNC 服务器。使用 yum install tigervnc 命令安装 vncviwer。
- vncserver ：在您的服务器上启动远程桌面。让您可以通过远程会话运行图形用户界面，如 virt-manager。使用 yum install tigervnc-server 命令安装 vncserver。

21.2. 准备灾难恢复

如果可能，最好在设备因天气或其他原因而遭入侵的情况。强烈建议您在主机物理机器上执行以下文件和目录备份：

- 从 /etc/libvirt 目录中，所有文件。
- 从 /var/lib/libvirt 目录中备份以下项目：
  - 在 /var/lib/libvirt/dnsmasq 中找到的当前 dnsmasq DHCP 租期
  - 在 /var/lib/libvirt/network 中找到运行的虚拟网络配置文件
  - 由 virt-manager 创建的虚拟机文件在保存客户机的当前状态时（如果存在）。这些可以在 /var/lib/libvirt/qemu/save/ 目录中找到。如果使用 virsh save 命令 创建虚拟机，则可以在为用户指定 virsh save 的位置找到这些文件。
  - 由 qemu-img 创建和 virsh snapshot-create 命令创建的 guest 虚拟机快照文件，并在用户为命令指定的位置找到。

- virt-manager 创建的 guest 虚拟机磁盘镜像（如果有），可在 /var/lib/libvirt/images/ 目录中找到。如果使用 virsh pool-define 命令创建虚拟存储，则镜像文件可在为 virsh pool-define 指定的位置中找到。有关如何备份客户端镜像文件的步骤，请使用 过程 21.1, “为灾难恢复目的创建客户机虚拟机的磁盘镜像备份” 中介绍的步骤。

- 如果使用网桥，您还需要备份位于 /etc/sysconfig/network-scripts/ifcfg-<bridge_name> 中的文件

- 另外，还可备份在 /var/lib/libvirt/qemu/dump 中找到的客户机虚拟机核心转储文件，用于分析故障的原因。但请注意，这些文件对于某些系统来说可能非常大。

过程 21.1. 为灾难恢复目的创建客户机虚拟机的磁盘镜像备份

此流程将介绍如何备份多个不同的磁盘镜像类型。

1. 要仅备份客户机虚拟机磁盘镜像，请备份位于 /var/lib/libvirt/images 中的文件。要使用 LVM 逻辑卷备份客户端虚拟机磁盘镜像，请运行以下命令：

```
# lvcreate --snapshot --name snap --size 8G /dev/vg0/data
```

这个命令会创建一个名为 snap 的快照卷，大小为 8G，作为 64G 卷的一部分。

2. 使用类似此快照的命令为快照创建一个文件：

```
# mkdir /mnt/virt.snapshot
```

3. 使用以下命令挂载您创建的和快照卷的目录：

```
# mount /dev/vg0/snap /mnt/virt.snapshot
```

4. 使用以下命令之一备份卷：

a.
```
# tar -pzc -f /mnt/backup/virt-snapshot-MM-DD-YYYY.tgz
/mnt/virt.snapshot++++++++++++
```

b.
```
# rsync -a /mnt/virt.snapshot/ /mnt/backup/virt-snapshot.MM-DD-YYYY/
```

-

21.3. 创建 VIRSH DUMP 文件

执行 virsh dump 命令将客户端虚拟机的核心转储到文件，以便可以诊断虚拟机中的错误。运行此命令可能需要您手动确保文件和通过参数 corefilepath 指定的路径的正确权限。virsh 转储 命令与 coredump（或 crash 实用程序）类似。要创建 virsh dump 文件，请运行：

```
#virsh dump <domain> <corefilepath> [--bypass-cache] { [--live] | [--crash] | [--reset] } [--verbose] [--memory-only]
```

尽管域（guest 虚拟机域名）和 corefilepath（新创建的内核转储文件的位置）是必需的，以下参数是可选的：

- --live 在运行的机器上创建转储文件，且不会暂停它。
- --crash 会停止客户机虚拟机并生成转储文件。主要区别在于，guest 虚拟机不会被列为 Stopped，其原因为 Crashed。请注意，在 virt-manager 中，其状态将显示为 Paused。
- --reset 将在成功转储后重置 guest 虚拟机。请注意，这三个交换机是相互排斥的。
- --bypass-cache 使用 O_DIRECT 绕过文件系统缓存。
- -- 仅 内存转储文件将保存为 elf 文件，并且仅包括域内存和 cpu 常见的寄存器值。如果域直接使用主机设备，此选项非常有用。
- --verbose 显示转储的进度

整个转储过程可以使用 virsh domjobinfo 监控，并可通过运行 virsh domjobabort 来取消。

21.4. KVM_STAT

kvm_stat 命令是一个 python 脚本，该脚本从 kvm 内核模块中检索运行时统计信息。kvm_stat 命令可用于诊断对 kvm 可见的 guest 行为。特别是，与客户机相关的性能相关问题。目前，所报告的统计数据适用于整个系统；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。

kvm_stat 命令要求已加载 kvm 内核模块并挂载 debugfs。如果没有启用这些功能，命令会输出启用 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 Output

kvm_stat 命令输出所有虚拟客户机和主机的统计数据。在该命令终止前（使用 Ctrl+c 或 q 键）会更新输出。

```
# kvm_stat

kvm statistics

efer_reload               94       0
exits                  4003074   31272
fpu_reload             1313881   10796
halt_exits               14050     259
halt_wakeup               4496     203
host_state_reload      1638354   24893
hypercalls                   0       0
insn_emulation         1093850    1909
insn_emulation_fail          0       0
invlpg                   75569       0
io_exits                1596984   24509
irq_exits                21013     363
irq_injections           48039    1222
irq_window               24656     870
largepages                   0       0
mmio_exits               11873       0
mmu_cache_miss           42565       8
mmu_flooded              14752       0
mmu_pde_zapped           58730       0
mmu_pte_updated              6       0
mmu_pte_write           138795       0
mmu_recycled                 0       0
mmu_shadow_zapped        40358       0
mmu_unsync                 793       0
nmi_injections               0       0
nmi_window                   0       0
pf_fixed                697731    3150
pf_guest                279349       0
remote_tlb_flush             5       0
```

```
request_irq               0      0
signal_exits              1      0
tlb_flush            200190      0
```

变量说明：

efer_reload

扩展功能启用注册(EFER)的数量将重新加载。

退出

所有 VMEXIT 调用的数量。

fpu_reload

VMENTRY 重新加载 FPU 状态的次数。当客户机使用浮动点单元(FPU)时，fpu_reload 会被递增。

halt_exits

由于调用 停止，客户机数量会退出。在客户机闲置时，通常会看到这种退出。

halt_wakeup

暂停 中唤醒的数量。

host_state_reload

主机状态的完整重新加载计数（当前为 MSR 设置和客户机 MSR 读取）。

hypercalls

虚拟机管理程序服务调用的数量。

insn_emulation

主机模拟的客户机说明数。

insn_emulation_fail

预告尝试 中失败数。

io_exits

从 I/O 端口访问中退出 guest 的数量。

irq_exits

由于外部中断而退出的客户机数量。

irq_injections

发送到客户机的中断数。

irq_window

客户机数量从未完成的中断窗口退出。

largepages

当前正在使用的大页面数。

mmio_exits

由于内存映射 I/O(MMIO)访问，客户机数量会退出。

mmu_cache_miss

创建 KVM MMU 影子页面的数量。

mmu_flooded

在 MMU 页面中检测过多的写操作数。这统计检测到的写入操作不受单个写入操作的影响。

mmu_pde_zapped

页面目录条目(PDE)破坏性操作数量。

mmu_pte_updated

页面表条目(PTE)破坏性操作的数量。

mmu_pte_write

客户机页表条目(PTE)写入操作的数量。

mmu_recycled

可重新声明的影子页面数。

mmu_shadow_zapped

影子页面的数量。

mmu_unsync

尚未链接的非同步页面数量。

nmi_injections

将不可屏蔽中断(NMI)注入给客户机的数量。

nmi_window

客户机数量从（否）不可屏蔽中断(NMI)窗口退出。

pf_fixed

固定数量（非过期）页表条目(PTE)映射。

pf_guest

注入到客户机中的页面错误数。

remote_tlb_flush

远程（同级 CPU）翻译缓冲(TLB)刷新请求的数量。

request_irq

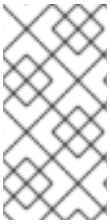
客户机中断窗口请求将退出。

signal_exits

由于主机中待处理的信号，guest 数量会退出。

tlb_flush

管理程序执行的 tlb_flush 操作数量。



注意

kvm_stat 命令的输出信息由 KVM 管理程序导出，作为位于 /sys/kernel/debug/kvm/ 目录中的伪文件。

21.5. GUEST VIRTUAL MACHINE FAILS TO SHUTDOWN

通常，执行 virsh shutdown 命令会导致发送电源按钮 ACPI 事件，因此当有人在物理机上按下电源按钮时复制同样的操作。在每个物理计算机中，操作系统可以处理此事件。在过去的操作系统中，只会静默关闭。今天，最常见的操作是显示询问应做什么的对话框。有些操作系统甚至可以完全忽略此事件，特别是在没有用户登录的情况下。当在客户端虚拟机中安装此类操作系统时，运行 virsh shutdown 无法正常工作（它会被忽略，或者在虚拟机显示对话框）。但是，如果将 qemu-guest-agent 频道添加到客户机虚拟机，并且此代理正在客户端虚拟机操作系统中运行，则 virsh shutdown 命令将请求代理关闭客户端操作系统而不是发送 ACPI 事件。代理将从客户机虚拟机操作系统内部调用关机，一切均按预期运行。

过程 21.2. 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域 XML 并添加以下片断：

图 21.1. 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] 来启动 guest 虚拟机。

4. 在客户机虚拟机上安装 qemu-guest-agent (yum install qemu-guest-agent)，并在每次引导时都作为服务(qemu-guest-agent.service)自动运行。如需更多信息，请参阅 第 10 章 qemu-img 和 QEMU 客户机代理。

21.6. 使用 SERIAL CONSOLE 进行故障排除

Linux 内核可以将信息输出为串行端口。这可用于调试使用视频设备或无外设服务器的内核 panic 和硬件问题。本节中的小节介绍了使用 KVM 管理程序为主机物理机器设置串行控制台输出。

本节介绍如何为完全虚拟化的虚拟机启用串口控制台输出。

可使用 virsh console 命令查看完全虚拟化的 guest 串行控制台输出。

请注意，完全虚拟化的客户机串行控制台存在一些限制。存在限制包括：

- 输出数据可能会被丢弃或有缺陷。

在 Linux 或 Windows 上的 COM1 上，串行端口称为 ttyS0。

您必须将虚拟化操作系统配置为将信息输出到虚拟串行端口。

要将完全虚拟化 Linux 客户机的内核信息输出到域，请修改 /boot/grub/grub.conf 文件。在 kernel 行中附加以下内容： console=tty0 console=ttyS0,115200

```
title Red Hat Enterprise Linux Server (2.6.32-36.x86-64)
 root (hd0,0)
 kernel /vmlinuz-2.6.32-36.x86-64 ro root=/dev/volgroup00/logvol00 \
 console=tty0 console=ttyS0,115200
 initrd /initrd-2.6.32-36.x86-64.img
```

重启客户机。

在主机上，使用以下命令访问串行控制台：

```
# virsh console
```

您还可以使用 virt-manager 显示虚拟文本控制台。在 guest 控制台窗口中，从 View 菜单选择 Serial 1 in Text Consoles。

21.7. 虚拟化日志文件

- 每个完全虚拟化的虚拟机日志位于 /var/log/libvirt/qemu/ 目录中。每个 guest 日志命名为 GuestName.log，当达到大小限制时，将定期压缩。

如果您在 Virtual Machine Manager 遇到任何错误，您可以查看位于 $HOME/.virt-manager 目录中的 virt-manager.log 文件中生成的数据。

21.8. LOOP 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 个活跃循环设备。如果需要超过八个文件的虚拟机或循环设备，可以在 /etc/modprobe.d/目录中调整配置循环设备的数量。添加以下行：

```
options loop max_loop=64
```

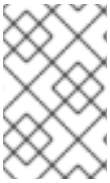
这个示例使用 64，但您可以指定另一个数字来设置最大循环值。您可能还必须在系统中实施 loop 设备支持的客户端。要将支持循环设备的虚拟机用于完全虚拟化系统，请使用 phy: device 或 file: file 命令。

21.9. 实时迁移错误

有些情况下，实时迁移会导致内存内容被重新转移。这个过程会导致客户机处于持续写入内存的状态，因此会降低迁移速度。如果应该发生这种情况，并且客户机每秒写入超过十倍的 MB，则实时迁移可能无法完成（聚合）。这个问题尚未计划在 Red Hat Enterprise Linux 6 中解决，并计划在 Red Hat Enterprise Linux 7 中解决。

当前实时迁移的实现会将默认迁移时间配置为 30ms。这个值决定了迁移结束时 guest 暂停时间，以便传输左侧。更高数值会增加实时迁移的奇数

21.10. 在 BIOS 中启用 INTEL VT-X 和 AMD-V 虚拟化硬件扩展



注意

要扩展您的专业知识，您可能还对红帽 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。

Intel VT-x 扩展可以在 BIOS 中被禁用。某些笔记本电脑供应商已在其 CPU 中默认禁用 Intel VT-x 扩展。

在 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，请参考以下部分。

验证在 BIOS 中启用了虚拟化扩展。Intel VT 或 AMD-V 的 BIOS 设置通常在 Chipset 或 Processor 菜单中。菜单名称可能因本指南而异，虚拟化扩展设置可在 Security Settings 或其他非标准菜单名称中找到。

过程 21.3. 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。通常可以通过按 delete 键、F1 键或 Alt 和 F4 键（取决于系统）来完成。

2. 在 BIOS 中启用虚拟化扩展

注意

以下许多步骤可能因您的主板、处理器类型、芯片组和 OEM 而异。有关配置系统的详情，请参考您的系统附带文档。

a.
打开处理器子菜单，处理器设置菜单可以在 Chipset、高级 CPU 配置 或 北bridge 中隐藏。

b.
启用 Intel 虚拟化技术 （也称为 Intel VT-x）。AMD-V 扩展无法在 BIOS 中被禁用，且应该已经启用。虚拟化扩展可能被标记为 虚拟化扩展、Vanderpool 或各种其他名称，具体取决于 OEM 和系统 BIOS。

c.
如果选项可用，请启用 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。

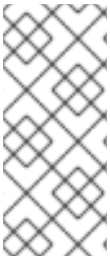
4.
机器引导时，运行 cat /proc/cpuinfo |grep -E "vmx|svm"。指定 --color 是可选的，但如果想突出显示搜索词，则很有用。如果命令输出，则虚拟化扩展现已启用。如果没有输出结果，您的系统可能没有启用虚拟化扩展或启用了正确的 BIOS 设置。

21.11. KVM 网络性能

默认情况下，会为 KVM 虚拟机分配虚拟 Realtek 8139(rtl8139)NIC（网络接口控制器）。Red Hat Enterprise Linux 虚拟机默认分配了 virtio NIC，但未指定 Windows 客户机或客户机类型。

rtl8139 虚拟化 NIC 在大多数环境中可以正常工作，但该设备可能会遭遇一些网络的性能降级问题，如 10 千兆位以太网。

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)驱动程序也作为仿真驱动程序选择提供支持。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 21.4. 切换到 virtio 驱动程序

1. 关闭客户端操作系统。

2. 使用 virsh 命令编辑客户端的配置文件（其中 GUEST 是客户端的名称）：

```
# virsh edit GUEST
```

virsh edit 命令使用 $EDITOR shell 变量来确定要使用哪个编辑器。

3. 查找配置的网络接口部分。本节类似以下代码片段：

```
<interface type='network'>
  [output truncated]
  <model type='rtl8139' />
</interface>
```

4. 将 model 元素的 type 属性从 'rtl8139' 改为 'virtio'。这会将 rtl8139 驱动程序改为 e1000 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机

或者，也可以使用不同的网络驱动程序创建新 guest。如果您在通过网络连接安装客户机时遇到问题，则可能需要这样做。此方法要求您至少有一个虚拟机已创建（可能从 CD 或者 DVD 安装）以用作模板。

1. 从现有 guest（在这个示例中，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 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模型行：

```
<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
```

21.12. 使用 LIBVIRT 创建外部快照的临时解决方案

QEMU guest 有两种快照。内部快照完全包含在 qcow2 文件中，由 libvirt 完全支持，允许创建、删除和恢复快照。这是在创建快照时 libvirt 使用的默认设置，特别是未指定选项时。虽然此文件类型比创建快照中的其他人要长，但 libvirt 需要它才能使用 qcow2 磁盘。此文件类型的另一个缺点是 qcow2 磁盘不可能从 QEMU 接收改进。

另一方面，外部快照可以在不停机的情况下获取任何类型的原始磁盘镜像，并可从 QEMU 接收主动改进。在 libvirt 中，当使用 **--disk-only** 选项作为 **snapshot-create -as**（或者在为快照创建时指定显式 XML 文件时）创建它们。目前，外部快照是一个单向操作，因为 libvirt 可以创建它们，但无法对快照做进一步操作。

21.13. 客户机控制台中缺少带有日语键盘的字符

在 Red Hat Enterprise Linux 6 主机上，在本地连接日语键盘可能会导致键入的字符，如下划线（_ 字符）在客户机控制台中无法正确显示。这是因为默认没有正确设置所需的 keymap。

借助 Red Hat Enterprise Linux 3 和 Red Hat Enterprise Linux 6 虚拟机，在按关联的密钥时通常不会产生任何错误消息。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 中解决这个问题，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 Keymap 下拉菜单中将 Auto 更改为 ja。**
- **点 应用 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 <target guest>**
- **将以下属性添加到 tag: {c> keymap='ja'。例如：**

  ```
  <graphics type='vnc' port='-1' autoport='yes' keymap='ja'/>
  ```

21.14. 验证虚拟化扩展

使用这个部分来确定您的系统是否有硬件虚拟化扩展。完全虚拟化需要虚拟化扩展（Intel VT-x 或 AMD-V）。

1. **运行以下命令来验证 CPU 虚拟化扩展是否可用：**

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

2. 分析 输出。

- 以下输出包含一个 vmx 条目，指明了 Intel VT-x 扩展带有 Intel 处理器：

```
flags   : fpu tsc msr pae mce cx8 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
```

- 以下输出包含一个 svm 条目，表示 AMD 处理器带有 AMD-V 扩展：

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

如果收到任何输出，则处理器具有硬件虚拟化扩展。然而，在一些情况下，制造商会在 BIOS 中禁用虚拟化扩展。

"标志："输出内容可能会多次出现，一次用于系统中的每个超线程、核心或 CPU。

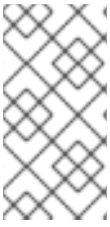
虚拟化扩展可以在 BIOS 中被禁用。如果扩展没有显示或者完全虚拟化无法正常工作 过程 21.3, “在 BIOS 中启用虚拟化扩展”。

3. 确保载入 KVM 子系统

作为额外的检查，验证 KVM 模块 是否在内核中被加载：

```
# lsmod | grep kvm
```

如果输出包含 kvm_intel 或 kvm_amd，则会加载 kvm 硬件虚拟化模块，且您的系统满足要求。



注意

如果安装了 libvirt 软件包，virsh 命令可输出虚拟化系统功能的完整列表。以 root 用户身份运行 virsh capabilities 以接收完整列表。

附录 A. 虚拟主机指标守护进程(VHOSTMD)

vhostmd （虚拟主机指标守护进程）允许虚拟机查看其运行所在主机的有限信息。这个守护进程只为 SAP 提供 Red Hat Enterprise Linux。

在主机中，守护进程(vhostmd)运行，它会定期将指标写入磁盘镜像中。此磁盘镜像以只读方式导出到客户机虚拟机。客户机虚拟机可以读取磁盘镜像以查看指标。简单同步会阻止客户机虚拟机查看日期或损坏指标。

系统管理员会选择哪些指标可用于每个虚拟客户机使用。另外，系统管理员可能会阻止一个或多个虚拟客户机访问指标配置。

希望使用 vhostmd 和 vm-dump-metrics 的客户需要 "RHEL for SAP Business Applications" 订阅，以便其运行 SAP 的 RHEL 系统订阅到客户门户网站或 Red Hat Subscription Management 的 "RHEL for SAP" 频道。客户门户网站中的以下知识库文章描述了 RHEL 中的 vhostmd 设置：
https://access.redhat.com/knowledge/solutions/41566

附录 B. 其它资源

要了解有关虚拟化和 Red Hat Enterprise Linux 的更多信息，请参阅以下资源。

B.1. 在线资源

- http://www.libvirt.org/ 是 libvirt 虚拟化 API 的官方网站。
- https://virt-manager.org/ 是 虚拟机管理器 (virt-manager)的项目网站，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

B.2. 安装的文档

- man virsh 和 /usr/share/doc/libvirt- <version-number > - 包含 virsh virtual machine management utility 的子命令和选项，以及有关 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 虚拟化管理库的接口的程序。
- /usr/share/doc/python-virtinst-<version-number > - 提供 virt-install 命令的文档，以帮助开始安装 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。

附录 C. 修订历史记录

修订 1-502 6.9 GA 发行版本的更新	Mon Mar 08 2017	Jiri Herrmann
修订 1-501 6.8 GA 发行版本的更新	Mon May 02 2016	Jiri Herrmann
修订 1-500 6.8 beta 版发行版本的多个更新	Thu Mar 01 2016	Jiri Herrmann
修订 1-449 清理修订历史	Thu Oct 08 2015	Jiri Herrmann
修订 1-447 6.7 GA 发行版本的更新。	Fri Jul 10 2015	Dayle Parker