Red Hat Enterprise Linux 6

Virtualization Administration Guide

Virtualization Documentation

Red Hat Enterprise Linux 6 Virtualization Administration Guide
Virtualization Documentation
Edition 0.1

Author

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

The Virtualization Administration Guide covers administration of hosts, networking, storage, device and guest management, and troubleshooting.

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click

[1] https://fedorahosted.org/liberation-fonts/

> **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;
```

```
public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at *http://access.redhat.com*. Through the customer portal, you can:

- search or browse through a knowledgebase of technical support articles about Red Hat products.
- submit a support case to Red Hat Global Support Services (GSS).

- access other product documentation.

Red Hat also hosts a large number of electronic mailing lists for discussion of Red Hat software and technology. You can find a list of publicly available mailing lists at *https://www.redhat.com/mailman/listinfo*. Click on the name of any mailing list to subscribe to that list or to access the list archives.

2.2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/* against the product **Red Hat Enterprise Linux 6.**

When submitting a bug report, be sure to mention the manual's identifier: *doc-Virtualization_Administration_Guide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Server best practices

The following tasks and tips can assist you with securing and ensuring reliability of your Red Hat Enterprise Linux host.

- Run SELinux in enforcing mode. Set SELinux to run in enforcing mode with the **setenforce** command.

```
# setenforce 1
```

- Remove or disable any unnecessary services such as **AutoFS**, **NFS**, **FTP**, **HTTP**, **NIS**, **telnetd**, **sendmail** and so on.
- Only add the minimum number of user accounts needed for platform management on the server and remove unnecessary user accounts.
- Avoid running any unessential applications on your host. Running applications on the host may impact virtual machine performance and can affect server stability. Any application which may crash the server will also cause all virtual machines on the server to go down.
- Use a central location for virtual machine installations and images. Virtual machine images should be stored under **/var/lib/libvirt/images/**. If you are using a different directory for your virtual machine images make sure you add the directory to your SELinux policy and relabel it before starting the installation. Use of shareable, network storage for a central location is highly recommended.

Security for virtualization

When deploying virtualization technologies, you must ensure that the host cannot be compromised. The host is a Red Hat Enterprise Linux system that manages the system, devices, memory and networks as well as all virtualized guests. If the host is insecure, all guests in the system are vulnerable. There are several ways to enhance security on systems using virtualization. You or your organization should create a *Deployment Plan* containing the operating specifications and specifies which services are needed on your virtualized guests and host servers as well as what support is required for these services. Here are a few security issues to consider while developing a deployment plan:

- Run only necessary services on hosts. The fewer processes and services running on the host, the higher the level of security and performance.
- Enable SELinux on the hypervisor. Read *Section 2.2, "SELinux and virtualization"* for more information on using SELinux and virtualization.
- Use a firewall to restrict traffic to the host. You can setup a firewall with default-reject rules that will help secure the host from attacks. It is also important to limit network-facing services.
- Do not allow normal users to access the host. The host is privileged, and granting access to unprivileged accounts may compromise the level of security.

2.1. Storage security issues

Administrators of virtualized guests can change the partitions the host boots in certain circumstances. To prevent this administrators should follow these recommendations:

The host should not use disk labels to identify file systems in the **fstab** file, the **initrd** file or used by the kernel command line. If less privileged users, especially virtualized guests, have write access to whole partitions or LVM volumes.

Guests should not be given write access to whole disks or block devices (for example, **/dev/sdb**). Use partitions (for example, **/dev/sdb1**) or LVM volumes.

2.2. SELinux and virtualization

Security Enhanced Linux was developed by the NSA with assistance from the Linux community to provide stronger security for Linux. SELinux limits an attacker's abilities and works to prevent many common security exploits such as buffer overflow attacks and privilege escalation. It is because of these benefits that all Red Hat Enterprise Linux systems should run with SELinux enabled and in enforcing mode.

Adding LVM based storage with SELinux in enforcing mode

The following section is an example of adding a logical volume to a virtualized guest with SELinux enabled. These instructions also work for hard drive partitions.

Procedure 2.1. Creating and mounting a logical volume on a virtualized guest with SELinux enabled

1. Create a logical volume. This example creates a 5 gigabyte logical volume named *NewVolumeName* on the volume group named *volumegroup*.

```
# lvcreate -n NewVolumeName -L 5G volumegroup
```

2. Format the *NewVolumeName* logical volume with a file system that supports extended attributes, such as ext3.

```
# mke2fs -j /dev/volumegroup/NewVolumeName
```

3. Create a new directory for mounting the new logical volume. This directory can be anywhere on your file system. It is advised not to put it in important system directories (**/etc**, **/var**, **/sys**) or in home directories (**/home** or **/root**). This example uses a directory called **/virtstorage**

```
# mkdir /virtstorage
```

4. Mount the logical volume.

```
# mount /dev/volumegroup/NewVolumeName /virtstorage
```

5. Set the correct SELinux type for the libvirt image location.

```
# semanage fcontext -a -t virt_image_t "/virtstorage(/.*)?"
```

If the targeted policy is used (targeted is the default policy) the command appends a line to the **/etc/selinux/targeted/contexts/files/file_contexts.local** file which makes the change persistent. The appended line may resemble this:

```
/virtstorage(/.*)?    system_u:object_r:virt_image_t:s0
```

6. Run the command to change the type of the mount point (**/virtstorage**) and all files under it to **virt_image_t** (the **restorecon** and **setfiles** commands read the files in **/etc/selinux/targeted/contexts/files/**).

```
# restorecon -R -v /virtstorage
```

Testing new attributes

Create a new file (using the **touch** command) on the file system.

```
# touch /virtstorage/newfile
```

Verify the file has been relabeled using the following command:

```
# sudo ls -Z /virtstorage
-rw-------. root root system_u:object_r:virt_image_t:s0 newfile
```

The output shows that the new file has the correct attribute, **virt_image_t**.

2.3. SELinux

This section contains topics to consider when using SELinux with your virtualization deployment. When you deploy system changes or add devices, you must update your SELinux policy accordingly.

To configure an LVM volume for a guest, you must modify the SELinux context for the respective underlying block device and volume group.

```
# semanage fcontext -a -t virt_image _t -f -b /dev/sda2
# restorecon /dev/sda2
```

KVM and SELinux

The following table shows the SELinux Booleans which affect KVM when launched by libvirt.

KVM SELinux Booleans

SELinux Boolean	Description
virt_use_comm	Allow virt to use serial/parallel communication ports.
virt_use_fusefs	Allow virt to read fuse files.
virt_use_nfs	Allow virt to manage NFS files.
virt_use_samba	Allow virt to manage CIFS files.
virt_use_sanlock	Allow sanlock to manage virt lib files.
virt_use_sysfs	Allow virt to manage device configuration (PCI).
virt_use_xserver	Allow virtual machine to interact with the xserver.
virt_use_usb	Allow virt to use USB devices.

2.4. Virtualization firewall information

Various ports are used for communication between virtualized guests and management utilities.

Guest network services

Any network service on a virtualized guest must have the applicable ports open on the guest to allow external access. If a network service on a guest is firewalled it will be inaccessible. Always verify the guests network configuration first.

- ICMP requests must be accepted. ICMP packets are used for network testing. You cannot ping guests if ICMP packets are blocked.
- Port 22 should be open for SSH access and the initial installation.
- Ports 80 or 443 (depending on the security settings on the RHEV Manager) are used by the vdsm-reg service to communicate information about the host.
- Ports 5634 to 6166 are used for guest console access with the SPICE protocol.
- Ports 49152 to 49216 are used for migrations with KVM. Migration may use any port in this range depending on the number of concurrent migrations occurring.
- Enabling IP forwarding (**`net.ipv4.ip_forward = 1`**) is also required for shared bridges and the default bridge. Note that installing libvirt enables this variable so it will be enabled when the virtualization packages are installed unless it was manually disabled.

Note

Note that enabling IP forwarding is **not** required for physical bridge devices. When a guest is connected through a physical bridge, traffic only operates at a level that does not require IP configuration such as IP forwarding.

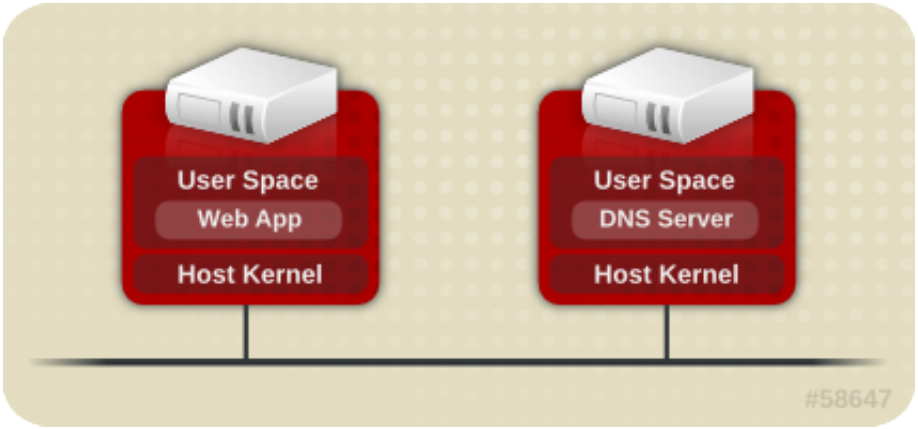
sVirt

sVirt is a technology included in Red Hat Enterprise Linux 6 that integrates SELinux and virtualization. sVirt applies Mandatory Access Control (MAC) to improve security when using virtualized guests. The main reasons for integrating these technologies are to improve security and harden the system against bugs in the hypervisor that might be used as an attack vector aimed toward the host or to another virtualized guest.

This chapter describes how sVirt integrates with virtualization technologies in Red Hat Enterprise Linux 6.

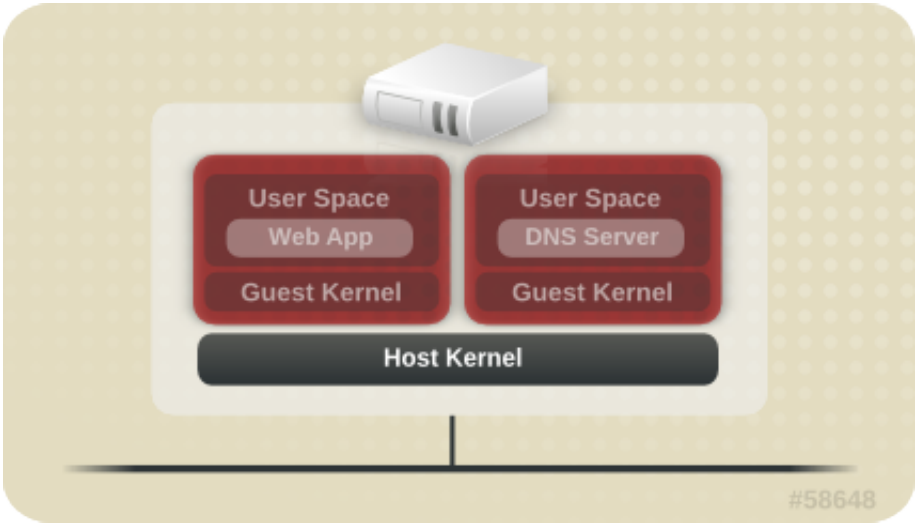
Non-virtualized environments

In a non-virtualized environment, hosts are separated from each other physically and each host has a self-contained environment, consisting of services such as a web server, or a DNS server. These services communicate directly to their own user space, host kernel and physical host, offering their services directly to the network. The following image represents a non-virtualized environment:



Virtualized environments

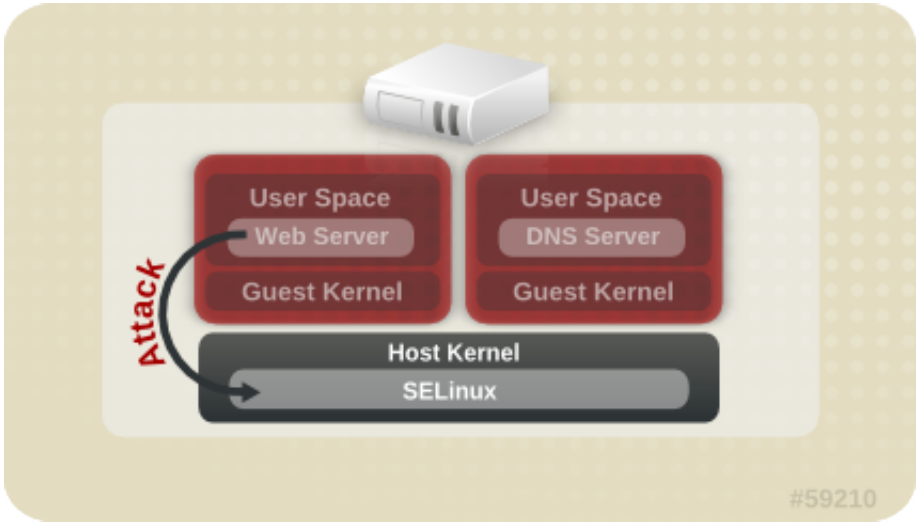
In a virtualized environment, several operating systems can run on a single host kernel and physical host. The following image represents a virtualized environment:



3.1. Security and Virtualization

When services are not virtualized, machines are physically separated. Any exploit is usually contained to the affected machine, with the obvious exception of network attacks. When services are grouped together in a virtualized environment, extra vulnerabilities emerge in the system. If there is a security flaw in the hypervisor that can be exploited by a guest instance, this guest may be able to not only attack the host, but also other guests running on that host. These attacks can extend beyond the guest instance and could expose other guests to attack.

sVirt is an effort to isolate guests and limit their ability to launch further attacks if exploited. This is demonstrated in the following image, where an attack can not break out of the virtualized guest and extend to another guest instance:



SELinux introduces a pluggable security framework for virtualized instances in its implementation of Mandatory Access Control (MAC). The sVirt framework allows guests and their resources to be uniquely labeled. Once labeled, rules can be applied which can reject access between different guests.

3.2. sVirt labeling

Like other services under the protection of SELinux, sVirt uses process-based mechanisms and restrictions to provide an extra layer of security over guest instances. Under typical use, you should not even notice that sVirt is working in the background. This section describes the labeling features of sVirt.

As shown in the following output, when using sVirt, each virtualized guest process is labeled and runs with a dynamically generated level. Each process is isolated from other VMs with different levels:

```
# ps -eZ | grep qemu

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

The actual disk images are automatically labeled to match the processes, as shown in the following output:

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

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

The following table outlines the different labels that can be assigned when using sVirt:

Table 3.1. sVirt labels

Type/Description	SELinux Context
Virtualized guest processes. MCS1 is a random MCS field. Approximately 500,000 labels are supported.	system_u:system_r:svirt_t:MCS1
Virtualized guest images. Only *svirt_t* processes with the same MCS fields can read/write these images.	system_u:object_r:svirt_image_t:MCS1
Virtualized guest shared read/write content. All *svirt_t* processes can write to the svirt_image_t:s0 files.	system_u:object_r:svirt_image_t:s0
Virtualized guest shared read only content. All svirt_t processes can read these files/devices.	system_u:object_r:svirt_content_t:s0
Virtualized guest images. Default label for when an image exits. No *svirt_t* virtual processes can read files/devices with this label.	system_u:object_r:virt_content_t:s0

It is also possible to perform static labeling when using sVirt. Static labels allow the administrator to select a specific label, including the MCS/MLS field, for a virtualized guest. Administrators who run statically-labeled virtualized guests are responsible for setting the correct label on the image files. The virtualized guest will always be started with that label, and the sVirt system will never modify the label of a statically-labeled virtual machine's content. This allows the sVirt component to run in an MLS environment. You can also run multiple virtualized guests with different sensitivity levels on a system, depending on your requirements.

KVM live migration

This chapter covers migrating guests running on a KVM hypervisor to another KVM host.

Migration describes the process of moving a guest from one host to another. This is possible because guests are running in a virtualized environment instead of directly on the hardware. Migration is useful for:

- Load balancing - guests can be moved to hosts with lower usage when their host becomes overloaded, or another host is under-utilized.
- Hardware independence - when we need to upgrade, add, or remove hardware devices on the host, we can safely relocate guests to other hosts. This means that guests do not experience any downtime for hardware improvements.
- Energy saving - guests can be redistributed to other hosts and host systems powered off to save energy and cut costs in low usage periods.
- Geographic migration - guests can be moved to another location for lower latency or in serious circumstances.

Migration works by sending the state of the guest's memory and any virtualized devices to a destination host. It requires networked storage to be shared between the source and destination hosts, so that guest storage can be omitted from the migration process.

Migrations can be performed live or offline.

An offline migration suspends the guest, then moves an image of the guest's memory to the destination host. The guest is then resumed on the destination host and the memory the guest used on the source host is freed.

The time an offline migration takes depends on network bandwidth and latency. If the network is experiencing heavy use or low bandwidth, the migration will take much longer.

In a live migration, the guest continues to run on the source host while its memory pages are transferred, in order, to the destination host. During migration, KVM monitors the source for any changes in pages it has already transferred, and begins to transfer these changes when all of the initial pages have been transferred. KVM also estimates transfer speed during migration, so when the remaining amount of data to transfer will take a certain configurable period of time (10ms by default), KVM stops the original guest, transfers the remaining data, and resumes the guest on the destination host.

If the original guest modifies pages faster than KVM can transfer them to the destination host, offline migration must be used, as live migration would never complete.

4.1. Live migration requirements

Migrating guests requires the following:

Migration requirements

- A guest installed on shared networked storage using one of the following protocols:
  - Fibre Channel-based LUNs.
  - iSCSI
  - NFS

- GFS2
- SCSI RDMA protocols (SCSI RCP), the block export protocol used in Infiniband and 10GbE iWARP adapters.

- Two or more Red Hat Enterprise Linux systems of the same version with the same updates.
- Both systems must have the appropriate network/IP ports open.
- Both systems must have identical network configurations. All bridging and network configurations must be exactly the same on both hosts.
- A separate system exporting the shared storage medium. Storage should not reside on either of the two hosts being used for migration.
- Shared storage must mount at the same location on source and destination systems. The mounted directory name must be identical.
- When migration is attempted on an existing guest in a public bridge+tap network, the source and destination hosts must be located in the same network. Otherwise, the guest network will not operate after migration.

Configuring network storage

Configure shared storage and install a guest on the shared storage.

Alternatively, use the NFS example in *Section 4.2, "Shared storage example: NFS for a simple migration"*

4.2. Shared storage example: NFS for a simple migration

Important

This example uses NFS to share guest images with other KVM hosts. This example is not practical for large installations, and is an example intended to demonstrate migration techniques only. Do not use this example for migrating or running more than a few guests.

iSCSI storage is a better choice for large deployments. Refer to *Section 11.1.5, "iSCSI-based storage pools"* for configuration details.

1. **Export your libvirt image directory**
   Migration requires storage to reside on a system that is separate to the migration target systems. On this separate system, export the storage by adding the default image directory to the **`/etc/exports`** file:

   ```
   /var/lib/libvirt/images *.example.com(rw,no_root_squash,sync)
   ```

   Change the hostname parameter as required for your environment.

2. **Start NFS**
   a. Install the NFS packages if they are not yet installed:

```
# yum install nfs
```

b. Open the ports for NFS in **iptables** and add NFS to the **/etc/hosts.allow** file.

c. Start the NFS service:

```
# service nfs start
```

3. **Mount the shared storage on the destination**
On the migration destination system, mount the **/var/lib/libvirt/images** directory:

```
# mount storage_host:/var/lib/libvirt/images /var/lib/libvirt/images
```

Locations must be the same on source and destination

Whichever directory is chosen for the guests must be exactly the same on host and guest. This applies to all types of shared storage. The directory must be the same or the migration will fail.

4.3. Live KVM migration with virsh

A guest can be migrated to another host with the **virsh** command. The **migrate** command accepts parameters in the following format:

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

The *GuestName* parameter represents the name of the guest which you want to migrate.

The *DestinationURL* parameter is the URL or hostname of the destination system. The destination system must run the same version of Red Hat Enterprise Linux, be using the same hypervisor and have **libvirt** running.

Once the command is entered you will be prompted for the root password of the destination system.

Important

An entry for the destination host, in the **/etc/hosts** file on the source server is required for migration to succeed. Enter the IP address and hostname for the destination host in this file as shown in the following example, substituting your destination host's IP address and hostname:

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

Example: live migration with virsh

This example migrates from `host1.example.com` to `host2.example.com`. Change the host names for your environment. This example migrates a virtual machine named **`guest1-rhel6-64`**.

This example assumes you have fully configured shared storage and meet all the prerequisites (listed here: *Migration requirements*).

1. **Verify the guest is running**
   From the source system, `host1.example.com`, verify `guest1-rhel6-64` is running:

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

2. **Migrate the guest**
   Execute the following command to live migrate the guest to the destination, `host2.example.com`. Append **`/system`** to the end of the destination URL to tell libvirt that you need full access.

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

   Once the command is entered you will be prompted for the root password of the destination system.

3. **Wait**
   The migration may take some time depending on load and the size of the guest. **`virsh`** only reports errors. The guest continues to run on the source host until fully migrated.

4. **Verify the guest has arrived at the destination host**
   From the destination system, `host2.example.com`, verify `guest1-rhel6-64` is running:

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

The live migration is now complete.

Other networking methods

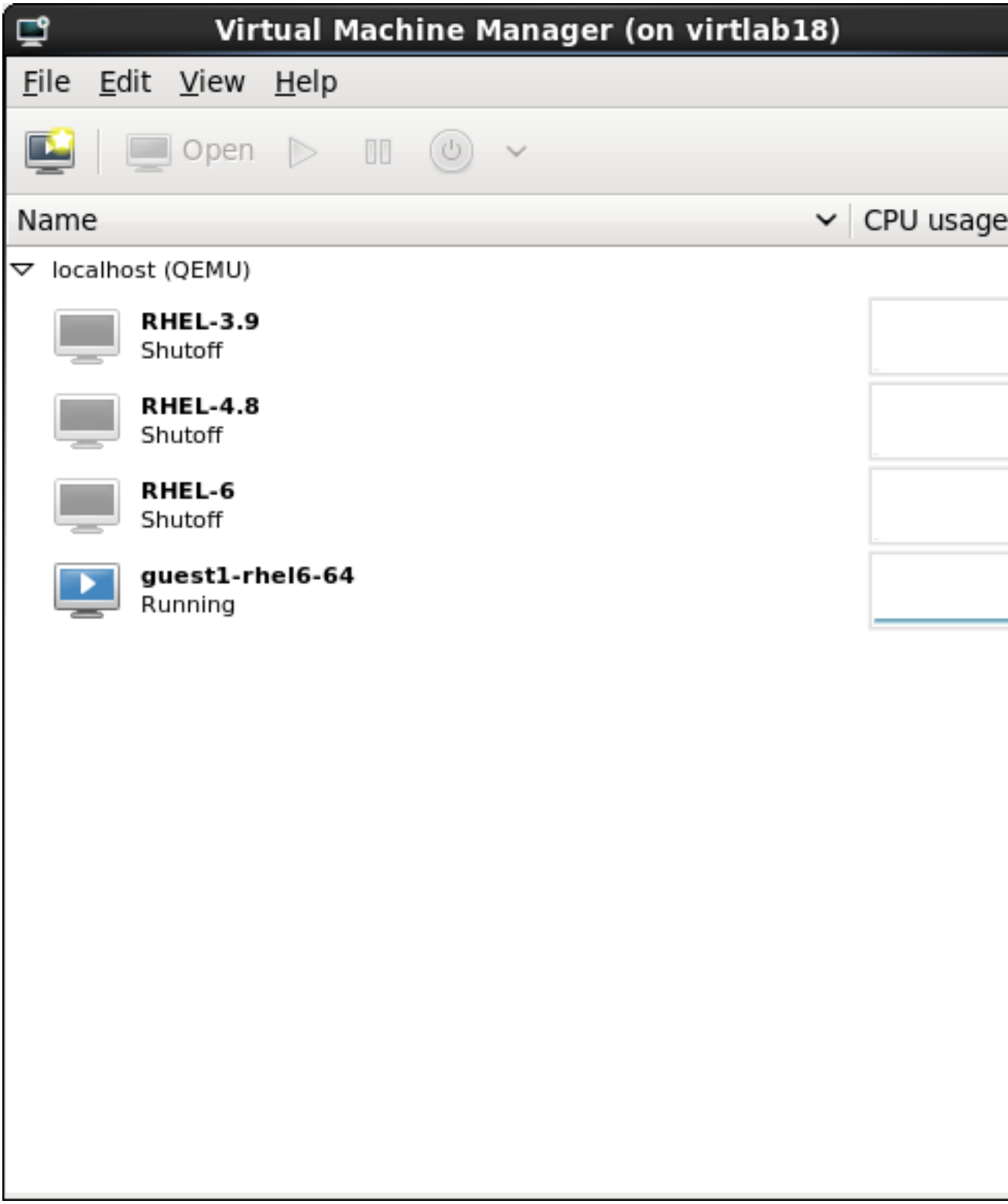
libvirt supports a variety of networking methods including TLS/SSL, unix sockets, SSH, and unencrypted TCP. Refer to *Chapter 5, Remote management of guests* for more information on using other methods.

4.4. Migrating with virt-manager

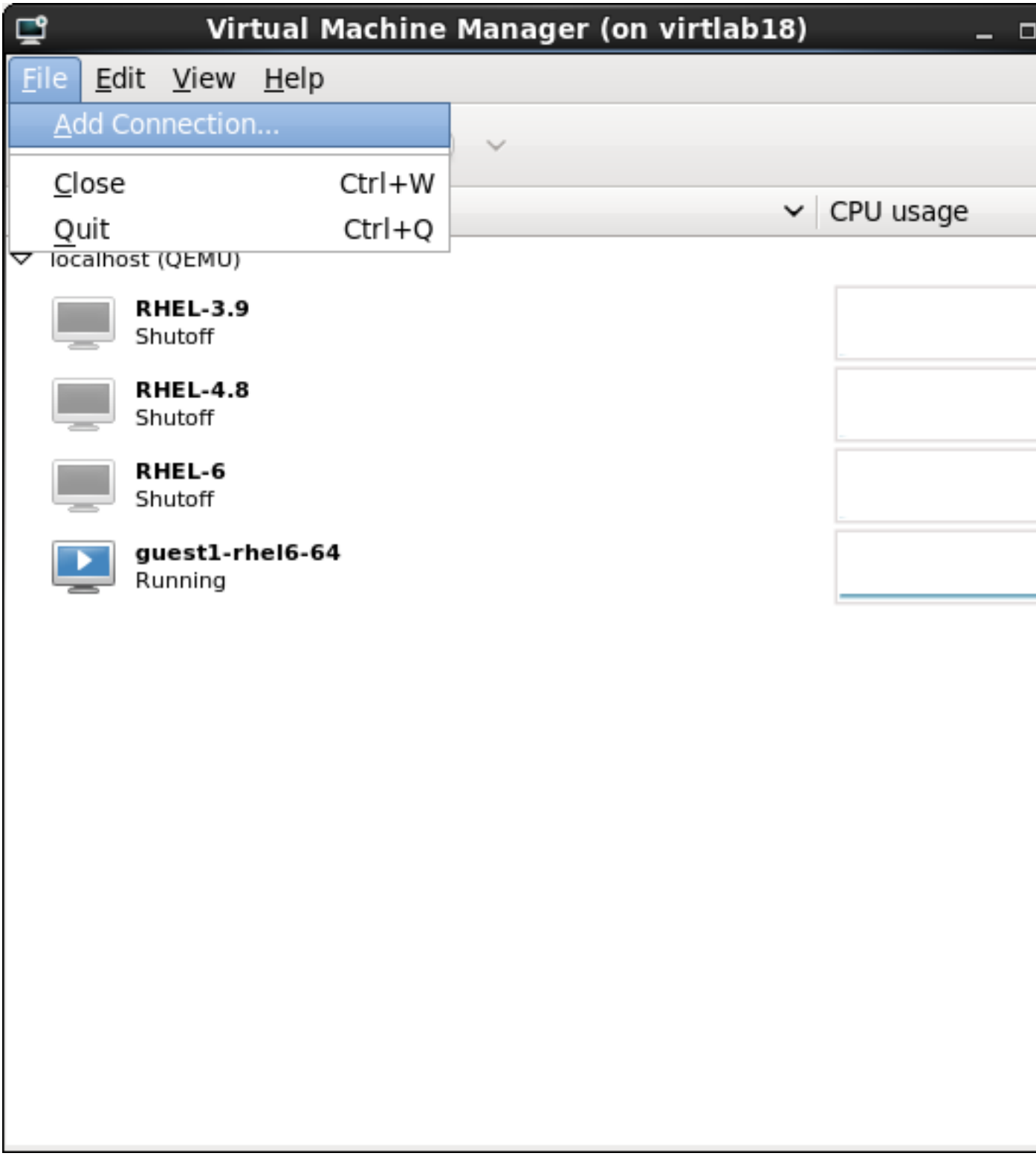
This section covers migrating a KVM guest with **`virt-manager`** from one host to another.

1. **Open virt-manager**

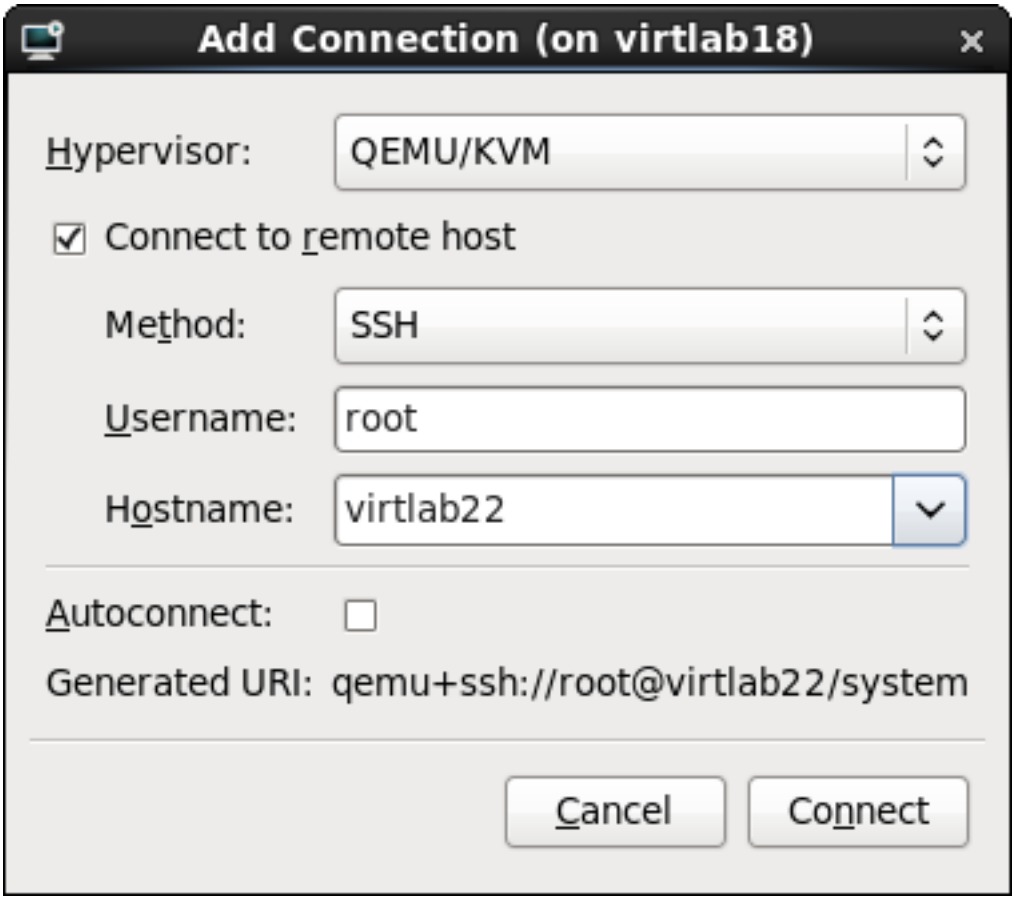
   Open **`virt-manager`**. Choose **Applications** → **System Tools** → **Virtual Machine Manager** from the main menu bar to launch **`virt-manager`**.



2. **Connect to the target host**
   Connect to the target host by clicking on the **File** menu, then click **Add Connection**.



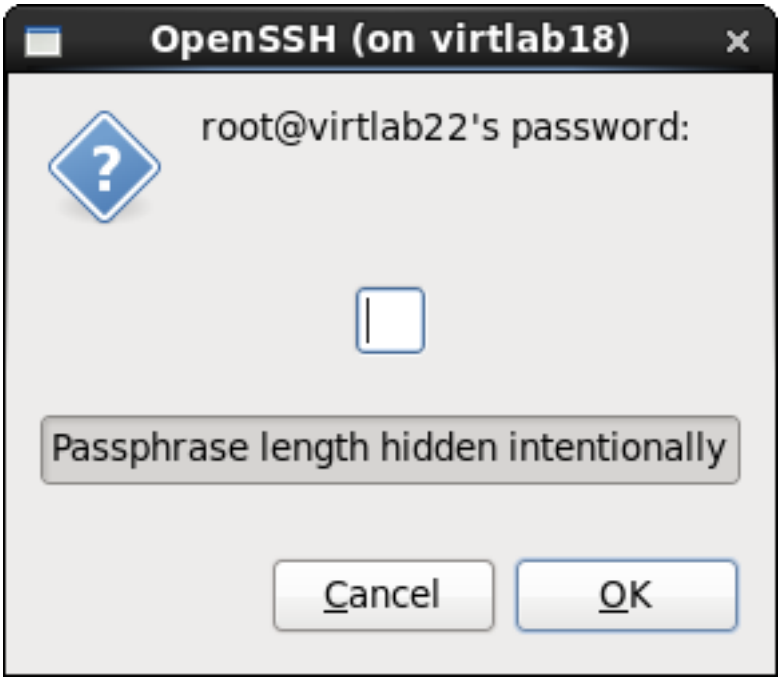
3. **Add connection**
   The **Add Connection** window appears.



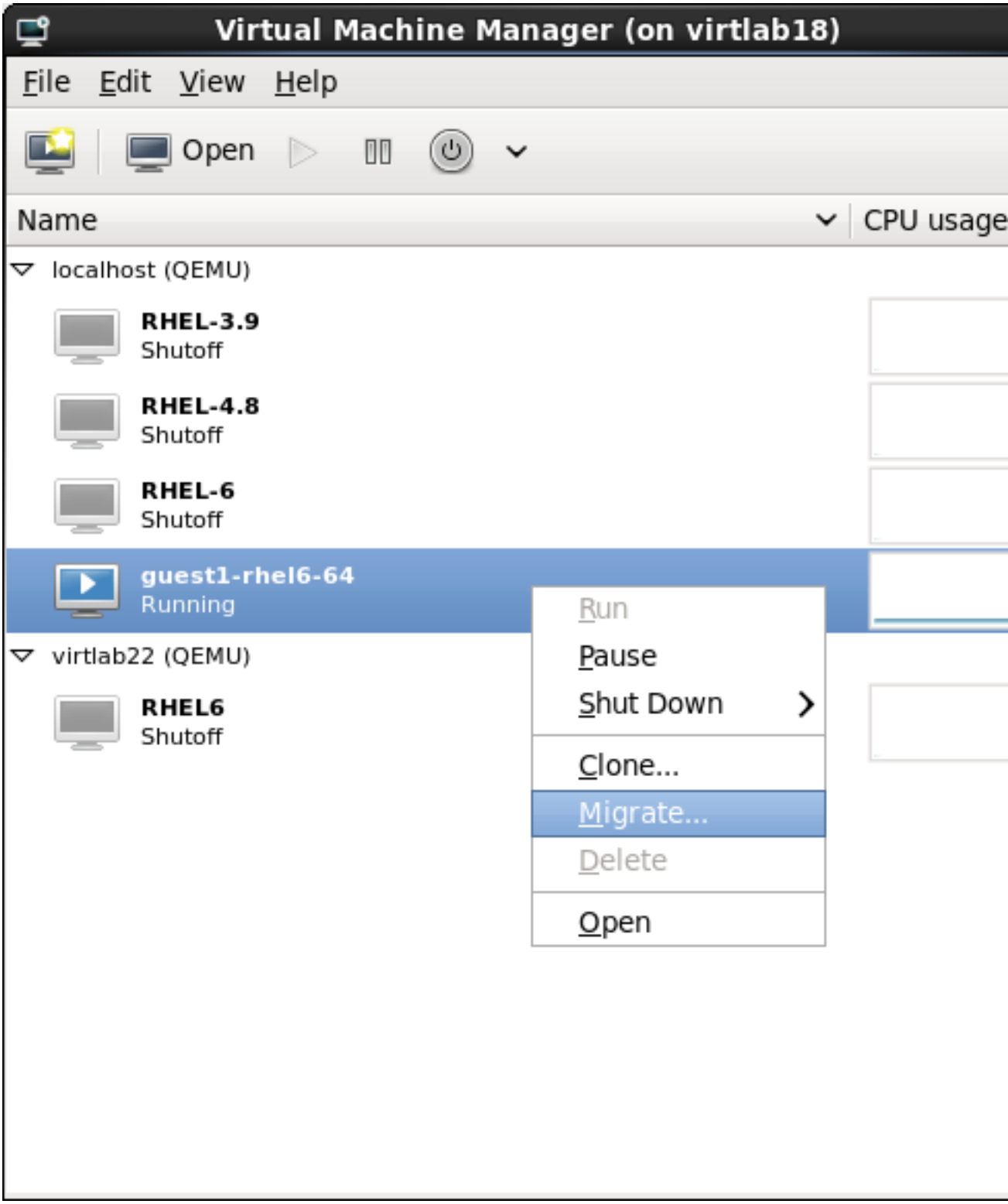
Enter the following details:

- **Hypervisor**: Select **QEMU/KVM**.
- **Method**: Select the connection method.
- **Username**: Enter the username for the remote host.
- **Hostname**: Enter the hostname for the remote host.

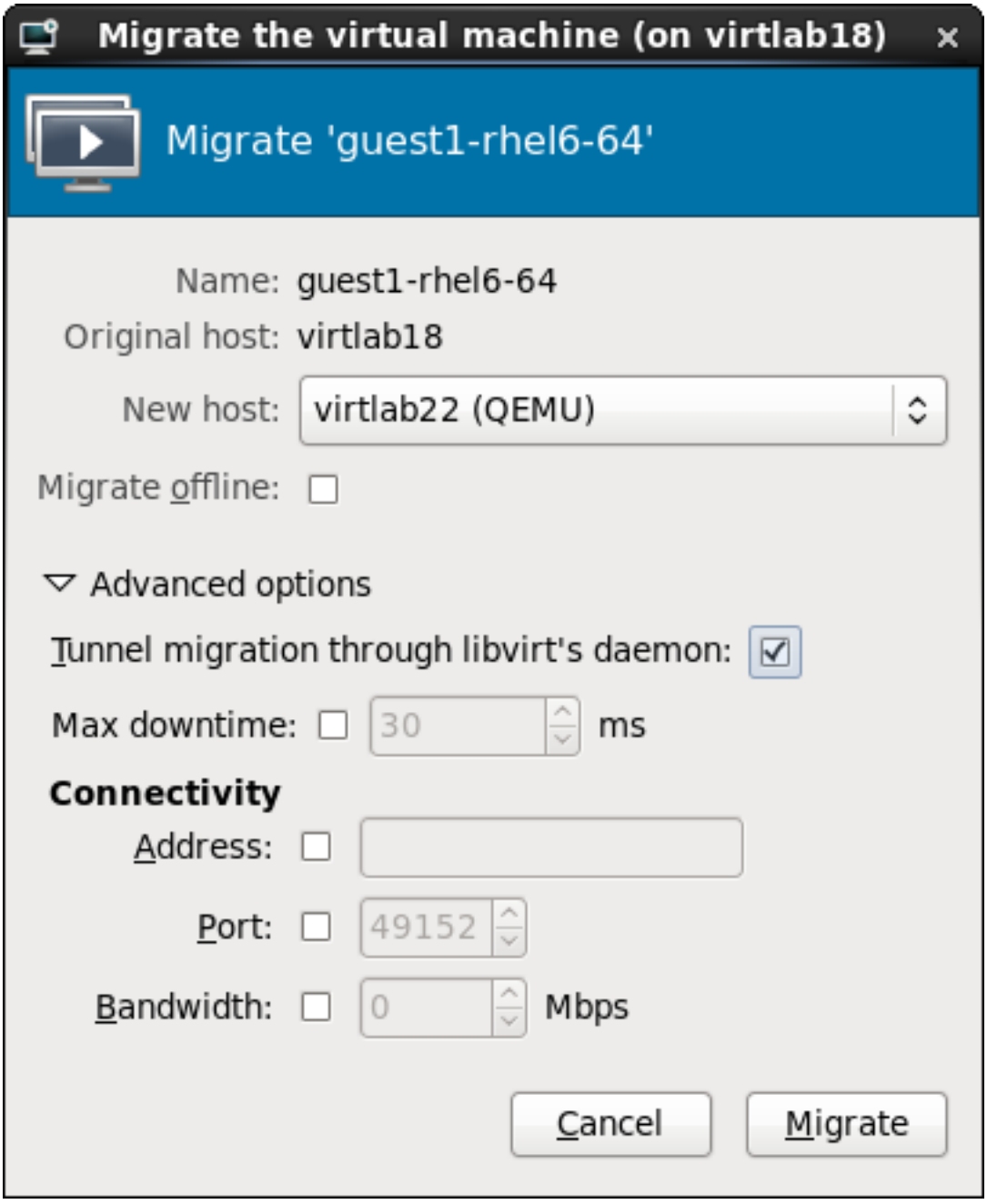
Click the **Connect** button. An SSH connection is used in this example, so the specified user's password must be entered in the next step.

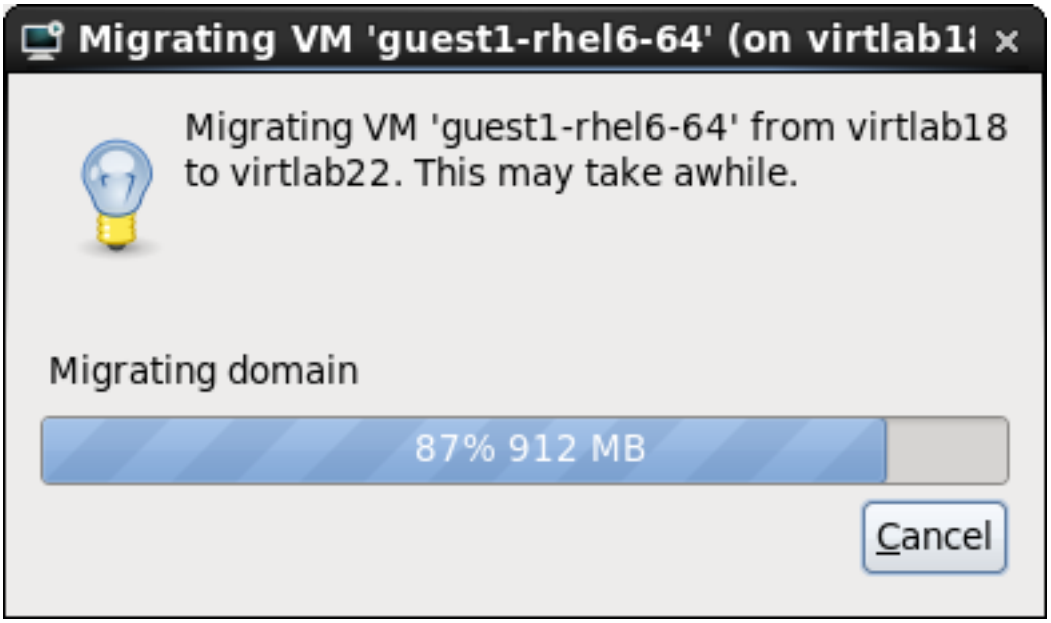


4. **Migrate guest**
   Right-click on the host to be migrated (**guest1-rhel6-64** in this example) and click **Migrate**.

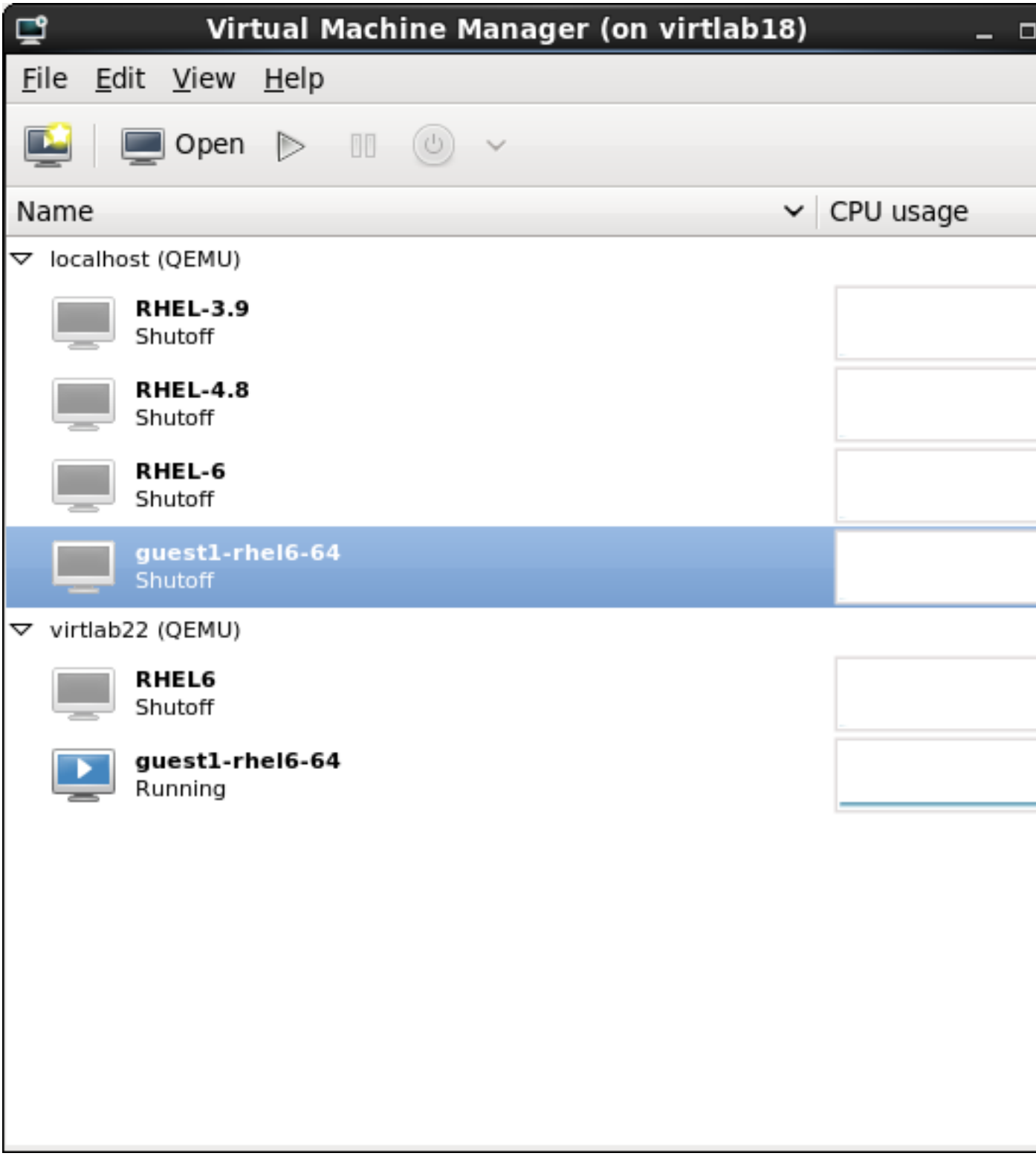


Select the host you wish to migrate to and click **Migrate**. A progress window will appear.


Migrate the virtual machine (on virtlab18)
Migrate 'guest1-rhel6-64'
Name: guest1-rhel6-64
Original host: virtlab18
New host: virtlab22 (QEMU)
Migrate offline:
Advanced options
Tunnel migration through libvirt's daemon:
Max downtime: 30 ms
Connectivity
Address:
Port: 49152
Bandwidth: 0 Mbps
Cancel
Migrate



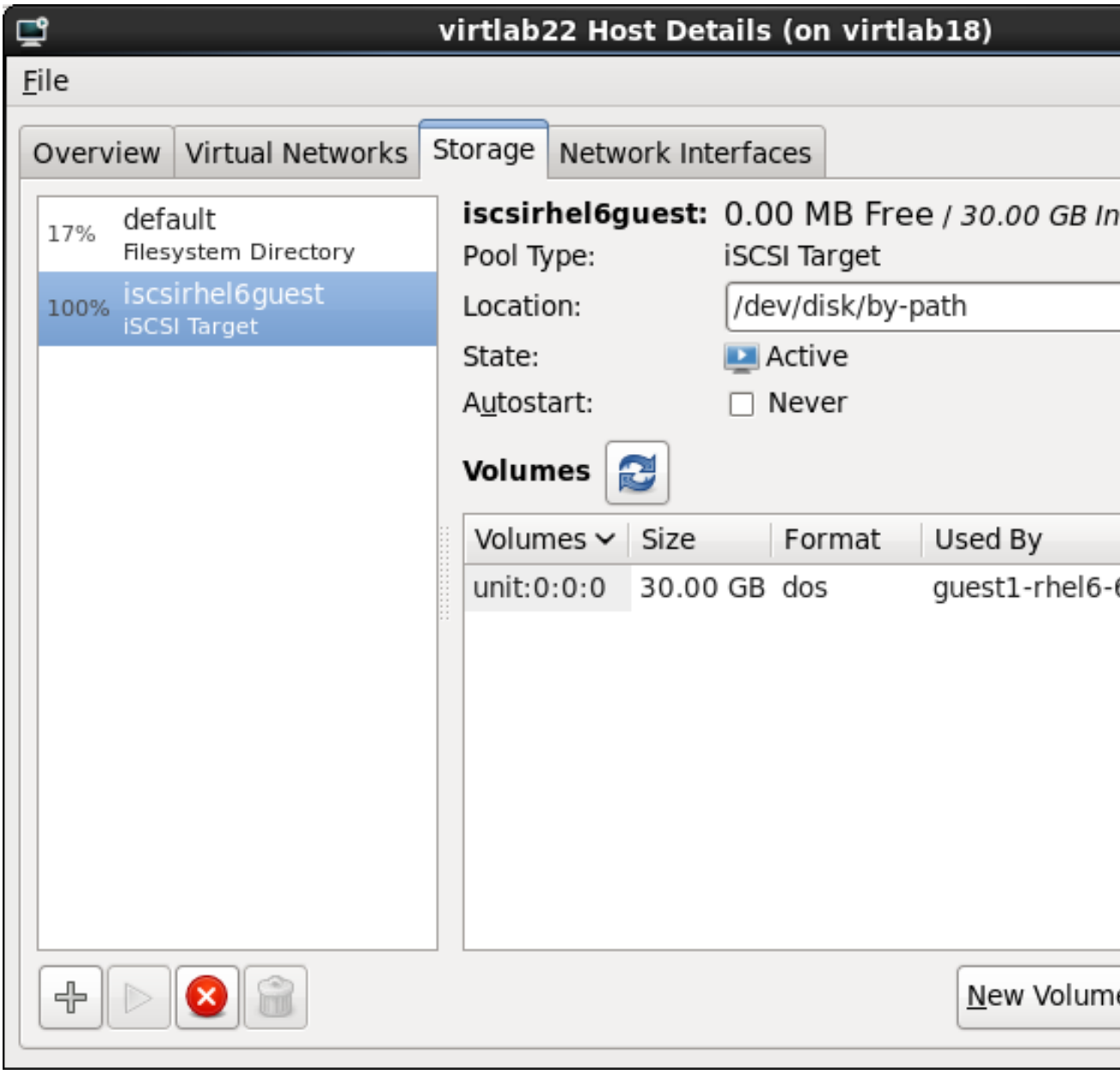
`virt-manager` now displays the newly migrated guest.



5. **View the storage details for the host**
   In the **Edit** menu, click **Host Details**, the Host Details window appears.

   Click the **Storage** tab. The iSCSI target details for this host is shown. This host was defined by the following XML configuration:

```
<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>
```



Remote management of guests

This section explains how to remotely manage your guests using **ssh** or TLS and SSL.

5.1. Remote management with SSH

The *ssh* package provides an encrypted network protocol which can securely send management functions to remote virtualization servers. The method described uses the **libvirt** management connection securely tunneled over an **SSH** connection to manage the remote machines. All the authentication is done using **SSH** public key cryptography and passwords or passphrases gathered by your local **SSH** agent. In addition the **VNC** console for each guest is tunneled over **SSH**.

SSH is usually configured by default so you probably already have SSH keys setup and no extra firewall rules needed to access the management service or **VNC** console.

Be aware of the issues with using **SSH** for remotely managing your virtual machines, including:

- you require root log in access to the remote machine for managing virtual machines,
- the initial connection setup process may be slow,
- there is no standard or trivial way to revoke a user's key on all hosts or guests, and
- ssh does not scale well with larger numbers of remote machines.

Note

Red Hat Enterprise Virtualization enables remote management of large numbers of virtual machines. Refer to the Red Hat Enterprise Virtualization documentation for further details.

The following packages are required for ssh access:

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

Configuring password less or password managed SSH access for virt-manager

The following instructions assume you are starting from scratch and do not already have **SSH** keys set up. If you have SSH keys set up and copied to the other systems you can skip this procedure.

The user is important for remote management

SSH keys are user dependent. Only the user who owns the key may access that key.

virt-manager must be run by the user who owns the keys to connect to the remote host. That means, if the remote systems are managed by a non-root user **virt-manager** must be run in unprivileged mode. If the remote systems are managed by the local root user then the SSH keys must be owned and created by root.

You cannot manage the local host as an unprivileged user with **virt-manager**.

1. **Optional: Changing user**
   Change user, if required. This example uses the local root user for remotely managing the other hosts and the local host.

   ```
   $ su -
   ```

2. **Generating the SSH key pair**
   Generate a public key pair on the machine **virt-manager** is used. This example uses the default key location, in the **~/.ssh/** directory.

   ```
   $ ssh-keygen -t rsa
   ```

3. **Copying the keys to the remote hosts**
   Remote login without a password, or with a passphrase, requires an SSH key to be distributed to the systems being managed. Use the **ssh-copy-id** command to copy the key to root user at the system address provided (in the example, *root@host2.example.com*).

   ```
   $ ssh-copy-id -i ~/.ssh/id_rsa.pub root@host2.example.com
   root@host2.example.com's password:
   ```

   Now try logging into the machine, with the **ssh root@host2.example.com** command and check in the **.ssh/authorized_keys** file to make sure unexpected keys have not been added.

   Repeat for other systems, as required.

4. **Optional: Add the passphrase to the ssh-agent**
   Add the passphrase for the SSH key to the **ssh-agent**, if required. On the local host, use the following command to add the passphrase (if there was one) to enable password-less login.

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

The SSH key was added to the remote system.

The libvirt daemon (libvirtd)

The `libvirt` daemon provide an interface for managing virtual machines. You must have the `libvirtd` daemon installed and running on every remote host that needs managing.

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

After `libvirtd` and **SSH** are configured you should be able to remotely access and manage your virtual machines. You should also be able to access your guests with **VNC** at this point.

Accessing remote hosts with virt-manager

Remote hosts can be managed with the virt-manager GUI tool. SSH keys must belong to the user executing virt-manager for password-less login to work.

1. Start virt-manager.
2. Open the **File**->**Add Connection** menu.
3. Input values for the hypervisor type, the connection, Connection->Remote tunnel over SSH, and enter the desired hostname, then click connection.

5.2. Remote management over TLS and SSL

You can manage virtual machines using TLS and SSL. TLS and SSL provides greater scalability but is more complicated than ssh (refer to *Section 5.1, "Remote management with SSH"*). TLS and SSL is the same technology used by web browsers for secure connections. The **libvirt** management connection opens a TCP port for incoming connections, which is securely encrypted and authenticated based on x509 certificates.

It is necessary to place the Certificate Authority Certificate, Client Certificate, and Client Certificate Private Key, in the following locations:

- The Certificate Authority Certificate should be placed in **/etc/pki/CA/cacert.pem**.
- The Client Certificate, signed by the CA, should be placed in either of:
  - **/etc/pki/libvirt/clientcert.pem** for system wide use, or
  - **$HOME/.pki/libvirt/clientcert.pem** for an individual user.
- The Private Key for the Client Certificate should be placed in either of:
  - **/etc/pki/libvirt/private/clientkey.pem** for system wide use, or
  - **$HOME/.pki/libvirt/private/clientkey.pem** for an individual user.

5.3. Transport modes

For remote management, **libvirt** supports the following transport modes:

Transport Layer Security (TLS)

Transport Layer Security TLS 1.0 (SSL 3.1) authenticated and encrypted TCP/IP socket, usually listening on a public port number. To use this you will need to generate client and server certificates. The standard port is 16514.

UNIX sockets

Unix domain sockets are only accessible on the local machine. Sockets are not encrypted, and use UNIX permissions or SELinux for authentication. The standard socket names are **/var/run/libvirt/libvirt-sock** and **/var/run/libvirt/libvirt-sock-ro** (for read-only connections).

SSH

Transported over a Secure Shell protocol (SSH) connection. Requires Netcat (the *nc* package) installed. The libvirt daemon (**libvirtd**) must be running on the remote machine. Port 22 must be open for SSH access. You should use some sort of SSH key management (for example, the **ssh-agent** utility) or you will be prompted for a password.

ext

The *ext* parameter is used for any external program which can make a connection to the remote machine by means outside the scope of libvirt. This parameter is unsupported.

TCP

Unencrypted TCP/IP socket. Not recommended for production use, this is normally disabled, but an administrator can enable it for testing or use over a trusted network. The default port is 16509.

The default transport, if no other is specified, is TLS.

Remote URIs

A Uniform Resource Identifier (URI) is used by **virsh** and **libvirt** to connect to a remote host. URIs can also be used with the **--connect** parameter for the **virsh** command to execute single commands or migrations on remote hosts.

libvirt URIs take the general form (content in square brackets, "[]", represents optional functions):

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

The transport method or the hostname must be provided to target an external location.

Examples of remote management parameters

- Connect to a remote KVM host named `host2`, using SSH transport and the SSH username `virtuser`.

```
qemu+ssh://virtuser@host2/
```

- Connect to a remote KVM hypervisor on the host named `host2` using TLS.

```
qemu://host2/
```

Testing examples

- Connect to the local KVM hypervisor with a non-standard UNIX socket. The full path to the Unix socket is supplied explicitly in this case.

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

- Connect to the libvirt daemon with an unencrypted TCP/IP connection to the server with the IP address 10.1.1.10 on port 5000. This uses the test driver with default settings.

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

Extra URI parameters

Extra parameters can be appended to remote URIs. The table below *Table 5.1, "Extra URI parameters"* covers the recognized parameters. All other parameters are ignored. Note that parameter values must be URI-escaped (that is, a question mark (?) is appended before the parameter and special characters are converted into the URI format).

Table 5.1. Extra URI parameters

Name	Transport mode	Description	Example usage
name	all modes	The name passed to the remote virConnectOpen function. The name is normally formed by removing transport, hostname, port number, username and extra parameters from the remote URI, but in certain very complex cases it may be better to supply the name explicitly.	name=qemu:///system
command	ssh and ext	The external command. For ext transport this is required. For ssh the default is ssh. The PATH is searched for the command.	command=/opt/openssh/bin/ssh
socket	unix and ssh	The path to the UNIX domain socket, which overrides the default. For ssh transport, this is passed to the remote netcat command (see netcat).	socket=/opt/libvirt/run/libvirt/libvirt-sock
netcat	ssh	The **netcat** command can be used to connect to remote systems. The default netcat parameter uses the **nc** command. For SSH transport, libvirt constructs an SSH command using the form below:	netcat=/opt/netcat/bin/nc

Name	Transport mode	Description	Example usage
		command -p *port* [-l *username*] *hostname* *netcat* -U socket The *port*, *username* and *hostname* parameters can be specified as part of the remote URI. The *command*, *netcat* and *socket* come from other extra parameters.	
no_verify	tls	If set to a non-zero value, this disables client checks of the server's certificate. Note that to disable server checks of the client's certificate or IP address you must change the libvirtd configuration.	no_verify=1
no_tty	ssh	If set to a non-zero value, this stops ssh from asking for a password if it cannot log in to the remote machine automatically (for using ssh-agent or similar). Use this when you do not have access to a terminal - for example in graphical programs which use libvirt.	no_tty=1

Overcommitting with KVM

The KVM hypervisor supports overcommitting CPUs and overcommitting memory. Overcommitting is allocating more virtualized CPUs or memory than there are physical resources on the system. With CPU overcommit, under-utilized virtualized servers or desktops can run on fewer servers which saves a number of system resources, with the net effect of less power, cooling and investment in server hardware.

Overcommitting memory

Most operating systems and applications do not use 100% of the available RAM all the time. This behavior can be exploited with KVM. KVM can allocate more memory for guests than the host has physically available. Overcommitting requires sufficient swap space for all guests and all host processes.

With KVM, virtual machines are Linux processes. Guests on the KVM hypervisor do not have dedicated blocks of physical RAM assigned to them, instead guests function as Linux processes. The Linux kernel allocates each process memory when the process requests more memory. KVM guests are allocated memory when requested by the guest operating system.

Warning

Ensure that the total sum of swap and memory space is greater than or equal to all the memory configured for running guests. A shortage less than this sum can cause a guest to be forcibly shut down.

Configuring swap for overcommitting memory

The swap partition is used for swapping underused memory to the hard drive to speed up memory performance. The default size of the swap partition is calculated from the physical RAM of the host.

Red Hat *Knowledgebase*[1] has an article on safely and efficiently determining the size of the swap partition.

The swap partition must be large enough to provide virtual memory for all guests and the host system.

Important

The example below is provided as a guide for configuring swap only. The settings listed may not be appropriate for your environment.

> Example 6.1. Memory overcommit example
>
> ExampleServer1 has 32GB of RAM. The system is being configured to run 56 guests with 1GB of virtualized memory. The host system rarely uses more than 4GB of memory for system processes, drivers and storage caching.

[1] http://kbase.redhat.com/faq/docs/DOC-15252

> 32GB minus 4GB for the host leaves 28GB of physical RAM for guests. Each guest uses 1GB of RAM, a total of 56GB of virtual RAM is required for the guests.
>
> The Red Hat Knowledgebase recommends 8GB of swap for a system with 32GB of RAM. To safely overcommit memory there must be sufficient virtual memory for all guests and the host. The host has 28GB of RAM for guests (which need 56GB of RAM). Therefore, the system needs at least 28GB of swap for the guests.
>
> ExampleServer1 requires at least 36GB (8GB for the host and 28GB for the guests) of swap to safely overcommit for all 56 guests.

It is possible to overcommit memory over ten times the amount of physical RAM in the system. This only works with certain types of guest, for example, desktop virtualization with minimal intensive usage or running several identical guests with KSM. Configuring swap and memory overcommit is not a formula, each environment and setup is different. Your environment must be tested and customized to ensure stability and performance.

For more information on KSM and overcommitting, refer to *Chapter 7, KSM*.

Overcommitting virtualized CPUs

The KVM hypervisor supports overcommitting virtualized CPUs. Virtualized CPUs can be overcommitted as far as load limits of guests allow. Use caution when overcommitting VCPUs as loads near 100% may cause dropped requests or unusable response times.

Virtualized CPUs are overcommitted best when each guest only has a single VCPU. The Linux scheduler is very efficient with this type of load. KVM should safely support guests with loads under 100% at a ratio of five VCPUs. Overcommitting single VCPU guests is not an issue.

You cannot overcommit symmetric multiprocessing guests on more than the physical number of processing cores. For example a guest with four VCPUs should not be run on a host with a dual core processor. Overcommitting symmetric multiprocessing guests in over the physical number of processing cores will cause significant performance degradation.

Assigning guests VCPUs up to the number of physical cores is appropriate and works as expected. For example, running guests with four VCPUs on a quad core host. Guests with less than 100% loads should function effectively in this setup.

Always test first

Do not overcommit memory or CPUs in a production environment without extensive testing. Applications which use 100% of memory or processing resources may become unstable in overcommitted environments. Test before deploying.

KSM

The concept of shared memory is common in modern operating systems. For example, when a program is first started it shares all of its memory with the parent program. When either the child or parent program tries to modify this memory, the kernel allocates a new memory region, copies the original contents and allows the program to modify this new region. This is known as copy on write.

KSM is a new Linux feature which uses this concept in reverse. KSM enables the kernel to examine two or more already running programs and compare their memory. If any memory regions or pages are identical, KSM reduces multiple identical memory pages to a single page. This page is then marked copy on write. If the contents of the page is modified by a guest, a new page is created for that guest.

This is useful for virtualization with KVM. When a guest is started, it only inherits the memory from the parent `qemu-kvm` process. Once the guest is running the contents of the guest operating system image can be shared when guests are running the same operating system or applications. KSM only identifies and merges identical pages which does not interfere with the guest or impact the security of the host or the guests. KSM allows KVM to request that these identical guest memory regions be shared.

KSM provides enhanced memory speed and utilization. With KSM, common process data is stored in cache or in main memory. This reduces cache misses for the KVM guests which can improve performance for some applications and operating systems. Secondly, sharing memory reduces the overall memory usage of guests which allows for higher densities and greater utilization of resources.

Red Hat Enterprise Linux uses two separate methods for controlling KSM:

- The `ksm` service starts and stops the KSM kernel thread.
- The `ksmtuned` service controls and tunes the `ksm`, dynamically managing same-page merging. The `ksmtuned` service starts `ksm` and stops the `ksm` service if memory sharing is not necessary. The `ksmtuned` service must be told with the *`retune`* parameter to run when new guests are created or destroyed.

Both of these services are controlled with the standard service management tools.

The KSM service

The `ksm` service is a standard Linux daemon that uses the KSM kernel features.

KSM is included in the *qemu-kvm* package. KSM is enabled by default in Red Hat Enterprise Linux. When the `ksm` service is not started, KSM shares only 2000 pages. This default is low and provides limited memory saving benefits.

When the `ksm` service is started, KSM will share up to half of the host system's main memory. Start the `ksm` service to enable KSM to share more memory.

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

The `ksm` service can be added to the default startup sequence. Make the `ksm` service persistent with the chkconfig command.

```
# chkconfig ksm on
```

The KSM tuning service

The `ksmtuned` service does not have any options. The `ksmtuned` service loops and adjusts `ksm`. The `ksmtuned` service is notified by libvirt when a guest is created or destroyed.

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

The `ksmtuned` service can be tuned with the *`retune`* parameter. The *`retune`* parameter instructs `ksmtuned` to run tuning functions manually.

The **`/etc/ksmtuned.conf`** file is the configuration file for the `ksmtuned` service. The file output below is the default **`ksmtuned.conf`** file.

```
# 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=300
# KSM_NPAGES_DECAY=-50
# KSM_NPAGES_MIN=64
# KSM_NPAGES_MAX=1250

# KSM_THRES_COEF=20
# KSM_THRES_CONST=2048

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

KSM variables and monitoring

KSM stores monitoring data in the **`/sys/kernel/mm/ksm/`** directory. Files in this directory are updated by the kernel and are an accurate record of KSM usage and statistics.

The variables in the list below are also configurable variables in the **`/etc/ksmtuned.conf`** file as noted below.

The **`/sys/kernel/mm/ksm/`** files

full_scans
: Full scans run.

pages_shared
: Total pages shared.

pages_sharing
: Pages presently shared.

pages_to_scan
: Pages not scanned.

pages_unshared
: Pages no longer shared.

pages_volatile
: Number of volatile pages.

run
: Whether the KSM process is running.

sleep_millisecs
: Sleep milliseconds.

KSM tuning activity is stored in the **/var/log/ksmtuned** log file if the *DEBUG=1* line is added to the **/etc/ksmtuned.conf** file. The log file location can be changed with the *LOGFILE* parameter. Changing the log file location is not advised and may require special configuration of SELinux settings.

Deactivating KSM

KSM has a performance overhead which may be too large for certain environments or host systems.

KSM can be deactivated by stopping the `ksm` service and the `ksmtuned` service. Stopping the services deactivates KSM but does not persist after restarting.

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

Persistently deactivate KSM with the **chkconfig** command. To turn off the services, run the following commands:

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

Overcommitting with KSM

Ensure the swap size is sufficient for the committed RAM even with KSM. KSM reduces the RAM usage of identical or similar guests. Overcommitting guests with KSM without sufficient swap space may be possible but is not recommended because guest memory use can result in pages becoming unshared.

Advanced virtualization administration

This chapter covers advanced administration tools for fine tuning and controlling guests and host system resources.

8.1. Control Groups (cgroups)

Red Hat Enterprise Linux 6 provides a new kernel feature: *control groups*, which are often referred to as *cgroups*. Cgroups allow you to allocate resources such as CPU time, system memory, network bandwidth, or combinations of these resources among user-defined groups of tasks (processes) running on a system. You can monitor the cgroups you configure, deny cgroups access to certain resources, and even reconfigure your cgroups dynamically on a running system.

The cgroup functionality is fully supported by libvirt. By default, libvirt puts each guest into a separate control group for various controllers (such as memory, cpu, blkio, device).

When a guest is started, it is already in a cgroup. The only configuration that may be required is the setting of policies on the cgroups. Refer to the *Red Hat Enterprise Linux Resource Management Guide* for more information on cgroups.

8.2. Hugepage support

Introduction

x86 CPUs usually address memory in 4kB pages, but they are capable of using larger pages known as **huge pages**. KVM guests can be deployed with huge page memory support in order to reduce memory consumption and improve performance by reducing CPU cache usage.

By using huge pages for a KVM guest, less memory is used for page tables and TLB (Translation Lookaside Buffer) misses are reduced, thereby significantly increasing performance, especially for memory-intensive situations.

Transparent Hugepage Support is a kernel feature that reduces TLB entries needed for an application. By also allowing all free memory to be used as cache, performance is increased.

Using Transparent Hugepage Support

To use Transparent Hugepage Support, no special configuration in the **`qemu.conf`** file is required. Hugepages are used by default if **`/sys/kernel/mm/redhat_transparent_hugepage/enabled`** is set to **always**.

Transparent Hugepage Support does not prevent the use of hugetlbfs. However, when hugetlbfs is not used, KVM will use transparent hugepages instead of the regular 4kB page size.

Miscellaneous administration tasks

This chapter contain useful hints and tips to improve virtualization performance, scale and stability.

9.1. Automatically starting guests

This section covers how to make guests start automatically during the host system's boot phase.

This example uses **virsh** to set a guest, *TestServer*, to automatically start when the host boots.

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

The guest now automatically starts with the host.

To stop a guest automatically booting use the *--disable* parameter

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

The guest no longer automatically starts with the host.

9.2. Using qemu-img

The **qemu-img** command line tool is used for formatting, modifying and verifying various file systems used by KVM. **qemu-img** options and usages are listed below.

Check

Perform a consistency check on the disk image *filename*.

```
# qemu-img check [-f format] filename
```

Note

Only the *qcow2*, *qed* and *vdi* formats support consistency checks.

Commit

Commit any changes recorded in the specified file (*filename*) to the file's base image with the **qemu-img commit** command. Optionally, specify the file's format type (*format*).

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

Convert

The *convert* option is used to convert one recognized image format to another image format.

Command format:

```
# qemu-img convert [-c] [-f format] [-o options] [-O output_format] filename output_filename
```

Convert the disk image *filename* to disk image *output_filename* using format *output_format*. The disk image can be optionally compressed with the *-c* option, or encrypted with the *-o* option by setting **-o encryption**. Note that the options available with the *-o* parameter differ with the selected format.

Only the **qcow2** format supports encryption or compression. **qcow2** encryption uses the AES format with secure 128-bit keys. **qcow2** compression is read-only, so if a compressed sector is converted from **qcow2** format, it is written to the new format as uncompressed data.

Image conversion is also useful to get a smaller image when using a format which can grow, such as **qcow** or **cow**. The empty sectors are detected and suppressed from the destination image.

Create

Create the new disk image *filename* of size *size* and format *format*.

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

If a base image is specified with **-o backing_file=*filename***, the image will only record differences between itself and the base image. The backing file will not be modified unless you use the **commit** command. No size needs to be specified in this case.

Info

The **info** parameter displays information about a disk image *filename*. The format for the **info** option is as follows:

```
# qemu-img info [-f format] filename
```

This command is often used to discover the size reserved on disk which can be different from the displayed size. If snapshots are stored in the disk image, they are displayed also.

Rebase

Changes the backing file of an image.

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

The backing file is changed to *backing_file* and (if the format of *filename* supports the feature), the backing file format is changed to *backing_format*.

Note

Only the *qcow2* and *qed* formats support changing the backing file (rebase).

There are two different modes in which *rebase* can operate: **Safe** and **Unsafe**.

Safe mode is used by default and performs a real rebase operation. The new backing file may differ from the old one and the **qemu-img rebase** command will take care of keeping the guest-visible content of *filename* unchanged. In order to achieve this, any clusters that differ between *backing_file* and old backing file of *filename* are merged into *filename* before making any changes to the backing file.

Note that safe mode is an expensive operation, comparable to converting an image. The old backing file is required for it to complete successfully.

Unsafe mode is used if the *-u* option is passed to **qemu-img rebase**. In this mode, only the backing file name and format of *filename* is changed, without any checks taking place on the file contents. Make sure the new backing file is specified correctly or the guest-visible content of the image will be corrupted.

This mode is useful for renaming or moving the backing file. It can be used without an accessible old backing file. For instance, it can be used to fix an image whose backing file has already been moved or renamed.

Resize

Change the disk image *filename* as if it had been created with size *size*. Only images in raw format can be resized regardless of version. Red Hat Enterprise Linux 6.1 and later adds the ability to grow (but not shrink) images in qcow2 format.

Use the following to set the size of the disk image *filename* to *size* bytes:

```
# qemu-img resize filename size
```

You can also resize relative to the current size of the disk image. To give a size relative to the current size, prefix the number of bytes with **+** to grow, or **-** to reduce the size of the disk image by that number of bytes. Adding a unit suffix allows you to set the image size in kilobytes (K), megabytes (M), gigabytes (G) or terabytes (T).

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

Warning

Before using this command to shrink a disk image, you *must* use file system and partitioning tools inside the VM itself to reduce allocated file systems and partition sizes accordingly. Failure to do so will result in data loss.

After using this command to grow a disk image, you must use file system and partitioning tools inside the VM to actually begin using the new space on the device.

Snapshot

List, apply, create, or delete an existing snapshot (*snapshot*) of an image (*filename*).

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

-l lists all snapshots associated with the specified disk image. The apply option, **-a**, reverts the disk image (*filename*) to the state of a previously saved *snapshot*. **-c** creates a snapshot (*snapshot*) of an image (*filename*). **-d** deletes the specified snapshot.

Supported formats

qemu-img is designed to convert files to one of the following formats:

raw
Raw disk image format (default). This can be the fastest file-based format. If your file system supports holes (for example in ext2 or ext3 on Linux or NTFS on Windows), then only the written sectors will reserve space. Use **qemu-img info** to obtain the real size used by the image or **ls -ls** on Unix/Linux.

qcow2
QEMU image format, the most versatile format. Use it to have optional AES encryption, zlib-based compression, support of multiple VM snapshots, and smaller images, which are useful on file systems that do not support holes (non-NTFS file systems on Windows).

However, **qemu-img** also recognizes and supports the following formats in order to convert from them into either **raw** or **qcow2** format. The format of an image is usually detected automatically.

bochs
Bochs disk image format.

cloop
Linux Compressed Loop image, useful only to reuse directly compressed CD-ROM images present for example in the Knoppix CD-ROMs.

cow
User Mode Linux Copy On Write image format. The **cow** format is included only for compatibility with previous versions. It does not work with Windows.

dmg
Mac disk image format.

nbd
Network block device.

parallels
Parallels virtualization disk image format.

qcow
Old QEMU image format. Only included for compatibility with older versions.

vdi
Oracle VM VirtualBox hard disk image format.

vmdk
VMware 3 and 4 compatible image format.

vpc
Windows Virtual PC disk image format. Also referred to as **vhd**, or Microsoft virtual hard disk image format.

vvfat
Virtual VFAT disk image format.

9.3. Verifying virtualization extensions

Use this section to determine whether your system has the hardware virtualization extensions. Virtualization extensions (Intel VT-x or AMD-V) are required for full virtualization.

1. Run the following command to verify the CPU virtualization extensions are available:

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

2. Analyze the output.

   - The following output contains a **vmx** entry indicating an Intel processor with the Intel VT-x extension:

     ```
     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
     ```

   - The following output contains an **svm** entry indicating an AMD processor with the AMD-V extensions:

     ```
     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
     ```

   If any output is received, the processor has the hardware virtualization extensions. However in some circumstances manufacturers disable the virtualization extensions in BIOS.

   The "**flags:**" output content may appear multiple times, once for each hyperthread, core or CPU on the system.

   The virtualization extensions may be disabled in the BIOS. If the extensions do not appear or full virtualization does not work refer to *Procedure 22.1, "Enabling virtualization extensions in BIOS"*.

3. **Ensure KVM subsystem is loaded**
   As an additional check, verify that the **kvm** modules are loaded in the kernel:

   ```
   # lsmod | grep kvm
   ```

   If the output includes **kvm_intel** or **kvm_amd** then the **kvm** hardware virtualization modules are loaded and your system meets requirements.

Additional output

If the *libvirt* package is installed, the **virsh** command can output a full list of virtualization system capabilities. Run **virsh capabilities** as root to receive the complete list.

9.4. Setting KVM processor affinities

Terminology

libvirt refers to a NUMA node as a *cell*.

This section covers setting processor and processing core affinities with **libvirt** and KVM guests.

By default, libvirt provisions guests using the hypervisor's default policy. For most hypervisors, the policy is to run guests on any available processing core or CPU. There are times when an explicit policy may be better, particularly for systems with a NUMA (Non-Uniform Memory Access) architecture. A guest on a NUMA system can be pinned to a processing core so that its memory allocations are always local to the node it is running on. This avoids cross-node memory transports which have less bandwidth and can significantly degrade performance.

On non-NUMA systems some form of explicit placement across the hosts' sockets, cores and hyperthreads may be more efficient.

Identifying CPU and NUMA topology

The first step in deciding which policy to apply is to determine the host's memory and CPU topology. The **`virsh nodeinfo`** command provides information about how many sockets, cores and hyperthreads are attached to a host.

```
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              8
CPU frequency:       1000 MHz
CPU socket(s):       2
Core(s) per socket:  4
Thread(s) per core:  1
NUMA cell(s):        2
Memory size:         8179176 kB
```

This output shows that the system has eight CPU cores and two sockets. Each CPU socket has four cores. This splitting of CPU cores across multiple sockets suggests that the system has Non-Uniform Memory Access (NUMA) architecture.

NUMA architecture can be more complex than other architectures. Use the **`virsh capabilities`** command to get additional output data about the CPU configuration.

```
# virsh capabilities
<capabilities>
  <host>
    <cpu>
      <arch>x86_64</arch>
    </cpu>
    <migration_features>
      <live/>
      <uri_transports>
        <uri_transport>tcp</uri_transport>
      </uri_transports>
    </migration_features>
    <topology>
      <cells num='2'>
        <cell id='0'>
          <cpus num='4'>
            <cpu id='0'/>
            <cpu id='1'/>
            <cpu id='2'/>
            <cpu id='3'/>
          </cpus>
        </cell>
        <cell id='1'>
          <cpus num='4'>
            <cpu id='4'/>
            <cpu id='5'/>
```

```
            <cpu id='6'/>
            <cpu id='7'/>
          </cpus>
        </cell>
      </cells>
    </topology>
    <secmodel>
      <model>selinux</model>
      <doi>0</doi>
    </secmodel>
  </host>

 [ Additional XML removed ]

</capabilities>
```

This output shows two NUMA nodes (also know as NUMA cells), each containing four logical CPUs (four processing cores). This system has two sockets, therefore it can be inferred that each socket is a separate NUMA node. For a guest with four virtual CPUs, it is optimal to lock the guest to physical CPUs 0 to 3, or 4 to 7, to avoid accessing non-local memory, which is significantly slower than accessing local memory.

If a guest requires eight virtual CPUs, you could run two sets of four virtual CPU guests and split the work between them, since each NUMA node only has four physical CPUs. Running across multiple NUMA nodes significantly degrades performance for physical and virtualized tasks.

Decide which NUMA node can run the guest

Locking a guest to a particular NUMA node offers no benefit if that node does not have sufficient free memory for that guest. libvirt stores information on the free memory available on each node. Use the **`virsh freecell --all`** command to display the free memory on all NUMA nodes.

```
# virsh freecell --all
0: 2203620 kB
1: 3354784 kB
```

If a guest requires 3 GB of RAM allocated, then the guest should be run on NUMA node (cell) 1. Node 0 only has 2.2GB free which may not be sufficient for certain guests.

Lock a guest to a NUMA node or physical CPU set

Once you have determined which node to run the guest on, refer to the capabilities data (the output of the **`virsh capabilities`** command) about NUMA topology.

1. Extract from the **`virsh capabilities`** output.

```
<topology>
  <cells num='2'>
    <cell id='0'>
    <cpus num='4'>
      <cpu id='0'/>
      <cpu id='1'/>
      <cpu id='2'/>
      <cpu id='3'/>
    </cpus>
  </cell>
  <cell id='1'>
    <cpus num='4'>
      <cpu id='4'/>
      <cpu id='5'/>
```

```
      <cpu id='6'/>
      <cpu id='7'/>
    </cpus>
  </cell>
  </cells>
</topology>
```

2. Observe that the node 1, **<cell id='1'>**, uses physical CPUs 4 to 7.

3. The guest can be locked to a set of CPUs by appending the **cpuset** attribute to the configuration file.

   a. While the guest is offline, open the configuration file with **virsh edit**.

   b. Locate the guest's virtual CPU count, defined in the **vcpus** element.

   ```
   <vcpus>4</vcpus>
   ```

   The guest in this example has four CPUs.

   c. Add a **cpuset** attribute with the CPU numbers for the relevant NUMA cell.

   ```
   <vcpus cpuset='4-7'>4</vcpus>
   ```

4. Save the configuration file and restart the guest.

The guest has been locked to CPUs 4 to 7.

Automatically locking guests to CPUs with virt-install

The **virt-install** provisioning tool provides a simple way to automatically apply a 'best fit' NUMA policy when guests are created.

The *cpuset* option for **virt-install** can use a CPU set of processors or the parameter *auto*. The *auto* parameter automatically determines the optimal CPU locking using the available NUMA data.

For a NUMA system, use the *--cpuset=auto* with the **virt-install** command when creating new guests.

Tuning CPU affinity on running guests

There may be times where modifying CPU affinities on running guests is preferable to rebooting the guest. The **virsh vcpuinfo** and **virsh vcpupin** commands can perform CPU affinity changes on running guests.

The **virsh vcpuinfo** command gives up to date information about where each virtual CPU is running.

In this example, *guest1* is a guest with four virtual CPUs is running on a KVM host.

```
# virsh vcpuinfo guest1
VCPU:           0
CPU:            3
State:          running
CPU time:       0.5s
CPU Affinity:   yyyyyyyy
VCPU:           1
```

```
CPU:            1
State:          running
CPU Affinity:   yyyyyyyy
VCPU:           2
CPU:            1
State:          running
CPU Affinity:   yyyyyyyy
VCPU:           3
CPU:            2
State:          running
CPU Affinity:   yyyyyyyy
```

The **virsh vcpuinfo** output (the **yyyyyyyy** value of **CPU Affinity**) shows that the guest can presently run on any CPU.

To lock the virtual CPUs to the second NUMA node (CPUs four to seven), run the following commands.

```
# virsh vcpupin guest1 0 4
# virsh vcpupin guest1 1 5
# virsh vcpupin guest1 2 6
# virsh vcpupin guest1 3 7
```

The **virsh vcpuinfo** command confirms the change in affinity.

```
# virsh vcpuinfo guest1
VCPU:           0
CPU:            4
State:          running
CPU time:       32.2s
CPU Affinity:   ----y---
VCPU:           1
CPU:            5
State:          running
CPU time:       16.9s
CPU Affinity:   -----y--
VCPU:           2
CPU:            6
State:          running
CPU time:       11.9s
CPU Affinity:   ------y-
VCPU:           3
CPU:            7
State:          running
CPU time:       14.6s
CPU Affinity:   -------y
```

9.5. Generating a new unique MAC address

In some case you will need to generate a new and unique MAC address for a guest. There is no command line tool available to generate a new MAC address at the time of writing. The script provided below can generate a new MAC address for your guests. Save the script to your guest as **macgen.py**. Now from that directory you can run the script using **./macgen.py** and it will generate a new MAC address. A sample output would look like the following:

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

```
#!/usr/bin/python
```

```
# macgen.py script to generate a MAC address for guests
#
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()
```

Another method to generate a new MAC for your guest

You can also use the built-in modules of **`python-virtinst`** to generate a new MAC address and **`UUID`** for use in a guest configuration file:

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

The script above can also be implemented as a script file as seen below.

```
#!/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 ""
```

9.6. Improving guest response time

Guests can sometimes be slow to respond with certain workloads and usage patterns. Examples of situations which may cause slow or unresponsive guests:

- Severely overcommitted memory.
- Overcommitted memory with high processor usage
- Other (not **`qemu-kvm`** processes) busy or stalled processes on the host.

These types of workload may cause guests to appear slow or unresponsive. Usually, the guest's memory is eventually fully loaded into the host's main memory from swap. Once the guest is loaded in main memory, the guest will perform normally. Note, the process of loading a guest from swap to main memory may take several seconds per gigabyte of RAM assigned to the guest, depending on the type of storage used for swap and the performance of the components.

KVM guests function as Linux processes. Linux processes are not permanently kept in main memory (physical RAM). The kernel scheduler swaps process memory into virtual memory (swap). Swap, with conventional hard disk drives, is thousands of times slower than main memory in modern computers. If a guest is inactive for long periods of time, the guest may be placed into swap by the kernel.

KVM guests processes may be moved to swap regardless of whether memory is overcommitted or overall memory usage.

Using unsafe overcommit levels or overcommitting with swap turned off guest processes or other critical processes may be killed by the **pdflush** kernel function. **pdflush** automatically kills processes to keep the system from crashing and to free up memory. Always ensure the host has sufficient swap space when overcommitting memory.

For more information on overcommitting with KVM, refer to *Chapter 6, Overcommitting with KVM*.

Warning: turning off swap

Virtual memory allows a Linux system to use more memory than there is physical RAM on the system. Underused processes are swapped out which allows active processes to use memory, improving memory utilization. Disabling swap reduces memory utilization as all processes are stored in physical RAM.

If swap is turned off, do not overcommit guests. Overcommitting guests without any swap can cause guests or the host system to crash.

Turning off swap

Swap usage can be completely turned off to prevent guests from being unresponsive while they are moved back to main memory. Swap may also not be desired for guests as it can be resource-intensive on some systems.

The **swapoff** command can disable all swap partitions and swap files on a system.

```
# swapoff -a
```

To make this change permanent, remove *swap* lines from the **/etc/fstab** file and restart the host system.

Using SSDs for swap

Using Solid State Drives (SSDs) for swap storage may improve the performance of guests.

Using RAID arrays, faster disks or separate drives dedicated to swap may also improve performance.

9.7. Disable SMART disk monitoring for guests

SMART disk monitoring can be safely disabled as virtual disks and the physical storage devices are managed by the host.

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

9.8. Configuring a VNC Server

To configure a VNC server use the **Remote Desktop** application in **System** > **Preferences**. Alternatively, you can run the **vino-preferences** command.

The following steps set up a dedicated VNC server session:

1. Edit the **~/.vnc/xstartup** file to start a GNOME session whenever **vncserver** is started. The first time you run the **vncserver** script it will ask you for a password you want to use for your VNC session.

2. A sample **xstartup** file:

```
#!/bin/sh
# Uncomment the following two lines for normal desktop:
# unset SESSION_MANAGER
# exec /etc/X11/xinit/xinitrc
[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
#xsetroot -solid grey
#vncconfig -iconic &
#xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
#twm &
if test -z "$DBUS_SESSION_BUS_ADDRESS" ; then
 eval `dbus-launch --sh-syntax –exit-with-session`
 echo "D-BUS per-session daemon address is: \
 $DBUS_SESSION_BUS_ADDRESS"
fi
exec  gnome-session
```

9.9. Gracefully shutting down guests

Installing virtualized Red Hat Enterprise Linux 6 guests with the **Minimal installation** installation option will not install the *acpid* package.

Without the *acpid* package, the Red Hat Enterprise Linux 6 guest does not shut down when the **virsh shutdown** command is executed. The **virsh shutdown** command is designed to gracefully shut down guests.

Using **virsh shutdown** is easier and safer for system administration. Without graceful shut down with the **virsh shutdown** command a system administrator must log into a guest manually or send the **Ctrl**-**Alt**-**Del** key combination to each guest.

Other virtualized operating systems

Other virtualized operating systems may be affected by this issue. The **virsh shutdown** command requires that the guest operating system is configured to handle ACPI shut down requests. Many operating systems require additional configuration on the guest operating system to accept ACPI shut down requests.

Procedure 9.1. Workaround for Red Hat Enterprise Linux 6

1. **Install the acpid package**
   The **acpid** service listen and processes ACPI requests.

   Log into the guest and install the *acpid* package on the guest:

```
# yum install acpid
```

2. **Enable the acpid service**
   Set the `acpid` service to start during the guest boot sequence and start the service:

```
# chkconfig acpid on
# service acpid start
```

The guest is now configured to shut down when the **virsh shutdown** command is used.

9.10. Virtual machine timer management with libvirt

Accurate time keeping on guests is a key challenge for virtualization platforms. Different hypervisors attempt to handle the problem of time keeping in a variety of ways. Libvirt provides hypervisor independent configuration settings for time management, using the <clock> and <timer> elements in the domain XML. The domain XML can be edited using the **virsh edit** command. See *Editing a guest's configuration file* for details.

<clock>

The clock element is used to determine how the guest clock is synchronized with the host clock. The clock element has the following attributes:

- **offset**
  Determines how the guest clock is offset from the host clock. The offset attribute has the following possible values:

Table 9.1. Offset attribute values

Value	Description
utc	The guest clock will be synchronized to UTC when booted.
localtime	The guest clock will be synchronized to the host's configured timezone when booted, if any.
timezone	The guest clock will be synchronized to a given timezone, specified by the *timezone* attribute.
variable	The guest clock will be synchronized to an arbitrary offset from UTC. The delta relative to UTC is specified in seconds, using the *adjustment* attribute. The guest is free to adjust the Real Time Clock (RTC) over time and expect that it will be honored following the next reboot. This is in contrast to *utc* mode, where any RTC adjustments are lost at each reboot.

Note

The value **utc** is set as the clock offset in a virtual machine by default. However, if the guest clock is run with the **localtime** value, the clock offset needs to be changed to a different value in order to have the guest clock synchronized with the host clock.

- **timezone**
  The timezone to which the guest clock is to be synchronized.

- **adjustment**
  The delta for guest clock synchronization. In seconds, relative to UTC.

Example 9.1. Always synchronize to UTC

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

Example 9.2. Always synchronize to the host timezone

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

Example 9.3. Synchronize to an arbitrary timezone

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

Example 9.4. Synchronize to UTC + arbitrary offset

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

<timer>

A clock element can have zero or more timer elements as children. The timer element specifies a time source used for guest clock synchronization. The timer element has the following attributes. Only the *name* is required, all other attributes are optional.

- **name**
  The name of the time source to use.

Table 9.2. name attribute values

Value	Description
platform	The master virtual time source which may be used to drive the policy of other time sources.
pit	Programmable Interval Timer - a timer with periodic interrupts.
rtc	Real Time Clock - a continuously running timer with periodic interrupts.
hpet	High Precision Event Timer - multiple timers with periodic interrupts.
tsc	Time Stamp Counter - counts the number of ticks since reset, no interrupts.

- **track**
  The *track* attribute specifies what is tracked by the timer. Only valid for a name value of *`platform`* or *`rtc`*.

Table 9.3. wallclock attribute values

Value	Description
boot	Corresponds to old *host* option, this is an unsupported tracking option.
guest	RTC always tracks guest time.
wall	RTC always tracks host time.

- **tickpolicy**
  The policy used to pass ticks on to the guest.

Table 9.4. tickpolicy attribute values

Value	Description
none	Continue to deliver at normal rate (i.e. ticks are delayed).
catchup	Deliver at a higher rate to catch up.
merge	Ticks merged into one single tick.
discard	All missed ticks are discarded.

- **frequency**
  Used to set a fixed frequency, measured in Hz. This attribute is only relevant for a name value of `tsc`. All other timers operate at a fixed frequency (`pit`, `rtc`), or at a frequency fully controlled by the guest (`hpet`).

- **mode**
  Determines how the time source is exposed to the guest. This attribute is only relevant for a name value of `tsc`. All other timers are always emulated.

Table 9.5. mode attribute values

Value	Description
auto	Native if safe, otherwise emulated.
native	Always native.
emulate	Always emulate.
paravirt	Native + para-virtualized.

- **present**
  Used to override the default set of timers visible to the guest. For example, to enable or disable the HPET.

Table 9.6. present attribute values

Value	Description
yes	Force this timer to the visible to the guest.
no	Force this timer to not be visible to the guest.

Example 9.5. Clock synchronizing to local time with RTC and PIT timers, and the HPET timer disabled

```
<clock offset="localtime">
 <timer name="rtc" tickpolicy="catchup" wallclock="guest" />
```

```
  <timer name="pit" tickpolicy="none" />
  <timer name="hpet" present="no" />
</clock>
```

Storage concepts

This chapter introduces the concepts used for describing and managing storage devices.

Local storage

Local storage is directly attached to the host server. Local storage includes local directories, directly attached disks, and LVM volume groups on local storage devices.

Networked storage

Networked storage covers storage devices shared over a network using standard protocols. Networked storage includes shared storage devices using Fibre Channel, iSCSI, NFS, GFS2, and SCSI RDMA protocols. Networked storage is a requirement for migrating guests between hosts. Networked storage pools are managed by libvirt.

10.1. Storage pools

A *storage pool* is a file, directory, or storage device managed by libvirt for the purpose of providing storage to guests. Storage pools are divided into storage *volumes* that store guest images or are attached to guests as additional storage.

libvirt uses a directory-based storage pool, the **`/var/lib/libvirt/images/`** directory, as the default storage pool. The default storage pool can be changed to another storage pool.

- **Local storage pools** - As local storage pools are directly attached to the host server, they are useful for development, testing and small deployments that do not require migration or large numbers of guests. Local storage pools are not suitable for many production environments as local storage pools do not support live migration.
- **Networked (shared) storage pools** - Networked storage pools cover storage devices shared over a network using standard protocols. Supported protocols for networked storage pools:
  - Fibre Channel-based LUNs
  - iSCSI
  - NFS
  - GFS2
  - SCSI RDMA protocols (SCSI RCP), the block export protocol used in Infiniband and 10GbE iWARP adapters.

10.2. Volumes

Storage pools are divided into storage volumes. Storage volumes are an abstraction of physical partitions, LVM logical volumes, file-based disk images and other storage types handled by libvirt. Storage volumes are presented to guests as local storage devices regardless of the underlying hardware.

Referencing volumes

To reference a specific volume, three approaches are possible:

The name of the volume and the storage pool

A volume may be referred to by name, along with an identifier for the storage pool it belongs in. On the virsh command line, this takes the form *--pool storage_pool volume_name*.

For example, a volume named *firstimage* in the *guest_images* pool.

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

virsh #
```

The full path to the storage on the host system

A volume may also be referred to by its full path on the file system. When using this approach, a pool identifier does not need to be included.

For example, a volume named *secondimage.img*, visible to the host system as */images/secondimage.img*. The image can be referred to as */images/secondimage.img*.

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

The unique volume key

When a volume is first created in the virtualization system, a unique identifier is generated and assigned to it. The unique identifier is termed the *volume key*. The format of this volume key varies upon the storage used.

When used with block based storage such as LVM, the volume key may follow this format:

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

When used with file based storage, the volume key may instead be a copy of the full path to the volume storage.

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

For example, a volume with the volume key of *Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr*:

```
# virsh vol-info Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
Name:           firstimage
Type:           block
Capacity:       20.00 GB
Allocation:     20.00 GB
```

virsh provides commands for converting between a volume name, volume path, or volume key:

vol-name

Returns the volume name when provided with a volume path or volume key.

```
# virsh vol-name /dev/guest_images/firstimage
```

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

vol-path

Returns the volume path when provided with a volume key, or a storage pool identifier and volume name.

```
# 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
```

The vol-key command

Returns the volume key when provided with a volume path, or a storage pool identifier and volume name.

```
# 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
```

Storage pools

11.1. Creating storage pools

11.1.1. Dedicated storage device-based storage pools

This section covers dedicating storage devices to guests.

Problems with dedicated disks

Guests should not be given write access to whole disks or block devices (for example, **`/dev/sdb`**). Use partitions (for example, **`/dev/sdb1`**) or LVM volumes.

If you pass an entire block device to the guest, the guest will likely partition it or create its own LVM groups on it. This can cause the host to detect these partitions or LVM groups and cause errors.

11.1.1.1. Creating a dedicated disk storage pool using virsh

This procedure creates a new storage pool using a dedicated disk device with the **`virsh`** command.

Warning

Dedicating a disk to a storage pool will reformat and erase all data presently stored on the disk device. Back up the storage device before commencing the following procedure.

1. **Create a GPT disk label on the disk**
   The disk must be relabeled with a *GUID Partition Table* (GPT) disk label. GPT disk labels allow for creating a large numbers of partitions, up to 128 partitions, on each device. GPT partition tables can store partition data for far more partitions than the **`msdos`** partition table.

   ```
   # 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. **Create the storage pool configuration file**
   Create a temporary XML text file containing the storage pool information required for the new device.

   The file must be in the format shown below, and contain the following fields:

<name>guest_images_disk</name>
The *name* parameter determines the name of the storage pool. This example uses the name *guest_images_disk* in the example below.

<device path='*/dev/sdb*'/>
The *device* parameter with the *path* attribute specifies the device path of the storage device. This example uses the device */dev/sdb*.

<target> <path>*/dev*</path></target>
The file system *target* parameter with the *path* sub-parameter determines the location on the host file system to attach volumes created with this storage pool.

For example, sdb1, sdb2, sdb3. Using */dev/*, as in the example below, means volumes created from this storage pool can be accessed as */dev*/sdb1, */dev*/sdb2, */dev*/sdb3.

<format type='*gpt*'/>
The *format* parameter specifies the partition table type. his example uses the *gpt* in the example below, to match the GPT disk label type created in the previous step.

Create the XML file for the storage pool device with a text editor.

Example 11.1. Dedicated storage device storage pool

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

3. **Attach the device**
Add the storage pool definition using the **virsh pool-define** command with the XML configuration file created in the previous step.

```
# 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. **Start the storage pool**
Start the storage pool with the **virsh pool-start** command. Verify the pool is started with the **virsh pool-list --all** command.

```
# 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. **Turn on autostart**
   Turn on *`autostart`* for the storage pool. Autostart configures the `libvirtd` service to start the storage pool when the service starts.

```
# 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. **Verify the storage pool configuration**
   Verify the storage pool was created correctly, the sizes reported correctly, and the state reports as **`running`**.

```
# 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. **Optional: Remove the temporary configuration file**
   Remove the temporary storage pool XML configuration file if it is not needed.

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

A dedicated storage device storage pool is now available.

11.1.2. Partition-based storage pools

This section covers using a pre-formatted block device, a partition, as a storage pool.

For the following examples, a host has a 500GB hard drive (`/dev/sdc`) partitioned into one 500GB, ext4 formatted partition (`/dev/sdc1`). We set up a storage pool for it using the procedure below.

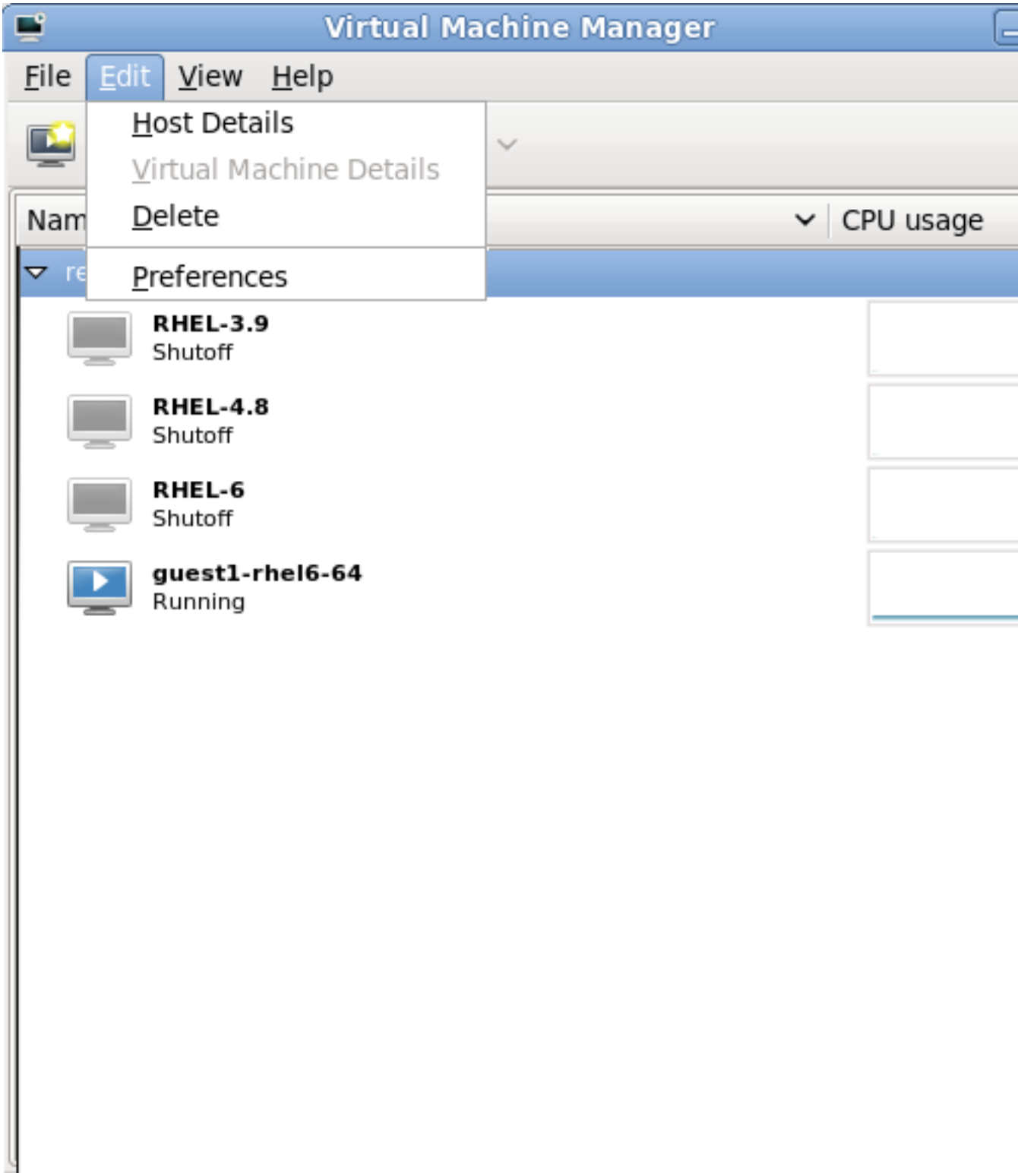
11.1.2.1. Creating a partition-based storage pool using virt-manager

This procedure creates a new storage pool using a partition of a storage device.

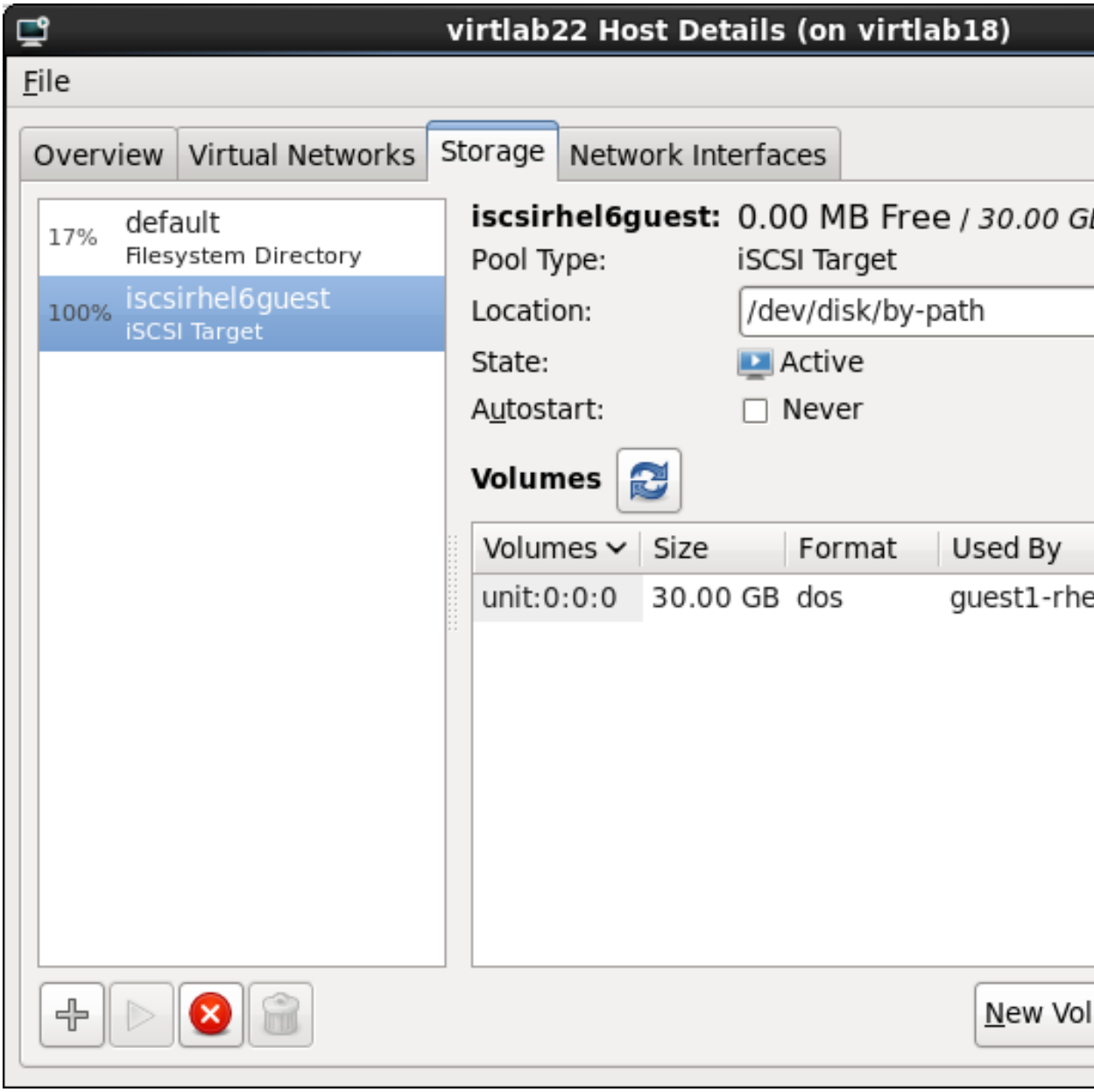
Procedure 11.1. Creating a partition-based storage pool with virt-manager

1. **Open the storage pool settings**
   a. In the **`virt-manager`** graphical interface, select the host from the main window.

      Open the **Edit** menu and select **Host Details**

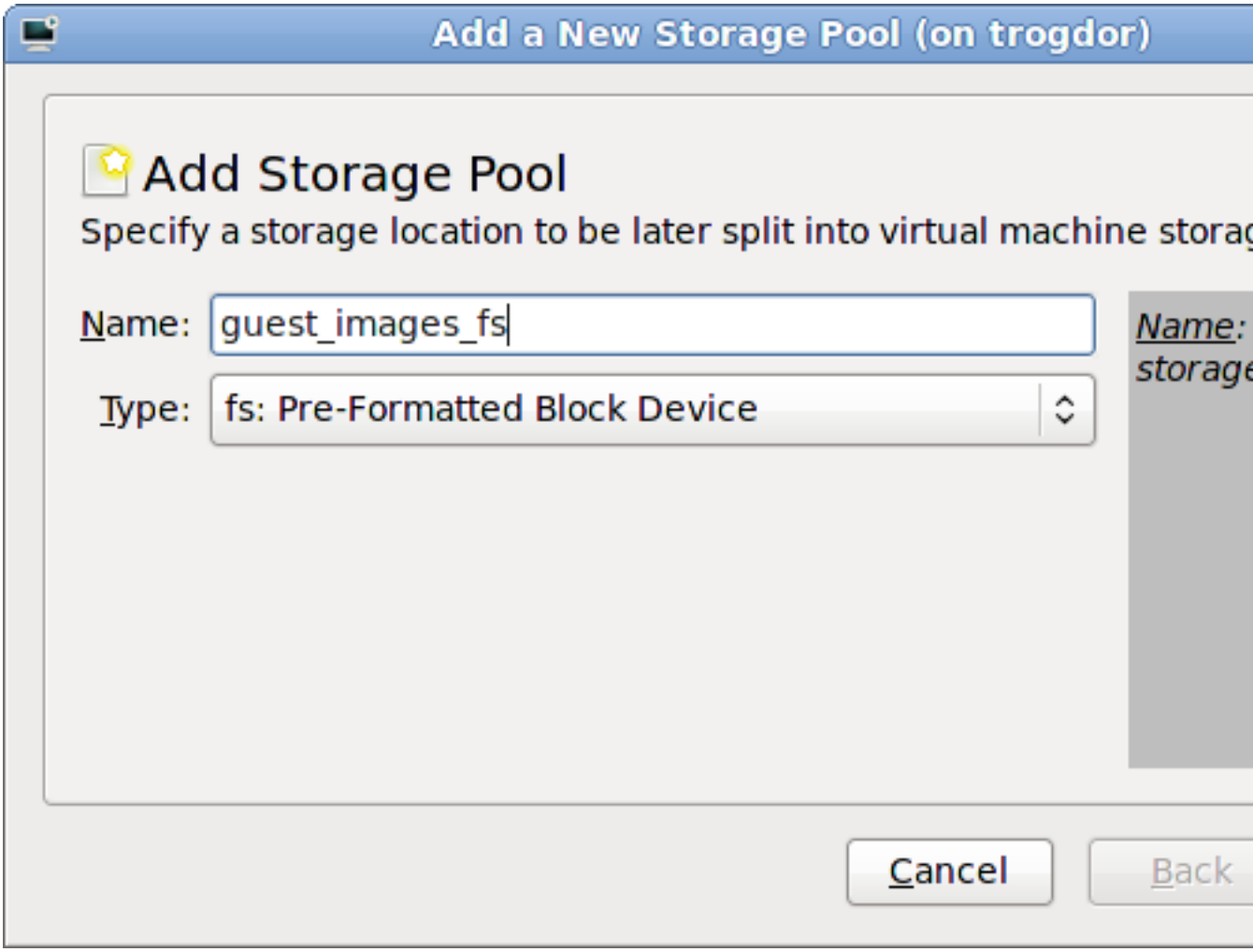


b. Click on the **Storage** tab of the **Host Details** window.



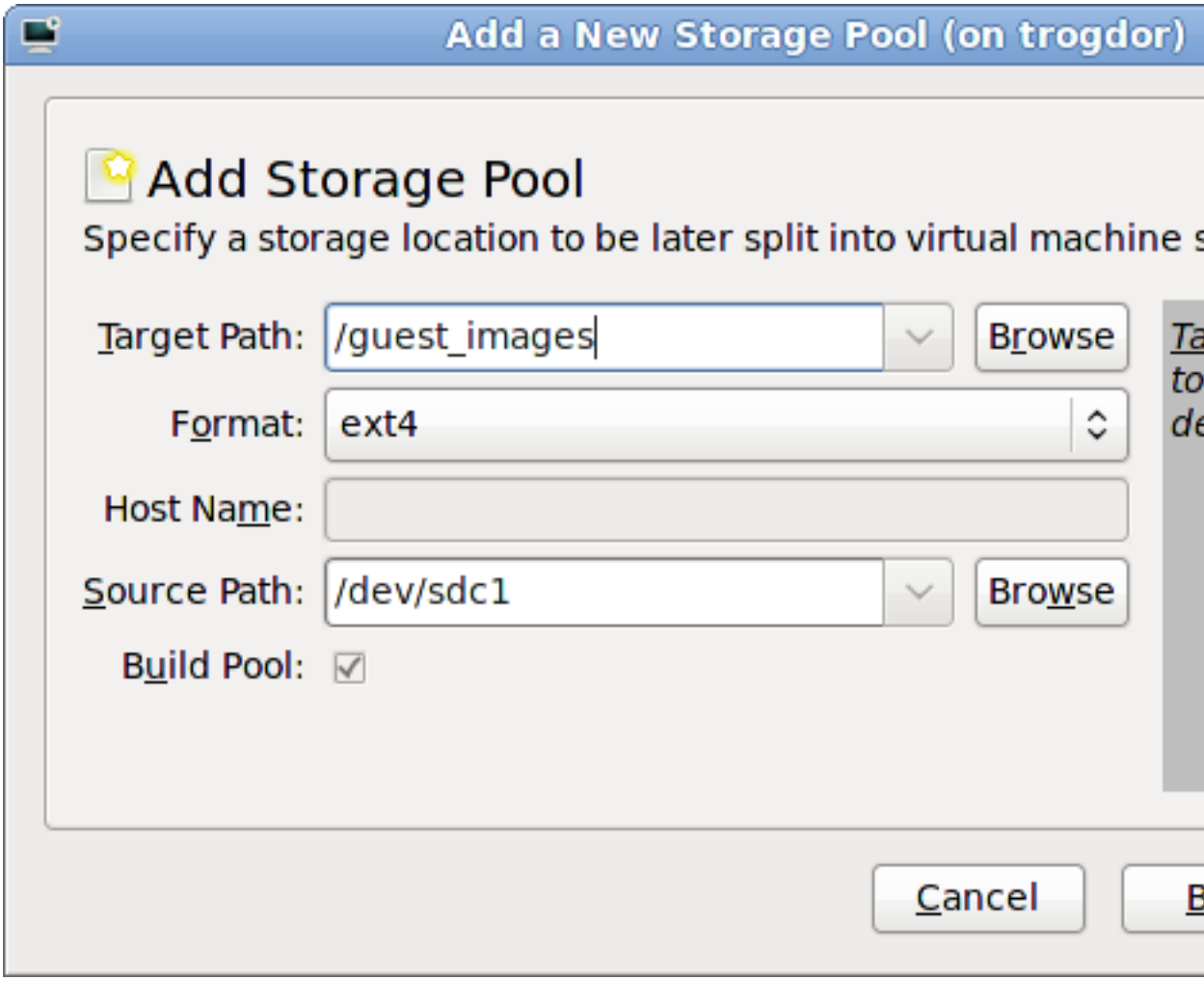
2. **Create the new storage pool**
   a. **Add a new pool (part 1)**
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

   Choose a **Name** for the storage pool. This example uses the name *guest_images_fs*. Change the **Type** to **fs: Pre-Formatted Block Device**.



Press the **Forward** button to continue.

b. **Add a new pool (part 2)**
Change the **Target Path**, **Format**, and **Source Path** fields.



Target Path
: Enter the location to mount the source device for the storage pool in the **Target Path** field. If the location does not already exist, **`virt-manager`** will create the directory.

Format
: Select a format from the **Format** list. The device is formatted with the selected format.

  This example uses the *`ext4`* file system, the default Red Hat Enterprise Linux file system.

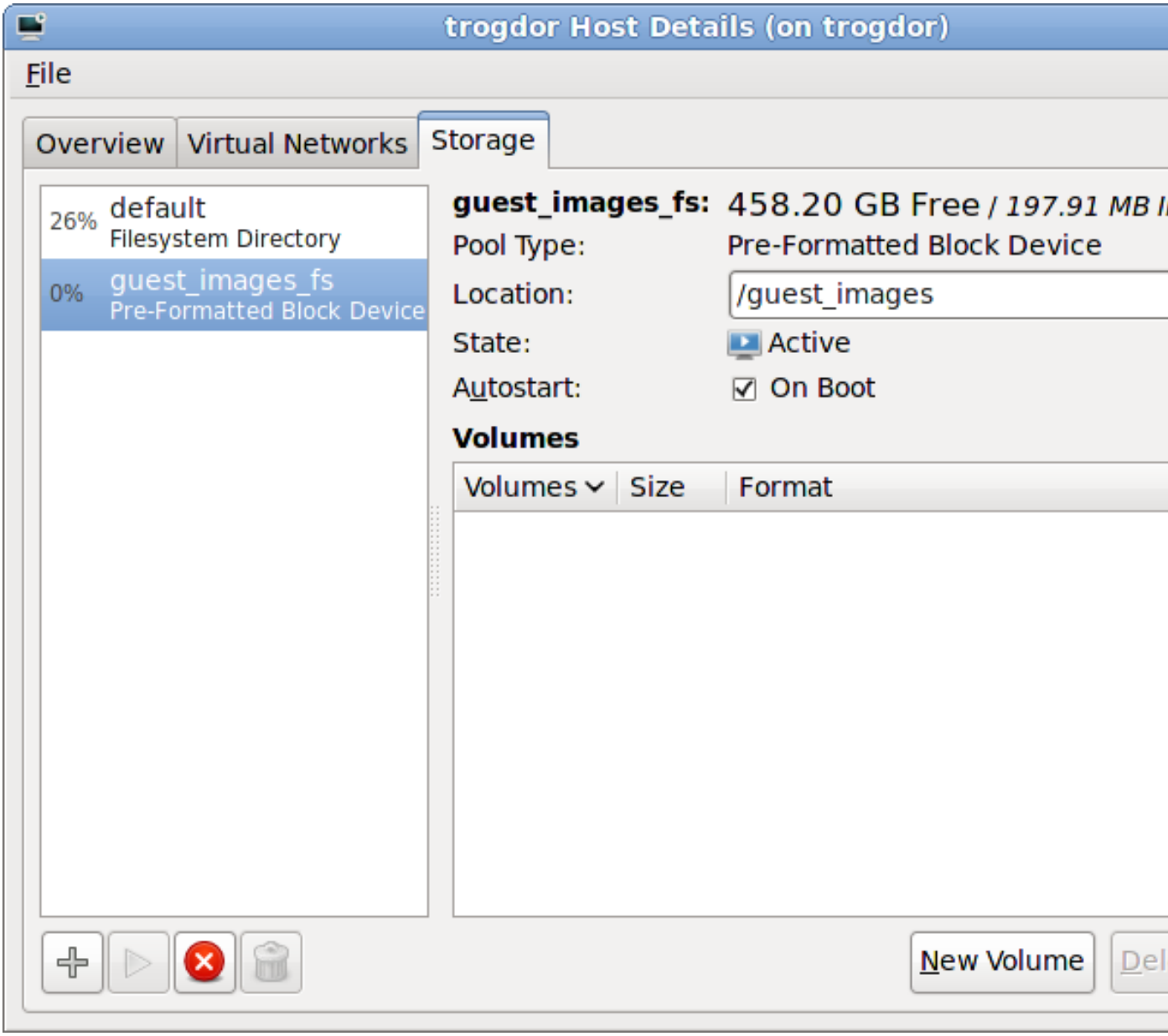
Source Path
: Enter the device in the **`Source Path`** field.

  This example uses the */dev/sdc1* device.

Verify the details and press the **Finish** button to create the storage pool.

3. **Verify the new storage pool**
   The new storage pool appears in the storage list on the left after a few seconds. Verify the size is reported as expected, *`458.20 GB Free`* in this example. Verify the **State** field reports the new storage pool as *`Active`*.

   Select the storage pool. In the **Autostart** field, click the **On Boot** checkbox. This will make sure the storage device starts whenever the `libvirtd` service starts.



The storage pool is now created, close the **Host Details** window.

11.1.2.2. Creating a partition-based storage pool using virsh

This section covers creating a partition-based storage pool with the **`virsh`** command.

Security warning

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

Procedure 11.2. Creating pre-formatted block device storage pools using virsh

1. **Create the storage pool definition**

   Use the virsh **pool-define-as** command to create a new storage pool definition. There are three options that must be provided to define a pre-formatted disk as a storage pool:

   Partition name

   The *name* parameter determines the name of the storage pool. This example uses the name *guest_images_fs* in the example below.

   device

   The *device* parameter with the *path* attribute specifies the device path of the storage device. This example uses the partition */dev/sdc1*.

   mountpoint

   The *mountpoint* on the local file system where the formatted device will be mounted. If the mount point directory does not exist, the **virsh** command can create the directory.

   The directory */guest_images* is used in this example.

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

   The new pool and mount points are now created.

2. **Verify the new pool**

   List the present storage pools.

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

3. **Ceate the mount point**

   Use the **virsh pool-build** command to create a mount point for a pre-formatted file system storage pool.

```
# 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. **Start the storage pool**

   Use the **virsh pool-start** command to mount the file system onto the mount point and make the pool available for use.

```
# 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. **Turn on autostart**
   By default, a storage pool is defined with **virsh** is not set to automatically start each time `libvirtd` starts. Turn on automatic start with the **virsh pool-autostart** command. The storage pool is now automatically started each time `libvirtd` starts.

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

```
# virsh pool-info guest_images_fs
Name:           guest_images_fs
UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
State:          running
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
```

11.1.3. Directory-based storage pools

This section covers storing guests in a directory on the host.

Directory-based storage pools can be created with **virt-manager** or the **virsh** command line tools.

11.1.3.1. Creating a directory-based storage pool with virt-manager

1. **Create the local directory**
   a. **Optional: Create a new directory for the storage pool**
      Create the directory on the host for the storage pool. An existing directory can be used if permissions and SELinux are configured correctly. This example uses a directory named */guest_images*.

```
# mkdir /guest_images
```

b. **Set directory ownership**
Change the user and group ownership of the directory. The directory must be owned by the root user.

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

c. **Set directory permissions**
Change the file permissions of the directory.

```
# chmod 700 /guest_images
```

d. **Verify the changes**
Verify the permissions were modified. The output shows a correctly configured empty directory.

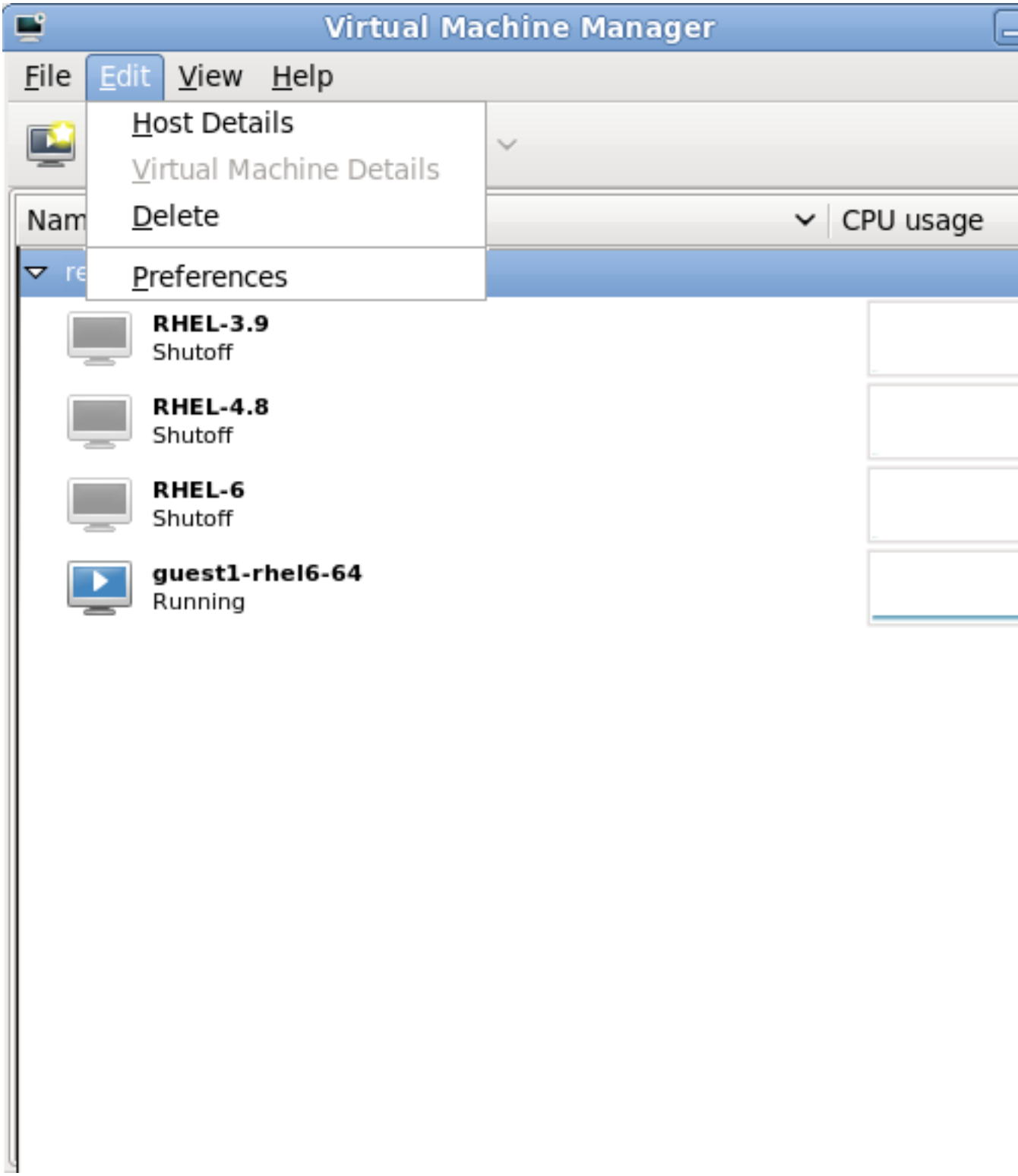
```
# ls -la /guest_images
total 8
drwx------.  2 root root 4096 May 28 13:57 .
dr-xr-xr-x. 26 root root 4096 May 28 13:57 ..
```

2. **Configure SELinux file contexts**
Configure the correct SELinux context for the new directory.

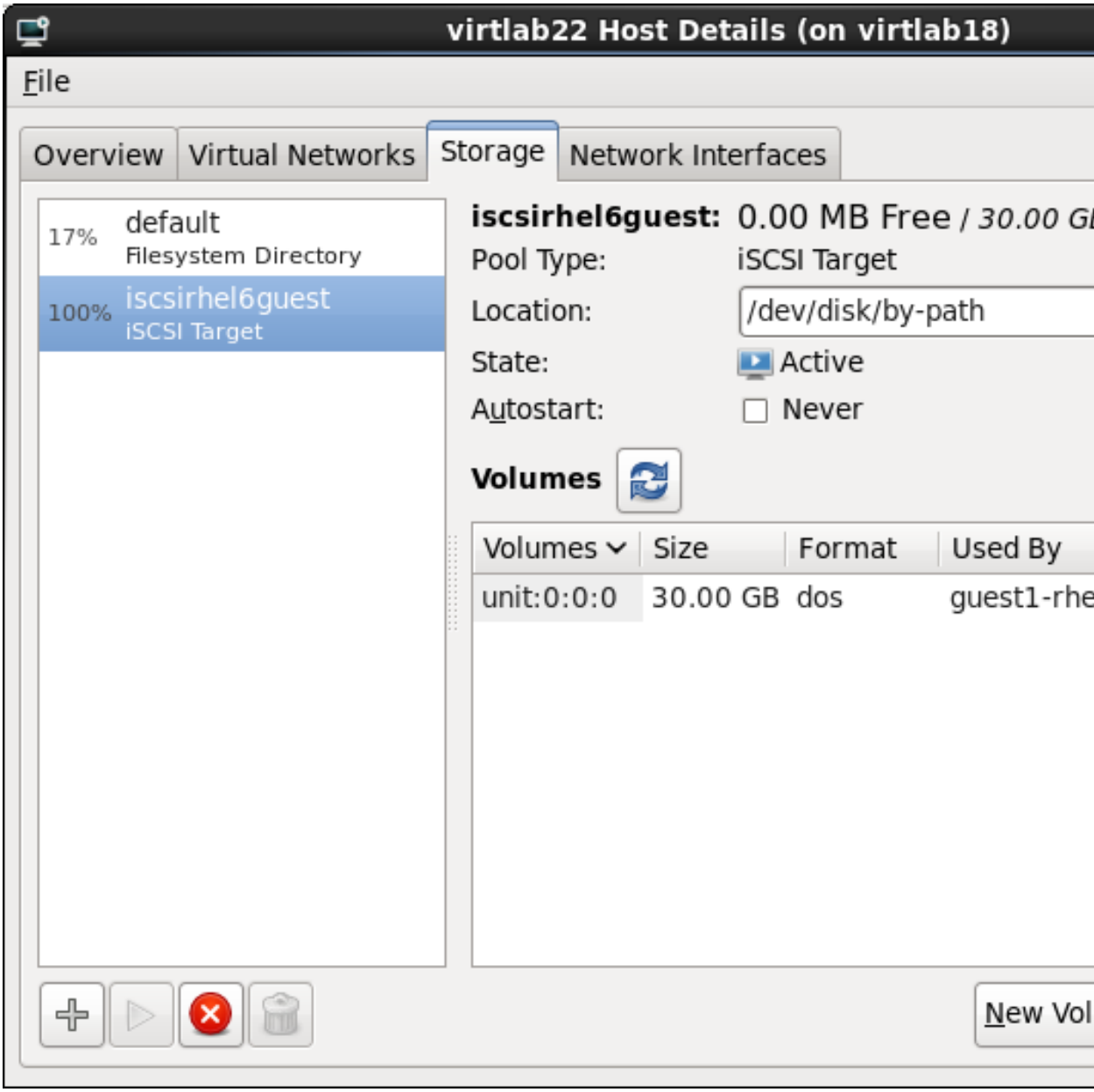
```
# semanage fcontext -a -t virt_image_t /guest_images
```

3. **Open the storage pool settings**
a. In the **`virt-manager`** graphical interface, select the host from the main window.

Open the **Edit** menu and select **Host Details**



b. Click on the **Storage** tab of the **Host Details** window.

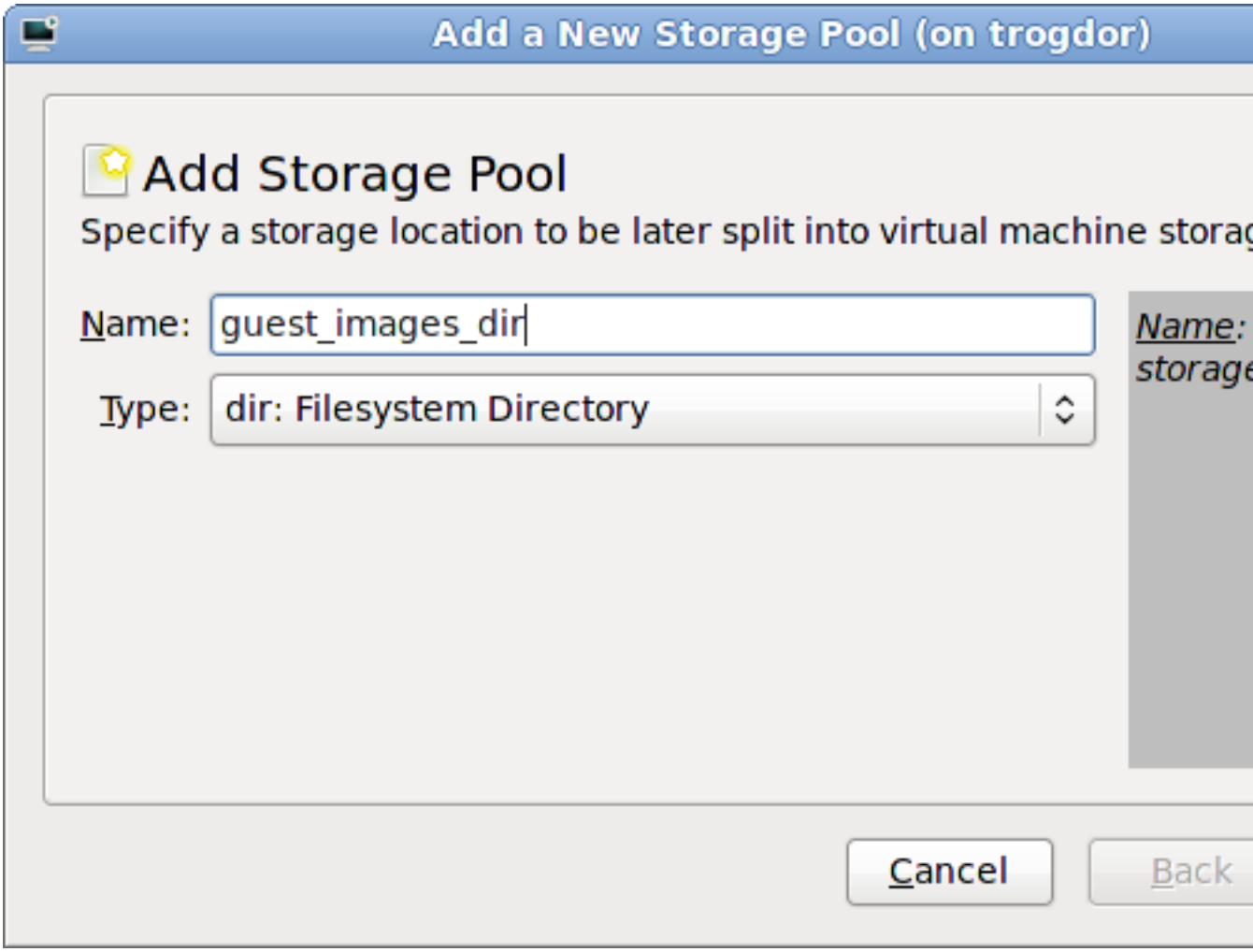


4. **Create the new storage pool**

   a. **Add a new pool (part 1)**

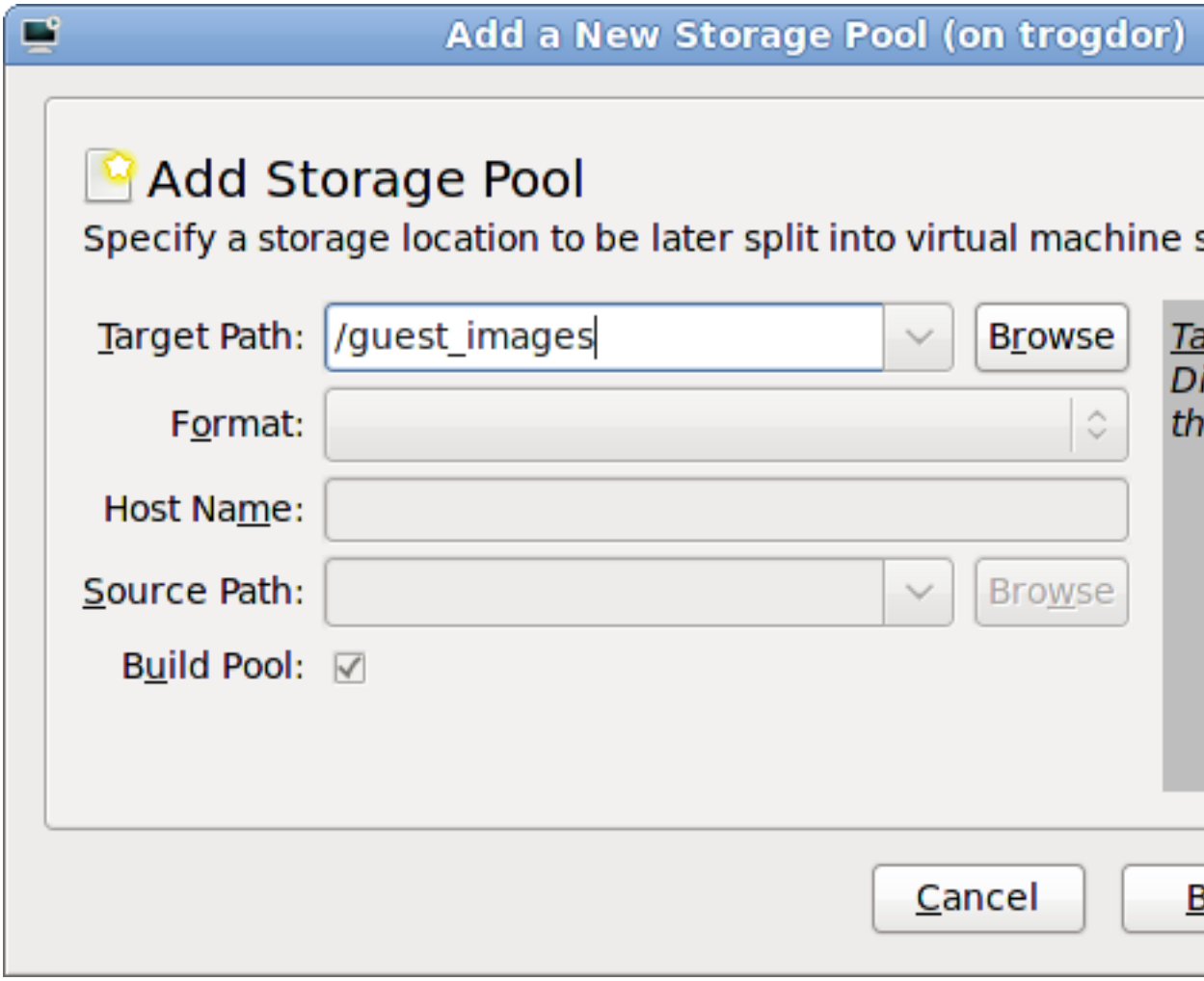
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

   Choose a **Name** for the storage pool. This example uses the name *`guest_images_dir`*. Change the **Type** to **`dir: Filesystem Directory`**.



Press the **Forward** button to continue.

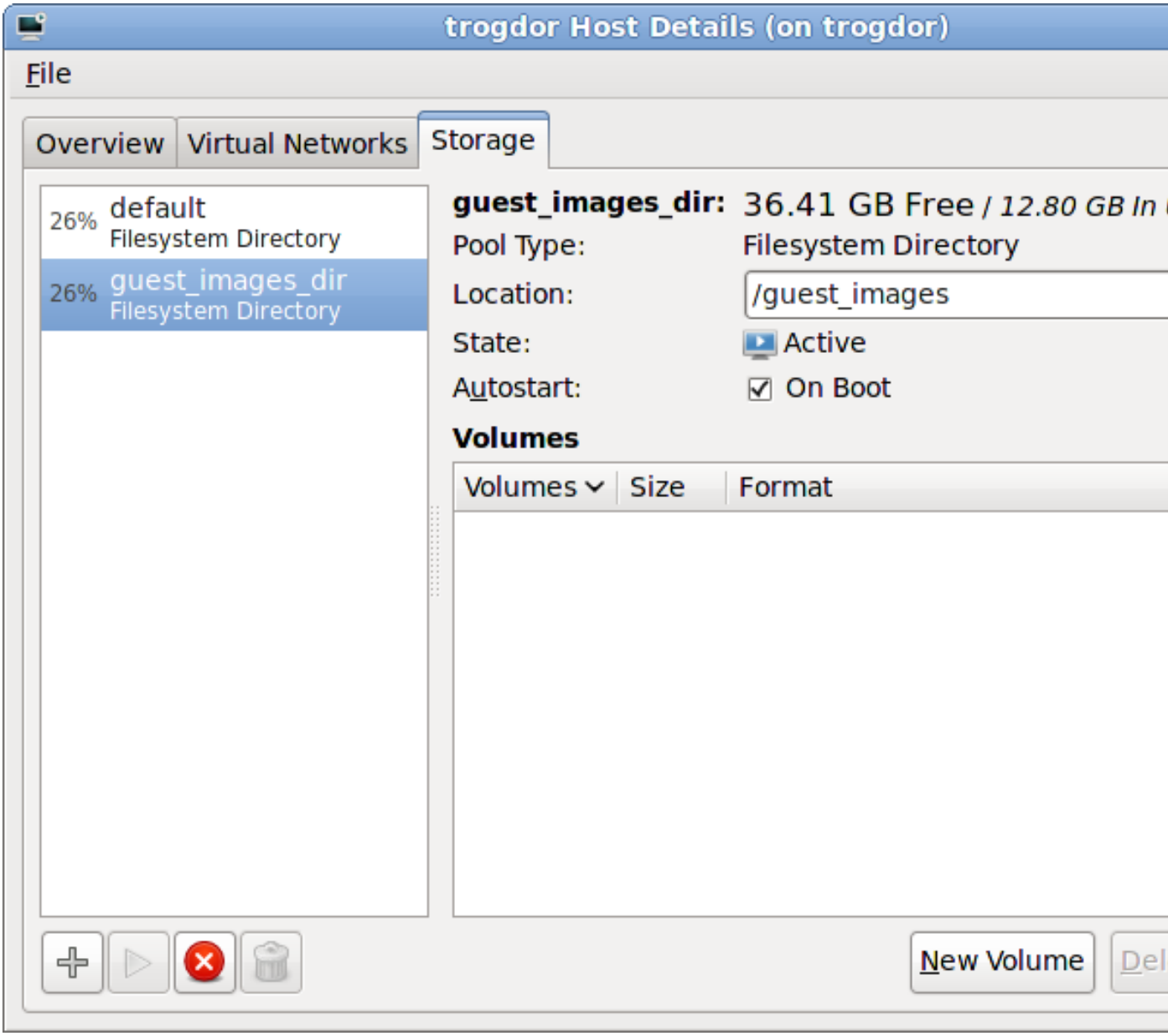
b. **Add a new pool (part 2)**
Change the **Target Path** field. This example uses */guest_images*.



Verify the details and press the **Finish** button to create the storage pool.

5. **Verify the new storage pool**
   The new storage pool appears in the storage list on the left after a few seconds. Verify the size is reported as expected, *`36.41 GB Free`* in this example. Verify the **State** field reports the new storage pool as *`Active`*.

   Select the storage pool. In the **Autostart** field, click the **On Boot** checkbox. This will make sure the storage pool starts whenever the `libvirtd` service starts.



The storage pool is now created, close the **Host Details** window.

11.1.3.2. Creating a directory-based storage pool with virsh

1. **Create the storage pool definition**
   Use the **`virsh pool-define-as`** command to define a new storage pool. There are two options required for creating directory-based storage pools:

   - The **`name`** of the storage pool.

     This example uses the name *`guest_images_dir`*. All further **`virsh`** commands used in this example use this name.

   - The **`path`** to a file system directory for storing guest image files. If this directory does not exist, **`virsh`** will create it.

     This example uses the */`guest_images`* directory.

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

2. **Verify the storage pool is listed**
   Verify the storage pool object is created correctly and the state reports it as **`inactive`**.

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

3. **Create the local directory**
   Use the **`virsh pool-build`** command to build the directory-based storage pool. **`virsh pool-build`** sets the required permissions and SELinux settings for the directory and creates the directory if it does not exist.

```
# virsh pool-build guest_images_dir
Pool guest_images_dir 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_dir     inactive   no
```

4. **Start the storage pool**
   Use the virsh command **`pool-start`** for this. **`pool-start`** enables a directory storage pool, allowing it to be used for volumes and guests.

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

5. **Turn on autostart**
   Turn on *`autostart`* for the storage pool. Autostart configures the `libvirtd` service to start the storage pool when the service starts.

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

6. **Verify the storage pool configuration**
   Verify the storage pool was created correctly, the sizes reported correctly, and the state reports as **`running`**.

```
# virsh pool-info guest_images_dir
Name:           guest_images_dir
UUID:           779081bf-7a82-107b-2874-a19a9c51d24c
State:          running
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 ..
#
```

A directory-based storage pool is now available.

11.1.4. LVM-based storage pools

This chapter covers using LVM volume groups as storage pools.

LVM-based storage groups provide the full flexibility of LVM.

LVM

Please refer to the *Red Hat Enterprise Linux Storage Administration Guide* for more details on LVM.

Warning

LVM-based storage pools require a full disk partition. If activating a new partition/device with these procedures, the partition will be formatted and all data will be erased. If using the host's existing Volume Group (VG) nothing will be erased. It is recommended to back up the storage device before commencing the following procedure.

11.1.4.1. Creating an LVM-based storage pool with virt-manager

LVM-based storage pools can use existing LVM volume groups or create new LVM volume groups on a blank partition.

1. **Optional: Create new partition for LVM volumes**
   These steps describe how to create a new partition and LVM volume group on a new hard disk drive.

Warning

This procedure will remove all data from the selected storage device.

a. **Create a new partition**
Use the **fdisk** command to create a new disk partition from the command line. The following example creates a new partition that uses the entire disk on the storage device **/dev/sdb**.

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

Press *n* for a new partition.

b. Press *p* for a primary partition.

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

c. Choose an available partition number. In this example the first partition is chosen by entering *1*.

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

d. Enter the default first cylinder by pressing *Enter*.

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

e. Select the size of the partition. In this example the entire disk is allocated by pressing *Enter*.

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

f. Set the type of partition by pressing *t*.

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

g. Choose the partition you created in the previous steps. In this example, the partition number is *1*.

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

h. Enter *8e* for a Linux LVM partition.

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

i. write changes to disk and quit.

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

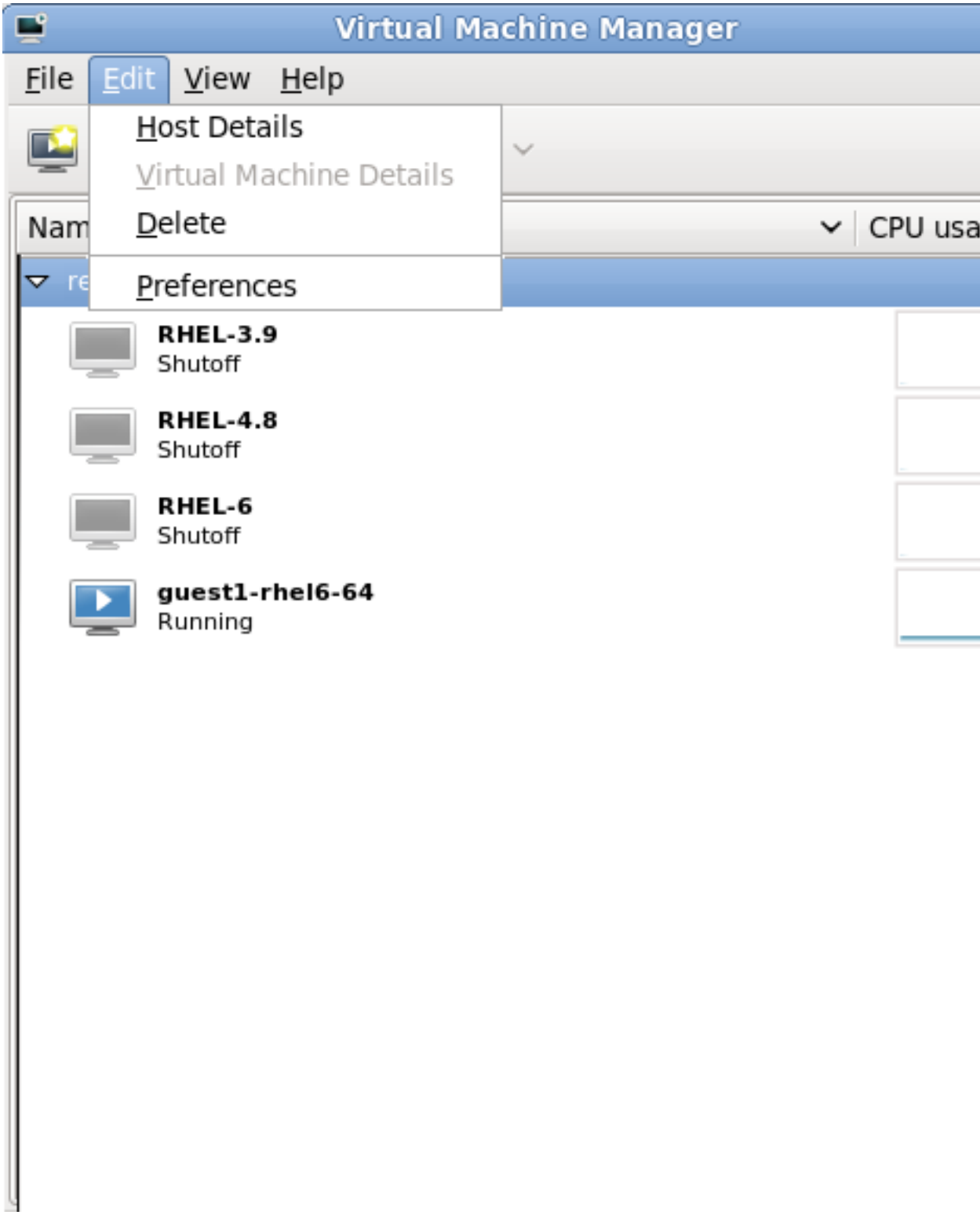
j. **Create a new LVM volume group**
Create a new LVM volume group with the vgcreate command. This example creates a volume group named *guest_images_lvm*.

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

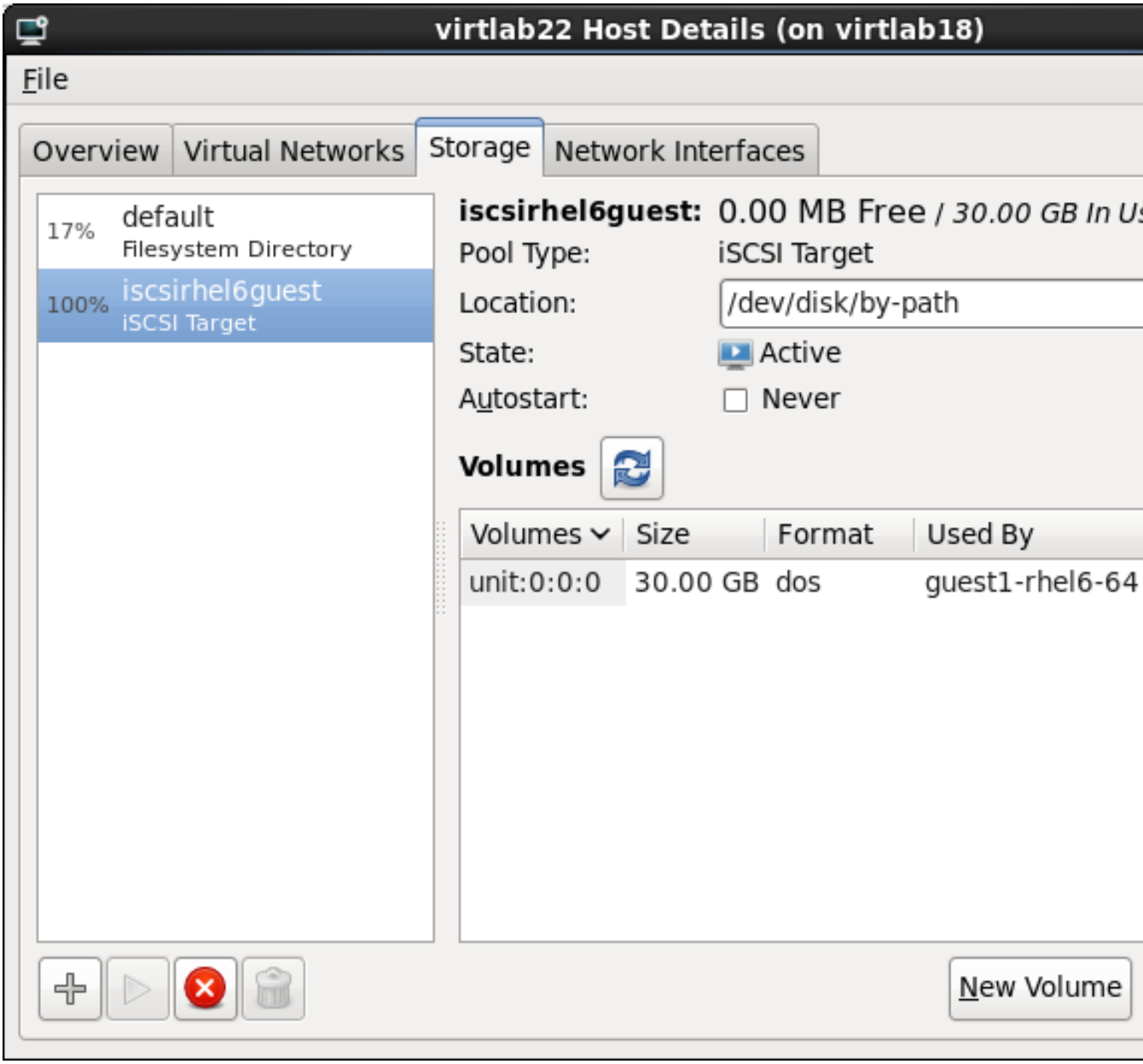
The new LVM volume group, *guest_images_lvm*, can now be used for an LVM-based storage pool.

2. **Open the storage pool settings**
a. In the **virt-manager** graphical interface, select the host from the main window.

Open the **Edit** menu and select **Host Details**



b. Click on the **Storage** tab of the **Host Details** window.

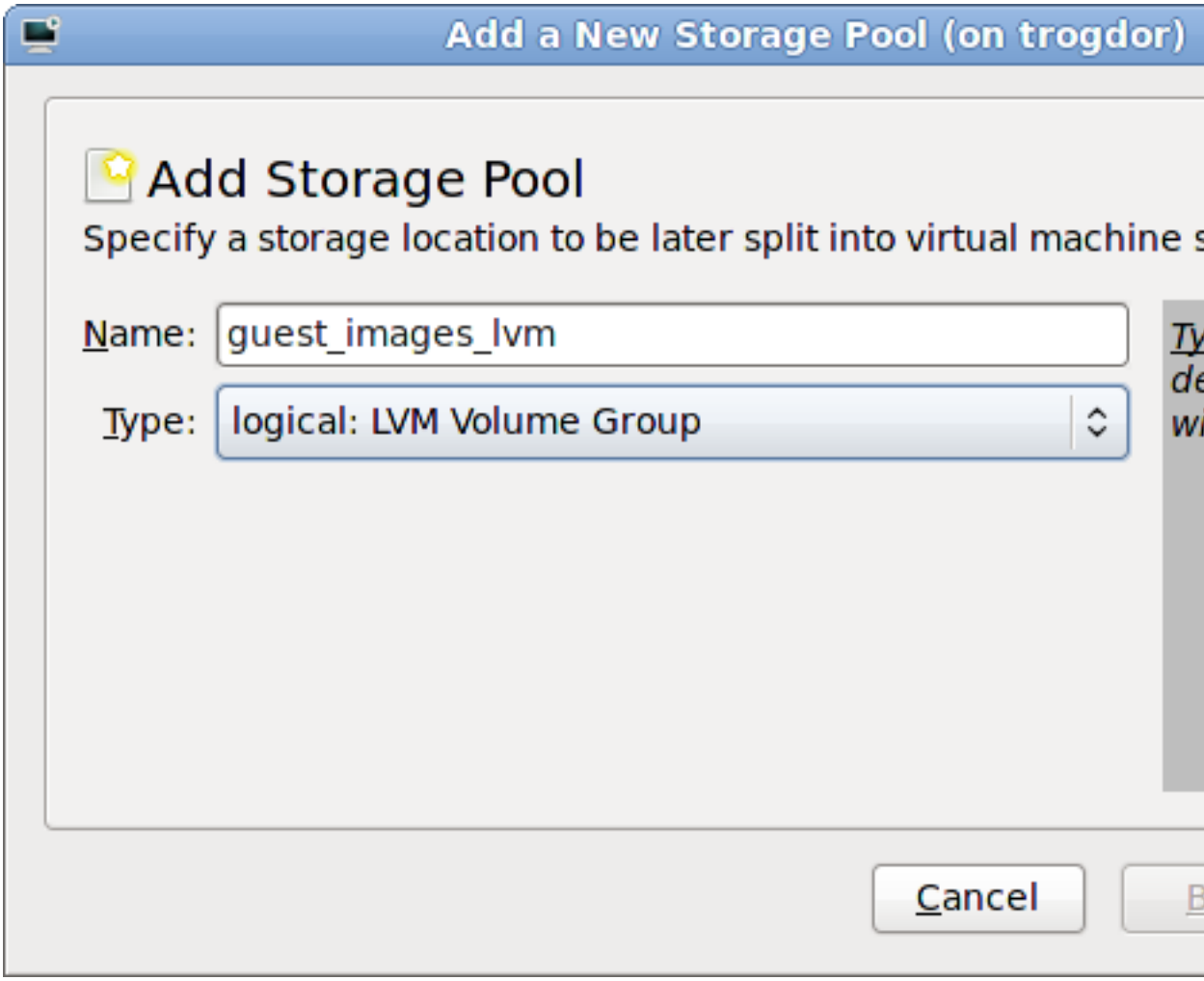


3. **Create the new storage pool**

   a. **Start the Wizard**

   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

   Choose a **Name** for the storage pool. We use *guest_images_lvm* for this example. Then change the **Type** to **logical: LVM Volume Group**, and



Press the **Forward** button to continue.

b. **Add a new pool (part 2)**
Change the **Target Path** field. This example uses */guest_images*.

Now fill in the **Target Path** and **Source Path** fields, then tick the **Build Pool** check box.

- Use the **Target Path** field to *either* select an existing LVM volume group or as the name for a new volume group. The default format is **/dev/***storage_pool_name*.

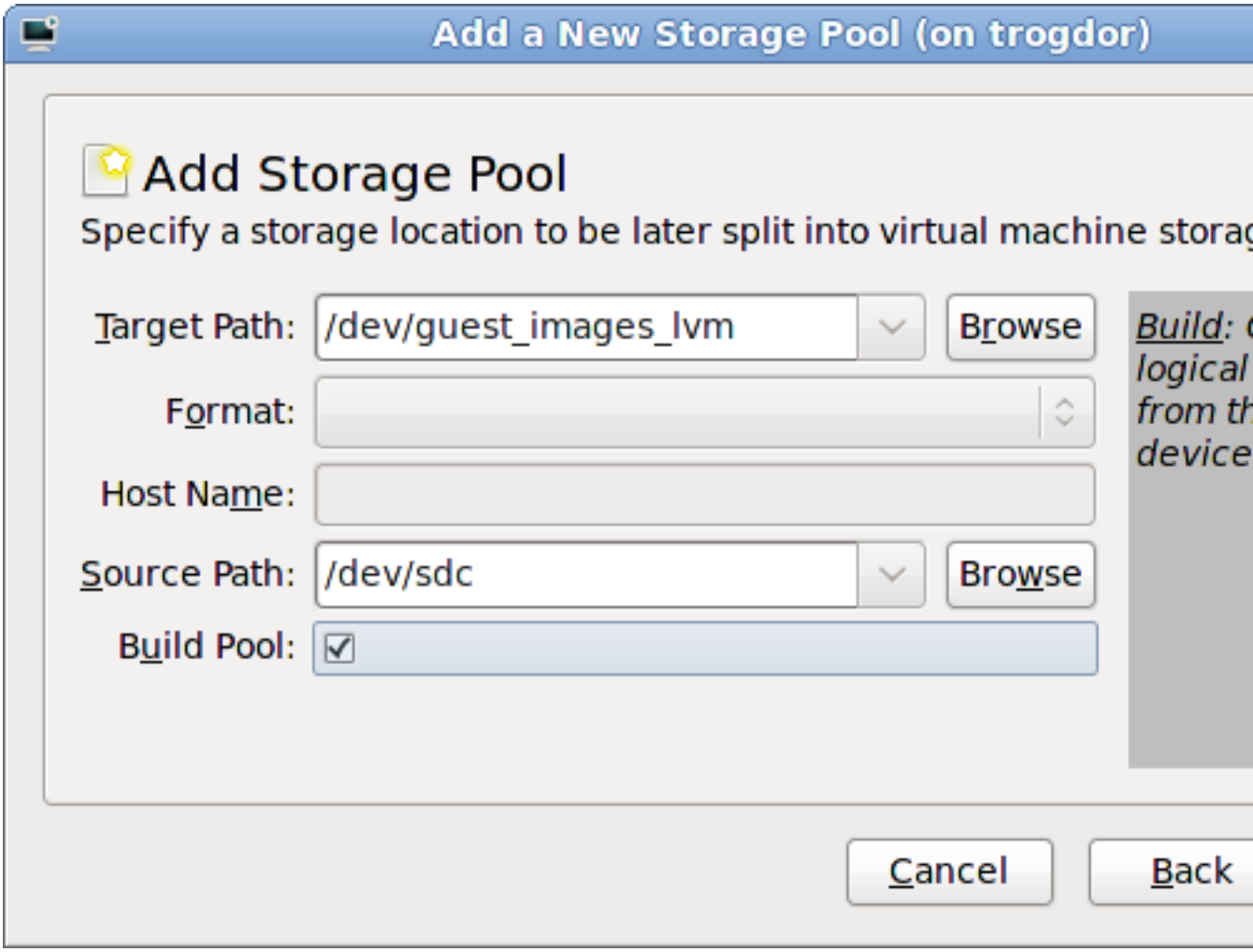
  This example uses a new volume group named */dev/guest_images_lvm*.

- The **Source Path** field is optional if an existing LVM volume group is used in the **Target Path**.

  For new LVM volume groups, input the location of a storage device in the **Source Path** field. This example uses a blank partition */dev/sdc*.

- The **Build Pool** checkbox instructs **virt-manager** to create a new LVM volume group. If you are using an existing volume group you should not select the **Build Pool** checkbox.

  This example is using a blank partition to create a new volume group so the **Build Pool** checkbox must be selected.



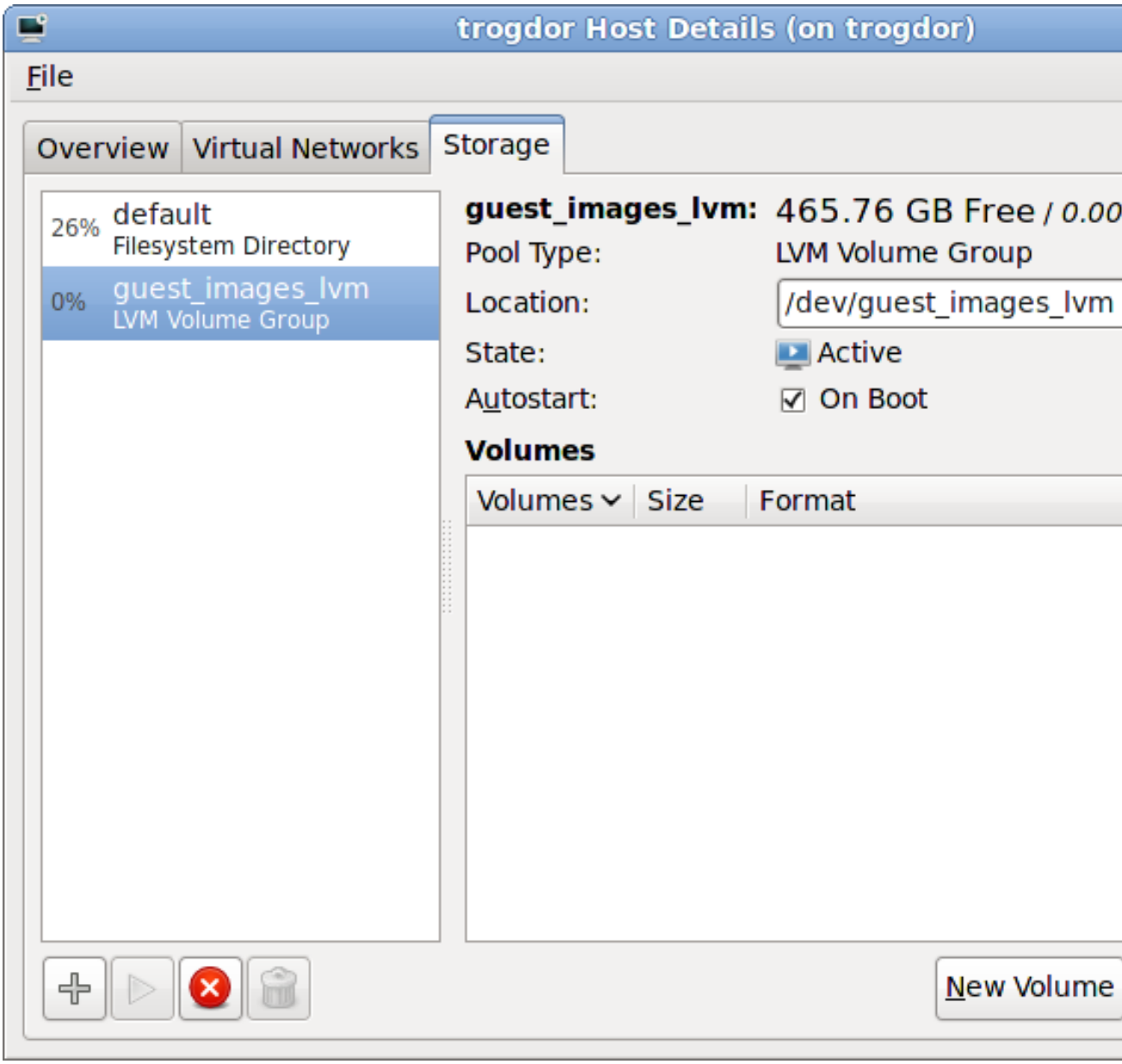
Verify the details and press the **Finish** button format the LVM volume group and create the storage pool.

c. **Confirm the device to be formatted**
A warning message appears.

Press the **Yes** button to proceed to erase all data on the storage device and create the storage pool.

4. **Verify the new storage pool**
   The new storage pool will appear in the list on the left after a few seconds. Verify the details are what you expect, *`465.76 GB Free`* in our example. Also verify the **State** field reports the new storage pool as *`Active`*.

   It is generally a good idea to have the **Autostart** check box enabled, to ensure the storage pool starts automatically with libvirtd.



   Close the Host Details dialog, as the task is now complete.

11.1.4.2. Creating an LVM-based storage pool with virsh

This section outlines the steps required to create an LVM-based storage pool with the **`virsh`** command. It uses the example of a pool named **guest_images_lvm** from a single drive (**`/dev/sdc`**). This is only an example and your settings should be substituted as appropriate.

Procedure 11.3. Creating an LVM-based storage pool with virsh

1. Define the pool name **guest_images_lvm**.

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

2. Build the pool according to the specified name.

```
# virsh pool-build guest_images_lvm

Pool guest_images_lvm built
```

3. Initialize the new pool.

```
# virsh pool-start guest_images_lvm

Pool guest_images_lvm started
```

4. Show the volume group information with the **vgs** command.

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

5. Set the pool to start automatically.

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

6. List the available pools with the **virsh** command.

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

7. The following commands demonstrate the creation of three volumes (*volume1*, *volume2* and *volume3*) within this pool.

```
# 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. List the available volumes in this pool with the **virsh** command.

```
# virsh vol-list guest_images_lvm
Name                 Path
-----------------------------------------
volume1              /dev/libvirt_lvm/volume1
volume2              /dev/libvirt_lvm/volume2
volume3              /dev/libvirt_lvm/volume3
```

9. The following two commands (**lvscan** and **lvs**) display further information about the newly created volumes.

```
# 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   Origin Snap%  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
```

11.1.5. iSCSI-based storage pools

This section covers using iSCSI-based devices to store guests.

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

11.1.5.1. Configuring a software iSCSI target

The *scsi-target-utils* package provides a tool for creating software-backed iSCSI targets.

Procedure 11.4. Creating an iSCSI target

1. **Install the required packages**
   Install the *scsi-target-utils* package and all dependencies

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

2. **Start the tgtd service**
   The **tgtd** service hosts SCSI targets and uses the iSCSI protocol to host targets. Start the **tgtd** service and make the service persistent after restarting with the **chkconfig** command.

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

3. **Optional: Create LVM volumes**
   LVM volumes are useful for iSCSI backing images. LVM snapshots and resizing can be beneficial for guests. This example creates an LVM image named *virtimage1* on a new volume group named *virtstore* on a RAID5 array for hosting guests with iSCSI.

a. **Create the RAID array**
Creating software RAID5 arrays is covered by the *Red Hat Enterprise Linux Deployment Guide*.

b. **Create the LVM volume group**
Create a volume group named *virtstore* with the **vgcreate** command.

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

c. **Create a LVM logical volume**
Create a logical volume group named *virtimage1* on the *virtstore* volume group with a size of 20GB using the **lvcreate** command.

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

The new logical volume, *virtimage1*, is ready to use for iSCSI.

4. **Optional: Create file-based images**
File-based storage is sufficient for testing but is not recommended for production environments or any significant I/O activity. This optional procedure creates a file based imaged named *virtimage2.img* for an iSCSI target.

a. **Create a new directory for the image**
Create a new directory to store the image. The directory must have the correct SELinux contexts.

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

b. **Create the image file**
Create an image named *virtimage2.img* with a size of 10GB.

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

c. **Configure SELinux file contexts**
Configure the correct SELinux context for the new image and directory.

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

The new file-based image, *virtimage2.img*, is ready to use for iSCSI.

5. **Create targets**
Targets can be created by adding a XML entry to the **/etc/tgt/targets.conf** file. The **target** attribute requires an iSCSI Qualified Name (IQN). The IQN is in the format:

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

Where:

- *yyyy-mm* represents the year and month the device was started (for example: *2010-05*);

- *reversed domain name* is the hosts domain name in reverse (for example *server1.example.com* in an IQN would be *com.example.server1*); and
- *optional identifier text* is any text string, without spaces, that assists the administrator in identifying devices or hardware.

This example creates iSCSI targets for the two types of images created in the optional steps on *server1.example.com* with an optional identifier *trial*. Add the following to the **/etc/tgt/targets.conf** file.

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

Ensure that the **/etc/tgt/targets.conf** file contains the **default-driver iscsi** line to set the driver type as iSCSI. The driver uses iSCSI by default.

Important

This example creates a globally accessible target without access control. Refer to the scsi-target-utils for information on implementing secure access.

6. **Restart the tgtd service**
   Restart the `tgtd` service to reload the configuration changes.

```
# service tgtd restart
```

7. **iptables configuration**
   Open port 3260 for iSCSI access with **iptables**.

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

8. **Verify the new targets**
   View the new targets to ensure the setup was success with the **tgt-admin --show** command.

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

Security warning

The ACL list is set to all. This allows all systems on the local network to access this device. It is recommended to set host access ACLs for production environments.

9. **Optional: Test discovery**
   Test whether the new iSCSI device is discoverable.

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

10. **Optional: Test attaching the device**
    Attach the new device (*iqn.2010-05.com.example.server1:iscsirhel6guest*) to determine whether the device can be attached.

```
# 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.
```

    Detach the device.

```
# 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.
```

An iSCSI device is now ready to use for virtualization.

11.1.5.2. Adding an iSCSI target to virt-manager

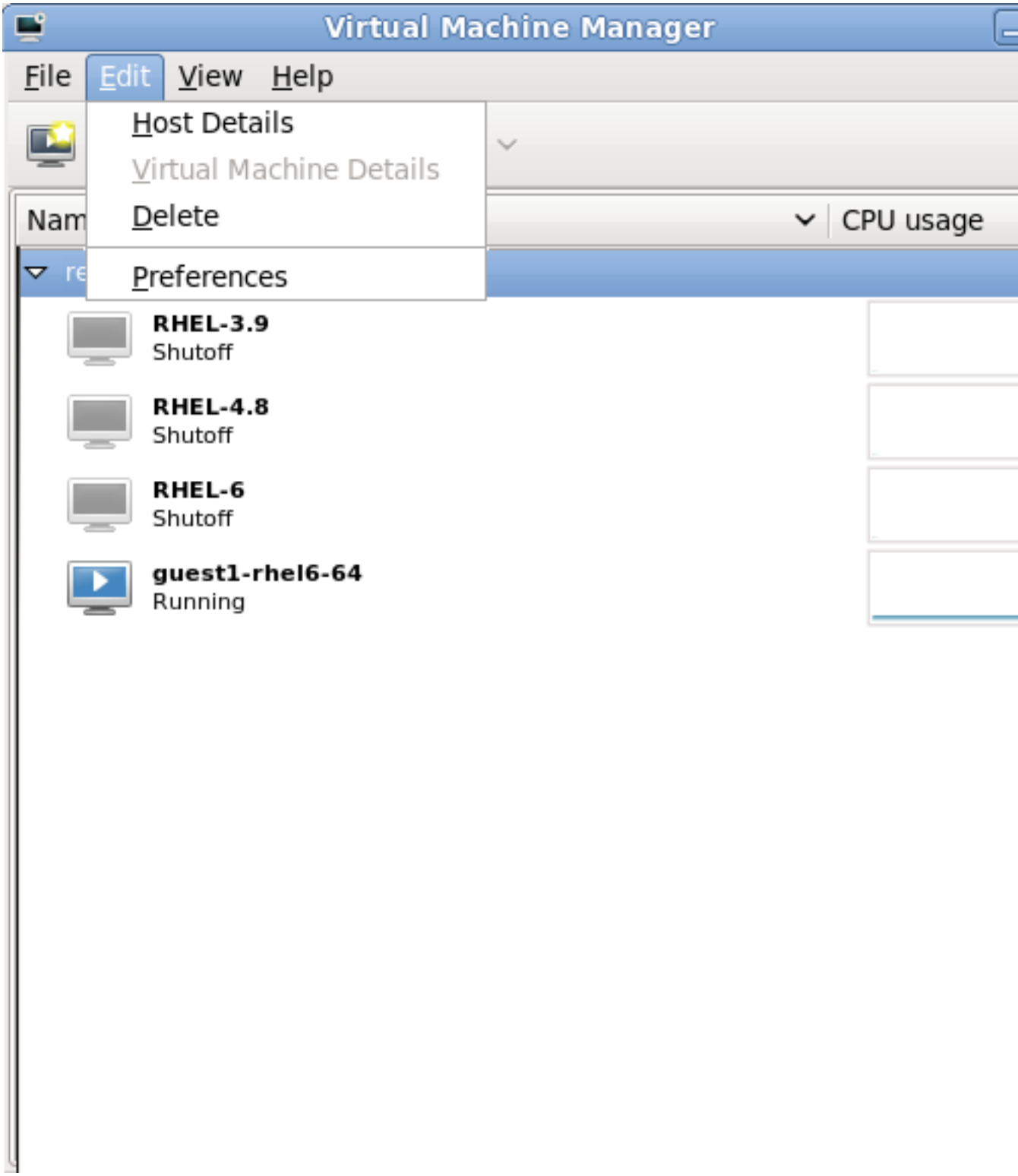
This procedure covers creating a storage pool with an iSCSI target in **`virt-manager`**.

Procedure 11.5. Adding an iSCSI device to virt-manager

1. **Open the host storage tab**
   Open the **Storage** tab in the **Host Details** window.

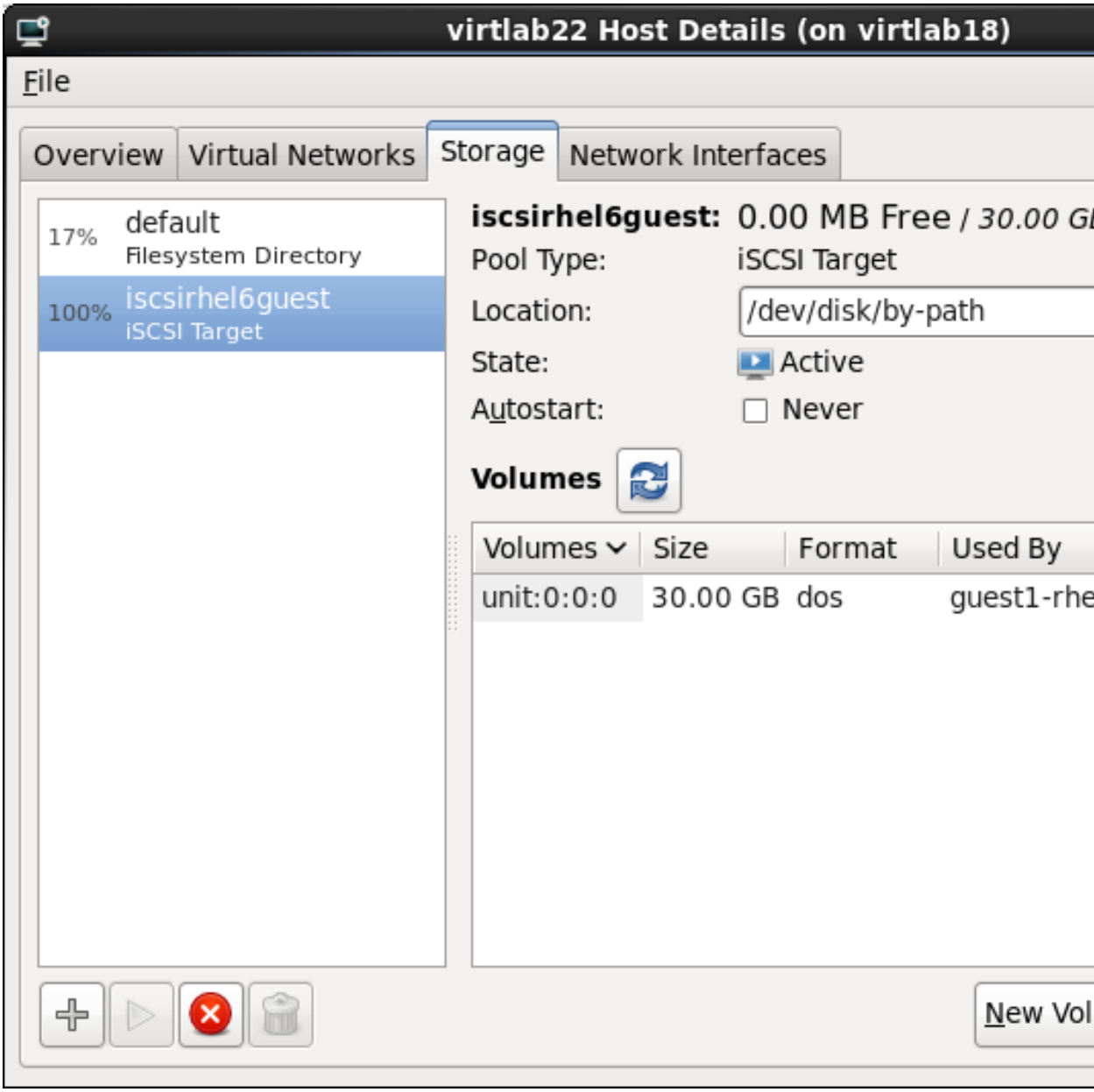
   a. Open **`virt-manager`**.

   b. Select a host from the main **`virt-manager`** window.

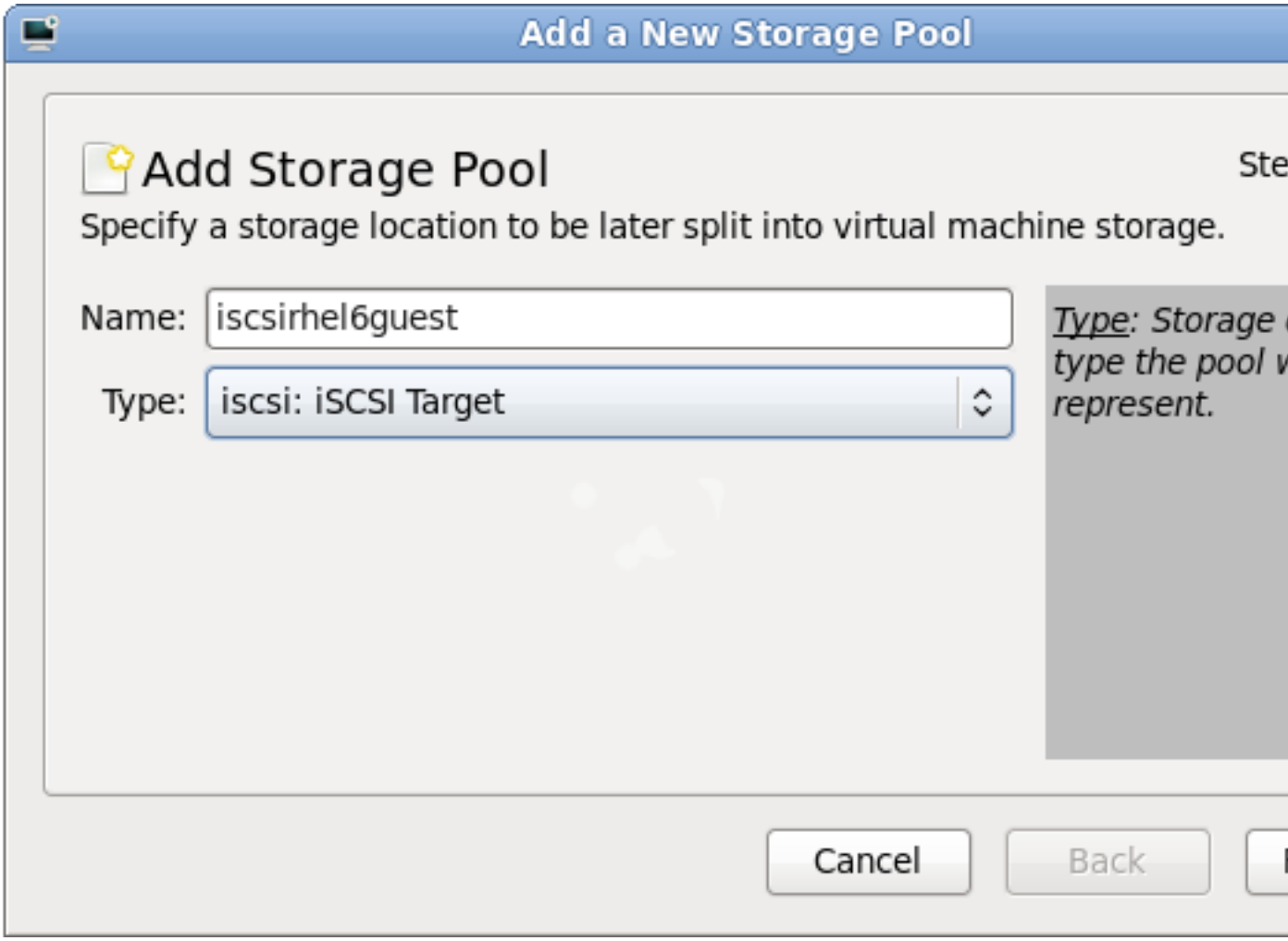


c. Open the Edit menu and select Host Details.

d. Click on the Storage tab of the Host Details window.



2. **Add a new pool (part 1)**
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

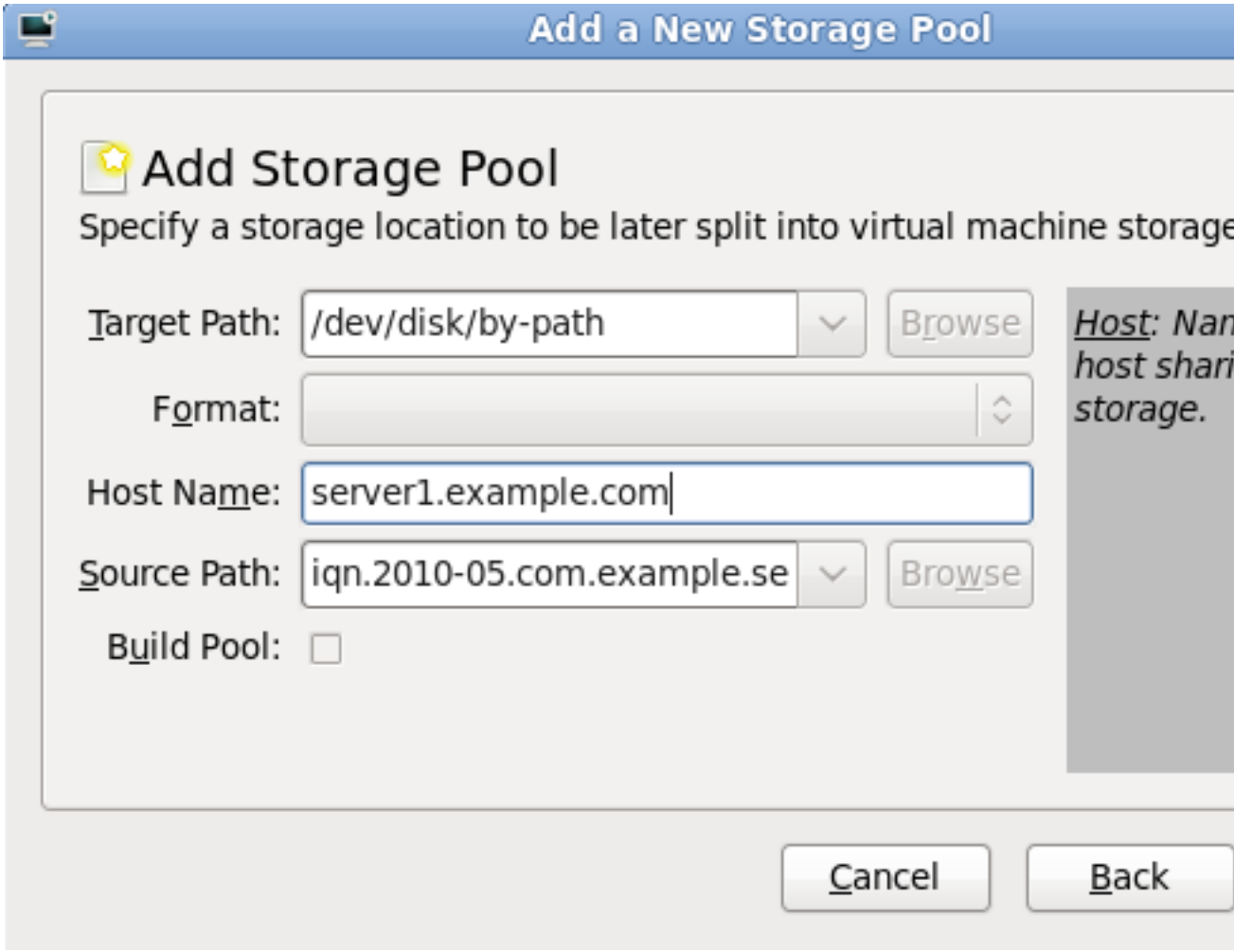


Choose a name for the storage pool, change the Type to iscsi, and press **Forward** to continue.

3. **Add a new pool (part 2)**
   Enter the target path for the device, the host name of the target and the source path (the IQN). The **Format** option is not available as formatting is handled by the guests. It is not advised to edit the **Target Path**. The default target path value, **`/dev/disk/by-path/`**, adds the drive path to that directory. The target path should be the same on all hosts for migration.

   Enter the hostname or IP address of the iSCSI target. This example uses **`server1.example.com`**.

   Enter the source path, the IQN for the iSCSI target. This example uses **`iqn.2010-05.com.example.server1:iscsirhel6guest`**.



Press **Finish** to create the new storage pool.

11.1.5.3. Creating an iSCSI-based storage pool with virsh

1. **Create the storage pool definition**
   The example below is an XML definition file for an iSCSI-based storage pool.

   <name>iscsirhel6guest</name>
   The **name** element sets the name for the storage pool. The name is required and must be unique.

   <uuid>afcc5367-6770-e151-bcb3-847bc36c5e28</uuid>
   The optional **uuid** element provides a unique global identifier for the storage pool. The **uuid** element can contain any valid UUID or an existing UUID for the storage device. If a UUID is not provided, **virsh** will generate a UUID for the storage pool.

   <host name='server1.example.com'/>
   The **host** element with the *name* attribute specifies the hostname of the iSCSI server. The **host** element attribute can contain a *port* attribute for a non-standard iSCSI protocol port number.

   <device path='*iqn.2010-05.com.example.server1:iscsirhel6guest*'/>
   The **device** element *path* attribute must contain the IQN for the iSCSI server.

With a text editor, create an XML file for the iSCSI storage pool. This example uses a XML definition named **iscsirhel6guest.xml**.

```
<pool type='iscsi'>
    <name>iscsirhel6guest</name>
    <uuid>afcc5367-6770-e151-bcb3-847bc36c5e28</uuid>
    <source>
        <host name='server1.example.com.'/>
        <device path='iqn.2001-05.com.example.server1:iscsirhel6guest'/>
    </source>
    <target>
        <path>/dev/disk/by-path</path>
    </target>
</pool>
```

Use the **pool-define** command to define the storage pool but not start it.

```
# virsh pool-define iscsirhel6guest.xml
Pool iscsirhel6guest defined
```

2. **Alternative step: Use pool-define-as to define the pool from the command line**
   Storage pool definitions can be created with the **virsh** command line tool. Creating storage pools with **virsh** is useful for systems administrators using scripts to create multiple storage pools.

   The **virsh pool-define-as** command has several parameters which are accepted in the following format:

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

   The type, *iscsi*, defines this pool as an iSCSI based storage pool. The *name* parameter must be unique and sets the name for the storage pool. The *source-host* and *source-path* parameters are the hostname and iSCSI IQN respectively. The *source-dev* and *source-name* parameters are not required for iSCSI-based pools, use a **-** character to leave the field blank. The *target* parameter defines the location for mounting the iSCSI device on the host.

   The example below creates the same iSCSI-based storage pool as the previous step.

   ```
   # virsh pool-define-as iscsirhel6guest iscsi server1.example.com
    iqn.2010-05.com.example.server1:iscsirhel6guest - - /dev/disk/by-path
   Pool iscsirhel6guest defined
   ```

3. **Verify the storage pool is listed**
   Verify the storage pool object is created correctly and the state reports as **inactive**.

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

4. **Start the storage pool**
   Use the virsh command **pool-start** for this. **pool-start** enables a directory storage pool, allowing it to be used for volumes and guests.

   ```
   # virsh pool-start guest_images_disk
   ```

```
Pool guest_images_disk started
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
iscsirhel6guest      active     no
```

5. **Turn on autostart**
   Turn on *autostart* for the storage pool. Autostart configures the `libvirtd` service to start the storage pool when the service starts.

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

   Verify that the *iscsirhel6guest* pool has autostart set:

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

6. **Verify the storage pool configuration**
   Verify the storage pool was created correctly, the sizes reported correctly, and the state reports as **`running`**.

```
# 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
```

An iSCSI-based storage pool is now available.

11.1.6. NFS-based storage pools

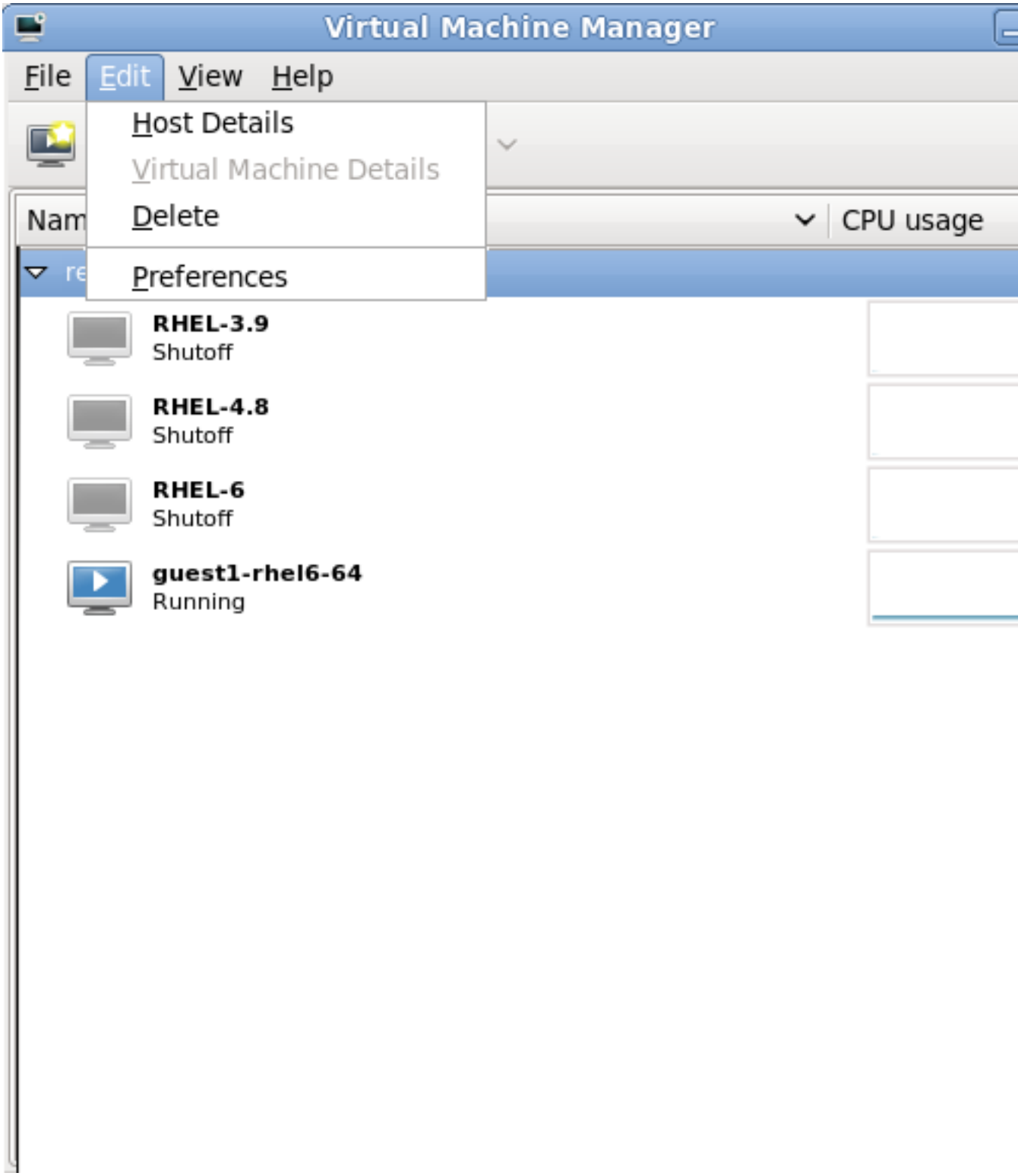
This procedure covers creating a storage pool with a NFS mount point in **`virt-manager`**.

11.1.6.1. Creating a NFS-based storage pool with virt-manager

1. **Open the host storage tab**
   Open the **Storage** tab in the **Host Details** window.

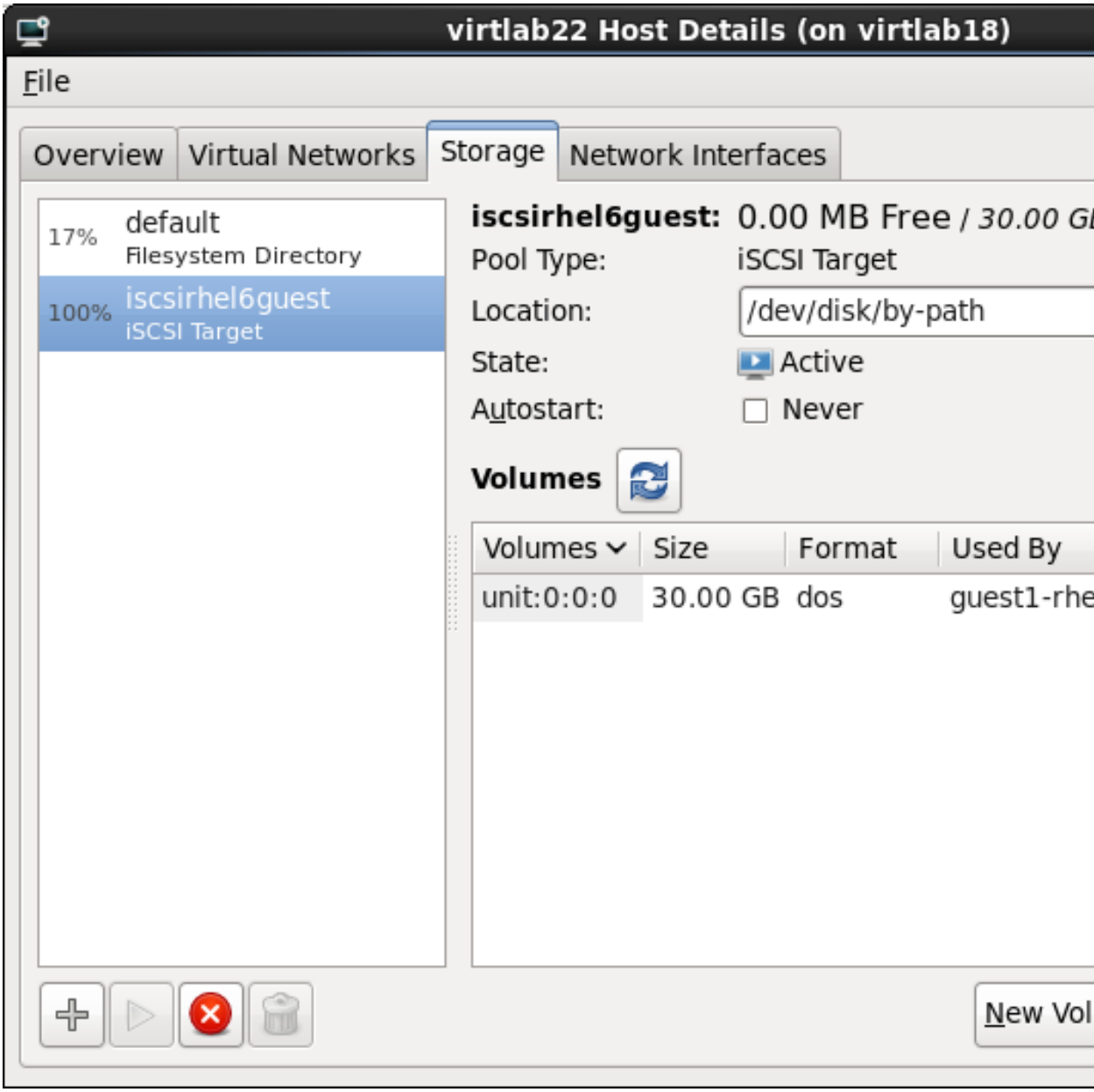
   a. Open **`virt-manager`**.

   b. Select a host from the main **`virt-manager`** window.

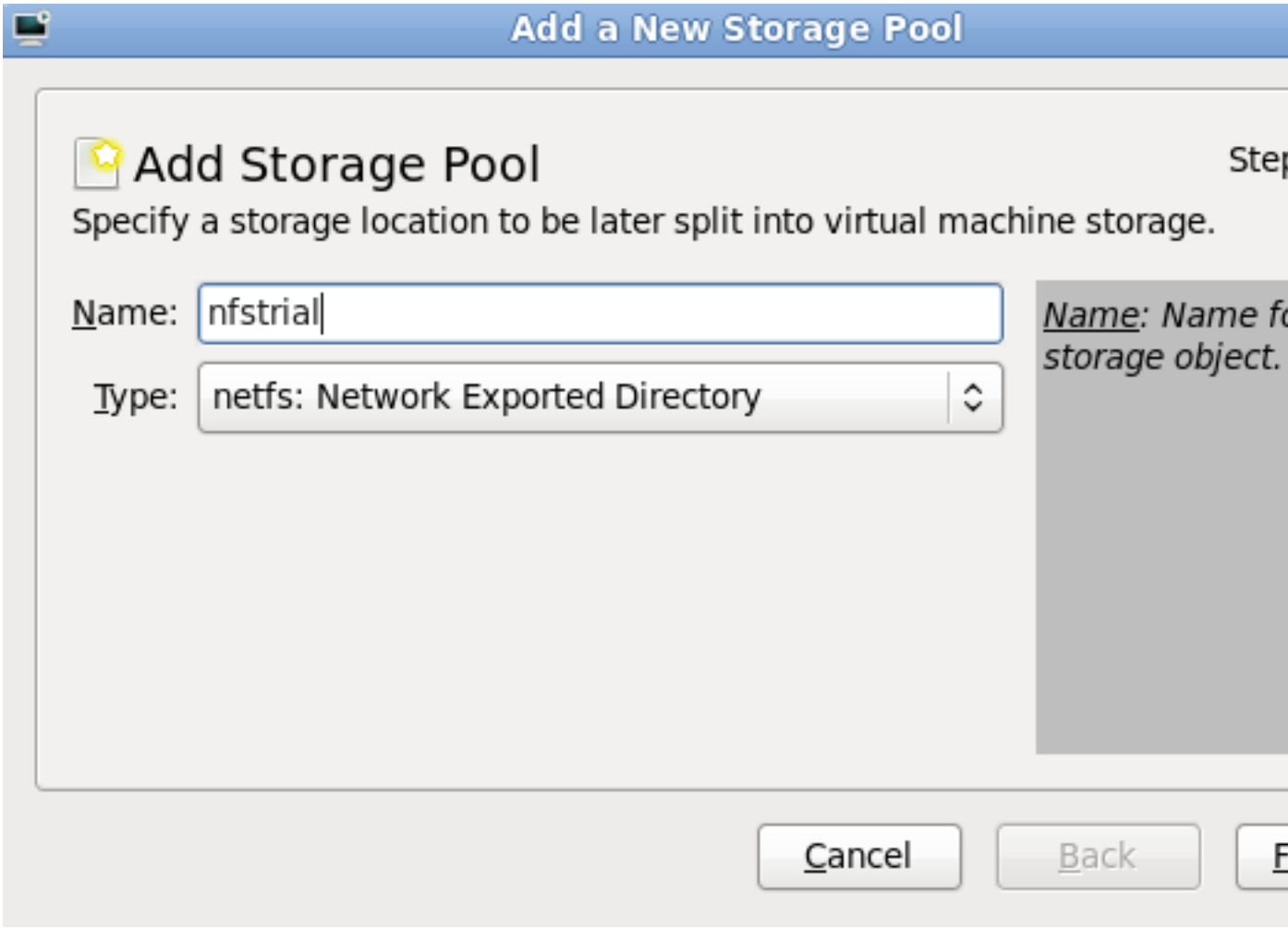


c. Open the Edit menu and select Host Details.

d. Click on the Storage tab of the Host Details window.



2. **Create a new pool (part 1)**
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

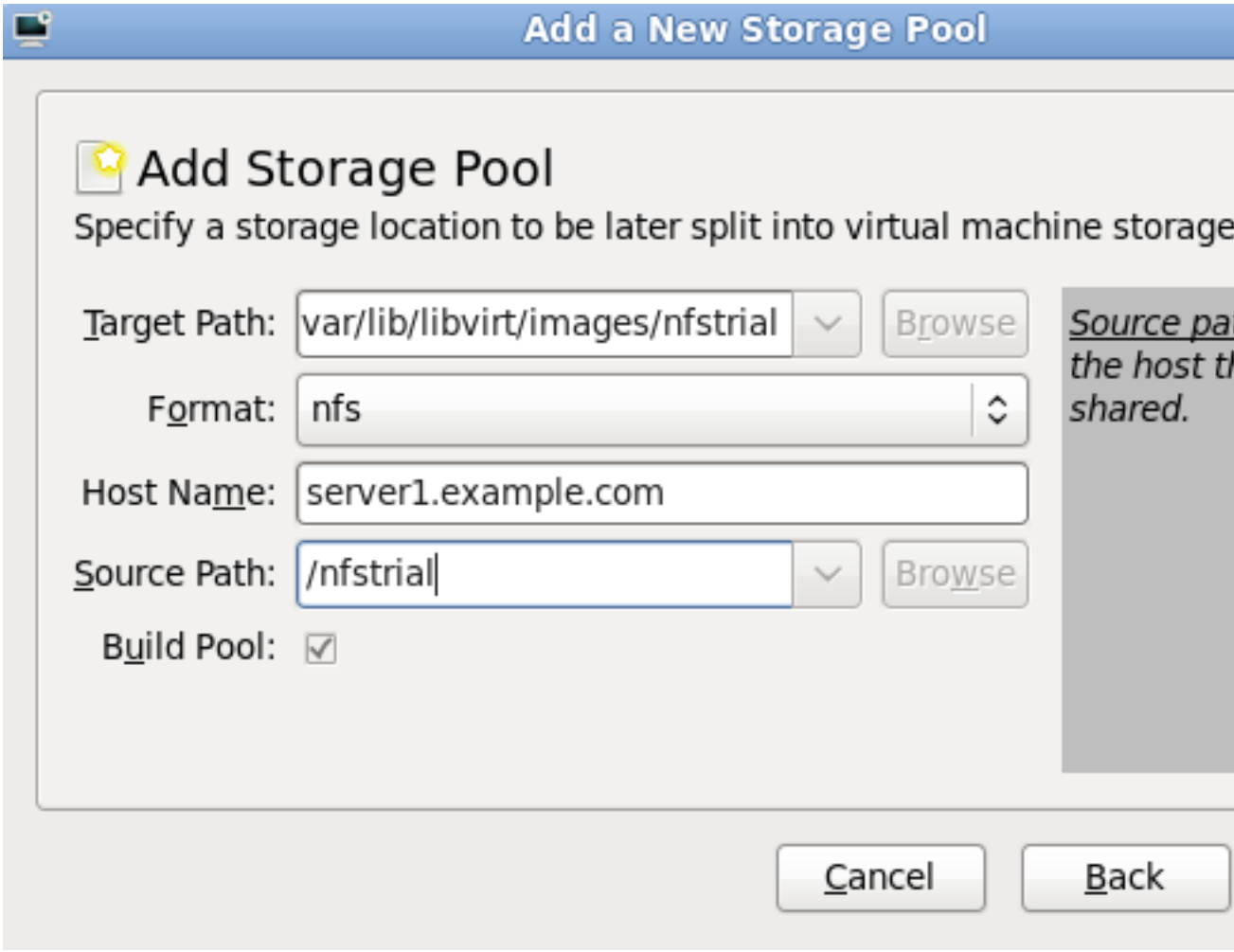


Choose a name for the storage pool and press **Forward** to continue.

3. **Create a new pool (part 2)**
   Enter the target path for the device, the hostname and the NFS share path. Set the **Format** option to **NFS** or **auto** (to detect the type). The target path must be identical on all hosts for migration.

   Enter the hostname or IP address of the NFS server. This example uses **`server1.example.com`**.

   Enter the NFS path. This example uses **`/nfstrial`**.



Press **Finish** to create the new storage pool.

Volumes

12.1. Creating volumes

This section shows how to create disk volumes inside a block based storage pool.

```
# 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

#
```

12.2. Cloning volumes

The new volume will be allocated from storage in the same storage pool as the volume being cloned.

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

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
clone1               /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      8590MB  17.2GB  8590MB               primary
3      17.2GB  25.8GB  8590MB               primary
1      25.8GB  34.4GB  8590MB               primary

#
```

12.3. Adding storage devices to guests

This section covers adding storage devices to a guest. Additional storage can only be added after guests are created.

12.3.1. Adding file based storage to a guest

File-based storage or file-based containers are files on the hosts file system that act as virtualized hard drives for guests. To add a file-based container, perform the following steps:

1. Create an empty container file or using an existing file container (such as an ISO file).

   a. Pre-allocated files are recommended for file-based storage images. Create a pre-allocated file using the **dd**:

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

   b. Alternatively, create a sparse file instead of a pre-allocated file. Sparse files are created much faster and can be used for testing, but are not recommended for production environments due to data integrity and performance issues.

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

   Both commands create a 4GB file which can be used as additional storage for a guest.

2. Create the additional storage by writing a **<disk>** element in a new file. In this example, this file will be known as **NewStorage.xml**. Ensure that you specify a device name for the virtual block device attributes. These attributes must be unique for each guest configuration file. The following example is a configuration file section which contains an additional file-based storage container named **FileName.img**.

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

   Note that the *<address>* sub-element is omitted in order to let libvirt locate and assign the next available PCI slot.

3. Start the guest to which your new device should be attached. In this case, our guest is called **Guest1**.

   ```
   # virsh start Guest1
   ```

4. Associate the device defined in **NewStorage.xml** with your guest (**Guest1**):

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

5. Reboot the guest machine:

   ```
   # virsh reboot Guest1
   ```

6.

Note

The following steps are Linux guest specific. Other operating systems handle new storage devices in different ways. For other systems, refer to that operating system's documentation.

The guest now uses the file **`FileName.img`** as the device called **`/dev/vdb`**. This device requires formatting from the guest. On the guest, partition the device into one primary partition for the entire device, then format the device.

a. Start **`fdisk`** for the new device:

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

b. Press *n* for a new partition.

c. The following appears:

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

Press *p* for a primary partition.

d. Choose an available partition number. In this example, the first partition is chosen by entering *1*.

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

e. Enter the default first cylinder by pressing *Enter*.

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

f. Select the size of the partition. In this example the entire disk is allocated by pressing *Enter*.

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

g. Enter *t* to configure the partition type.

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

h. Select the partition you created in the previous steps. In this example, the partition number is *1*.

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

i. Enter *83* for a Linux partition.

```
Hex code (type L to list codes): 83
```

j. Enter *w* to write changes to disk, and *q* to quit.

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

k. Format the new partition with the `ext3` file system.

```
# mke2fs -j /dev/vdb1
```

7. Create a mount directory, and mount the disk on the guest.

```
# mkdir /myfiles
# mount /dev/vdb1 /myfiles
```

The guest now has an additional virtualized file-based storage device. Note, however, that this storage will not mount persistently across reboot unless defined in the guest's **`/etc/fstab`** file:

```
/dev/vdb1    /myfiles    ext3     defaults    0 0
```

12.3.2. Adding hard drives and other block devices to a guest

System administrators use additional hard drives to provide increased storage space for a guest, or to separate system data from user data.

Procedure 12.1. Adding physical block devices to guests

This procedure describes how to add a hard drive on the host to a guest. It applies to all physical block devices, including CD-ROM, DVD and floppy devices.

1. Physically attach the hard disk device to the host. Configure the host if the drive is not accessible by default.
2. Configure the device with **`multipath`** and persistence on the host if required.
3. Use the **`virsh attach`** command as below, replacing:

```
# virsh attach-disk myguest /dev/sdc1 vdc --driver qemu --mode readonly
```

- *`myguest`* with the name of the guest.
- */dev/sdc1* with the device on the host to add.
- *vdc* with the location on the guest where the device should be added. It must be an unused device name.

  Use the *sd** notation for Windows guests as well, the guest will recognize the device correctly.
- Only include the *--mode readonly* parameter if the device should be read only to the guest.

Additionally, there are optional arguments that may be added:

- Append the *--type hdd* parameter to the command for CD-ROM or DVD devices.

- Append the *--type floppy* parameter to the command for floppy devices.

4. The guest now has a new hard disk device called **/dev/sdb** on Linux or **D: drive**, or similar, on Windows. This device may require formatting.

Block device security - disk labels

The host should not use disk labels to identify file systems in the **fstab** file, the **initrd** file or on the kernel command line. Doing so presents a security risk if less privileged users, such as guests, have write access to whole partitions or LVM volumes, because a guest could potentially write a disk label belonging to the host, to its own block device storage. Upon reboot of the host, the host could then mistakenly use the guest's disk as a system disk, which would compromise the host system.

It is preferable to use the UUID of a device to identify it in the **fstab** file, the **initrd** file or on the kernel command line. While using UUIDs is still not completely secure on certain file systems, a similar compromise with UUID is significantly less feasible.

Block device security - whole disk access

Guests should not be given write access to whole disks or block devices (for example, **/dev/sdb**). Guests with access to whole block devices may be able to access other disk partitions that correspond to those block device names on the system, or modify volume labels, which can be used to compromise the host system. Use partitions (for example, **/dev/sdb1**) or LVM volumes to prevent this issue.

12.4. Deleting and removing volumes

This section shows how to delete a disk volume from a block based storage pool.

```
# virsh vol-delete --pool guest_images_disk volume1
Vol volume1 deleted

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
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      8590MB  17.2GB  8590MB               primary
3      17.2GB  25.8GB  8590MB               primary

#
```

N_Port ID Virtualization (NPIV)

N_Port ID Virtualization (NPIV) is a function available with some Fibre Channel devices. NPIV shares a single physical N_Port as multiple N_Port IDs. NPIV provides similar functionality for Host Bus Adaptors (HBAs) that SR-IOV provides for network interfaces. With NPIV, guests can be provided with a virtual Fibre Channel initiator to Storage Area Networks (SANs).

N_Ports are addressed with a 24 bit N_Port ID, which is assigned by the Fibre Channel switch.

Why use NPIV

- Without NPIV guests must share an HBA's WWN on the SAN. With NPIV, it is possible to use LUN masking and zoning for guest.
- With NPIV migration with zones and LUN masking is possible.
- Physical HBAs are expensive and use an expansion slot. With NPIV, more guests can access SAN resources and guest density can be increased.

Each N_Port has a unique identity (port WWN and node WWN) on the SAN and can be used for zoning and LUN masking. Soft zoning, which you can use to group ports together by port WWN, is the preferred method of zoning.

13.1. Enabling NPIV on the switch

Enabling the NPIV on a Fibre Channel port on a switch

```
admin> portcfgshow 0
......
NPIV capability                    ON
......
Usage
portCfgNPIVPort <PortNumber> <Mode>
Mode      Meaning
0             Disable the NPIV capability on the port
1             Enable the NPIV capability on the port
```

Example:

```
admin> portCfgNPIVPort 0 1
```

13.1.1. Identifying HBAs in a Host System

To determine the types of HBAs in the system, enter the following command:

```
# ls /proc/scsi
QLogic HBAs are listed as qla2xxx. Emulex HBAs are listed as lpfc.
```

QLogic Example

```
# ls /proc/scsi/qla2xxx
```

Emulex Example

```

# ls /proc/scsi/lpfc
```

13.1.2. Verify NPIV is used on the HBA

Output the data from the kernel on the port nodes of the HBA.

Example 13.1. QLogic controller example

```
# cat /proc/scsi/qla2xxx/7
FC Port Information for Virtual Ports:
Virtual Port index = 1
Virtual Port 1:VP State = <ACTIVE>, Vp Flags = 0x0
scsiqla2port3=500601609020fd54:500601601020fd54:a00000:1000: 1;
scsiqla2port4=500601609020fd54:500601681020fd54:a10000:1000: 1;
Virtual Port 1 SCSI LUN Information:
( 0:10): Total reqs 10, Pending reqs 0, flags 0x0, 2:0:1000,
```

Example 13.2. Emulex controller example

```
# cat /proc/scsi/lpfc/3
SLI Rev: 3
NPIV Supported: VPIs max 127 VPIs used 1
RPIs max 512 RPIs used 13
Vports list on this physical port:
Vport DID 0x2f0901, vpi 1, state 0x20
Portname: 48:19:00:0c:29:00:00:0d Nodename: 48:19:00:0c:29:00:00:0b
```

13.1.2.1. Create and destroy a virtual HBA with NPIV

Issue an NPIV create call. Confirm that the host has started a new virtual HBA and that any storage zones are usable.

To create virtual HBAs using **libvirt**, you require a NPIV capable HBA and switch.

Confirm that you have those by manually creating a new HBA by printing the contents of the **/sys/*class*/*fc_host*/hostN** directory where *class* is the type of adaptor and *fc_host* is the host number.

Note that the WWN used below are for demonstrative purposes only. Use WWN customized for your SAN environment.

Add a new virtual HBA with the following command where '*1111222233334444:5555666677778888*' is **WWPN:WWNN** and *host5* is the physical HBA which the virtual HBA is a client of.

```
# echo '1111222233334444:5555666677778888' > /sys/class/fc_host/host5/vport_create
```

If the creation is successful, a new HBA in the system with the next available host number.

Note

The virtual HBAs can be destroyed with the following command:

```
# echo '1111222233334444:5555666677778888' > /sys/class/fc_host/host5/vport_delete
```

Adding the virtual HBA with virsh

This procedure covers creating virtual HBA devices on a host with **virsh**. This procedure requires a compatible HBA device.

1. **List available HBAs**
   Find the node device name of the HBA with the virtual adapters. List of all the HBAs on the host with the following command:

```
# virsh nodedev-list --cap=scsi_host
pci_10df_fe00_0_scsi_host
pci_10df_fe00_0_scsi_host_0
pci_10df_fe00_scsi_host
pci_10df_fe00_scsi_host_0
pci_10df_fe00_scsi_host_0_scsi_host
pci_10df_fe00_scsi_host_0_scsi_host_0
```

2. **Gather parent HBA device data**
   Output the XML definition for each required HBA. This example uses the HBA, **pci_10df_fe00_scsi_host**.

```
# virsh nodedev-dumpxml pci_10df_fe00_scsi_host
<device>
  <name>pci_10df_fe00_scsi_host</name>
  <parent>pci_10df_fe00</parent>
  <capability type='scsi_host'>
    <host>5</host>
    <capability type='fc_host'>
      <wwnn>20000000c9848140</wwnn>
      <wwpn>10000000c9848140</wwpn>
    </capability>
    <capability type='vport_ops' />
  </capability>
</device>
```

   HBAs capable of creating virtual HBAs have a capability **type='vport_ops'** in the XML definition.

3. **Create the XML definition for the virtual HBA**
   With information gathered in the previous step, create an XML definition for the virtual HBA. This example uses a file named **newHBA.xml**.

```
<device>
  <parent>pci_10df_fe00_0_scsi_host</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
      <wwpn>1111222233334444</wwpn>
      <wwnn>5555666677778888</wwnn>
    </capability>
  </capability>
</device>
```

   The **<parent>** element is the name of the parent HBA listed by the **virsh nodedev-list** command. The **<wwpn>** and **<wwnn>** elements are the WWNN and WWPN for the virtual HBA.

4. **Create the virtual HBA**
   Create the virtual HBA with the **virsh nodedev-create** command using the file from the previous step.

```
# virsh nodedev-create newHBA.xml
Node device pci_10df_fe00_0_scsi_host_0_scsi_host created from newHBA.xml
```

The new virtual HBA should be detected and available to the host. The create command output gives you the node device name of the newly created device.

Destroying a virtual HBA with virsh

To destroy the device, use **`virsh nodedev-destroy`**:

```
# virsh nodedev-destroy pci_10df_fe00_0_scsi_host_0_scsi_host
Destroyed node device 'pci_10df_fe00_0_scsi_host_0_scsi_host'
```

The Virtual Host Metrics Daemon (vhostmd)

vhostmd (the Virtual Host Metrics Daemon) allows virtual machines to see limited information about the host they are running on.

In the host, a daemon (**vhostmd**) runs which writes metrics periodically into a disk image. This disk image is exported read-only to guests. Guests can read the disk image to see metrics. Simple synchronization stops guests from seeing out of date or corrupt metrics.

The system administrator chooses which metrics the guests can see, and also which guests get to see the metrics at all.

14.1. Installing vhostmd on the host

The **vhostmd** package is available from RHN. It must be installed on each host where guests are required to get host metrics.

14.2. Configuration of vhostmd

After installing the package, but before starting the daemon, it is a good idea to understand exactly what metrics **vhostmd** will expose to guests, and how this happens.

The metrics are controlled by the file **/etc/vhostmd/vhostmd.conf**.

There are two parts of particular importance in this XML file. Firstly *<update_period>60</update_period>* controls how often the metrics are updated (in seconds). Since updating metrics can be an expensive operation, you can reduce the load on the host by increasing this period. Secondly, each *<metric>...</metric>* section controls what information is exposed by **vhostmd**. For example:

```
<metric type="string" context="host">
   <name>HostName</name>
   <action>hostname</action>
</metric>
```

means that the hostname of the host is exposed to selected guests. To disable particular metrics, you can comment out *<metric>* sections by putting **<!-- ... -->** around them. Note that disabling metrics may cause problems for guest software such as SAP that may rely on these metrics being available.

When the daemon (also called **vhostmd**) is running, it writes the metrics into a temporary file called **/dev/shm/vhostmd0**. This file contains a small binary header followed by the selected metrics encoded as XML. In practice you can display this file with a tool like **less**. The file is updated every 60 seconds (or however often *<update_period>* was set).

The vhostmd(8) man page contains a detailed description of the configuration file, as well as examples of the XML output in **/dev/shm/vhostmd0**. To read this, do:

```
# man vhostmd
```

In addition, there is a README file which covers some of the same information:

```
less /usr/share/doc/vhostmd-*/README
```

14.3. Starting and stopping the daemon

The daemon (**vhostmd**) will not be started automatically. To enable it to be started at boot, run:

```
# /sbin/chkconfig vhostmd on
```

To start the daemon running, do:

```
# /sbin/service vhostmd start
```

To stop the daemon running, do:

```
# /sbin/service vhostmd stop
```

To disable the daemon from being started at boot, do:

```
# /sbin/chkconfig vhostmd off
```

14.4. Verifying that vhostmd is working from the host

A short time after the daemon has started, you should see a metrics disk appearing. Do:

```
# ls -l /dev/shm
# less /dev/shm/vhostmd0
```

This file has a short binary header, followed by XML. The **less** program identifies it as binary and asks:

```
"/dev/shm/vhostmd0" may be a binary file.  See it anyway?
```

Press the **y** key to indicate that you wish to view it.

You should see the binary header appearing as garbled characters, followed by the <*metrics*> XML, and after that, many zero bytes (displayed as ^@^@^@...).

14.5. Configuring guests to see the metrics

Although metrics are written to **/dev/shm/vhostmd0**, they are not made available to guests by default. The administrator must choose which guests get to see metrics, and must manually change the configuration of selected guests to see metrics.

The guest must be shut down before the disk is attached. (Hot attaching the metrics disk is also possible, but only for a limited number of guest configurations. In particular it is NOT possible to hot-add the metrics disk to guests that don't have virtio / PV drivers installed. See the vhostmd README file for more information).

There are two ways to do this, depending on whether the host hypervisor is KVM or Xen (Red Hat Enterprise Linux 5).

Important

It is extremely important that the metrics disk is added in readonly mode to all guests. If this is not done, then it would be possible for a guest to modify the metrics and possibly subvert other guests that are reading it.

Procedure 14.1. Configuring KVM guests

1. Shut down the guest.
2. Do:

```
# virsh edit GuestName
```

and add the following section into *<devices>*:

```
<disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/dev/shm/vhostmd0'/>
      <target dev='vdd' bus='virtio'/>
      <readonly/>
   </disk>
```

3. Reboot the guest.

Procedure 14.2. Configuring Xen guests

1. Shut down the guest.
2. Do:

```
# virsh edit GuestName
```

and add the following section into *<devices>*:

```
<disk type='file' device='disk'>
      <source dev='/dev/shm/vhostmd0'/>
      <target dev='hdd' bus='ide'/>
      <readonly/>
   </disk>
```

3. Reboot the guest.

14.6. Using vm-dump-metrics in Red Hat Enterprise Linux guests to verify operation

Optionally, the *vm-dump-metrics* package from RHN may be installed in Red Hat Enterprise Linux guests. This package provides a simple command line tool (also called **vm-dump-metrics**) which allows host metrics to be displayed in the guest.

This is useful for verifying correct operation of **vhostmd** from a guest.

In the guest, you simply run the command as root:

```
# vm-dump-metrics
```

If everything is working, this should print out a long XML document starting with **<metrics>**.

If this does not work, then verify that the metrics disk has appeared in the guest. It should appear as **/dev/vd*** (for example, **/dev/vdb**, **/dev/vdd**).

On the host, verify that the libvirt configuration changes have been made by using the command:

```
# virsh dumpxml GuestName
```

Verify that vhostmd is running on the host and the /dev/shm/vhostmd0 file exists.

Managing guests with virsh

virsh is a command line interface tool for managing guests and the hypervisor. The **virsh** command-line tool is built on the **libvirt** management API and operates as an alternative to the **qemu-kvm** command and the graphical **virt-manager** application. The **virsh** command can be used in read-only mode by unprivileged users or, with root access, full administration functionality. The **virsh** command is ideal for scripting virtualization administration.

15.1. virsh command quick reference

The following tables provide a quick reference for all virsh command line options.

Table 15.1. Guest management commands

Command	Description
help	Prints basic help information.
list	Lists all guests.
dumpxml	Outputs the XML configuration file for the guest.
create	Creates a guest from an XML configuration file and starts the new guest.
start	Starts an inactive guest.
destroy	Forces a guest to stop.
define	Creates a guest from an XML configuration file without starting the new guest.
domid	Displays the guest's ID.
domuuid	Displays the guest's UUID.
dominfo	Displays guest information.
domname	Displays the guest's name.
domstate	Displays the state of a guest.
quit	Quits the interactive terminal.
reboot	Reboots a guest.
restore	Restores a previously saved guest stored in a file.
resume	Resumes a paused guest.
save	Save the present state of a guest to a file.
shutdown	Gracefully shuts down a guest.
suspend	Pauses a guest.
undefine	Deletes all files associated with a guest.
migrate	Migrates a guest to another host.

The following **virsh** command options manage guest and hypervisor resources:

Table 15.2. Resource management options

Command	Description
setmem	Sets the allocated memory for a guest. Refer to the **virsh** manpage for more details.

Command	Description
setmaxmem	Sets maximum memory limit for the hypervisor. Refer to the **virsh** manpage for more details.
setvcpus	Changes number of virtual CPUs assigned to a guest. Refer to the **virsh** manpage for more details.
vcpuinfo	Displays virtual CPU information about a guest.
vcpupin	Controls the virtual CPU affinity of a guest.
domblkstat	Displays block device statistics for a running guest.
domifstat	Displays network interface statistics for a running guest.
attach-device	Attach a device to a guest, using a device definition in an XML file.
attach-disk	Attaches a new disk device to a guest.
attach-interface	Attaches a new network interface to a guest.
update-device	Detach a disk image from a guest's CD-ROM drive. See *Section 15.2, "Attaching and updating a device with virsh"* for more details.
detach-device	Detach a device from a guest, takes the same kind of XML descriptions as command **attach-device**.
detach-disk	Detach a disk device from a guest.
detach-interface	Detach a network interface from a guest.

The **virsh** commands for managing and creating storage pools and volumes.

For more information on using storage pools with virsh, refer to *http://libvirt.org/formatstorage.html*

Table 15.3. Storage Pool options

Command	Description
find-storage-pool-sources	Returns the XML definition for all storage pools of a given type that could be found.
find-storage-pool-sources port	Returns data on all storage pools of a given type that could be found as XML. If the host and port are provided, this command can be run remotely.
pool-autostart	Sets the storage pool to start at boot time.
pool-build	The **pool-build** command builds a defined pool. This command can format disks and create partitions.
pool-create	**pool-create** creates and starts a storage pool from the provided XML storage pool definition file.
pool-create-as name	Creates and starts a storage pool from the provided parameters. If the *--print-xml* parameter is specified, the command prints the XML definition for the storage pool without creating the storage pool.

Command	Description
pool-define	Creates a storage bool from an XML definition file but does not start the new storage pool.
pool-define-as name	Creates but does not start, a storage pool from the provided parameters. If the *--print-xml* parameter is specified, the command prints the XML definition for the storage pool without creating the storage pool.
pool-destroy	Permanently destroys a storage pool in **libvirt**. The raw data contained in the storage pool is not changed and can be recovered with the **pool-create** command.
pool-delete	Destroys the storage resources used by a storage pool. This operation cannot be recovered. The storage pool still exists after this command but all data is deleted.
pool-dumpxml	Prints the XML definition for a storage pool.
pool-edit	Opens the XML definition file for a storage pool in the users default text editor.
pool-info	Returns information about a storage pool.
pool-list	Lists storage pools known to libvirt. By default, **pool-list** lists pools in use by active guests. The *--inactive* parameter lists inactive pools and the *--all* parameter lists all pools.
pool-undefine	Deletes the definition for an inactive storage pool.
pool-uuid	Returns the UUID of the named pool.
pool-name	Prints a storage pool's name when provided the UUID of a storage pool.
pool-refresh	Refreshes the list of volumes contained in a storage pool.
pool-start	Starts a storage pool that is defined but inactive.

Table 15.4. Volume options

Command	Description
vol-create	Create a volume from an XML file.
vol-create-from	Create a volume using another volume as input.
vol-create-as	Create a volume from a set of arguments.
vol-clone	Clone a volume.
vol-delete	Delete a volume.
vol-wipe	Wipe a volume.
vol-dumpxml	Show volume information in XML.
vol-info	Show storage volume information.
vol-list	List volumes.

Command	Description
vol-pool	Returns the storage pool for a given volume key or path.
vol-path	Returns the volume path for a given volume name or key.
vol-name	Returns the volume name for a given volume key or path.
vol-key	Returns the volume key for a given volume name or path.

Table 15.5. Secret options

Command	Description
secret-define	Define or modify a secret from an XML file.
secret-dumpxml	Show secret attributes in XML.
secret-set-value	Set a secret value.
secret-get-value	Output a secret value.
secret-undefine	Undefine a secret.
secret-list	List secrets.

Table 15.6. Network filter options

Command	Description
nwfilter-define	Define or update a network filter from an XML file.
nwfilter-undefine	Undefine a network filter.
nwfilter-dumpxml	Show network filter information in XML.
nwfilter-list	List network filters.
nwfilter-edit	Edit XML configuration for a network filter.

This table contains **virsh** command options for snapshots:

Table 15.7. Snapshot options

Command	Description
snapshot-create	Create a snapshot.
snapshot-current	Get the current snapshot.
snapshot-delete	Delete a domain snapshot.
snapshot-dumpxml	Dump XML for a domain snapshot.
snapshot-list	List snapshots for a domain.
snapshot-revert	Revert a domain to a snapshot.

This table contains miscellaneous **virsh** commands:

Table 15.8. Miscellaneous options

Command	Description
version	Displays the version of **virsh**.

Command	Description
nodeinfo	Outputs information about the hypervisor.

15.2. Attaching and updating a device with virsh

Attaching a disk image to a guest's CD-ROM drive with **virsh**:

1. Attach a disk image to a guest's CD-ROM drive:

```
# virsh attach-disk <GuestName> sample.iso hdc --type cdrom --mode readonly
Disk attached successfully
```

2. Create an XML file to update a specific device. To detach, remove a line for the source device:

```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
  <alias name='ide0-1-0'/>
  <address type='drive' controller='0' bus='1' unit='0'/>
</disk>
```

3. Detach a disk image by updating the device:

```
virsh update-device <GuestName> guest-device.xml
Device updated successfully
```

15.3. Connecting to the hypervisor

Connect to a hypervisor session with **virsh**:

```
# virsh connect {name}
```

Where *{name}* is the machine name (hostname) or URL (the output of the **virsh uri** command) of the hypervisor. To initiate a read-only connection, append the above command with **--readonly**.

15.4. Creating a virtual machine XML dump (configuration file)

Output a guest's XML configuration file with **virsh**:

```
# virsh dumpxml {guest-id, guestname or uuid}
```

This command outputs the guest's XML configuration file to standard out (**stdout**). You can save the data by piping the output to a file. An example of piping the output to a file called *guest.xml*:

```
# virsh dumpxml GuestID > guest.xml
```

This file **guest.xml** can recreate the guest (refer to *Editing a guest's configuration file*. You can edit this XML configuration file to configure additional devices or to deploy additional guests. Refer to

Section 20.1, "Using XML configuration files with virsh" for more information on modifying files created with **`virsh dumpxml`**.

An example of **`virsh dumpxml`** output:

```
# 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>
```

Note that the <shareable/> flag is set. This indicates the device is expected to be shared between domains (assuming the hypervisor and OS support this), which means that caching should be deactivated for that device.

Creating a guest from a configuration file

Guests can be created from XML configuration files. You can copy existing XML from previously created guests or use the **dumpxml** option (refer to *Section 15.4, "Creating a virtual machine XML dump (configuration file)"*). To create a guest with **virsh** from an XML file:

```
# virsh create configuration_file.xml
```

Editing a guest's configuration file

Instead of using the **dumpxml** option (refer to *Section 15.4, "Creating a virtual machine XML dump (configuration file)"*) guests can be edited either while they run or while they are offline. The **virsh edit** command provides this functionality. For example, to edit the guest named *softwaretesting*:

```
# virsh edit softwaretesting
```

This opens a text editor. The default text editor is the **$EDITOR** shell parameter (set to **vi** by default).

15.5. Suspending, resuming, saving and restoring a guest

Suspending a guest

Suspend a guest with **virsh**:

```
# virsh suspend {domain-id, domain-name or domain-uuid}
```

When a guest is in a suspended state, it consumes system RAM but not processor resources. Disk and network I/O does not occur while the guest is suspended. This operation is immediate and the guest can be restarted with the **resume** (*Resuming a guest*) option.

Resuming a guest

Restore a suspended guest with **virsh** using the **resume** option:

```
# virsh resume {domain-id, domain-name or domain-uuid}
```

This operation is immediate and the guest parameters are preserved for **suspend** and **resume** operations.

Save a guest

Save the current state of a guest to a file using the **virsh** command:

```
# virsh save {domain-name, domain-id or domain-uuid} filename
```

This stops the guest you specify and saves the data to a file, which may take some time given the amount of memory in use by your guest. You can restore the state of the guest with the **restore**

(*Restore a guest*) option. Save is similar to pause, instead of just pausing a guest the present state of the guest is saved.

Restore a guest

Restore a guest previously saved with the **virsh save** command (*Save a guest*) using **virsh**:

```
# virsh restore filename
```

This restarts the saved guest, which may take some time. The guest's name and UUID are preserved but are allocated for a new id.

15.6. Shutting down, rebooting and force-shutdown of a guest

Shut down a guest

Shut down a guest using the **virsh** command:

```
# virsh shutdown {domain-id, domain-name or domain-uuid}
```

You can control the behavior of the rebooting guest by modifying the **on_shutdown** parameter in the guest's configuration file.

Rebooting a guest

Reboot a guest using **virsh** command:

```
#virsh reboot {domain-id, domain-name or domain-uuid}
```

You can control the behavior of the rebooting guest by modifying the **on_reboot** element in the guest's configuration file.

Forcing a guest to stop

Force a guest to stop with the **virsh** command:

```
# virsh destroy {domain-id, domain-name or domain-uuid}
```

This command does an immediate ungraceful shutdown and stops the specified guest. Using **virsh destroy** can corrupt guest file systems. Use the **destroy** option only when the guest is unresponsive.

15.7. Retrieving guest information

Getting the domain ID of a guest

To get the domain ID of a guest:

```
# virsh domid {domain-name or domain-uuid}
```

Getting the domain name of a guest

To get the domain name of a guest:

```
# virsh domname {domain-id or domain-uuid}
```

Getting the UUID of a guest

To get the Universally Unique Identifier (UUID) for a guest:

```
# virsh domuuid {domain-id or domain-name}
```

An example of **`virsh domuuid`** output:

```
# virsh domuuid r5b2-mySQL01
4a4c59a7-ee3f-c781-96e4-288f2862f011
```

Displaying guest Information

Using **`virsh`** with the guest's domain ID, domain name or UUID you can display information on the specified guest:

```
# virsh dominfo {domain-id, domain-name or domain-uuid}
```

This is an example of **`virsh dominfo`** output:

```
# virsh dominfo r5b2-mySQL01
id:             13
name:           r5b2-mysql01
uuid:           4a4c59a7-ee3f-c781-96e4-288f2862f011
os type:       linux
state:          blocked
cpu(s):         1
cpu time:      11.0s
max memory:     512000 kb
used memory:    512000 kb
```

15.8. Retrieving host information

Displaying host information

To display information about the host:

```
# virsh nodeinfo
```

An example of **`virsh nodeinfo`** output:

```
# virsh nodeinfo
CPU model                    x86_64
CPU (s)                      8
CPU frequency                2895 Mhz
CPU socket(s)                2
Core(s) per socket           2
```

```
Threads per core:            2
Numa cell(s)                 1
Memory size:                 1046528 kb
```

This displays the node information and the machines that support the virtualization process.

15.9. Storage pool information

Editing a storage pool definition

The **virsh pool-edit** command takes the name or UUID for a storage pool and opens the XML definition file for a storage pool in the users default text editor.

The **virsh pool-edit** command is equivalent to running the following commands:

```
# virsh pool-dumpxml pool > pool.xml
# vim pool.xml
# virsh pool-define pool.xml
```

Note

The default editor is defined by the **$VISUAL** or **$EDITOR** environment variables, and default is **vi**.

15.10. Displaying per-guest information

Displaying the guests

To display the guest list and their current states with **virsh**:

```
# virsh list
```

Other options available include:

the **--inactive** option to list inactive guests (that is, guests that have been defined but are not currently active), and

the **--all** option lists all guests. For example:

```
# virsh list --all
 Id Name                 State
----------------------------------
  0 Domain-0             running
  1 Domain202            paused
  2 Domain010            inactive
  3 Domain9600           crashed
```

The output from **virsh list** is categorized as one of the six states (listed below).

- The **running** state refers to guests which are currently active on a CPU.

- Guests listed as **blocked** are blocked, and are not running or runnable. This is caused by a guest waiting on I/O (a traditional wait state) or guests in a sleep mode.
- The **paused** state lists domains that are paused. This occurs if an administrator uses the **pause** button in **virt-manager**, **xm pause** or **virsh suspend**. When a guest is paused it consumes memory and other resources but it is ineligible for scheduling and CPU resources from the hypervisor.
- The **shutdown** state is for guests in the process of shutting down. The guest is sent a shutdown signal and should be in the process of stopping its operations gracefully. This may not work with all guest operating systems; some operating systems do not respond to these signals.
- Domains in the **dying** state are in is in process of dying, which is a state where the domain has not completely shut-down or crashed.
- **crashed** guests have failed while running and are no longer running. This state can only occur if the guest has been configured not to restart on crash.

Displaying virtual CPU information

To display virtual CPU information from a guest with **virsh**:

```
# virsh vcpuinfo {domain-id, domain-name or domain-uuid}
```

An example of **virsh vcpuinfo** output:

```
# virsh vcpuinfo r5b2-mySQL01
VCPU:           0
CPU:            0
State:          blocked
CPU time:       0.0s
CPU Affinity:   yy
```

Configuring virtual CPU affinity

To configure the affinity of virtual CPUs with physical CPUs:

```
# virsh vcpupin domain-id vcpu cpulist
```

The **domain-id** parameter is the guest's ID number or name.

The **vcpu** parameter denotes the number of virtualized CPUs allocated to the guest.The **vcpu** parameter must be provided.

The **cpulist** parameter is a list of physical CPU identifier numbers separated by commas. The **cpulist** parameter determines which physical CPUs the VCPUs can run on.

Configuring virtual CPU count

To modify the number of CPUs assigned to a guest with **virsh**:

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count
```

The new *count* value cannot exceed the count above the amount specified when the guest was created.

Configuring memory allocation

To modify a guest's memory allocation with **virsh**:

```
# virsh setmem {domain-id or domain-name} count
```

You must specify the *count* in kilobytes. The new count value cannot exceed the amount you specified when you created the guest. Values lower than 64 MB are unlikely to work with most guest operating systems. A higher maximum memory value does not affect an active guests. If the new value is lower the available memory will shrink and the guest may crash.

Displaying guest block device information

Use **virsh domblkstat** to display block device statistics for a running guest.

```
# virsh domblkstat GuestName block-device
```

Displaying guest network device information

Use **virsh domifstat** to display network interface statistics for a running guest.

```
# virsh domifstat GuestName interface-device
```

15.11. Managing virtual networks

This section covers managing virtual networks with the **virsh** command. To list virtual networks:

```
# virsh net-list
```

This command generates output similar to:

```
# virsh net-list
Name                 State      Autostart
-----------------------------------------
default              active     yes
vnet1              active     yes
vnet2              active     yes
```

To view network information for a specific virtual network:

```
# virsh net-dumpxml NetworkName
```

This displays information about a specified virtual network in XML format:

```
# 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>
```

Other **virsh** commands used in managing virtual networks are:

- **virsh net-autostart *network-name*** — Autostart a network specified as *network-name*.
- **virsh net-create *XMLfile*** — generates and starts a new network using an existing XML file.
- **virsh net-define *XMLfile*** — generates a new network device from an existing XML file without starting it.
- **virsh net-destroy *network-name*** — destroy a network specified as *network-name*.
- **virsh net-name *networkUUID*** — convert a specified *networkUUID* to a network name.
- **virsh net-uuid *network-name*** — convert a specified *network-name* to a network UUID.
- **virsh net-start *nameOfInactiveNetwork*** — starts an inactive network.
- **virsh net-undefine *nameOfInactiveNetwork*** — removes the definition of an inactive network.

15.12. Migrating guests with virsh

A guest can be migrated to another host with **virsh**. Migrate domain to another host. Add --live for live migration. The **migrate** command accepts parameters in the following format:

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

The *--live* parameter is optional. Add the *--live* parameter for live migrations.

The *GuestName* parameter represents the name of the guest which you want to migrate.

The *DestinationURL* parameter is the URL or hostname of the destination system. The destination system requires:

- Red Hat Enterprise Linux 5.4 (ASYNC update 4) or newer,
- the same hypervisor version, and
- the **libvirt** service must be started.

Once the command is entered you will be prompted for the root password of the destination system.

Note

Non-running guests cannot be migrated with the **virsh migrate** command. To migrate a non-running guest, the following script should be used:

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

15.13. Guest CPU model configuration

15.13.1. Introduction

Every hypervisor has its own policies for what a guest will see for its CPUs by default. Some hypervisors simply pass through the host CPU directly. QEMU/KVM presents the guest with a generic model called **qemu32** or **qemu64**. Some hypervisors perform more advanced filtering, classifying all physical CPUs into a handful of groups and have one baseline CPU model for each group that is presented to the guest. Such behaviour enables the safe migration of guests between hosts, provided they all have physical CPUs that classify into the same group. libvirt does not typically enforce policy itself, rather it provides the mechanism on which the higher layers define their own desired policy. Understanding how to obtain CPU model information and define a suitable guest CPU model is critical to ensure guest migration is successful between hosts.

15.13.2. Learning about the host CPU model

The **`virsh capabilities`** command displays an XML document describing the capabilities of the hypervisor connection and host. The XML schema displayed has been extended to provide information about the host CPU model. One of the big challenges in describing a CPU model is that every architecture has a different approach to exposing their capabilities. On x86, the capabilities of a modern CPU are exposed via the CPUID instruction. Essentially this comes down to a set of 32-bit integers with each bit given a specific meaning. Fortunately AMD and Intel agree on common semantics for these bits. Other hypervisors expose the notion of CPUID masks directly in their guest configuration format. However, QEMU/KVM supports far more than just the x86 architecture, so CPUID is clearly not suitable as the canonical configuration format. QEMU ended up using a scheme which combines a CPU model name string, with a set of named flags. On x86, the CPU model maps to a baseline CPUID mask, and the flags can be used to then toggle bits in the mask on or off. libvirt decided to follow this lead and uses a combination of a model name and flags. Here is an example of what libvirt reports as the capabilities on a development workstation:

```
# virsh capabilities
<capabilities>

  <host>
    <uuid>c4a68e53-3f41-6d9e-baaf-d33a181ccfa0</uuid>
    <cpu>
      <arch>x86_64</arch>
      <model>core2duo</model>
      <topology sockets='1' cores='4' threads='1'/>
      <feature name='lahf_lm'/>
      <feature name='sse4.1'/>
      <feature name='xtpr'/>
      <feature name='cx16'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='acpi'/>
      <feature name='ds'/>
    </cpu>

   ... snip ...
  </host>
```

```
</capabilities>
```

It is not practical to have a database listing all known CPU models, so libvirt has a small list of baseline CPU model names. It chooses the one that shares the greatest number of CPUID bits with the actual host CPU and then lists the remaining bits as named features. Notice that libvirt does not display which features the baseline CPU contains. This might seem like a flaw at first, but as will be explained in this section, it is not actually necessary to know this information.

15.13.3. Determining a compatible CPU model to suit a pool of hosts

Now that it is possible to find out what CPU capabilities a single host has, the next step is to determine what CPU capabilities are best to expose to the guest. If it is known that the guest will never need to be migrated to another host, the host CPU model can be passed straight through unmodified. A virtualized data center may have a set of configurations that can guarantee all servers will have 100% identical CPUs. Again the host CPU model can be passed straight through unmodified. The more common case, though, is where there is variation in CPUs between hosts. In this mixed CPU environment, the lowest common denominator CPU must be determined. This is not entirely straightforward, so libvirt provides an API for exactly this task. If libvirt is provided a list of XML documents, each describing a CPU model for a host, libvirt will internally convert these to CPUID masks, calculate their intersection, and convert the CPUID mask result back into an XML CPU description. Taking the CPU description from a server:

```
# virsh capabilities
<capabilities>

  <host>
    <uuid>8e8e4e67-9df4-9117-bf29-ffc31f6b6abb</uuid>
    <cpu>
      <arch>x86_64</arch>
      <model>Westmere</model>
      <vendor>Intel</vendor>
      <topology sockets='2' cores='4' threads='2'/>
      <feature name='rdtscp'/>
      <feature name='pdpe1gb'/>
      <feature name='dca'/>
      <feature name='xtpr'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='monitor'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='acpi'/>
      <feature name='ds'/>
      <feature name='vme'/>
    </cpu>

   ... snip ...

</capabilities>
```

A quick check can be made to see whether this CPU description is compatible with the previous workstation CPU description, using the **`virsh cpu-compare`** command. To do so, the **`virsh capabilities > virsh-caps-workstation-full.xml`** command was executed on the workstation. The file **`virsh-caps-workstation-full.xml`** was edited and reduced to just the following content:

```
<cpu>
      <arch>x86_64</arch>
      <model>core2duo</model>
      <topology sockets='1' cores='4' threads='1'/>
      <feature name='lahf_lm'/>
      <feature name='sse4.1'/>
      <feature name='xtpr'/>
      <feature name='cx16'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='acpi'/>
      <feature name='ds'/>
    </cpu>
```

The reduced content was stored in a file named **virsh-caps-workstation-cpu-only.xml** and the **virsh cpu-compare** command can be executed using this file:

```
virsh cpu-compare virsh-caps-workstation-cpu-only.xml
Host CPU is a superset of CPU described in virsh-caps-workstation-cpu-only.xml
```

As seen in this output, libvirt is correctly reporting the CPUs are not strictly compatible, because there are several features in the server CPU that are missing in the workstation CPU. To be able to migrate between the workstation and the server, it will be necessary to mask out some features, but to determine which ones, libvirt provides an API for this, shown via the **virsh cpu-baseline** command:

```
# virsh cpu-baseline virsh-cap-weybridge-strictly-cpu-only.xml
<cpu match='exact'>
  <model>Penryn</model>
  <feature policy='require' name='xtpr'/>
  <feature policy='require' name='tm2'/>
  <feature policy='require' name='est'/>
  <feature policy='require' name='vmx'/>
  <feature policy='require' name='ds_cpl'/>
  <feature policy='require' name='monitor'/>
  <feature policy='require' name='pbe'/>
  <feature policy='require' name='tm'/>
  <feature policy='require' name='ht'/>
  <feature policy='require' name='ss'/>
  <feature policy='require' name='acpi'/>
  <feature policy='require' name='ds'/>
  <feature policy='require' name='vme'/>
</cpu>
```

Similarly, if the two <cpu>...</cpu> elements are put into a single file named **both-cpus.xml**, the following command would generate the same result:

```
 # virsh cpu-baseline both-cpus.xml
```

In this case, libvirt has determined that in order to safely migrate a guest between the workstation and the server, it is necessary to mask out 3 features from the XML description for the server, and 3 features from the XML description for the workstation.

15.13.4. Configuring the guest CPU model

For simple defaults, the guest CPU configuration accepts the same basic XML representation as the host capabilities XML exposes. In other words, the XML from the **`cpu-baseline`** virsh command can now be copied directly into the guest XML at the top level under the <domain> element. As the observant reader will have noticed from the previous XML snippet, there are a few extra attributes available when describing a CPU in the guest XML. These can mostly be ignored, but for the curious here is a quick description of what they do. The top level <cpu> element has an attribute called *match* with possible values of:

- match='minimum' - the host CPU must have at least the CPU features described in the guest XML. If the host has additional features beyond the guest configuration, these will also be exposed to the guest.
- match='exact' - the host CPU must have at least the CPU features described in the guest XML. If the host has additional features beyond the guest configuration, these will be masked out from the guest.
- match='strict' - the host CPU must have exactly the same CPU features described in the guest XML.

The next enhancement is that the <feature> elements can each have an extra 'policy' attribute with possible values of:

- policy='force' - expose the feature to the guest even if the host does not have it. This is usually only useful in the case of software emulation.
- policy='require' - expose the feature to the guest and fail if the host does not have it. This is the sensible default.
- policy='optional' - expose the feature to the guest if it happens to support it.
- policy='disable' - if the host has this feature, then hide it from the guest.
- policy='forbid' - if the host has this feature, then fail and refuse to start the guest.

The 'forbid' policy is for a niche scenario where an incorrectly functioning application will try to use a feature even if it is not in the CPUID mask, and you wish to prevent accidentally running the guest on a host with that feature. The 'optional' policy has special behaviour with respect to migration. When the guest is initially started the flag is optional, but when the guest is live migrated, this policy turns into 'require', since you cannot have features disappearing across migration.

Managing guests with the Virtual Machine Manager (virt-manager)

This section describes the Virtual Machine Manager (**virt-manager**) windows, dialog boxes, and various GUI controls.

virt-manager provides a graphical view of hypervisors and guests on your host system and on remote host systems. **virt-manager** can perform virtualization management tasks, including:

- defining and creating guests,
- assigning memory,
- assigning virtual CPUs,
- monitoring operational performance,
- saving and restoring, pausing and resuming, and shutting down and starting guests,
- links to the textual and graphical consoles, and
- live and offline migrations.

16.1. Starting virt-manager

To start **virt-manager** session open the **Applications** menu, then the **System Tools** menu and select **Virtual Machine Manager** (**virt-manager**).

The **virt-manager** main window appears.

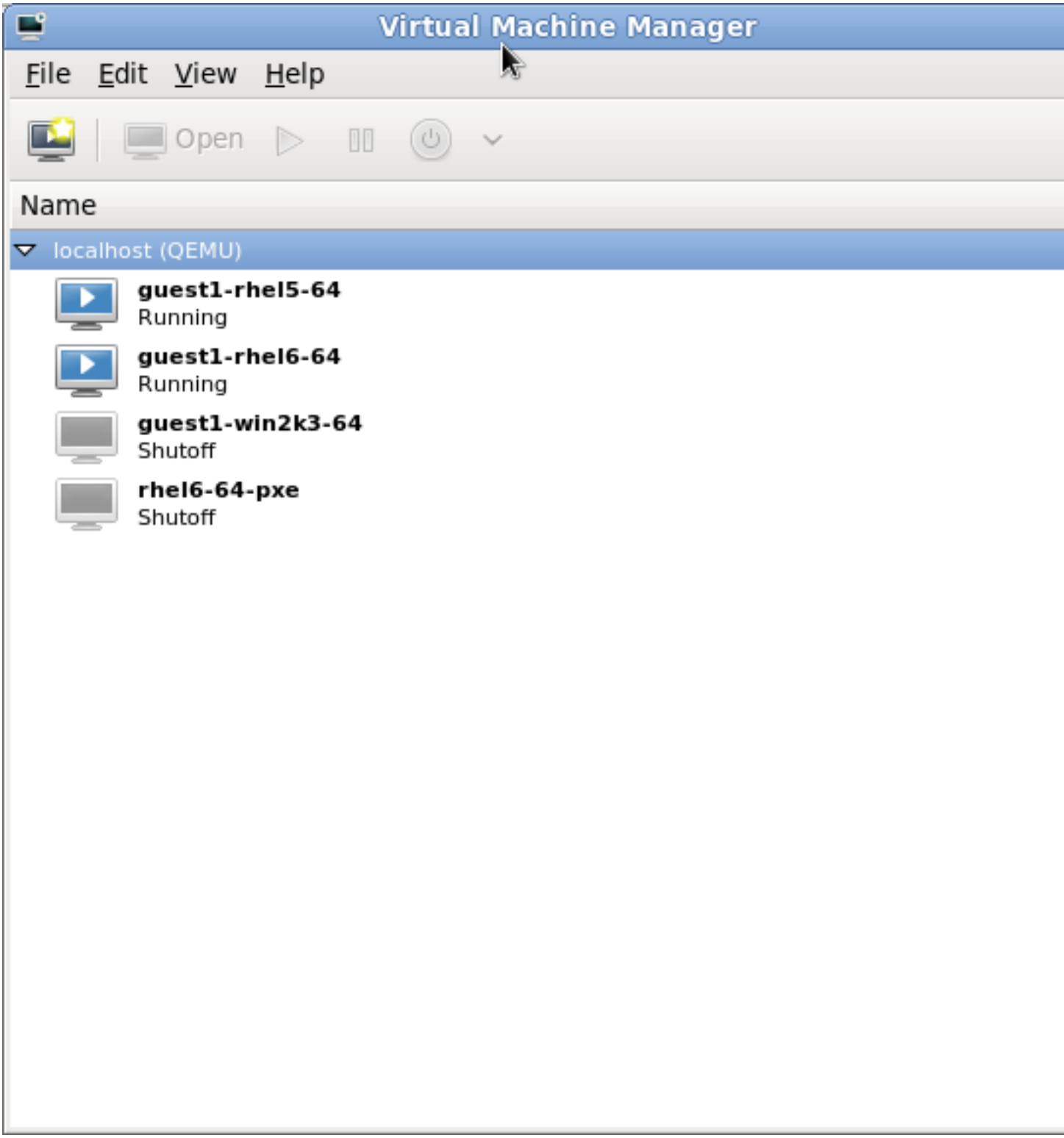


Figure 16.1. Starting **virt-manager**

Alternatively, **virt-manager** can be started remotely using ssh as demonstrated in the following command:

```
ssh -X host's address
[remotehost]# virt-manager
```

Using **ssh** to manage virtual machines and hosts is discussed further in *Section 5.1, "Remote management with SSH"*.

16.2. The Virtual Machine Manager main window

This main window displays all the running guests and resources used by guests. Select a guest by double clicking the guest's name.

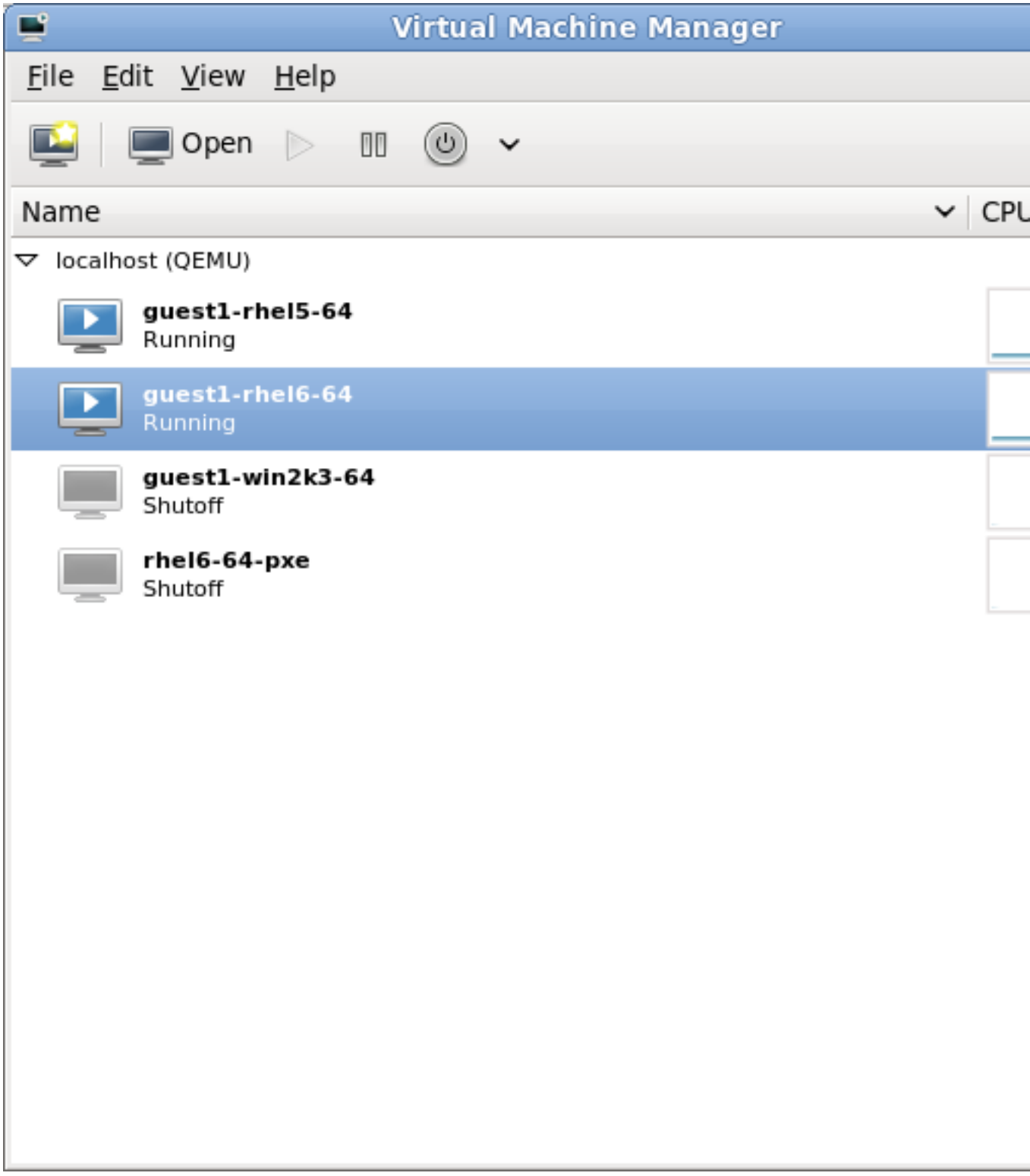


Figure 16.2. Virtual Machine Manager main window

16.3. The virtual hardware details window

The virtual hardware details window displays information about the virtual hardware configured for the guest. Virtual hardware resources can be added, removed and modified in this window. To access the virtual hardware details window, click on the icon in the toolbar.

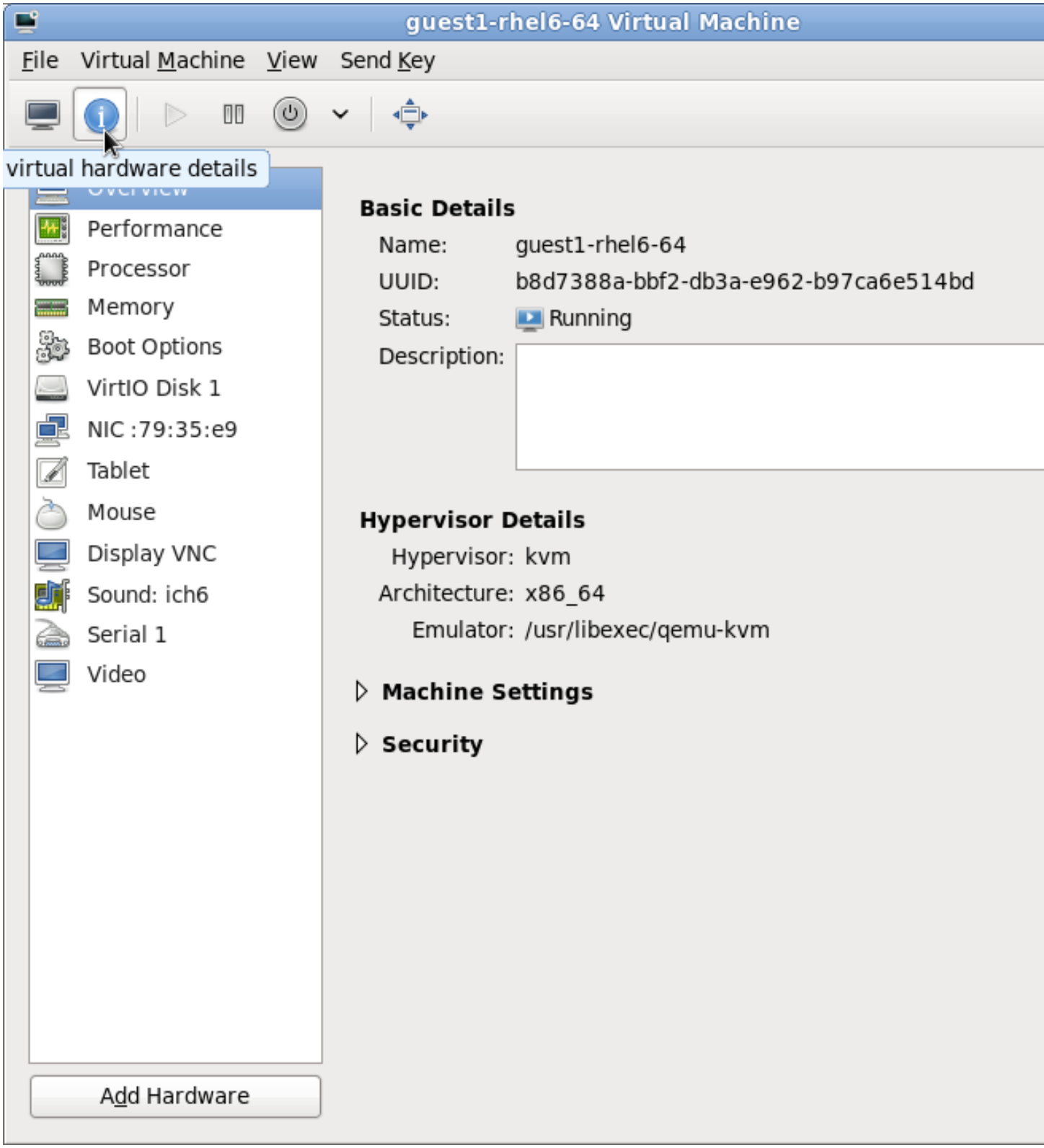


Figure 16.3. The virtual hardware details icon

Clicking the icon displays the virtual hardware details window.

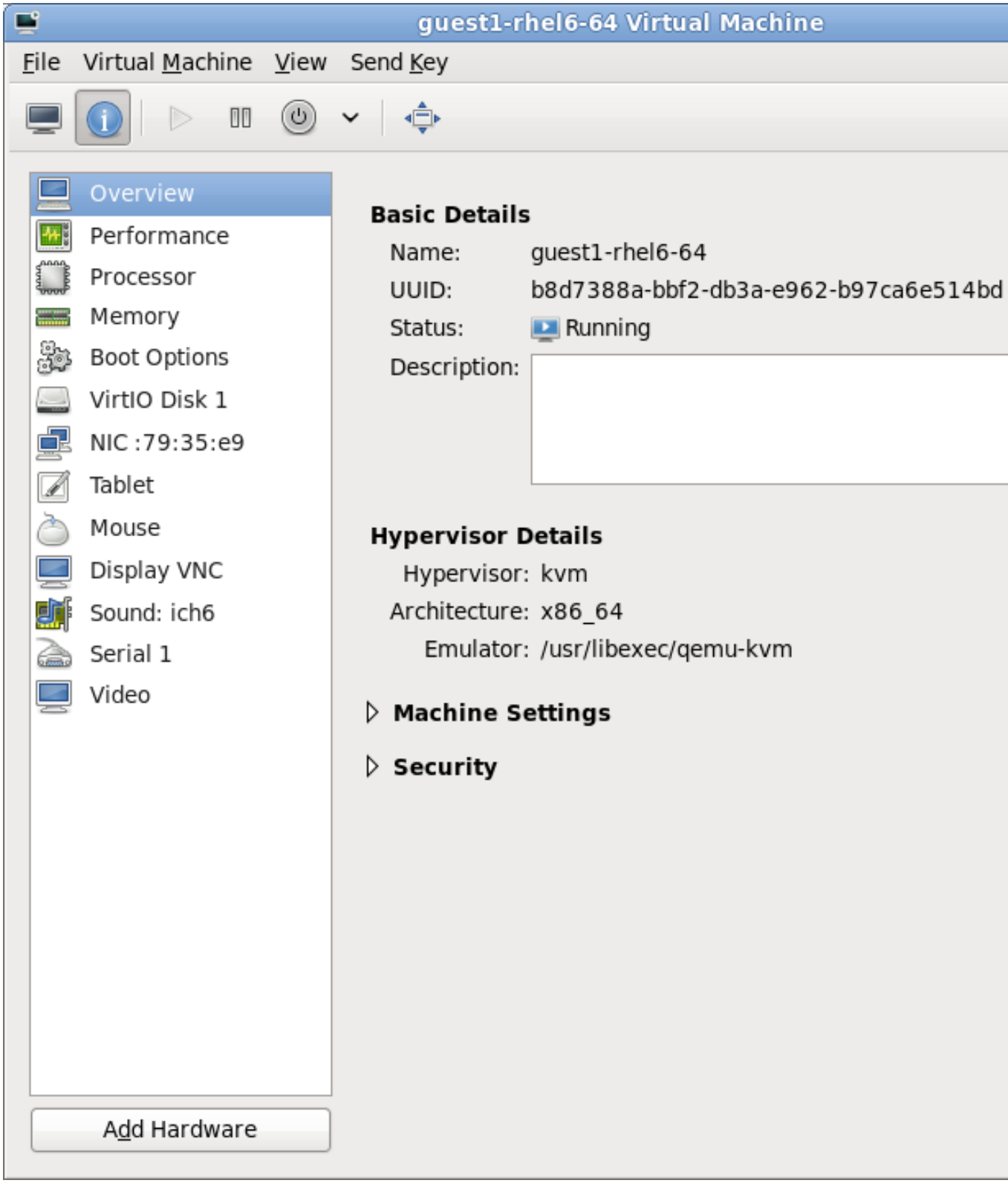


Figure 16.4. The virtual hardware details window

16.4. Virtual Machine graphical console

This window displays a guest's graphical console. Guests can use several different protocols to export their graphical framebuffers: **`virt-manager`** supports **VNC** and **SPICE**. If your virtual machine is set

to require authentication, the Virtual Machine graphical console prompts you for a password before the display appears.

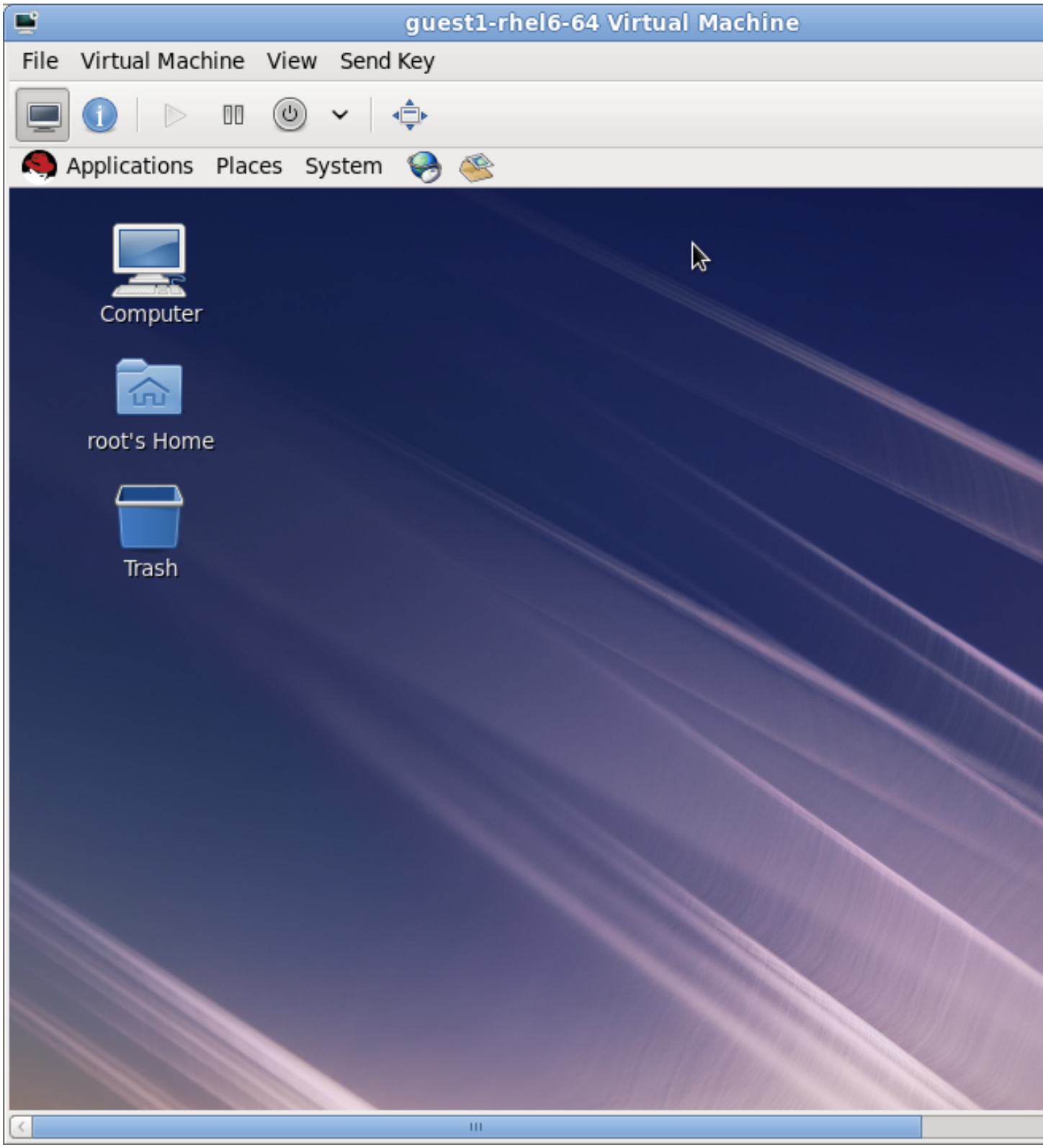


Figure 16.5. Graphical console window

A note on security and VNC

VNC is considered insecure by many security experts, however, several changes have been made to enable the secure usage of VNC for virtualization on Red Hat enterprise Linux. The guest machines only listen to the local host's loopback address (`127.0.0.1`). This ensures only those with shell privileges on the host can access virt-manager and the virtual machine through VNC.

Remote administration can be performed following the instructions in *Chapter 5, Remote management of guests*. TLS can provide enterprise level security for managing guest and host systems.

Your local desktop can intercept key combinations (for example, Ctrl+Alt+F11) to prevent them from being sent to the guest machine. You can use **`virt-manager`**sticky key' capability to send these sequences. You must press any modifier key (Ctrl or Alt) 3 times and the key you specify gets treated as active until the next non-modifier key is pressed. Then you can send Ctrl-Alt-F11 to the guest by entering the key sequence 'Ctrl Ctrl Ctrl Alt+F1'.

SPICE is an alternative to VNC available for Red Hat Enterprise Linux.

16.5. Adding a remote connection

This procedure covers how to set up a connection to a remote system using **`virt-manager`**.

1. To create a new connection open the **File** menu and select the **Add Connection...** menu item.
2. The **Add Connection** wizard appears. Select the hypervisor. For Red Hat Enterprise Linux 6 systems select **QEMU/KVM**. Select Local for the local system or one of the remote connection options and click **Connect**. This example uses Remote tunnel over SSH which works on default

installations. For more information on configuring remote connections refer to *Chapter 5, Remote management of guests*

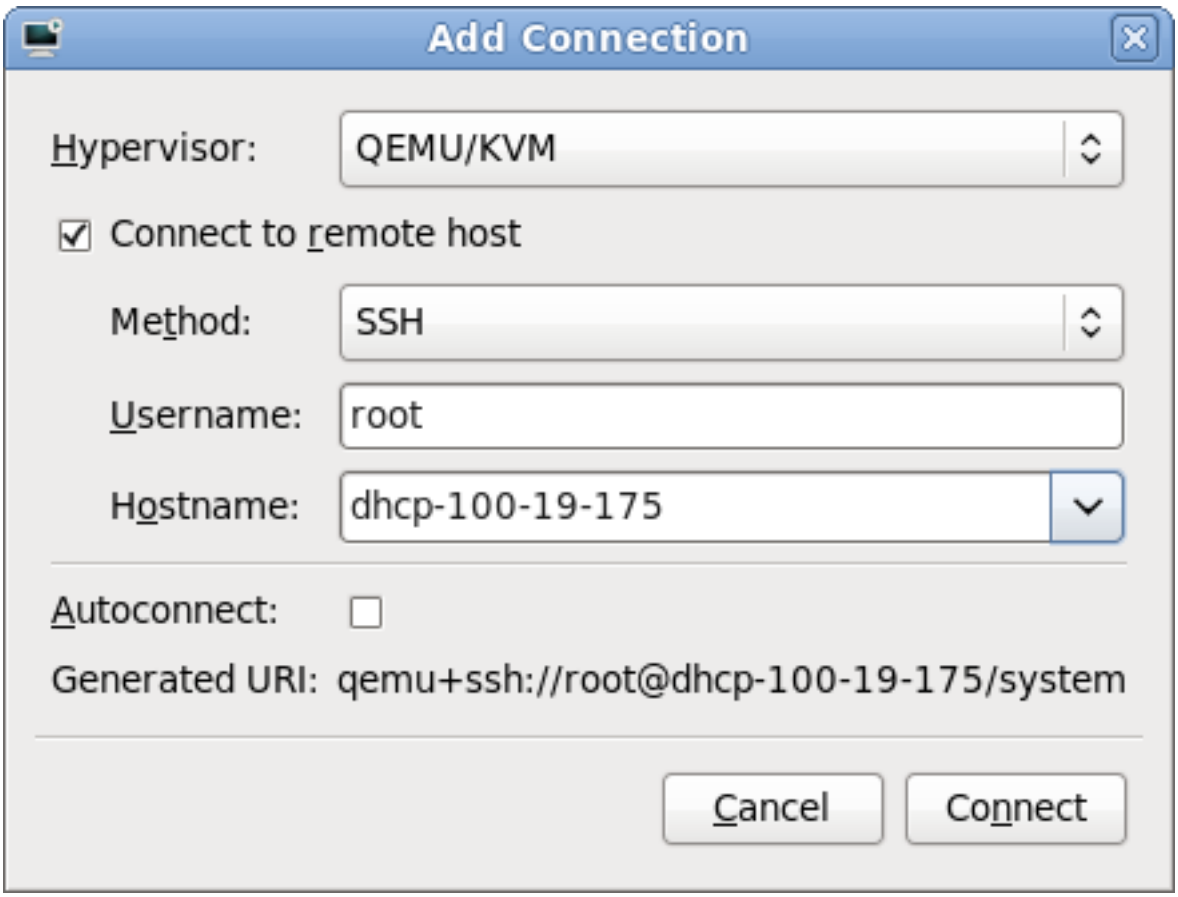


Figure 16.6. Add Connection

3. Enter the root password for the selected host when prompted.

A remote host is now connected and appears in the main **`virt-manager`** window.

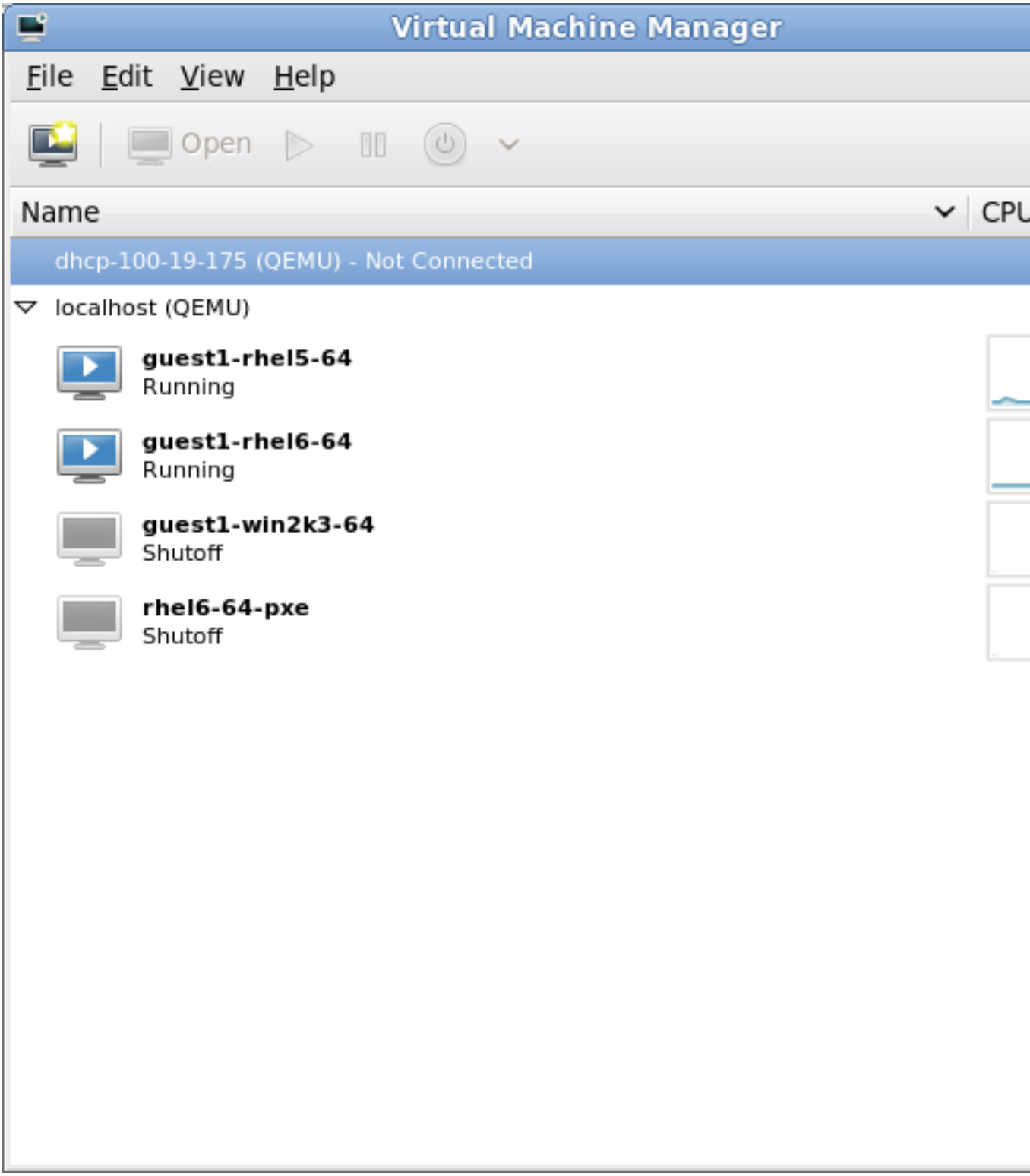


Figure 16.7. Remote host in the main virt-manager window

16.6. Displaying guest details

You can use the Virtual Machine Monitor to view activity information for any virtual machines on your system.

To view a virtual system's details:

1. In the Virtual Machine Manager main window, highlight the virtual machine that you want to view.

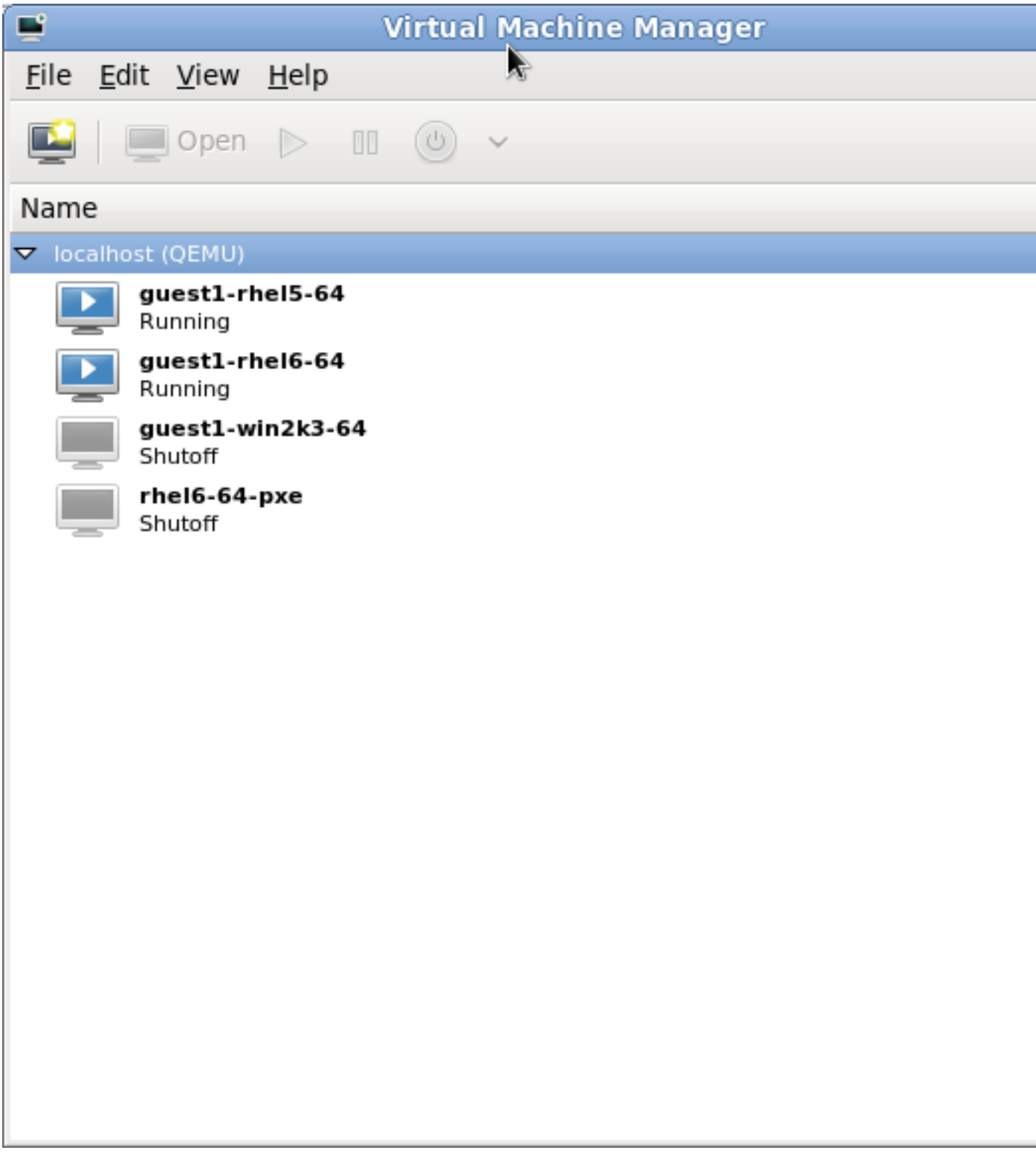


Figure 16.8. Selecting a virtual machine to display

2. From the Virtual Machine Manager **Edit** menu, select **Virtual Machine Details**.

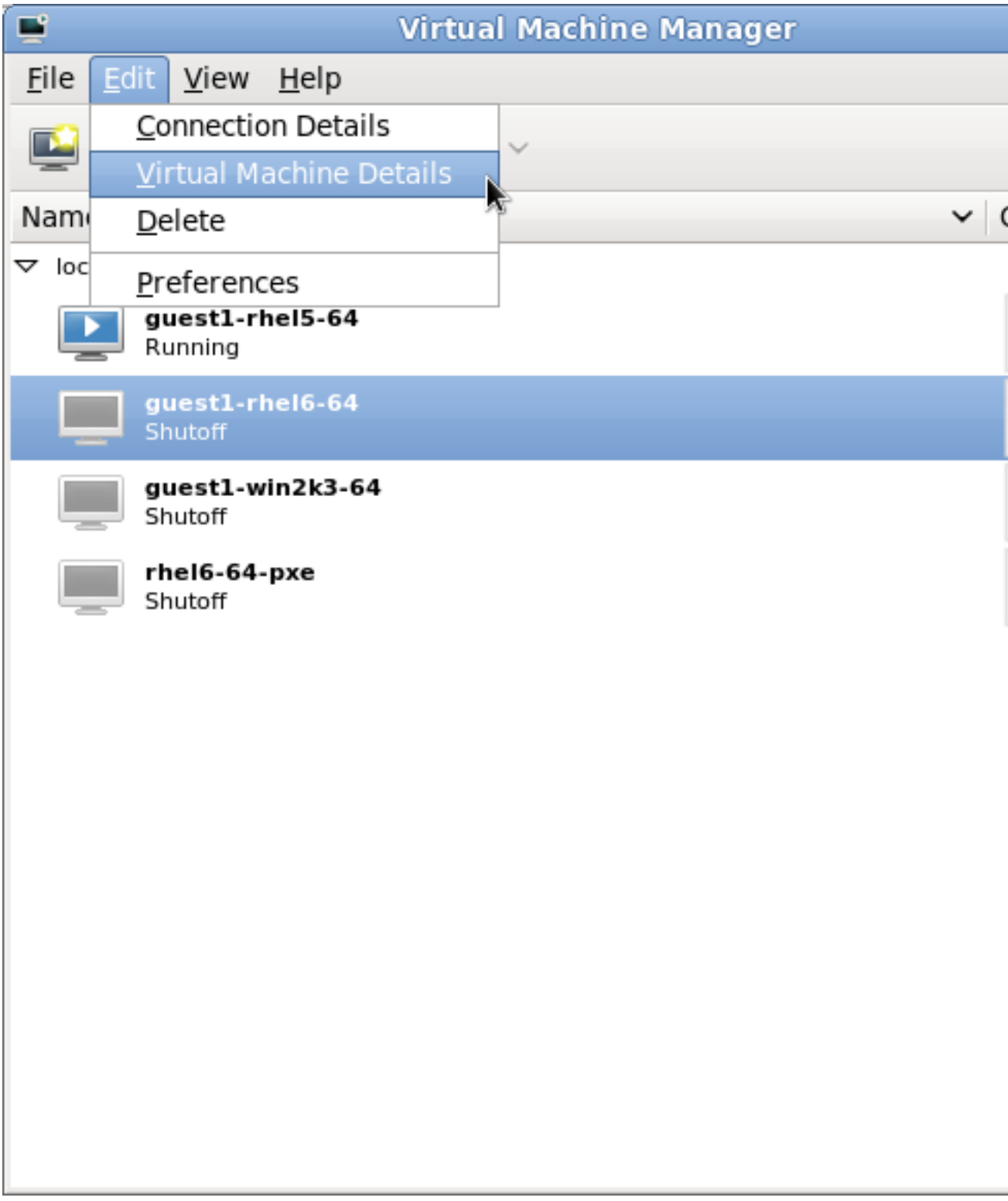


Figure 16.9. Displaying the virtual machine details

On the **Virtual Machine** window, select **Overview** from the navigation pane on the left hand side.

The **Overview** view shows a summary of configuration details for the guest.

3. Select **Performance** from the navigation pane on the left hand side.

   The **Performance** view shows a summary of guest performance, including CPU and Memory usage.

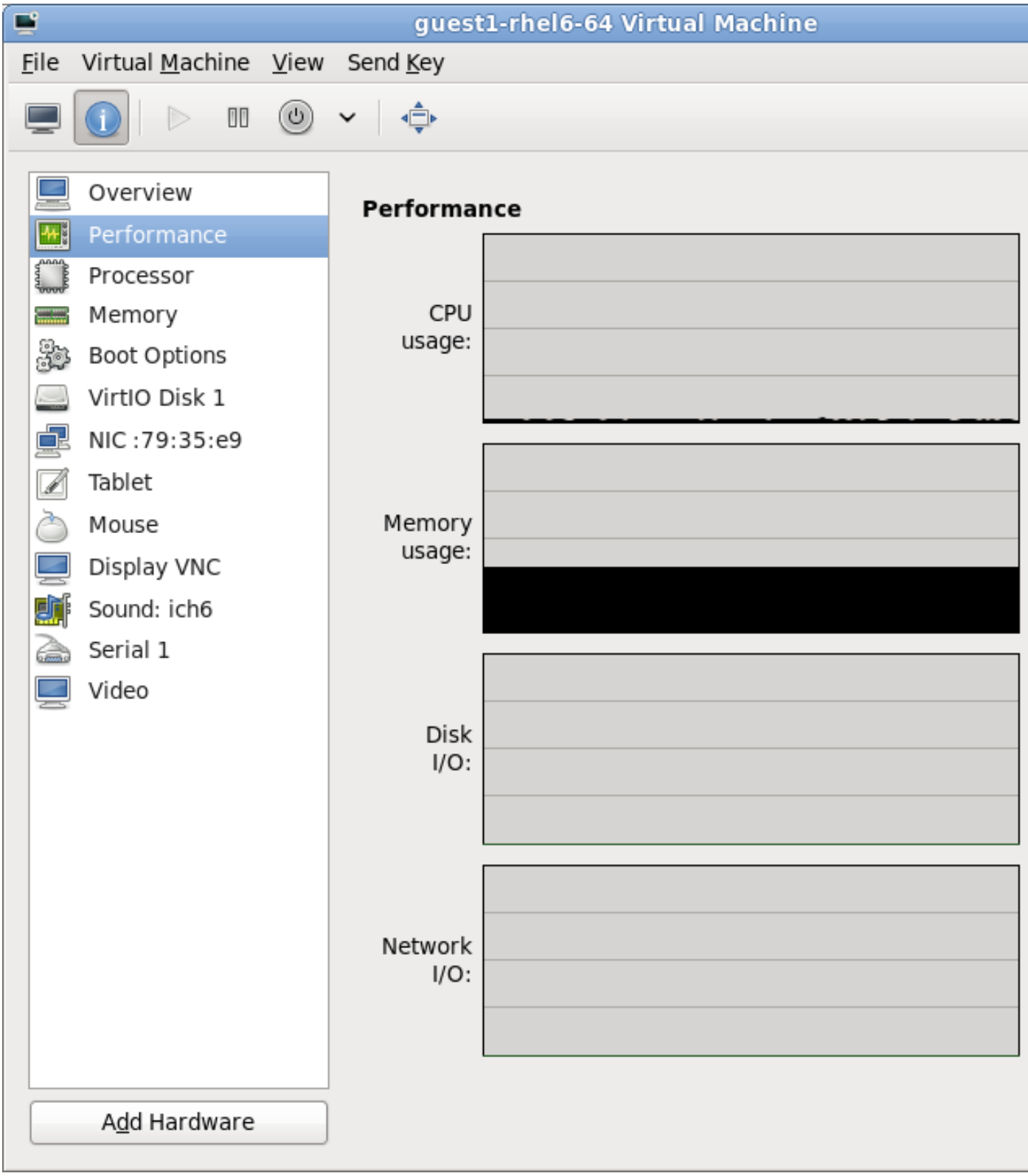


Figure 16.10. Displaying guest performance details

4. Select **Processor** from the navigation pane on the left hand side. The **Processor** view allows you to view or change the current processor allocation.

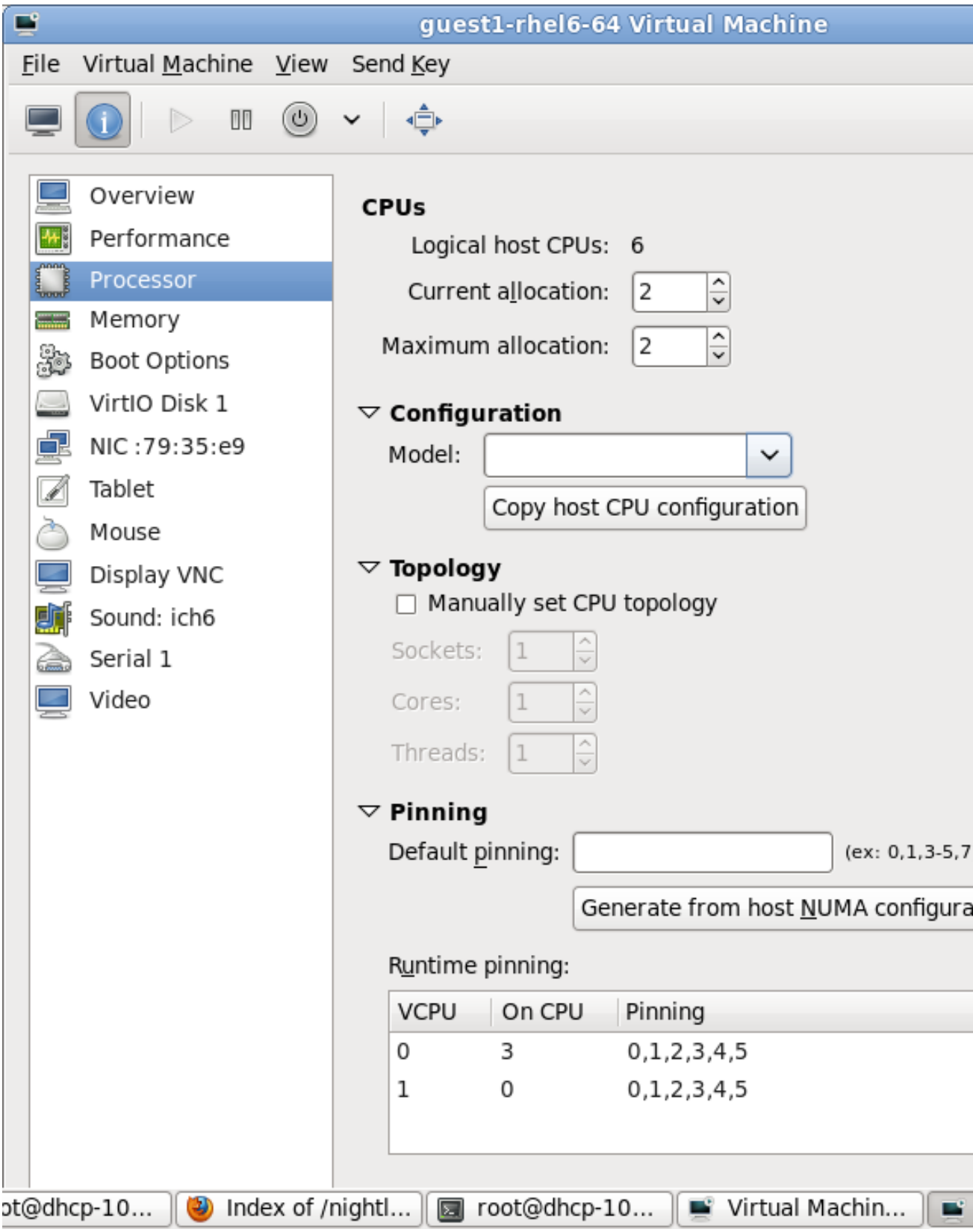


Figure 16.11. Processor allocation panel

5. Select **Memory** from the navigation pane on the left hand side. The **Memory** view allows you to view or change the current memory allocation.

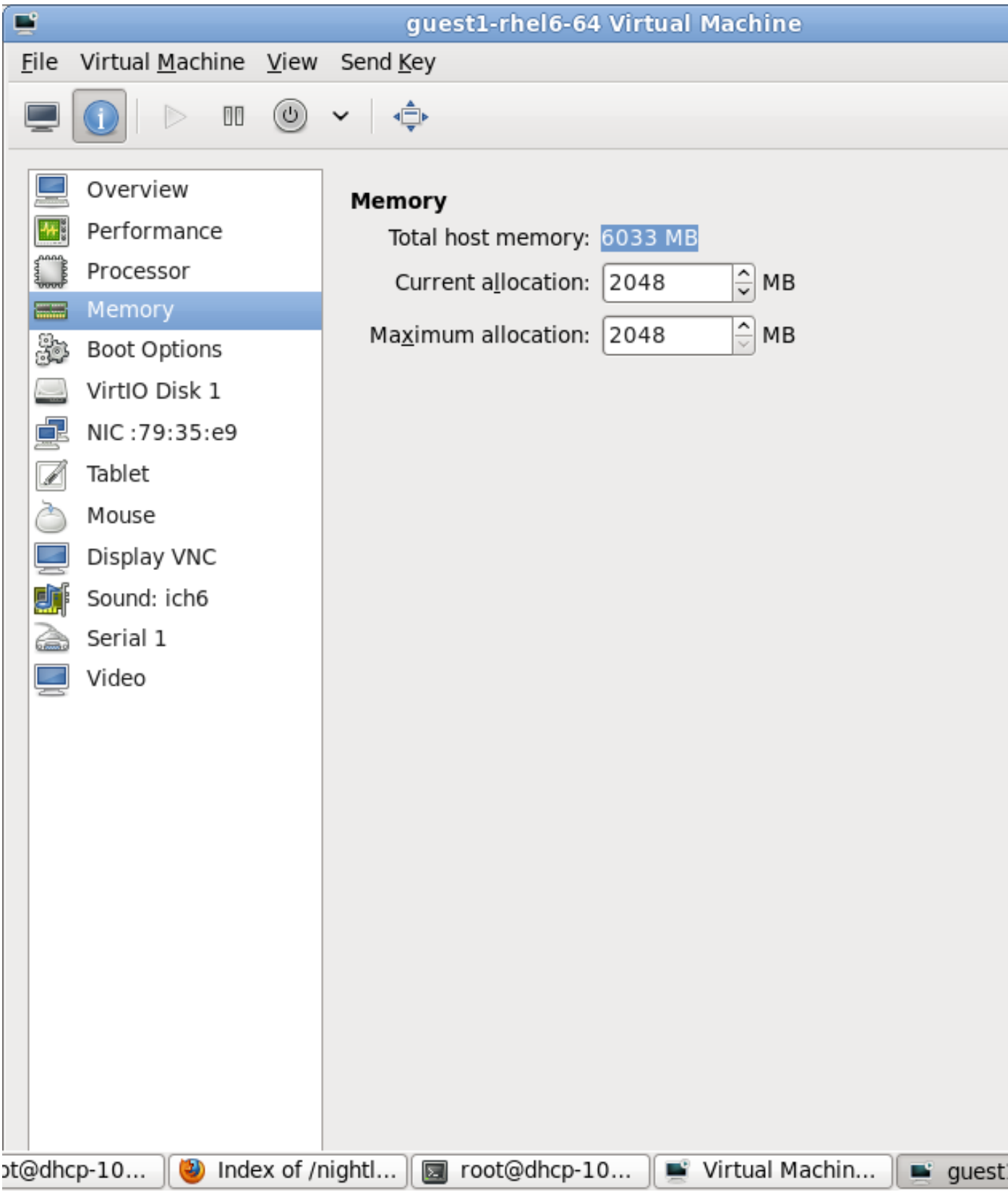


Figure 16.12. Displaying memory allocation

6. Each virtual disk attached to the virtual machine is displayed in the navigation pane. Click on a virtual disk to modify or remove it.

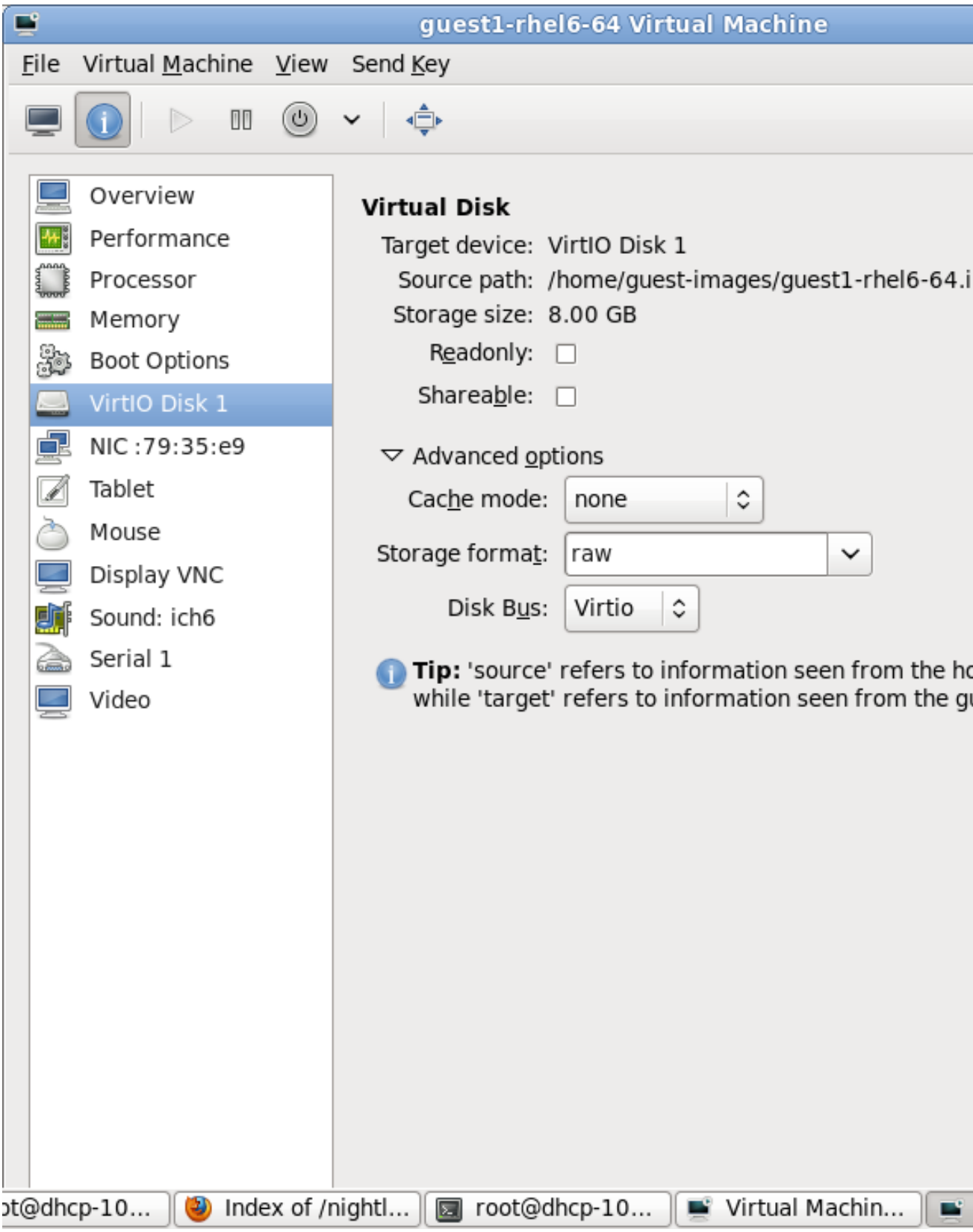


Figure 16.13. Displaying disk configuration

7. Each virtual network interface attached to the virtual machine is displayed in the nagivation pane. Click on a virtual network interface to modify or remove it.

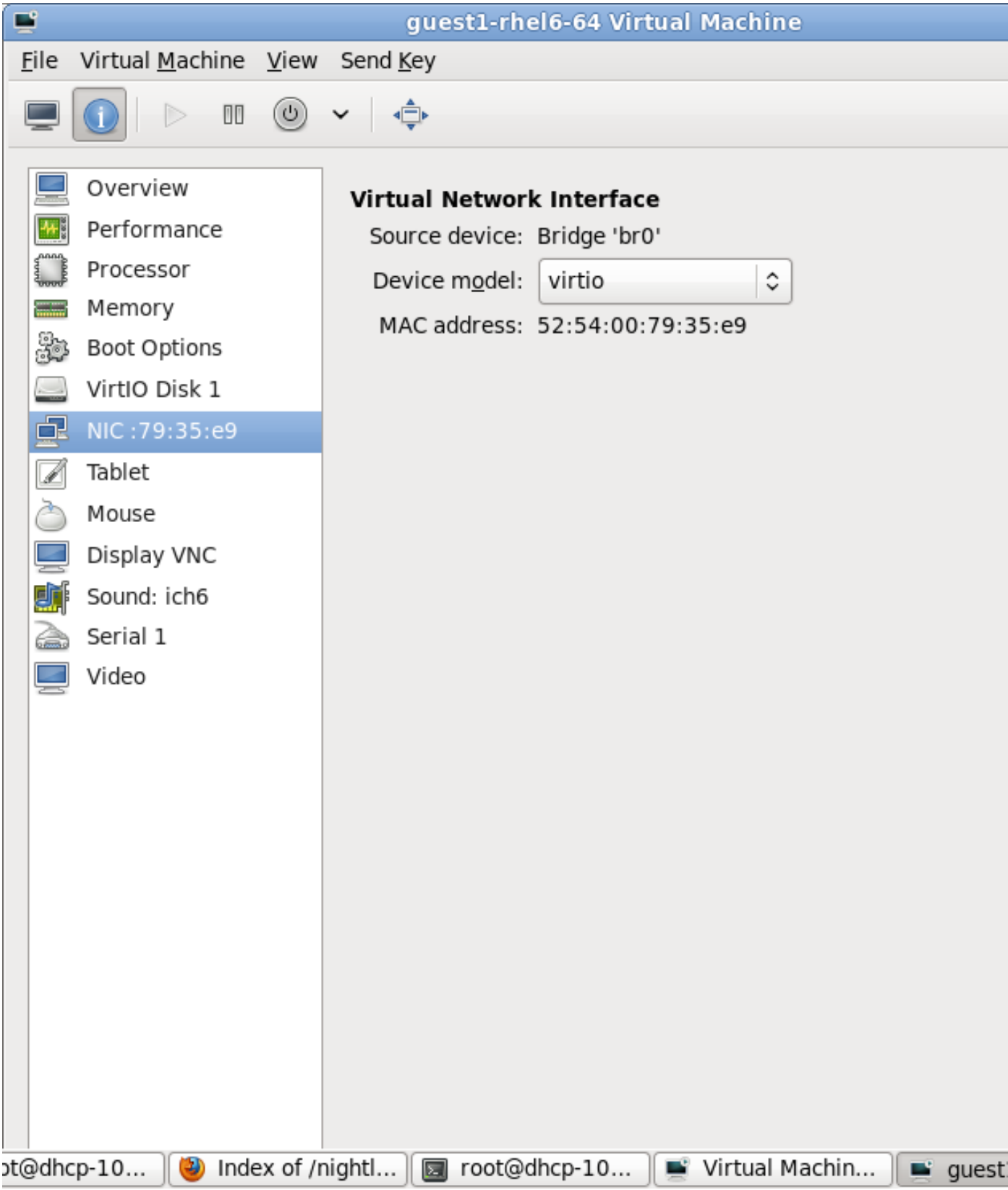


Figure 16.14. Displaying network configuration

16.7. Performance monitoring

Performance monitoring preferences can be modified with **`virt-manager`**'s preferences window.

To configure Performance monitoring:

1. From the **Edit** menu, select **Preferences**.

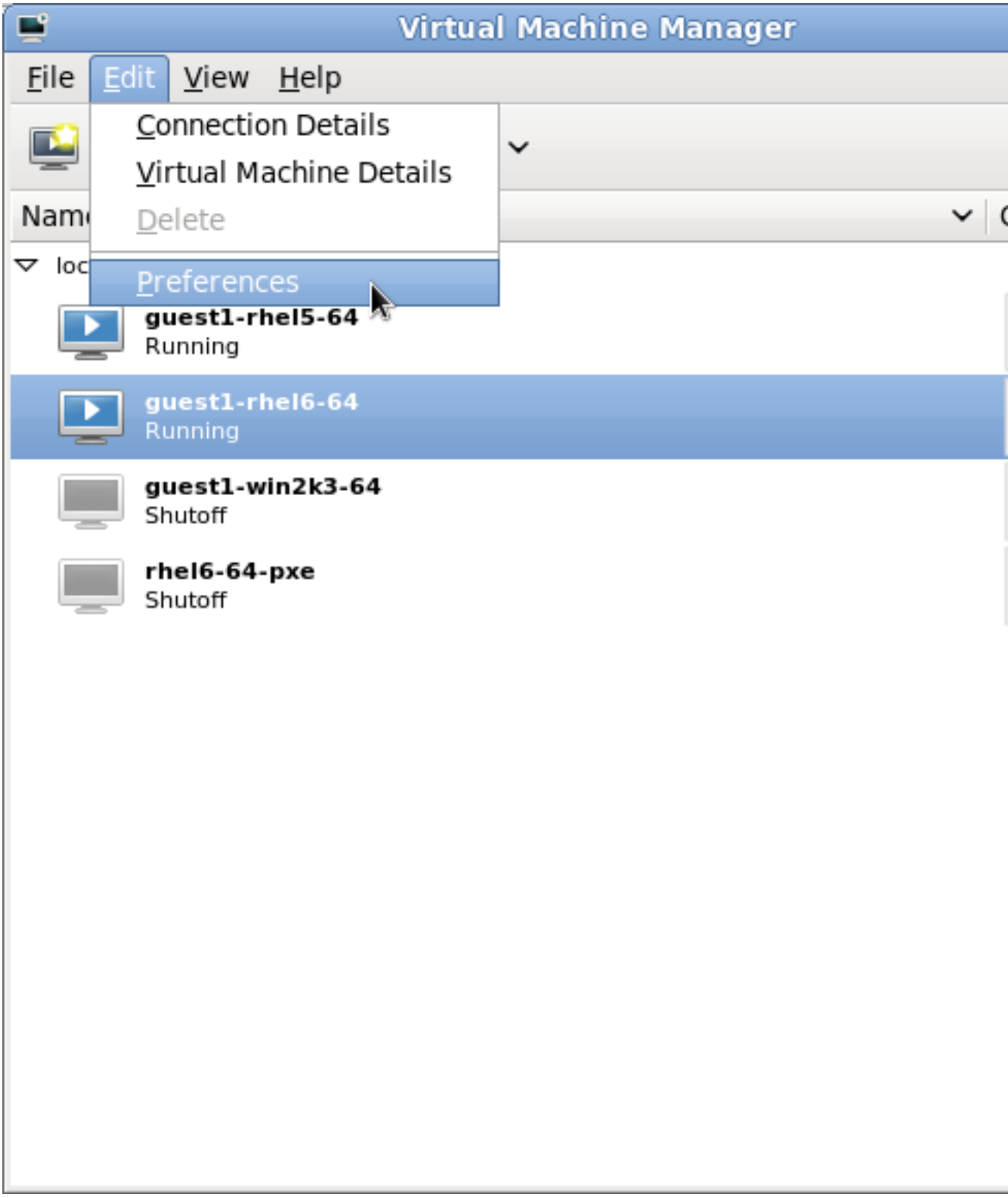


Figure 16.15. Modifying guest preferences

The **Preferences** window appears.

2. From the **Stats** tab specify the time in seconds or stats polling options.

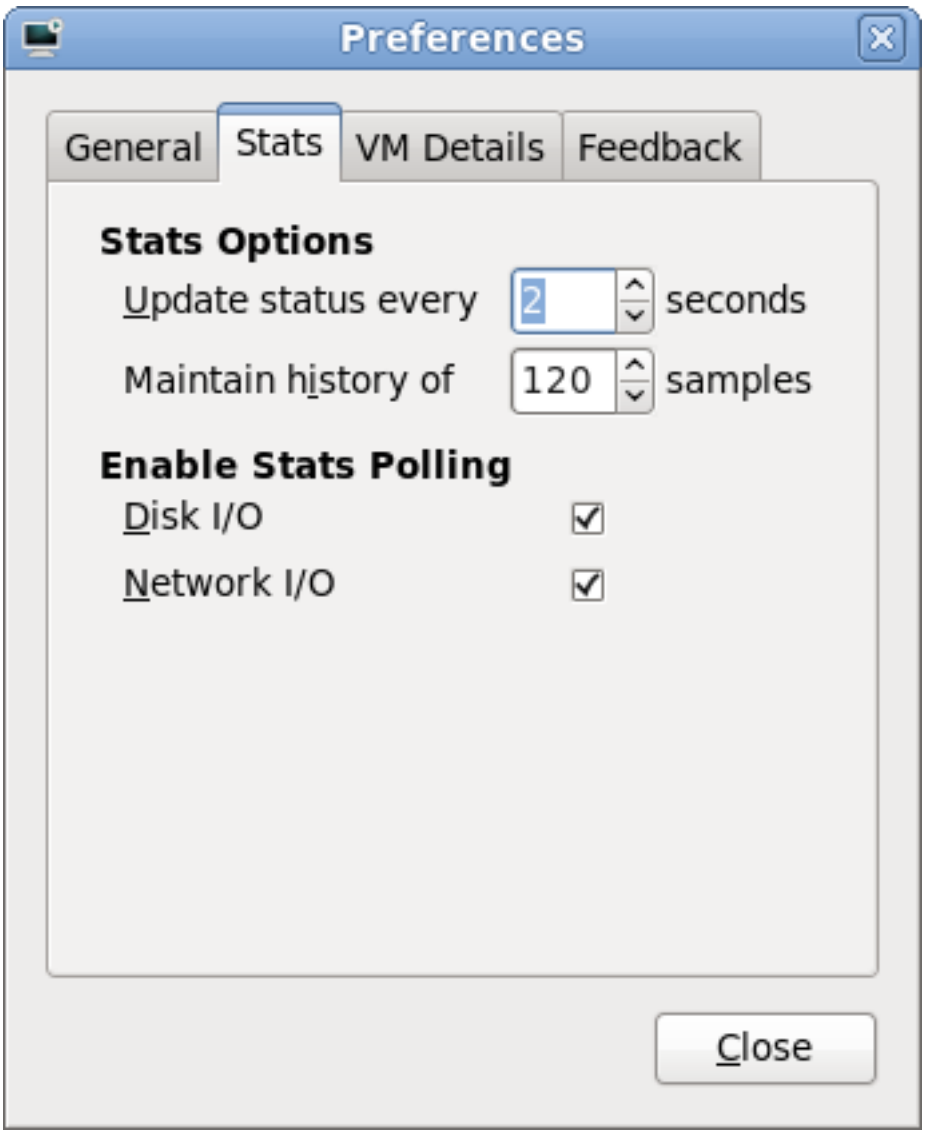


Figure 16.16. Configuring performance monitoring

16.8. Displaying CPU usage

To view the CPU usage for all virtual machines on your system:

1. From the **View** menu, select **Graph**, then the **CPU Usage** check box.

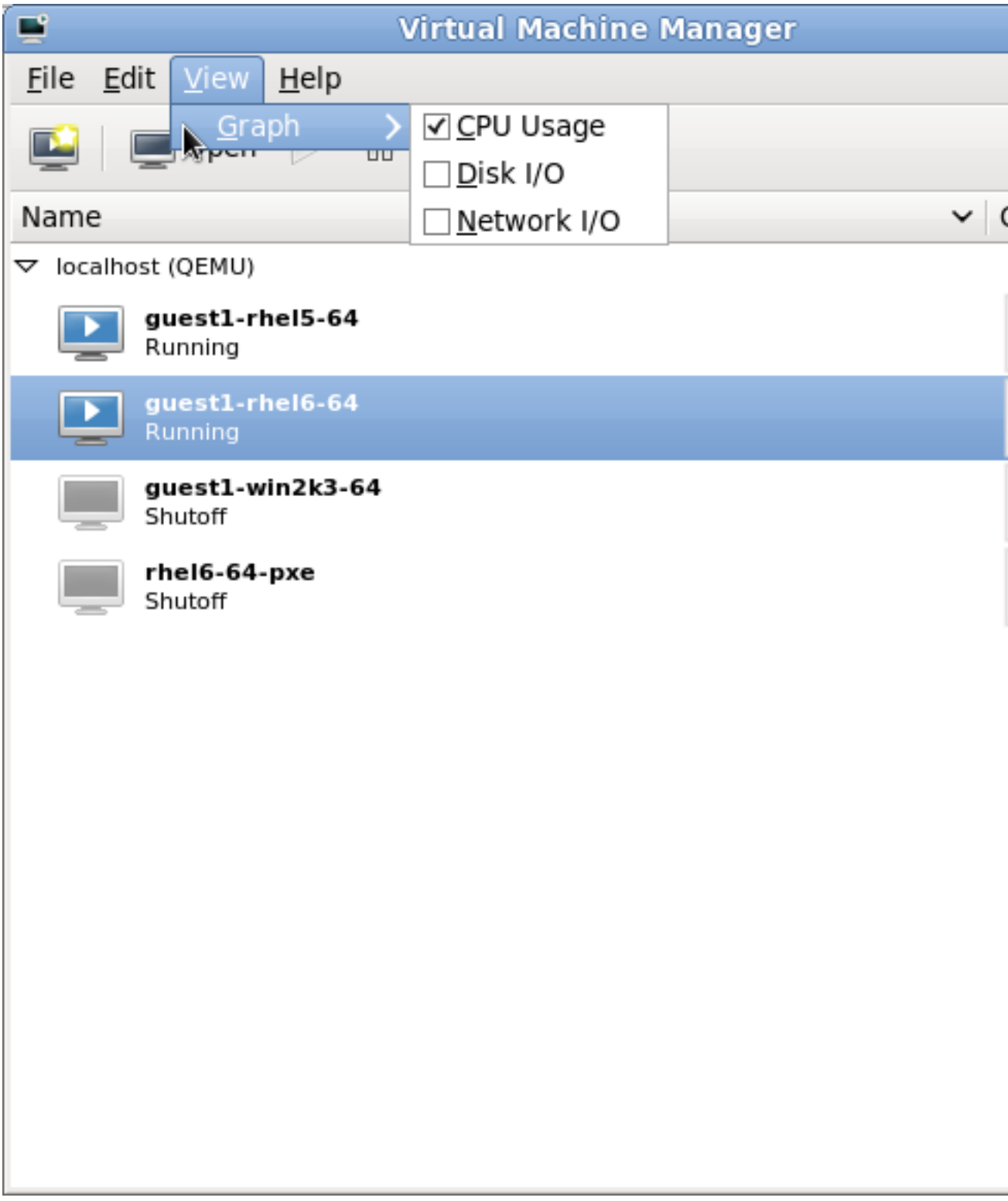


Figure 16.17. Selecting CPU usage

2. The Virtual Machine Manager shows a graph of CPU usage for all virtual machines on your system.

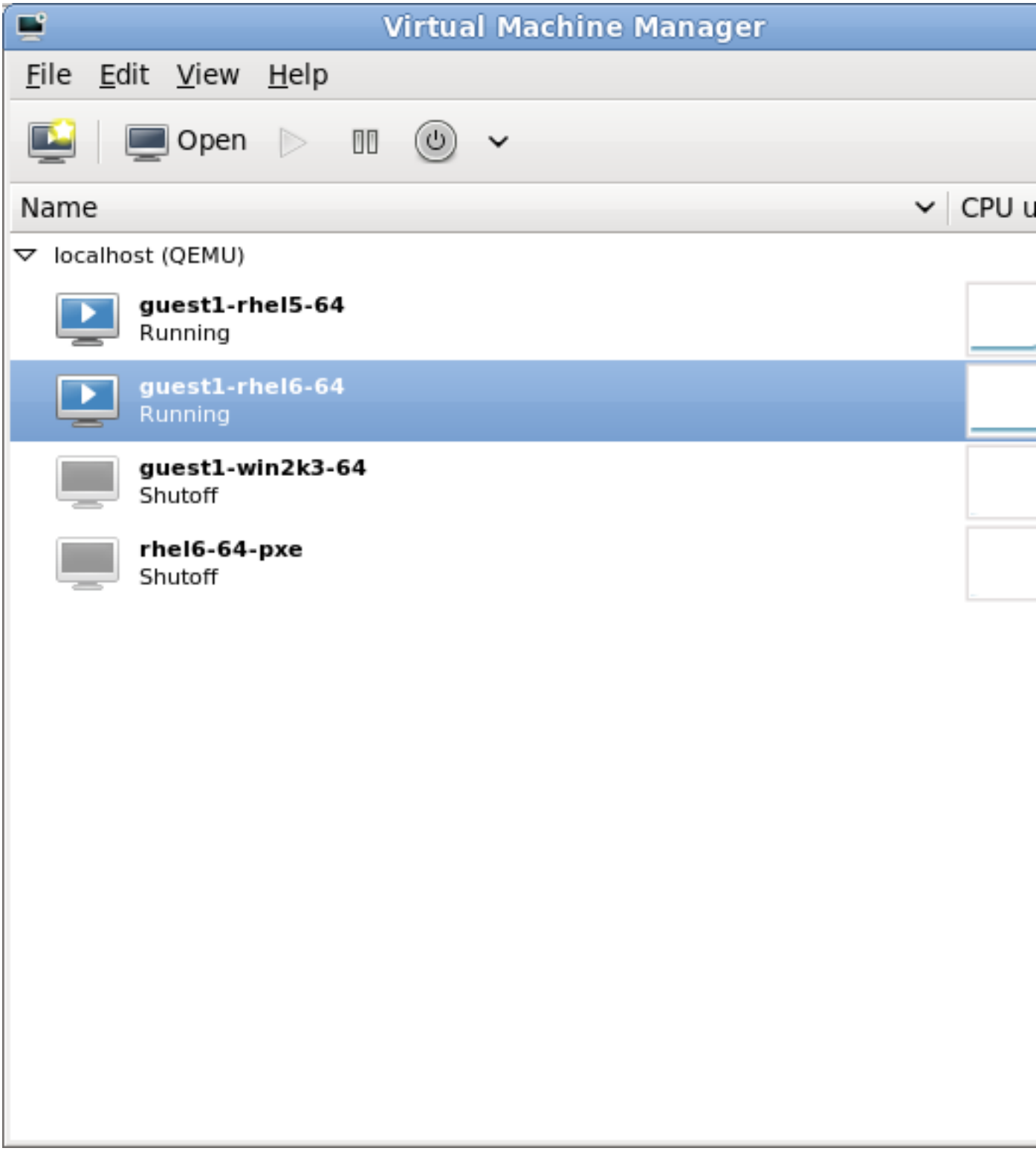


Figure 16.18. Displaying CPU usage

16.9. Displaying Disk I/O

To view the disk I/O for all virtual machines on your system:

1. From the **View** menu, select **Graph**, then the **Disk I/O** check box.

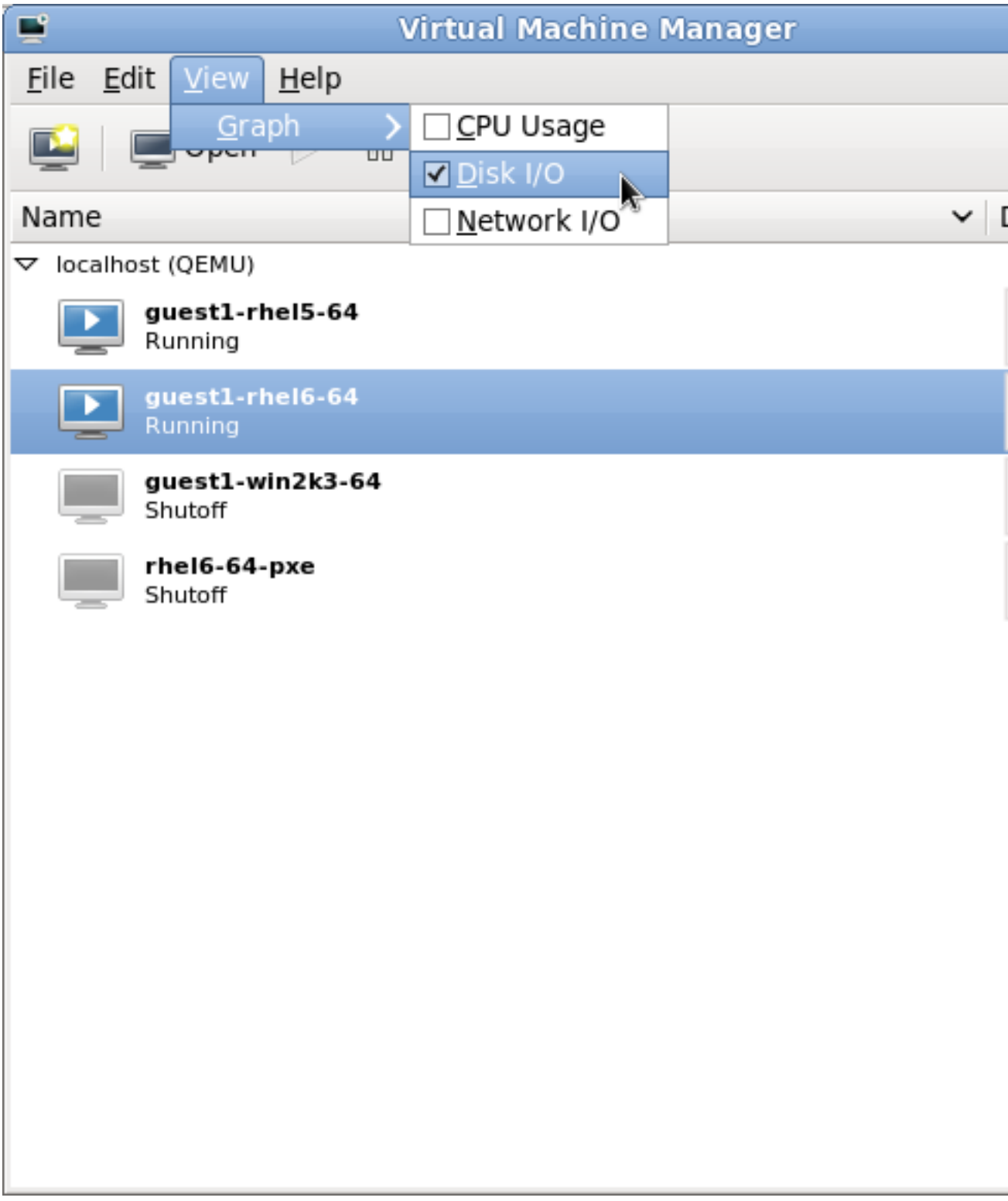


Figure 16.19. Selecting Disk I/O

2. The Virtual Machine Manager shows a graph of Disk I/O for all virtual machines on your system.

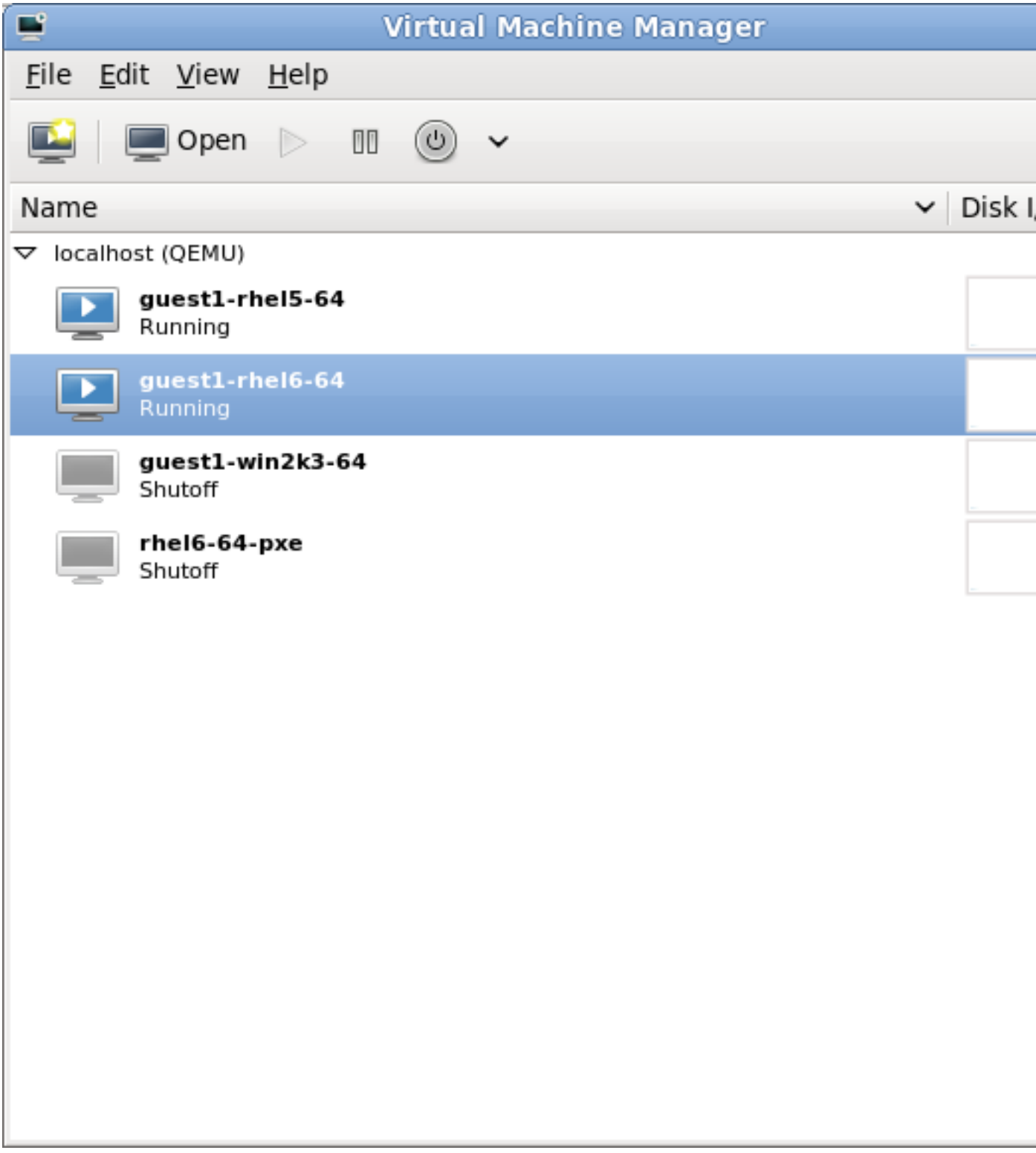


Figure 16.20. Displaying Disk I/O

16.10. Displaying Network I/O

To view the network I/O for all virtual machines on your system:

1. From the **View** menu, select **Graph**, then the **Network I/O** check box.

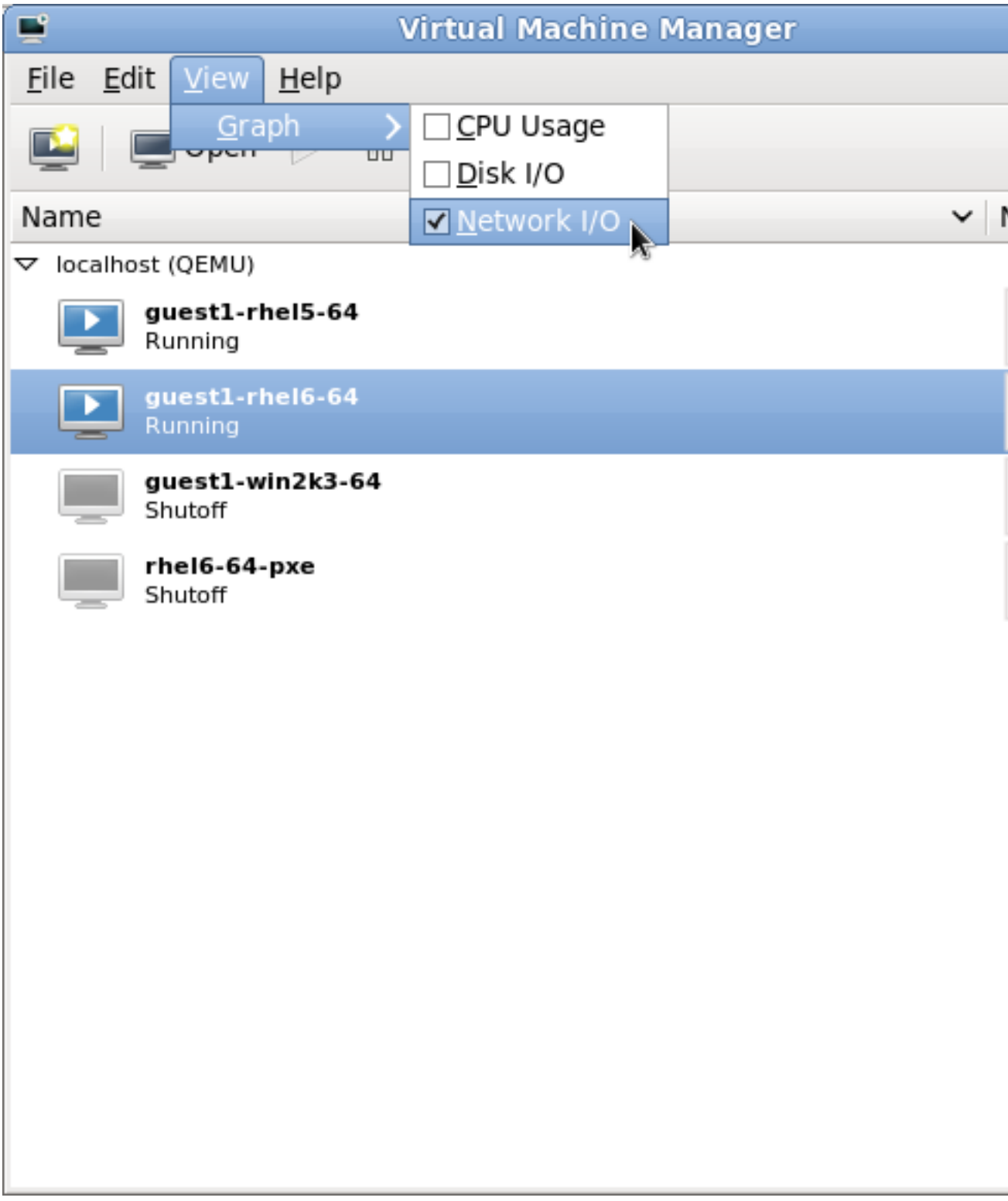


Figure 16.21. Selecting Network I/O

2. The Virtual Machine Manager shows a graph of Network I/O for all virtual machines on your system.

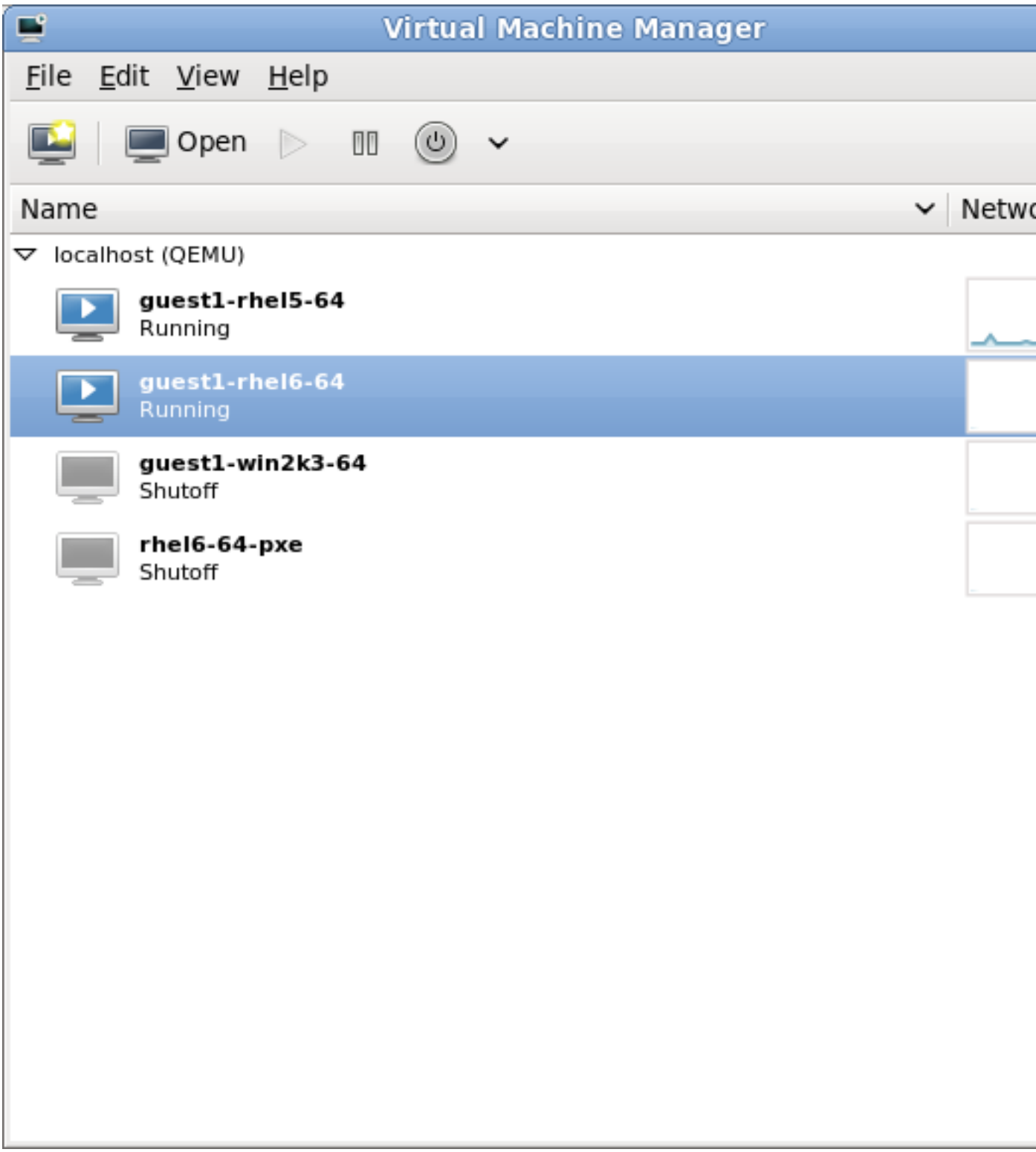


Figure 16.22. Displaying Network I/O

Guest disk access with offline tools

17.1. Introduction

Red Hat Enterprise Linux 6 comes with tools to access, edit and create guest disks or other disk images. There are several uses for these tools, including:

- Viewing or downloading files located on a guest disk.
- Editing or uploading files onto a guest disk.
- Reading or writing guest configuration.
- Reading or writing the Windows Registry in Windows guests.
- Preparing new disk images containing files, directories, file systems, partitions, logical volumes and other options.
- Rescuing and repairing guests that fail to boot or those that need boot configuration changes.
- Monitoring disk usage of guests.
- Auditing compliance of guests, for example to organizational security standards.
- Deploying guests by cloning and modifying templates.
- Reading CD and DVD ISO and floppy disk images.

Warning

You must **never** use these tools to write to a guest or disk image which is attached to a running virtual machine, not even to open such a disk image in write mode. Doing so will result in disk corruption of the guest. The tools try to prevent you from doing this, however do not catch all cases. If there is any suspicion that a guest might be running, it is strongly recommended that the tools not be used, or at least **always** use the tools in read-only mode.

17.2. Terminology

This section explains the terms used throughout this chapter.

- **libguestfs (GUEST FileSystem LIBrary)** - the underlying C library that provides the basic functionality for opening disk images, reading and writing files and so on. You can write C programs directly to this API, but it is quite low level.
- **guestfish (GUEST Filesystem Interactive SHell)** is an interactive shell that you can use from the command line or from shell scripts. It exposes all of the functionality of the libguestfs API.
- Various virt tools are built on top of libguestfs, and these provide a way to perform specific single tasks from the command line. Tools include **virt-df**, **virt-rescue**, **virt-resize** and **virt-edit**.
- **hivex** and **Augeas** are libraries for editing the Windows Registry and Linux configuration files respectively. Although these are separate from libguestfs, much of the value of libguestfs comes from the combination of these tools.

- **guestmount** is an interface between libguestfs and FUSE. It is primarily used to mount file systems from disk images on your host. This functionality is not necessary, but can be useful.

17.3. Installation

To install libguestfs, guestfish, the libguestfs tools, guestmount and support for Windows guests, run the following command:

```
# yum install libguestfs guestfish libguestfs-tools libguestfs-mount libguestfs-winsupport
```

To install every libguestfs-related package including the language bindings, run the following command:

```
# yum install '*guestf*'
```

17.4. The guestfish shell

guestfish is an interactive shell that you can use from the command line or from shell scripts to access guest file systems. All of the functionality of the libguestfs API is available from the shell.

To begin viewing or editing a virtual machine disk image, run the following command, substituting the path to your desired disk image:

```
guestfish --ro -a /path/to/disk/image
```

--ro means that the disk image is opened read-only. This mode is always safe but does not allow write access. Only omit this option when you are **certain** that the guest is not running, or the disk image is not attached to a live guest. It is not possible to use libguestfs to edit a live guest, and attempting to will assuredly result in irreversible disk corruption.

/path/to/disk/image is the path to the disk. This can be a file, a host logical volume (such as /dev/VG/LV), a host device (/dev/cdrom) or a SAN LUN (/dev/sdf3).

Note

libguestfs and guestfish do not require root privileges. You only need to run them as root if the disk image being accessed needs root to read and/or write.

When you start guestfish interactively, it will display this prompt:

```
 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>
```

At the prompt, type **run** to initiate the library and attach the disk image. This can take up to 30 seconds the first time it is done. Subsequent starts will complete much faster.

Note

libguestfs will use hardware virtualization acceleration such as KVM (if available) to speed up this process.

Once the **run** command has been entered, other commands can be used, as the following section demonstrates.

17.4.1. Viewing file systems with guestfish

17.4.1.1. Manual listing and viewing

The **`list-filesystems`** command will list file systems found by libguestfs. This output shows a Red Hat Enterprise Linux 4 disk image:

```
><fs> run
><fs> list-filesystems
/dev/vda1: ext3
/dev/VolGroup00/LogVol00: ext3
/dev/VolGroup00/LogVol01: swap
```

This output shows a Windows disk image:

```
><fs> run
><fs> list-filesystems
/dev/vda1: ntfs
/dev/vda2: ntfs
```

Other useful commands are **`list-devices`**, **`list-partitions`**, **`lvs`**, **`pvs`**, **`vfs-type`** and **`file`**. You can get more information and help on any command by typing **`help`** *command*, as shown in the following output:

```
><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 filesystem type corresponding to the filesystem on
    "device".

    For most filesystems, the result is the name of the Linux VFS module
```

```
    which would be used to mount this filesystem if you mounted it without
    specifying the filesystem type. For example a string such as "ext3" or
    "ntfs".
```

To view the actual contents of a file system, it must first be mounted. This example uses one of the Windows partitions shown in the previous output (**/dev/vda2**), which in this case is known to correspond to the **C:** drive:

```
><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
```

You can use guestfish commands such as **ls**, **ll**, **cat**, **more**, **download** and **tar-out** to view and download files and directories.

Note

There is no concept of a current working directory in this shell. Unlike ordinary shells, you can not for example use the **cd** command to change directories. All paths must be fully qualified starting at the top with a forward slash (*/*) character. Use the *Tab* key to complete paths.

To exit from the guestfish shell, type **exit** or enter **Ctrl+d**.

17.4.1.2. Via guestfish inspection

Instead of listing and mounting file systems by hand, it is possible to let guestfish itself inspect the image and mount the file systems as they would be in the guest. To do this, add the **-i** option on the command line:

```
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]
```

Because guestfish needs to start up the libguestfs back end in order to perform the inspection and mounting, the **run** command is not necessary when using the **-i** option. The **-i** option works for many common Linux and Windows guests.

17.4.1.3. Accessing a guest by name

A guest can be accessed from the command line when you specify its name as known to libvirt (in other words, as it appears in **virsh list --all**). Use the **-d** option to access a guest by its name, with or without the **-i** option:

```
guestfish --ro -d GuestName -i
```

17.4.2. Modifying files with guestfish

To modify files, create directories or make other changes to a guest, first heed the warning at the beginning of this section: *your guest must be shut down*. Editing or changing a running disk with guestfish **will** result in disk corruption. This section gives an example of editing the **/boot/grub/grub.conf** file. When you are sure the guest is shut down you can omit the **--ro** flag in order to get write access via a command such as:

```
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
```

Commands to edit files include **edit**, **vi** and **emacs**. Many commands also exist for creating files and directories, such as **write**, **mkdir**, **upload** and **tar-in**.

17.4.3. Other actions with guestfish

You can also format file systems, create partitions, create and resize LVM logical volumes and much more, with commands such as **mkfs**, **part-add**, **lvresize**, **lvcreate**, **vgcreate** and **pvcreate**.

17.4.4. Shell scripting with guestfish

Once you are familiar with using guestfish interactively, according to your needs, writing shell scripts with it may be useful. The following is a simple shell script to add a new MOTD (message of the day) to a guest:

```
#!/bin/bash -
 set -e
```

```
guestname="$1"

guestfish -d "$guestname" -i <<'EOF'
  write /etc/motd "Welcome to Acme Incorporated."
  chmod 0644 /etc/motd
EOF
```

17.4.5. Augeas and libguestfs scripting

Combining libguestfs with Augeas can help when writing scripts to manipulate Linux guest configuration. For example, the following script uses Augeas to parse the keyboard configuration of a guest, and to print out the layout. Note that this example only works with guests running Red Hat Enterprise Linux:

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$1" -i --ro <<'EOF'
   aug-init / 0
   aug-get /files/etc/sysconfig/keyboard/LAYOUT
 EOF
```

Augeas can also be used to modify configuration files. You can modify the above script to **change** the keyboard layout:

```
#!/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
```

Note the three changes between the two scripts:

1. The **`--ro`** option has been removed in the second example, giving the ability to write to the guest.
2. The **`aug-get`** command has been changed to **`aug-set`** to modify the value instead of fetching it. The new value will be **`"gb"`** (including the quotes).
3. The **`aug-save`** command is used here so Augeas will write the changes out to disk.

Note

More information about Augeas can be found on the website *http://augeas.net*.

guestfish can do much more than we can cover in this introductory document. For example, creating disk images from scratch:

```
guestfish -N fs
```

Or copying out whole directories from a disk image:

```
><fs> copy-out /home /tmp/home
```

For more information see the man page guestfish(1).

17.5. Other commands

This section describes tools that are simpler equivalents to using guestfish to view and edit guest disk images.

- **virt-cat** is similar to the guestfish **download** command. It downloads and displays a single file to the guest. For example:

  ```
  $ virt-cat RHEL3 /etc/ntp.conf | grep ^server
   server     127.127.1.0       # local clock
  ```

- **virt-edit** is similar to the guestfish **edit** command. It can be used to interactively edit a single file within a guest. For example, you may need to edit the **grub.conf** file in a Linux-based guest that will not boot:

  ```
  $ virt-edit LinuxGuest /boot/grub/grub.conf
  ```

  virt-edit has another mode where it can be used to make simple non-interactive changes to a single file. For this, the **-e** option is used. This command, for example, changes the root password in a Linux guest to having no password:

  ```
  $ virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/'
  ```

- **virt-ls** is similar to the guestfish **ls**, **ll** and **find** commands. It is used to list a directory or directories (recursively). For example, the following command would recursively list files and directories under /home in a Linux guest:

  ```
  $ virt-ls -R LinuxGuest /home/ | less
  ```

17.6. virt-rescue: The rescue shell

17.6.1. Introduction

This section describes **virt-rescue**, which can be considered analogous to a rescue CD for virtual machines. It boots a guest into a rescue shell so that maintenance can be performed to correct errors and the guest can be repaired.

There is some overlap between virt-rescue and guestfish. It is important to distinguish their differing uses. virt-rescue is for making interactive, ad-hoc changes using ordinary Linux file system tools. It is particularly suited to rescuing a guest that has gone wrong. virt-rescue cannot be scripted.

In contrast, guestfish is particularly useful for making scripted, structured changes through a formal set of commands (the libguestfs API), although it can also be used interactively.

17.6.2. Running virt-rescue

Before you use **virt-rescue** on a guest, make sure the guest is not running, otherwise disk corruption will occur. When you are sure the guest is not live, enter:

```
virt-rescue GuestName
```

(where GuestName is the guest name as known to libvirt), or:

```
virt-rescue /path/to/disk/image
```

(where the path can be any file, any logical volume, LUN, or so on) containing a guest disk.

You will first see output scroll past, as virt-rescue boots the rescue VM. In the end you will see:

```
Welcome to virt-rescue, the libguestfs rescue shell.

 Note: The contents of / are the rescue appliance.
 You have to mount the guest'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>
```

The shell prompt here is an ordinary bash shell, and a reduced set of ordinary Red Hat Enterprise Linux commands is available. For example, you can enter:

```
><rescue> fdisk -l /dev/vda
```

The previous command will list disk partitions. To mount a file system, it is suggested that you mount it under **/sysroot**, which is an empty directory in the rescue machine for the user to mount anything you like. Note that the files under **/** are files from the rescue VM itself:

```
><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
 [...]
```

When you are finished rescuing the guest, exit the shell by entering **exit** or **Ctrl+d**.

virt-rescue has many command line options. The options most often used are:

- **--ro**: Operate in read-only mode on the guest. No changes will be saved. You can use this to experiment with the guest. As soon as you exit from the shell, all of your changes are discarded.

- **--network**: Enable network access from the rescue shell. Use this if you need to, for example, download RPM or other files into the guest.

17.7. virt-df: Monitoring disk usage

17.7.1. Introduction

This section describes **virt-df**, which displays file system usage from a disk image or a guest. It is similar to the Linux **df** command, but for virtual machines.

17.7.2. Running virt-df

To display file system usage for all file systems found in a disk image, enter the following:

```
# 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%
```

(Where **/dev/vg_guests/RHEL6** is a Red Hat Enterprise Linux 4 guest disk image. The path in this case is the host logical volume where this disk image is located.)

You can also use **virt-df** on its own to list information about all of your guests (ie. those known to libvirt). The **virt-df** command recognizes some of the same options as the standard **df** such as **-h** (human-readable) and **-i** (show inodes instead of blocks).

virt-df also works on Windows guests:

```
# 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%
```

Note

You can use **virt-df** safely on live guests, since it only needs read-only access. However, you should not expect the numbers to be precisely the same as those from a **df** command running inside the guest. This is because what is on disk will be slightly out of synch with the state of the live guest. Nevertheless it should be a good enough approximation for analysis and monitoring purposes.

virt-df is designed to allow you to integrate the statistics into monitoring tools, databases and so on. This allows system administrators to generate reports on trends in disk usage, and alerts if a guest is about to run out of disk space. To do this you should use the **--csv** option to generate machine-readable Comma-Separated-Values (CSV) output. CSV output is readable by most databases,

spreadsheet software and a variety of other tools and programming languages. The raw CSV looks like the following:

```
# 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%
```

For resources and ideas on how to process this output to produce trends and alerts, refer to the following URL: *http://virt-tools.org/learning/advanced-virt-df/*.

17.8. virt-resize: resizing guests offline

17.8.1. Introduction

This section describes **virt-resize**, a tool for expanding or shrinking guests. It only works for guests which are offline (shut down). It works by copying the guest image and leaving the original disk image untouched. This is ideal because you can use the original image as a backup, however there is a trade-off as you need twice the amount of disk space.

17.8.2. Expanding a disk image

This section demonstrates a simple case of expanding a disk image:

1. Locate the disk image to be resized. You can use the command **virsh dumpxml GuestName** for a libvirt guest.

2. Decide on how you wish to expand the guest. Run **virt-df -h** and **virt-list-partitions -lh** on the guest disk, as shown in the following output:

```
# 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
```

This example will demonstrate how to:

- Increase the size of the first (boot) partition, from approximately 100MB to 500MB.
- Increase the total disk size from 8GB to 16GB.
- Expand the second partition to fill the remaining space.
- Expand **/dev/VolGroup00/LogVol00** to fill the new space in the second partition.

1. Make sure the guest is shut down.

2. Rename the original disk as the backup. How you do this depends on the host storage environment for the original disk. If it is stored as a file, use the **mv** command. For logical volumes (as demonstrated in this example), use **lvrename**:

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. Create the new disk. The requirements in this example are to expand the total disk size up to 16GB. Since logical volumes are used here, the following command is used:

```
# lvcreate -L 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. The requirements from step 2 are expressed by this command:

```
# virt-resize \
       /dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 \
       --resize /dev/sda1=500M \
       --expand /dev/sda2 \
       --LV-expand /dev/VolGroup00/LogVol00
```

   The first two arguments are the input disk and output disk. **`--resize /dev/sda1=500M`** resizes the first partition up to 500MB. **`--expand /dev/sda2`** expands the second partition to fill all remaining space. **`--LV-expand /dev/VolGroup00/LogVol00`** expands the guest logical volume to fill the extra space in the second partition.

   `virt-resize` describes what it is doing in the output:

```
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. Try to boot the virtual machine. If it works (and after testing it thoroughly) you can delete the backup disk. If it fails, shut down the virtual machine, delete the new disk, and rename the backup disk back to its original name.

6. Use **`virt-df`** and/or **`virt-list-partitions`** to show the new size:

```
# 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%
```

Resizing guests is not an exact science. If **virt-resize** fails, there are a number of tips that you can review and attempt in the virt-resize(1) man page. For some older Red Hat Enterprise Linux guests, you may need to pay particular attention to the tip regarding GRUB.

17.9. virt-inspector: inspecting guests

17.9.1. Introduction

virt-inspector is a tool for inspecting a disk image to find out what operating system it contains.

Note

Red Hat Enterprise Linux 6.2 ships with two variations of this progam: **virt-inspector** is the original program as found in Red Hat Enteprise Linux 6.0 and is now deprecated upstream. **virt-inspector2** is the same as the new upstream **virt-inspector** program.

17.9.2. Installation

To install virt-inspector and the documentation, enter the following command:

```
# yum install libguestfs-tools libguestfs-devel
```

To process Windows guests you must also install *libguestfs-winsupport*. The documentation, including example XML output and a Relax-NG schema for the output, will be installed in **/usr/share/doc/libguestfs-devel-*/** where "*" is replaced by the version number of libguestfs.

17.9.3. Running virt-inspector

You can run **virt-inspector** against any disk image or libvirt guest as shown in the following example:

```
virt-inspector --xml disk.img > report.xml
```

Or as shown here:

```
virt-inspector --xml GuestName > report.xml
```

The result will be an XML report (**report.xml**). The main components of the XML file are a top-level <operatingsytems> element containing usually a single <operatingsystem> element, similar to the following:

```
<operatingsystems>
  <operatingsystem>

    <!-- the type of operating system and Linux distribution -->
    <name>linux</name>
    <distro>fedora</distro>
```

```
    <!-- the name, version and architecture -->
    <product_name>Fedora release 12 (Constantine)</product_name>
    <major_version>12</major_version>
    <arch>x86_64</arch>

    <!-- how the filesystems would be mounted when live -->
    <mountpoints>
      <mountpoint dev="/dev/vg_f12x64/lv_root">/</mountpoint>
      <mountpoint dev="/dev/sda1">/boot</mountpoint>
    </mountpoints>

    <!-- the filesystems -->
    <filesystems>
      <filesystem dev="/dev/sda1">
        <type>ext4</type>
      </filesystem>
      <filesystem dev="/dev/vg_f12x64/lv_root">
        <type>ext4</type>
      </filesystem>
      <filesystem dev="/dev/vg_f12x64/lv_swap">
        <type>swap</type>
      </filesystem>
    </filesystems>

    <!-- packages installed -->
    <applications>
      <application>
        <name>firefox</name>
        <version>3.5.5</version>
        <release>1.fc12</release>
      </application>
    </applications>

  </operatingsystem>
</operatingsystems>
```

Processing these reports is best done using W3C standard XPath queries. Red Hat Enterprise Linux 6 comes with a command line program (**xpath**) which can be used for simple instances; however, for long-term and advanced usage, you should consider using an XPath library along with your favorite programming language.

As an example, you can list out all file system devices using the following XPath query:

```
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"
```

Or list the names of all applications installed by entering:

```
 virt-inspector --xml GuestName | xpath //application/name
 [...long list...]
```

The version of virt-inspector in Red Hat Enterprise Linux 6.2 has a number of shortcomings. It has limited support for Windows guests and the XML output is over-complicated for Linux guests. These limitations will addressed in future releases.

17.10. virt-win-reg: Reading and editing the Windows Registry

17.10.1. Introduction

virt-win-reg is a tool that manipulates the Registry in Windows guests. It can be used to read out registry keys. You can also use it to make changes to the Registry, but you must **never** try to do this for live/running guests, as it will result in disk corruption.

17.10.2. Installation

To use **virt-win-reg** you must run the following:

```
# yum install libguestfs-tools libguestfs-winsupport
```

17.10.3. Using virt-win-reg

To read out Registry keys, specify the name of the guest (or its disk image) and the name of the Registry key. You must use single quotes to surround the name of the desired key:

```
# virt-win-reg WindowsGuest \
    'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall' \
    | less
```

The output is in the standard text-based format used by **.REG** files on Windows.

Note

Hex-quoting is used for strings because the format does not properly define a portable encoding method for strings. This is the only way to ensure fidelity when transporting **.REG** files from one machine to another.

You can make hex-quoted strings printable by piping the output of **virt-win-reg** through this simple Perl script:

```
perl -MEncode -pe's?hex\((\d+)\):(\S+)?$t=$1;$_=$2;s,\,,,g;"str($t):
\"".decode(utf16le=>pack("H*",$_))."\""?eg'
```

To merge changes into the Windows Registry of an offline guest, you must first prepare a **.REG** file. There is a great deal of documentation about doing this available from MSDN, and there is a good summary in the following Wikipedia page: *https://secure.wikimedia.org/wikipedia/en/wiki/Windows_Registry#.REG_files*. When you have prepared a **.REG** file, enter the following:

```
# virt-win-reg --merge WindowsGuest input.reg
```

This will update the registry in the guest.

17.11. Using the API from Programming Languages

The libguestfs API can be used directly from the following languages in Red Hat Enterprise Linux 6.2: C, C++, Perl, Python, Java, Ruby and OCaml.

- To install C and C++ bindings, enter the following command:

```
# yum install libguestfs-devel
```

- To install Perl bindings:

```
# yum install 'perl(Sys::Guestfs)'
```

- To install Python bindings:

```
# yum install python-libguestfs
```

- To install Java bindings:

```
# yum install libguestfs-java libguestfs-java-devel libguestfs-javadoc
```

- To install Ruby bindings:

```
# yum install ruby-libguestfs
```

- To install OCaml bindings:

```
# yum install ocaml-libguestfs ocaml-libguestfs-devel
```

The binding for each language is essentially the same, but with minor syntactic changes. A C statement:

```
guestfs_launch (g);
```

Would appear like the following in Perl:

```
$g->launch ()
```

Or like the following in OCaml:

```
g#launch ()
```

Only the API from C is detailed in this section.

In the C and C++ bindings, you must manually check for errors. In the other bindings, errors are converted into exceptions; the additional error checks shown in the examples below are not necessary for other languages, but conversely you may wish to add code to catch exceptions. Refer to the following list for some points of interest regarding the architecture of the libguestfs API:

- The libguestfs API is synchronous. Each call blocks until it has completed. If you want to make calls asynchronously, you have to create a thread.
- The libguestfs API is not thread safe: each handle should be used only from a single thread, or if you want to share a handle between threads you should implement your own mutex to ensure that two threads cannot execute commands on one handle at the same time.
- You should not open multiple handles on the same disk image. It is permissible if all the handles are read-only, but still not recommended.
- You should not add a disk image for writing if anything else could be using that disk image (eg. a live VM). Doing this will cause disk corruption.
- Opening a read-only handle on a disk image which is currently in use (eg. by a live VM) is possible; however, the results may be unpredictable or inconsistent particularly if the disk image is being heavily written to at the time you are reading it.

17.11.1. Interaction with the API via a C program

Your C program should start by including the <guestfs.h> header file, and creating a handle:

```
#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);
 }
```

Save this program to a file (**`test.c`**). Compile this program and run it with the following two commands:

```
gcc -Wall test.c -o test -lguestfs
./test
```

At this stage it should print no output. The rest of this section demonstrates an example showing how to extend this program to create a new disk image, partition it, format it with an ext4 file system, and create some files in the file system. The disk image will be called **`disk.img`** and be created in the current directory.

The outline of the program is:

- Create the handle.
- Add disk(s) to the handle.
- Launch the libguestfs back end.
- Create the partition, file system and files.
- Close the handle and exit.

Here is the modified program:

```
#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);
}
```

Compile and run this program with the following two commands:

```
gcc -Wall test.c -o test -lguestfs
./test
```

If the program runs to completion successfully then you should be left with a disk image called **`disk.img`**, which you can examine with guestfish:

```
guestfish --ro -a disk.img -m /dev/sda1
><fs> ll /
><fs> cat /foo/resolv.conf
```

By default (for C and C++ bindings only), libguestfs prints errors to stderr. You can change this behavior by setting an error handler. The guestfs(3) man page discusses this in detail.

17.12. Troubleshooting

A test tool is available to check that libguestfs is working. Run the following command after installing libguestfs (root access not required) to test for normal operation:

```
$ libguestfs-test-tool
```

This tool prints a large amount of text to test the operation of libguestfs. If the test is successful, the following text will appear near the end of the output:

```
===== TEST FINISHED OK =====
```

17.13. Where to find further documentation

The primary source for documentation for libguestfs and the tools are the Unix man pages. The API is documented in guestfs(3). guestfish is documented in guestfish(1). The virt tools are documented in their own man pages (eg. virt-df(1)).

Virtual Networking

This chapter introduces the concepts needed to create, start, stop, remove and modify virtual networks with libvirt.

18.1. Virtual network switches

Libvirt virtual networking uses the concept of a *virtual network switch*. A virtual network switch is a software construct that operates on a host server, to which virtual machines (guests) connect. The network traffic for a guest is directed through this switch:

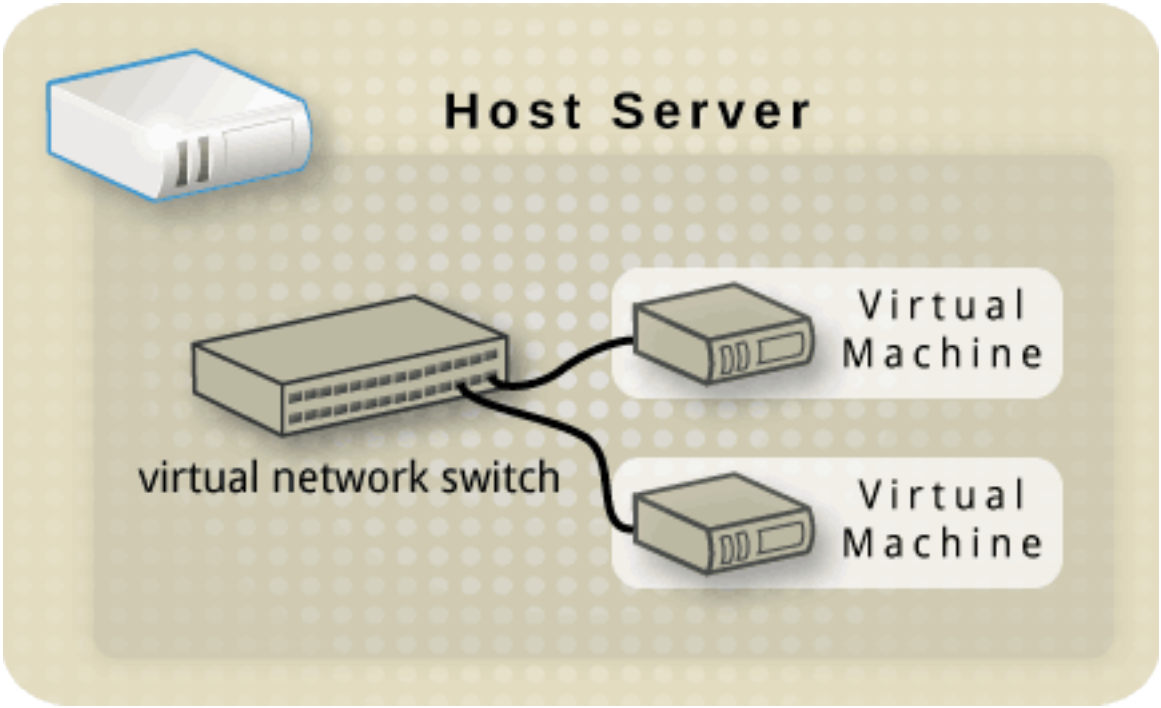


Figure 18.1. Virtual network switch with two guests

Linux host servers represent a virtual network switch as a network interface. When the `libvirt` daemon is first installed and started, the default network interface representing the virtual network switch is **`virbr0`**.

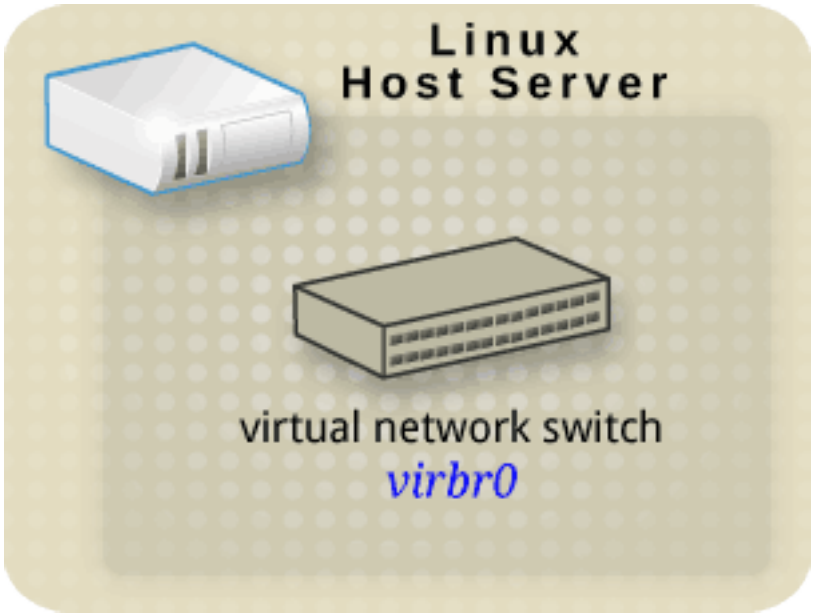


Figure 18.2. Linux host with an interface to a virtual network switch

This **virbr0** interface can be viewed with the **ifconfig** and **ip** commands like any other interface:

```
$ ifconfig virbr0
 virbr0    Link encap:Ethernet  HWaddr 1B:C4:94:CF:FD:17
           inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
           UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
           RX bytes:0 (0.0 b)  TX bytes:3097 (3.0 KiB)
```

```
 $ 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
```

18.1.1. Network Address Translation

By default, virtual network switches operate in NAT mode. They use IP masquerading rather than SNAT (Source-NAT) or DNAT (Destination-NAT). IP masquerading enables connected guest to use the host IP address for communication to any external network. By default, computers that are placed externally to the host can not communicate to the guests inside when the virtual network switch is operating in NAT mode, as shown in the following diagram:

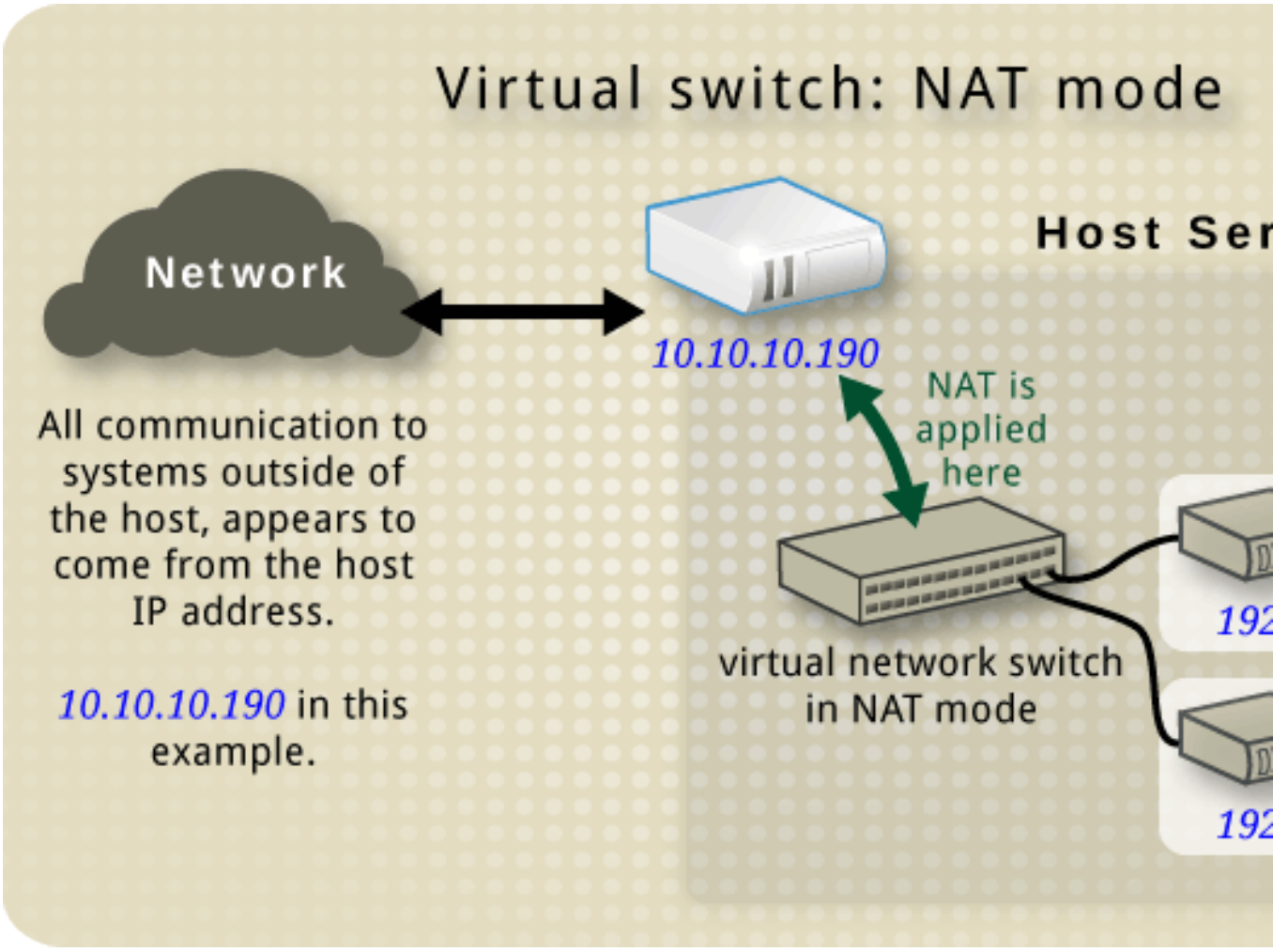


Figure 18.3. Virtual network switch using NAT with two guests

Warning

Virtual network switches use NAT configured by iptables rules. Editing these rules while the switch is running is not recommended, as incorrect rules may result in the switch being unable to communicate.

18.2. DNS and DHCP

IP information can be assigned to guests via DHCP. A pool of addresses can be assigned to a virtual network switch for this purpose. Libvirt uses the **dnsmasq** program for this. An instance of dnsmasq is automatically configured and started by libvirt for each virtual network switch that needs it.

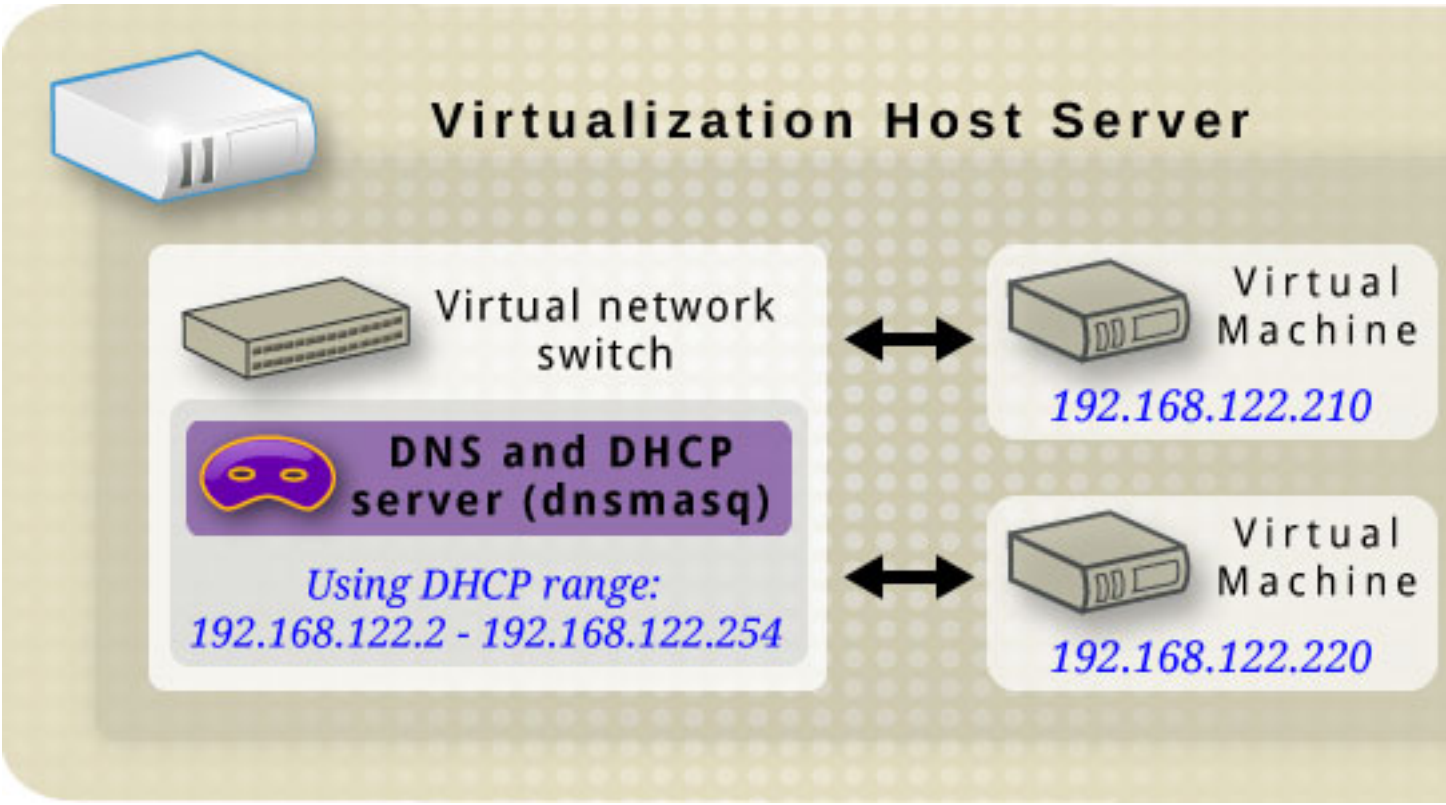


Figure 18.4. Virtual network switch running dnsmasq

18.3. Other virtual network switch routing types

Virtual network switches can operate in two other modes besides NAT, *Routed mode* and *Isolated mode*:

Routed mode

When using *routed mode*, the virtual switch connects to the physical LAN connected to the host, passing traffic back and forth without the use of NAT. The virtual switch can examine all traffic and use the information contained within the network packets to make routing decisions. When using this mode, all of the virtual machines are in their own subnet, routed through a virtual switch. This situation is not always ideal as no other hosts on the physical network are aware of the virtual machines without manual physical router configuration, and can not access the virtual machines. Routed mode operates at Layer 3 of the ISO networking model.

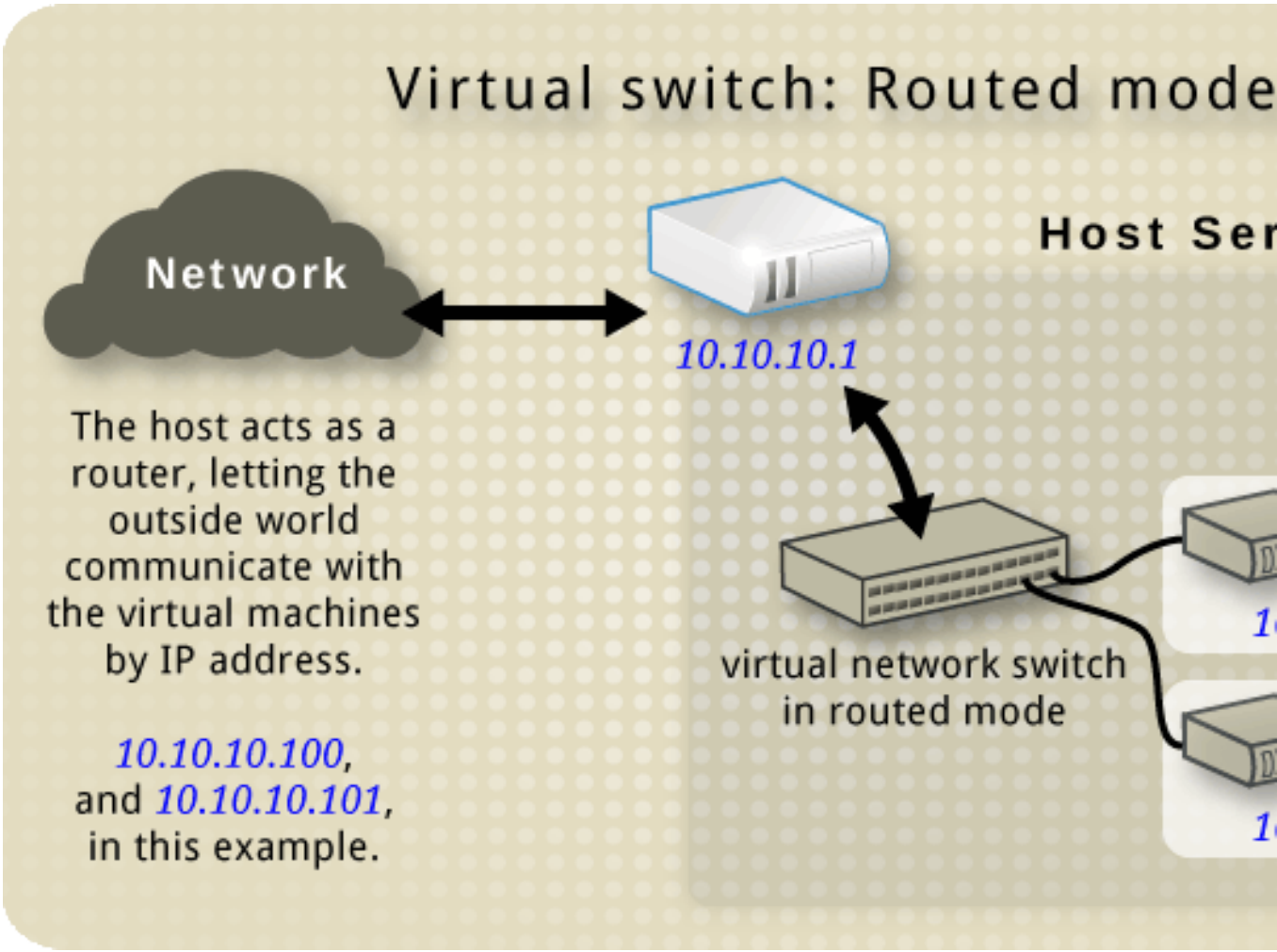


Figure 18.5. Virtual network switch in routed mode

Isolated mode

When using *Isolated mode*, guests connected to the virtual switch can communicate with each other, and with the host, but their traffic will not pass outside of the host, nor can they receive traffic from outside the host. Using dnsmasq in this mode is required for basic functionality such as DHCP. However, even if this network is isolated from any physical network, DNS names are still resolved. Therefore a situation can arise when DNS names resolve but ICMP echo request (ping) commands fail.

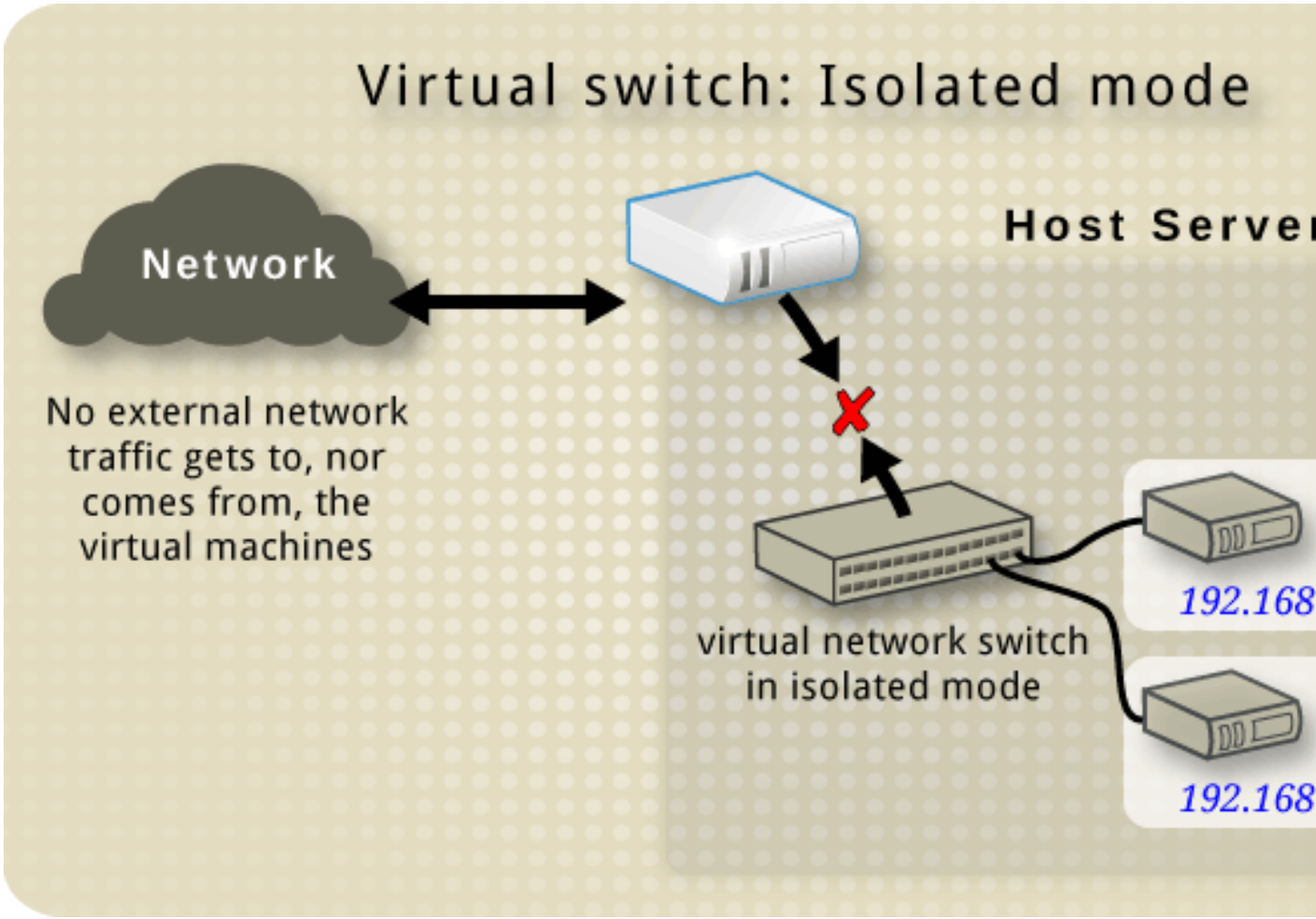


Figure 18.6. Virtual network switch in isolated mode

18.4. The default configuration

When the `libvirtd` daemon is first installed, it contains an initial virtual network switch configuration in NAT mode. This configuration is used so that installed guests can communicate to the external network, through the host. The following image demonstrates this default configuration for `libvirtd`:

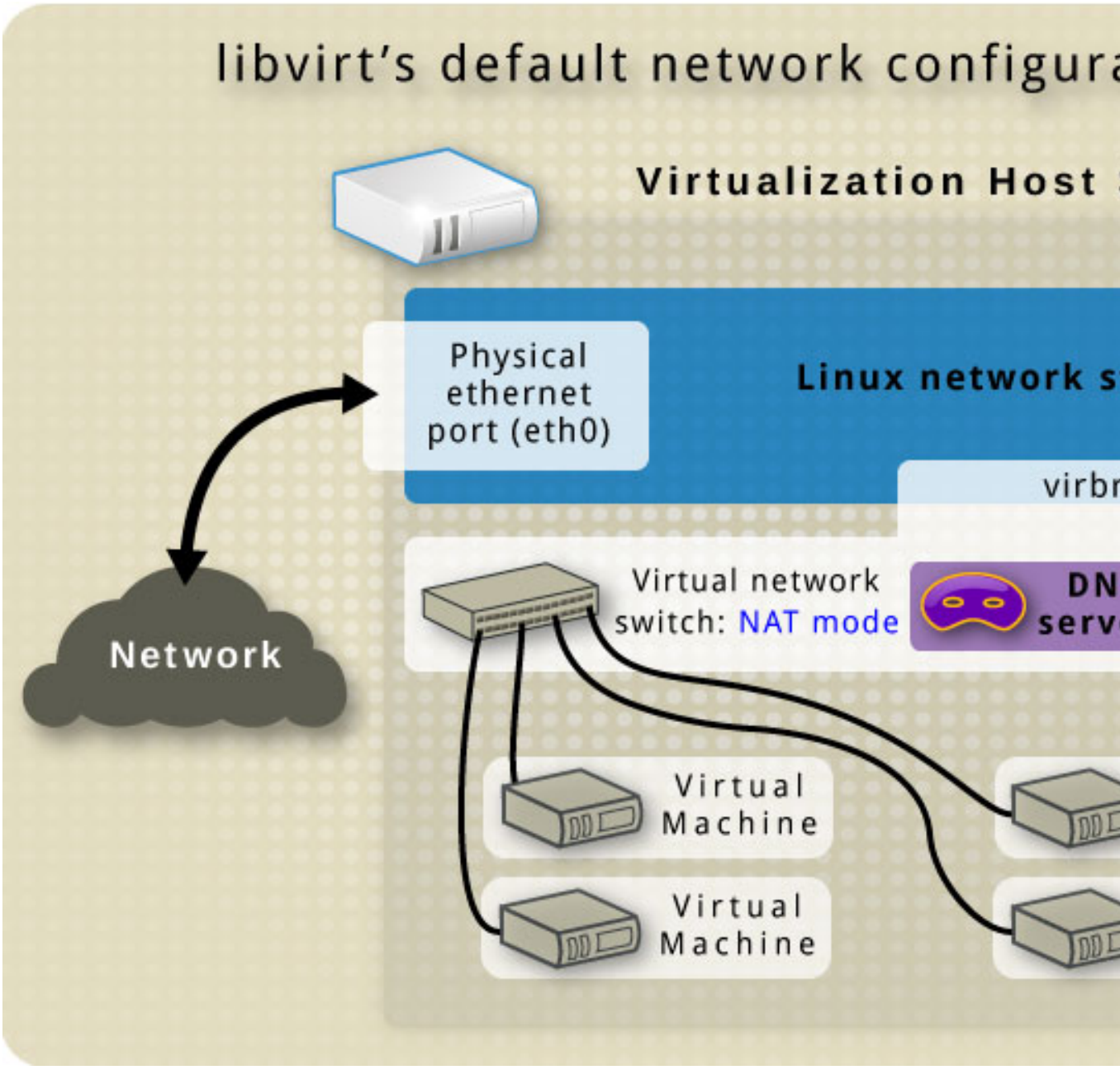


Figure 18.7. Default libvirt network configuration

Restricting virtual network traffic by interface

A virtual network can be restricted to a specific physical interface. This may be useful on a physical system that has several interfaces (for example, **`eth0`**, **`eth1`** and **`eth2`**). This is only useful in routed and NAT modes, and can be defined in the **`dev=<interface>`** option, or in **`virt-manager`** when creating a new virtual network.

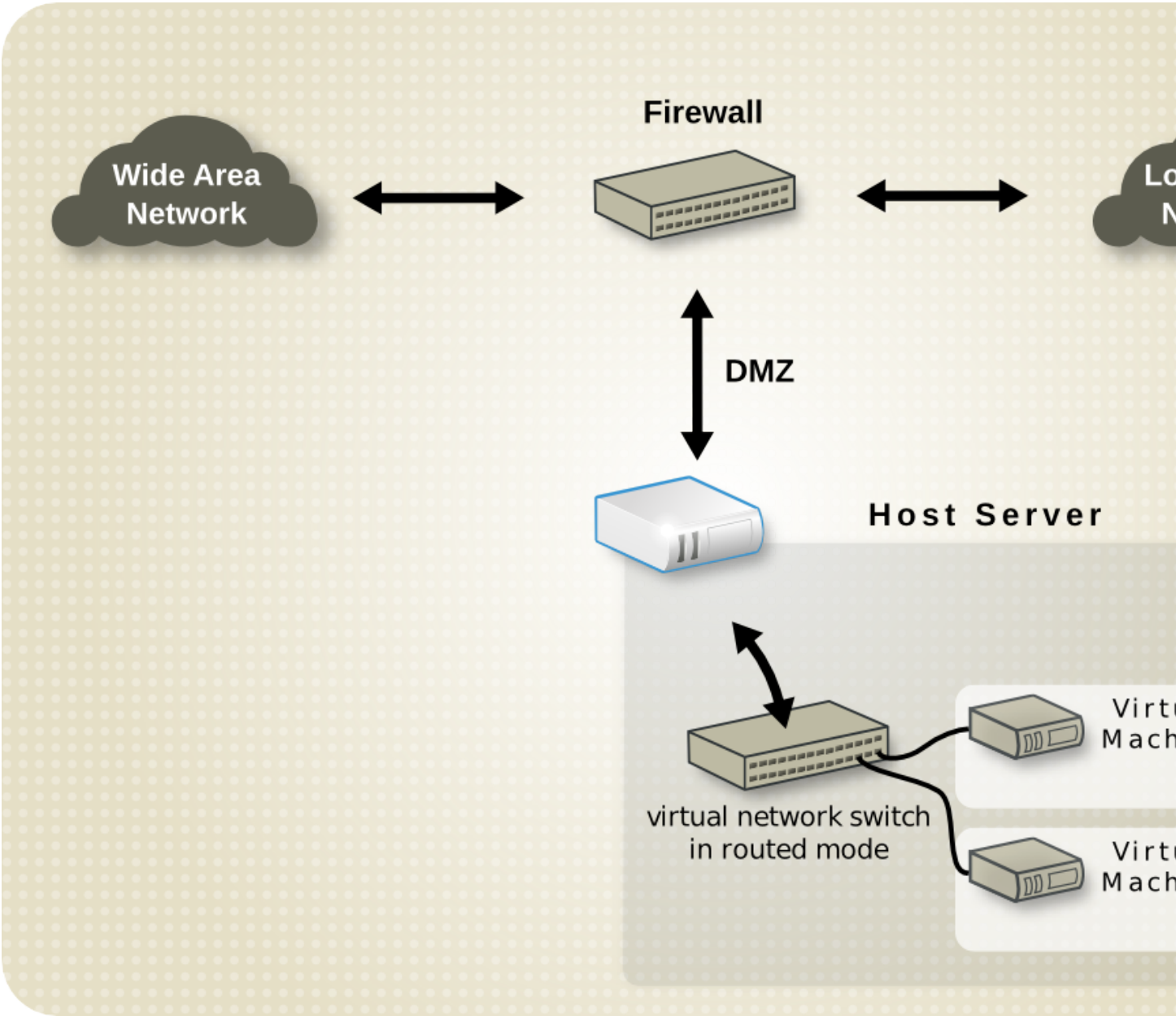
18.5. Examples of common scenarios

This section demonstrates different virtual networking modes and provides some example scenarios.

18.5.1. Routed mode

DMZ

Consider a network where one or more nodes are placed in a controlled subnetwork for security reasons. The deployment of a special subnetwork such as this is a common practice, and the subnetwork is known as a DMZ. Refer to the following diagram for more details on this layout:

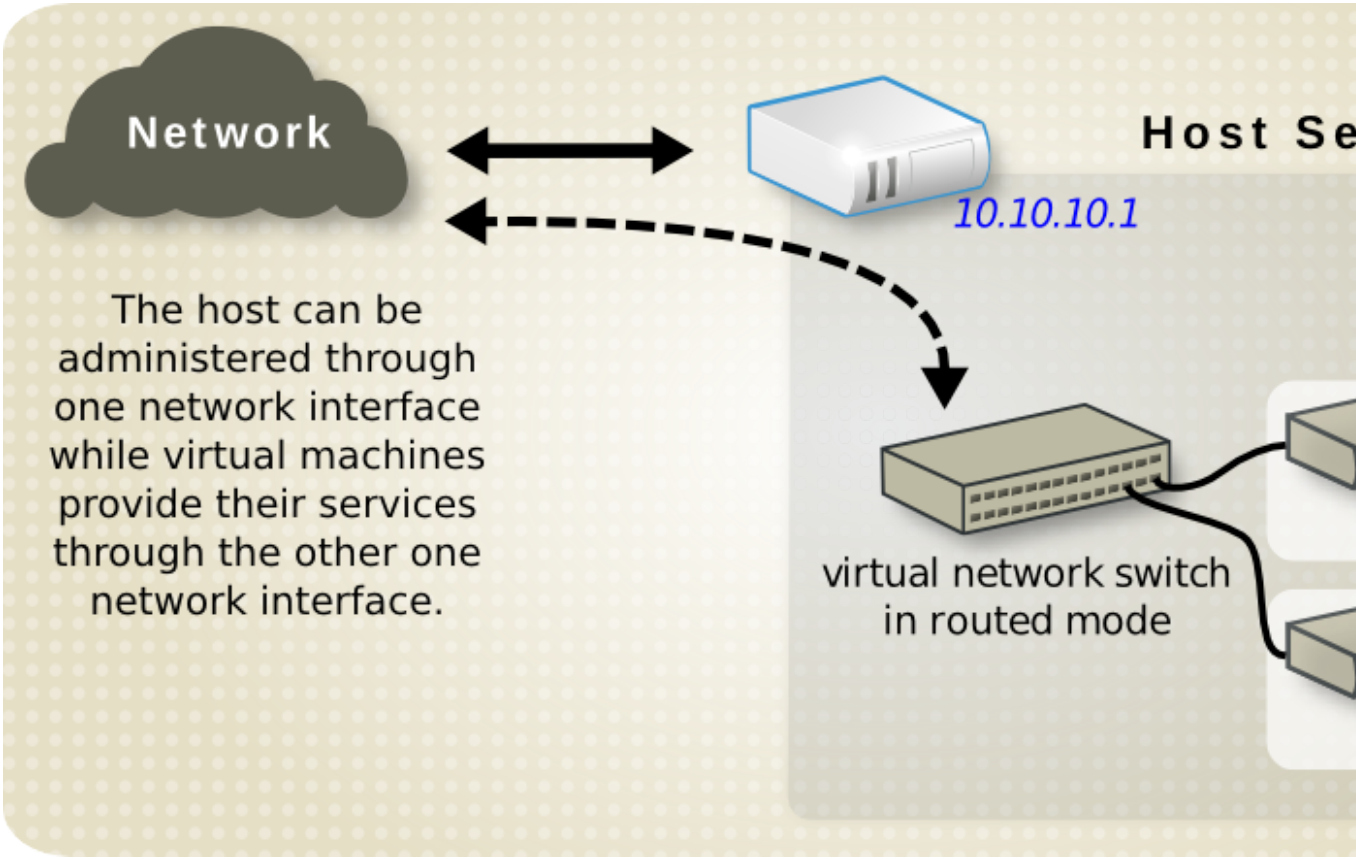


Hosts in a DMZ typically provide services to WAN (external) hosts as well as LAN (internal) hosts. As this requires them to be accessible from multiple locations, and considering that these locations are controlled and operated in different ways based on their security and trust level, routed mode is the best configuration for this environment.

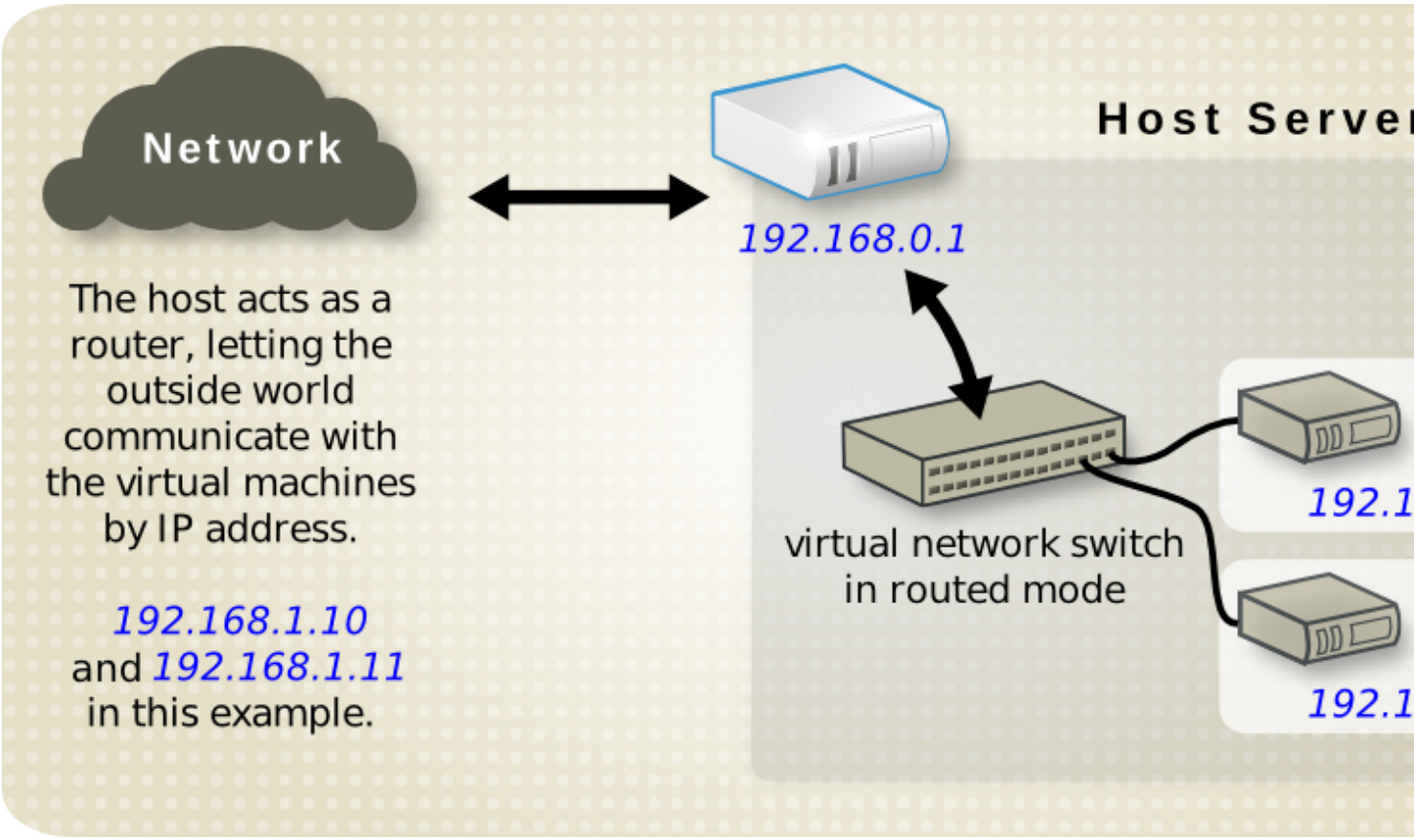
Virtual Server hosting

Consider a virtual server hosting company that has several hosts, each with two physical network connections. One interface is used for management and accounting, the other is for the virtual machines to connect through. Each guest has its own public IP address, but the hosts use private IP

address as management of the guests can only be performed by internal administrators. Refer to the following diagram to understand this scenario:



When the host has a public IP address and the virtual machines have static public IPs, bridged networking can not be used, as the provider only accepts packets from the MAC adress of the public host. The following diagram demonstrates this:



18.5.2. NAT mode

NAT (Network Address Translation) mode is the default mode. It can be used for testing when there is no need for direct network visibility.

18.5.3. Isolated mode

Isolated mode allows virtual machines to communicate with each other only. They are unable to interact with the physical network.

18.6. Managing a virtual network

To configure a virtual network on your system:

1. From the **Edit** menu, select **Connection Details**.

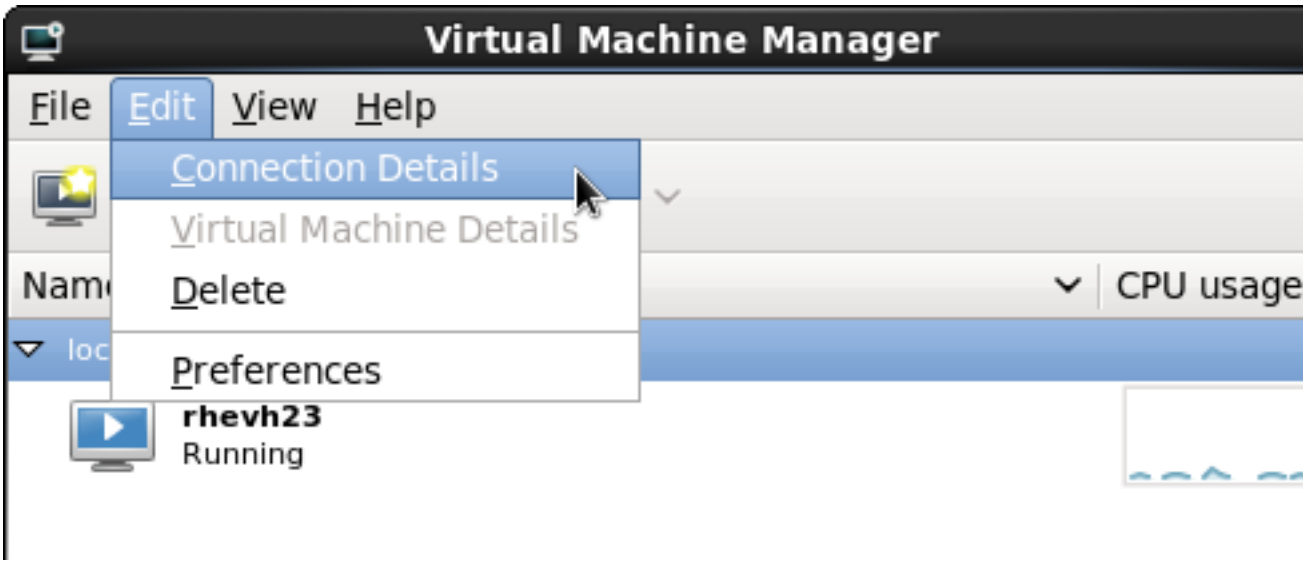
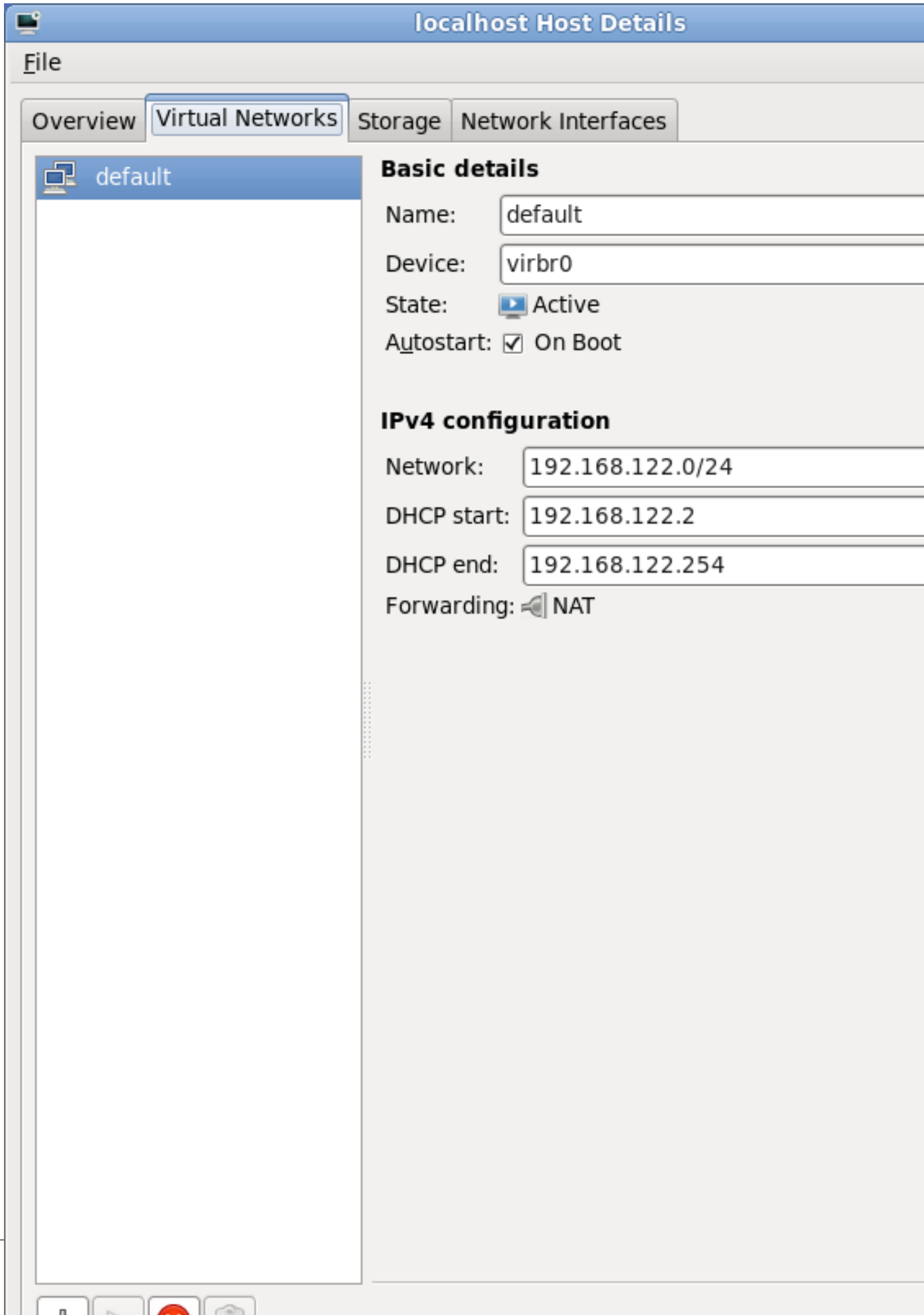


Figure 18.8. Selecting a host's details

2. This will open the **Host Details** menu. Click the **Virtual Networks** tab.

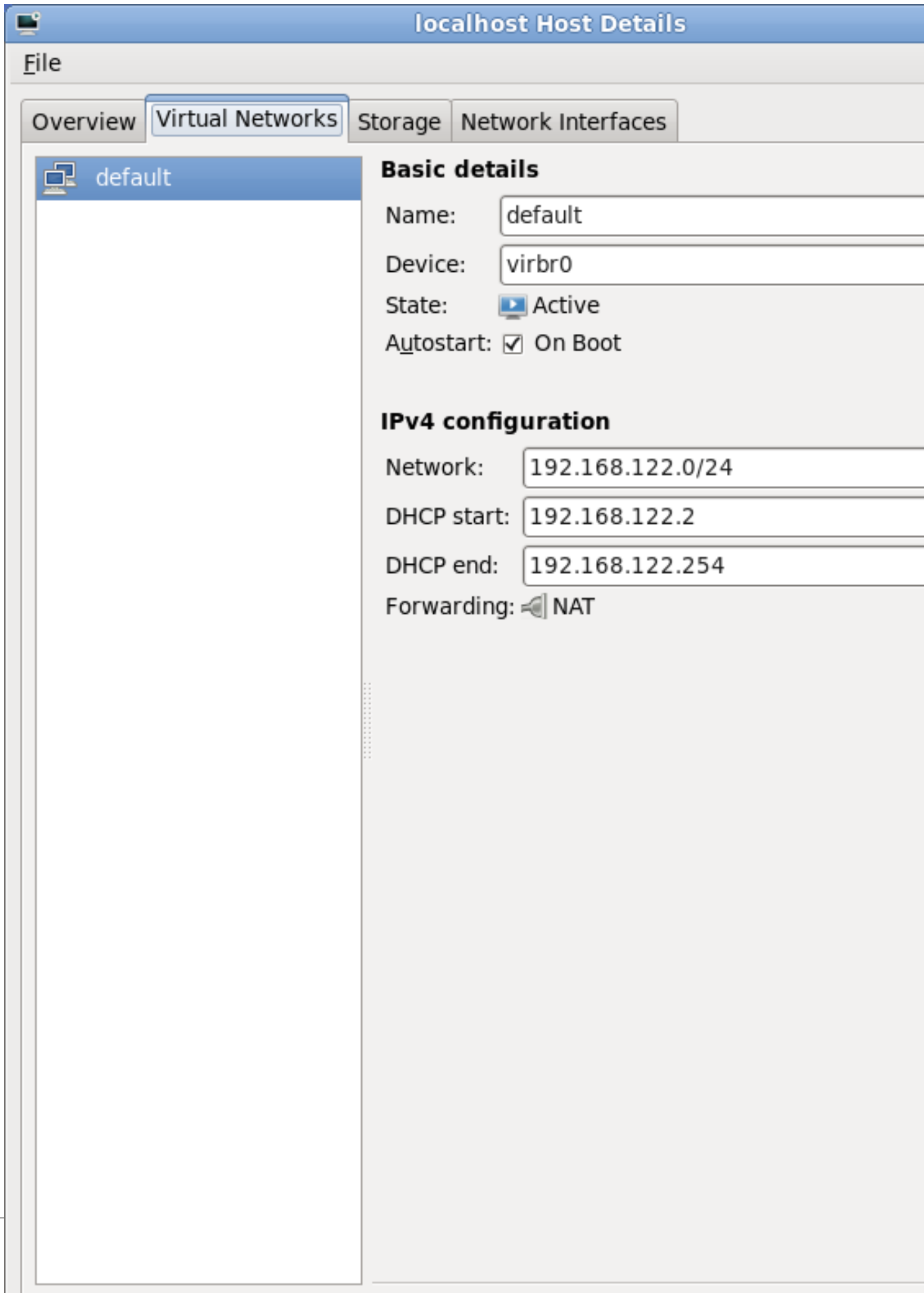


3. All available virtual networks are listed on the left-hand box of the menu. You can edit the configuration of a virtual network by selecting it from this box and editing as you see fit.

18.7. Creating a virtual network

To create a virtual network on your system:

1. Open the **Host Details** menu (refer to *Section 18.6, "Managing a virtual network"*) and click the **Add Network** button, identified by a plus sign (+) icon.



This will open the **Create a new virtual network** window. Click **Forward** to continue.

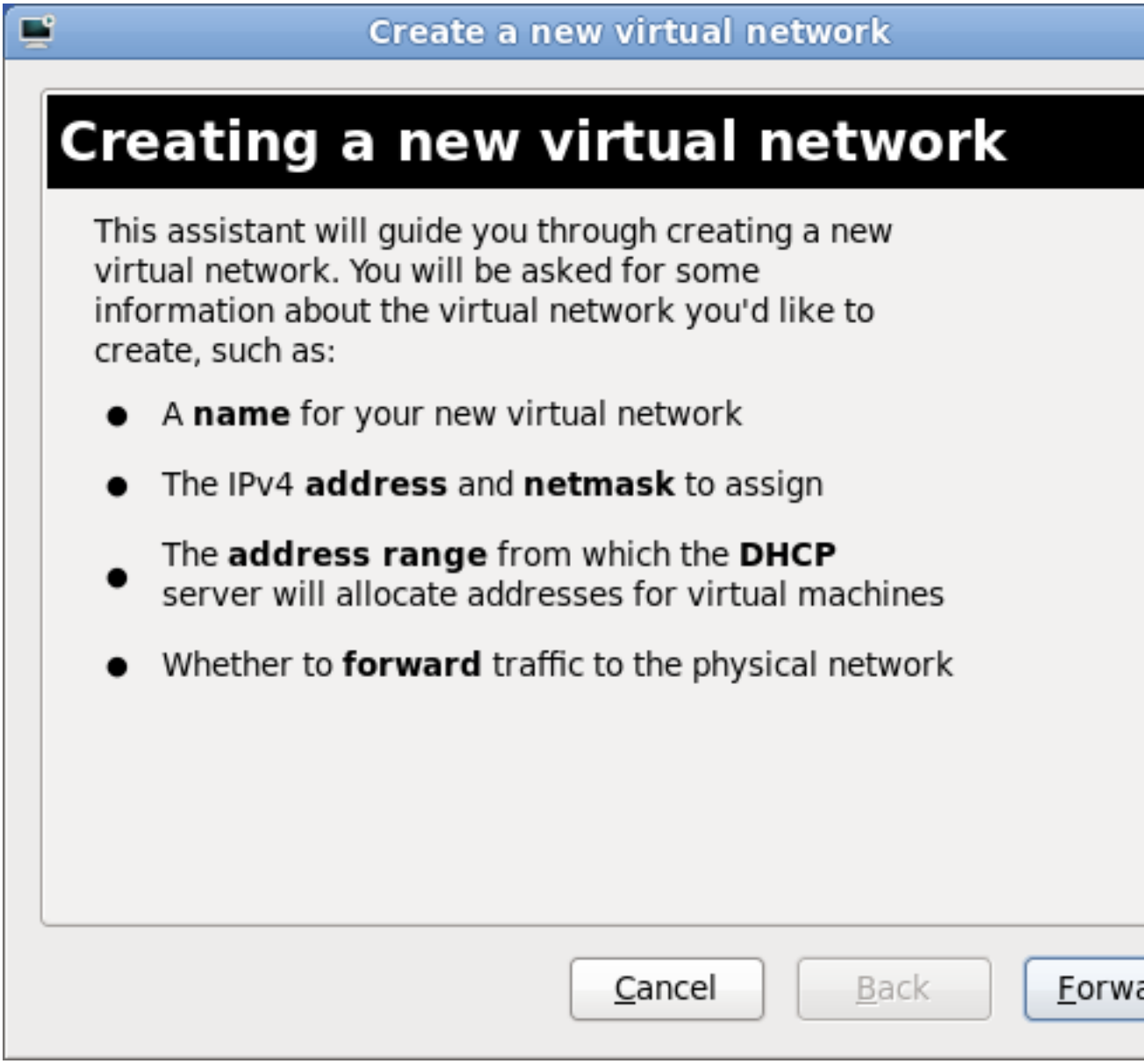


Figure 18.11. Creating a new virtual network

2. Enter an appropriate name for your virtual network and click **Forward**.

Figure 18.12. Naming your virtual network

3. Enter an IPv4 address space for your virtual network and click **Forward**.

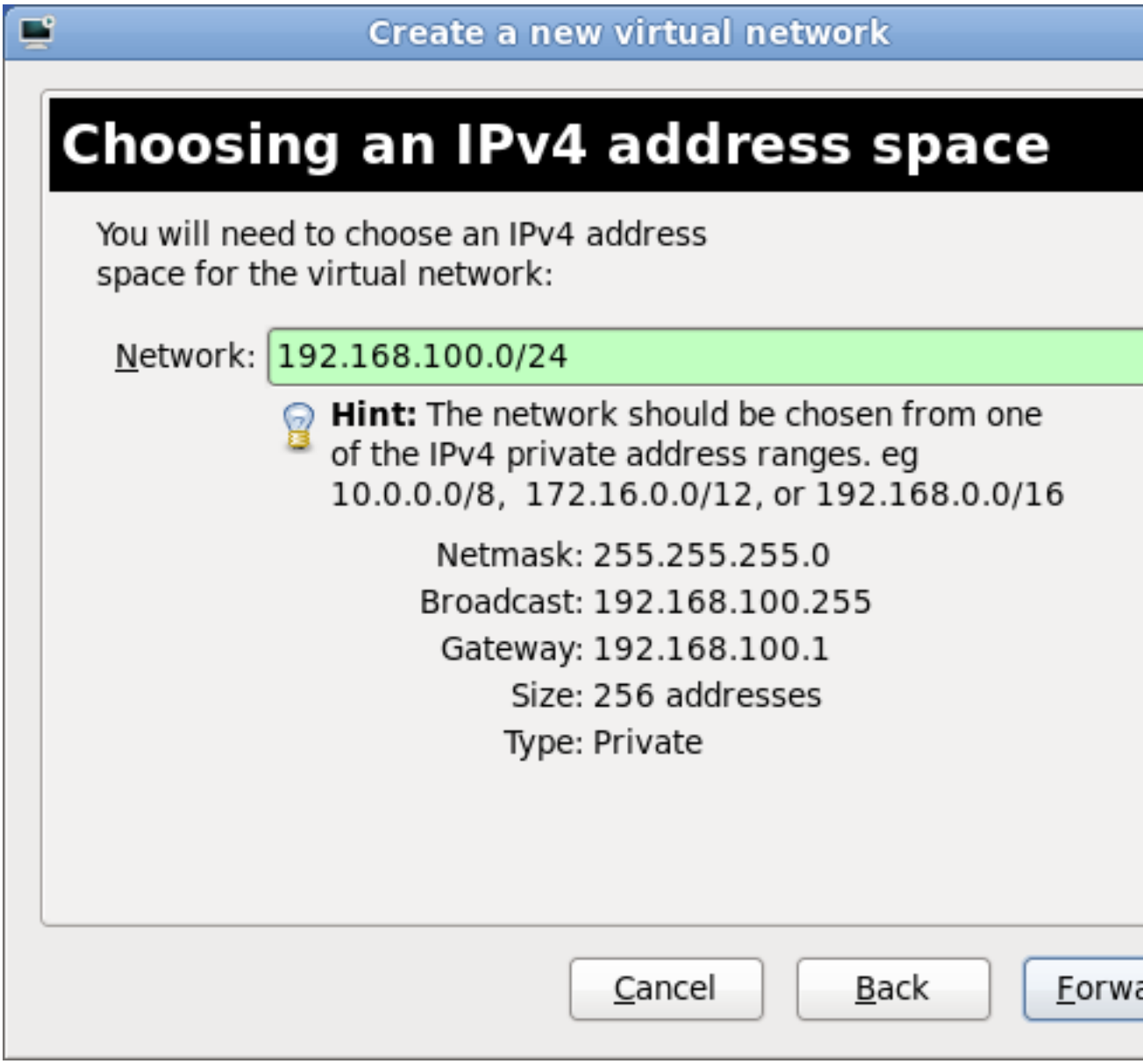


Figure 18.13. Choosing an IPv4 address space

4. Define the DHCP range for your virtual network by specifying a **Start** and **End** range of IP addresses. Click **Forward** to continue.

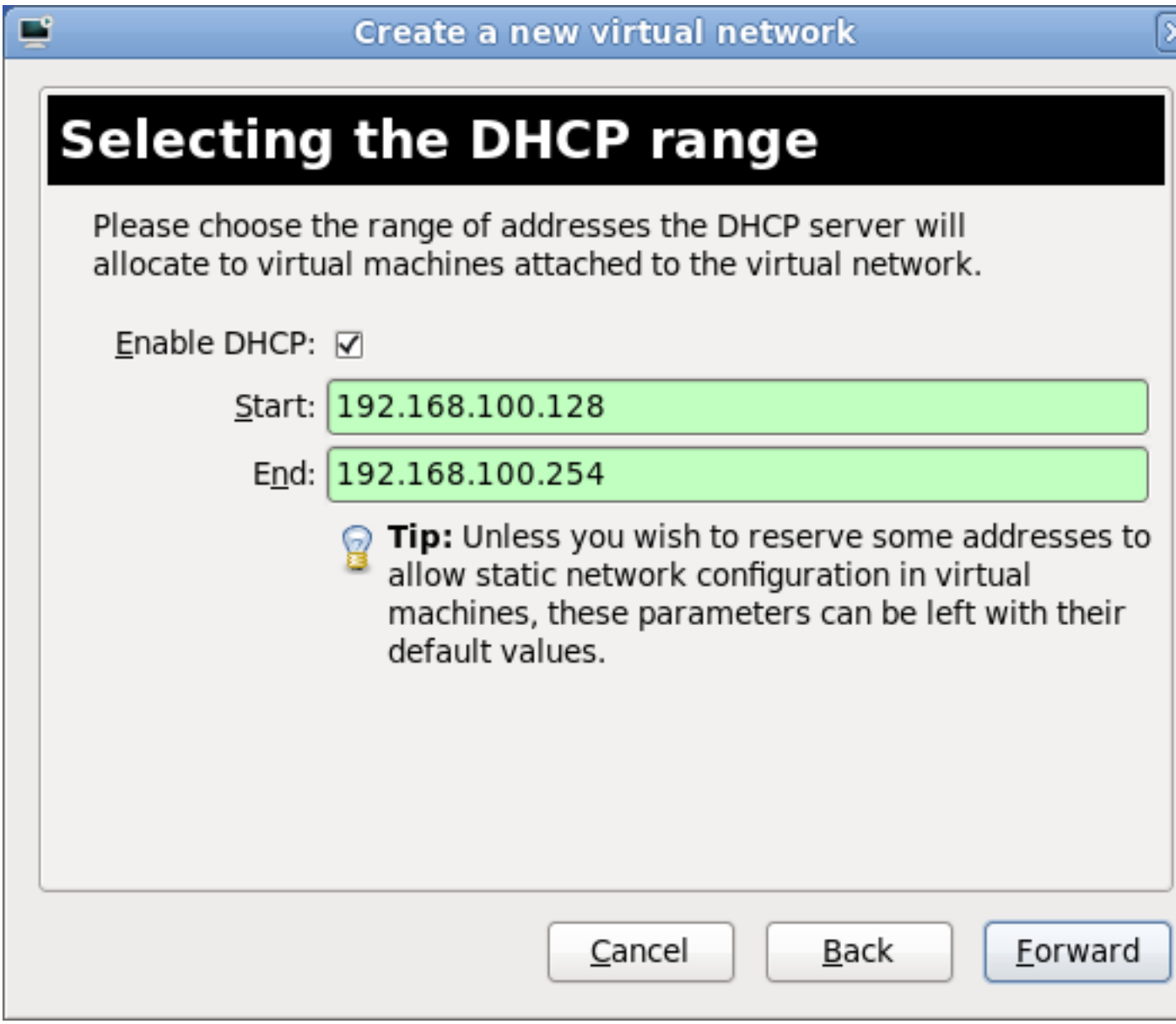


Figure 18.14. Selecting the DHCP range

5. Select how the virtual network should connect to the physical network.

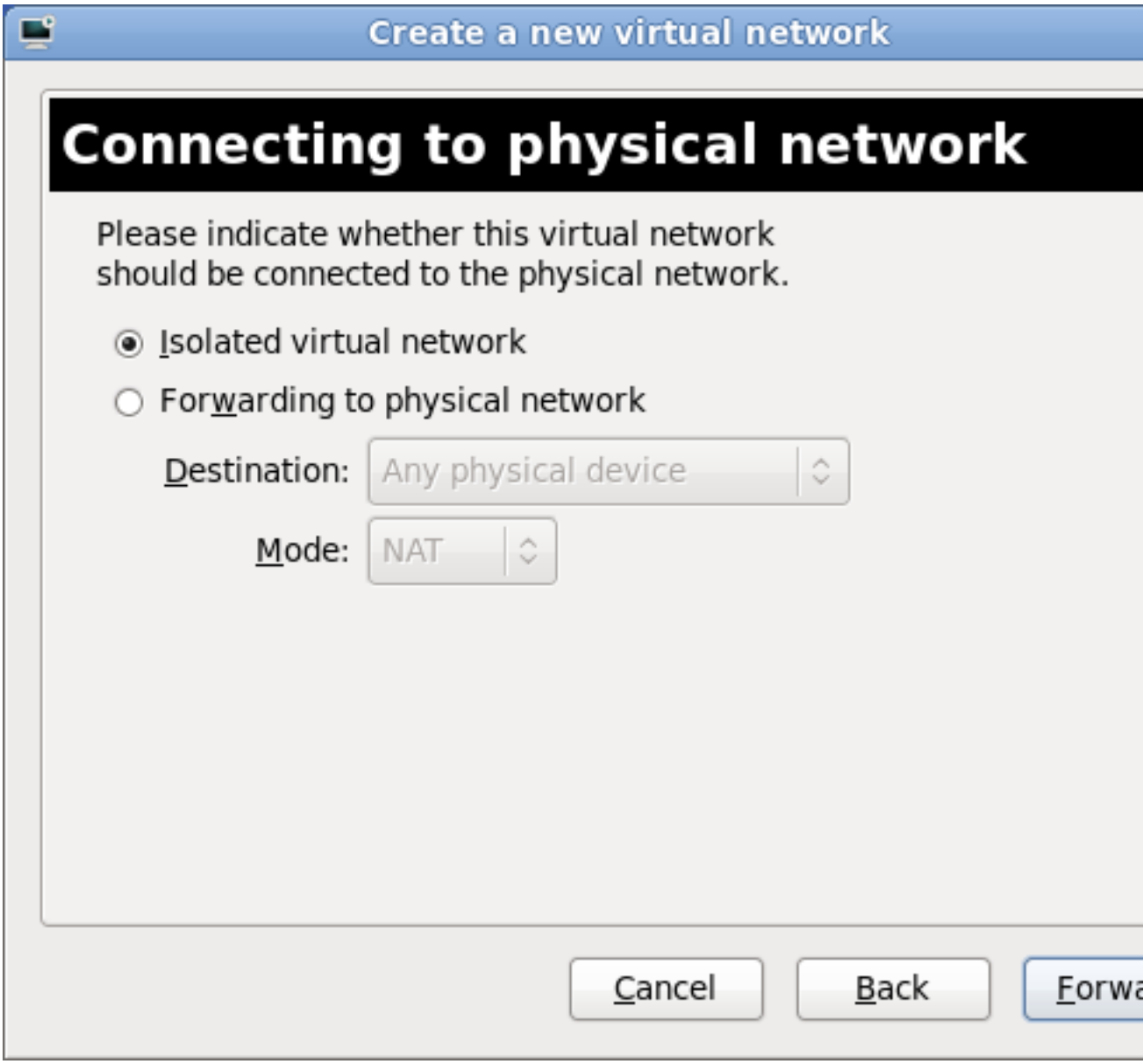


Figure 18.15. Connecting to physical network

If you select **Forwarding to physical network**, choose whether the **Destination** should be **Any physical device** or a specific physical device. Also select whether the **Mode** should be **NAT** or **Routed**.

Click **Forward** to continue.

6. You are now ready to create the network. Check the configuration of your network and click **Finish**.

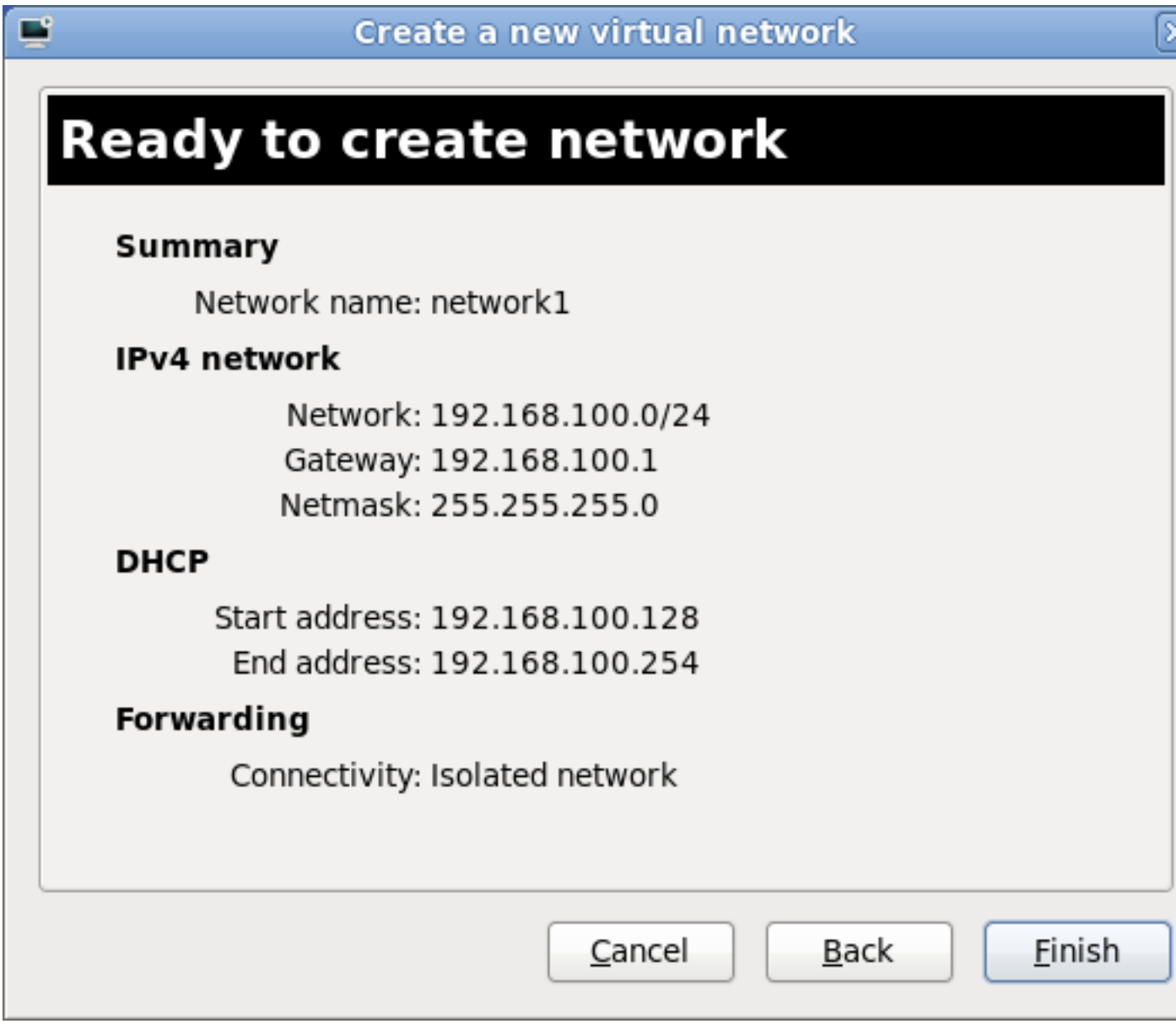
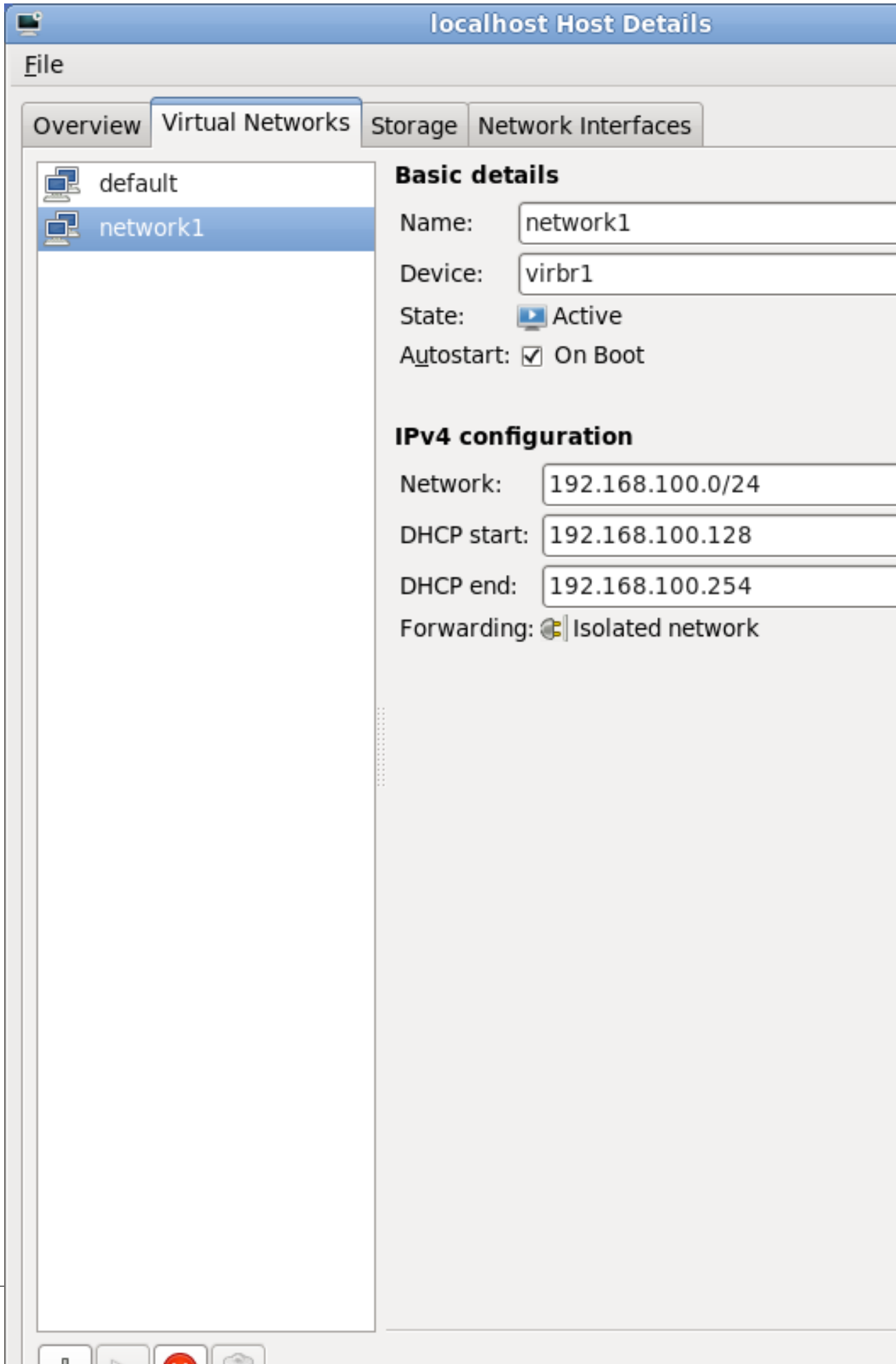


Figure 18.16. Ready to create network

7. The new virtual network is now available in the **Virtual Networks** tab of the **Host Details** window.



18.8. Attaching virtual network to host

To attach a virtual network to a guest:

1. In the **Virtual Machine Manager** window, highlight the guest that will have the network assigned.

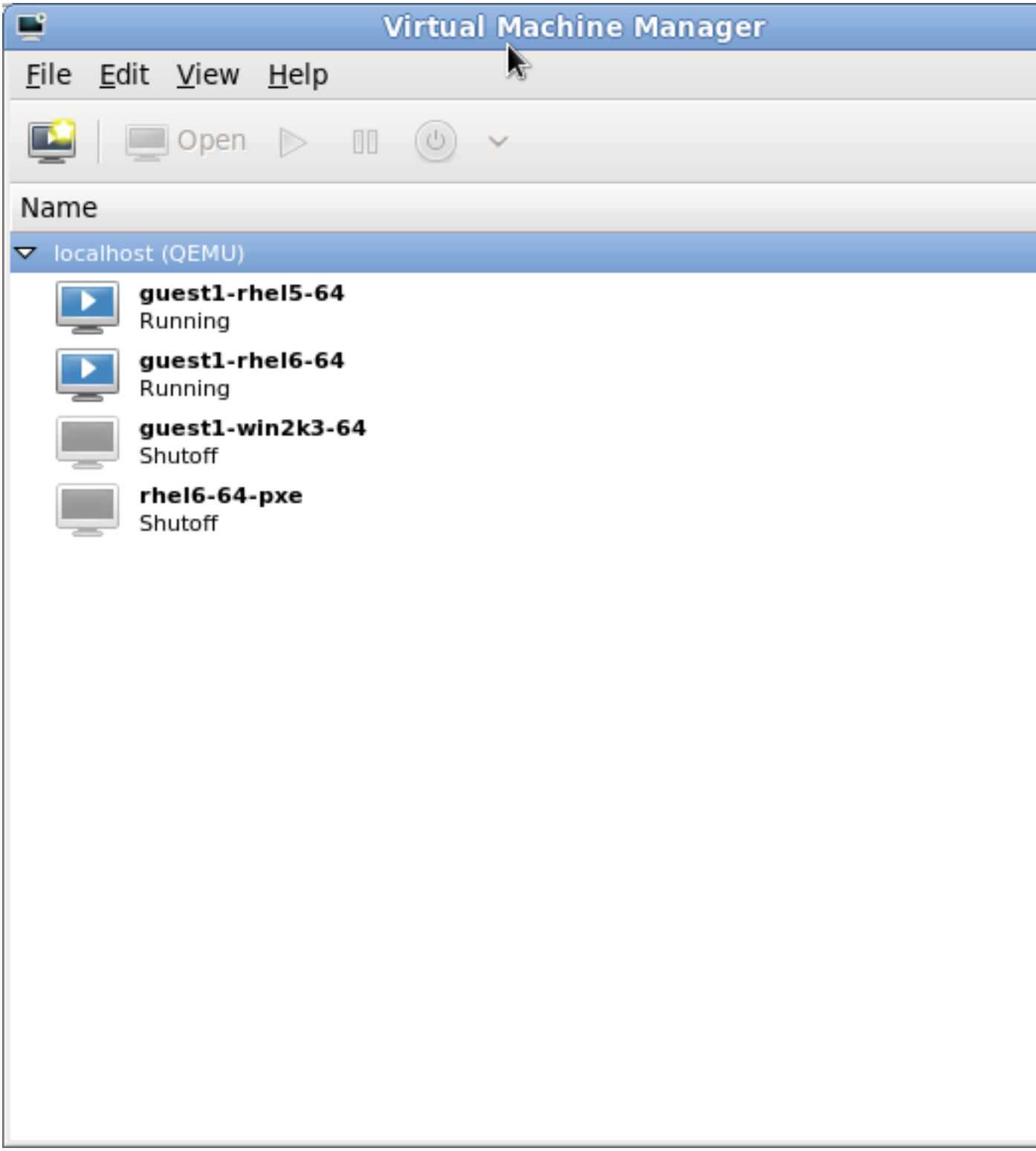


Figure 18.18. Selecting a virtual machine to display

2. From the Virtual Machine Manager **Edit** menu, select **Virtual Machine Details**.

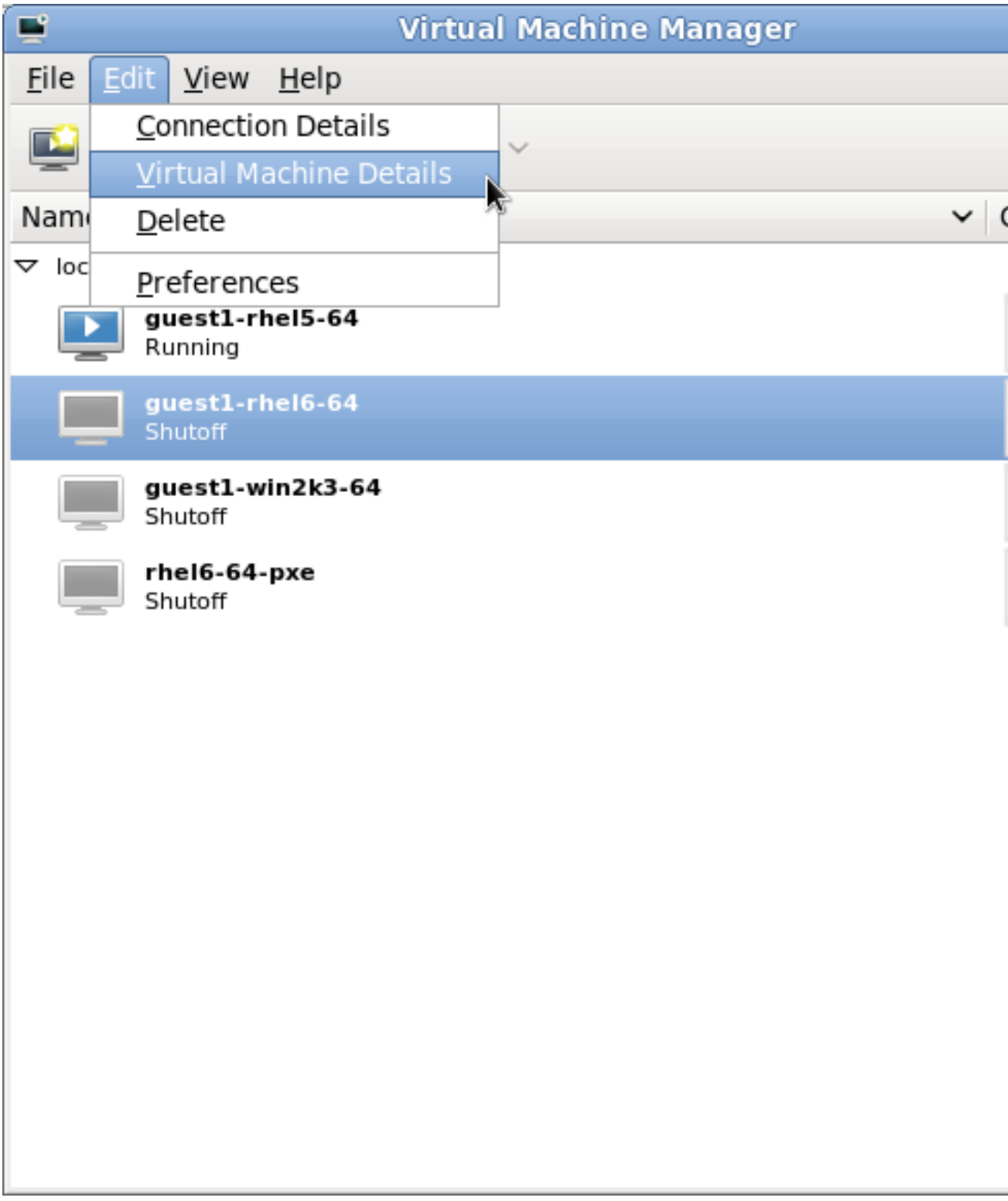


Figure 18.19. Displaying the virtual machine details

3. Click the **Add Hardware** button on the Virtual Machine Details window.

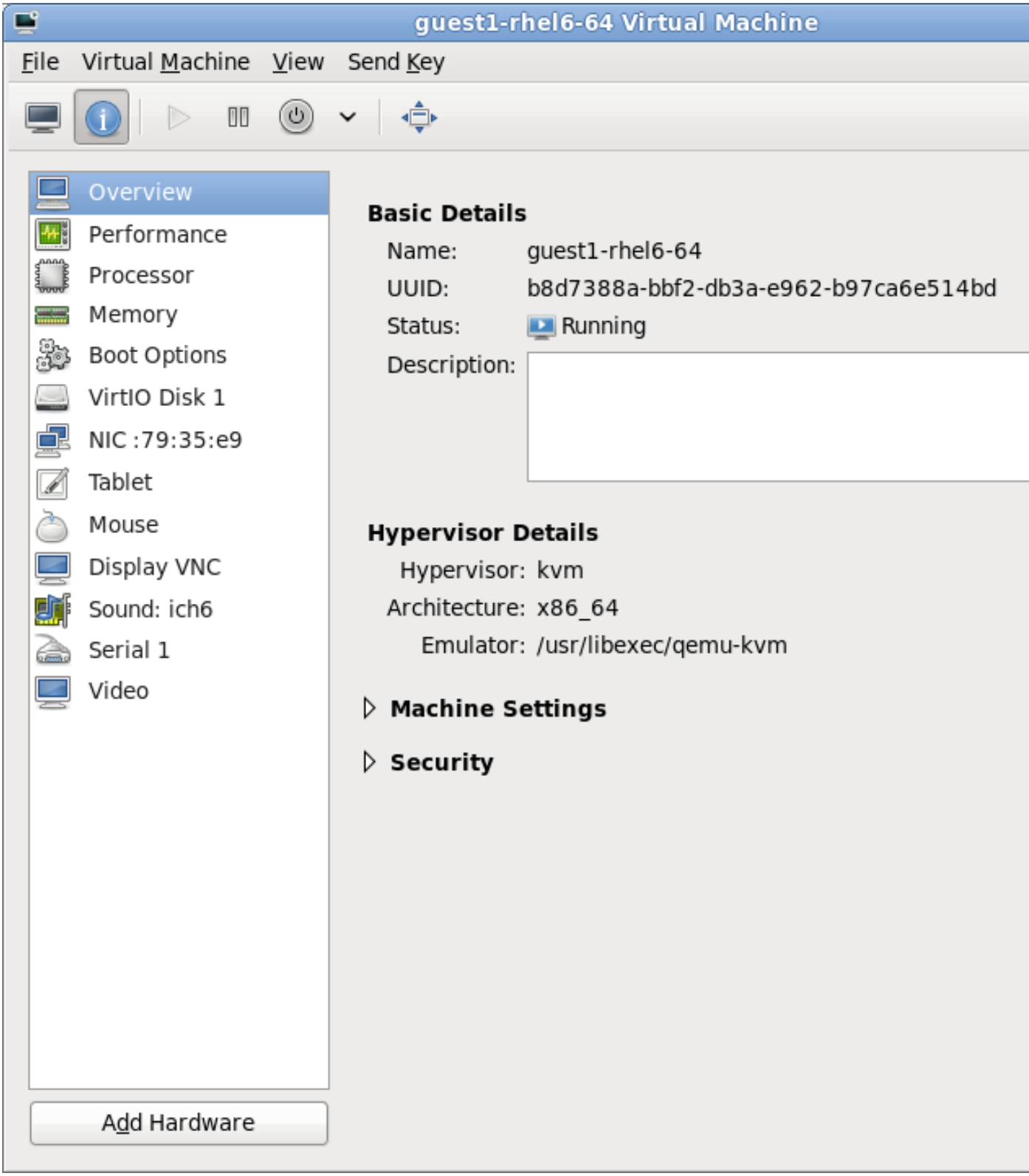


Figure 18.20. The Virtual Machine Details window

4. In the **Add new virtual hardware** window, select **Network** from the left pane, and select your network name (*network1* in this example) from the **Host device** menu and click **Finish**.

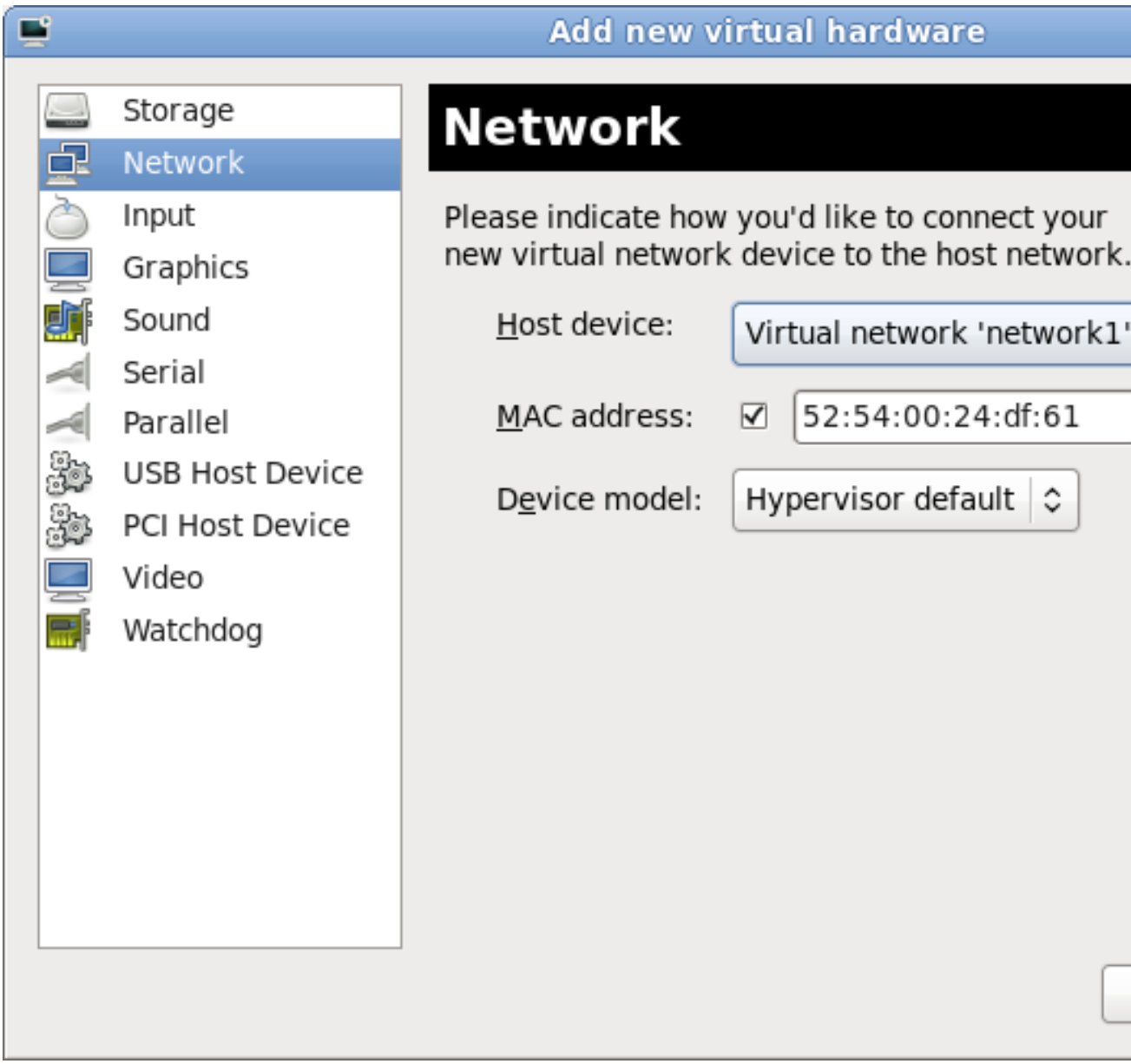


Figure 18.21. Select your network from the Add new virtual hardware window

5. The new network is now displayed as a virtual network interface that will be presented to the guest upon launch.

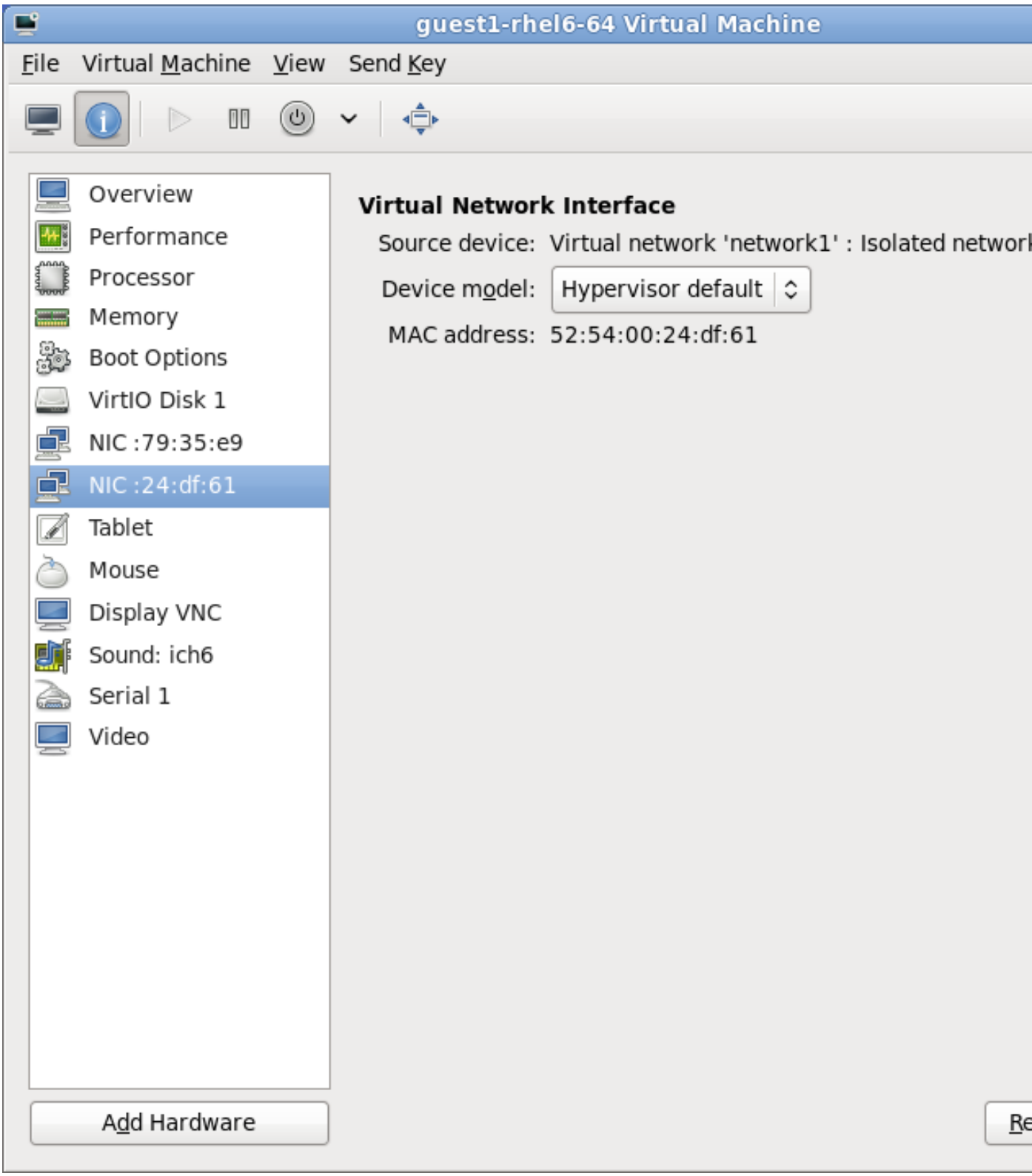


Figure 18.22. New network shown in guest hardware list

libvirt configuration reference

This chapter provides is a references for various parameters of libvirt XML configuration files

Table 19.1. libvirt configuration files

Item	Description
pae	Specifies the physical address extension configuration data.
apic	Specifies the advanced programmable interrupt controller configuration data.
memory	Specifies the memory size in megabytes.
vcpus	Specifies the numbers of virtual CPUs.
console	Specifies the port numbers to export the domain consoles to.
nic	Specifies the number of virtual network interfaces.
vif	Lists the randomly-assigned MAC addresses and bridges assigned to use for the domain's network addresses.
disk	Lists the block devices to export to the domain and exports physical devices to domain with read only access.
dhcp	Enables networking using DHCP.
netmask	Specifies the configured IP netmasks.
gateway	Specifies the configured IP gateways.
acpi	Specifies the advanced configuration power interface configuration data.

Creating custom libvirt scripts

This section provides some information which may be useful to programmers and system administrators intending to write custom scripts to make their lives easier by using **libvirt**.

Chapter 9, Miscellaneous administration tasks is recommended reading for programmers thinking of writing new applications which use **libvirt**.

20.1. Using XML configuration files with virsh

virsh can handle XML configuration files. You may want to use this to your advantage for scripting large deployments with special options. You can add devices defined in an XML file to a running guest. For example, to add an ISO file as **hdc** to a running guest create an XML file:

```
# cat satelliteiso.xml
<disk type="file" device="disk">
 <driver name="file"/>
 <source file="/var/lib/libvirt/images/rhn-satellite-5.0.1-11-redhat-linux-as-i386-4-
embedded-oracle.iso"/>
 <target dev="hdc"/>
 <readonly/>
</disk>
```

Run **virsh attach-device** to attach the ISO as **hdc** to a guest called "satellite":

```
# virsh attach-device satellite satelliteiso.xml
```

qemu-kvm Whitelist

21.1. Introduction

Product identification

Red Hat Enterprise Linux 6

Objectives

The primary objective of this whitelist is to provide a complete list of the supported options of the **qemu-kvm** utility used as an emulator and a virtualizer in Red Hat Enterprise Linux 6. This is a comprehensive summary of the supported options.

Background

Red Hat Enterprise Linux 6 uses KVM as an underlying virtualization technology. The machine emulator and virtualizer used is a modified version of QEMU called qemu-kvm. This version does not support all configuration options of the original QEMU and it adds some additional options.

Scope of the chapter

This chapter lists only the supported options of the qemu-kvm utility. Options not listed here are not supported by Red Hat.

Used format

- **<name>** - When used in a syntax description, this string shoud be replaced by user-defined value.
- **[a|b|c]** - When used in a syntax description, only one of the strings separated by | is used.
- When no comment is present, an option is supported with all possible values.

21.2. Basic options

Emulated machine

-M <machine-id>

Processor type

-cpu <model>[,<FEATURE>][...]

We support exposing additional features and placing restrictions. Supported models are:

- **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)
- **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 supported QEMU Virtual CPU version
- **cpu64-rhel6** - Red Hat Enterprise Linux 6 supported QEMU Virtual CPU version
- **default** - special option use default option from above.

Processor Topology

-smp <n>[,cores=<ncores>][,threads=<nthreads>][,sockets=<nsocks>][,maxcpus=<maxcpus>]

Hypervisor and guest operating system limits on processor topology apply.

NUMA system

-numa <nodes>[,mem=<size>][,cpus=<cpu[-cpu>]][,nodeid=<node>]

Hypervisor and guest operating system limits on processor topology apply.

Memory size

-m <megs>

Supported values are limited by guest minimal and maximal values and hypervisor limits.

Keyboard layout

-k <language>

Guest name

-name <name>

Guest UUID

-uuid <uuid>

21.3. Disk options

Generic drive

-drive <option>[,<option>[,<option>[,...]]]

Supported with the following options:

- **id**=<id>

  Id of the drive has the following limitaton for if=none:

  - IDE disk has to have <id> in following format: drive-ide0-<BUS>-<UNIT>

    Example of correct format:

    -drive if=none,id=drive-ide0-<BUS>-<UNIT>,... -device ide-drive,drive=drive-ide0-<BUS>-<UNIT>,bus=ide.<BUS>,unit=<UNIT>

- **file**=<file>

  Value of <file> is parsed with the following rules:

  - Passing floppy device as <file> is not supported.
  - Passing cd-rom device as <file> is supported only with cdrom media type (media=cdrom) and only as IDE drive (either if=ide or if=none + -device ide-drive).
  - If <file> is neither block nor character device, it must not contain ':'.

- **if**=<interface>

  The following interfaces are supported: none, ide, virtio, floppy.

- **index**=<index>
- **media**=<media>
- **cache**=<cache>

  Supported values: none, writeback or writethrough.

- **snapshot**=[yes|no]
- **serial**=<serial>
- **aio**=<aio>
- **format**=<format>

  This option is not required and can be omitted. However, this is not recommended for raw images because it represents security risk. Supported formats are:

  - **qcow2**
  - **raw**

Boot option

-boot [order=<drives>][,menu=[on|off]]

Snapshot mode

-snapshot

21.4. Display options

Disable graphics

-nographic

VGA card emulation

-vga <type>

Supported types:

- **cirrus** - Cirrus Logic GD5446 Video card.
- **std** - Standard VGA card with Bochs VBE extensions.
- **qxl** - Spice paravirtual card.
- **none** - Disable VGA card.

VNC display

-vnc <display>[,<option>[,<option>[,...]]]

Supported display value:

- [<host>]:<port>
- unix:<path>
- none - Supported with no other options specified.

Supported options are:

- **to**=<port>
- **reverse**
- **password**
- **tls**
- **x509**=</path/to/certificate/dir> - Supported when **tls** specified.
- **x509verify**=</path/to/certificate/dir> - Supported when **tls** specified.
- **sasl**
- **acl**

Spice desktop

-spice option[,option[,...]]

Supported options are:

- **port**=<number>
- **addr**=<addr>
- **ipv4**

  ipv6
- **password**=<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**=<channel>

  plaintext-channel=<channel>
- **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]

21.5. Network options

TAP network

-netdev tap,id=<id>][,<options>...]

The following options are supported (all use name=value format):

- ifname
- fd
- script
- downscript
- sndbuf
- vnet_hdr
- vhost
- vhostfd
- vhostforce

21.6. Device options

General device

-device <driver>[,<prop>[=<value>][,...]]

All drivers support following properties

- id
- bus

Following drivers are supported (with available properties):

- **pci-assign**
  - host
  - bootindex
  - configfd
  - addr
  - rombar
  - romfile
  - multifunction

  If the device has multiple functions, all of them need to be assigned to the same guest.
- **rtl8139**
  - mac
  - netdev
  - bootindex
  - addr
- **e1000**
  - mac
  - netdev
  - bootindex
  - addr
- **virtio-net-pci**
  - ioeventfd
  - vectors
  - 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
  - revision
  - cmdlog

  - addr
- **ide-drive**
  - unit
  - drive
  - physical_block_size
  - bootindex
  - ver
- **virtio-blk-pci**
  - class
  - drive
  - logical_block_size
  - physical_block_size
  - min_io_size
  - opt_io_size
  - bootindex
  - ioeventfd
  - vectors
  - indirect_desc
  - event_idx
  - scsi
  - addr
- **isa-serial**
  - index
  - iobase
  - irq
  - chardev
- **virtserialport**
  - nr
  - chardev
  - name

- **virtconsole**
  - nr
  - chardev
  - name
- **virtio-serial-pci**
  - vectors
  - class
  - indirect_desc
  - event_idx
  - max_ports
  - flow_control
  - addr
- **ES1370**
  - addr
- **AC97**
  - addr
- **intel-hda**
  - addr
- **hda-duplex**
  - cad
- **i6300esb**
  - addr
- **ib700** - no properties
- **sga** - no properties
- **virtio-balloon-pci**
  - indirect_desc
  - event_idx
  - addr
- **usb-tablet**
  - migrate

  - port
- **usb-kbd**
  - migrate
  - port
- **usb-mouse**
  - migrate
  - port
- **usb-ccid** - supported since 6.2
  - port
  - slot
- **usb-host** - tech preview since 6.2
  - hostbus
  - hostaddr
  - hostport
  - vendorid
  - productid
  - isobufs
  - port
- **usb-hub** - supported since 6.2
  - port
- **usb-ehci** - tech preview since 6.2
  - freq
  - maxframes
  - port
- **usb-storage** - tech preview since 6.2
  - drive
  - logical_block_size
  - physical_block_size
  - min_io_size
  - opt_io_size

- bootindex
- serial
- removable
- port

Global device setting

-global <device>.<property>=<value>

Supported devices and properties as in "General device" section with these additional devices:

- **isa-fdc**
  - driveA
  - driveB
  - bootindexA
  - bootindexB
- **qxl-vga**
  - ram_size
  - vram_size
  - revision
  - cmdlog
  - addr

Character device

-chardev backend,id=<id>[,<options>]

Supported backends are:

- **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> - standard i/o
- **spicevmc**,id=<id>,name=<name> - spice channel

Enable USB

-usb

21.7. Linux/Multiboot boot

Kernel file

-kernel <bzImage>

Ram disk

-initrd <file>

-initrd "<file1> arg=<foo>,<file2>

Command line parameter

-append <cmdline>

21.8. Expert options

KVM virtualization

-enable-kvm

Qemu-kvm supports only KVM virtualization and it is used by default if available. If -enable-kvm is used and KVM is not available, qemu-kvm fails. However, if -enable-kvm is not used and KVM is not available, qemu-kvm runs in TCG mode, which is not supported.

Disable kernel mode PIT reinjection

-no-kvm-pit-reinjection

No reboot

-no-reboot

Serial port, monitor, QMP

-serial <dev>

-monitor <dev>

-qmp <dev>

Supported devices are:

- **stdio** - standard input/output
- **null** - null device
- **file**:<filename> - output to file.
- **tcp**:[<host>]:<port>[,server][,nowait][,nodelay] - TCP Net console.
- **unix**:<path>[,server][,nowait] - Unix domain socket.
- **mon**:<dev_string> - Any device above, used to multiplex monitor too.
- **none** - disable, valid only for -serial.

- **chardev**:<id> - character device created with -chardev.

Monitor redirect

-mon <chardev_id>[,mode=[readline|control]][,default=[on|off]]

Manual CPU start

-S

RTC

-rtc [base=utc|localtime|date][,clock=host|vm][,driftfix=none|slew]

Watchdog

-watchdog model

Watchdog reaction

-watchdog-action <action>

Guest memory backing

-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>]

21.9. Help and information options

Help

-h

-help

Version

-version

Audio help

-audio-help

21.10. Miscellaneous options

Migration

-incoming

No default configuration

-nodefconfig

-nodefaults

Running without -nodefaults is not supported

Device configuration file

-readconfig <file>

-writeconfig <file>

Loaded saved state

-loadvm <file>

Troubleshooting

This chapter covers common problems and solutions for Red Hat Enterprise Linux 6 virtualization issues.

Read this chapter to develop an understanding of some of the common problems associated with virtualization technologies. Troubleshooting takes practice and experience which are difficult to learn from a book. It is recommended that you experiment and test virtualization on Red Hat Enterprise Linux 6 to develop your troubleshooting skills.

If you cannot find the answer in this document there may be an answer online from the virtualization community. Refer to *Section A.1, "Online resources"* for a list of Linux virtualization websites.

22.1. Debugging and troubleshooting tools

This section summarizes the System Administrator applications, the networking utilities, and debugging tools. You can employ these standard System administration tools and logs to assist with troubleshooting:

- **kvm_stat**
- **trace-cmd**
- **vmstat**
- **iostat**
- **lsof**
- **systemtap**
- **crash**
- **sysrq**
- **sysrq t**
- **sysrq w**

These networking tools can assist with troubleshooting virtualization networking problems:

- **ifconfig**
- **tcpdump**

  The **tcpdump** command 'sniffs' network packets. **tcpdump** is useful for finding network abnormalities and problems with network authentication. There is a graphical version of **tcpdump** named **wireshark**.

- **brctl**

  brctl is a networking tool that inspects and configures the Ethernet bridge configuration in the Virtualization linux kernel. You must have root access before performing these example commands:

```
# 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.feffffffff
designated-root        8000.feffffffff
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
```

Listed below are some other useful commands for troubleshooting virtualization.

- **strace** is a command which traces system calls and events received and used by another process.
- **vncviewer**: connect to a VNC server running on your server or a virtual machine. Install **vncviwer** using the **`yum install vnc`** command.
- **vncserver**: start a remote desktop on your server. Gives you the ability to run graphical user interfaces such as virt-manager via a remote session. Install **vncserver** using the **`yum install vnc-server`** command.

22.2. kvm_stat

The **`kvm_stat`** command is a python script which retrieves runtime statistics from the `kvm` kernel module. The **`kvm_stat`** command can be used to diagnose guest behavior visible to `kvm`. In particular, performance related issues with guests. Currently, the reported statistics are for the entire system; the behavior of all running guests is reported.

The **`kvm_stat`** command requires that the `kvm` kernel module is loaded and **`debugfs`** is mounted. If either of these features are not enabled, the command will output the required steps to enable **`debugfs`** or the `kvm` module. For example:

```
# kvm_stat
Please mount debugfs ('mount -t debugfs debugfs /sys/kernel/debug')
and ensure the kvm modules are loaded
```

Mount **`debugfs`** if required:

```
# mount -t debugfs debugfs /sys/kernel/debug
```

kvm_stat output

The **`kvm_stat`** command outputs statistics for all guests and the host. The output is updated until the command is terminated (using **`Ctrl`**+**`c`** or the **`q`** key).

```
# 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
```

Explanation of variables:

efer_reload
: The number of Extended Feature Enable Register (EFER) reloads.

exits
: The count of all **VMEXIT** calls.

fpu_reload
: The number of times a **VMENTRY** reloaded the FPU state. The **fpu_reload** is incremented when a guest is using the Floating Point Unit (FPU).

halt_exits
: Number of guest exits due to **halt** calls. This type of exit is usually seen when a guest is idle.

halt_wakeup
: Number of wakeups from a **halt**.

host_state_reload
: Count of full reloads of the host state (currently tallies MSR setup and guest MSR reads).

hypercalls
: Number of guest hypervisor service calls.

insn_emulation
: Number of guest instructions emulated by the host.

insn_emulation_fail
: Number of failed **insn_emulation** attempts.

io_exits
: Number of guest exits from I/O port accesses.

irq_exits
: Number of guest exits due to external interrupts.

irq_injections
: Number of interrupts sent to guests.

irq_window
: Number of guest exits from an outstanding interrupt window.

largepages
: Number of large pages currently in use.

mmio_exits
: Number of guest exits due to memory mapped I/O (MMIO) accesses.

mmu_cache_miss
: Number of KVM MMU shadow pages created.

mmu_flooded
: Detection count of excessive write operations to an MMU page. This counts detected write operations not of individual write operations.

mmu_pde_zapped
: Number of page directory entry (PDE) destruction operations.

mmu_pte_updated
: Number of page table entry (PTE) destruction operations.

mmu_pte_write
: Number of guest page table entry (PTE) write operations.

mmu_recycled
: Number of shadow pages that can be reclaimed.

mmu_shadow_zapped
: Number of invalidated shadow pages.

mmu_unsync
: Number of non-synchronized pages which are not yet unlinked.

nmi_injections
: Number of Non-maskable Interrupt (NMI) injections to the guest.

nmi_window
: Number of guest exits from (outstanding) Non-maskable Interrupt (NMI) windows.

pf_fixed
: Number of fixed (non-paging) page table entry (PTE) maps.

pf_guest
: Number of page faults injected into guests.

remote_tlb_flush
: Number of remote (sibling CPU) Translation Lookaside Buffer (TLB) flush requests.

request_irq
: Number of guest interrupt window request exits.

signal_exits
: Number of guest exits due to pending signals from the host.

tlb_flush
: Number of **`tlb_flush`** operations performed by the hypervisor.

Note

The output information from the **`kvm_stat`** command is exported by the KVM hypervisor as pseudo files located in the **`/sys/kernel/debug/kvm/`** directory.

22.3. Troubleshooting with serial consoles

Linux kernels can output information to serial ports. This is useful for debugging kernel panics and hardware issues with video devices or headless servers. The subsections in this section cover setting up serial console output for machines running Red Hat Enterprise Linux 6 virtualization kernels and their guests.

This section covers how to enable serial console output for fully virtualized guests.

Fully virtualized guest serial console output can be viewed with the **`virsh console`** command.

Be aware fully virtualized guest serial consoles have some limitations. Present limitations include:

- output data may be dropped or scrambled.

The serial port is called `ttyS0` on Linux or `COM1` on Windows.

You must configure the virtualized operating system to output information to the virtual serial port.

To output kernel information from a fully virtualized Linux guest into the domain, modify the **`/boot/grub/grub.conf`** file. Append the following to the *`kernel`* line: *`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
```

Reboot the guest.

On the host, access the serial console with the following command:

```
# virsh console
```

You can also use **`virt-manager`** to display the virtual text console. In the guest console window, select **Serial 1** in **Text Consoles** from the **View** menu.

22.4. Virtualization log files

- Each fully virtualized guest log is in the **/var/log/libvirt/qemu/** directory. Each guest log is named as *GuestName*.log and will be periodically compressed once a size limit is reached.

If you encounter any errors with the Virtual Machine Manager, you can review the generated data in the **virt-manager.log** file that resides in the **$HOME/.virt-manager** directory.

22.5. Loop device errors

If file-based guest images are used you may have to increase the number of configured loop devices. The default configuration allows up to eight active loop devices. If more than eight file-based guests or loop devices are needed the number of loop devices configured can be adjusted in the **/etc/modprobe.d/**directory. Add the following line:

```
options loop max_loop=64
```

This example uses 64 but you can specify another number to set the maximum loop value. You may also have to implement loop device backed guests on your system. To use a loop device backed guests for a full virtualized system, use the **phy: device** or **file: file** commands.

22.6. Enabling Intel VT-x and AMD-V virtualization hardware extensions in BIOS

This section describes how to identify hardware virtualization extensions and enable them in your BIOS if they are disabled.

The Intel VT-x extensions can be disabled in the BIOS. Certain laptop vendors have disabled the Intel VT-x extensions by default in their CPUs.

The virtualization extensions cannot be disabled in the BIOS for AMD-V.

The virtualization extensions are sometimes disabled in BIOS, usually by laptop manufacturers. Refer to the following section for instructions on enabling disabled virtualization extensions.

Verify the virtualization extensions are enabled in BIOS. The BIOS settings for Intel® VT or AMD-V are usually in the **Chipset** or **Processor** menus. The menu names may vary from this guide, the virtualization extension settings may be found in **Security Settings** or other non standard menu names.

Procedure 22.1. Enabling virtualization extensions in BIOS

1. Reboot the computer and open the system's BIOS menu. This can usually be done by pressing the **delete** key, the **F1** key or **Alt** and **F4** keys depending on the system.

2. **Enabling the virtualization extensions in BIOS**

Note: BIOS steps

Many of the steps below may vary depending on your motherboard, processor type, chipset and OEM. Refer to your system's accompanying documentation for the correct information on configuring your system.

a. Open the **Processor** submenu The processor settings menu may be hidden in the **Chipset**, **Advanced CPU Configuration** or **Northbridge**.

b. Enable **Intel Virtualization Technology** (also known as Intel VT-x). **AMD-V** extensions cannot be disabled in the BIOS and should already be enabled. The virtualization extensions may be labeled **Virtualization Extensions**, **Vanderpool** or various other names depending on the OEM and system BIOS.

c. Enable Intel VT-d or AMD IOMMU, if the options are available. Intel VT-d and AMD IOMMU are used for PCI device assignment.

d. Select **Save & Exit**.

3. Reboot the machine.

4. When the machine has booted, run **`cat /proc/cpuinfo | grep "vmx svm"`**. If the command outputs, the virtualization extensions are now enabled. If there is no output your system may not have the virtualization extensions or the correct BIOS setting enabled.

22.7. KVM networking performance

By default, KVM virtual machines are assigned a virtual Realtek 8139 (rtl8139) NIC (network interface controller) if they are Windows guests or the guest type is not specified. Red Hat Enterprise Linux guests are assigned a virtio NIC by default.

The rtl8139 virtualized NIC works fine in most environments. However, this device can suffer from performance degradation problems on some networks, for example, a 10 Gigabit Ethernet network.

To improve performance switch to the para-virtualized network driver.

Note

Note that the virtualized Intel PRO/1000 (`e1000`) driver is also supported as an emulated driver choice. To use the **`e1000`** driver, replace **`virtio`** in the procedure below with **`e1000`**. For the best performance it is recommended to use the **`virtio`** driver.

Procedure 22.2. Switching to the virtio driver

1. Shutdown the guest operating system.

2. Edit the guest's configuration file with the **`virsh`** command (where *GUEST* is the guest's name):

```
# virsh edit GUEST
```

The **`virsh edit`** command uses the **`$EDITOR`** shell variable to determine which editor to use.

3. Find the network interface section of the configuration. This section resembles the snippet below:

```
<interface type='network'>
  [output truncated]
  <model type='rtl8139' />
</interface>
```

4. Change the type attribute of the model element from *'rtl8139'* to *'virtio'*. This will change the driver from the rtl8139 driver to the e1000 driver.

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

5. Save the changes and exit the text editor

6. Restart the guest operating system.

Creating new guests using other network drivers

Alternatively, new guests can be created with a different network driver. This may be required if you are having difficulty installing guests over a network connection. This method requires you to have at least one guest already created (possibly installed from CD or DVD) to use as a template.

1. Create an XML template from an existing guest (in this example, named *Guest1*):

```
# virsh dumpxml Guest1 > /tmp/guest-template.xml
```

2. Copy and edit the XML file and update the unique fields: virtual machine name, UUID, disk image, MAC address, and any other unique parameters. Note that you can delete the UUID and MAC address lines and virsh will generate a UUID and MAC address.

```
# cp /tmp/guest-template.xml /tmp/new-guest.xml
# vi /tmp/new-guest.xml
```

Add the model line in the network interface section:

```
 <interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

3. Create the new virtual machine:

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

Appendix A. Additional resources

To learn more about virtualization and Red Hat Enterprise Linux, refer to the following resources.

A.1. Online resources

- *http://www.libvirt.org/* is the official website for the **libvirt** virtualization API.
- *http://virt-manager.et.redhat.com/* is the project website for the **Virtual Machine Manager** (virt-manager), the graphical application for managing virtual machines.
- Open Virtualization Center

  http://www.openvirtualization.com[1]
- Red Hat Documentation

  http://www.redhat.com/docs/
- Virtualization technologies overview

  http://virt.kernelnewbies.org[2]
- Red Hat Emerging Technologies group

  http://et.redhat.com[3]

A.2. Installed documentation

- **man virsh** and **/usr/share/doc/libvirt-<*version-number*>** — Contains sub commands and options for the **virsh** virtual machine management utility as well as comprehensive information about the **libvirt** virtualization library API.
- **/usr/share/doc/gnome-applet-vm-<*version-number*>** — Documentation for the GNOME graphical panel applet that monitors and manages locally-running virtual machines.
- **/usr/share/doc/libvirt-python-<*version-number*>** — Provides details on the Python bindings for the **libvirt** library. The **libvirt-python** package allows python developers to create programs that interface with the **libvirt** virtualization management library.
- **/usr/share/doc/python-virtinst-<*version-number*>** — Provides documentation on the **virt-install** command that helps in starting installations of Fedora and Red Hat Enterprise Linux related distributions inside of virtual machines.
- **/usr/share/doc/virt-manager-<*version-number*>** — Provides documentation on the Virtual Machine Manager, which provides a graphical tool for administering virtual machines.

Appendix B. Revision History

Revision 0.1-91 **Fri Dec 02 2011** **Scott Radvan** *sradvan@redhat.com*

Release for GA of Red Hat Enterprise Linux 6.2.

Revision 0.1-87 **Mon Nov 28 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#753068.

Revision 0.1-86 **Mon Nov 28 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#755487.

Revision 0.1-85 **Mon Nov 21 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#754910.

Revision 0.1-84 **Tue Nov 15 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#753041.

Revision 0.1-83 **Tue Nov 15 2011** **Scott Radvan** *sradvan@redhat.com*

Further typographical errors.

Revision 0.1-82 **Tue Nov 15 2011** **Scott Radvan** *sradvan@redhat.com*

Minor typographical errors.

Revision 0.1-81 **Tue Nov 15 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#753018.

Revision 0.1-80 **Tue Nov 15 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#753711.

Revision 0.1-79 **Thu Nov 03 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#746336.

Revision 0.1-78 **Thu Nov 03 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#745474.

Revision 0.1-76 **Tue Oct 25 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#744990.

Revision 0.1-75 **Tue Oct 25 2011** **Scott Radvan** *sradvan@redhat.com*

Resolves BZ#748310.

Revision 0.1-73 **Tue Oct 18 2011** **Scott Radvan** *sradvan@redhat.com*

Add script for migrating non-running guests.

Revision 0.1-69 **Mon Sep 19 2011** **Scott Radvan** *sradvan@redhat.com*

QEMU/KVM whitelist chapter updated with minor fixes.

Revision 0.1-68 **Fri Sep 16 2011** **Scott Radvan** *sradvan@redhat.com*

Chapter 15 restructure, add Guest CPU model configuration details (BZ#737109)

Revision 0.1-67 **Thu Sep 15 2011** **Scott Radvan** *sradvan@redhat.com*

Add qemu-kvm whitelist

Revision 0.1-66 **Tue Sep 13 2011** **Scott Radvan** *sradvan@redhat.com*

BZ#735224

Revision 0.1-65 **Mon Sep 12 2011** **Scott Radvan** *sradvan@redhat.com*

BZ#735207
BZ#735209
BZ#735211
BZ#735219
BZ#735220

BZ#735223

Revision 0.1-64 **Tue Sep 06 2011** **Scott Radvan** ***sradvan@redhat.com***

BZ#734998

Revision 0.1-03 **Thu Jun 2 2011** **Scott Radvan** ***sradvan@redhat.com***

Add draft watermark

Revision 0.1-02 **Thu Jun 2 2011** **Scott Radvan** ***sradvan@redhat.com***

Import majority of text, add alpha channels to PNGs, bump for publish.

Index