Red Hat Enterprise Linux 6

Virtualization Host Configuration and Guest Installation Guide

Virtualization Documentation

Red Hat Enterprise Linux 6 Virtualization Host Configuration and Guest Installation Guide
Virtualization Documentation
Edition 0.2

Author

Copyright © 2011 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at *http://creativecommons.org/licenses/by-sa/3.0/*. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

All other trademarks are the property of their respective owners.

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

This guide covers KVM packages, compatibility and restrictions. Also included are host configuration details and instructions for installing guests of different types, PCI device assignment and SR-IOV.

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_Host_Configuration_and_Guest_Installation_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.

Introduction

1.1. What's in this guide?

This guide provides information on system requirements and restrictions, package details, host configuration and detailed instructions for installing different types of guests.

System Requirements

This chapter lists system requirements for successfully running virtualized guest machines, referred to as VMs on Red Hat Enterprise Linux 6. Virtualization is available for Red Hat Enterprise Linux 6 on the Intel 64 and AMD64 architecture.

The KVM hypervisor is provided with Red Hat Enterprise Linux 6.

For information on installing the virtualization packages, see *Chapter 5, Installing the virtualization packages*.

Minimum system requirements

- 6GB free disk space
- 2GB of RAM.

Recommended system requirements

- 6GB plus the required disk space recommended by each guest operating system. For most operating systems more than 6GB of disk space is recommended.
- One processor core or hyper-thread for the maximum number of virtualized CPUs in a guest and one for the host.
- 2GB of RAM plus additional RAM for virtualized guests.

> **KVM overcommit**
>
> KVM can overcommit physical resources for virtualized guests. Overcommitting resources means the total virtualized RAM and processor cores used by the guests can exceed the physical RAM and processor cores on the host. For information on safely overcommitting resources with KVM refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

KVM requirements

The KVM hypervisor requires:

- an Intel processor with the Intel VT-x and Intel 64 extensions for x86-based systems, or
- an AMD processor with the AMD-V and the AMD64 extensions.

Refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* to determine if your processor has the virtualization extensions.

Storage support

The guest storage methods are:

- files on local storage,
- physical disk partitions,
- locally connected physical LUNs,
- LVM partitions,

- NFS shared file systems,
- iSCSI,
- GFS2 clustered file systems,
- Fibre Channel-based LUNs, and
- SRP devices (SCSI RDMA Protocol), the block export protocol used in Infiniband and 10GbE iWARP adapters.

KVM Guest VM Compatibility

To verify whether your processor supports the virtualization extensions and for information on enabling the virtualization extensions if they are disabled, refer to the *Red Hat Enterprise Linux Virtualization Administration Guide*.

3.1. Red Hat Enterprise Linux 6 support limits

Red Hat Enterprise Linux 6 servers have certain support limits.

The following URLs explain the processor and memory amount limitations for Red Hat Enterprise Linux:

- For host systems: *http://www.redhat.com/rhel/compare/*
- For hypervisors: *http://www.redhat.com/rhel/virtualization/compare/*

The following URL is a complete chart showing supported operating systems and host and guest combinations:

- *http://www.redhat.com/rhel/server/virtualization_support.html#virt_matrix*

3.2. Supported CPU Models

Red Hat Enterprise Linux 6 supports the use of the following QEMU CPU model definitions:

`Opteron_G3`
: AMD Opteron 23xx (Gen 3 Class Opteron)

`Opteron_G2`
: AMD Opteron 22xx (Gen 2 Class Opteron)

`Opteron_G1`
: AMD Opteron 240 (Gen 1 Class Opteron)

`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-rhel5)

`cpu64-rhel6`
: Red Hat Enterprise Linux 6 supported QEMU Virtual CPU version (cpu64-rhel6)

Virtualization restrictions

This chapter covers additional support and product restrictions of the virtualization packages in Red Hat Enterprise Linux 6.

4.1. KVM restrictions

The following restrictions apply to the KVM hypervisor:

Maximum VCPUs per guest
: Virtualized guests support up to a maximum of 64 virtualized CPUs in Red Hat Enterprise Linux 6.

Constant TSC bit
: Systems without a Constant Time Stamp Counter require additional configuration. Refer to *Chapter 14, KVM guest timing management* for details on determining whether you have a Constant Time Stamp Counter and configuration steps for fixing any related issues.

Memory overcommit
: KVM supports memory overcommit and can store the memory of guests in swap. A guest will run slower if it is swapped frequently. Red Hat *Knowledgebase*[1] has an article on safely and efficiently determining the size of the swap partition. When KSM is used for memory overcommitting, make sure that the swap size follows the recommendations described in this article.

CPU overcommit
: It is not recommended to have more than 10 virtual CPUs per physical processor core. Customers are encouraged to use a capacity planning tool in order to determine the CPU overcommit ratio. Estimating an ideal ratio is difficult as it is highly dependent on each workload. For instance, a guest may consume 100% CPU on one use case, and multiple guests may be completely idle on another.

  Red Hat does not support running more VCPUs to a single guest than the amount of overall physical cores that exist on the system. While Hyperthreads can be considered as cores, their performance can also vary from one scenario to the next, and they should not be expected to perform as well as regular cores.

  Refer to the *Red Hat Enterprise Linux Virtualization Administration Guide* for tips and recommendations on overcommitting CPUs.

Virtualized SCSI devices
: SCSI emulation is not supported with KVM in Red Hat Enterprise Linux.

Virtualized IDE devices
: KVM is limited to a maximum of four virtualized (emulated) IDE devices per guest.

Para-virtualized devices
: Para-virtualized devices, which use the **`virtio`** drivers, are PCI devices. Presently, guests are limited to a static definition of 221 PCI devices. Some PCI devices are critical for the guest to run and these devices cannot be removed. The default, required devices are:

  - the host bridge,
  - the ISA bridge and usb bridge (The usb and isa bridges are the same device),

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

- the graphics card (using either the Cirrus or qxl driver), and
- the memory balloon device.

Migration restrictions

Live migration is only possible between hosts with the same CPU type (that is, Intel to Intel or AMD to AMD only).

For live migration, both hosts must have the same value set for the No eXecution (NX) bit, either **on** or **off** for both hosts.

Storage restrictions

Guest should not be given write access to whole disks or block devices (for example, **/dev/sdb**). Virtualized guests with access to block devices may be able to access other block devices 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.

SR-IOV restrictions

SR-IOV is only thoroughly tested with the following devices (other SR-IOV devices may work but have not been tested at the time of release):

Intel® 82576NS Gigabit Ethernet Controller (**igb** driver)

Intel® 82576EB Gigabit Ethernet Controller (**igb** driver)

Neterion X3100 Series 10GbE PCIe (**vxge** driver)

Intel® 82599ES 10 Gigabit Ethernet Controller (**ixgbe** driver)

Intel® 82599EB 10 Gigabit Ethernet Controller (**ixgbe** driver)

For live migration, both hosts must have the same value set for the No eXecution (NX) bit, either **on** or **off** for both hosts.

PCI device assignment restrictions

PCI device assignment (attaching PCI devices to guests) requires host systems to have AMD IOMMU or Intel VT-d support to enable device assignment of PCI-e devices.

For parallel/legacy PCI, only single devices behind a PCI bridge are supported.

Multiple PCIe endpoints connected through a non-root PCIe switch require ACS support in the PCIe bridges of the PCIe switch. This restriction can be disabled in **/etc/libvirt/qemu.conf**, setting **relaxed_acs_check=1**

Red Hat Enterprise Linux 6 has limited PCI configuration space access by guest device drivers. This limitation could cause drivers that are dependent on PCI configuration space to fail configuration.

Red Hat Enterprise Linux 6.2 introduces interrupt remapping as a requirement for PCI device assignment. If your platform does not provide support for interrupt remapping, the KVM check for this support can be circumvented with the following command: **echo 1 > /sys/module/kvm/parameters/allow_unsafe_assigned_interrupts**

4.2. Application restrictions

There are aspects of virtualization which make virtualization unsuitable for certain types of applications.

Applications with high I/O throughput requirements should use the para-virtualized drivers for fully virtualized guests. Without the para-virtualized drivers certain applications may be unpredictable under heavy I/O loads.

The following applications should be avoided for their high I/O requirement reasons:

- **kdump** server
- **netdump** server

You should carefully evaluate applications and tools that heavily utilize I/O or those that require real-time performance. Consider the para-virtualized drivers or PCI device assignment for increased I/O performance. Refer to *Chapter 10, KVM Para-virtualized Drivers* for more information on the para-virtualized drivers for fully virtualized guests. Refer to *Chapter 12, PCI device assignment* for more information on PCI device assignment.

Applications still suffer a small performance loss from running in virtualized environments. The performance benefits of virtualization through consolidating to newer and faster hardware should be evaluated against the potential application performance issues associated with using virtualization.

4.3. Other restrictions

For the list of all other restrictions and issues affecting virtualization read the *Red Hat Enterprise Linux 6 Release Notes*. The *Red Hat Enterprise Linux 6 Release Notes* cover the present new features, known issues and restrictions as they are updated or discovered.

Installing the virtualization packages

Before you can use virtualization, the virtualization packages must be installed on your computer. Virtualization packages can be installed either during the host installation sequence or after host installation using the **yum** command and the Red Hat Network (RHN).

The KVM hypervisor uses the default Red Hat Enterprise Linux kernel with the *kvm* kernel module.

5.1. Installing KVM with a new Red Hat Enterprise Linux installation

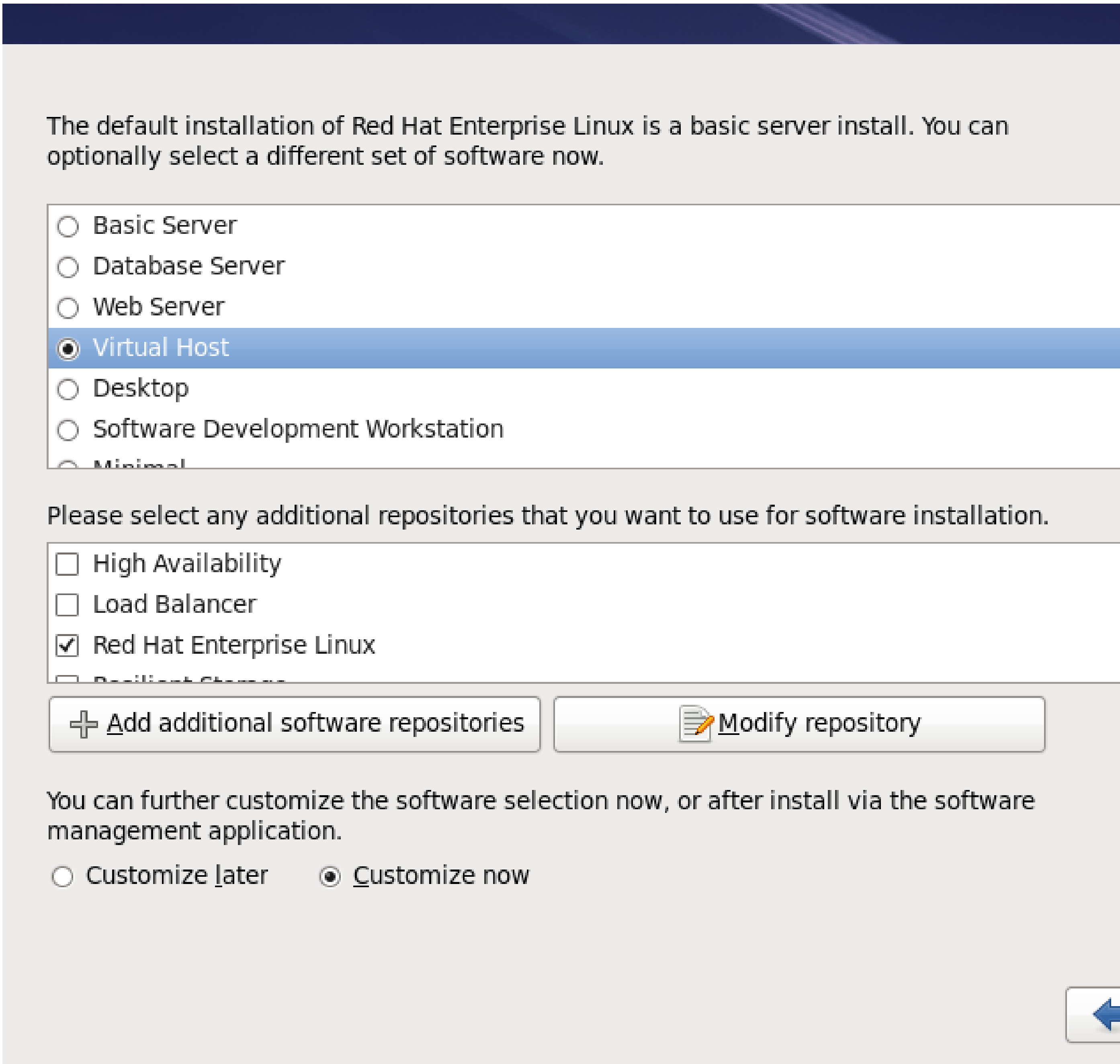
This section covers installing virtualization tools and virtualization packages as part of a fresh Red Hat Enterprise Linux installation.

Need help installing?

The *Installation Guide* (available from *http://docs.redhat.com*[1]) covers installing Red Hat Enterprise Linux in detail.

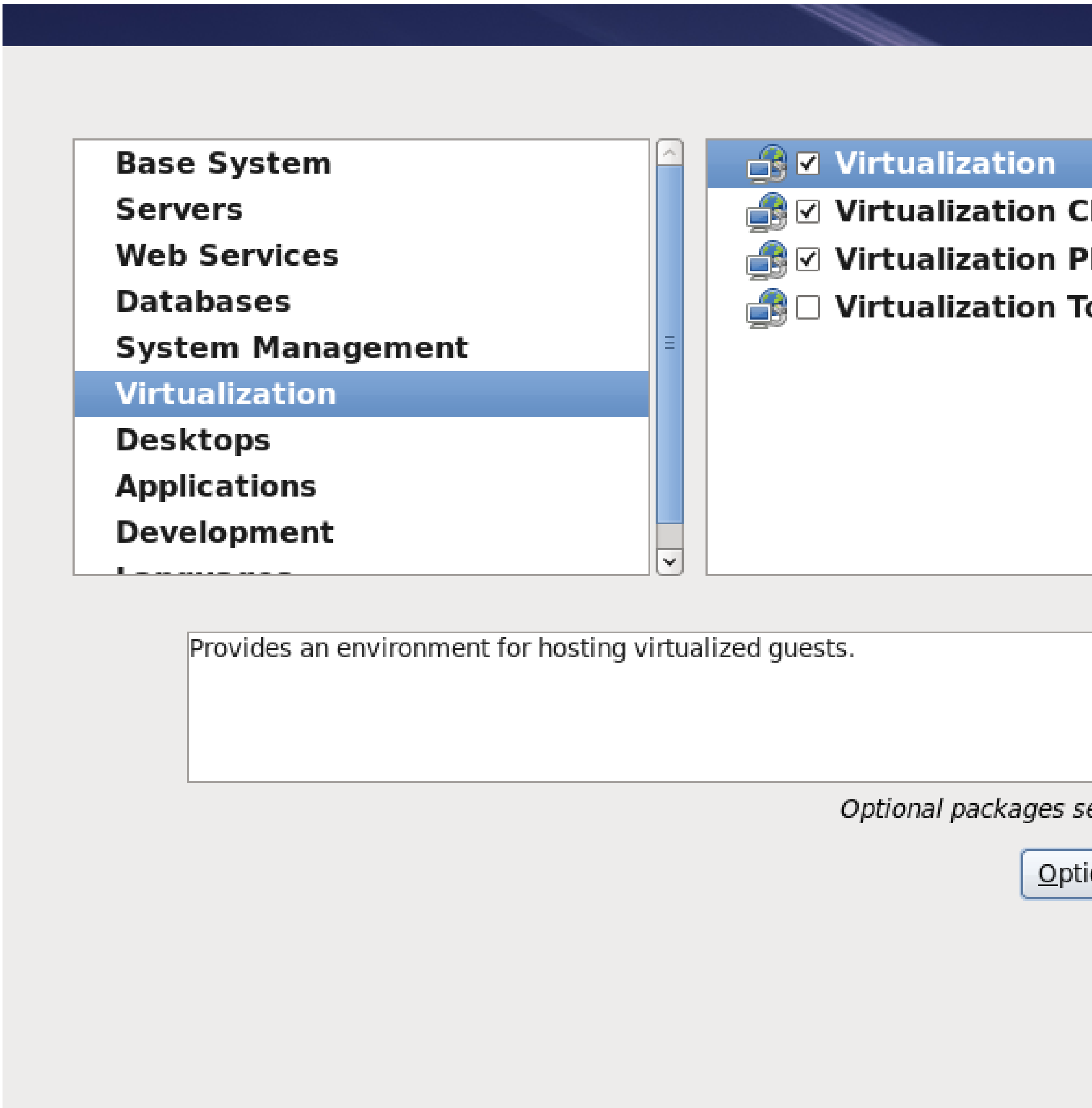
1. Start an interactive Red Hat Enterprise Linux installation from the Red Hat Enterprise Linux Installation CD-ROM, DVD or PXE.
2. You must enter a valid installation number when prompted to receive access to the virtualization and other Advanced Platform packages.
3. Complete the other steps up to the package selection step.

[1] http://docs.redhat.com/

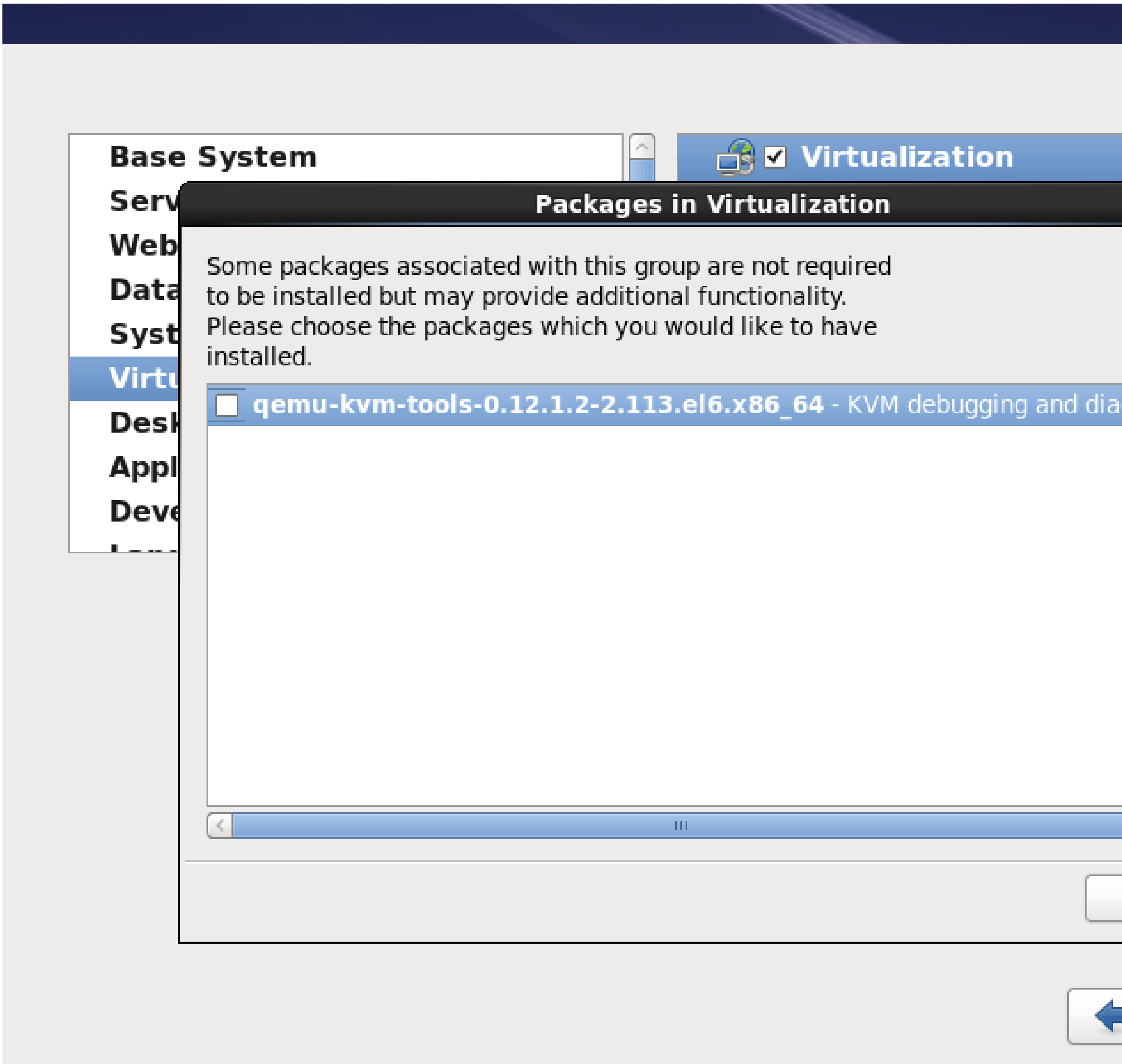


Select the **Virtual Host** server role to install a platform for virtualized guests. Alternatively, select the **Customize Now** radio button to specify individual packages.

4. Select the **Virtualization** package group. This selects the qemu-kvm emulator, **`virt-manager`**, **`libvirt`** and **`virt-viewer`** for installation.



5. **Customize the packages (if required)**
   Customize the **Virtualization** group if you require other virtualization packages.



Press the **Close** button then the **Next** button to continue the installation.

Note

You require a valid RHN virtualization entitlement to receive updates for the virtualization packages.

Installing KVM packages with Kickstart files

This section describes how to use a Kickstart file to install Red Hat Enterprise Linux with the Virtualization packages. Kickstart files allow for large, automated installations without a user manually installing each individual host system. The steps in this section will assist you in creating and using a Kickstart file to install Red Hat Enterprise Linux with the virtualization packages.

In the **%packages** section of your Kickstart file, append the following package groups:

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

More information on Kickstart files can be found on Red Hat's website, *http://docs.redhat.com*, in the *Installation Guide*.

5.2. Installing virtualization packages on an existing Red Hat Enterprise Linux system

This section describes the steps for installing the KVM hypervisor on a working Red Hat Enterprise Linux 6 or newer system.

To install the packages, your machines must be registered. To register an unregistered installation of Red Hat Enterprise Linux, run the **rhn_register** command and follow the prompts.

If you do not have a valid Red Hat subscription, visit the *Red Hat online store*[2] to obtain one.

Installing the virtualization packages with yum

To use virtualization on Red Hat Enterprise Linux you require at least the **qemu-kvm** and **qemu-img** packages. These packages provide the user-level KVM emulator and disk image manager on the host Red Hat Enterprise Linux system.

To install the **qemu-kvm** and **qemu-img** packages, run the following command:

```
# yum install qemu-kvm qemu-img
```

Several additional virtualization management packages are also available:

Recommended virtualization packages

python-virtinst
: Provides the **virt-install** command for creating virtual machines.

libvirt
: The *libvirt* package provides the server and host side libraries for interacting with hypervisors and host systems. The *libvirt* package provides the `libvirtd` daemon that handles the library calls, manages virtualized guests and controls the hypervisor.

libvirt-python
: The *libvirt-python* package contains a module that permits applications written in the Python programming language to use the interface supplied by the *libvirt* API.

virt-manager
: **virt-manager**, also known as **Virtual Machine Manager**, provides a graphical tool for administering virtual machines. It uses *libvirt-client* library as the management API.

[2] https://www.redhat.com/wapps/store/catalog.html

libvirt-client

The *libvirt-client* package provides the client-side APIs and libraries for accessing *libvirt* servers. The *libvirt-client* package includes the **`virsh`** command line tool to manage and control virtualized guests and hypervisors from the command line or a special virtualization shell.

Install all of these recommended virtualization packages with the following command:

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

Installing Virtualization package groups

The virtualization packages can also be installed from package groups. The following table describes the virtualization package groups and what they provide.

Note

Note that the **`qemu-img`** package is not distributed in any of the following groups. It must be installed manually with the **`yum install qemu-img`** command as described previously.

Table 5.1. Virtualization Package Groups

Package Group	Description	Mandatory Packages	Optional Packages
Virtualization	Provides an environment for hosting virtualized guests	qemu-kvm	qemu-kvm-tools
Virtualization Client	Clients for installing and managing virtualization instances	python-virtinst, virt-manager, virt-viewer	virt-top
Virtualization Platform	Provides an interface for accessing and controlling virtualized guests and containers	libvirt, libvirt-client	fence-virtd-libvirt, fence-virtd-multicast, fence-virtd-serial, libvirt-cim, libvirt-java, libvirt-qpid, perl-Sys-Virt
Virtualization Tools	Tools for offline virtual image management	libguestfs	libguestfs-java, libguestfs-mount, libguestfs-tools, virt-v2v

To install a package group, run the **`yum groupinstall <groupname>`** command. For instance, to install the **Virtualization Tools** package group, run the **`yum groupinstall "Virtualization Tools"`** command.

Virtualized guest installation overview

After you have installed the virtualization packages on the host system you can create guest operating systems. This chapter describes the general processes for installing guest operating systems on virtual machines. You can create guests using the **New** button in **virt-manager** or use the command line interface **`virt-install`**. Both methods are covered by this chapter.

Detailed installation instructions are available in the following chapters for specific versions of Red Hat Enterprise Linux and Microsoft Windows.

6.1. Virtualized guest prerequisites and considerations

Various factors should be considered before creating any virtualized guests. Not only should the role of a virtualized guest be considered before deployment, but regular ongoing monitoring and assessment based on variable factors (load, amount of clients) should be performed. Some factors include:

- Performance - Virtualized guests should be deployed and configured based on their intended tasks. Some guest systems (for instance, guests running a database server) may require special performance considerations. Guests may require more assigned CPUs or memory based on their role, and projected system load.
- Input/output requirements and types of input/output - Some guests may have a particularly high I/O requirement or may require further considerations or projections based on the type of I/O (for instance, typical disk block size access, or the amount of clients).
- Storage - Some guests may require higher priority access to storage, to faster disk types, or may require exclusive access to areas of storage. The amount of storage used by guests should also be regularly monitored and taken into account when deploying and maintaining storage.
- Networking and network infrastructure - Depending upon your environment, some guests could require faster network links than other guests. Bandwidth or latency are often factors when deploying and maintaining guests, especially as requirements or load changes.

6.2. Creating guests with virt-install

You can use the **`virt-install`** command to create virtualized guests from the command line. **`virt-install`** is used either interactively or as part of a script to automate the creation of virtual machines. Using **`virt-install`** with Kickstart files allows for unattended installation of virtual machines.

The **`virt-install`** tool provides a number of options that can be passed on the command line. Note that you need root privileges in order for **`virt-install`** commands to complete successfully. To see a complete list of options run the following command:

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

The **`virt-install`** man page also documents each command option and important variables.

`qemu-img` is a related command which may be used before **`virt-install`** to configure storage options.

An important option is the `--`*vnc* option which opens a graphical window for the guest's installation.

Example 6.1. Using virt-install to install a RHEL 5 guest

The following example creates a Red Hat Enterprise Linux 5 guest:

```
# virt-install \
   --name=guest1-rhel5-64 \
   --file=/var/lib/libvirt/images/guest1-rhel5-64.dsk \
   --file-size=8 \
   --nonsparse --vnc \
   --vcpus=2 --ram=2048 \
   --location=http://example1.com/installation_tree/RHEL5.6-Server-x86_64/os \
   --network bridge=br0 \
   --os-type=linux \
   --os-variant=rhel5.4
```

Note

When installing a Windows guest with **`virt-install`**, the **`--os-type=windows`** option is recommended. This option prevents the CD-ROM from disconnecting when rebooting during the installation procedure. The **`--os-variant`** option further optimizes the configuration for a specific guest operating system.

Refer to **`man virt-install`** for more examples.

6.3. Creating guests with virt-manager

`virt-manager`, also known as Virtual Machine Manager, is a graphical tool for creating and managing virtualized guests.

Procedure 6.1. Creating a virtualized guest with virt-manager

1. **Open virt-manager**
   Start **`virt-manager`**. Launch the **Virtual Machine Manager** application from the **Applications** menu and **System Tools** submenu. Alternatively, run the **`virt-manager`** command as root.

2. **Optional: Open a remote hypervisor**
   Select the hypervisor and press the **Connect** button to connect to the remote hypervisor.

3. **Create a new guest**
   The **virt-manager** window allows you to create a new virtual machine. Click the **Create a new virtual machine** button (*Figure 6.1, "Virtual Machine Manager window"*) to open the **New VM** wizard.

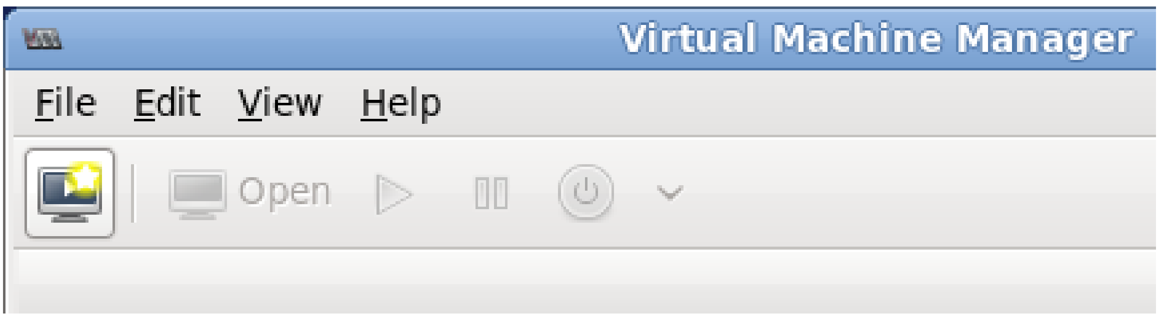


Figure 6.1. Virtual Machine Manager window

4. **New VM wizard**
   The **New VM** wizard breaks down the guest creation process into five steps:

   1. Naming the guest and choosing the installation type
   2. Locating and configuring the installation media
   3. Configuring memory and CPU options
   4. Configuring the guest's storage
   5. Configuring networking, architecture, and other hardware settings

   Ensure that **`virt-manager`** can access the installation media (whether locally or over the network).

5. **Specify name and installation type**
   The guest creation process starts with the selection of a name and installation type. Virtual machine names can have underscores (_), periods (`.`), and hyphens (`-`).

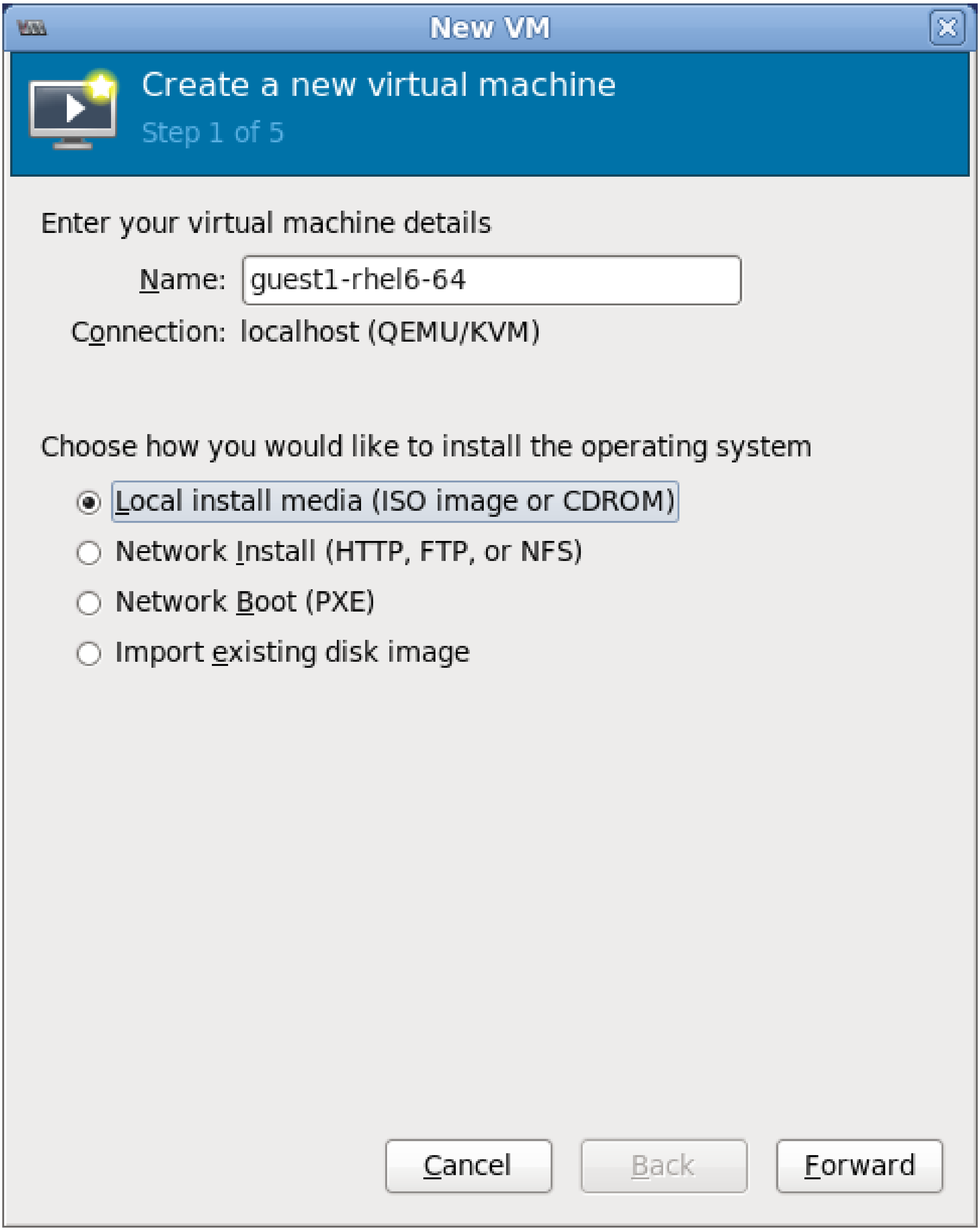


Figure 6.2. Name guest and select installation method

Type in a virtual machine name and choose an installation type:

Local install media (ISO image or CDROM)
: This method uses a CD-ROM, DVD, or image of an installation disk (e.g. **`.iso`**).

Network Install (HTTP, FTP, or NFS)
: Network installing involves the use of a mirrored Red Hat Enterprise Linux or Fedora installation tree to install a guest. The installation tree must be accessible through either HTTP, FTP, or NFS.

Network Boot (PXE)
: This method uses a Preboot eXecution Environment (PXE) server to install the guest. Setting up a PXE server is covered in the *Deployment Guide*. To install via network boot, the guest must have a routable IP address or shared network device. For information on the required networking configuration for PXE installation, refer to *Section 6.4, "Installing guests with PXE"*.

Import existing disk image
: This method allows you to create a new guest and import a disk image (containing a pre-installed, bootable operating system) to it.

Click **Forward** to continue.

6. **Configure installation**
   Next, configure the **OS type** and **Version** of the installation. Depending on the method of installation, provide the install URL or existing storage path.

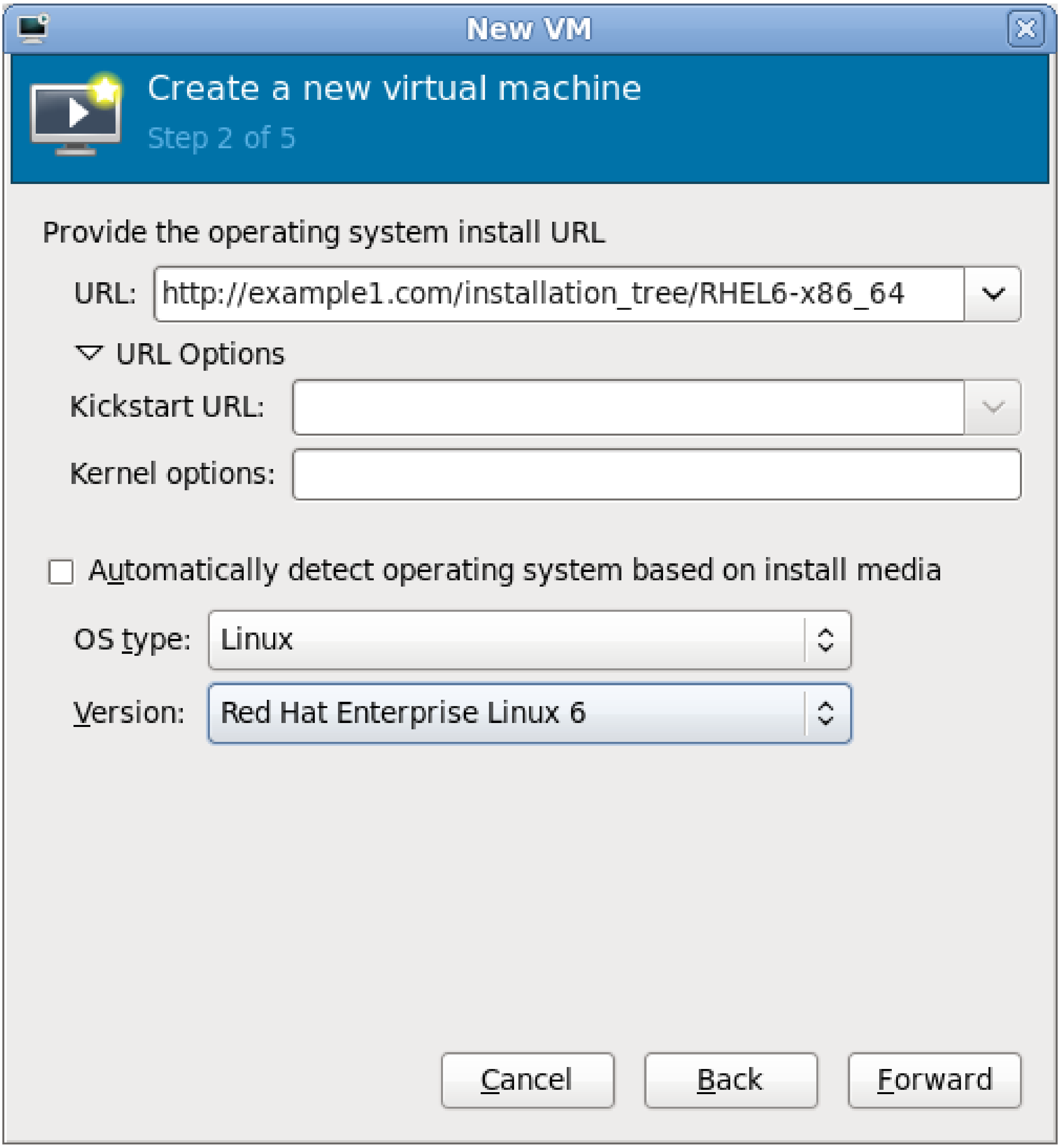


Figure 6.3. Remote installation URL

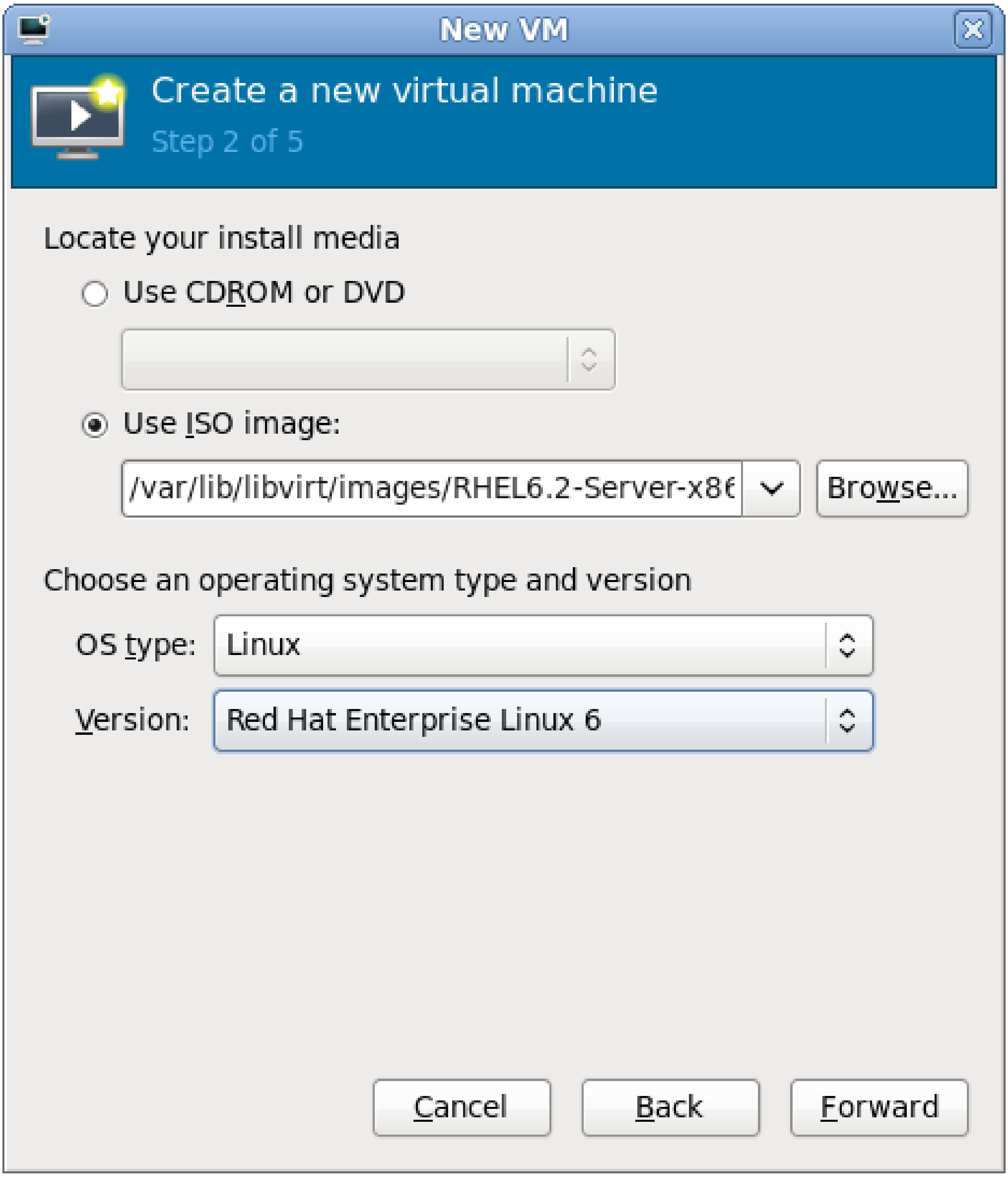


Figure 6.4. Local ISO image installation

7. **Configure CPU and memory**
   The next step involves configuring the number of CPUs and amount of memory to allocate to the virtual machine. The wizard shows the number of CPUs and amount of memory you can allocate; configure these settings and click **Forward**.

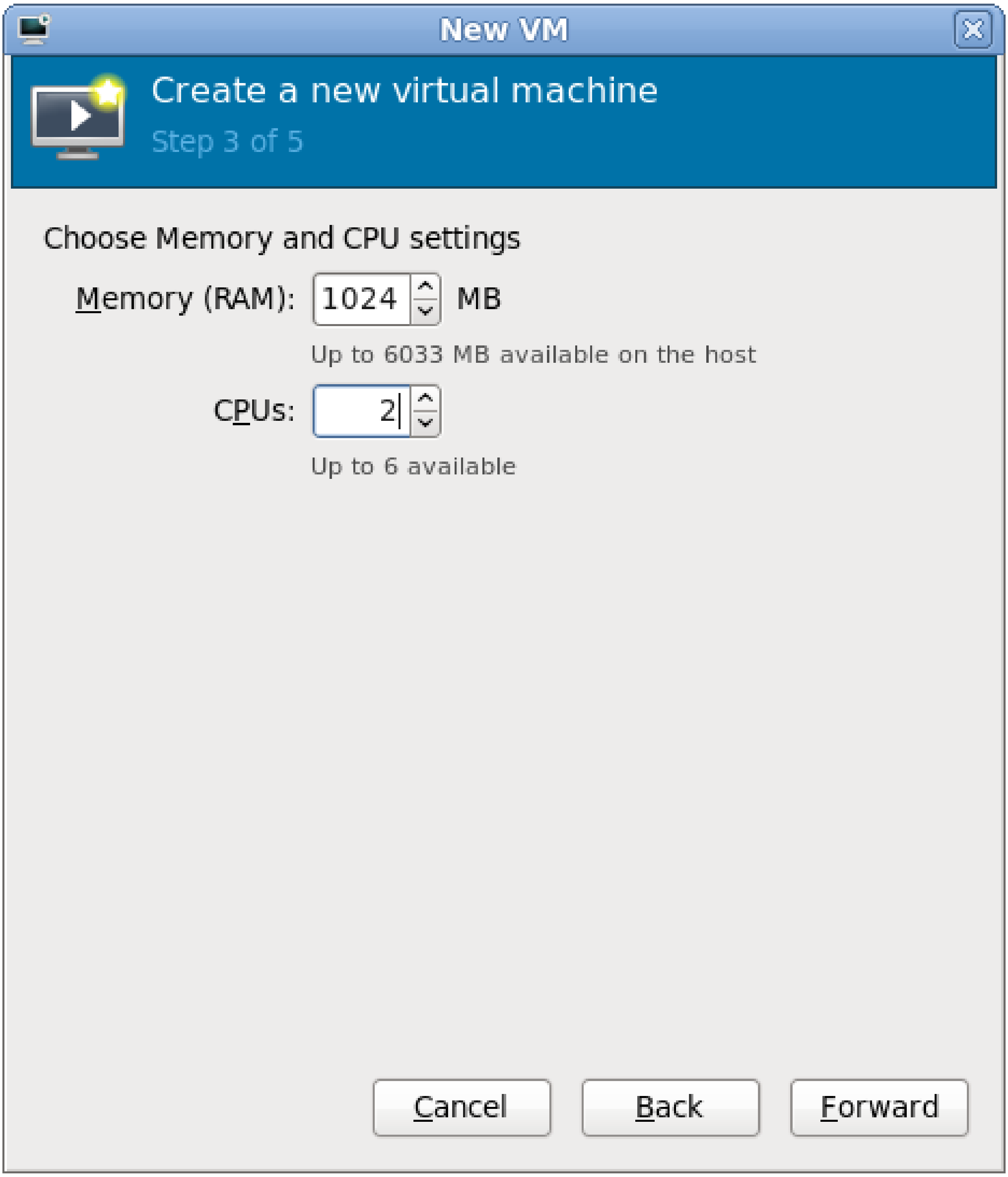


Figure 6.5. Configuring CPU and Memory

8. **Configure storage**
   Assign storage to the guest.

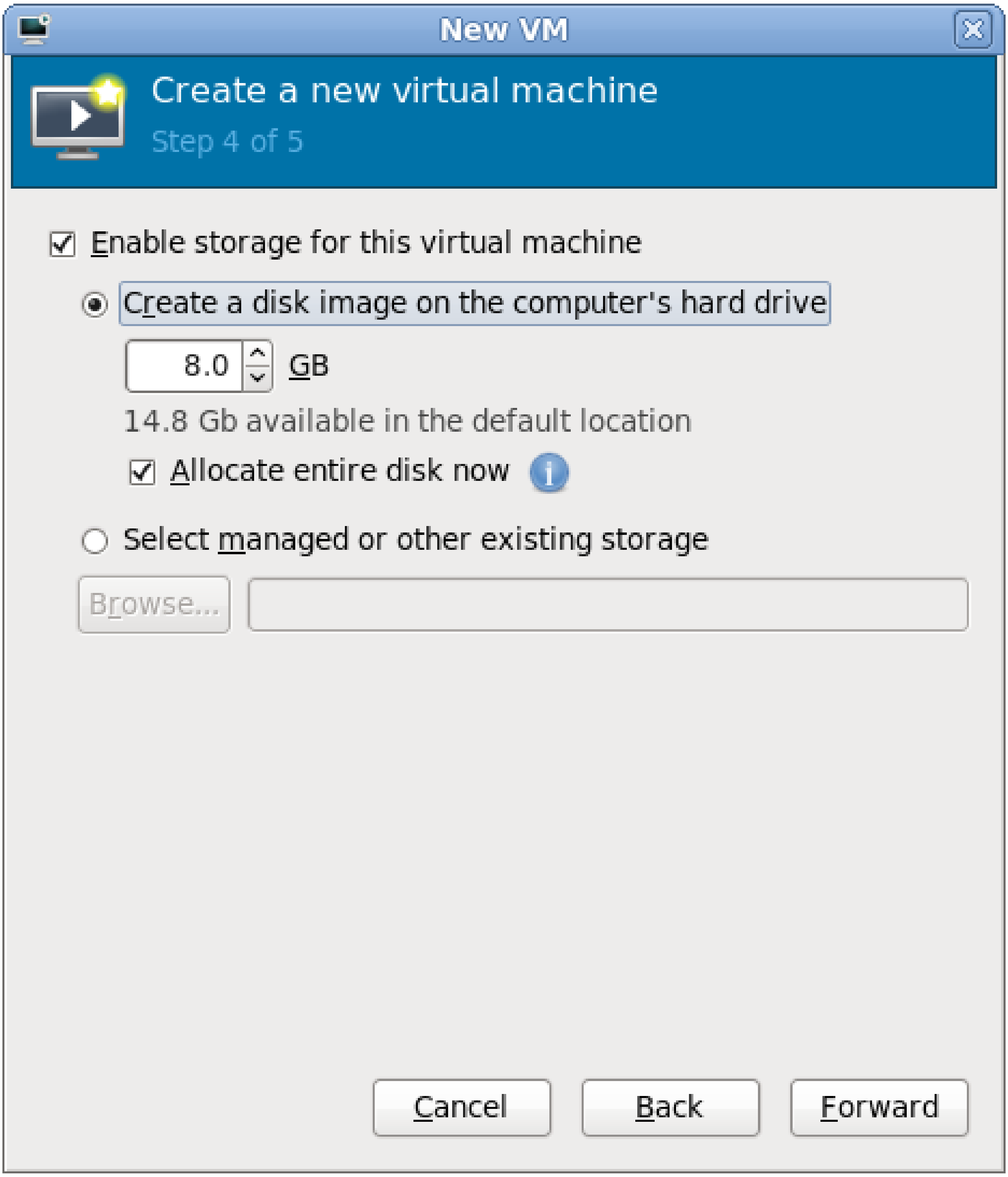


Figure 6.6. Configuring virtual storage

If you chose to import an existing disk image during the first step, **`virt-manager`** will skip this step.

Assign sufficient space for your virtualized guest and any applications the guest requires, then click **Forward** to continue.

9. **Final configuration**
   Verify the settings of the virtual machine and click **Finish** when you are satisfied; doing so will create the guest with default networking settings, virtualization type, and architecture.

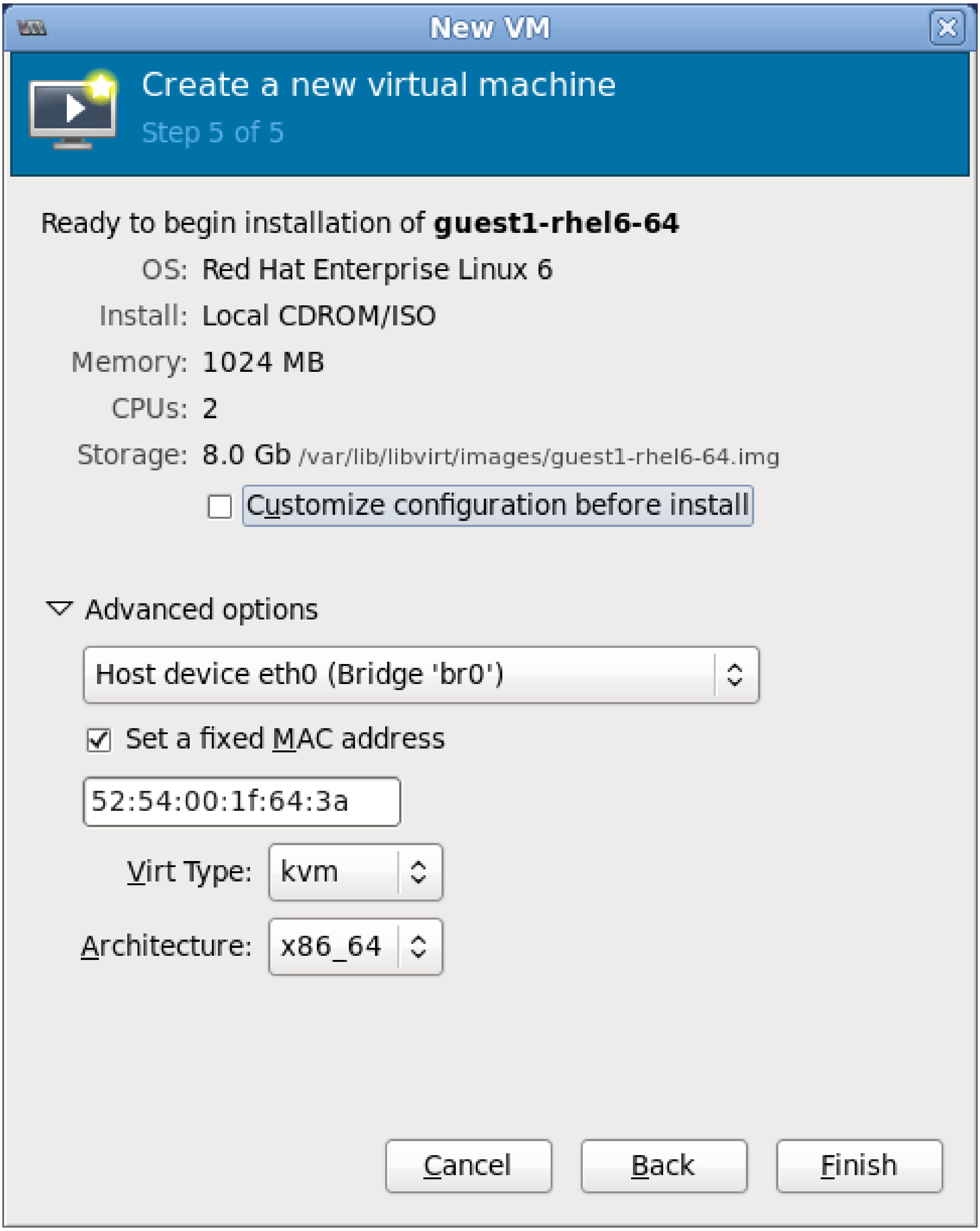


Figure 6.7. Verifying the configuration

If you prefer to further configure the virtual machine's hardware first, check the **Customize configuration before install** box first before clicking **Finish**. Doing so will open another wizard that will allow you to add, remove, and configure the virtual machine's hardware settings.

After configuring the virtual machine's hardware, click **Apply**. **virt-manager** will then create the guest with your specified hardware settings.

6.4. Installing guests with PXE

This section covers the steps required to install guests with PXE. PXE guest installation requires a shared network device, also known as a network bridge. The procedures below covers creating a bridge and the steps required to utilize the bridge for PXE installation.

1. **Create a new bridge**
   a. Create a new network script file in the **/etc/sysconfig/network-scripts/** directory. This example creates a file named **ifcfg-installation** which makes a bridge named *installation*.

```
# cd /etc/sysconfig/network-scripts/
# vim ifcfg-installation
DEVICE=installation
TYPE=Bridge
BOOTPROTO=dhcp
ONBOOT=yes
DELAY=0
```

Warning

The line, *TYPE=Bridge*, is case-sensitive. It must have uppercase 'B' and lower case 'ridge'.

   b. Start the new bridge by restarting the network service. The **ifup installation** command can start the individual bridge but it is safer to test the entire network restarts properly.

```
# service network restart
```

   c. There are no interfaces added to the new bridge yet. Use the **brctl show** command to view details about network bridges on the system.

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
installation    8000.000000000000       no
virbr0          8000.000000000000       yes
```

   The **virbr0** bridge is the default bridge used by **libvirt** for Network Address Translation (NAT) on the default Ethernet device.

2. **Add an interface to the new bridge**

   Edit the configuration file for the interface. Add the **BRIDGE** parameter to the configuration file with the name of the bridge created in the previous steps.

```
# Intel Corporation Gigabit Network Connection
DEVICE=eth1
BRIDGE=installation
```

```
BOOTPROTO=dhcp
HWADDR=00:13:20:F7:6E:8E
ONBOOT=yes
DELAY=0
```

After editing the configuration file, restart networking or reboot.

```
# service network restart
```

Verify the interface is attached with the **brctl show** command:

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
installation    8000.001320f76e8e       no              eth1
virbr0          8000.000000000000       yes
```

3. **Security configuration**
   Configure **iptables** to allow all traffic to be forwarded across the bridge.

```
# iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
# service iptables save
# service iptables restart
```

Disable iptables on bridges

Alternatively, prevent bridged traffic from being processed by **iptables** rules. In **/etc/sysctl.conf** append the following lines:

```
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

Reload the kernel parameters configured with **sysctl**.

```
# sysctl -p /etc/sysctl.conf
```

4. **Restart libvirt before the installation**
   Restart the **libvirt** daemon.

```
# service libvirtd reload
```

The bridge is configured, you can now begin an installation.

PXE installation with virt-install

For **virt-install** append the **--network=bridge:installation** installation parameter where *installation* is the name of your bridge. For PXE installations use the *--pxe* parameter.

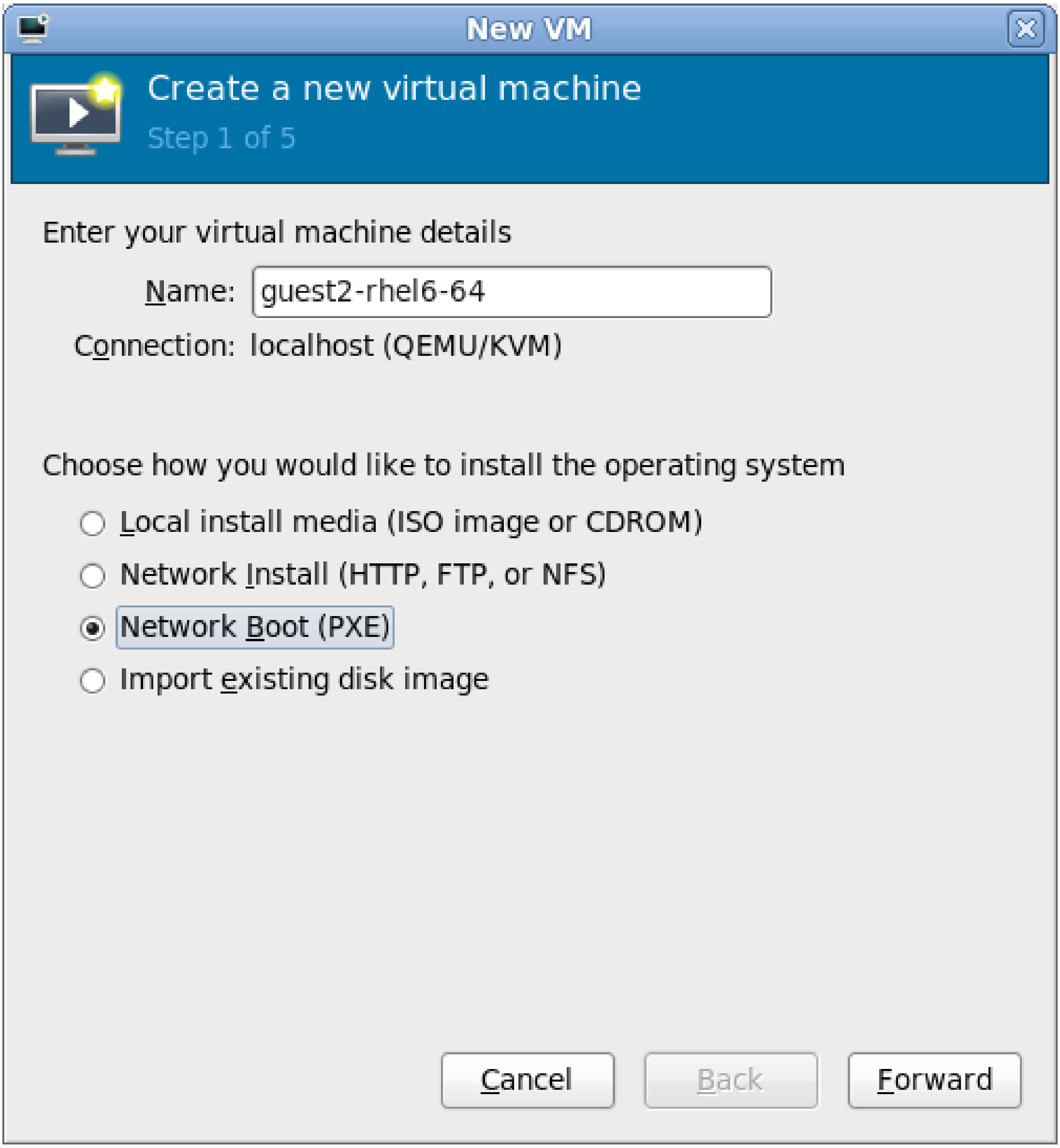
Example 6.2. PXE installation with virt-install

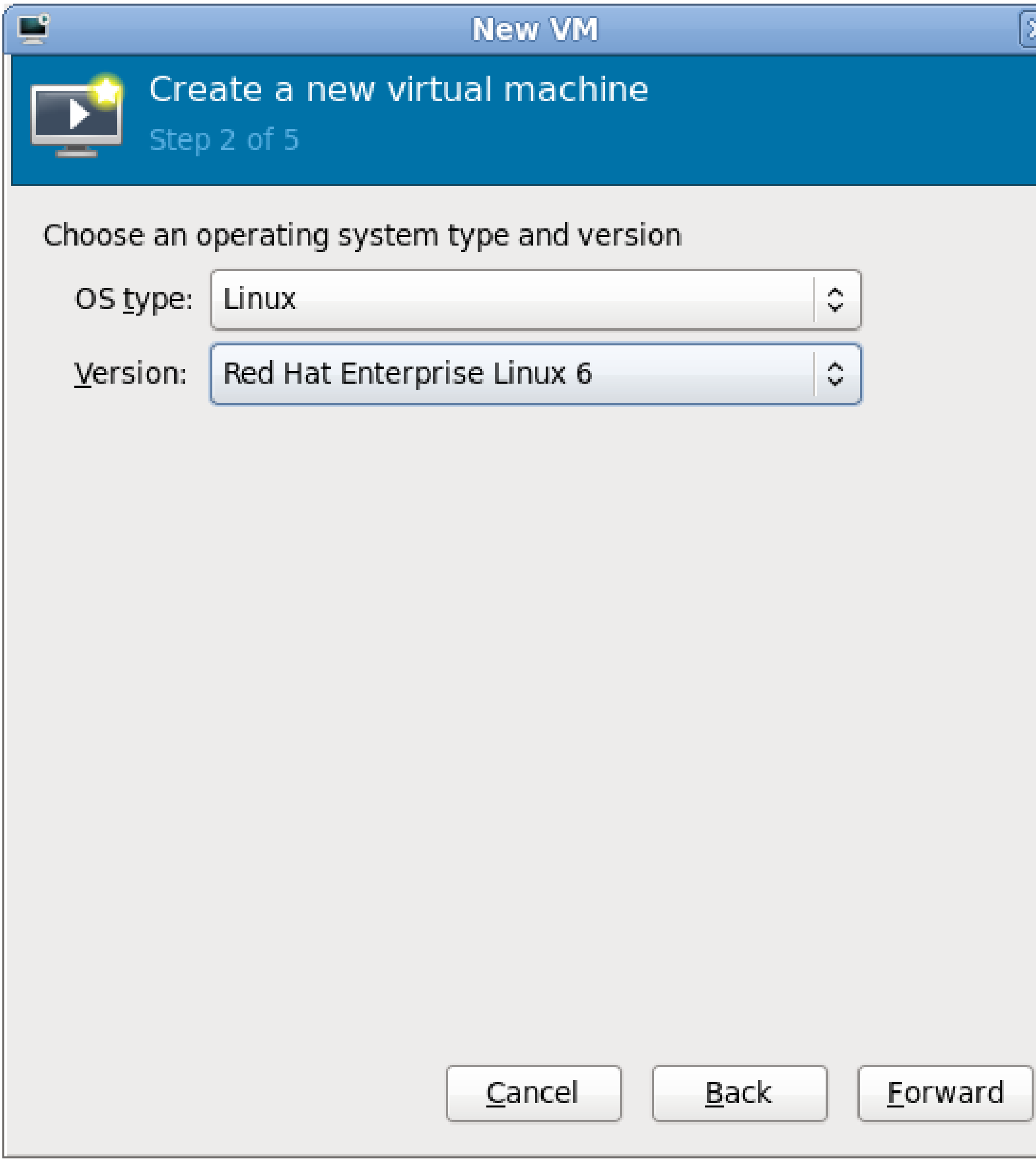
```
# virt-install --hvm --connect qemu:///system \
    --network=bridge:installation --pxe\
    --name EL10 --ram=756 \
    --vcpus=4
    --os-type=linux --os-variant=rhel5
    --file=/var/lib/libvirt/images/EL10.img \
```

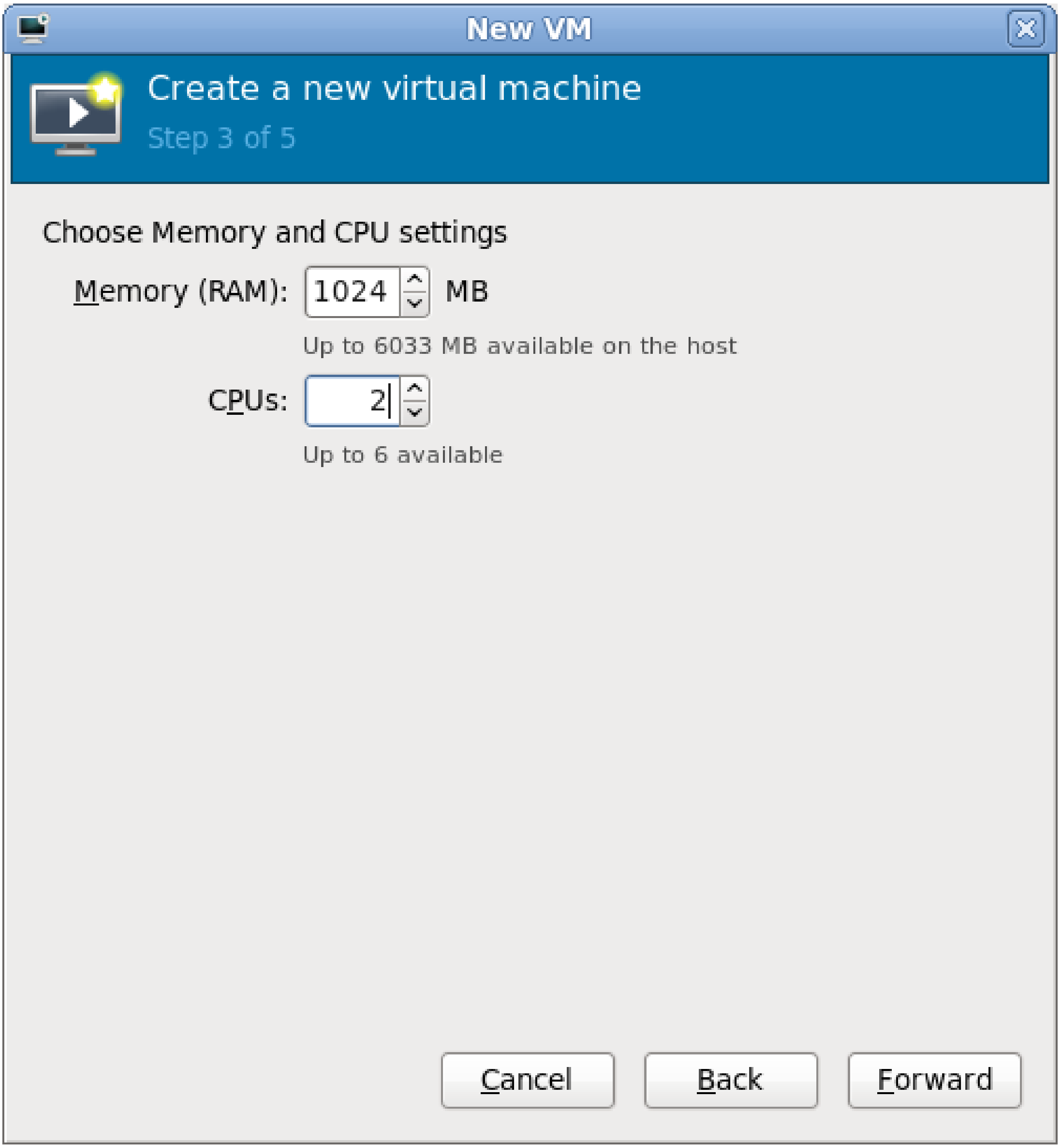
PXE installation with virt-manager

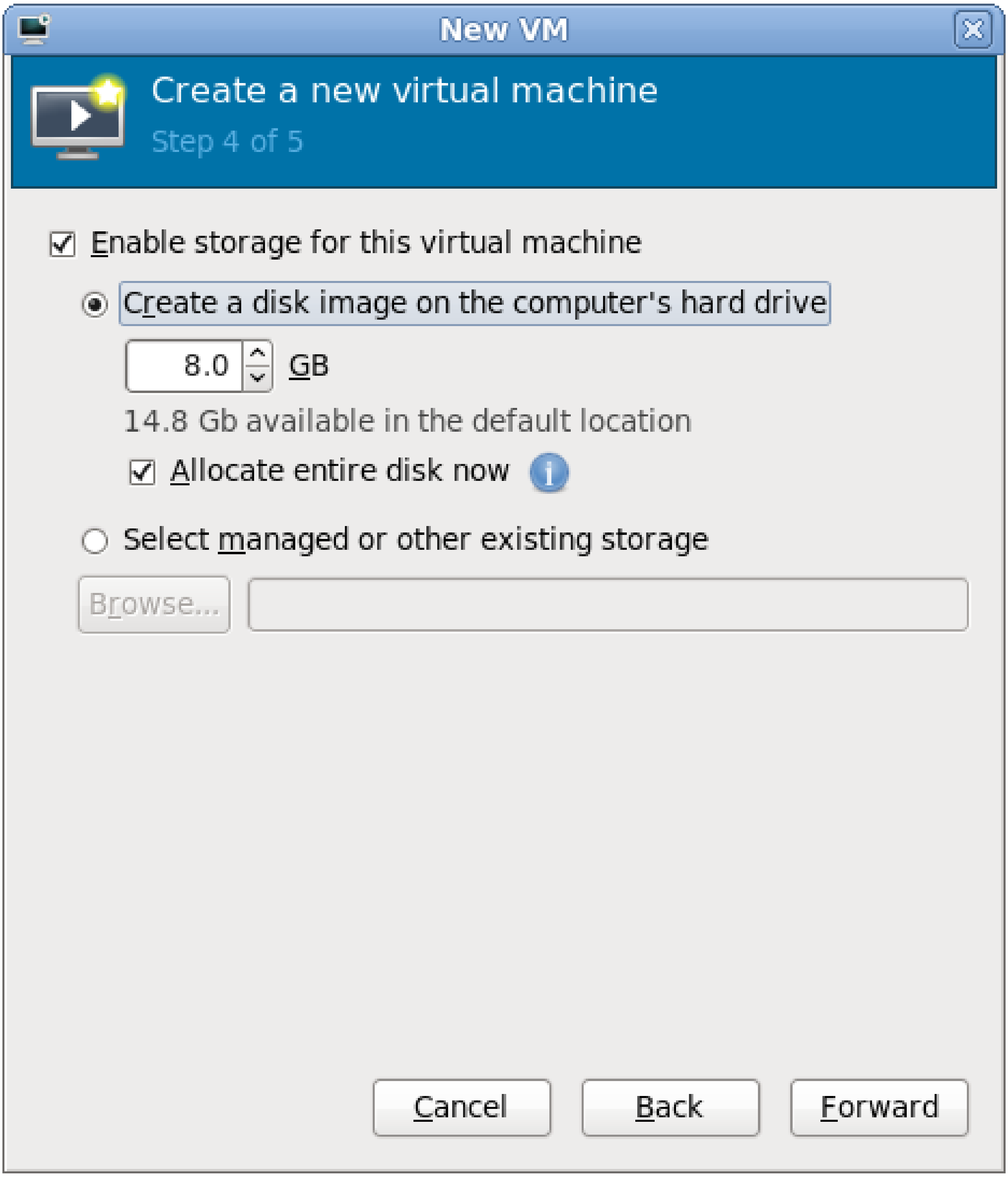
The steps below are the steps that vary from the standard **`virt-manager`** installation procedures.

1. **Select PXE**
   Select PXE as the installation method and follow the rest of the steps to configure the OS type, memory, CPU and storage settings.

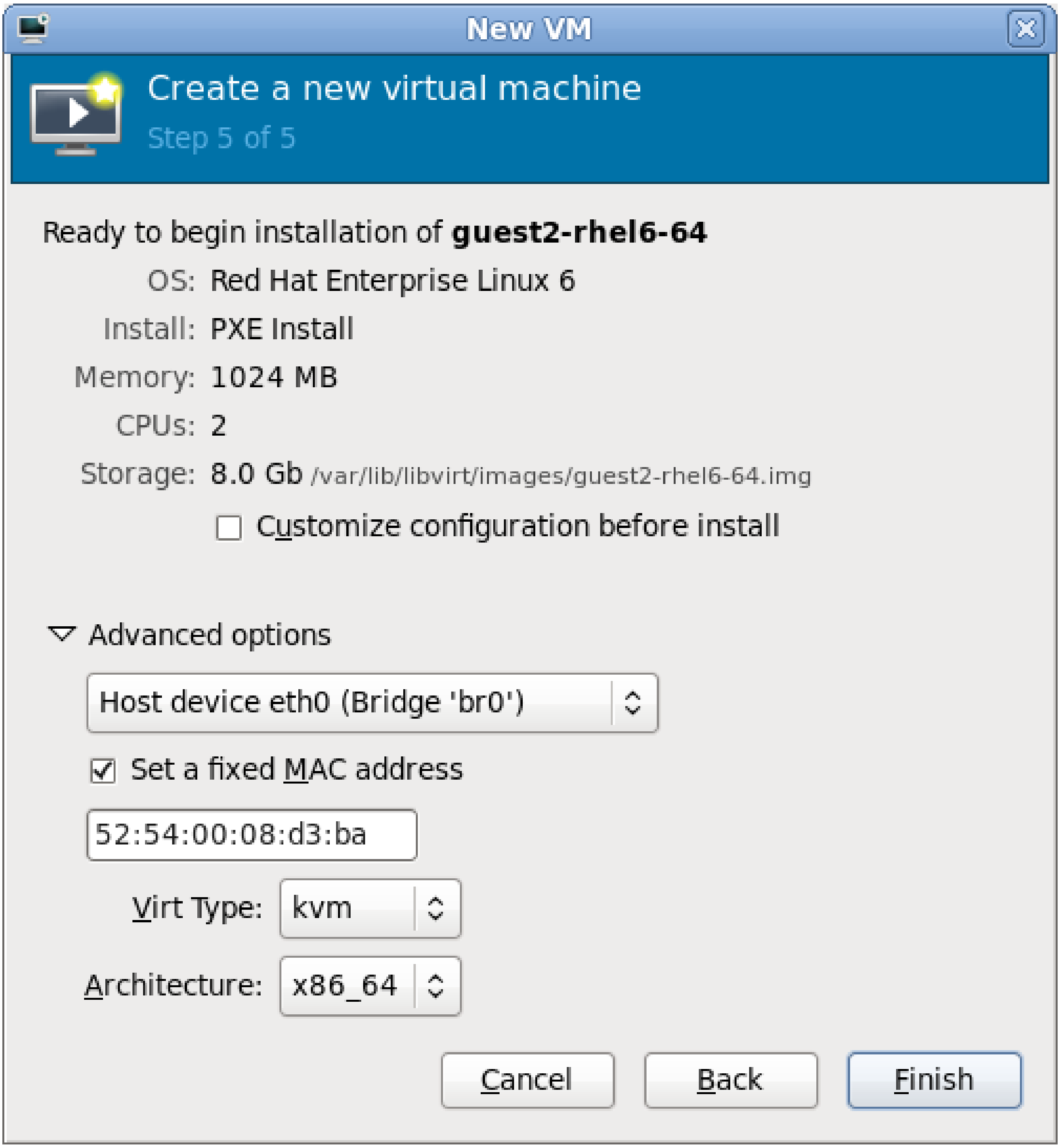

New VM
Create a new virtual machine
Step 1 of 5
Enter your virtual machine details
Name: guest2-rhel6-64
Connection: localhost (QEMU/KVM)
Choose how you would like to install the operating system
Local install media (ISO image or CDROM)
Network Install (HTTP, FTP, or NFS)
Network Boot (PXE)
Import existing disk image
Cancel
Back
Forward


New VM
Create a new virtual machine
Step 2 of 5
Choose an operating system type and version
OS type:
Linux
Version:
Red Hat Enterprise Linux 6
Cancel
Back
Forward


New VM
Create a new virtual machine
Step 3 of 5
Choose Memory and CPU settings
Memory (RAM): 1024 MB
Up to 6033 MB available on the host
CPUs: 2
Up to 6 available
Cancel
Back
Forward



2. **Start the installation**
   The installation is ready to start.



A DHCP request is sent and if a valid PXE server is found the guest installation processes will start.

Installing Red Hat Enterprise Linux 6 as a fully virtualized guest on Red Hat Enterprise Linux 6

This chapter covers how to install Red Hat Enterprise Linux 6 as a fully virtualized guest on a Red Hat Enterprise Linux 6 host.

These procedures assume that the KVM hypervisor and all other required packages are installed and the host is configured for virtualization.

> **Note**
>
> For more information on installing the virtualization packages, refer to *Chapter 5, Installing the virtualization packages*.

7.1. Creating a Red Hat Enterprise Linux 6 guest with local installation media

This procedure covers creating a virtualized Red Hat Enterprise Linux 6 guest with a locally stored installation DVD or DVD image. DVD images are available from *rhn.redhat.com*[1] for Red Hat Enterprise Linux 6.

Procedure 7.1. Creating a Red Hat Enterprise Linux 6 guest with virt-manager

1. **Optional: Preparation**
   Prepare the storage environment for the virtualized guest. For more information on preparing storage, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

   > **Note**
   >
   > Various storage types may be used for storing virtualized guests. However, for a guest to be able to use migration features the guest must be created on networked storage.

   Red Hat Enterprise Linux 6 requires at least 1GB of storage space. However, Red Hat recommends at least 5GB of storage space for a Red Hat Enterprise Linux 6 installation and for the procedures in this guide.

2. **Open virt-manager and start the wizard**
   Open virt-manager by executing the **`virt-manager`** command as root or opening **Applications** -> **System Tools** -> **Virtual Machine Manager**.

[1] http://rhn.redhat.com

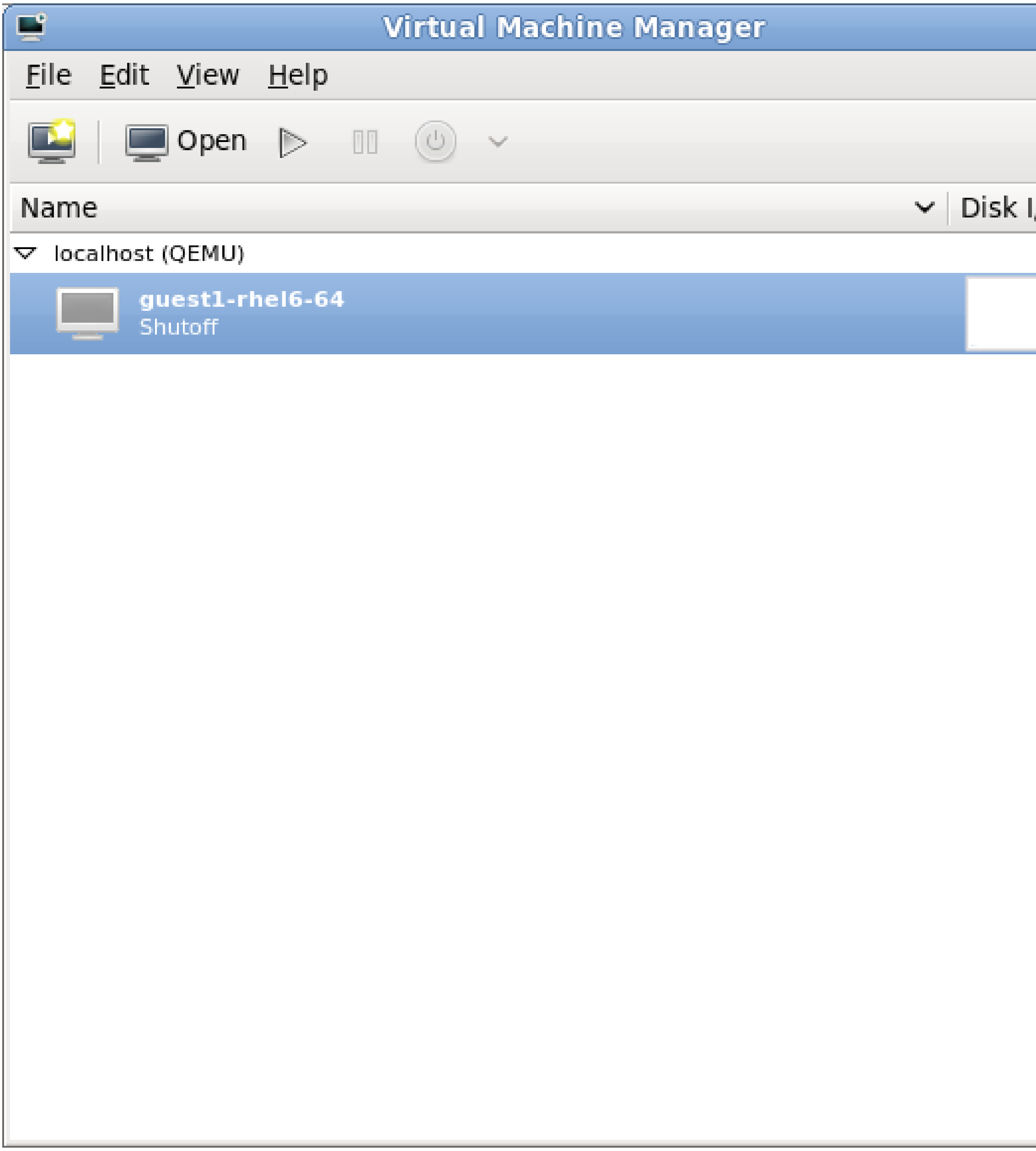


Figure 7.1. The Virtual Machine Manager window

Press the **Create new virtualized guest** button (see figure *Figure 7.2, “The create new virtualized guest button”*) to start the new virtualized guest wizard.

Figure 7.2. The create new virtualized guest button

The **New VM** window opens.

3. **Name the virtualized guest**
   Guest names can contain letters, numbers and the following characters: '_', '**.**' and '**-**'. Guest names must be unique for migration.

   Choose the **Local install media (ISO image or CDROM)** radio button.

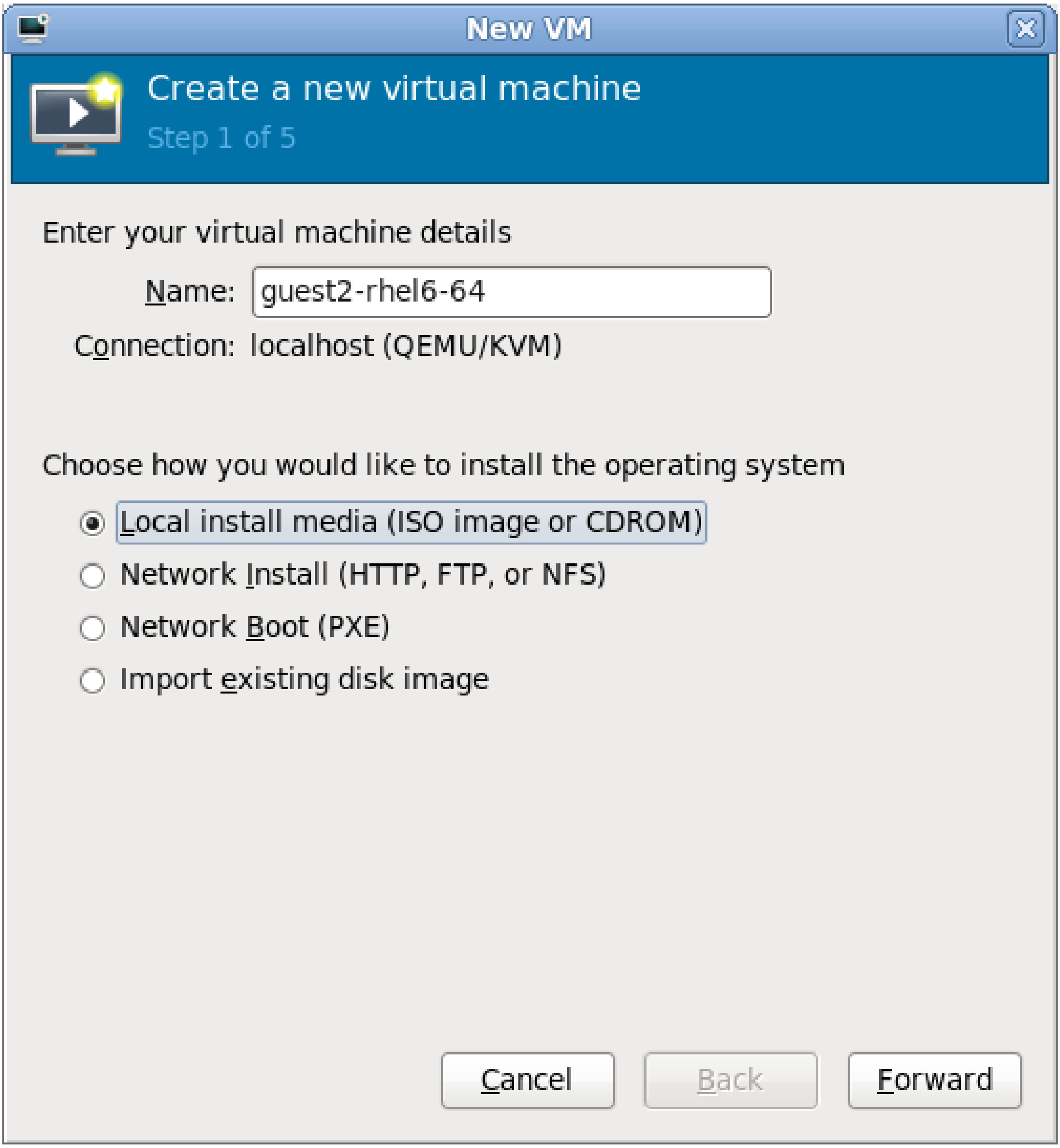


Figure 7.3. The New VM window - Step 1

Press **Forward** to continue.

4. **Select the installation media**
   Select the installation ISO image location or a DVD drive with the installation disc inside and press **Choose Volume** to continue. This example uses an ISO file image of a Red Hat Enterprise Linux installation DVD image.

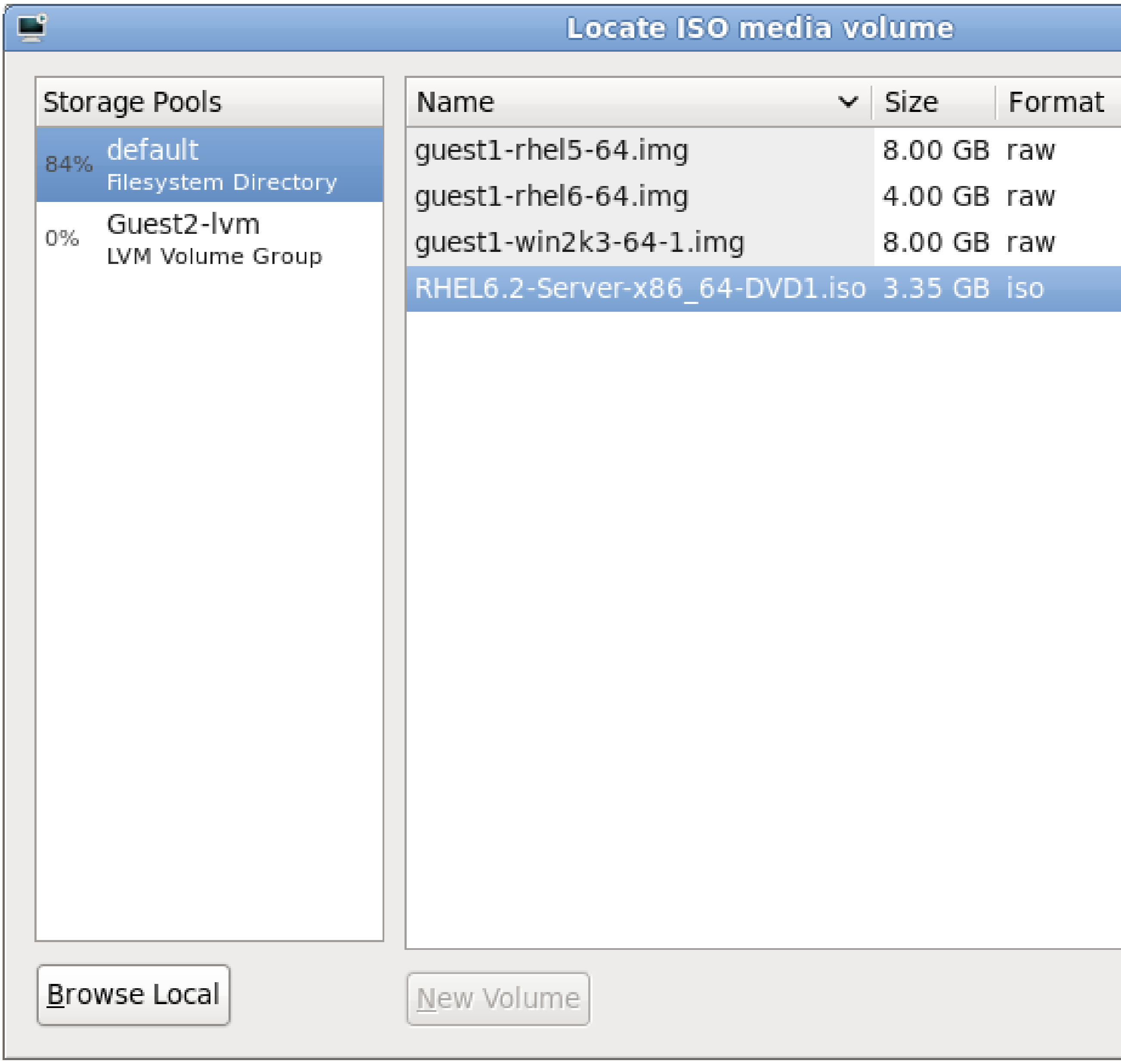


Figure 7.4. The Locate ISO media volume window

Image files and SELinux

For ISO image files and guest storage images, the recommended location to use is **`/var/lib/libvirt/images/`**. Any other location may require additional configuration by SELinux. Refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* for more details on configuring SELinux.

Select the operating system type and version which match the installation media you have selected.

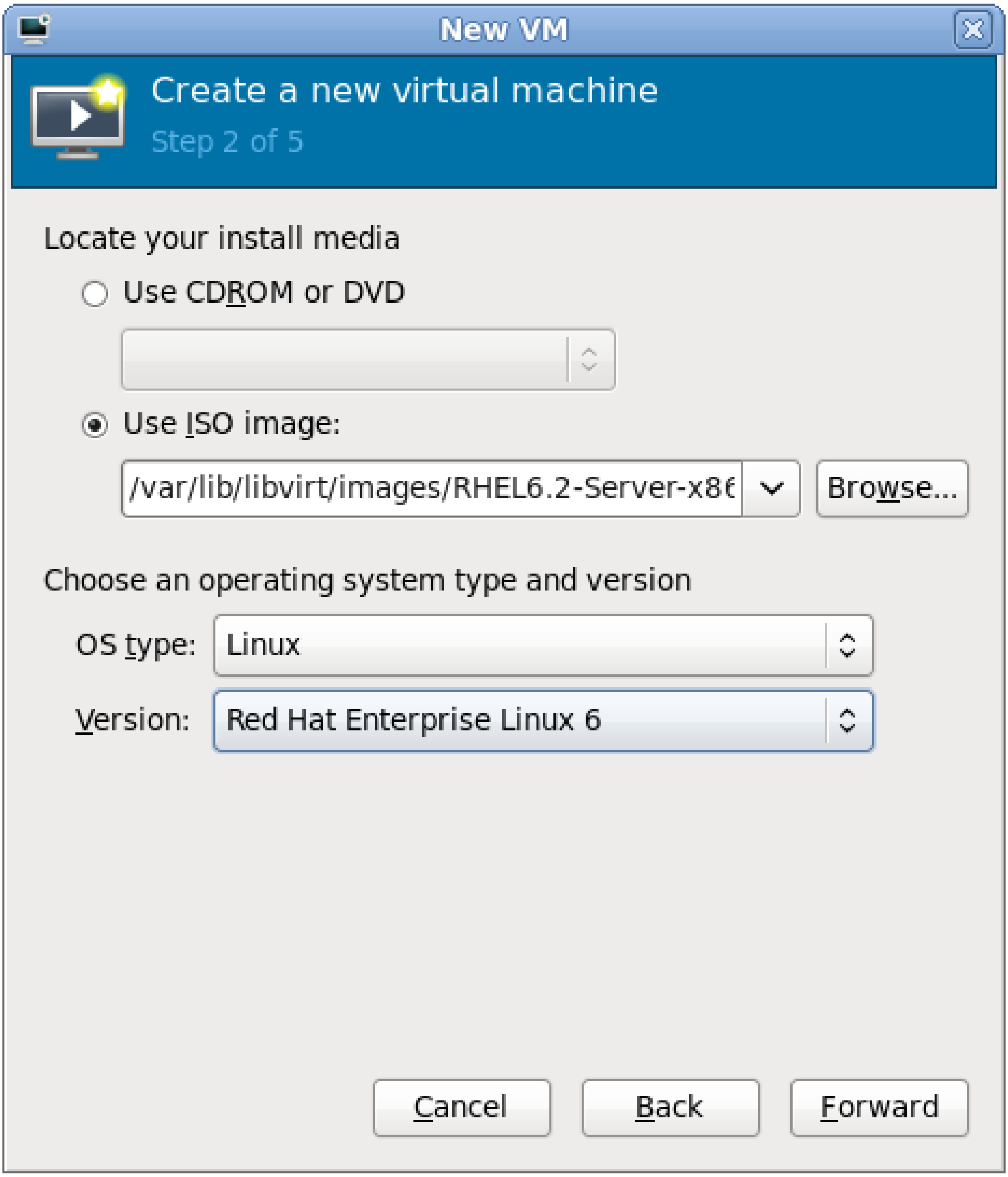


Figure 7.5. The New VM window - Step 2

Press **Forward** to continue.

5. **Set RAM and virtual CPUs**
   Choose appropriate values for the virtualized CPUs and RAM allocation. These values affect the host's and guest's performance. Memory and virtualized CPUs can be overcommitted. For more information on overcommitting, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

Virtualized guests require sufficient physical memory (RAM) to run efficiently and effectively. Red Hat supports a minimum of 512MB of RAM for a virtualized guest. Red Hat recommends at least 1024MB of RAM for each logical core.

Assign sufficient virtual CPUs for the virtualized guest. If the guest runs a multithreaded application, assign the number of virtualized CPUs the guest will require to run efficiently.

You cannot assign more virtual CPUs than there are physical processors (or hyper-threads) available on the host system. The number of virtual CPUs available is noted in the **Up to *X* available** field.

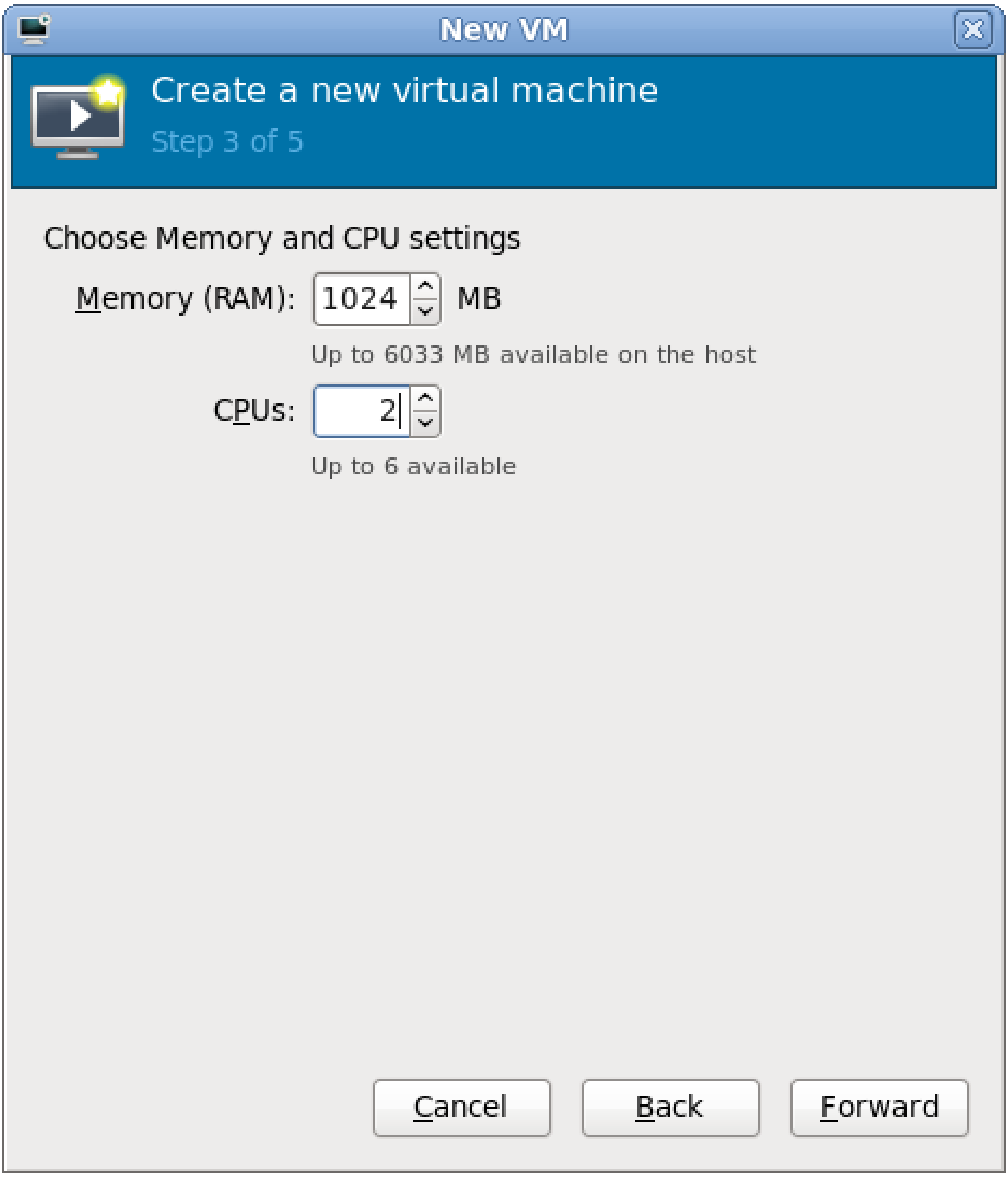


Figure 7.6. The new VM window - Step 3

Press **Forward** to continue.

6. **Storage**
   Enable and assign storage for the Red Hat Enterprise Linux 6 guest. Assign at least 5GB for a desktop installation or at least 1GB for a minimal installation.

Migration

Live and offline migrations require guests to be installed on shared network storage. For information on setting up shared storage for guests refer to the *Red Hat Enterprise Linux Virtualization Administration Guide*.

a. **With the default local storage**
   Select the **Create a disk image on the computer's hard drive** radio button to create a file-based image in the default storage pool, the **`/var/lib/libvirt/images/`** directory. Enter the size of the disk image to be created. If the **Allocate entire disk now** check box is selected, a disk image of the size specified will be created immediately. If not, the disk image will grow as it becomes filled.

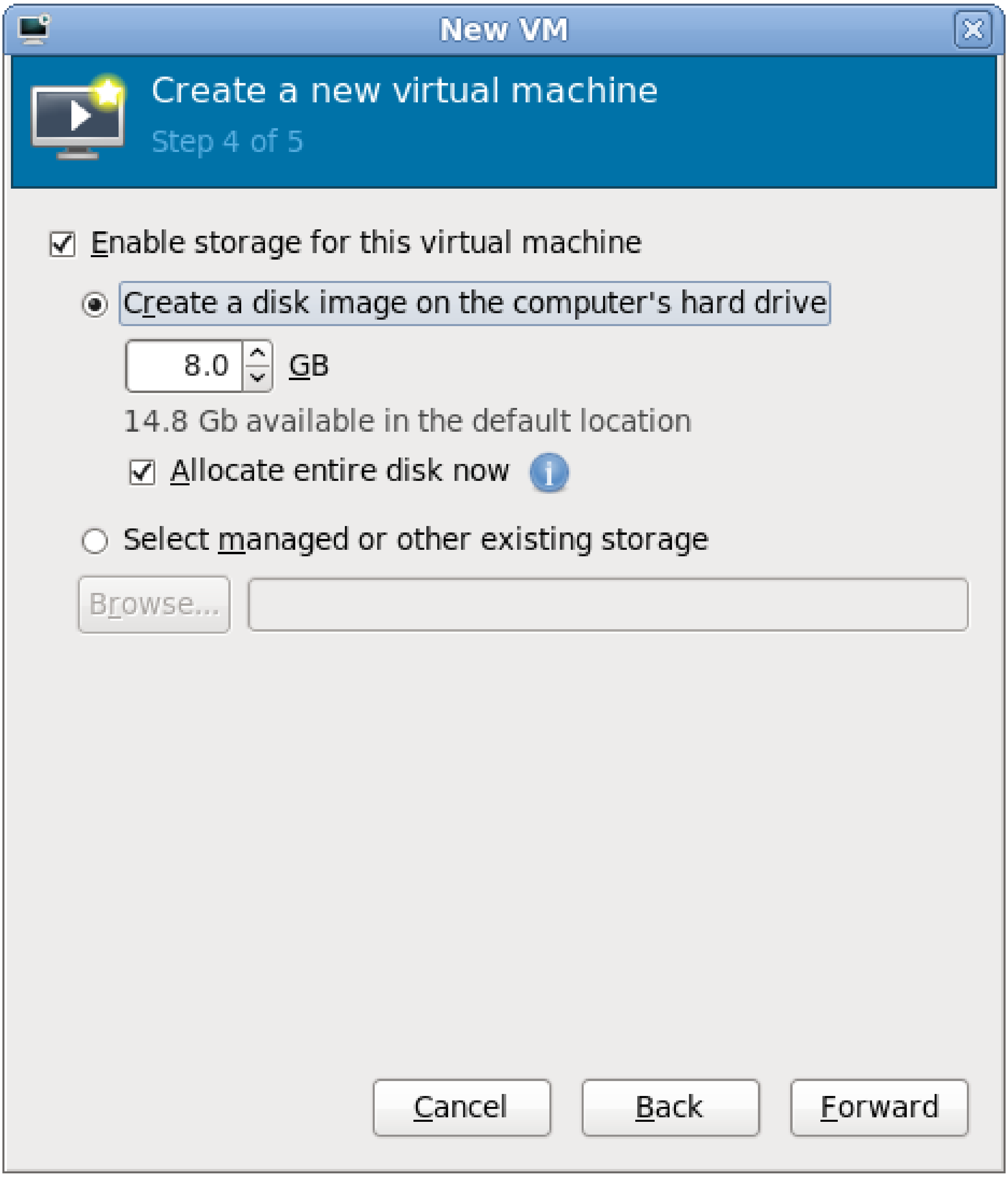


Figure 7.7. The New VM window - Step 4

Press **Forward** to create a disk image on the local hard drive. Alternatively, select **Select managed or other existing storage**, then select **Browse** to configure managed storage.

b. **With a storage pool**
If you selected **Select managed or other existing storage** in the previous step to use a storage pool and clicked **Browse**, the **Locate or create storage volume** window will appear.

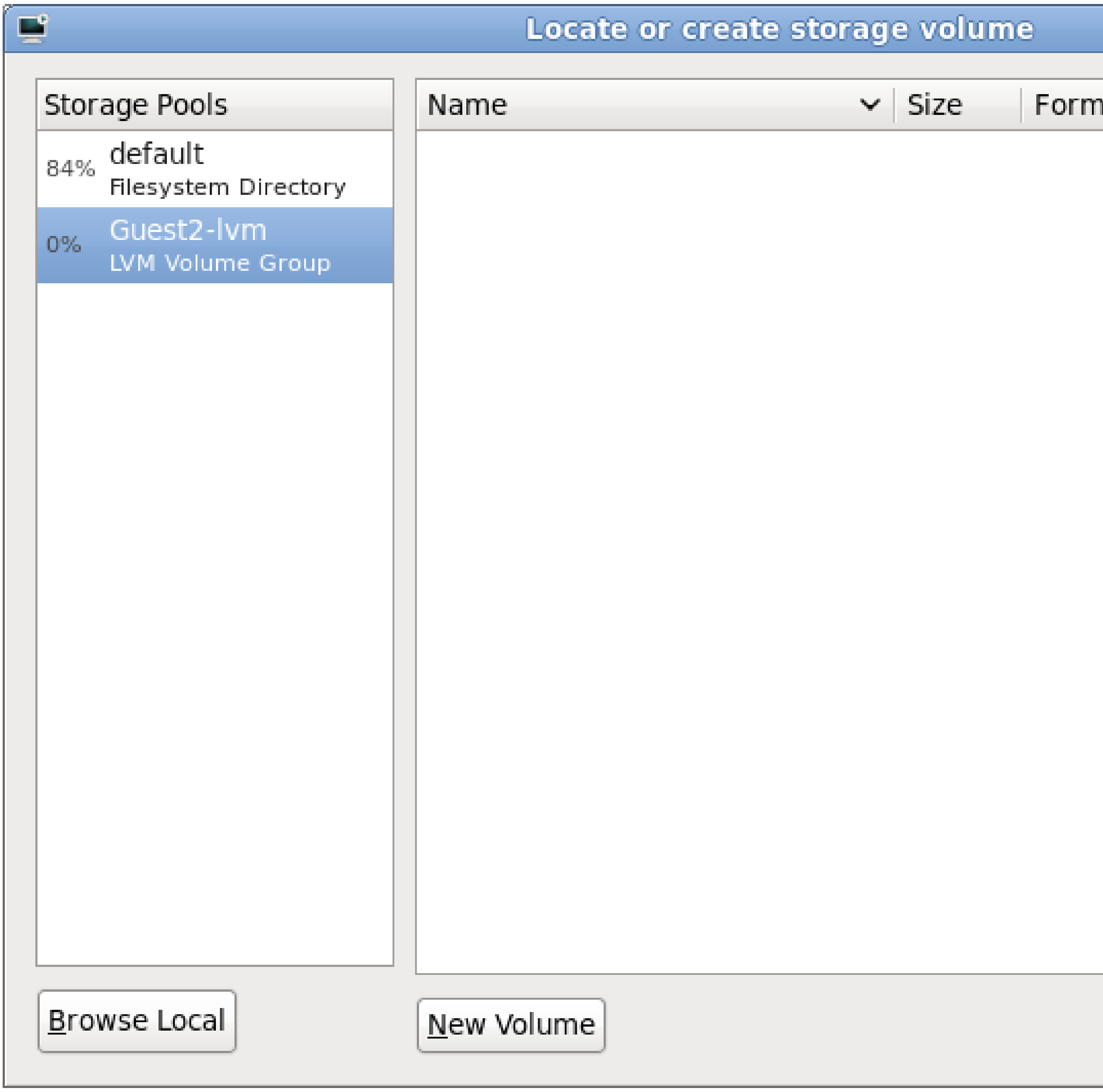


Figure 7.8. The Locate or create storage volume window

i. Select a storage pool from the **Storage Pools** list.

ii. Optional: Press the **New Volume** button to create a new storage volume. The **Add a Storage Volume** screen will appear. Enter the name of the new storage volume.

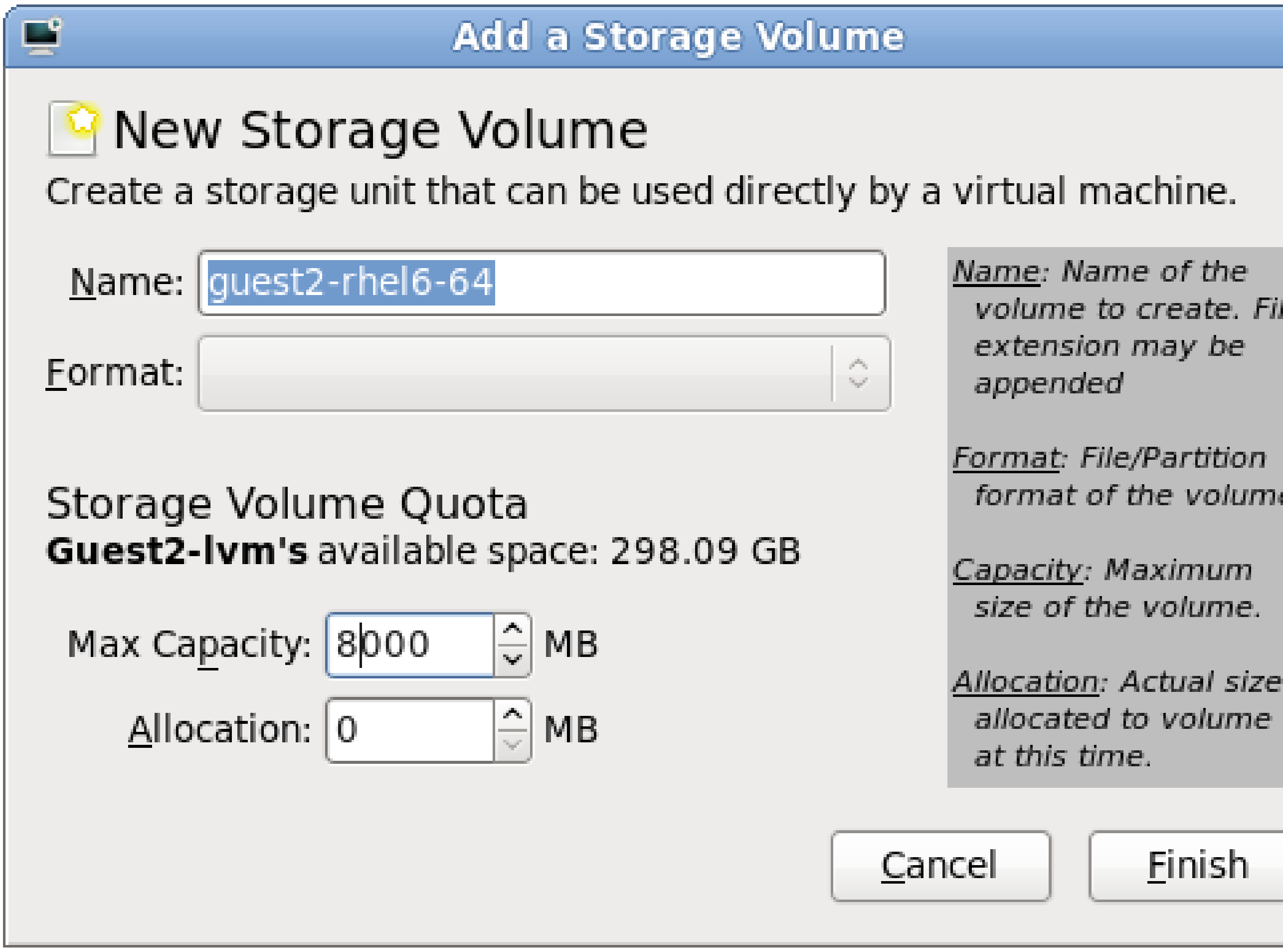


Figure 7.9. The Add a Storage Volume window

Press **Finish** to continue.

7. **Verify and finish**
Verify there were no errors made during the wizard and everything appears as expected.

Select the **Customize configuration before install** check box to change the guest's storage or network devices, to use the para-virtualized drivers or, to add additional devices.

Press the Advanced options down arrow to inspect and modify advanced options. For a standard Red Hat Enterprise Linux 6 none of these options require modification.

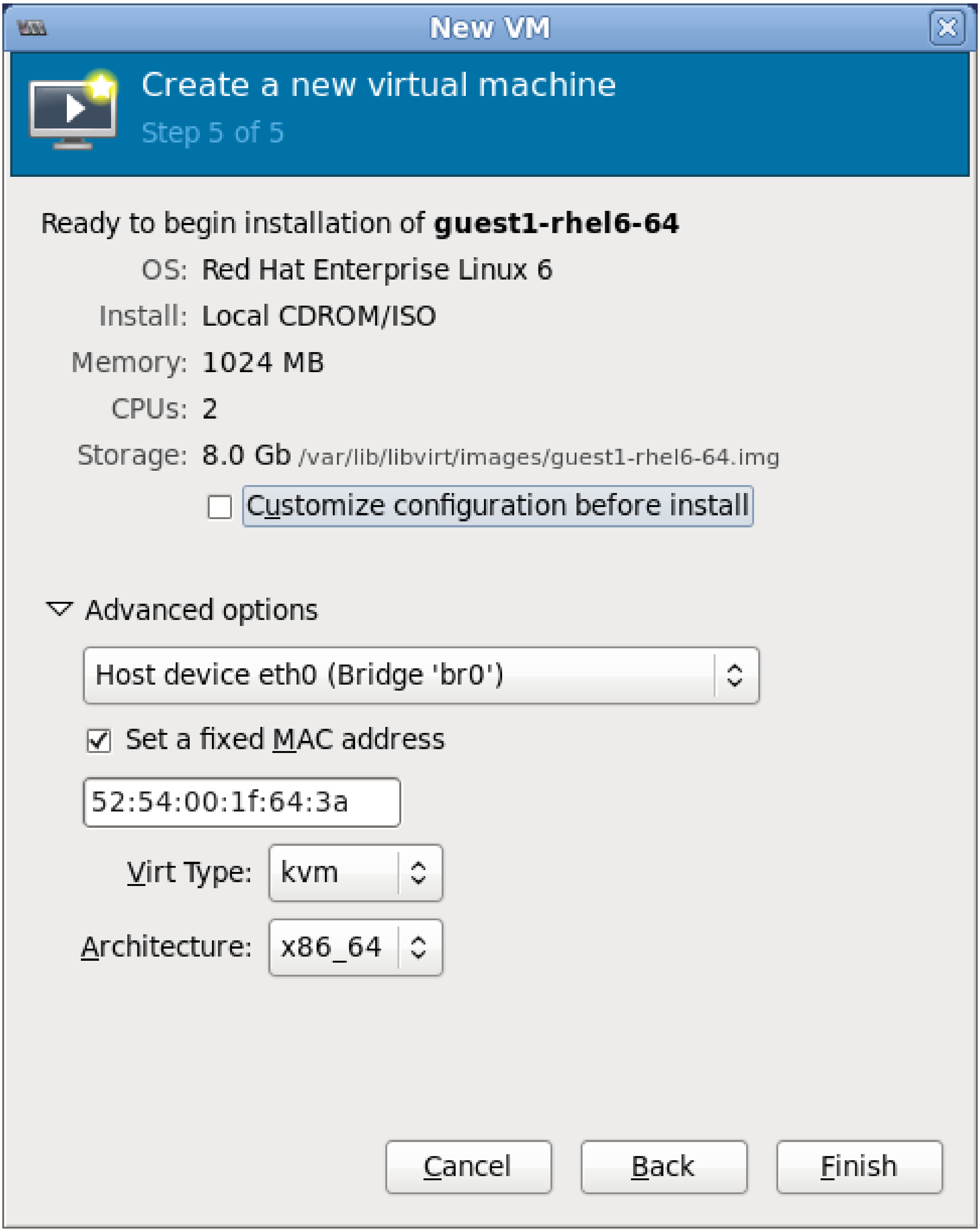


Figure 7.10. The New VM window - local storage

Figure 7.11. The New VM window - storage volume

Press **Finish** to continue into the Red Hat Enterprise Linux installation sequence. For more information on installing Red Hat Enterprise Linux 6 refer to the Red Hat Enterprise Linux 6 *Installation Guide*.

A Red Hat Enterprise Linux 6 guest is now created from a an ISO installation disc image.

7.2. Creating a Red Hat Enterprise Linux 6 guest with a network installation tree

Procedure 7.2. Creating a Red Hat Enterprise Linux 6 guest with virt-manager

1. **Optional: Preparation**
   Prepare the storage environment for the virtualized guest. For more information on preparing storage, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

   > **Note**
   >
   > Various storage types may be used for storing virtualized guests. However, for a guest to be able to use migration features the guest must be created on networked storage.

   Red Hat Enterprise Linux 6 requires at least 1GB of storage space. However, Red Hat recommends at least 5GB of storage space for a Red Hat Enterprise Linux 6 installation and for the procedures in this guide.

2. **Open virt-manager and start the wizard**
   Open virt-manager by executing the `virt-manager` command as root or opening **Applications** -> **System Tools** -> **Virtual Machine Manager**.

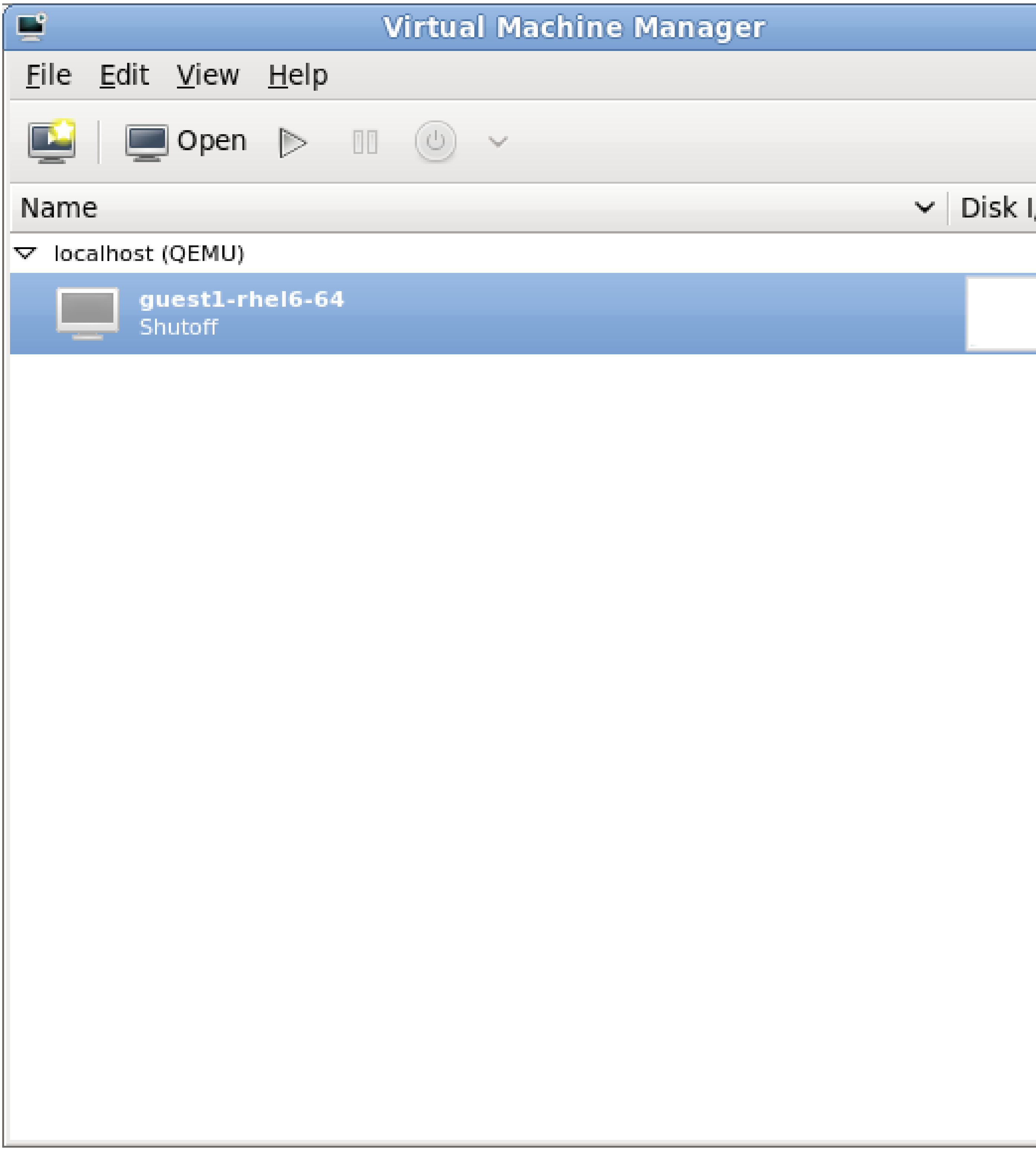


Figure 7.12. The main virt-manager window

Press the **create new virtualized guest button** (see figure *Figure 7.13, “The create new virtualized guest button”*) to start the new virtualized guest wizard.

Figure 7.13. The create new virtualized guest button

The **Create a new virtual machine** window opens.

3. **Name the virtualized guest**
   Guest names can contain letters, numbers and the following characters: '_', '**.**' and '**-**'. Guest names must be unique for migration.

   Choose the installation method from the list of radio buttons.

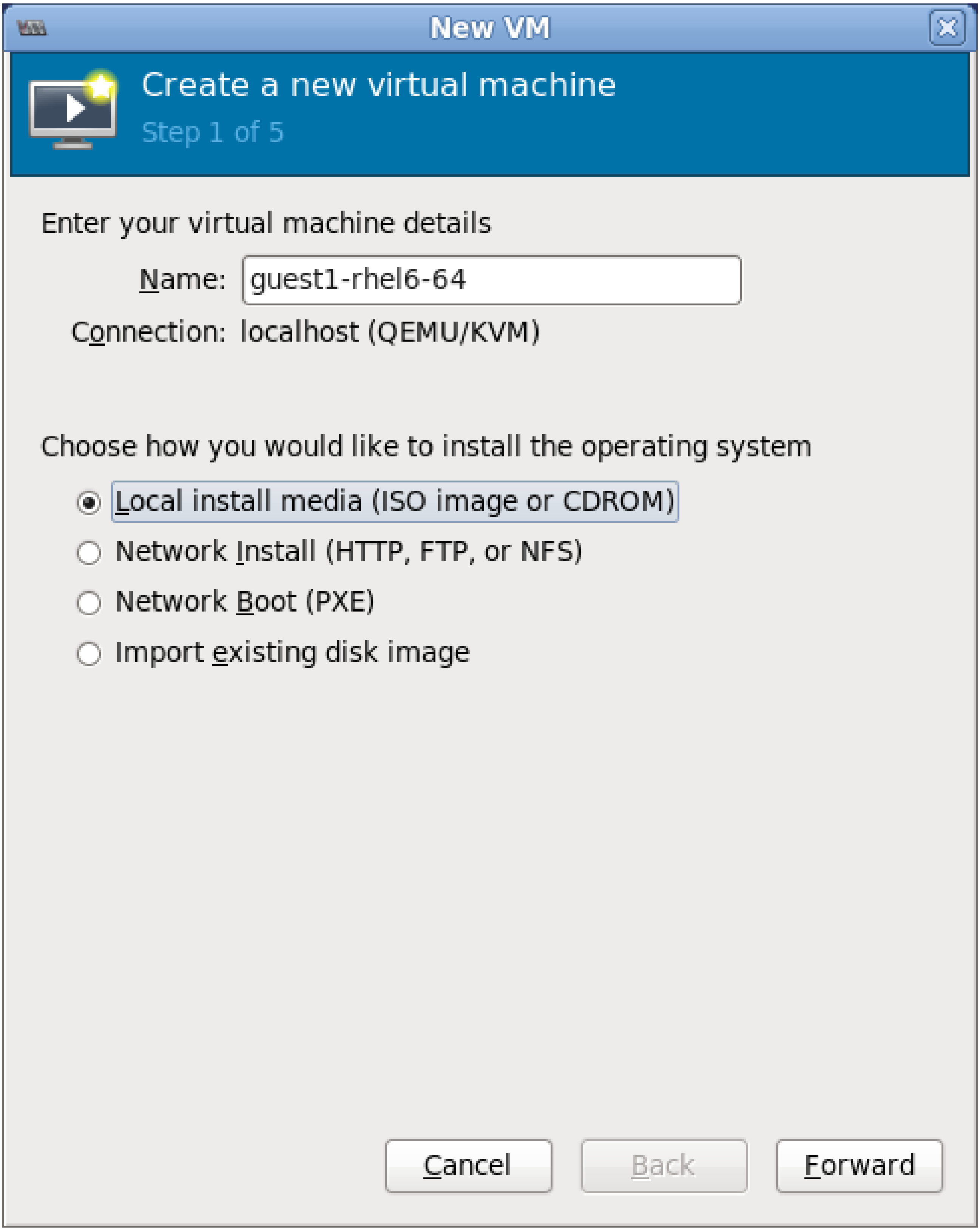


Figure 7.14. The New VM window - Step 1

Press **Forward** to continue.

4. Provide the installation URL, and the Kickstart URL and Kernel options if required.

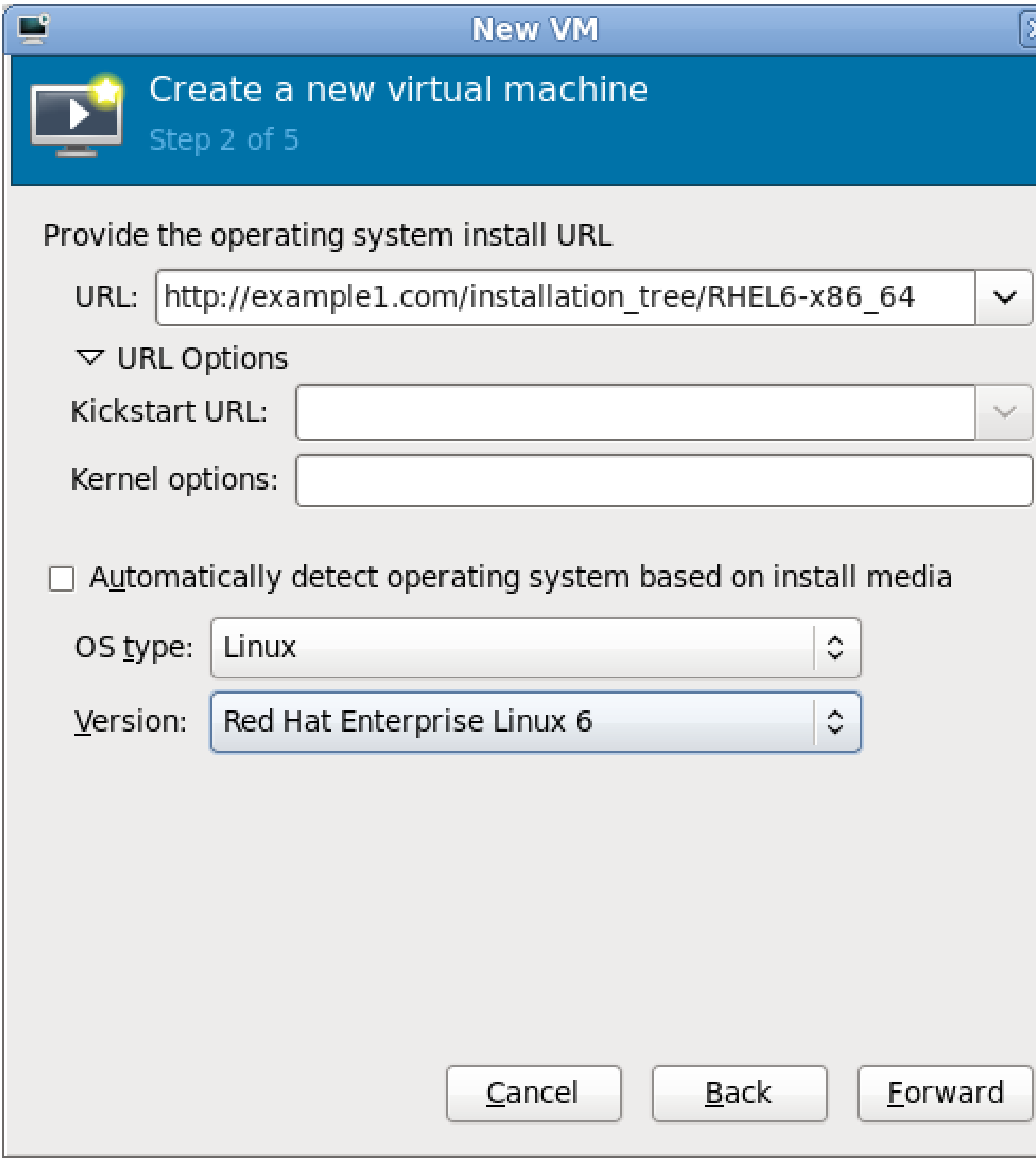


Figure 7.15. The New VM window - Step 2

Press **Forward** to continue.

5. The remaining steps are the same as the ISO installation procedure. Continue from *Step 5* of the ISO installation procedure.

7.3. Creating a Red Hat Enterprise Linux 6 guest with PXE

Procedure 7.3. Creating a Red Hat Enterprise Linux 6 guest with virt-manager

1. **Optional: Preparation**
   Prepare the storage environment for the virtualized guest. For more information on preparing storage, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

   Note

   Various storage types may be used for storing virtualized guests. However, for a guest to be able to use migration features the guest must be created on networked storage.

   Red Hat Enterprise Linux 6 requires at least 1GB of storage space. However, Red Hat recommends at least 5GB of storage space for a Red Hat Enterprise Linux 6 installation and for the procedures in this guide.

2. **Open virt-manager and start the wizard**
   Open virt-manager by executing the **`virt-manager`** command as root or opening **Applications** -> **System Tools** -> **Virtual Machine Manager**.

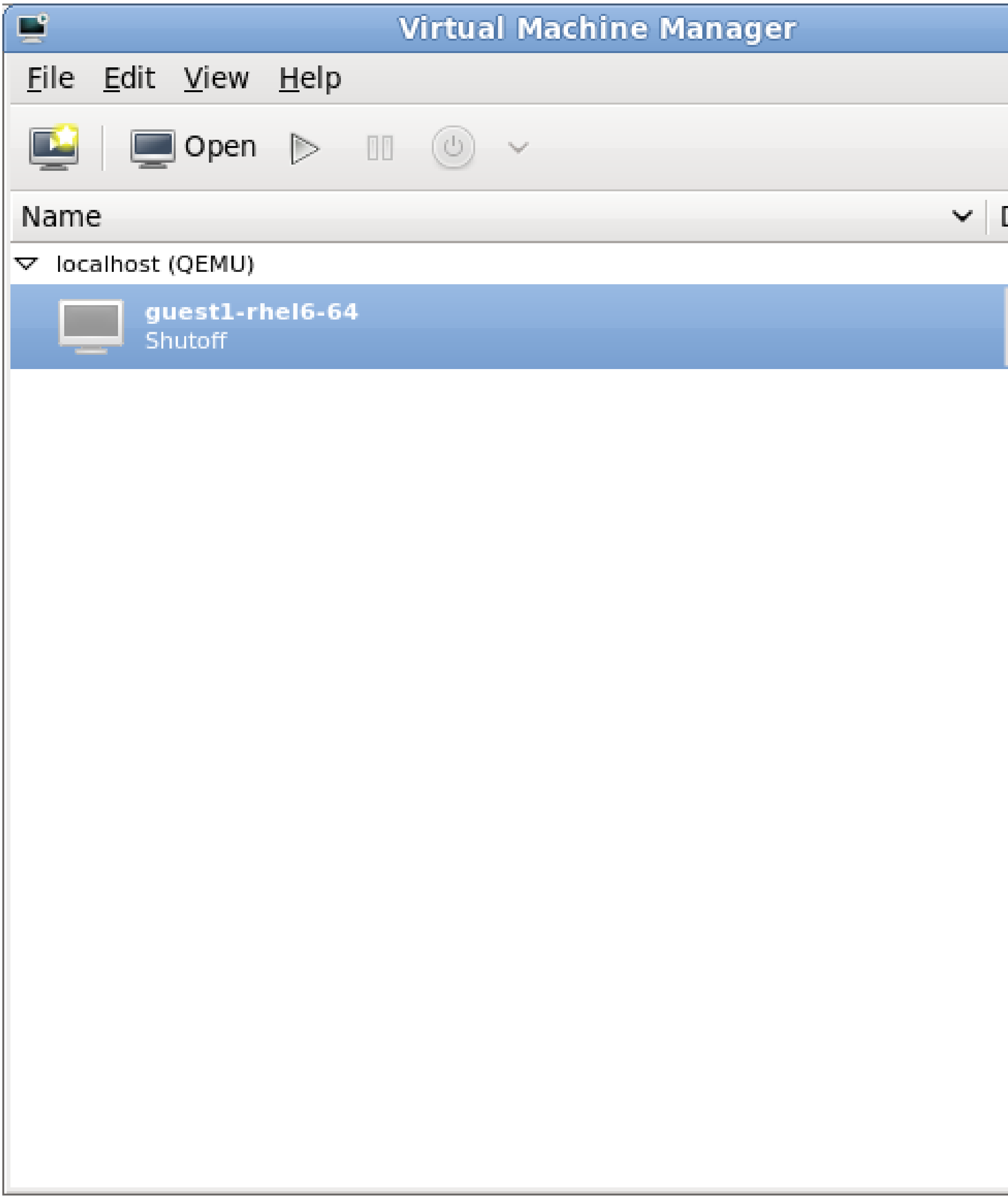


Figure 7.16. The main virt-manager window

Press the **create new virtualized guest button** (see figure *Figure 7.17, “The create new virtualized guest button”*) to start the new virtualized guest wizard.

Figure 7.17. The create new virtualized guest button

The **New VM** window opens.

3. **Name the virtualized guest**
   Guest names can contain letters, numbers and the following characters: '_', '**.**' and '**-**'. Guest names must be unique for migration.

   Choose the installation method from the list of radio buttons.

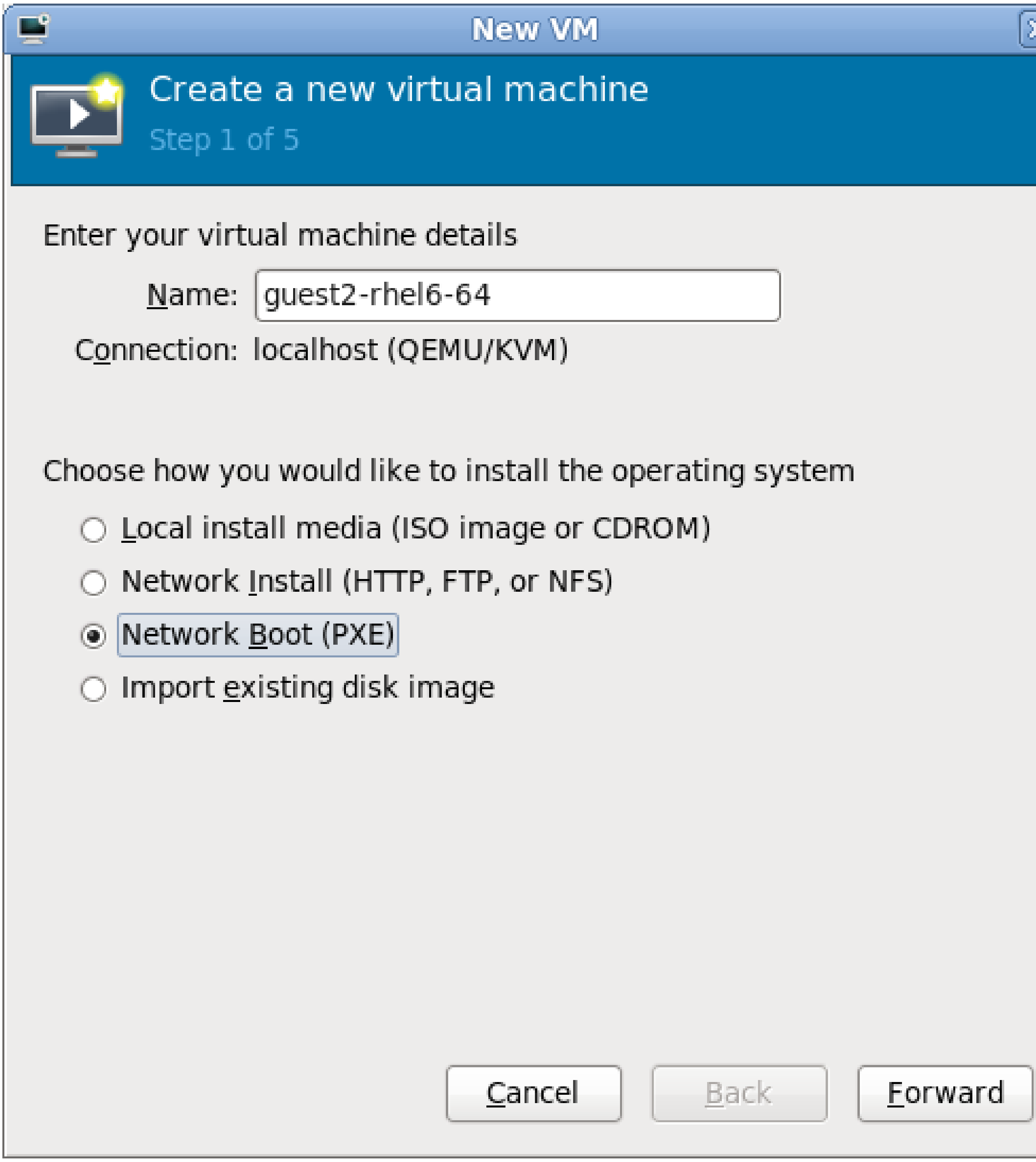


Figure 7.18. The New VM window - Step 1

Press **Forward** to continue.

4. The remaining steps are the same as the ISO installation procedure. Continue from *Step 5* of the ISO installation procedure. From this point, the only difference in this PXE procedure is on the final **New VM** screen, which shows the **Install: PXE Install** field.

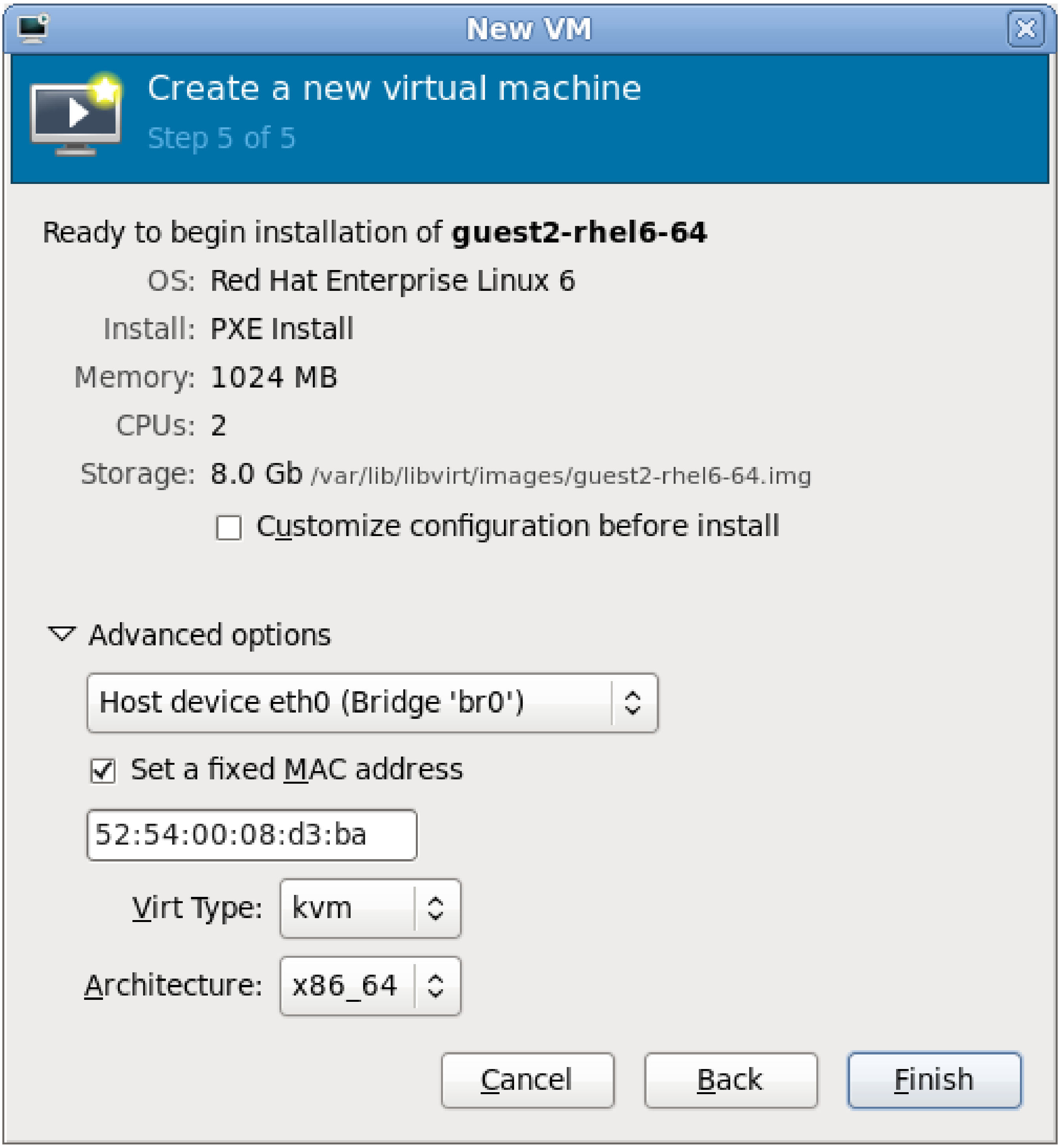


Figure 7.19. The New VM window - Step 5 - PXE Install

Installing Red Hat Enterprise Linux 6 as a Xen para-virtualized guest on Red Hat Enterprise Linux 5

This section describes how to install Red Hat Enterprise Linux 6 as a Xen para-virtualized guest on Red Hat Enterprise Linux 5. Para-virtualization is only available for Red Hat Enterprise Linux 5 hosts. Red Hat Enterprise Linux 6 uses the PV-opts features of the Linux kernel to appear as a compatible Xen para-virtualized guest.

Important note on para-virtualization

Para-virtualization only works with the Xen hypervisor. Para-virtualization does not work with the KVM hypervisor. This procedure is for Red Hat Enterprise Linux 5.4 or newer.

8.1. Using virt-install

This section covers creating a Xen para-virtualized Red Hat Enterprise Linux 6 guest on a Red Hat Enterprise Linux 5 host using the **`virt-install`** command. For instructions on **`virt-manager`**, refer to the procedure in *Section 8.2, “Using virt-manager”*.

This method installs Red Hat Enterprise Linux 6 from a remote server hosting the network installation tree. The installation instructions presented in this section are similar to installing from the minimal installation live CD-ROM.

Automating with virt-install

Guests can be created with the command line **`virt-install`** tool. The name of the guest in the example is *`rhel6pv-64`*, the disk image file is *`rhel6pv-64.img`* and a local mirror of the Red Hat Enterprise Linux 6 installation tree is *`http://example.com/installation_tree/RHEL6-x86/`*. Replace those values with values for your system and network.

```
# virt-install --name=rhel6pv-64 \
--disk path=/var/lib/xen/images/rhel6pv-64.img,size=6,sparse=false \
--vnc --paravirt --vcpus=2 --ram=2048 \
--location=http://example.com/installation_tree/RHEL6-x86/
```

Red Hat Enterprise Linux can be installed without a graphical interface or manual input. Use a Kickstart file to automate the installation process. This example extends the previous example with a Kickstart file, located at **`http://example.com/kickstart/ks.cfg`**, to fully automate the installation.

```
# virt-install --name=rhel6pv-64 \
--disk path=/var/lib/xen/images/rhel6pv-64.img,size=6,sparse=false \
--vnc --paravirt --vcpus=2 --ram=2048 \
--location=http://example.com/installation_tree/RHEL6-x86/ \
-x "ks=http://example.com/kickstart/ks.cfg"
```

The graphical console opens showing the initial boot phase of the guest.

After your guest has completed its initial boot, the standard installation process for Red Hat Enterprise Linux 6 starts.

Refer to the Red Hat Enterprise Linux 6 Installation Guide for more information on installing Red Hat Enterprise Linux 6.

8.2. Using virt-manager

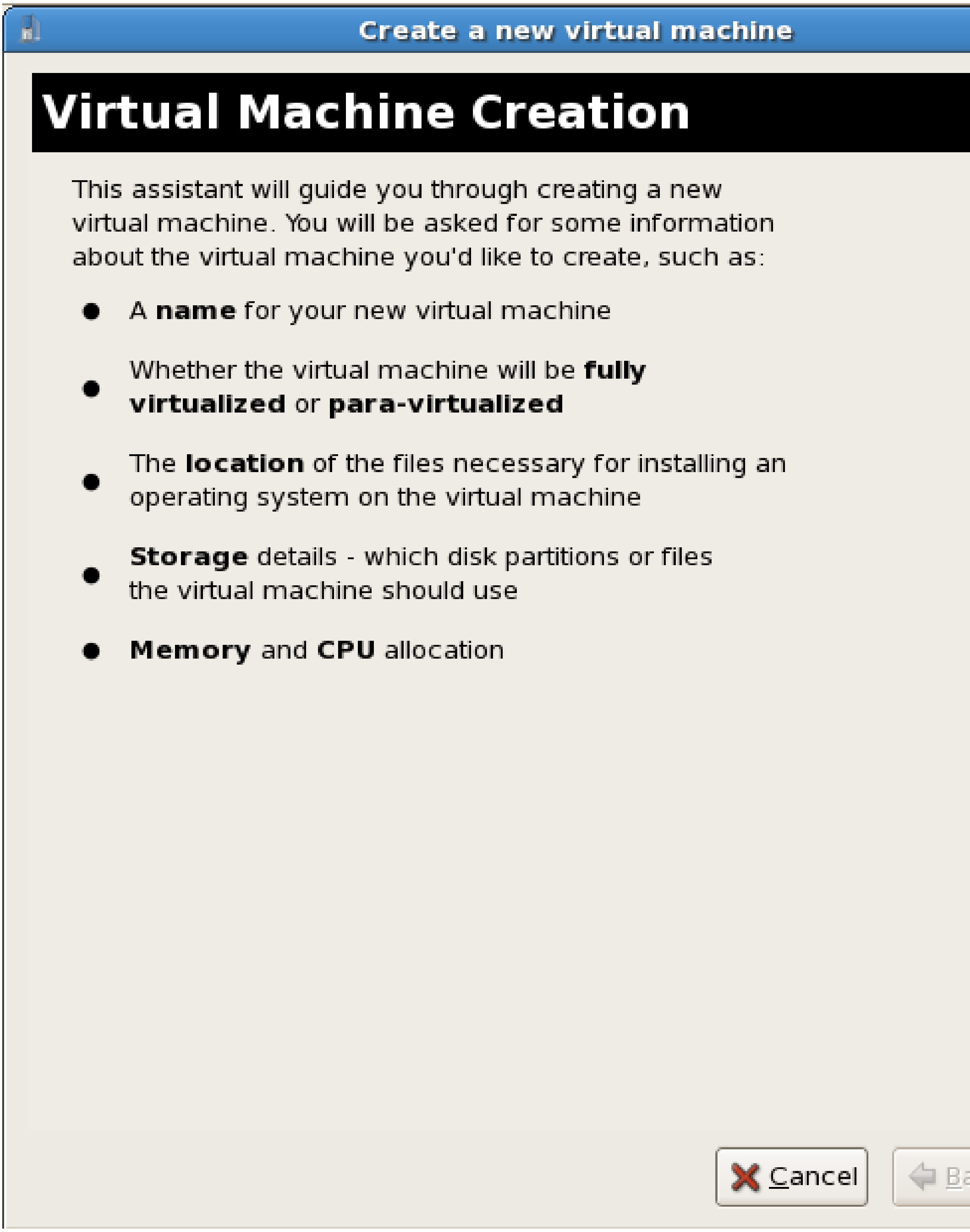
Procedure 8.1. Creating a Xen para-virtualized Red Hat Enterprise Linux 6 guest with virt-manager

1. **Open virt-manager**
   Start **`virt-manager`**. Launch the **Virtual Machine Manager** application from the **Applications** menu and **System Tools** submenu. Alternatively, run the **`virt-manager`** command as root.

2. **Select the hypervisor**
   Select the Xen hypervisor connection. Note that presently the KVM hypervisor is named **`qemu`**.

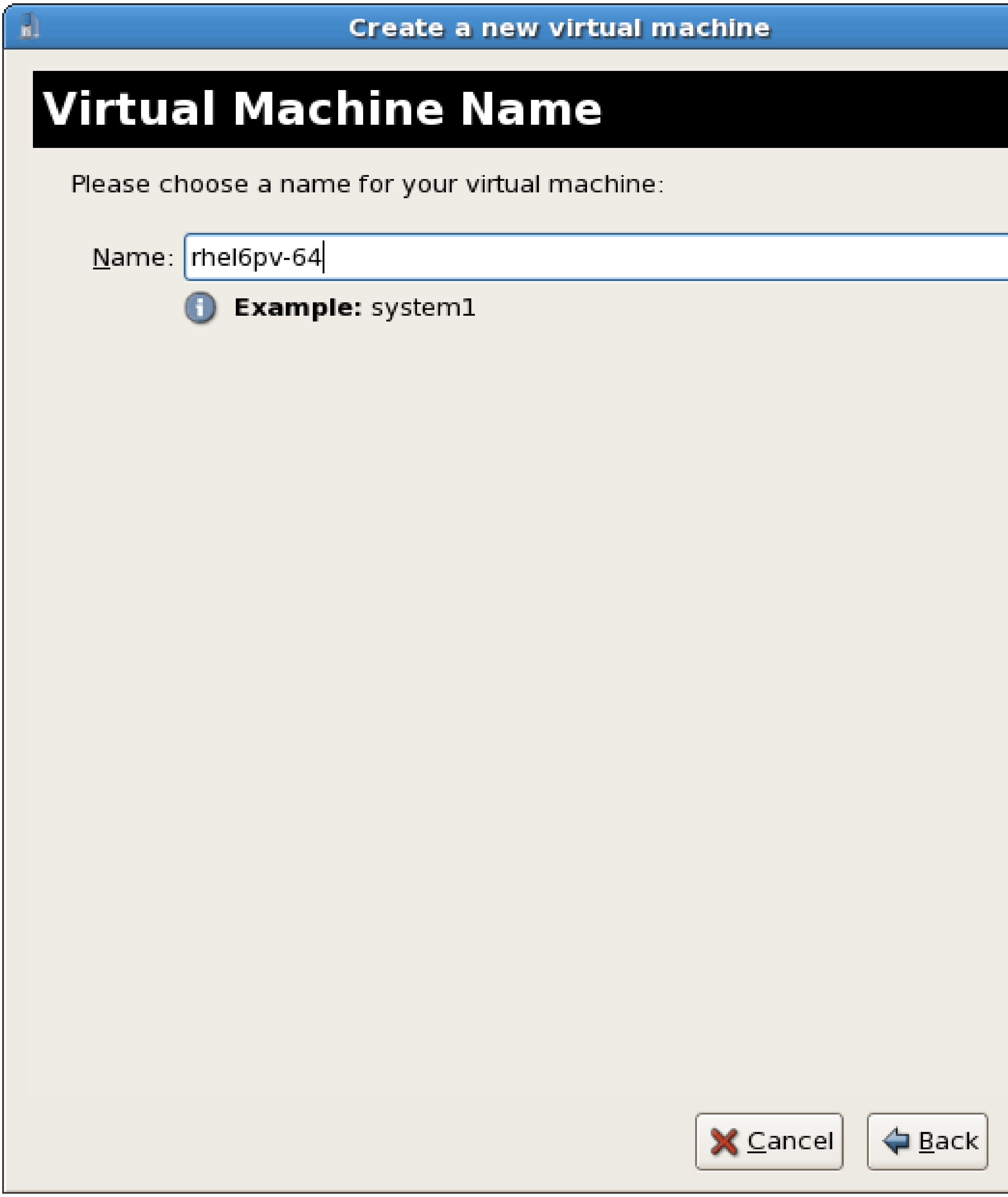
   Connect to a hypervisor if you have not already done so. Open the **File** menu and select the **Add Connection...** option. Refer to the *Red Hat Enterprise Linux Virtualization Administration Guide* for further details about adding a remote connection.

3. **Start the new virtual machine wizard**
   Once a hypervisor connection is selected the **New** button becomes available. Pressing the **New** button starts the virtual machine creation wizard, which explains the steps that follow.



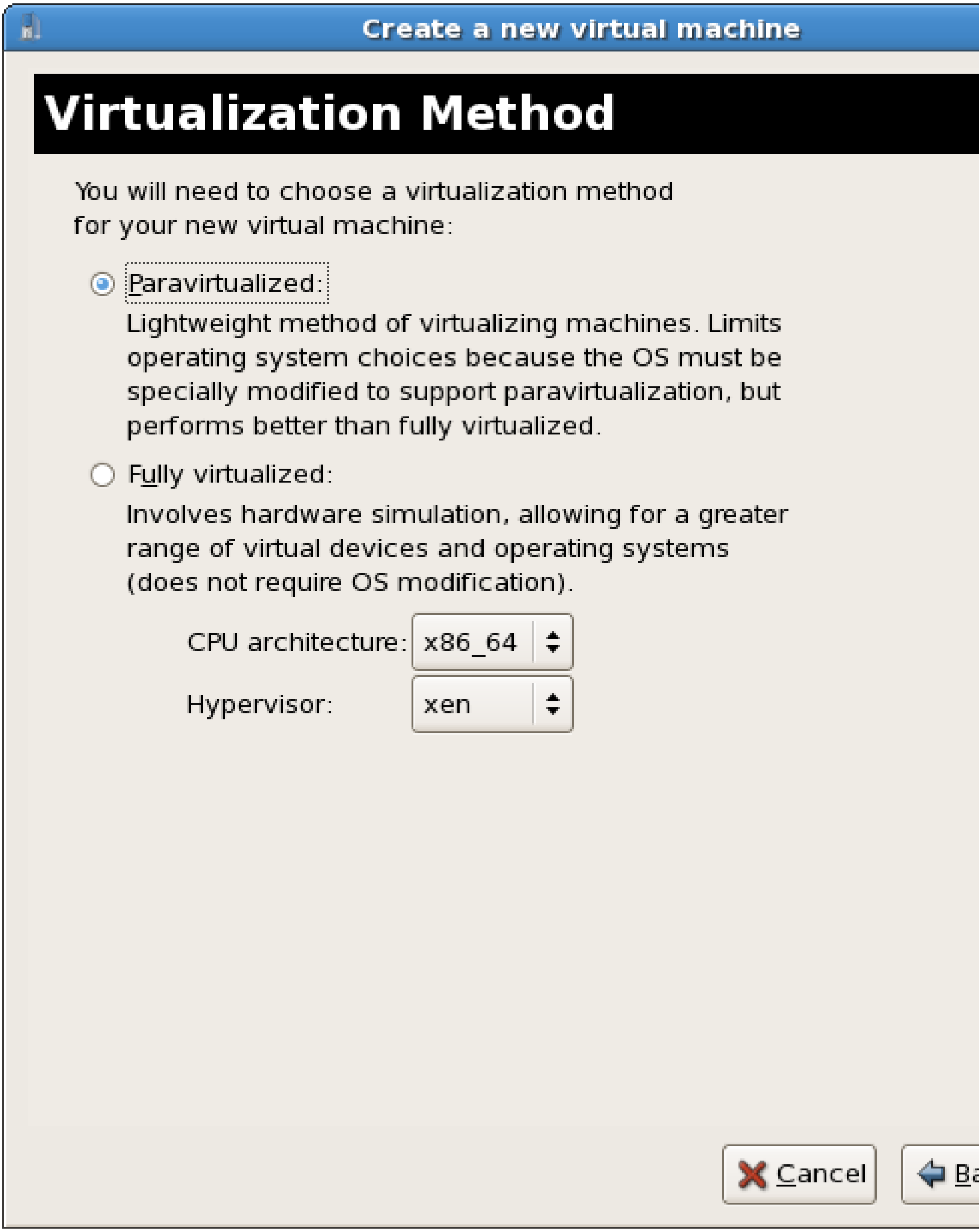
Press **Forward** to continue.

4. **Name the virtual machine**
   Provide a name for your virtualized guest. The following punctuation and whitespace characters are permitted: '_', '**.**' and '**-**' characters.



Press **Forward** to continue.

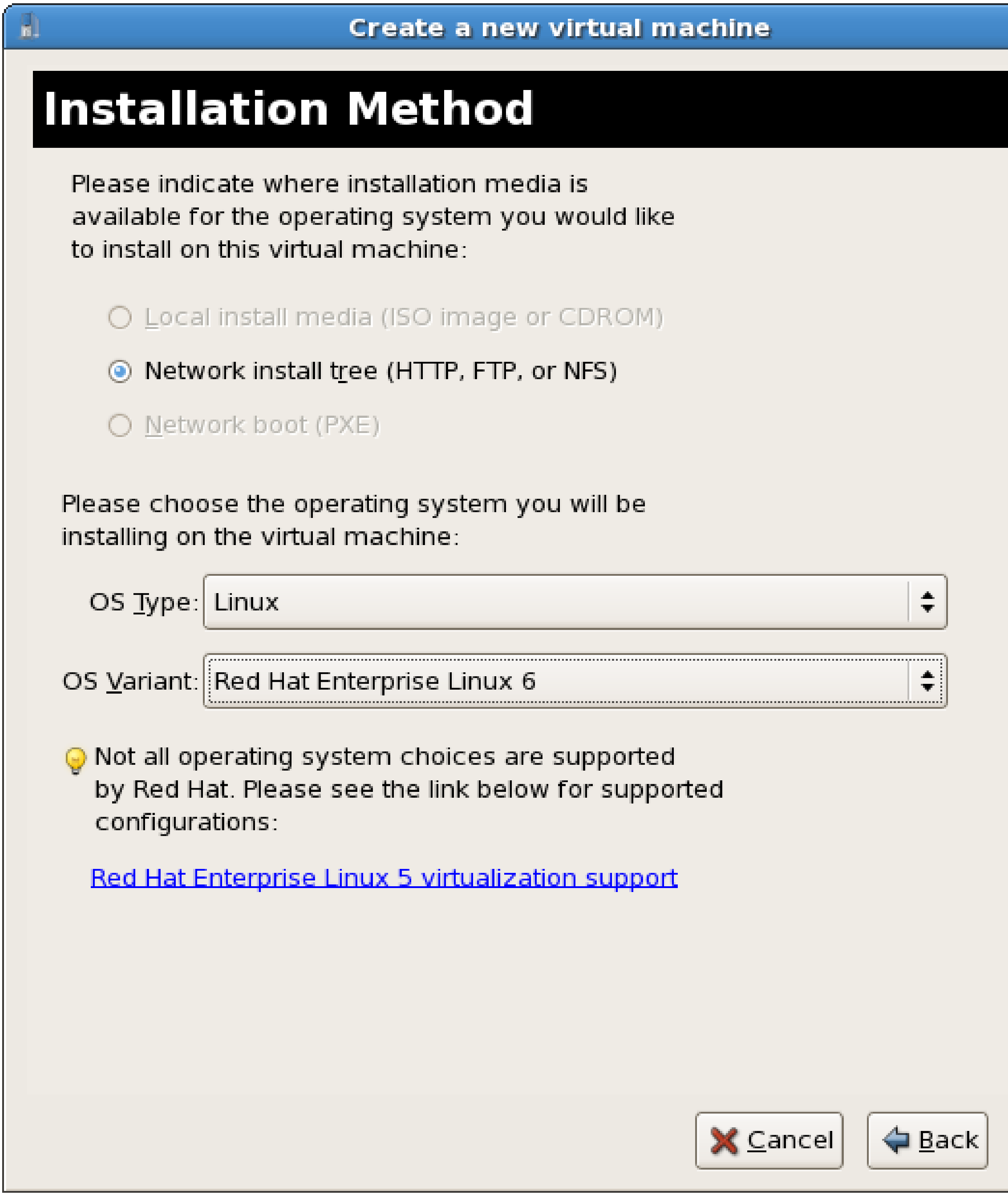
5. **Select the virtualization method**
   Select **Paravirtualized** as the virtualization method, as shown:



Press **Forward** to continue.

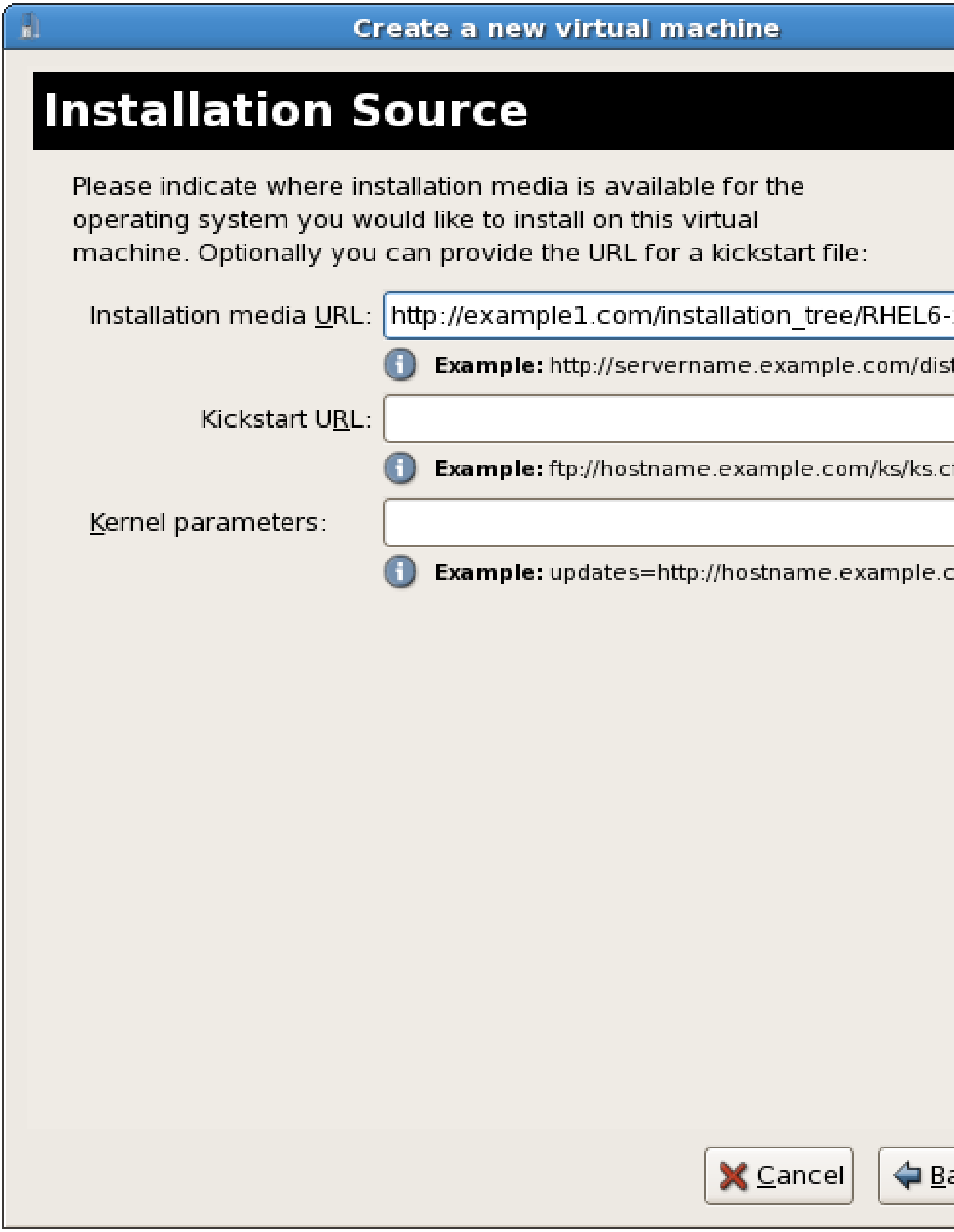
6. **Select the installation method and type**
   Select the **Network install tree** method, as this is the only method that makes sense for para-virtualized guests.

Set **OS Type** to **Linux** and **OS Variant** to **Red Hat Enterprise Linux 6**, as shown in the screenshot.



Press **Forward** to continue.

7. **Locate installation media**
   Enter the location of the installation tree.



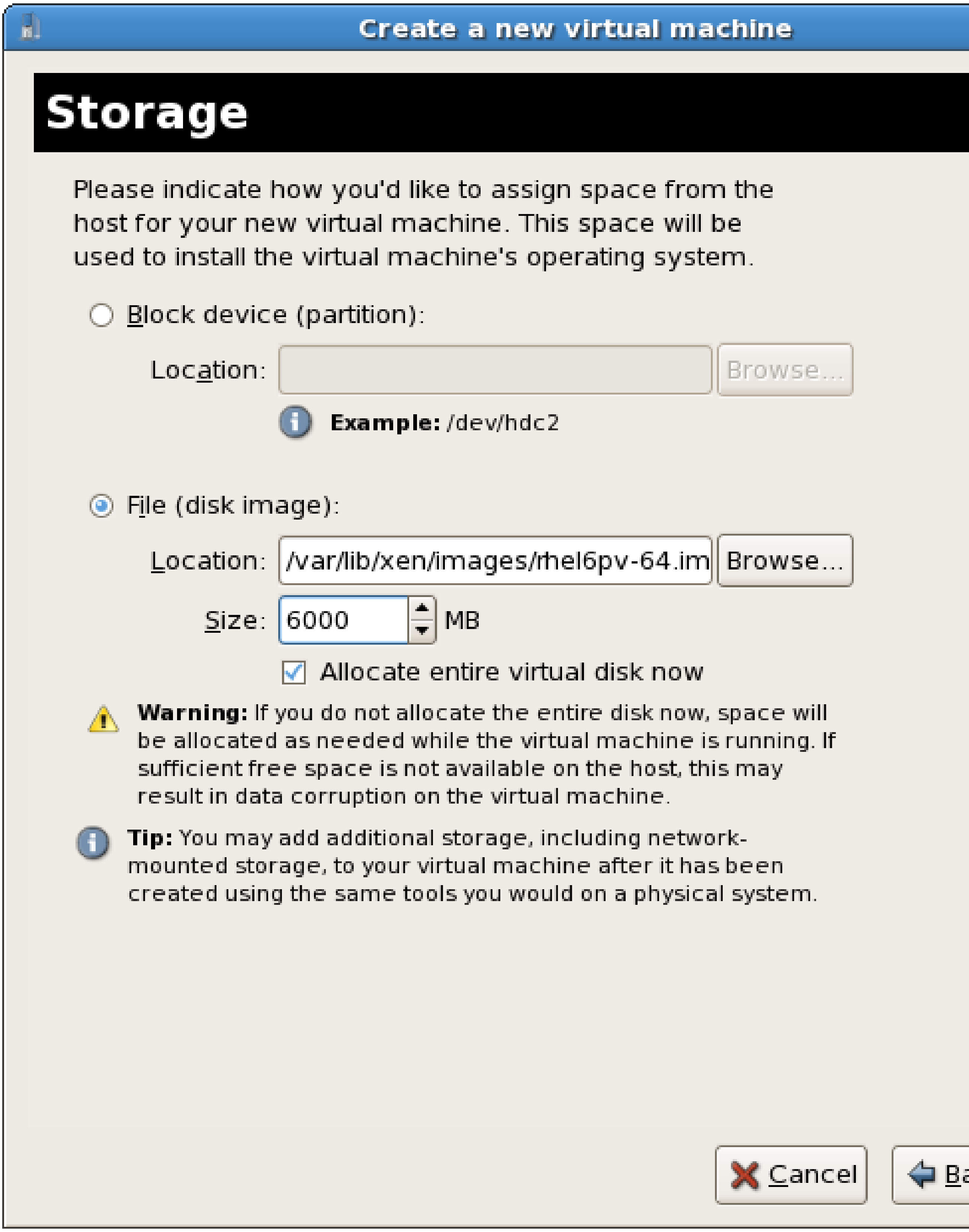
   Press **Forward** to continue.

8. **Storage setup**

   Image files and SELinux

   Xen file-based images should be stored in the `/var/lib/xen/images/` directory. Any other location may require additional configuration for SELinux. Refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* for more information on configuring SELinux.

   Assign a physical storage device (**Block device**) or a file-based image (**File**). Assign sufficient space for your virtualized guest and any applications the guest requires.



Press **Forward** to continue.

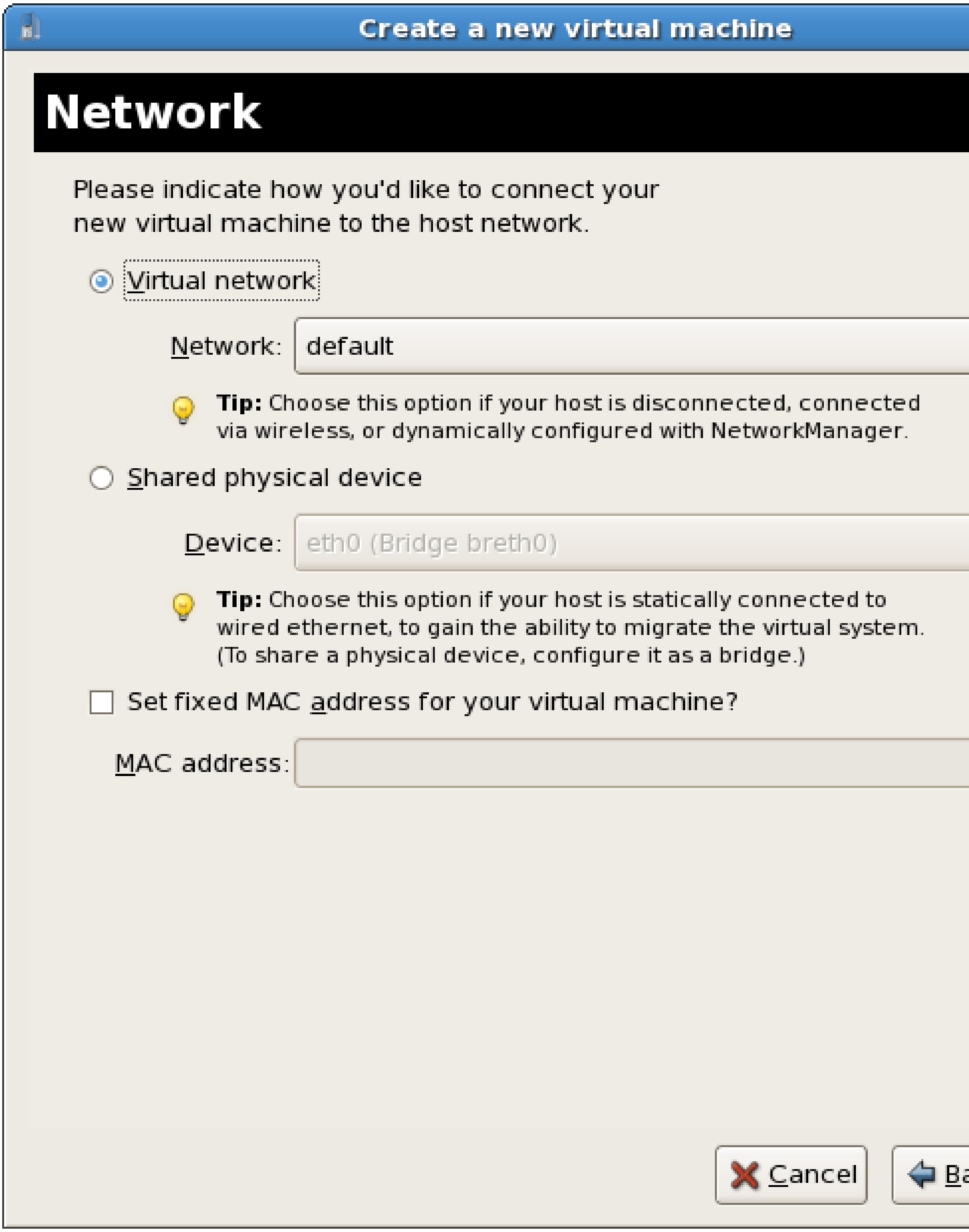
Migration

Live and offline migrations require guests to be installed on shared network storage. For information on setting up shared storage for guests refer to the *Virtualization Administration Guide chapter on Storage Pools.*[1]

9. **Network setup**
   Select either **Virtual network** or **Shared physical device**.

   The virtual network option uses Network Address Translation (NAT) to share the default network device with the virtualized guest.

   The shared physical device option uses a network bridge to give the virtualized guest full access to a network device.

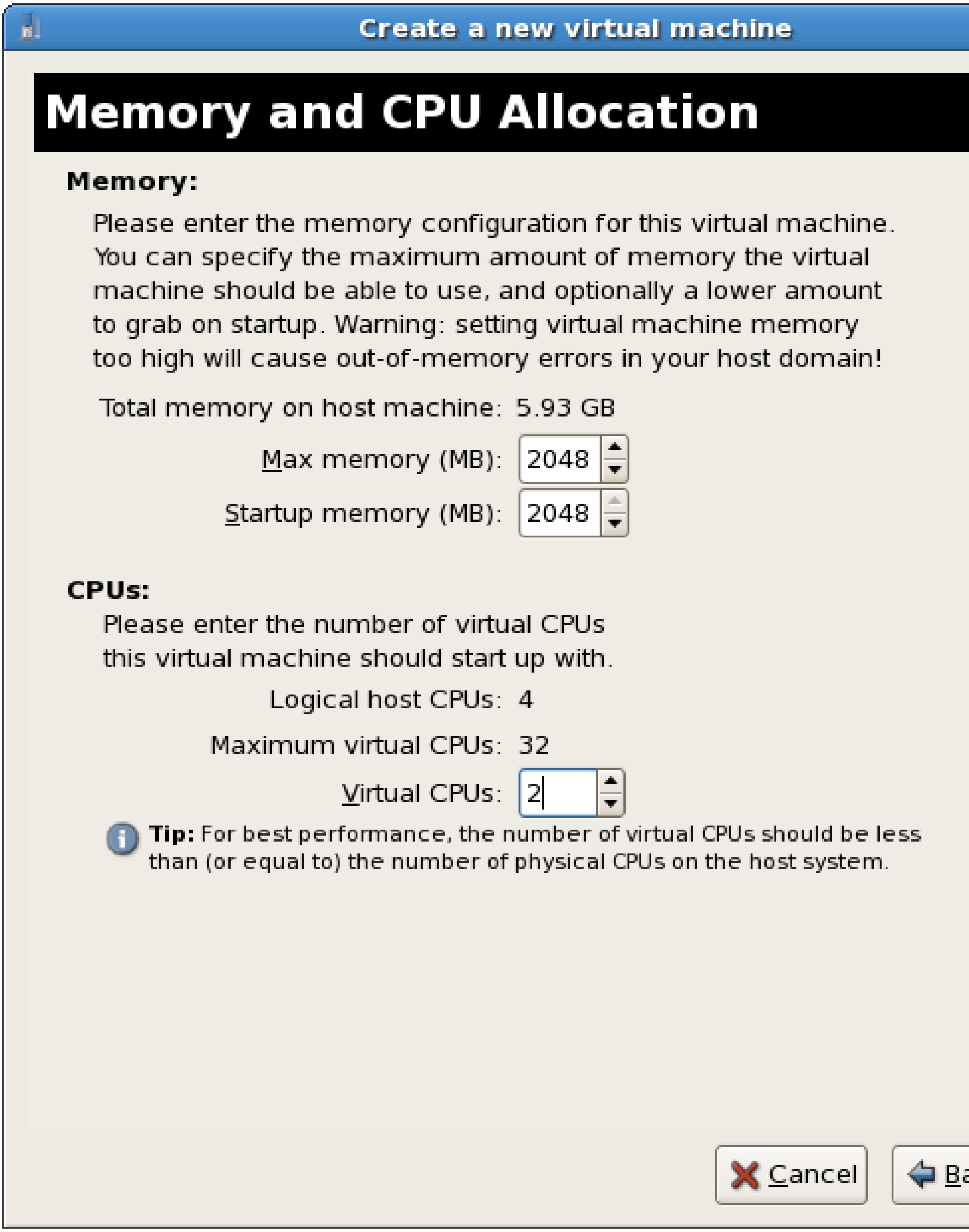


Press **Forward** to continue.

10. **Memory and CPU allocation**
    The **Memory and CPU Allocation** window displays. Choose appropriate values for the virtualized CPUs and RAM allocation. These values affect the host's and guest's performance.

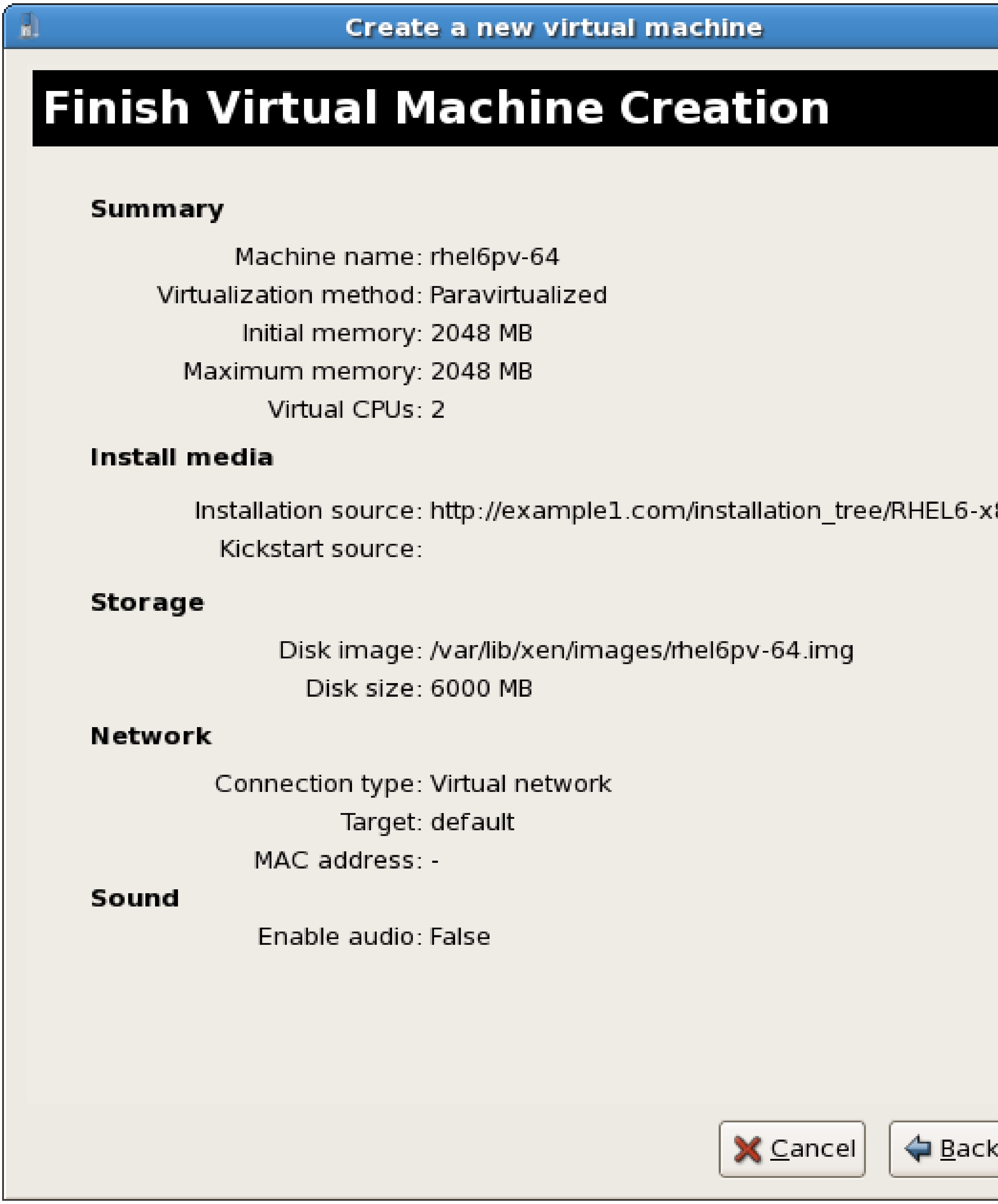
Virtualized guests require sufficient physical memory (RAM) to run efficiently and effectively. Choose a memory value which suits your guest operating system and application requirements. Remember, Xen guests use physical RAM. Running too many guests or leaving insufficient memory for the host system results in significant usage of virtual memory and swapping. Virtual memory is significantly slower which causes degraded system performance and responsiveness. Ensure you allocate sufficient memory for all guests and the host to operate effectively.

Assign sufficient virtual CPUs for the virtualized guest. If the guest runs a multithreaded application, assign the number of virtualized CPUs the guest will require to run efficiently. Do not assign more virtual CPUs than there are physical processors (or hyper-threads) available on the host system. It is possible to over allocate virtual processors, however, over allocating VCPUs has a significant, negative effect on Xen guest and host performance.



Press **Forward** to continue.

11. **Verify and start guest installation**
    Verify the configuration.



Press **Finish** to start the guest installation procedure.

12. **Installing Red Hat Enterprise Linux**
    Complete the Red Hat Enterprise Linux installation sequence. The installation sequence is covered by the Red Hat Enterprise Linux 6 *Installation Guide*. Refer to Red Hat Documentation[2] for the Red Hat Enterprise Linux 6 *Installation Guide*.

Installing a fully-virtualized Windows guest

This chapter describes how to create a fully virtualized Windows guest using the command-line (**`virt-install`**), launch the operating system's installer inside the guest, and access the installer through **`virt-viewer`**.

To install a Windows operating system on the guest, use the **`virt-viewer`** tool. This tool allows you to display the graphical console of a virtual machine (via the VNC protocol). In doing so, **`virt-viewer`** allows you to install a fully virtualized guest's operating system through that operating system's installer (e.g. the Windows XP installer).

Installing a Windows operating system involves two major steps:

1. Creating the guest (using either **`virt-install`** or **`virt-manager`**)

2. Installing the Windows operating system on the guest (through **`virt-viewer`**)

Note that this chapter does not describe how to install a Windows operating system on a fully-virtualized guest. Rather, it only covers how to create the guest and launch the installer within the guest. For information on how to install a Windows operating system, refer to the relevant Microsoft installation documentation.

9.1. Using virt-install to create a guest

The **`virt-install`** command allows you to create a fully-virtualized guest from a terminal, i.e. without a GUI.

Important

Before creating the guest, consider first if the guest needs to use KVM Windows para-virtualized drivers. If it does, keep in mind that you can do so *during* or *after* installing the Windows operating system on the guest. For more information about para-virtualized drivers, refer to *Chapter 10, KVM Para-virtualized Drivers*.

For instructions on how to install KVM para-virtualized drivers, refer to *Section 10.1, "Installing the KVM Windows para-virtualized drivers"*.

It is possible to create a fully-virtualized guest with only a single command. To do so, simply run the following program (replace the values accordingly):

```
# virt-install \
   --name=guest-name  \
   --network network=default \
   --disk path=path-to-disk,size=disk-size \
   --cdrom=path-to-install-disk \
   --vnc --ram=1024
```

The ***`path-to-disk`*** must be a device (for example, **`/dev/sda3`**) or image file (**`/var/lib/libvirt/images/`*`name`*`.img`**). It must also have enough free space to support the ***`disk-size`***.

> **Important**
>
> All image files are stored in **`/var/lib/libvirt/images/`** by default. Other directory locations for file-based images are possible, but may require SELinux configuration. If you run SELinux in enforcing mode, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* for more information on SELinux.

You can also run **`virt-install`** interactively. To do so, use the **`--prompt`** command, as in:

```
# virt-install --prompt
```

Once the fully-virtualized guest is created, **`virt-viewer`** will launch the guest and run the operating system's installer. Refer to to the relevant Microsoft installation documentation for instructions on how to install the operating system.

9.2. Installing the Windows Balloon driver

The Windows Balloon driver allows you to dynamically change the amount of memory assigned to a Windows guest, without the need for pausing or rebooting the guest. A kernel driver is required to enable this feature. This section describes how to install the driver.

1. Install the *virtio-win* package, either directly from RHN or through the yum packaging system.

   > **Note**
   >
   > The *virtio-win* package can be found here in RHN: *https://rhn.redhat.com/rhn/software/packages/details/Overview.do?pid=602010*. It requires access to one of the following channels:
   >
   > - RHEL Client Supplementary (v. 6)
   > - RHEL Server Supplementary (v. 6)
   > - RHEL Workstation Supplementary (v. 6)

   Alternatively, run the **`yum install virtio-win`** command on the host.

2. Mount the **`/usr/share/virtio-win/virtio-win.iso`** file as a CD-ROM to the Windows virtual machine.

3. Copy the **Balloon** directory from the mounted CD-ROM to the system drive (C:\).

4. Download the **`devcon.exe`** utility for your system architecture by following the instructions at this URL: *http://social.technet.microsoft.com/wiki/contents/articles/how-to-obtain-the-current-version-of-device-console-utility-devcon-exe.aspx*. Then, copy it to **`C:\Balloon\2k8\x86`** or **`C:\Balloon\2k8\x64`**, depending on your system architecture.

5. Open a command terminal. Navigate to the location of the **devcon** utility and run the following command:

```
devcon install BALLOON.inf "PCI\VEN_1AF4&DEV_1002&SUBSYS_00051AF4&REV_00"
```

6. Restart Windows for changes to take effect.

KVM Para-virtualized Drivers

Para-virtualized drivers are available for virtualized Windows guests running on KVM hosts. These para-virtualized drivers are included in the virtio package. The virtio package supports block (storage) devices and network interface controllers.

Para-virtualized drivers enhance the performance of fully virtualized guests. With the para-virtualized drivers guest I/O latency decreases and throughput increases to near bare-metal levels. It is recommended to use the para-virtualized drivers for fully virtualized guests running I/O heavy tasks and applications.

The KVM para-virtualized drivers are automatically loaded and installed on the following:

- Red Hat Enterprise Linux 4.8 and newer
- Red Hat Enterprise Linux 5.3 and newer
- Red Hat Enterprise Linux 6 and newer
- Some versions of Linux based on the 2.6.27 kernel or newer kernel versions.

Versions of Red Hat Enterprise Linux in the list above detect and install the drivers, additional installation steps are not required.

In Red Hat Enterprise Linux 3 (3.9 and above), manual installation is required.

Note

PCI devices are limited by the virtualized system architecture. Out of the 32 PCI devices for a guest, 4 are always defined for a KVM guest, and are not removable. This means there are up to 28 PCI slots available for additional devices per guest. Each PCI device in a guest can have up to 8 functions.

Using KVM para-virtualizaed drives, the following Microsoft Windows versions are expected to run similarly to bare-metal-based systems.

- Windows XP (32-bit only)
- Windows Server 2003 (32-bit and 64-bit versions)
- Windows Server 2008 (32-bit and 64-bit versions)
- Windows 7 (32-bit and 64-bit versions)

10.1. Installing the KVM Windows para-virtualized drivers

This section covers the installation process for the KVM Windows para-virtualized drivers. The KVM para-virtualized drivers can be loaded during the Windows installation or installed after the guest is installed.

You can install the para-virtualized drivers on your guest by one of the following methods:

- hosting the installation files on a network accessible to the guest,
- using a virtualized CD-ROM device of the driver installation disk .iso file, or

- using a virtualized floppy device to install the drivers during boot time (for Windows guests).

This guide describes installation from the para-virtualized installer disk as a virtualized CD-ROM device.

1. **Download the drivers**
   The *virtio-win* package contains the para-virtualized block and network drivers for all supported Windows guests.

   > **Note**
   >
   > The *virtio-win* package can be found here in RHN: *https://rhn.redhat.com/rhn/software/packages/details/Overview.do?pid=602010*. It requires access to one of the following channels:
   >
   > - RHEL Client Supplementary (v. 6)
   > - RHEL Server Supplementary (v. 6)
   > - RHEL Workstation Supplementary (v. 6)

   Download and install the *virtio-win* package on the host with the **yum** command.

   ```
   # yum install virtio-win
   ```

   The list of virtio-win packages that are supported on Windows operating systems, and the current certified package version, can be found at the following URL: *windowsservercatalog.com*[1].

   Note that the Red Hat Enterprise Virtualization Hypervisor and Red Hat Enterprise Linux are created on the same code base so the drivers for the same version (for example, Red Hat Enterprise Virtualization Hypervisor 3.0 and Red Hat Enterprise Linux 6) are supported for both environments.

   The *virtio-win* package installs a CD-ROM image, **virtio-win.iso**, in the **/usr/share/virtio-win/** directory.

2. **Install the para-virtualized drivers**
   It is recommended to install the drivers on the guest before attaching or modifying a device to use the para-virtualized drivers.

   For block devices storing root file systems or other block devices required for booting the guest, the drivers must be installed before the device is modified. If the drivers are not installed on the guest and the driver is set to the virtio driver the guest will not boot.

10.1.1. Installing the drivers on an installed Windows guest

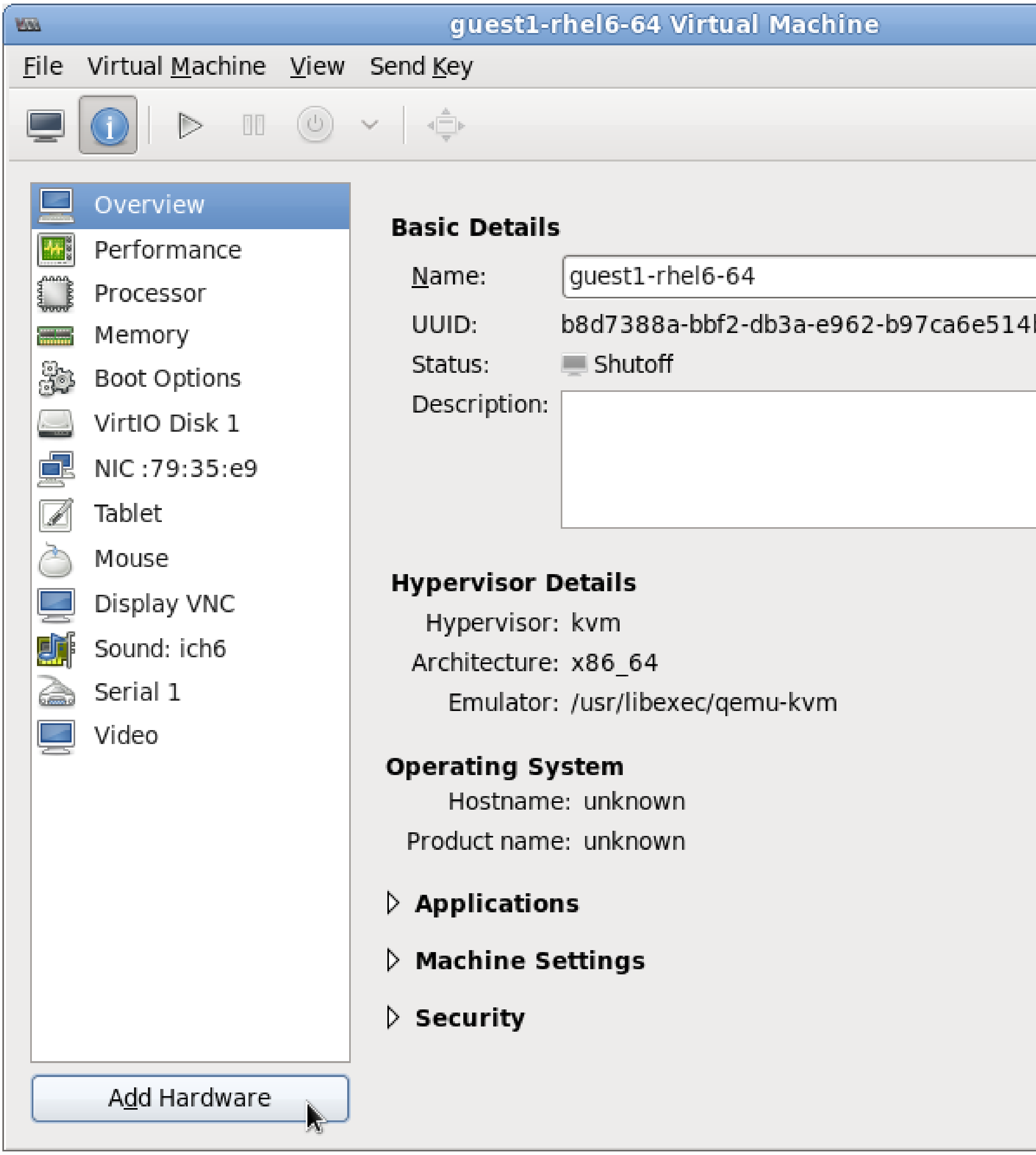
This procedure covers installing the para-virtualized drivers with a virtualized CD-ROM after Windows is installed.

Follow *Procedure 10.1, "Installing from the driver CD-ROM image with virt-manager"* to add a CD-ROM image with **virt-manager** and then install the drivers.

Procedure 10.1. Installing from the driver CD-ROM image with virt-manager

1. **Open virt-manager and the guest**
   Open **`virt-manager`**, select your virtualized guest from the list by double clicking the guest name.

2. **Open the hardware window**
   Click the blue **Information** button at the top to view guest details. Then click the **Add Hardware** button at the bottom of the window.

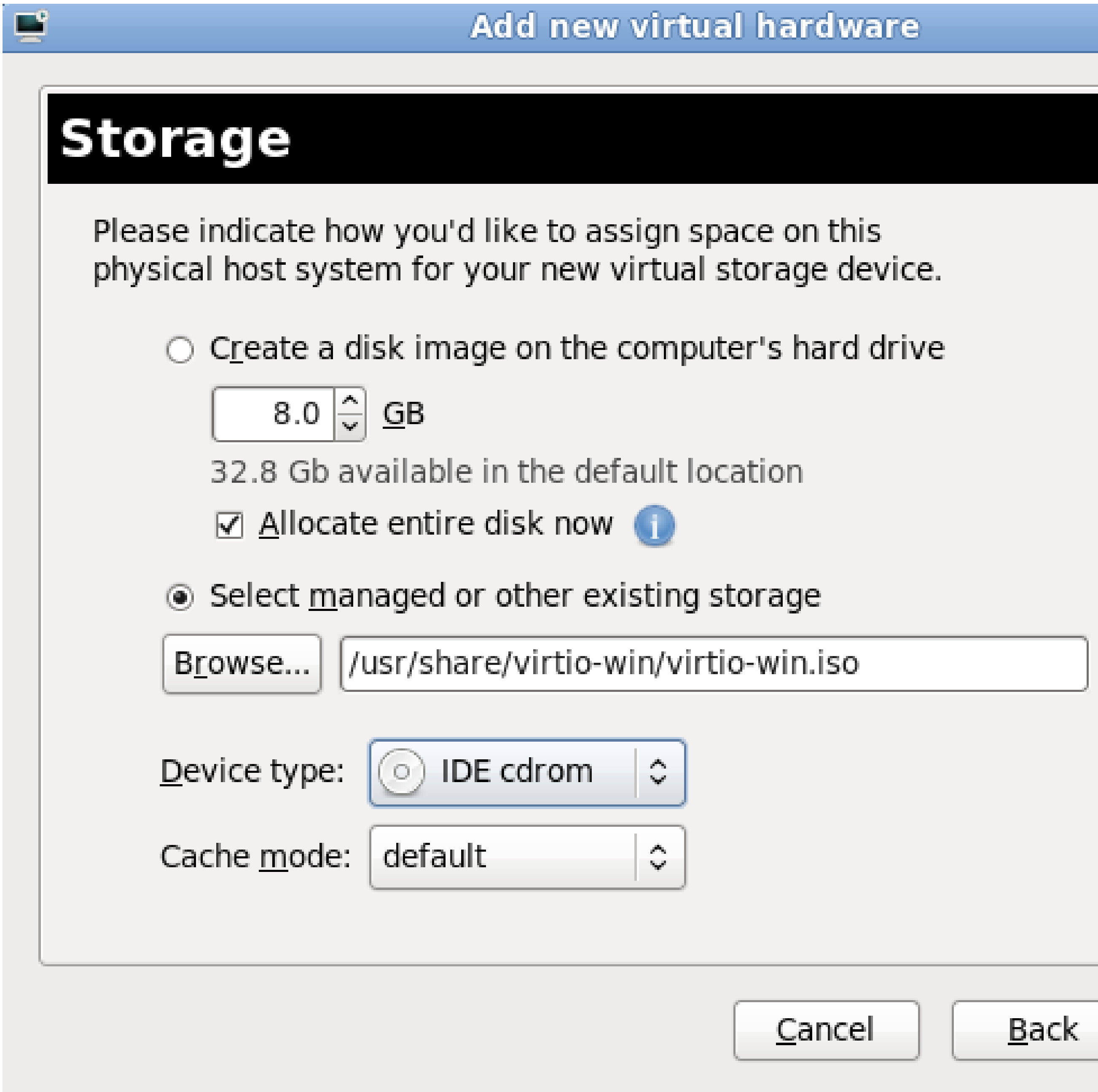


3. **Select the device type**
   This opens a wizard for adding the new device. Select **Storage** from the dropdown menu.

Click the **Forward** button to proceed.

4. **Select the ISO file**
   Select **Select managed or other existing storage** and set the file location of the para-virtualized drivers .iso image file. The default location for the latest version of the drivers is **`/usr/share/virtio-win/virtio-win.iso`**.

   Change the **Device type** to **IDE cdrom** and click the **Forward** button to proceed.



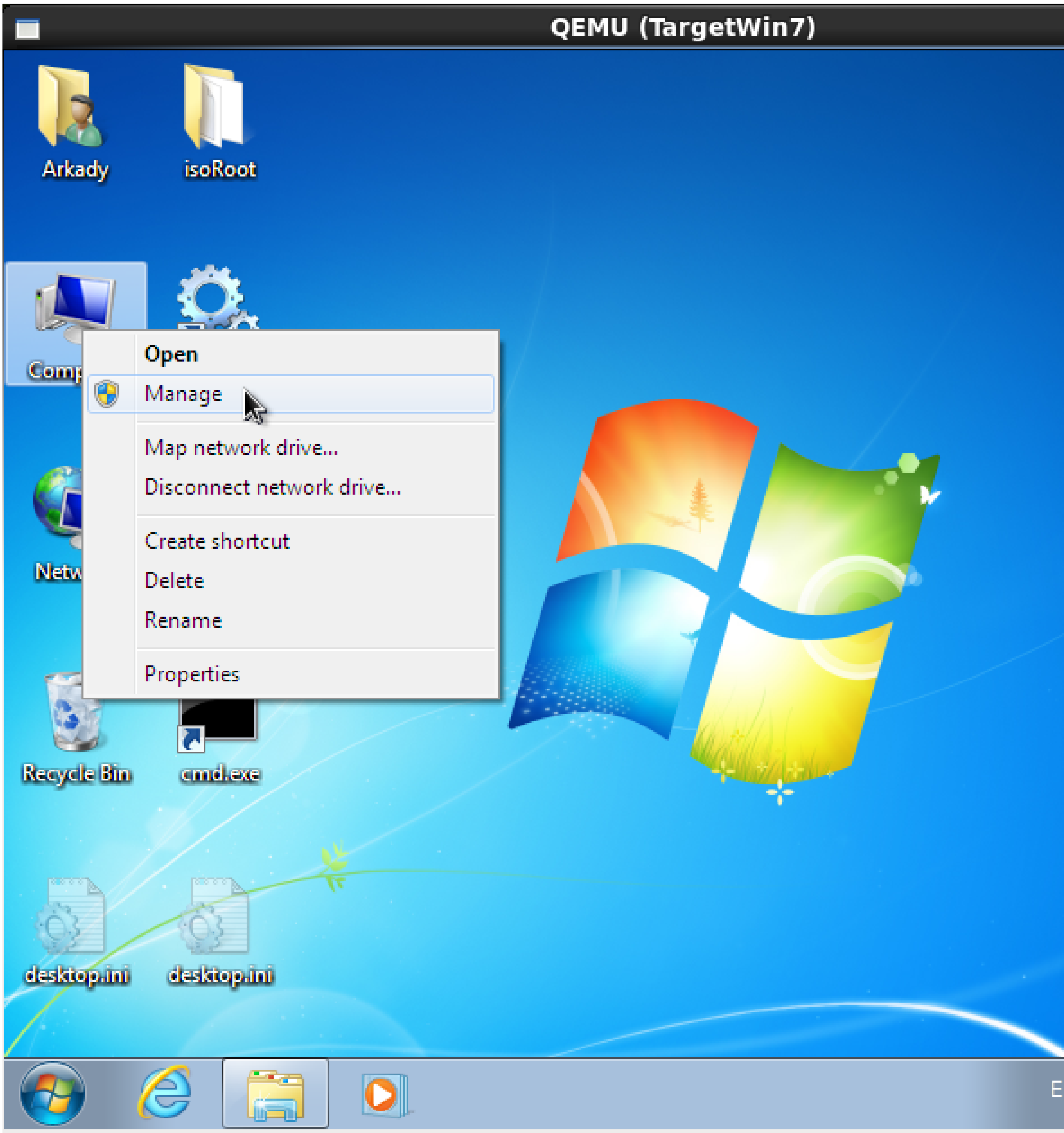
5. **Finish adding virtual hardware**
   Press the **Finish** button to complete the wizard.

6. **Reboot**
   Reboot or start the guest to begin using the driver disc. Virtualized IDE devices require a restart to for the guest to recognize the new device.

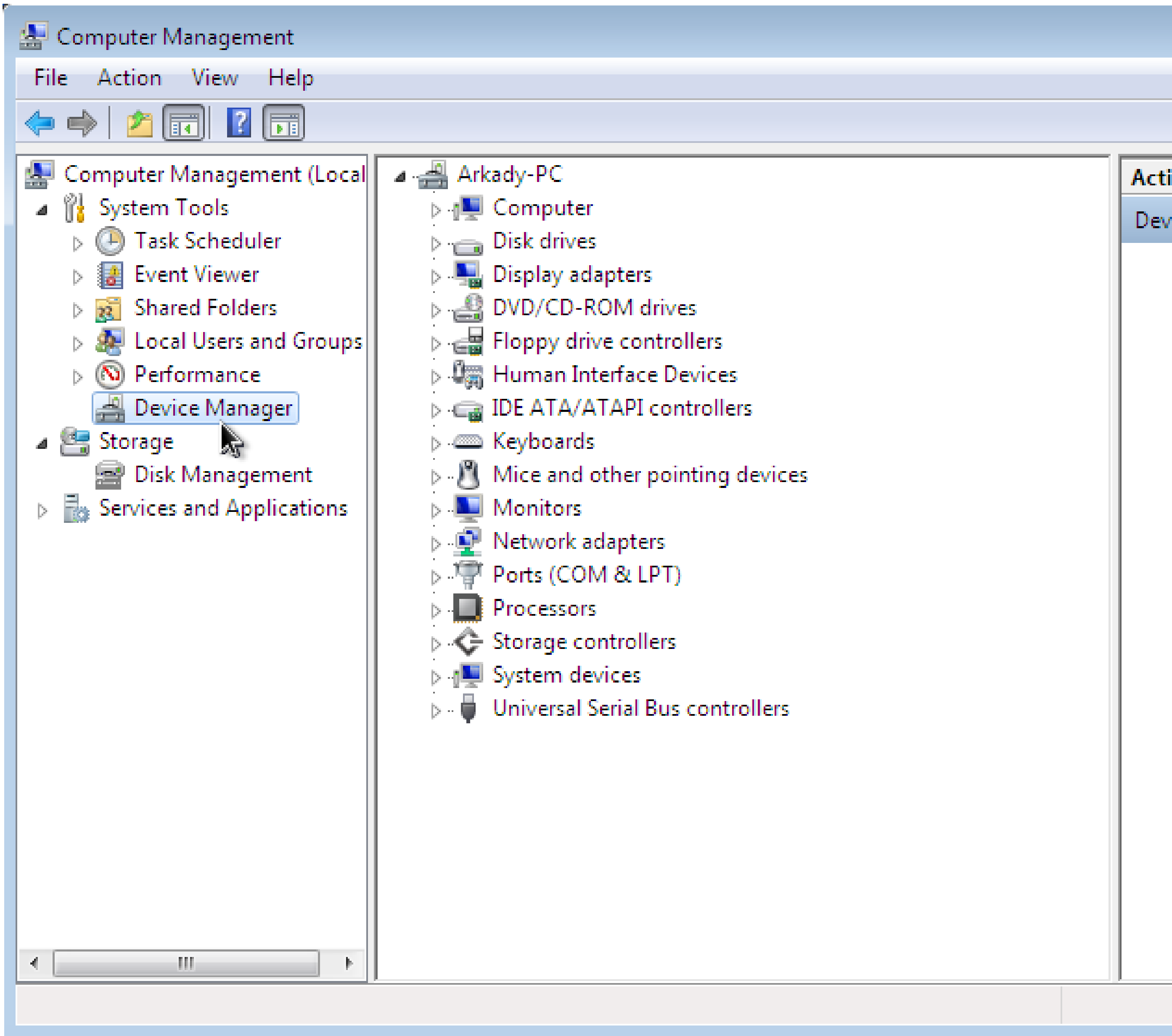
Once the CD-ROM with the drivers is attached and the guest has started, proceed with *Procedure 10.2, “Windows installation”*.

Procedure 10.2. Windows installation

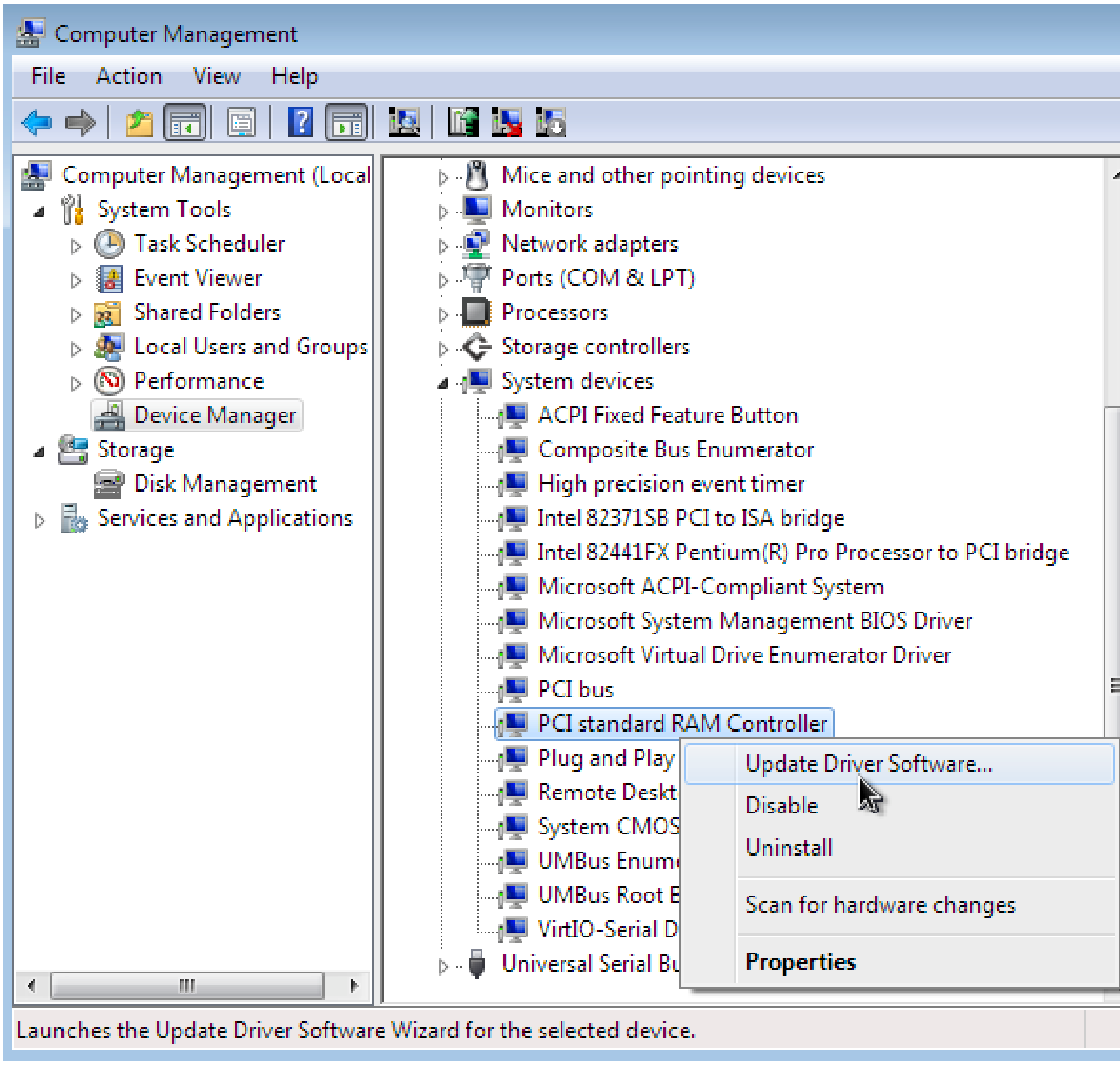
1. **Open the Computer Management window**
   On the desktop, right-click on **My Computer** and select **Manage** from the pop-up menu.



2. **Open the Device Manager**
   Select the **Device Manager** from the left-most pane. This can be found under **Computer Management** > **System Tools**.



3. **Start the driver update wizard**
   Expand **System devices** by clicking on the arrow to its left.

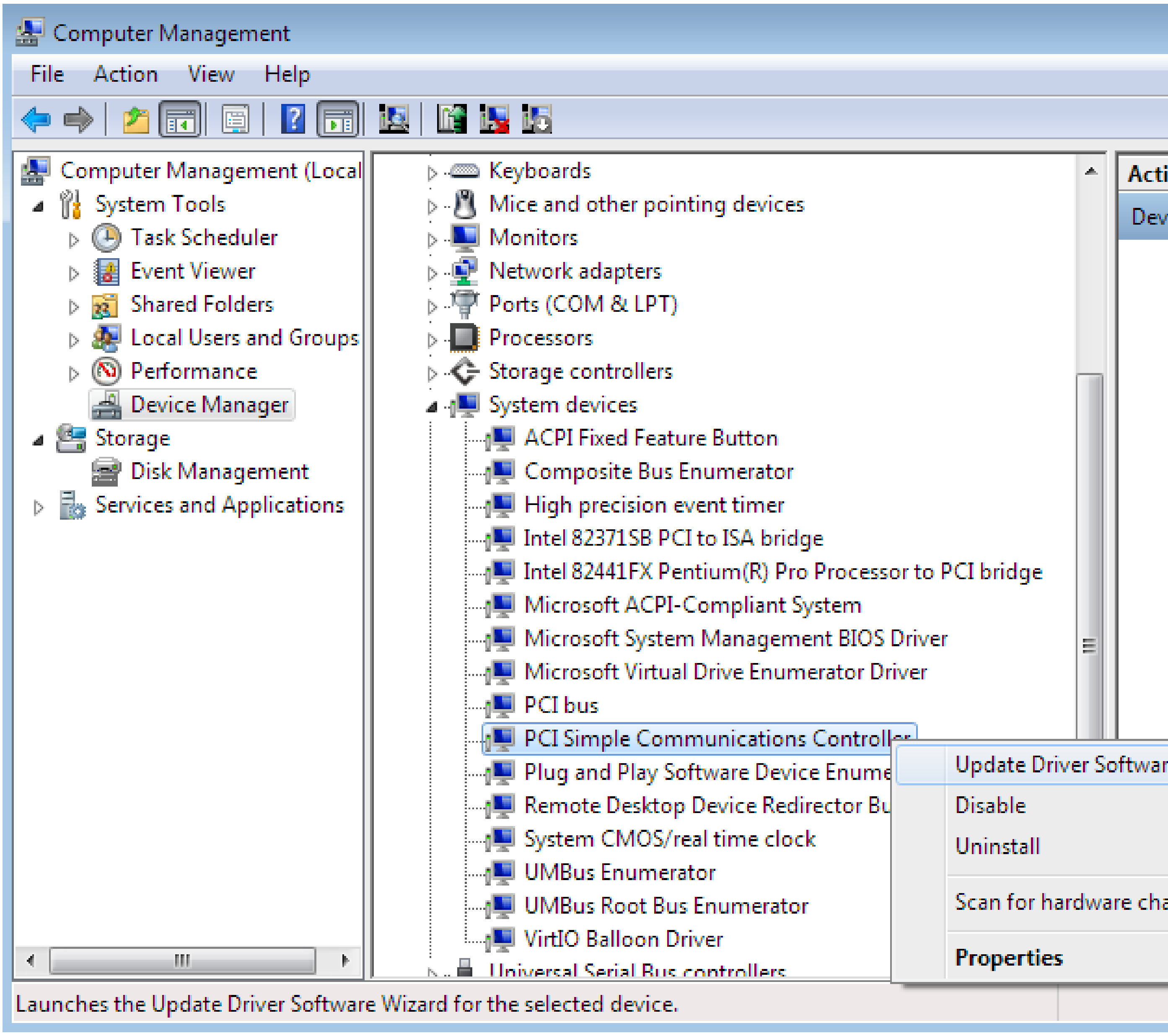


4. **Locate the appropriate device**
   There are four drivers available: the balloon driver, the network driver, the serial driver, and the block driver.

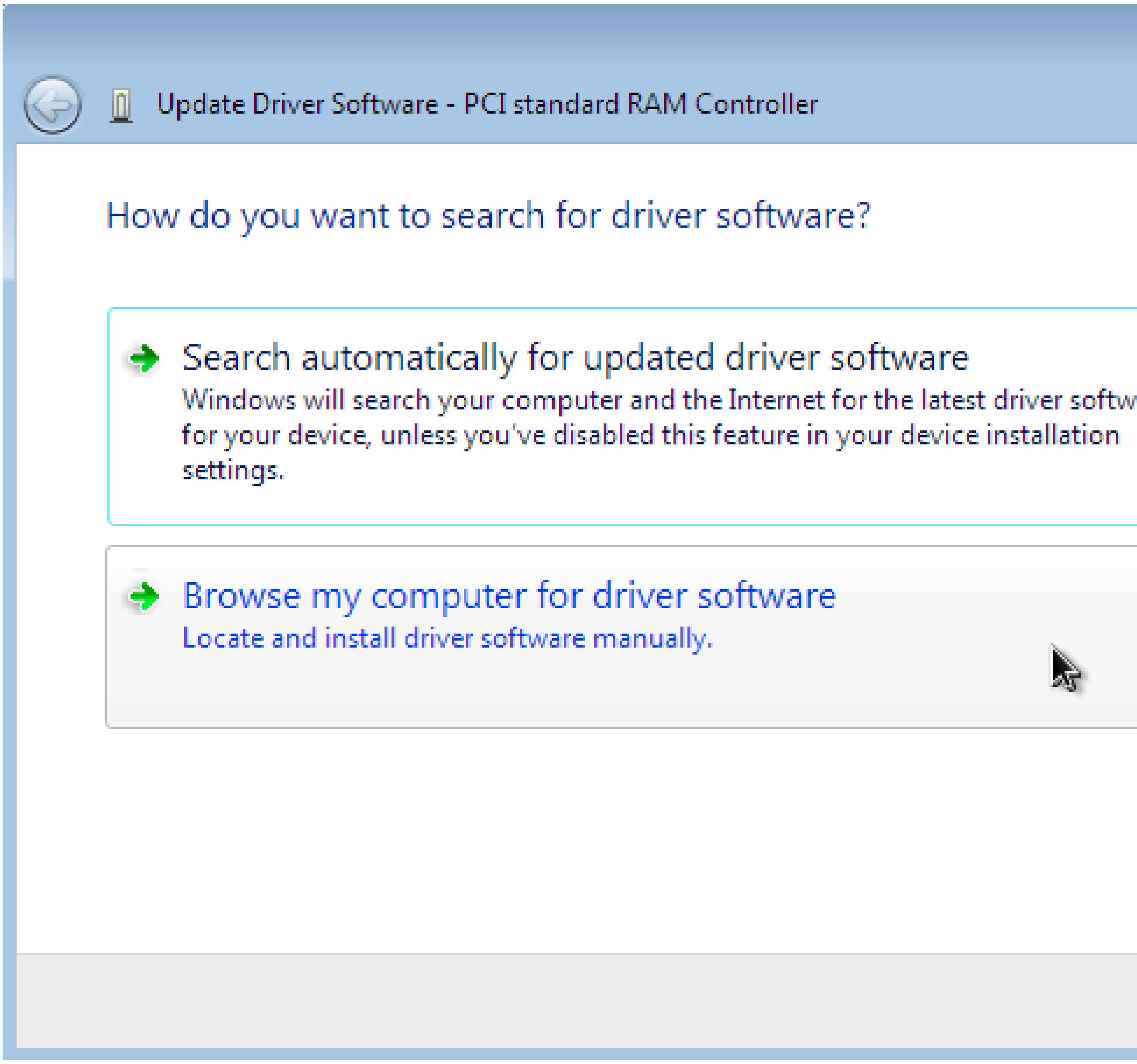
   - **`Balloon`**, the balloon driver, affects the **PCI standard RAM Controller** in the **System devices** group;
   - **`NetKVM`**, the network driver, affects the **Network adapters** group;
   - **`vioserial`**, the serial driver, affects the **PCI Simple Communication Controller** in the **System devices** group; and
   - **`viostor`**, the block driver, affects the **Disk drives** group.

   Right-click on the device whose driver you wish to update, and select **Update Driver Software...** from the pop-up menu.

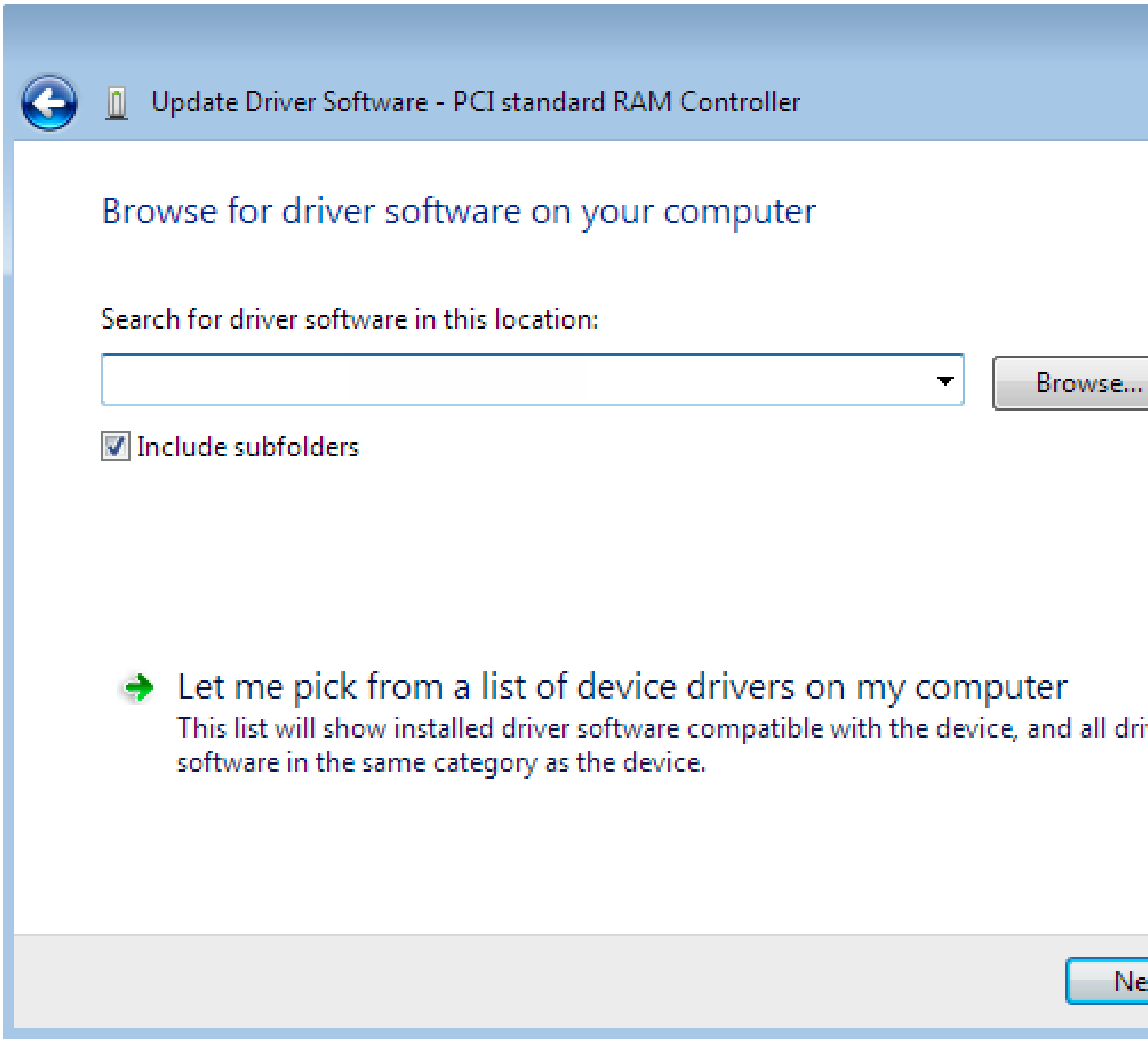
This example installs the balloon driver, so right-click on **PCI standard RAM Controller**.



5. **Specify how to find the driver**
   The first page of the driver update wizard asks how you want to search for driver software. Click on the second option, **Browse my computer for driver software**.

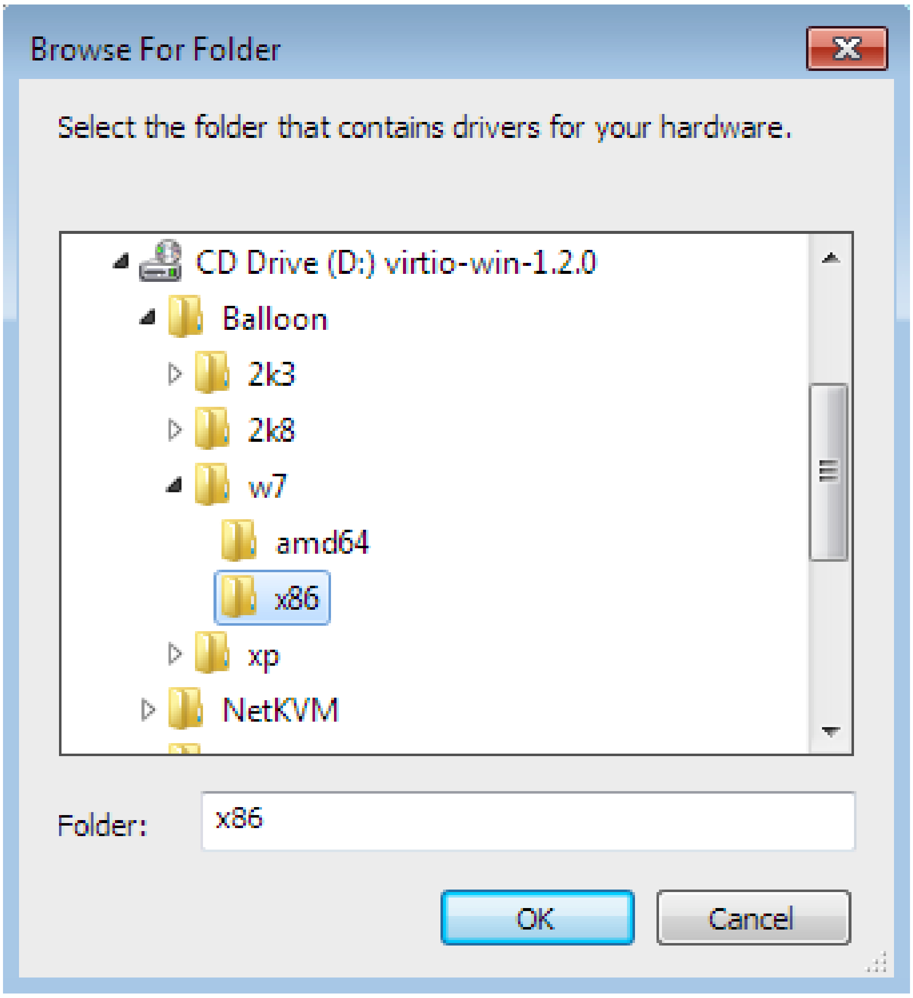


6. **Select the driver to install**
   a. **Open a file browser**
      Click on **Browse...**



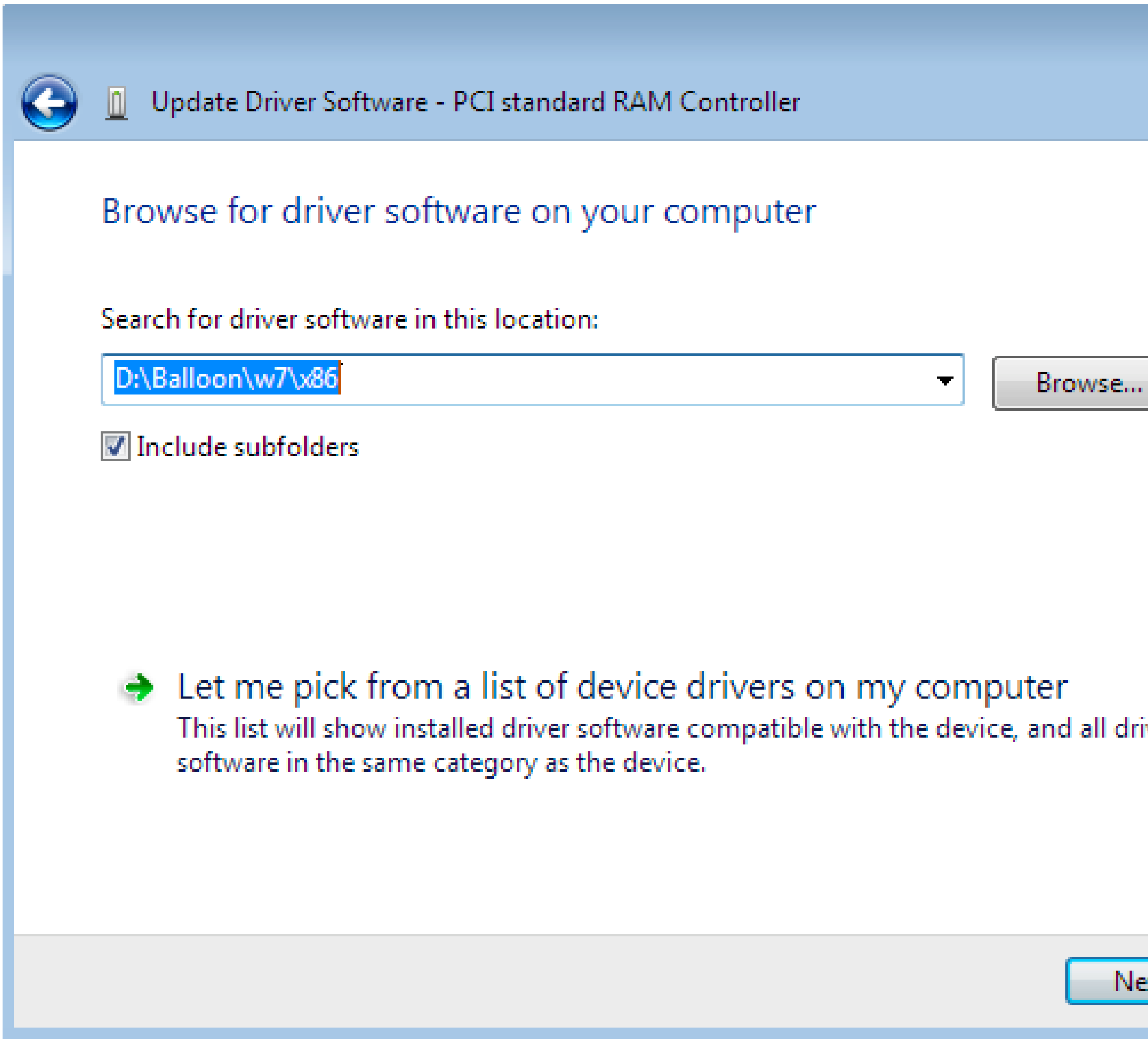
b. **Browse to the location of the driver**

A separate driver is provided for each of the various combinations of operating system and architecture. The executable files that install these drivers are arranged hierarchically according to their driver type, the operating system, and the architecture on which they will be installed: ***`driver_type/os/arch/`***. For example, the Balloon driver for a Windows 7 operating system with an x86 (32-bit) architecture, resides in the **`Balloon/w7/x86`** directory.

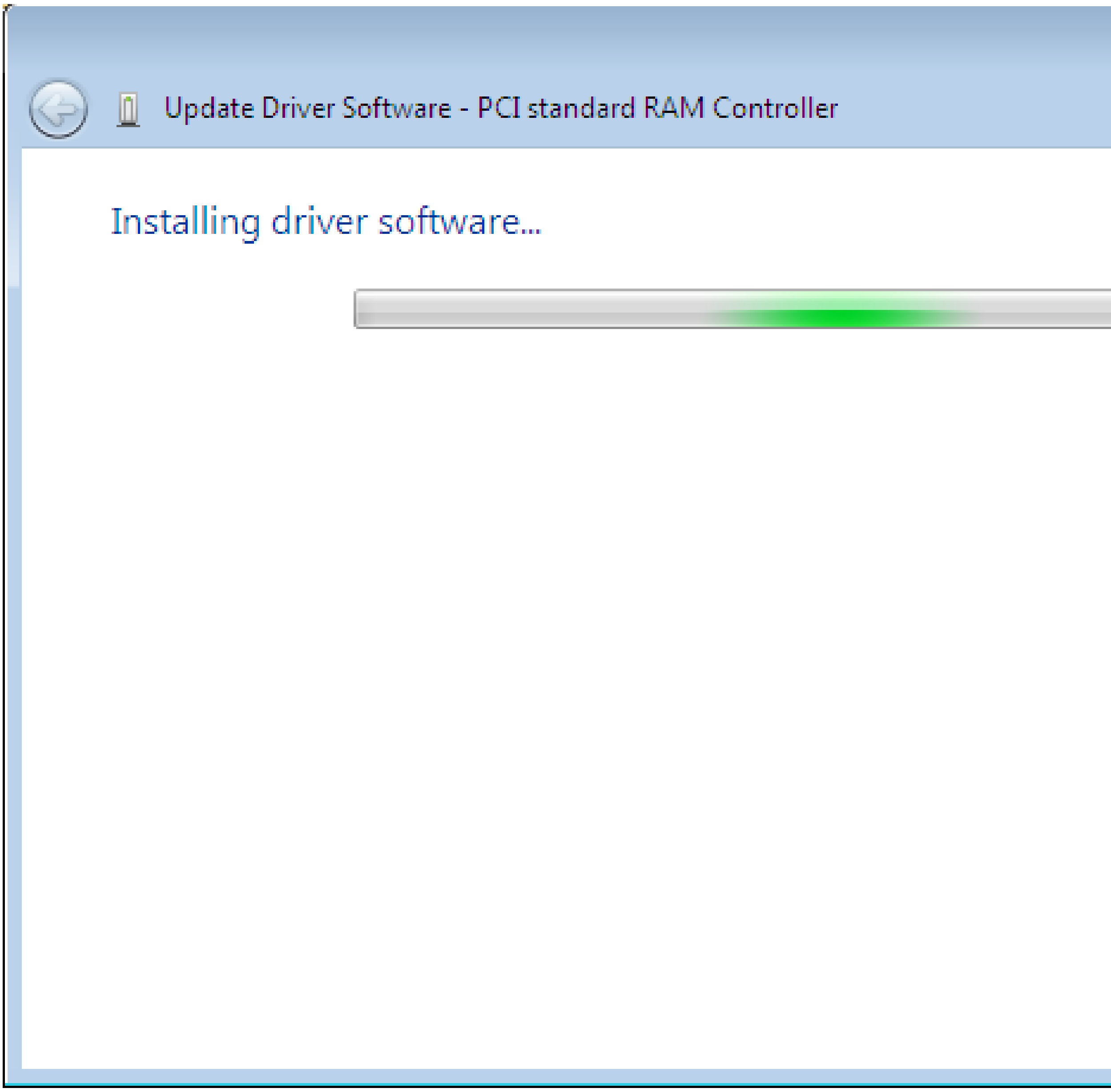


Once you have navigated to the correct location, click **OK**.

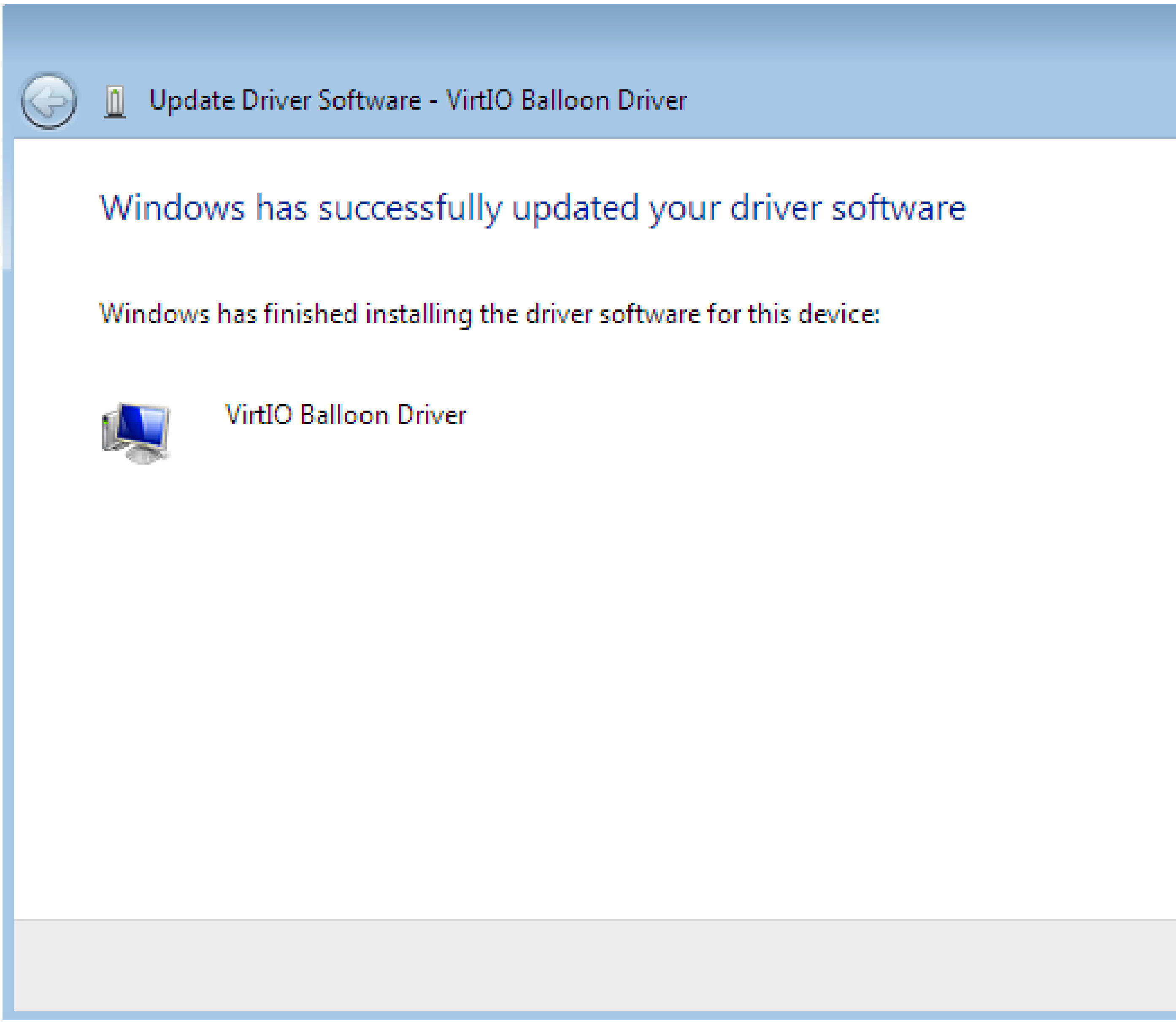
c. **Click Next to continue**



The following screen is displayed while the driver installs:



7. **Close the installer**
   The following screen is displayed when installation is complete:



Click **Close** to close the installer.

8. **Reboot**
   Reboot the guest to complete the driver installation.

Change an existing device to use the para-virtualized drivers or install a new device using the para-virtualized drivers.

10.1.2. Installing drivers during the Windows installation

This procedure covers installing the para-virtualized drivers during a Windows installation.

This method allows a Windows guest to use the para-virtualized (`virtio`) drivers for the default storage device.

1. Install the virtio-win package:

```
# yum install virtio-win
```

Note

The *virtio-win* package can be found here in RHN: *https://rhn.redhat.com/rhn/software/packages/details/Overview.do?pid=602010*. It requires access to one of the following channels:

- RHEL Client Supplementary (v. 6)
- RHEL Server Supplementary (v. 6)
- RHEL Workstation Supplementary (v. 6)

Creating guests

Create the guest, as normal, without starting the guest. Follow one of the procedures below.

2. **Creating the guest**
Select *one* of the following guest-creation methods, and follow the instructions.

   a. **Creating the guest with virsh**
   This method attaches the para-virtualized driver floppy disk to a Windows guest *before* the installation.

   If the guest is created from an XML definition file with **`virsh`** use the **`virsh define`** command not the **`virsh create`** command.

      i. Create, but do not start, the guest. Refer to the *Red Hat Enterprise Linux Virtualization Administration Guide* for details on creating guests with the **`virsh`** command.

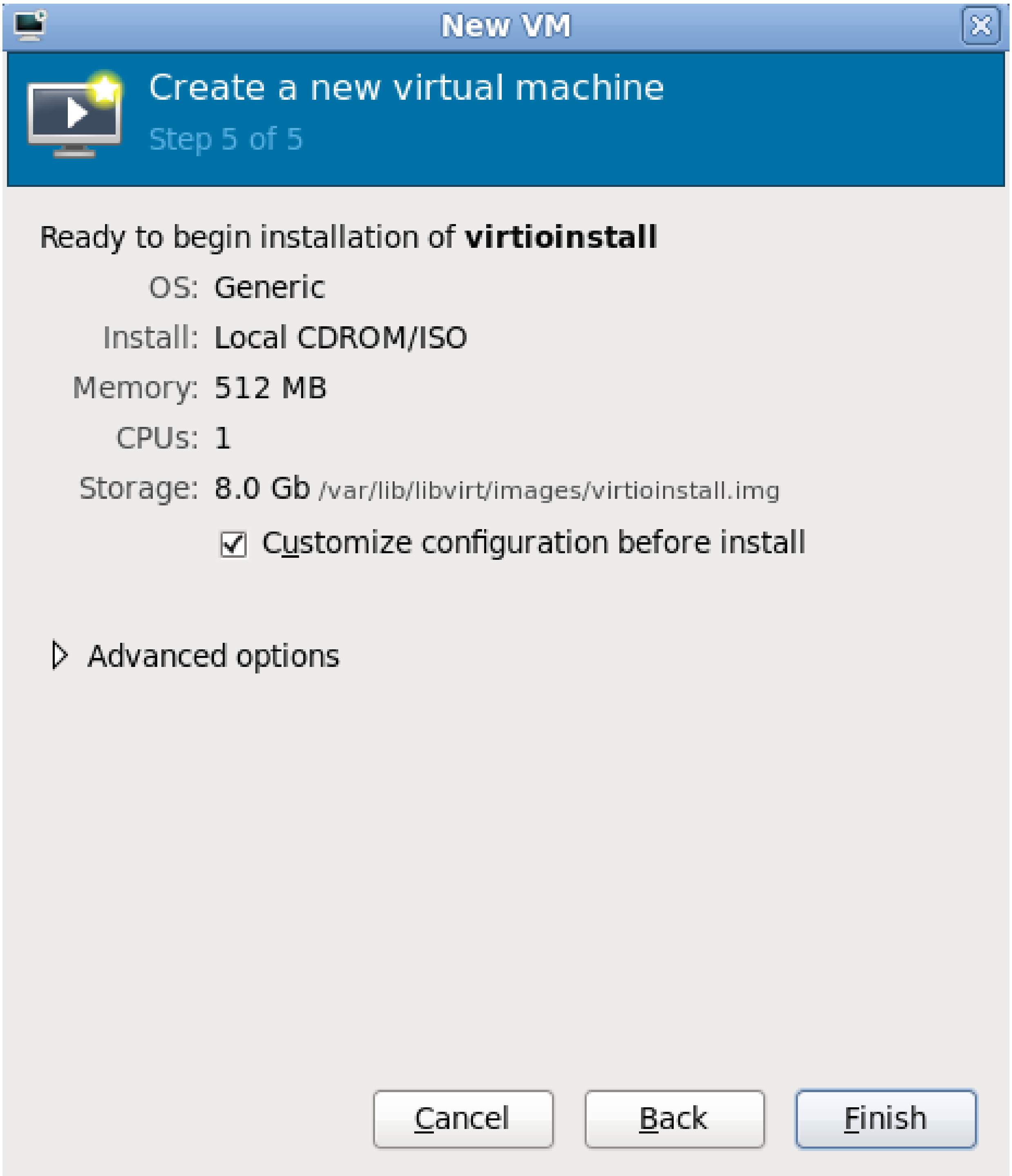
      ii. Add the driver disk as a virtualized floppy disk with the **`virsh`** command. This example can be copied and used if there are no other virtualized floppy devices attached to the virtualized guest.

      ```
      # virsh attach-disk guest1 /usr/share/virtio-win/virtio-win.vfd fda --type
       floppy
      ```

      You can now continue with *Step 3*.

   b. **Creating the guest with virt-manager and changing the disk type**

      i. At the final step of the virt-manager guest creation wizard, check the **Customize configuration before install** checkbox.



Press the **Finish** button to continue.

ii. **Add the new device**
Select **Storage** from the **Hardware type** list. Click **Forward** to continue.

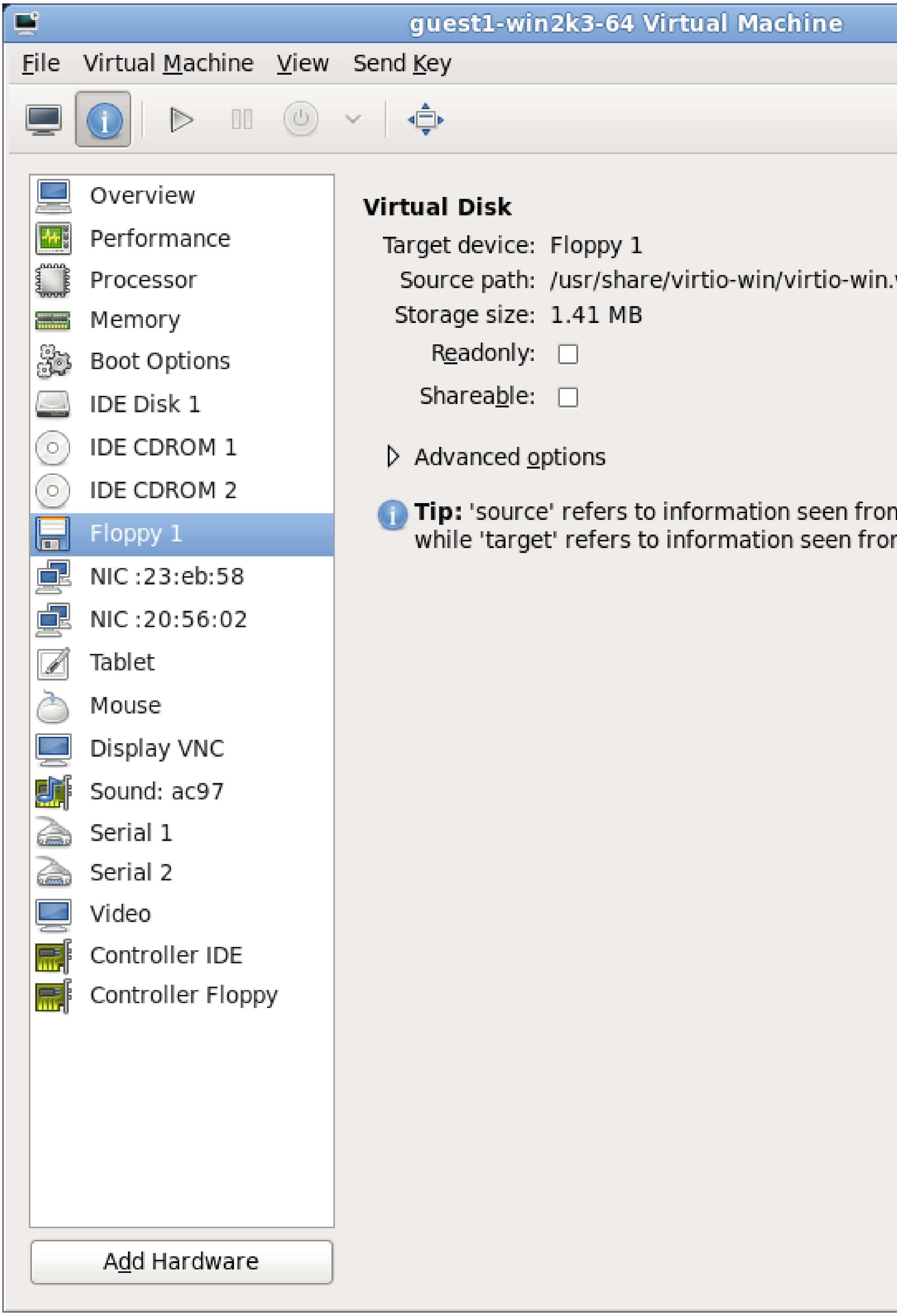
iii. **Select the driver disk**
Select **Select managed or existing storage**.

Set the location to **`/usr/share/virtio-win/virtio-win.vfd`**.

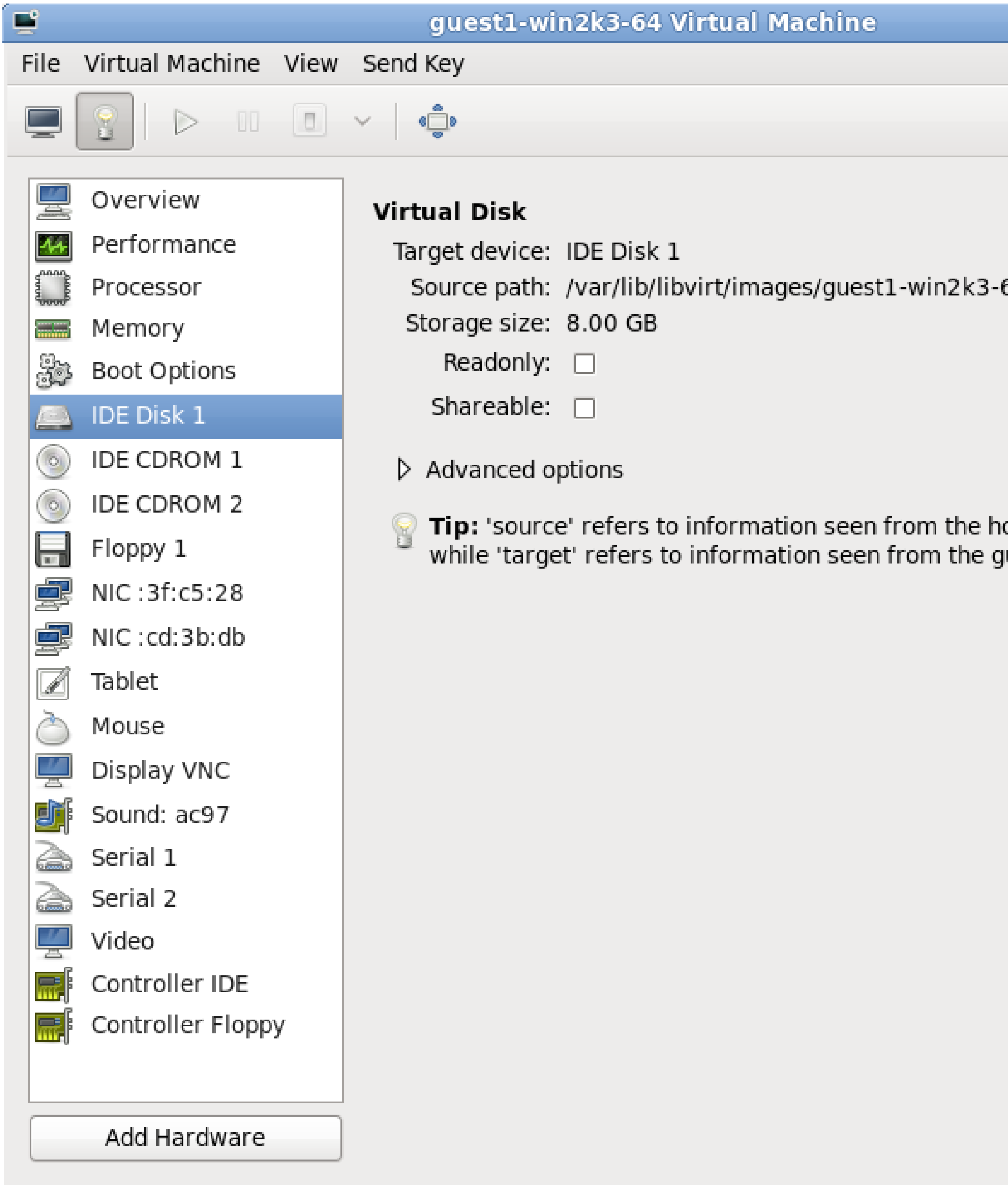
Change **Device type** to **Floppy disk**.

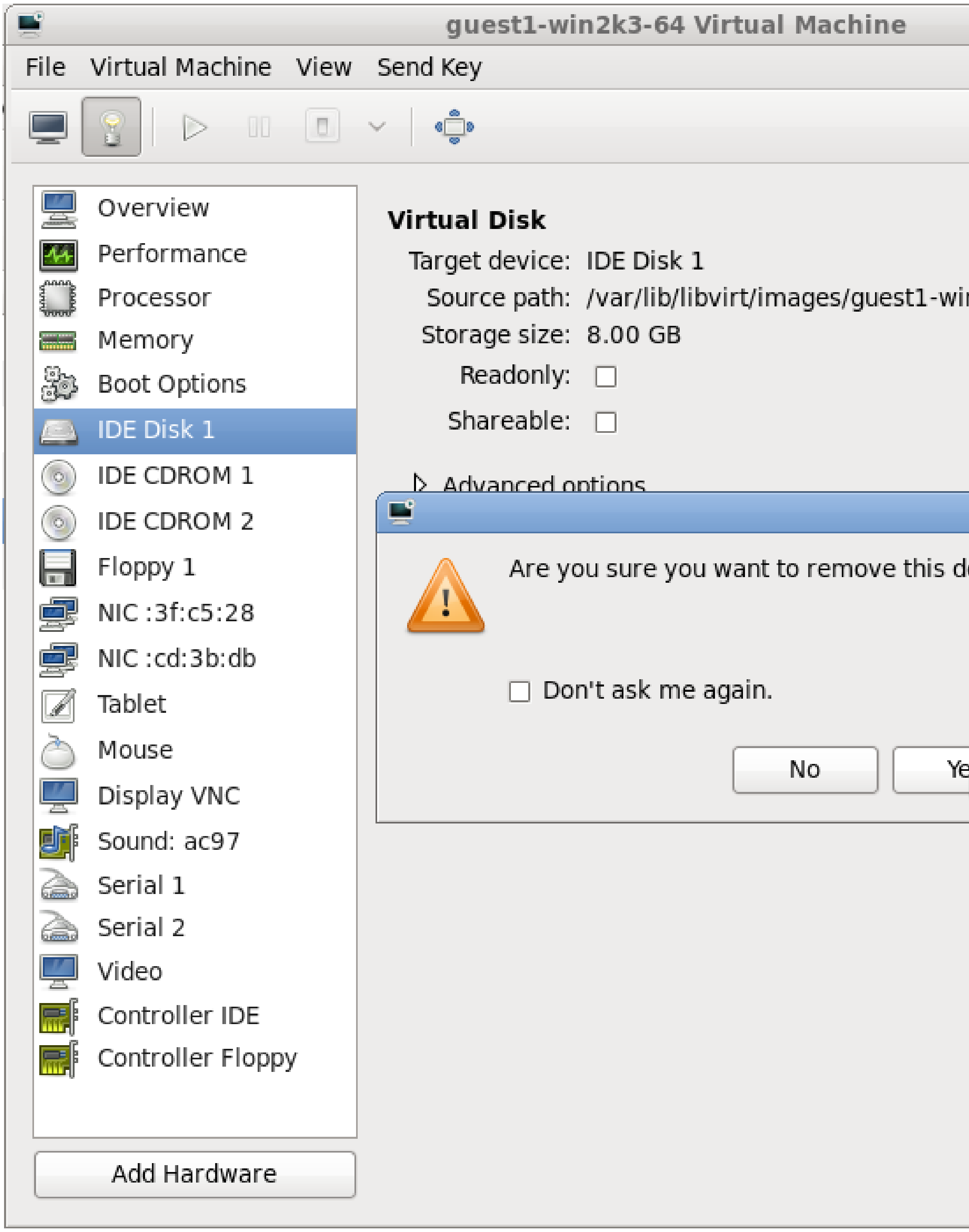
Press the **Finish** button to continue.

iv. **Confirm settings**
Review the device settings.

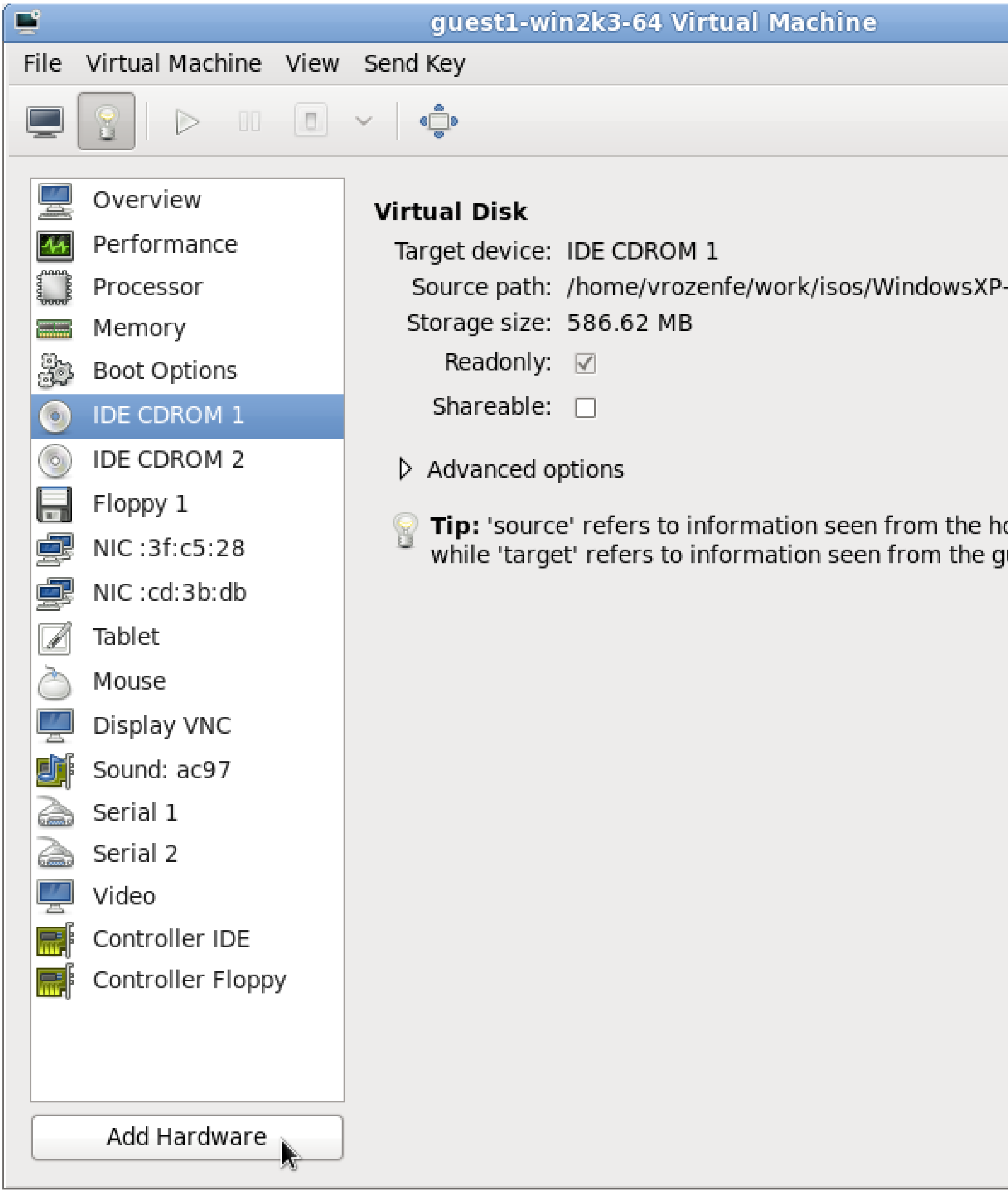

guest1-win2k3-64 Virtual Machine
File Virtual Machine View Send Key
Overview
Performance
Processor
Memory
Boot Options
IDE Disk 1
IDE CDROM 1
IDE CDROM 2
Floppy 1
NIC :23:eb:58
NIC :20:56:02
Tablet
Mouse
Display VNC
Sound: ac97
Serial 1
Serial 2
Video
Controller IDE
Controller Floppy
Add Hardware
Virtual Disk
Target device: Floppy 1
Source path: /usr/share/virtio-win/virtio-win.
Storage size: 1.41 MB
Readonly:
Shareable:
Advanced options
Tip: 'source' refers to information seen fron
while 'target' refers to information seen fror

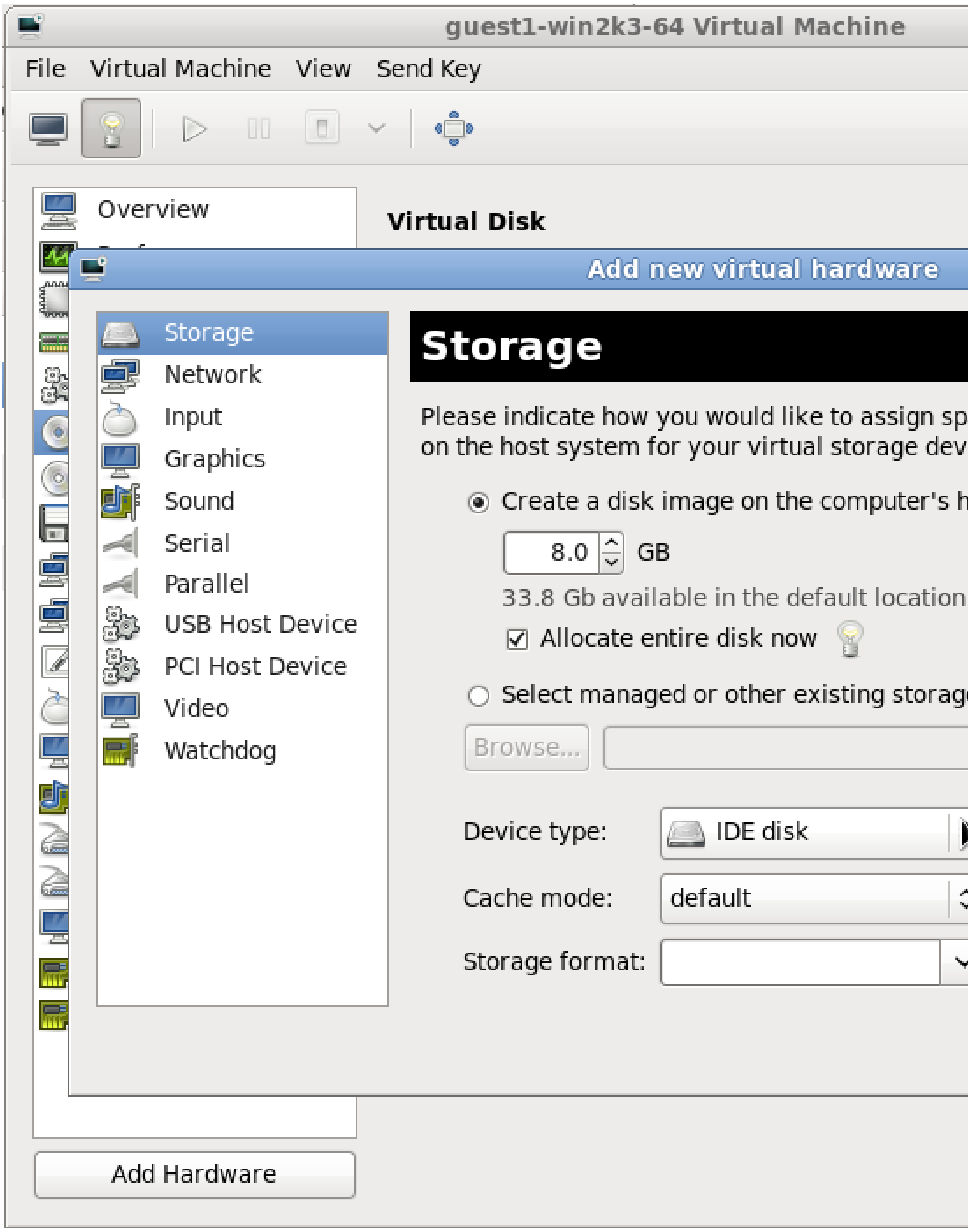
v. **Change the disk type**
Change the disk type from `IDE Disk` to `Virtio Disk` by removing IDE Disk 1. Select the disk, press **Remove** and then confirm the action by pressing **Yes**.

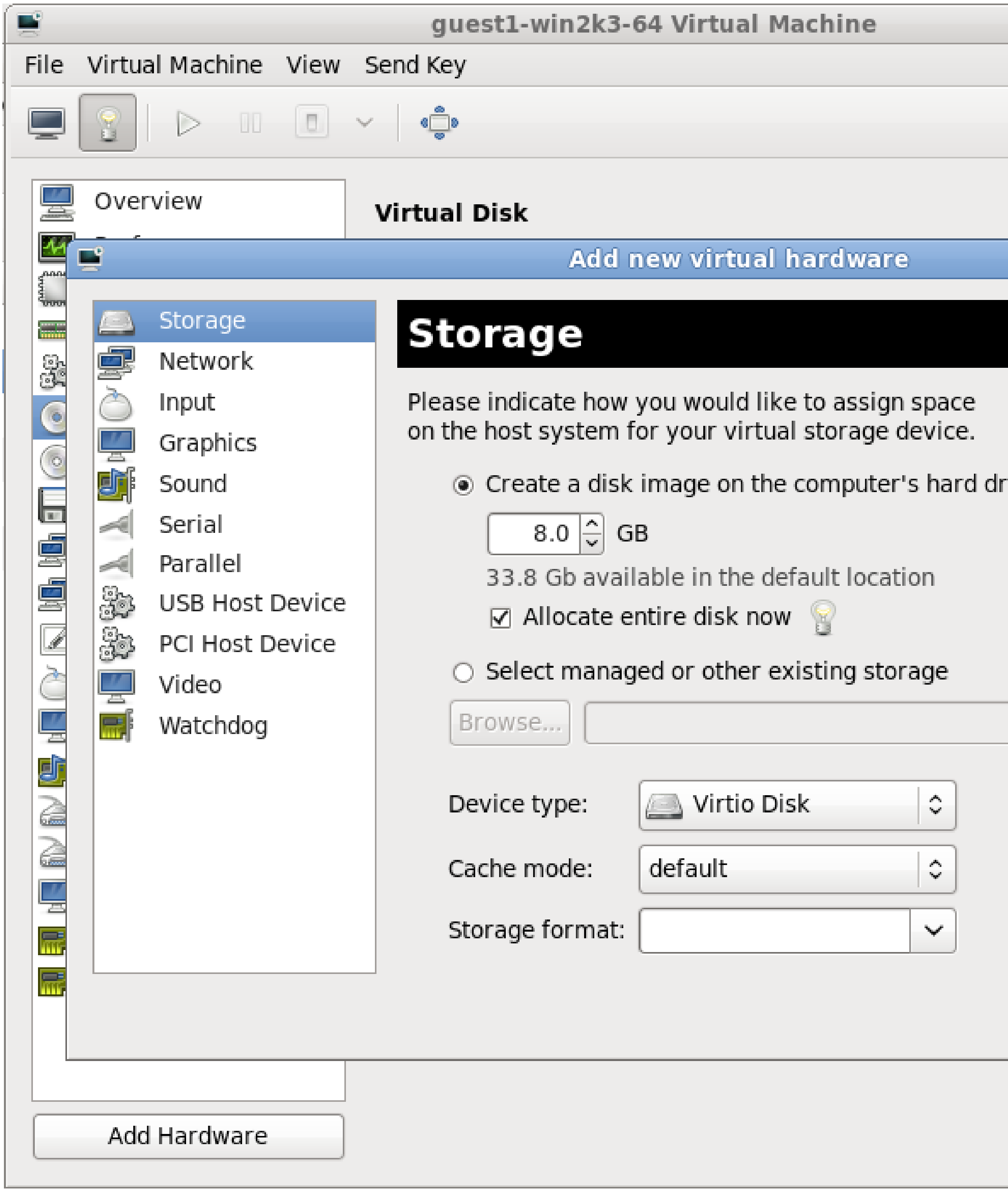


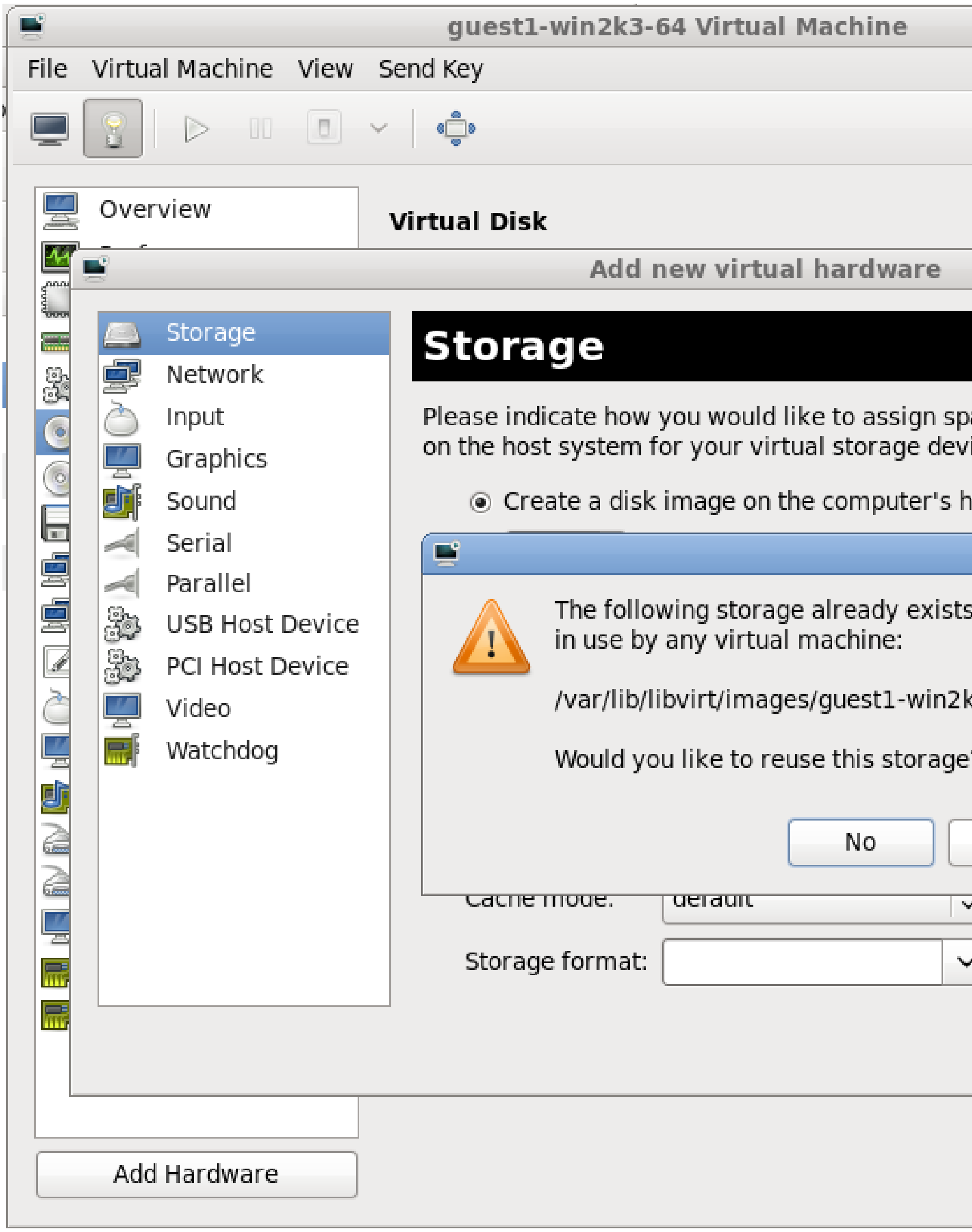


Add a new virtual storage device by pressing **Add Hardware**. Then, change the **Device type** from *`IDE disk`* to *`Virtio Disk`*. Then, press **Finish** and confirm the operation by pressing **Yes.**

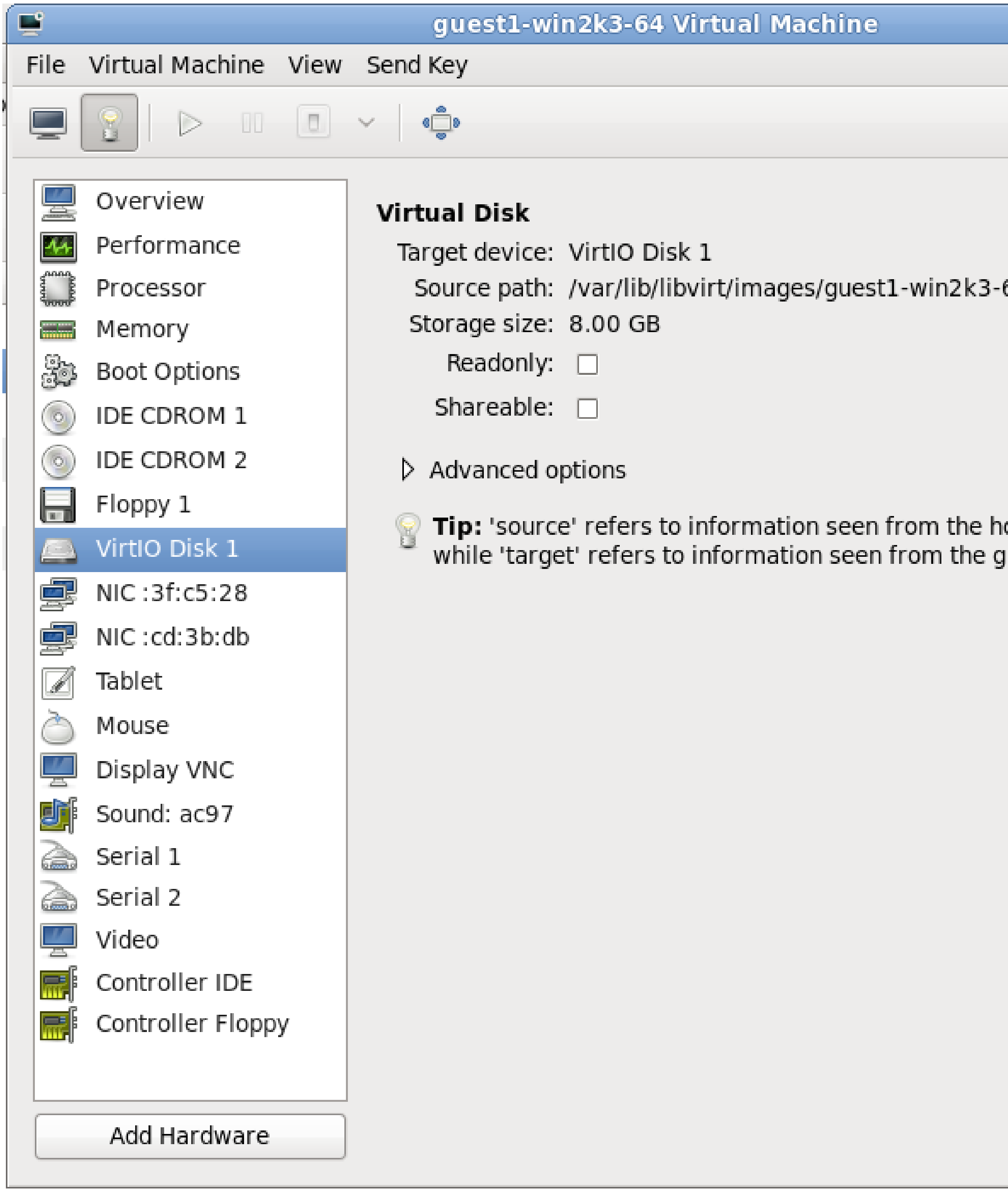

guest1-win2k3-64 Virtual Machine
File
Virtual Machine
View
Send Key
Overview
Performance
Processor
Memory
Boot Options
IDE CDROM 1
IDE CDROM 2
Floppy 1
NIC :3f:c5:28
NIC :cd:3b:db
Tablet
Mouse
Display VNC
Sound: ac97
Serial 1
Serial 2
Video
Controller IDE
Controller Floppy
Add Hardware
Virtual Disk
Target device: IDE CDROM 1
Source path: /home/vrozenfe/work/isos/WindowsXP-
Storage size: 586.62 MB
Readonly:
Shareable:
Advanced options
Tip: 'source' refers to information seen from the ho
while 'target' refers to information seen from the g


guest1-win2k3-64 Virtual Machine
File
Virtual Machine
View
Send Key
Overview
Virtual Disk
Add new virtual hardware
Storage
Network
Input
Graphics
Sound
Serial
Parallel
USB Host Device
PCI Host Device
Video
Watchdog
Storage
Please indicate how you would like to assign sp
on the host system for your virtual storage dev
Create a disk image on the computer's h
8.0
GB
33.8 Gb available in the default location
Allocate entire disk now
Select managed or other existing storag
Browse...
Device type:
IDE disk
Cache mode:
default
Storage format:
Add Hardware


guest1-win2k3-64 Virtual Machine
File
Virtual Machine
View
Send Key
Overview
Virtual Disk
Add new virtual hardware
Storage
Network
Input
Graphics
Sound
Serial
Parallel
USB Host Device
PCI Host Device
Video
Watchdog
Storage
Please indicate how you would like to assign space on the host system for your virtual storage device.
Create a disk image on the computer's hard dr
8.0
GB
33.8 Gb available in the default location
Allocate entire disk now
Select managed or other existing storage
Browse...
Device type:
Virtio Disk
Cache mode:
default
Storage format:
Add Hardware



Review the settings for *VirtIO Disk 1*.



You can now continue with *Step 3*.

c. **Creating the guest with virt-install**
Append the following parameter exactly as listed below to add the driver disk to the installation with the **`virt-install`** command:

```
--disk path=/usr/share/virtio-win/virtio-win.vfd,device=floppy
```

According to the version of Windows you are installing, append one of the following options to the **virt-install** command:

```
--os-variant winxp
```

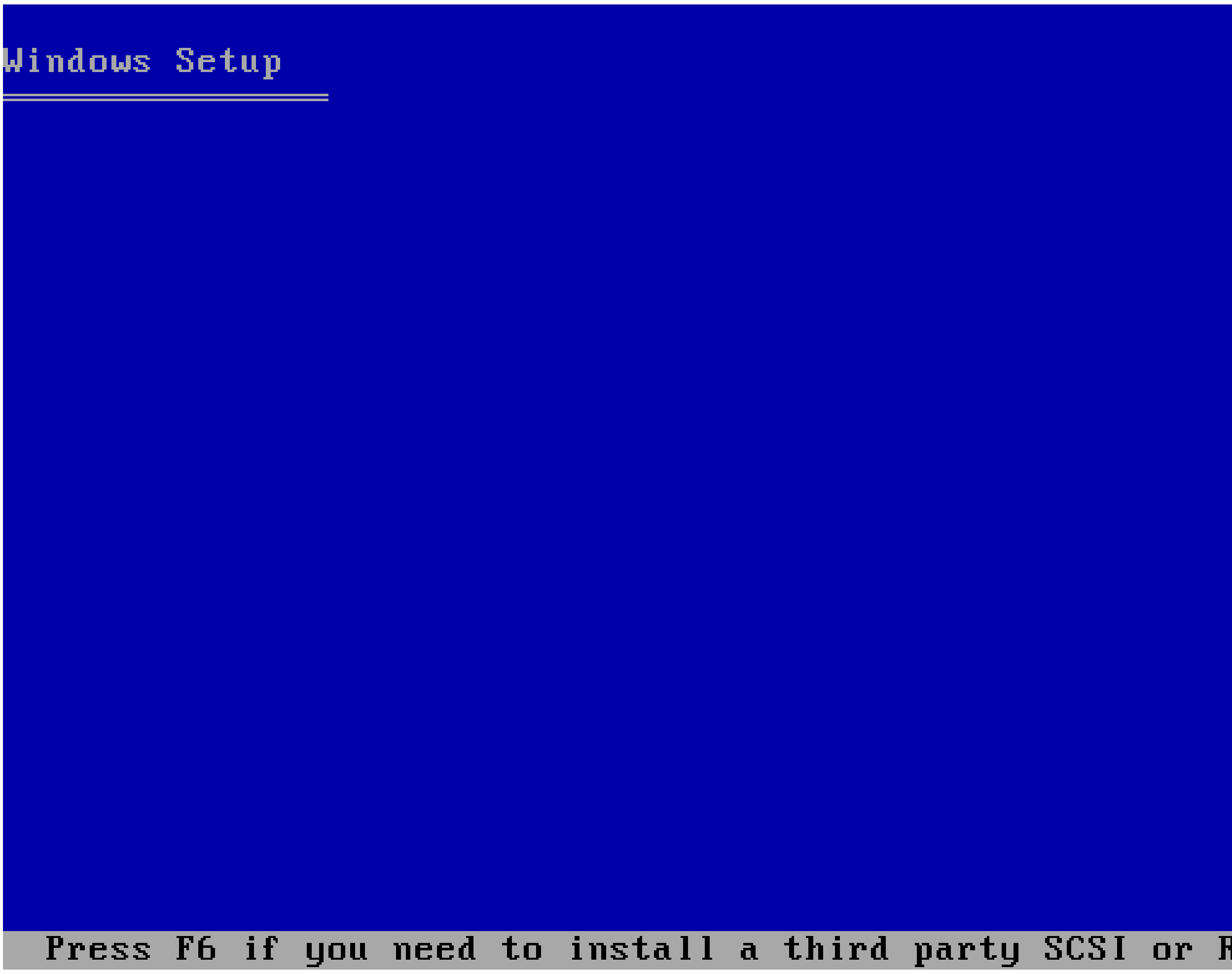
```
--os-variant win2003
```

```
--os-variant win7
```

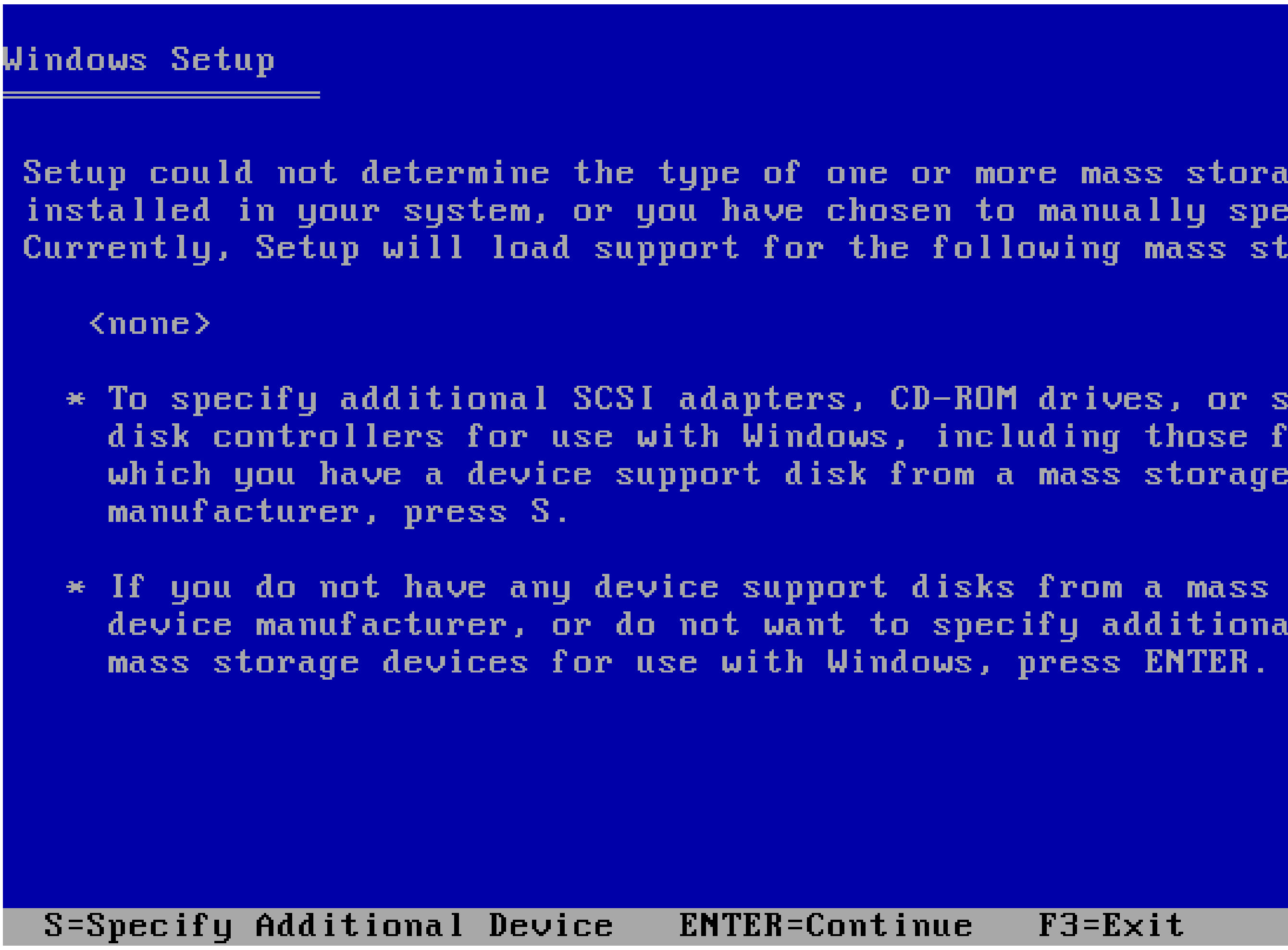
You can now continue with *Step 3*.

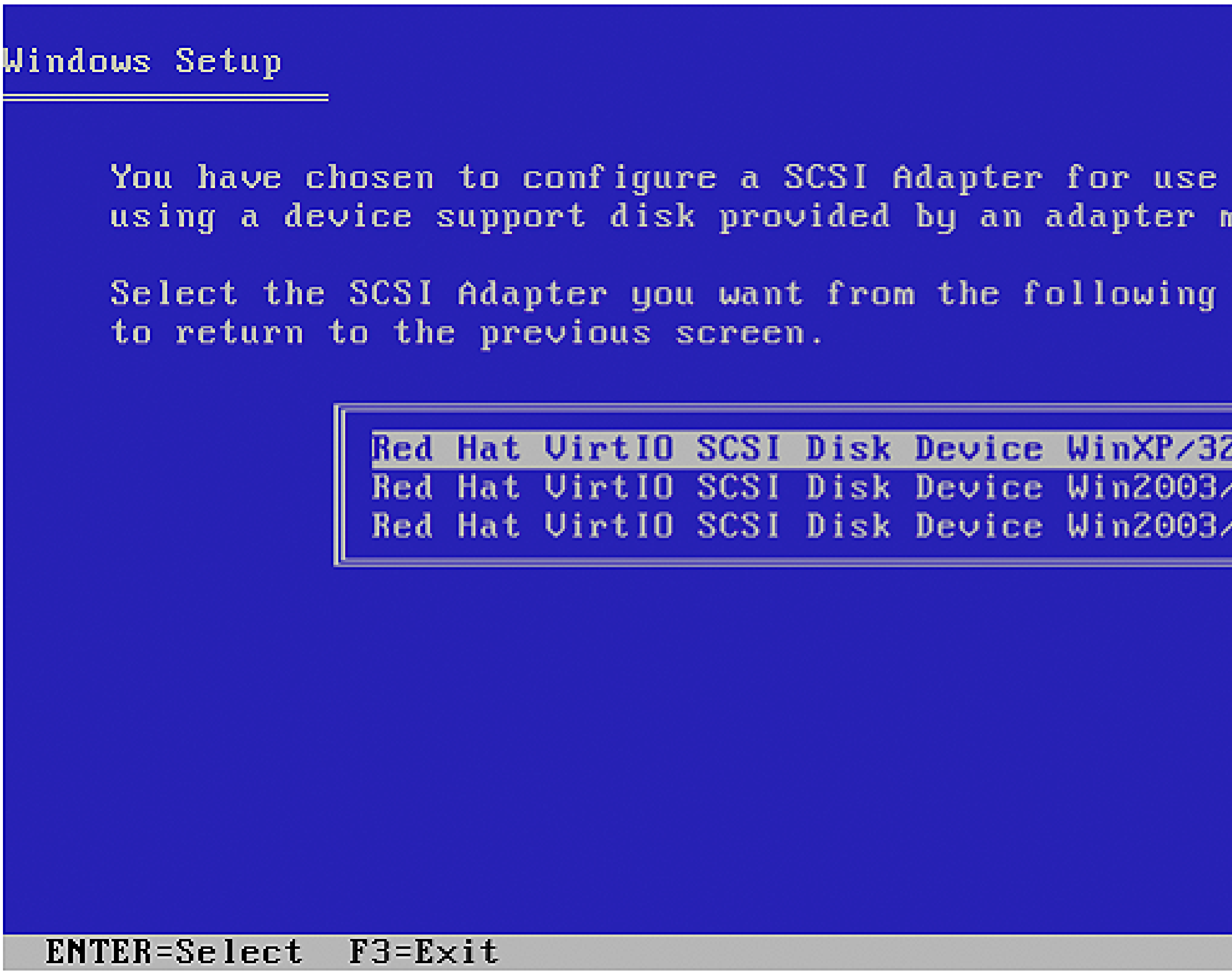
3. **Additional steps for driver installation**
   During the installation, additional steps are required to install drivers, depending on the type of Windows guest.

   a. **Windows Server 2003 and Windows XP**
      Before the installation blue screen repeatedly press **F6** for third party drivers.



Press S to install additional


Windows Setup
Setup could not determine the type of one or more mass stora
installed in your system, or you have chosen to manually spe
Currently, Setup will load support for the following mass st
<none>
* To specify additional SCSI adapters, CD-ROM drives, or s
disk controllers for use with Windows, including those f
which you have a device support disk from a mass storage
manufacturer, press S.
* If you do not have any device support disks from a mass
device manufacturer, or do not want to specify additiona
mass storage devices for use with Windows, press ENTER.
S=Specify Additional Device   ENTER=Continue   F3=Exit



Press **`Enter`** to continue the installation.

b. **Windows Server 2008**
Follow the same procedure for Windows Server 2003, but when the installer prompts you for the driver, click on **Load Driver**, point the installer to `Drive A:` and pick the driver that suits your guest operating system and architecture.

10.2. Using the para-virtualized drivers with Red Hat Enterprise Linux 3.9 guests

Para-virtualized drivers for Red Hat Enterprise Linux 3.9 consist of five kernel modules: **`virtio`**, **`virtio_blk`**, **`virtio_net`**, **`virtio_pci`** and **`virtio_ring`**. All five modules must be loaded to use both the para-virtualized block and network devices drivers.

Note

For Red Hat Enterprise Linux 3.9 guests, the *kmod-virtio* package is a requirement for the **`virtio`** module.

> **Note**
>
> To use the network device driver only, load the **`virtio`**, **`virtio_net`** and **`virtio_pci`** modules. To use the block device driver only, load the **`virtio`**, **`virtio_ring`**, **`virtio_blk`** and **`virtio_pci`** modules.

> **Modified initrd files**
>
> The *virtio* package modifies the initrd RAM disk file in the **`/boot`** directory. The original initrd file is saved to **`/boot/initrd-`** *`kernel-version`* **`.img.virtio.orig`**. The original initrd file is replaced with a new initrd RAM disk containing the **`virtio`** driver modules. The initrd RAM disk is modified to allow the guest to boot from a storage device using the para-virtualized drivers. To use a different initrd file, you must ensure that drivers are loaded with the **`sysinit`** script (*Loading the para-virtualized drivers with the sysinit script*) or when creating new initrd RAM disk (*Adding the para-virtualized drivers to the initrd RAM disk*).

Loading the para-virtualized drivers with the sysinit script

This procedure covers loading the para-virtualized driver modules during the boot sequence on a Red Hat Enterprise Linux 3.9 or newer guest with the **`sysinit`** script. Note that the guest cannot use the para-virtualized drivers for the default boot disk if the modules are loaded with the **`sysinit`** script.

The drivers must be loaded in the following order:

1. **`virtio`**
2. **`virtio_ring`**
3. **`virtio_pci`**
4. **`virtio_blk`**
5. **`virtio_net`**

`virtio_net` and **`virtio_blk`** are the only drivers whose order can be changed. If other drivers are loaded in a different order, they will not work.

Next, configure the modules. Locate the following section of the **`/etc/rc.d/rc.sysinit`** file.

```
if [ -f /etc/rc.modules ]; then
   /etc/rc.modules
fi
```

```
Append the following lines after that section:
```

```
if [ -f /etc/rc.modules ]; then
        /etc/rc.modules
fi
```

```
modprobe virtio
modprobe virtio_ring # Comment this out if you do not need block driver
modprobe virtio_blk  # Comment this out if you do not need block driver
modprobe virtio_net  # Comment this out if you do not need net driver
modprobe virtio_pci
```

Reboot the guest to load the kernel modules.

Adding the para-virtualized drivers to the initrd RAM disk

This procedure covers loading the para-virtualized driver modules with the kernel on a Red Hat Enterprise Linux 3.9 or newer guest by including the modules in the initrd RAM disk. The mkinitrd tool configures the initrd RAM disk to load the modules. Specify the additional modules with the `--with` parameter for the **`mkinitrd`** command. Append following set of parameters, in the exact order, when using the **`mkinitrd`** command to create a custom initrd RAM disk:

```
--with virtio --with virtio_ring --with virtio_blk --with virtio_net --with virtio_pci
```

AMD64 and Intel 64 issues

Use the **`x86_64`** version of the *virtio* package for AMD64 systems.

Use the **`ia32e`** version of the *virtio* package for Intel 64 systems. Using the **`x86_64`** version of the *virtio* may cause a '**`Unresolved symbol`**' error during the boot sequence on Intel 64 systems.

Network performance issues

If you experience low performance with the para-virtualized network drivers, verify the setting for the GSO and TSO features on the host system. The para-virtualized network drivers require that the GSO and TSO options are disabled for optimal performance.

Verify the status of the GSO and TSO settings, use the command on the host (replacing *`interface`* with the network interface used by the guest):

```
# ethtool -k interface
```

Disable the GSO and TSO options with the following commands on the host:

```
# ethtool -K interface gso off
# ethtool -K interface tso off
```

Para-virtualized driver swap partition issue

After activating the para-virtualized block device driver the swap partition may not be available. This issue is may be caused by a change in disk device name. To fix this issue, open the **`/etc/fstab`** file and locate the lines containing swap partitions, for example:

```
/dev/hda3  swap swap defaults 0 0
```

The para-virtualized drivers use the **`/dev/vd*`** naming convention, not the **`/dev/hd*`** naming convention. To resolve this issue modify the incorrect swap entries in the **`/etc/fstab`** file to use the **`/dev/vd*`** convention, for the example above:

```
/dev/vda3  swap swap defaults 0 0
```

Save the changes and reboot the virtualized guest. The guest should now correctly have swap partitions.

10.3. Using KVM para-virtualized drivers for existing devices

You can modify an existing hard disk device attached to the guest to use the **`virtio`** driver instead of the virtualized IDE driver. The example shown in this section edits libvirt configuration files. Note that the guest does not need to be shut down to perform these steps, however the change will not be applied until the guest is completely shut down and rebooted.

1. Ensure that you have installed the appropriate driver (**`viostor`**), as described in *Section 10.1, “Installing the KVM Windows para-virtualized drivers”*, before continuing with this procedure.

2. Run the **`virsh edit <guestname>`** command to edit the XML configuration file for your device. For example, **`virsh edit guest1`**. The configuration files are located in **`/etc/libvirt/qemu`**.

3. Below is a file-based block device using the virtualized IDE driver. This is a typical entry for a virtualized guest not using the para-virtualized drivers.

```
<disk type='file' device='disk'>
   <source file='/var/lib/libvirt/images/disk1.img'/>
   <target dev='hda' bus='ide'/>
</disk>
```

4. Change the entry to use the para-virtualized device by modifying the **bus=** entry to **`virtio`**. Note that if the disk was previously IDE it will have a target similar to hda, hdb, or hdc and so on. When changing to **bus=virtio** the target needs to be changed to vda, vdb, or vdc accordingly.

```
<disk type='file' device='disk'>
   <source file='/var/lib/libvirt/images/disk1.img'/>
   <target dev='vda' bus='virtio'/>
</disk>
```

5. Remove the **address** tag inside the **disk** tags. This must be done for this procedure to work. Libvirt will regenerate the **address** tag appropriately the next time the guest is started.

Alternatively, **`virt-manager`**, **`virsh attach-disk`** or **`virsh attach-interface`** can add a new device using the para-virtualized drivers.

Refer to the libvirt website for more details on using Virtio: *http://www.linux-kvm.org/page/Virtio*

10.4. Using KVM para-virtualized drivers for new devices

This procedure covers creating new devices using the KVM para-virtualized drivers with **`virt-manager`**.

Alternatively, the **`virsh attach-disk`** or **`virsh attach-interface`** commands can be used to attach devices using the para-virtualized drivers.

> **Install the drivers first**
>
> Ensure the drivers have been installed on the Windows guest before proceeding to install new devices. If the drivers are unavailable the device will not be recognized and will not work.

Procedure 10.3. Starting the new device wizard

1. Open the virtualized guest by double clicking on the name of the guest in `virt-manager`.
2. Open the **Information** tab by pressing the `i` information button.

   Figure 10.1. The information tab button

3. In the information tab, press the **Add Hardware** button.
4. In the Adding Virtual Hardware tab select **Storage** or **Network** for the type of device. The storage and network device wizards are covered in procedures *Procedure 10.4, "Adding a storage device using the para-virtualized storage driver"* and *Procedure 10.5, "Adding a network device using the para-virtualized network driver"*

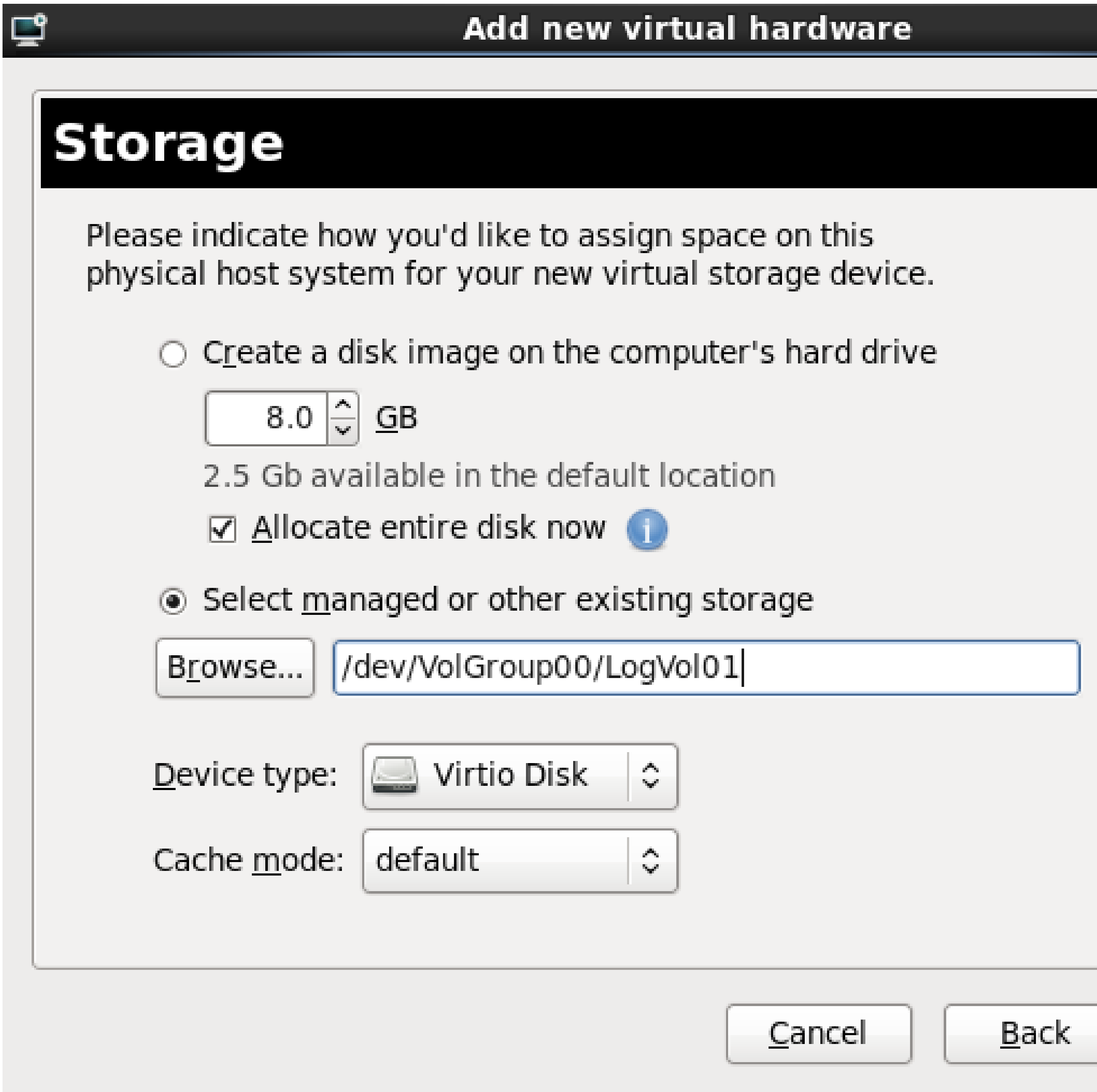
Procedure 10.4. Adding a storage device using the para-virtualized storage driver

1. **Select hardware type**
   Select **Storage** as the **Hardware type**.

Press **Forward** to continue.

2. **Select the storage device and driver**
   Create a new disk image or select a storage pool volume.

   Set the **Device type** to **Virtio Disk** to use the para-virtualized drivers.



Press **Forward** to continue.

3. **Finish the procedure**
   Confirm the details for the new device are correct.

Press **Finish** to complete the procedure.

Procedure 10.5. Adding a network device using the para-virtualized network driver

1. **Select hardware type**

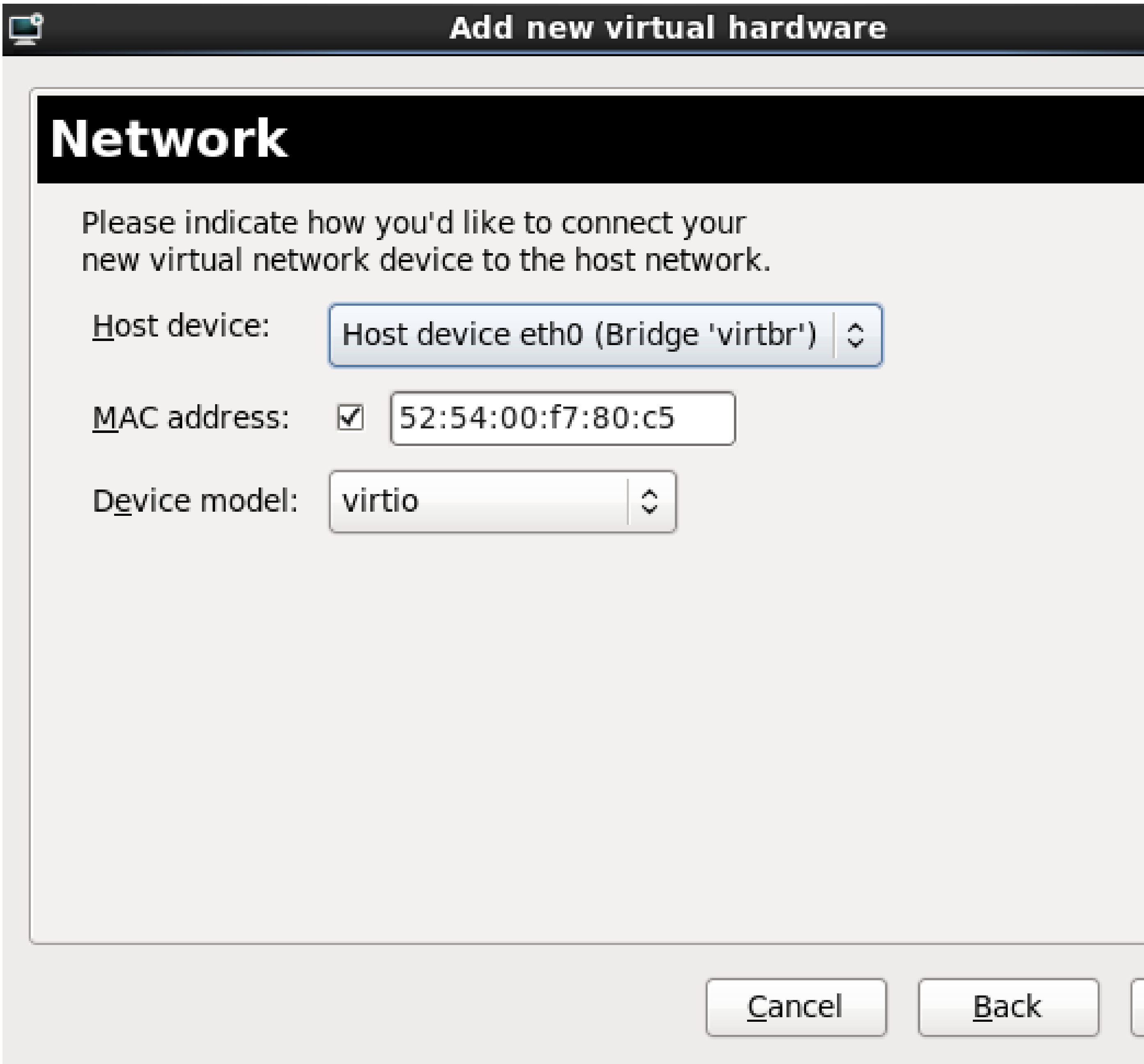
   Select **Network** as the **Hardware type**.

Press **Forward** to continue.

2. **Select the network device and driver**
   Select the network device from the **Host device** list.

   Create a custom MAC address or use the one provided.

   Set the **Device model** to **virtio** to use the para-virtualized drivers.



Press **Forward** to continue.

3. **Finish the procedure**
   Confirm the details for the new device are correct.

Press **Finish** to complete the procedure.

Once all new devices are added, reboot the guest. Windows guests may may not recognise the devices until the guest is rebooted.

Network Configuration

This chapter provides an introduction to the common networking configurations used by libvirt based guest VMs. For additional information consult the libvirt network architecture documentation: *http://libvirt.org/intro.html*.

Red Hat Enterprise Linux 6 supports the following networking setups for virtualization:

- virtual networks using Network Address Translation (*NAT*)
- directly allocated physical devices using PCI device assignment
- directly allocated virtual functions using PCIe SR-IOV
- bridged networks

You must enable NAT, network bridging or directly share a PCIe device to allow external hosts access to network services on virtualized guests.

11.1. Network Address Translation (NAT) with libvirt

One of the most common methods for sharing network connections is to use Network Address Translation (NAT) forwarding (also know as virtual networks).

Host configuration

Every standard `libvirt` installation provides NAT based connectivity to virtual machines as the default virtual network. Verify that it is available with the **`virsh net-list --all`** command.

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

If it is missing, the example XML configuration file can be reloaded and activated:

```
# virsh net-define /usr/share/libvirt/networks/default.xml
```

The default network is defined from **`/usr/share/libvirt/networks/default.xml`**

Mark the default network to automatically start:

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

Start the default network:

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

Once the `libvirt` default network is running, you will see an isolated bridge device. This device does *not* have any physical interfaces added. The new device uses NAT and IP forwarding to connect to the physical network. Do not add new interfaces.

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
```

```
virbr0          8000.000000000000       yes
```

libvirt adds **iptables** rules which allow traffic to and from guests attached to the `virbr0` device in the **INPUT**, **FORWARD**, **OUTPUT** and **POSTROUTING** chains. **libvirt** then attempts to enable the **ip_forward** parameter. Some other applications may disable **ip_forward**, so the best option is to add the following to **/etc/sysctl.conf**.

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

Guest configuration

Once the host configuration is complete, a guest can be connected to the virtual network based on its name. To connect a guest to the 'default' virtual network, the following could be used in the XML configuration file (such as **/etc/libvirtd/qemu/myguest.xml**) for the guest:

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

> **Note**
>
> Defining a MAC address is optional. A MAC address is automatically generated if omitted. Manually setting the MAC address may be useful to maintain consistency or easy reference throughout your environment, or to avoid the very small chance of a conflict.
>
> ```
> <interface type='network'>
>   <source network='default'/>
>   <mac address='00:16:3e:1a:b3:4a'/>
> </interface>
> ```

11.2. Disabling vhost-net

The **vhost-net** module is a kernel-level backend for virtio networking that reduces virtualization overhead by moving virtio descriptor conversion tasks out of the qemu user space and into the **vhost-net** driver. It is enabled by default when virtio is not in use, and when no other hardware acceleration for virtual networking is in place. It is disabled by default in all other situations, because some workloads can experience a degradation in performance when **vhost-net** is in use.

Specifically, when UDP traffic is sent from a host machine to a guest on that host, performance degradation can occur if the guest machine processes incoming data at a rate slower than the host machine sends it. In this situation, enabling **vhost-net** causes the UDP socket's receive buffer to overflow more quickly, which results in greater packet loss. It is therefore better to disable **vhost-net** in this situation to slow the traffic, and improve overall performance.

To disable **vhost-net**, edit the **<interface>** sub-element in your guest's XML configuration file and define the network as follows:

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

```
</interface>
```

Because virtio is enabled, vhost-net will not be used.

11.2.1. Checksum correction for older DHCP clients

The **`vhost-net`** module avoids using checksum computations in host to guest communication, and notifies guests that incoming packets do not include the checksum.

Some DHCP clients do not work correctly on guest machines running Red Hat Enterprise Linux versions earlier than 6.0 because of this "missing" checksum value. When used with other DHCP clients on the same host that were not started by libvirt, the DHCP client assumes that the checksum value for these packets is incorrect rather than missing, and discards those packets.

To work around this problem, you can create an iptables CHECKSUM target to compute and fill in the checksum value in packets that lack checksums, for example:

```
iptables -A POSTROUTING -t mangle -p udp --dport 68 -j CHECKSUM --checksum-fill
```

This iptables rule is programmed automatically on the host when the server is started by libvirt, so no further action is required.

11.3. Bridged networking with libvirt

Bridged networking (also known as physical device sharing) is used for dedicating a physical device to a virtual machine. Bridging is often used for more advanced setups and on servers with multiple network interfaces.

Disable NetworkManager

NetworkManager does not support bridging. NetworkManager must be disabled to use networking with the network scripts (located in the **`/etc/sysconfig/network-scripts/`** directory).

```
# chkconfig NetworkManager off
# chkconfig network on
# service NetworkManager stop
# service network start
```

> **Note**
>
> Instead of turning off **NetworkManager**, add "*`NM_CONTROLLED=no`*" to the **`ifcfg-*`** scripts used in the examples.

Host configuration

Create or edit the following two network configuration files. These steps can be repeated (with different names) for additional network bridges.

1. **Change to the network scripts directory**

   Change to the **`/etc/sysconfig/network-scripts`** directory:

   ```
   # cd /etc/sysconfig/network-scripts
   ```

2. **Modify a network interface to make a bridge**
   Edit the network script for the network device you are adding to the bridge. In this example, **`/etc/sysconfig/network-scripts/ifcfg-eth0`** is used. This file defines **`eth0`**, the physical network interface which is set as part of a bridge:

```
DEVICE=eth0
# change the hardware address to match the hardware address your NIC uses
HWADDR=00:16:76:D6:C9:45
ONBOOT=yes
BRIDGE=br0
```

Tip

You can configure the device's Maximum Transfer Unit (MTU) by appending an *`MTU`* variable to the end of the configuration file.

```
MTU=9000
```

3. **Create the bridge script**
   Create a new network script in the **`/etc/sysconfig/network-scripts`** directory called **`ifcfg-br0`** or similar. The *`br0`* is the name of the bridge, this can be anything as long as the name of the file is the same as the DEVICE parameter, and that it matches the bridge name used in step 2.

```
DEVICE=br0
TYPE=Bridge
BOOTPROTO=dhcp
ONBOOT=yes
DELAY=0
```

Warning

The line, *`TYPE=Bridge`*, is case-sensitive. It must have uppercase 'B' and lower case 'ridge'.

4. **Restart the network**
   After configuring, restart networking or reboot.

```
# service network restart
```

5. **Configure iptables**
   Configure **`iptables`** to allow all traffic to be forwarded across the bridge.

```
# iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
# service iptables save
```

```
# service iptables restart
```

Disable iptables on bridges

Alternatively, prevent bridged traffic from being processed by **`iptables`** rules. In **`/etc/sysctl.conf`** append the following lines:

```
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

Reload the kernel parameters configured with **`sysctl`**.

```
# sysctl -p /etc/sysctl.conf
```

6. **Verify the bridge**
   Verify the new bridge is available with the bridge control command (**`brctl`**).

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.000000000000       yes
br0             8000.000e0cb30550       no              eth0
```

A **Shared physical device** is now available through **`virt-manager`** and libvirt. Guests can now connect to this device for full network access.

Note, the bridge is completely independent of the **`virbr0`** bridge. Do *not* attempt to attach a physical device to **`virbr0`**. The **`virbr0`** bridge is only for Network Address Translation (NAT) connectivity.

Guest configuration

Once the host configuration is complete, a guest can be connected to the virtual network based on its name. To connect a guest to the bridged virtual network, the following could be used in the XML configuration file (such as **`/etc/libvirtd/qemu/myguest.xml`**) for the guest:

```
<interface type='bridge'>
   <source bridge='br0'/>
</interface>
```

PCI device assignment

This chapter covers using PCI device assignment with KVM.

Certain hardware platforms allow virtualized guests to directly access various hardware devices and components. This process in virtualization is known as *device assignment*. Device assignment is also known as *PCI passthrough*.

The KVM hypervisor supports attaching PCI devices on the host system to virtualized guests. PCI device assignment allows guests to have exclusive access to PCI devices for a range of tasks. PCI device assignment allows PCI devices to appear and behave as if they were physically attached to the guest operating system. PCI device assignment can improve the I/O performance of devices attached to virtualized guests.

Device assignment is supported on PCI Express devices, except graphics cards. Parallel PCI devices may be supported as assigned devices, but they have severe limitations due to security and system configuration conflicts.

Note

The Red Hat Enterprise Linux 6.0 release comes with limitations for operating system drivers of KVM guests to have full access to a device's standard and extended configuration space. The Red Hat Enterprise Linux 6.0 limitations have been significantly reduced in the Red Hat Enterprise Linux 6.1 release and enable a much larger set of PCI Express devices to be successfully assigned to KVM guests.

Secure device assignment also requires interrupt remapping support. If a platform does not support interrupt remapping, device assignment will fail. To use device assignment without interrupt remapping support in a development environment, set the `allow_unsafe_assigned_interrupts` KVM module parameter to **1**.

PCI device assignment is only available on hardware platforms supporting either Intel VT-d or AMD IOMMU. These Intel VT-d or AMD IOMMU extensions must be enabled in BIOS for PCI device assignment to function.

Red Hat Enterprise Linux 6.0 and newer supports hot plugging PCI device assignment devices into virtualized guests.

PCI devices are limited by the virtualized system architecture. Out of the 32 PCI devices for a guest, 4 are always defined for a KVM guest, and are not removable. This means there are up to 28 PCI slots available for additional devices per guest. Each PCI device in a guest can have up to 8 functions.

Procedure 12.1. Preparing an Intel system for PCI device assignment

1. **Enable the Intel VT-d extensions**

   The Intel VT-d extensions provides hardware support for directly assigning a physical devices to guest.

   The VT-d extensions are required for PCI device assignment with Red Hat Enterprise Linux. The extensions must be enabled in the BIOS. Some system manufacturers disable these extensions by default.

   These extensions are often called various terms in BIOS which differ from manufacturer to manufacturer. Consult your system manufacturer's documentation.

2. **Ready to use**
   Reboot the system to enable the changes. Your system is now PCI device assignment capable.

Procedure 12.2. Preparing an AMD system for PCI

1. **Enable AMD IOMMU extensions**
   The AMD IOMMU extensions are required for PCI device assignment with Red Hat Enterprise Linux. The extensions must be enabled in the BIOS. Some system manufacturers disable these extensions by default.

2. **Enable IOMMU kernel suppport**
   Append *amd_iommu=on* to the kernel command line so that AMD IOMMU extensions are enabled at boot.

12.1. Adding a PCI device with virsh

These steps cover adding a PCI device to a virtualized guest on a KVM hypervisor using hardware-assisted PCI device assignment.

This example uses a PCIe network controller with the PCI identifier code, **pci_0000_01_00_0**, and a fully virtualized guest named *guest1-rhel6-64*.

1. **Identify the device**
   Identify the PCI device designated for device assignment to the guest.

```
# lspci | grep Ethernet
00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit Network Connection
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
```

   The **virsh nodedev-list** command lists all devices attached to the system, and identifies each PCI device with a string. To limit output to only PCI devices, run the following command:

```
# virsh nodedev-list | grep pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
```

```
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

Record the PCI device number; the number is needed in other steps.

2. Information on the domain, bus and function are available from output of the **`virsh nodedev-dumpxml`** command:

```
# virsh nodedev-dumpxml pci_0000_01_00_0
<device>
  <name>pci_0000_01_00_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>1</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
    </capability>
  </capability>
</device>
```

3. Convert slot and function values to hexadecimal values (from decimal) to get the PCI bus addresses. Append "0x" to the beginning of the output to tell the computer that the value is a hexadecimal number.

   For example, if bus = 0, slot = 26 and function = 7 run the following:

```
$ printf %x 0
0
$ printf %x 26
1a
$ printf %x 7
7
```

   The values to use:

```
bus='0x00'
slot='0x1a'
function='0x7'
```

4. Run **`virsh edit`** (or virsh attach device) and add a device entry in the **`<devices>`** section to attach the PCI device to the guest.

```
# virsh edit guest1-rhel6-64
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
     <address domain='0x000' bus='0x01' slot='0x00' function='0x0'/>
  </source>
```

```
</hostdev>
```

5. Set a sebool to allow the management of the PCI device from the guest:

```
$ setsebool -P virt_use_sysfs 1
```

6. Start the guest system:

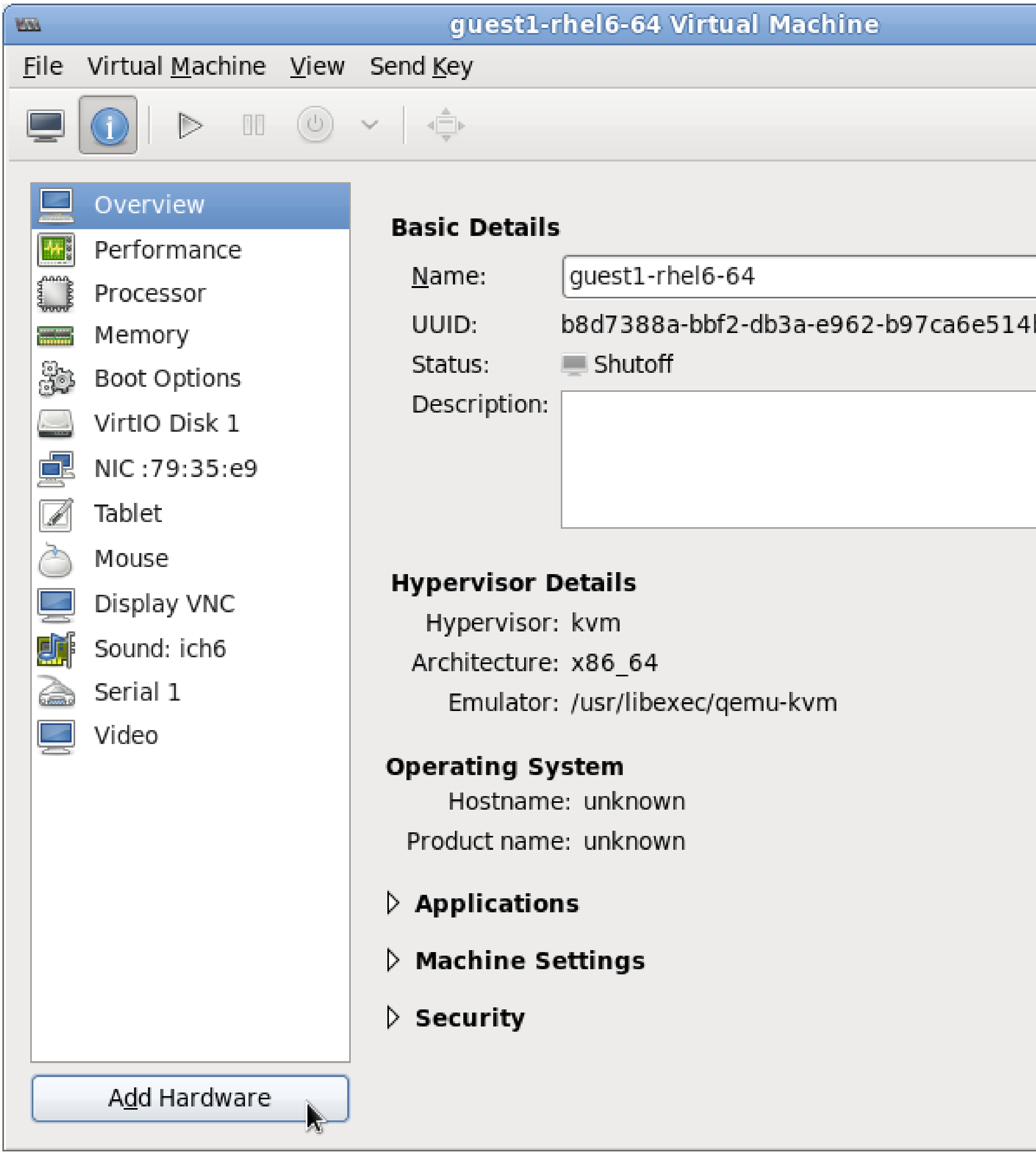
```
# virsh start guest1-rhel6-64
```

The PCI device should now be successfully attached to the guest and accessible to the guest operating system.

12.2. Adding a PCI device with virt-manager

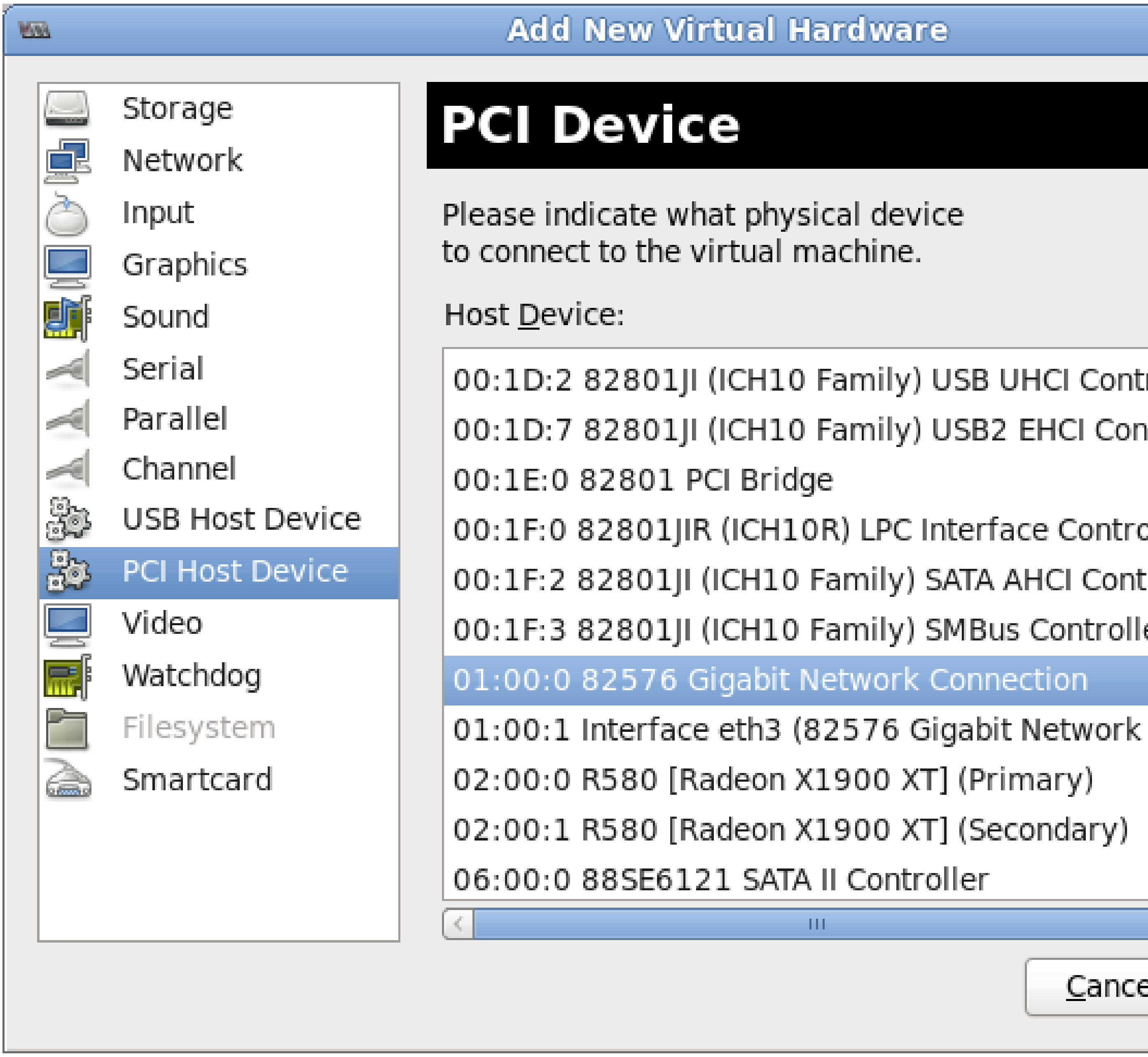
PCI devices can be added to guests using the graphical **`virt-manager`** tool. The following procedure adds a 2 port USB controller to a virtualized guest.

1. **Open the hardware settings**
   Open the virtual machine and click the **Add Hardware** button to add a new device to the guest.

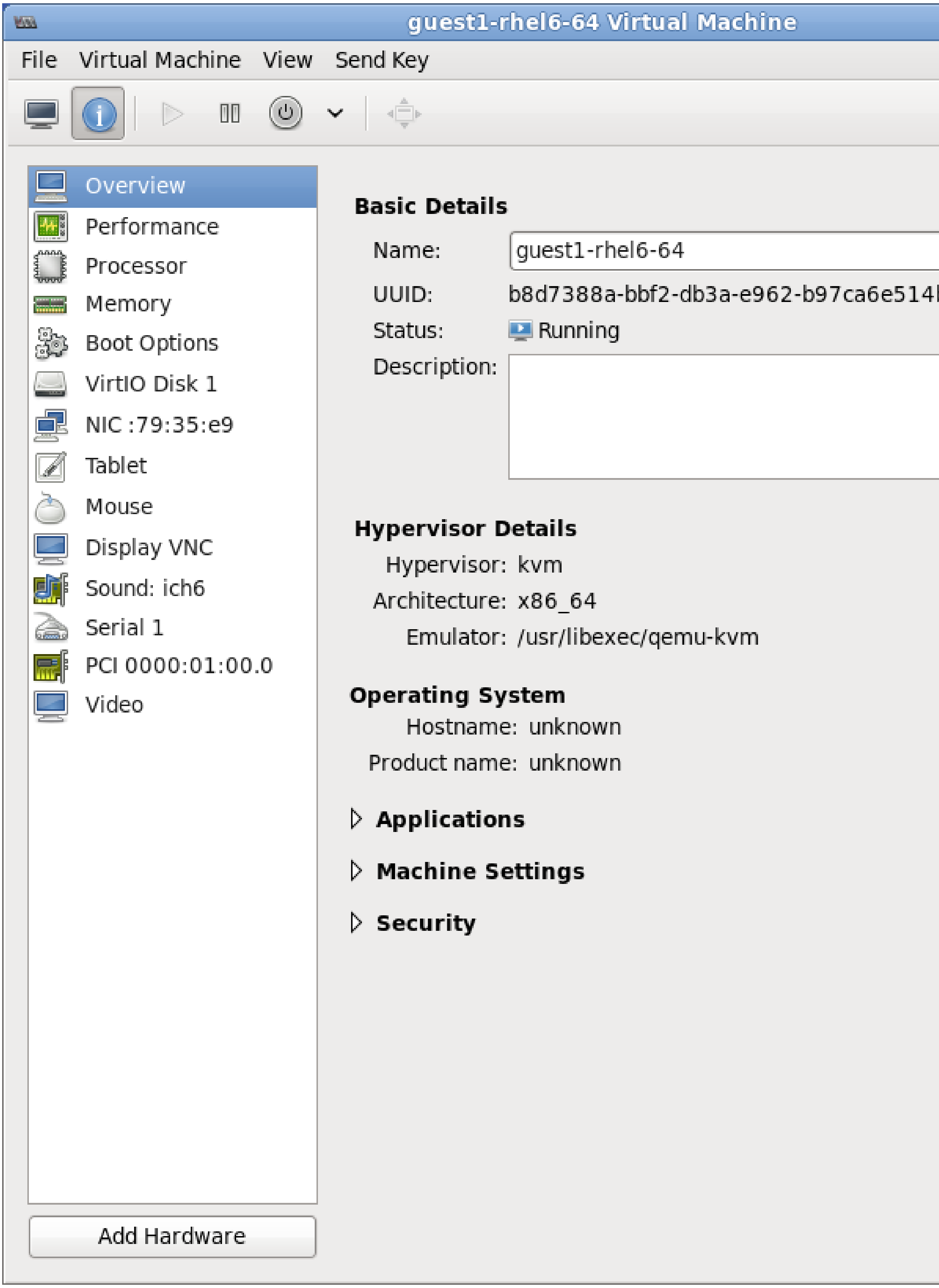


2. **Select a PCI device**
   Select **PCI Host Device** from the **Hardware** list on the left.

   Select an unused PCI device. Note that selecting PCI devices presently in use on the host causes errors. In this example, a spare 82576 network device is used. Click **Finish** to complete setup.



3. **Add the new device**
   The setup is complete and the guest can now use the PCI device.


guest1-rhel6-64 Virtual Machine
File
Virtual Machine
View
Send Key
Overview
Performance
Processor
Memory
Boot Options
VirtIO Disk 1
NIC :79:35:e9
Tablet
Mouse
Display VNC
Sound: ich6
Serial 1
PCI 0000:01:00.0
Video
Add Hardware
Basic Details
Name: guest1-rhel6-64
UUID: b8d7388a-bbf2-db3a-e962-b97ca6e514b
Status: Running
Description:
Hypervisor Details
Hypervisor: kvm
Architecture: x86_64
Emulator: /usr/libexec/qemu-kvm
Operating System
Hostname: unknown
Product name: unknown
Applications
Machine Settings
Security

12.3. PCI device assignment with virt-install

To use PCI device assignment with the virt-install parameter, use the additional *--host-device* parameter.

1. **Identify the device**

   Identify the PCI device designated for device assignment to the guest.

```
# lspci | grep Ethernet
00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit Network Connection
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
```

   The **virsh nodedev-list** command lists all devices attached to the system, and identifies each PCI device with a string. To limit output to only PCI devices, run the following command:

```
# virsh nodedev-list | grep pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

   Record the PCI device number; the number is needed in other steps.

   Information on the domain, bus and function are available from output of the **virsh nodedev-dumpxml** command:

```
# virsh nodedev-dumpxml pci_0000_01_00_0
<device>
  <name>pci_0000_01_00_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
```

```
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>1</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
    </capability>
  </capability>
</device>
```

2. **Add the device**
   Use the PCI identifier output from the **`virsh nodedev`** command as the value for the *`--host-device`* parameter.

```
virt-install \
--name=guest1-rhel6-64 \
--disk path=/var/lib/libvirt/images/guest1-rhel6-64.img,size=8 \
--nonsparse --vnc \
--vcpus=2 --ram=2048 \
--location=http://example1.com/installation_tree/RHEL6.1-Server-x86_64/os \
--nonetworks \
--os-type=linux \
--os-variant=rhel6
--host-device=pci_0000_01_00_0
```

3. **Complete the installation**
   Complete the guest installation. The PCI device should be attached to the guest.

SR-IOV

13.1. Introduction

Developed by the PCI-SIG (PCI Special Interest Group), the Single Root I/O Virtualization (SR-IOV) specification is a standard for a type of PCI device assignment that can share a single device to multiple guests. SR-IOV improves device performance for virtualized guests.

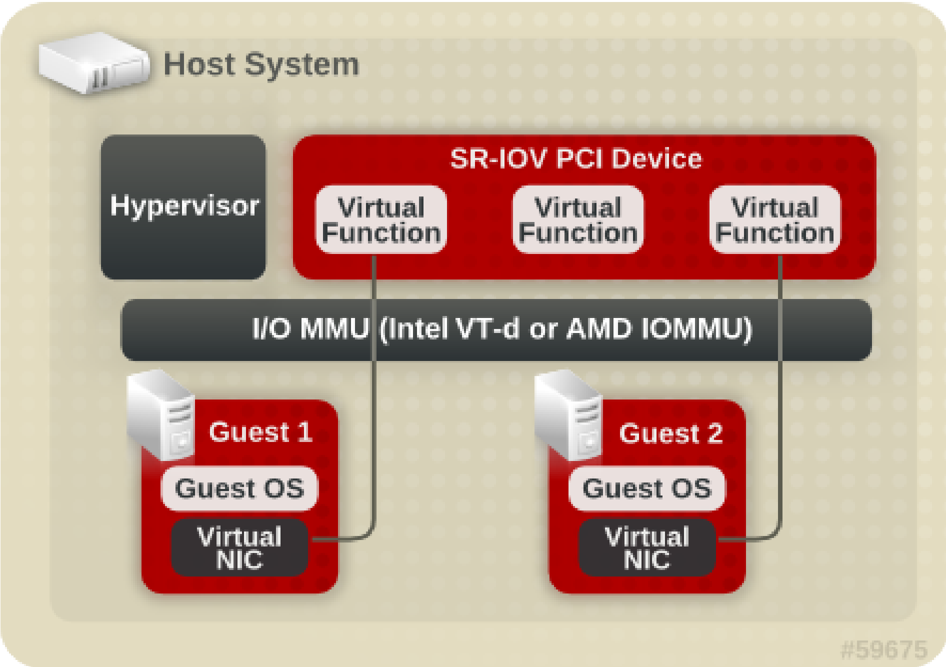


Figure 13.1. How SR-IOV works

SR-IOV enables a Single Root Function (for example, a single Ethernet port), to appear as multiple, separate, physical devices. A physical device with SR-IOV capabilities can be configured to appear in the PCI configuration space as multiple functions. Each device has its own configuration space complete with Base Address Registers (BARs).

SR-IOV uses two PCI functions:

- Physical Functions (PFs) are full PCIe devices that include the SR-IOV capabilities. Physical Functions are discovered, managed, and configured as normal PCI devices. Physical Functions configure and manage the SR-IOV functionality by assigning Virtual Functions.
- Virtual Functions (VFs) are simple PCIe functions that only process I/O. Each Virtual Function is derived from a Physical Function. The number of Virtual Functions a device may have is limited by the device hardware. A single Ethernet port, the Physical Device, may map to many Virtual Functions that can be shared to virtualized guests.

The hypervisor can map one or more Virtual Functions to a virtualized guest. The Virtual Function's configuration space is then mapped to the configuration space presented to the virtualized guest.

Each Virtual Function can only be mapped to a single guest at a time, as Virtual Functions require real hardware resources. A virtualized guest can have multiple Virtual Functions. A Virtual Function appears as a network card in the same way as a normal network card would appear to an operating system.

The SR-IOV drivers are implemented in the kernel. The core implementation is contained in the PCI subsystem, but there must also be driver support for both the Physical Function (PF) and Virtual Function (VF) devices. With an SR-IOV capable device one can allocate VFs from a PF. The VFs appear as PCI devices which are backed on the physical PCI device by resources (queues, and register sets).

Advantages of SR-IOV

SR-IOV devices can share a single physical port with multiple virtualized guests.

Virtual Functions have near-native performance and provide better performance than para-virtualized drivers and emulated access. Virtual Functions provide data protection between virtualized guests on the same physical server as the data is managed and controlled by the hardware.

These features allow for increased virtualized guest density on hosts within a data center.

SR-IOV is better able to utilize the bandwidth of devices with multiple guests.

13.2. Using SR-IOV

This section covers attaching Virtual Function to a guest as an additional network device.

SR-IOV requires Intel VT-d support.

Procedure 13.1. Attach an SR-IOV network device

1. **Enable Intel VT-d in BIOS and in the kernel**

   Skip this step if Intel VT-d is already enabled and working.

   Enable Intel VT-d in BIOS if it is not enabled already. Refer to *Procedure 12.1, “Preparing an Intel system for PCI device assignment”* for procedural help on enabling Intel VT-d in BIOS and the kernel.

2. **Verify support**

   Verify if the PCI device with SR-IOV capabilities are detected. This example lists an Intel 82576 network interface card which supports SR-IOV. Use the **lspci** command to verify if the device was detected.

   ```
   # lspci
   03:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
   03:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
   ```

   Note that the output has been modified to remove all other devices.

3. **Start the SR-IOV kernel modules**

   If the device is supported the driver kernel module should be loaded automatically by the kernel. Optional parameters can be passed to the module using the **modprobe** command. The Intel 82576 network interface card uses the **igb** driver kernel module.

   ```
   # modprobe igb [<option>=<VAL1>,<VAL2>,]
   # lsmod |grep igb
   igb    87592  0
   dca    6708    1 igb
   ```

4. **Activate Virtual Functions**
   The *max_vfs* parameter of the **igb** module allocates the maximum number of Virtual Functions. The *max_vfs* parameter causes the driver to spawn, up to the value of the parameter in, Virtual Functions. For this particular card the valid range is *0* to *7*.

   Remove the module to change the variable.

   ```
   # modprobe -r igb
   ```

   Restart the module with the *max_vfs* set to *1* or any number of Virtual Functions up to the maximum supported by your device.

   ```
   # modprobe igb max_vfs=7
   ```

5. **Make the Virtual Functions persistent**
   The **modprobe** command /etc/modprobe.d/igb.conf options igb max_vfs=7

6. **Inspect the new Virtual Functions**
   Using the **lspci** command, list the newly added Virtual Functions attached to the Intel 82576 network device.

   ```
   # lspci | grep 82576
   0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
   0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
   0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:10.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:10.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:10.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:10.6 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:11.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:11.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:11.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:11.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   0b:11.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   ```

   The identifier for the PCI device is found with the *-n* parameter of the **lspci** command. The Physical Functions corresponds to **0b:00.0** and **0b:00.1**. All the Virtual Functions have **Virtual Function** in the description.

7. **Verify devices exist with virsh**
   The `libvirt` service must recognize the device before adding a device to a guest. `libvirt` uses a similar notation to the **lspci** output. All punctuation characters, **;** and **.**, in lspci output are changed to underscores (_).

   Use the **virsh nodedev-list** command and the grep command to filter the Intel 82576 network device from the list of available host devices. *0b* is the filter for the Intel 82576 network devices in this example. This may vary for your system and may result in additional devices.

   ```
   # virsh nodedev-list | grep 0b
   pci_0000_0b_00_0
   pci_0000_0b_00_1
   pci_0000_0b_10_0
   pci_0000_0b_10_1
   ```

```
pci_0000_0b_10_2
pci_0000_0b_10_3
pci_0000_0b_10_4
pci_0000_0b_10_5
pci_0000_0b_10_6
pci_0000_0b_11_7
pci_0000_0b_11_1
pci_0000_0b_11_2
pci_0000_0b_11_3
pci_0000_0b_11_4
pci_0000_0b_11_5
```

The serial numbers for the Virtual Functions and Physical Functions should be in the list.

8. **Get device details with virsh**
The **`pci_0000_0b_00_0`** is one of the Physical Functions and **`pci_0000_0b_10_0`** is the first corresponding Virtual Function for that Physical Function. Use the **`virsh nodedev-dumpxml`** command to get advanced output for both devices.

```
# virsh nodedev-dumpxml pci_0000_0b_00_0
<device>
    <name>pci_0000_0b_00_0</name>
    <parent>pci_0000_00_01_0</parent>
    <driver>
      <name>igb</name>
    </driver>
    <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>0</slot>
      <function>0</function>
      <product id='0x10c9'>Intel Corporation</product>
      <vendor id='0x8086'>82576 Gigabit Network Connection</vendor>
    </capability>
  </device>
```

```
# virsh nodedev-dumpxml pci_0000_0b_10_0
  <device>
    <name>pci_0000_0b_10_0</name>
    <parent>pci_0000_00_01_0</parent>
    <driver>
      <name>igbvf</name>
    </driver>
    <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>16</slot>
      <function>0</function>
      <product id='0x10ca'>Intel Corporation</product>
      <vendor id='0x8086'>82576 Virtual Function</vendor>
    </capability>
  </device>
```

This example adds the Virtual Function `pci_0000_0b_10_0` to the guest in *Step 10*. Note the **`bus`**, **`slot`** and **`function`** parameters of the Virtual Function, these are required for adding the device.

9. **Detach the Virtual Functions**
Devices attached to a host cannot be attached to guests. Red Hat Enterprise Linux automatically attaches new devices to the host. Detach the Virtual Function from the host so that the Virtual Function can be used by the guest. Detaching a Physical Function causes errors when the same

Virtual Functions are already assigned to guests. Ensure that you only detach the required Virtual Functions.

```
# virsh nodedev-dettach pci_0000_0b_10_0
Device pci_0000_0b_10_0 dettached
```

10. **Add the Virtual Function to the guest**
    a. Shut down the guest.

    b. Use the output from the **`virsh nodedev-dumpxml pci_0000_0b_10_0`** command to calculate the values for the configuration file. Convert slot and function values to hexadecimal values (from decimal) to get the PCI bus addresses. Append "0x" to the beginning of the output to tell the computer that the value is a hexadecimal number.

    The example device has the following values: bus = 3, slot = 16 and function = 1. Use the **`printf`** utility to convert decimal values to hexadecimal values.

    ```
    $ printf %x 3
    3
    $ printf %x 16
    10
    $ printf %x 1
    1
    ```

    This example would use the following values in the configuration file:

    ```
    bus='0x0b'
    slot='0x10'
    function='0x01'
    ```

    c. Open the XML configuration file with the **`virsh edit`** command. This example edits a guest named *`MyGuest`*.

    ```
    # virsh edit MyGuest
    ```

    d. The default text editor will open the libvirt configuration file for the guest. Add the new device to the **`devices`** section of the XML configuration file.

    ```
    <hostdev mode='subsystem' type='pci' managed='yes'>
          <source>
            <address bus='0x0b' slot='0x10' function='0x01'/>
          </source>
    </hostdev>
    ```

    e. Save the configuration.

11. **Restart**
    Restart the guest to complete the installation.

    ```
    # virsh start MyGuest
    ```

The guest should start successfully and detect a new network interface card. This new card is the Virtual Function of the SR-IOV device.

13.3. Troubleshooting SR-IOV

This section contains solutions for problems which may affect SR-IOV.

Error starting the guest

When starting a configured virtual machine, an error occurs as follows:

```
# virsh start test
error: Failed to start domain test
error: internal error unable to start guest: char device redirected to
/dev/pts/2
get_real_device: /sys/bus/pci/devices/0000:03:10.0/config: Permission denied
init_assigned_device: Error: Couldn't get real device (03:10.0)!
Failed to initialize assigned device host=03:10.0
```

This error is often caused by a device that is already assigned to another guest or to the host itself.

KVM guest timing management

Virtualization involves several intrinsic challenges for time keeping in guests. Interrupts cannot always be delivered simultaneously and instantaneously to all guest virtual CPUs, because interrupts in virtual machines are not true interrupts; they are injected into the guest by the host machine. The host may be running another guest virtual CPU, or a different process, meaning that the precise timing typically required by interrupts may not always be possible.

Guests without accurate time keeping may experience issues with network applications and processes, as session validity, migration, and other network activities rely on timestamps to remain correct.

KVM avoids these issues by providing guests with a para-virtualized clock (**kvm-clock**). However, it is still vital to test timing before attempting activities that may be affected by time keeping inaccuracies.

kvm-clock

Red Hat Enterprise Linux 5.5 and newer, and Red Hat Enterprise Linux 6.0 and newer, use **kvm-clock** as their default clock source. Running without **kvm-clock** requires special configuration, and is not recommended.

NTP

The Network Time Protocol (NTP) daemon should be running on the host and the guests. Enable the `ntpd` service:

```
# service ntpd start
```

Add the ntpd service to the default startup sequence:

```
# chkconfig ntpd on
```

Using the `ntpd` service will minimize the effects of clock skew as long as the skew is less than or equal to 500 millionths of a second (0.0005 seconds).

Constant Time Stamp Counter (TSC)

Modern Intel and AMD CPUs provide a constant Time Stamp Counter (TSC). The count frequency of the constant TSC does not vary when the CPU core itself changes frequency, for example, to comply with a power saving policy. A CPU with a constant TSC frequency is necessary in order to use the TSC as a clock source for KVM guests.

Your CPU has a constant Time Stamp Counter if the **`constant_tsc`** flag is present. To determine if your CPU has the **`constant_tsc`** flag run the following command:

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

If any output is given your CPU has the **constant_tsc** bit. If no output is given follow the instructions below.

Configuring hosts without a constant Time Stamp Counter

Systems without a constant TSC frequency cannot use the TSC as a clock source for virtual machines, and require additional configuration. Power management features interfere with accurate time keeping and must be disabled for guests to accurately keep time with KVM.

> **Note**
>
> These instructions are for AMD revision F CPUs only.

If the CPU lacks the **constant_tsc** bit, disable all power management features (*BZ#513138*[1]). Each system has several timers it uses to keep time. The TSC is not stable on the host, which is sometimes caused by **cpufreq** changes, deep C state, or migration to a host with a faster TSC. Deep C sleep states can stop the TSC. To prevent the kernel using deep C states append **processor.max_cstate=1** to the kernel boot options in the **grub.conf** file on the host:

```
title Red Hat Enterprise Linux (2.6.32-36.x86-64)
        root (hd0,0)
 kernel /vmlinuz-2.6.32-36.x86-64 ro root=/dev/VolGroup00/LogVol00 rhgb quiet \
   processor.max_cstate=1
```

Disable **cpufreq** (only necessary on hosts without the **constant_tsc**) by editing the **/etc/sysconfig/cpuspeed** configuration file and change the **MIN_SPEED** and **MAX_SPEED** variables to the highest frequency available. Valid limits can be found in the **/sys/devices/system/cpu/cpu*/cpufreq/scaling_available_frequencies** files.

Required parameters for Red Hat Enterprise Linux guests

For certain Red Hat Enterprise Linux guests, additional kernel parameters are required. These parameters can be set by appending them to the end of the /kernel line in the /boot/grub/grub.conf file of the guest.

The table below lists versions of Red Hat Enterprise Linux and the parameters required on the specified systems.

Red Hat Enterprise Linux version	Additional guest kernel parameters
6.0 AMD64/Intel 64 with the para-virtualized clock	Additional parameters are not required
6.0 AMD64/Intel 64 without the para-virtualized clock	notsc lpj=*n*
5.5 AMD64/Intel 64 with the para-virtualized clock	divider=10[2]
5.5 AMD64/Intel 64 without the para-virtualized clock	divider=10 notsc lpj=*n*

[1] https://bugzilla.redhat.com/show_bug.cgi?id=513138

Red Hat Enterprise Linux version	Additional guest kernel parameters
5.5 x86 with the para-virtualized clock	Additional parameters are not required
5.5 x86 without the para-virtualized clock	divider=10 clocksource=acpi_pm lpj=*n*
5.4 AMD64/Intel 64	divider=10 notsc
5.4 x86	divider=10 clocksource=acpi_pm
5.3 AMD64/Intel 64	divider=10 notsc
5.3 x86	divider=10 clocksource=acpi_pm
4.8 AMD64/Intel 64	notsc divider=10
4.8 x86	clock=pmtmr divider=10
3.9 AMD64/Intel 64	Additional parameters are not required
3.9 x86	Additional parameters are not required

When to set the *`divider`* parameter

The *`divider`* kernel parameter divides the kernel Hertz rate by the value supplied (in this case, **`10`**). For interrupt-based time keepers, it can reduce timer interrupt load by a factor of ten. This improves performance on guests that are generally idle (do not require responsiveness), and can lower the required guest density.

Providing that interrupts can be delivered at a sufficient rate, this parameter need not be set on systems with high guest density, or for guests running applications that expect high responsiveness.

Red Hat Enterprise Linux 6 does not have a fixed-frequency clock interrupt; instead, it uses the timer dynamically as required. This is known as *tickless mode*. As such, the *`divider`* is not useful for Red Hat Enterprise Linux 6.

The *`lpj`* parameter

The *`lpj`* parameter requires a numeric value equal to the *loops per jiffy* value of the specific CPU on which the guest runs. If you do not know this value, do not set the *`lpj`* parameter.

Using the Real-Time Clock with Windows Server 2003 and Windows XP guests

Windows uses the both the Real-Time Clock (RTC) and the Time Stamp Counter (TSC). For Windows guests the Real-Time Clock can be used instead of the TSC for all time sources which resolves guest timing issues.

To enable the Real-Time Clock for the **`PMTIMER`** clock source (the **`PMTIMER`** usually uses the TSC), add the following option to the Windows boot settings. Windows boot settings are stored in the boot.ini file. Add the following option to the end of the Windows boot line in the **`boot.ini`** file:

```
/usepmtimer
```

For more information on Windows boot settings and the usepmtimer option, refer to *Available switch options for the Windows XP and the Windows Server 2003 Boot.ini files*[3].

Using the Real-Time Clock with Windows Vista, Windows Server 2008 and Windows 7 guests

Windows uses the both the Real-Time Clock (RTC) and the Time Stamp Counter (TSC). For Windows guests the Real-Time Clock can be used instead of the TSC for all time sources which resolves guest timing issues.

The **`boot.ini`** file is no longer used from Windows Vista and newer. Windows Vista, Windows Server 2008 and Windows 7 use the **Boot Configuration Data Editor** (**`bcdedit.exe`**) to modify the Windows boot parameters.

This procedure is only required if the guest is having time keeping issues. Time keeping issues may not affect guests on all host systems.

1. Open the Windows guest.

2. Open the **Accessories** menu of the **start** menu. Right click on the **Command Prompt** application, select **Run as Administrator**.

3. Confirm the security exception, if prompted.

4. Set the boot manager to use the platform clock. This should instruct Windows to use the PM timer for the primary clock source. The system UUID (*`{default}`*) in the example below) should be changed if the system UUID is different than the default boot device.

   ```
   C:\Windows\system32>bcdedit /set {default} USEPLATFORMCLOCK on
   The operation completed successfully
   ```

This fix should improve time keeping for Windows Vista, Windows Server 2008 and Windows 7 guests.

[3] http://support.microsoft.com/kb/833721

Network booting with libvirt

Guests can be booted with PXE enabled. PXE allows guests to boot and load their configuration off the network itself. This section demonstrates some basic configuration steps to configure PXE guests with libvirt.

Note

This section does not cover the creation of boot images or PXE servers. It is used to explain how to configure libvirt, in a private or bridged network, to boot a guest with PXE booting enabled.

Warning

These procedures are provided only as an example. Make sure you have sufficient backups before proceeding.

15.1. Preparing the boot server

To perform the steps in this chapter you will need:

- A PXE Server (DHCP and TFTP) - This can be a libvirt internal server, manually-configured dhcpd and tftpd, dnsmasq, a server configured by Cobbler, or other server.
- Boot images - for example, PXELINUX configured manually or by Cobbler.

15.1.1. Setting up a PXE boot server on a private libvirt network

This example uses the *default* network. Perform the following steps:

1. Place the PXE boot images and configuration in **`/var/lib/tftp`**.

2. Run the following commands:

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

3. Edit the <ip> element in the configuration file for the *default* network to look like the following. Note that the IP addresses and subnet masks displayed in this output are only examples, and should be replaced with settings to match your environment.

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

BOOT_FILENAME in the previous output should be replaced with the file name you are using to boot the guest.

4. Boot the guest using PXE (refer to *Section 15.2, “Booting a guest using PXE”*.

15.2. Booting a guest using PXE

This section demonstrates how to boot a guest with PXE. Note that this is an example only, and your specific settings will be different to those listed here.

15.2.1. Using bridged networking

1. Ensure bridging is enabled. Have a PXE boot server available in your network.

2. Boot a guest with PXE booting enabled. You can use the **`virt-install`** command to create a new guest with PXE booting enabled, as shown in the following example command:

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

Alternatively, ensure that the guest network is configured to use your bridged network, and that the XML guest configuration file has a **<boot dev='network'/>** element inside the **<os>** element, as shown in the following. Note that settings such as MAC addresses and interfaces in this output are only examples:

```
<os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='network'/>
    <boot dev='hd'/>
</os>
    <interface type='bridge'>
      <mac address='52:54:00:5a:ad:cb'/>
      <source bridge='breth0'/>
      <target dev='vnet0'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
```

15.2.2. Using a private libvirt network

1. Configure PXE booting on libvirt as shown in *Section 15.1.1, “Setting up a PXE boot server on a private libvirt network”*.

2. Boot a guest using libvirt with PXE booting enabled. You can use the **`virt-install`** command to create/install a new guest using PXE:

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

Alternatively, ensure that the guest network is configured to use your bridged network, and that the XML guest configuration file has a **<boot dev='network'/>** element inside the **<os>** element, as shown in the following. Note that settings such as MAC addresses and interfaces in this output are only examples:

```
<os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='network'/>
    <boot dev='hd'/>
</os>
  and make sure it is connected to the private network (also just an example):
    <interface type='network'>
      <mac address='52:54:00:66:79:14'/>
      <source network='default'/>
      <target dev='vnet0'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
```

Appendix A. Revision History

Revision 0.2-78	**Friday December 02 2011**	**Laura Bailey** *lbailey@redhat.com*

Release for GA of Red Hat Enterprise Linux 6.2

Revision 0.2-66	**Wed Sep 14 2011**	**Scott Radvan** *sradvan@redhat.com*

BZ#734639

Revision 0.2-62	**Wed Sep 14 2011**	**Scott Radvan** *sradvan@redhat.com*

BZ#736497

Revision 0.2-60	**Thu Sep 01 2011**	**Scott Radvan** *sradvan@redhat.com*

Add changes from SME review - BZ#734651.

Revision 0.2-59	**Thu Sep 01 2011**	**Scott Radvan** *sradvan@redhat.com*

Add changes from SME review - BZ#734647 and BZ#734653.

Revision 0.2-03	**Mon May 30 2011**	**Scott Radvan** *sradvan@redhat.com*

Add SR-IOV, Para Virt drivers and Full 6 install on 6.

Revision 0.2-02	**Mon May 30 2011**	**Scott Radvan** *sradvan@redhat.com*

Initial push to internal preview site.

Revision 0.2-01	**Sat May 28 2011**	**Scott Radvan** *sradvan@redhat.com*

Configure layout, import introductory text.

Revision 0-1	**Mon Apr 4 2011**	**Scott Radvan** *sradvan@redhat.com*

Initial creation of book by publican.