Fedora 13

Virtualization Guide

The definitive guide for virtualization on Fedora

Christopher Curran

Fedora 13 Virtualization Guide
The definitive guide for virtualization on Fedora
Edition 0

Author Christopher Curran ccurran@redhat.com

Copyright © 2008,2009,2010 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/*. The original authors of this document, and Red Hat, designate the Fedora Project as the "Attribution Party" for purposes of CC-BY-SA. 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.

For guidelines on the permitted uses of the Fedora trademarks, refer to *https://fedoraproject.org/wiki/Legal:Trademark_guidelines*.

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.

All other trademarks are the property of their respective owners.

The Fedora Virtualization Guide contains information on installation, configuring, administering, and troubleshooting virtualization technologies included with Fedora.

Please note: This document is still under development, is subject to heavy change, and is provided here as a preview. The content and instructions contained within should not be considered complete, and should be used with caution.

Preface **vii**
1. About this book ... vii
2. Document Conventions ... vii
2.1. Typographic Conventions ... vii
2.2. Pull-quote Conventions ... ix
2.3. Notes and Warnings ... ix
3. We Need Feedback! ... x

I. Requirements and limitations **1**

1. System requirements **3**

2. KVM compatibility **5**

3. Virtualization limitations **7**
3.1. General limitations for virtualization ... 7
3.2. KVM limitations ... 7
3.3. Application limitations ... 9

II. Installation **11**

4. Installing the virtualization packages **13**
4.1. Installing KVM with a new Fedora installation ... 13
4.2. Installing KVM packages on an existing Fedora system ... 15

5. Virtualized guest installation overview **17**
5.1. Virtualized guest prerequesites and considerations ... 17
5.2. Creating guests with virt-install ... 17
5.3. Creating guests with virt-manager ... 18
5.4. Installing guests with PXE ... 30

6. Installing Red Hat Enterprise Linux 5 as a fully virtualized guest **37**

7. Installing Windows XP as a fully virtualized guest **47**

8. Installing Windows Server 2003 as a fully virtualized guest **65**

9. Installing Windows Server 2008 as a fully virtualized guest **69**

III. Configuration **81**

10. Virtualized storage devices **83**
10.1. Creating a virtualized floppy disk controller ... 83
10.2. Adding storage devices to guests ... 84
10.3. Configuring persistent storage in Fedora ... 88
10.4. Add a virtualized CD-ROM or DVD device to a guest ... 90

11. Network Configuration **91**
11.1. Network address translation (NAT) with libvirt ... 91
11.2. Bridged networking with libvirt ... 92

12. KVM Para-virtualized Drivers **95**
12.1. Installing the KVM Windows para-virtualized drivers ... 95
12.2. Installing drivers with a virtualized floppy disk ... 106
12.3. Using KVM para-virtualized drivers for existing devices ... 107
12.4. Using KVM para-virtualized drivers for new devices ... 107

13. PCI passthrough **111**
13.1. Adding a PCI device with virsh ... 112
13.2. Adding a PCI device with virt-manager ... 114
13.3. PCI passthrough with virt-install ... 118

14. SR-IOV **121**
14.1. Introduction ... 121
14.2. Using SR-IOV ... 122
14.3. Troubleshooting SR-IOV ... 125

15. USB device passthrough **127**

16. N_Port ID Virtualization (NPIV) **129**

17. KVM guest timing management **131**

IV. Administration **135**

18. Server best practices **137**

19. Security for virtualization **139**
19.1. Storage security issues ... 139
19.2. SELinux and virtualization ... 139
19.3. SELinux ... 141
19.4. Virtualization firewall information ... 141

20. KVM live migration **143**
20.1. Live migration requirements ... 143
20.2. Share storage example: NFS for a simple migration ... 144
20.3. Live KVM migration with virsh ... 145
20.4. Migrating with virt-manager ... 146

21. Remote management of virtualized guests **157**
21.1. Remote management with SSH ... 157
21.2. Remote management over TLS and SSL ... 158
21.3. Transport modes ... 159

22. KSM **165**

23. Advanced virtualization administration **167**
23.1. Guest scheduling ... 167
23.2. Advanced memory management ... 167
23.3. Guest block I/O throttling ... 167
23.4. Guest network I/O throttling ... 167

24. Xen to KVM migration **169**
24.1. Xen to KVM ... 169
24.2. Older versions of KVM to KVM ... 169

25. Miscellaneous administration tasks **171**
25.1. Automatically starting guests ... 171
25.2. Using qemu-img ... 171
25.3. Overcommitting with KVM ... 172
25.4. Verifying virtualization extensions ... 174
25.5. Accessing data from a guest disk image ... 175
25.6. Setting KVM processor affinities ... 177
25.7. Generating a new unique MAC address ... 181

25.8. Very Secure `ftpd` 182
25.9. Configuring LUN Persistence 183
25.10. Disable SMART disk monitoring for guests 184
25.11. Configuring a VNC Server 185

V. Virtualization storage topics **187**

26. Using shared storage with virtual disk images **189**
26.1. Using iSCSI for storing virtual disk images 189
26.2. Using NFS for storing virtual disk images 189
26.3. Using GFS2 for storing virtual disk images 189
26.4. Storage Pools 189
26.4.1. Configuring storage devices for pools 189
26.4.2. Mapping virtualized guests to storage pools 189

VI. Virtualization reference guide **191**

27. Virtualization tools **193**

28. Managing guests with virsh **195**

29. Managing guests with the Virtual Machine Manager (virt-manager) **205**
29.1. The Add Connection window 205
29.2. The Virtual Machine Manager main window 206
29.3. The guest Overview tab 207
29.4. Virtual Machine graphical console 208
29.5. Starting virt-manager 209
29.6. Restoring a saved machine 210
29.7. Displaying guest details 212
29.8. Status monitoring 217
29.9. Displaying guest identifiers 219
29.10. Displaying a guest's status 220
29.11. Displaying virtual CPUs 221
29.12. Displaying CPU usage 222
29.13. Displaying memory usage 223
29.14. Managing a virtual network 224
29.15. Creating a virtual network 225

30. libvirt configuration reference **235**

31. Creating custom libvirt scripts **237**
31.1. Using XML configuration files with virsh 237

VII. Troubleshooting **239**

32. Troubleshooting **241**
32.1. Debugging and troubleshooting tools 241
32.2. Log files 242
32.3. Troubleshooting with serial consoles 242
32.4. Virtualization log files 243
32.5. Loop device errors 243
32.6. Enabling Intel VT and AMD-V virtualization hardware extensions in BIOS 243
32.7. KVM networking performance 245

A. Additional resources **247**
A.1. Online resources 247
A.2. Installed documentation 247
Glossary **249**
B. Revision History **253**
C. Colophon **255**

Preface

This book is the Fedora Virtualization Guide. The Guide covers all aspects of using and managing virtualization products included with Fedora.

1. About this book

This book is divided into 7 parts:

- System Requirements
- Installation
- Configuration
- Administration
- Reference
- Tips and Tricks
- Troubleshooting

Key terms and concepts used throughout this book are covered in the glossary, *Glossary*.

This book covers virtualization topics for Fedora. The *Kernel-based Virtual Machine* hypervisor is provided with Fedora. The KVM hypervisor supports *Full virtualization*.

2. 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.

2.1. 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.

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

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`**+**`F1`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F7`** 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 **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.

2.2. 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"));
   }
}
```

2.3. 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' won't cause data loss but may cause irritation and frustration.

Warning

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

3. 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/bugzilla/* against the product **Fedora Documentation.**

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

Part I. Requirements and limitations

System requirements, support restrictions and limitations for virtualization with Fedora

These chapters outline the system requirements, support restrictions, and limitations of virtualization on Fedora.

System requirements

This chapter lists system requirements for successfully running virtualization with Fedora. Virtualization is available for Fedora.

The *Kernel-based Virtual Machine* hypervisor is provided with Fedora.

For information on installing the virtualization packages, read *Chapter 4, 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 the guest operating system per guest. For most operating systems more than 6GB of disk space is recommended.
- One processor core or hyper-thread for each virtualized CPU and one for the hypervisor.
- 2GB of RAM plus additional RAM for virtualized guests.

KVM overcommit

KVM can overcommit physical resources for virtualized guests. Overcommiting 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 *Section 25.3, “Overcommitting with KVM”*.

KVM requirements

The KVM hypervisor requires:

- an Intel processor with the Intel VT and the Intel 64 extensions, or
- an AMD processor with the AMD-V and the AMD64 extensions.

Refer to *Section 25.4, “Verifying virtualization extensions”* to determine if your processor has the virtualization extensions.

Storage support

The working guest storage methods are:

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

- iSCSI, and
- Fibre Channel-based LUNs

File-based guest storage

File-based guest images should be stored in the **`/var/lib/libvirt/images/`** folder. If you use a different directory you must add the directory to the SELinux policy. Refer to *Section 19.2, "SELinux and virtualization"* for details.

KVM compatibility

The KVM hypervisor requires a processor with the Intel-VT or AMD-V virtualization extensions.

Note that this list is not complete. Help us expand it by sending in a bug with anything you get working.

To verify whether your processor supports the virtualization extensions and for information on enabling the virtualization extensions if they are disabled, refer to *Section 25.4, "Verifying virtualization extensions"*.

The Fedora *kvm* package is limited to 256 processor cores.

Should work guests

Operating system	Working level
BeOS	Worked
Red Hat Enterprise Linux 3 x86	Optimized with para-virtualized drivers
Red Hat Enterprise Linux 4 x86	Optimized with para-virtualized drivers
Red Hat Enterprise Linux 4 AMD 64 and Intel 64	Optimized with para-virtualized drivers
Red Hat Enterprise Linux 5 x86	Optimized with para-virtualized drivers
Red Hat Enterprise Linux 5 AMD 64 and Intel 64	Optimized with para-virtualized drivers
Red Hat Enterprise Linux 6 x86	Optimized with para-virtualized drivers
Red Hat Enterprise Linux 6 AMD 64 and Intel 64	Optimized with para-virtualized drivers
Fedora 12 x86	Optimized with para-virtualized drivers
Fedora 12 AMD 64 and Intel 64	Optimized with para-virtualized drivers
Windows Server 2003 R2 32-Bit	Optimized with para-virtualized drivers
Windows Server 2003 R2 64-Bit	Optimized with para-virtualized drivers
Windows Server 2003 Service Pack 2 32-Bit	Optimized with para-virtualized drivers
Windows Server 2003 Service Pack 2 64-Bit	Optimized with para-virtualized drivers
Windows XP 32-Bit	Optimized with para-virtualized drivers
Windows Vista 32-Bit	Should work
Windows Vista 64-Bit	Should work
Windows Server 2008 32-Bit	Optimized with para-virtualized drivers
Windows Server 2008 64-Bit	Optimized with para-virtualized drivers
Windows 7 32-Bit	Optimized with para-virtualized drivers
Windows 7 64-Bit	Optimized with para-virtualized drivers
Open Solaris 10	Worked
Open Solaris 11	Worked

Virtualization limitations

This chapter covers additional limitations of the virtualization packages in Fedora

3.1. General limitations for virtualization

Other limitations

For the list of all other limitations and issues affecting virtualization read the *Fedora 13 Release Notes*. The *Fedora 13 Release Notes* cover the present new features, known issues and limitations as they are updated or discovered.

Test before deployment

You should test for the maximum anticipated load and virtualized network stress before deploying heavy I/O applications. Stress testing is important as there are performance drops caused by virtualization with increased I/O usage.

3.2. KVM limitations

The following limitations apply to the KVM hypervisor:

Constant TSC bit	Systems without a Constant Time Stamp Counter require additional configuration. Refer to *Chapter 17, 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. When *KSM* is used, make sure that the swap size is the size of the overcommit ratio.
CPU overcommit	It is not recommended to have more than 10 virtual CPUs per physical processor core. Any number of overcommitted virtual CPUs above the number of physical processor cores may cause problems with certain virtualized guests. Overcommitting CPUs has some risk and can lead to instability. Refer to *Section 25.3, "Overcommitting with KVM"* for tips and recommendations on overcommitting CPUs.
Virtualized SCSI devices	SCSI emulation is limited to 16 virtualized (emulated) SCSI devices..
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 maximum of 32 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), • the graphics card (using either the Cirrus or qxl driver), and • the memory balloon device. Out of the 32 available PCI devices for a guest 4 are not removable. This means there are only 28 PCI slots available for additional devices per guest. Every para-virtualized network or block device uses one slot. Each guest can use up to 28 additional devices made up of any combination of para-virtualized network, para-virtualized disk devices, or other PCI devices using VT-d.
Migration limitations	Live migration is only possible with CPUs from the same vendor (that is, Intel to Intel or AMD to AMD only). The No eXecution (NX) bit must be set to on or off for both CPUs for live migration.
Storage limitations	The host should not use disk labels to identify file systems in the **`fstab`** file, the **`initrd`** file or used by the kernel command line. If less privileged users, especially virtualized guests, have write access to whole partitions or LVM volumes the host system could be compromised. 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.

PCI passthrough limitations	PCI passthrough (attaching PCI devices to guests) should work on systems with the AMD IOMMU or Intel VT-d technologies.

3.3. Application limitations

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 unstable 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 databasing applications before running them on a virtualized guest. Databases generally use network and storage I/O devices intensively. These applications may not be suitable for a fully virtualized environment. Consider para-virtualization or para-virtualized drivers for increased I/O performance. Refer to *Chapter 12, KVM Para-virtualized Drivers* for more information on the para-virtualized drivers for fully virtualized guests.

Other applications and tools which heavily utilize I/O or require real-time performance should be evaluated carefully. Using full virtualization with the para-virtualized drivers (refer to *Chapter 12, KVM Para-virtualized Drivers*) or para-virtualization results in better performance with I/O intensive applications. 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 fully virtualized hardware.

Part II. Installation

Virtualization installation topics

These chapters cover setting up the host and installing virtualized guests with Fedora. It is recommended to read these chapters carefully to ensure successful installation of virtualized guest operating systems.

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 installation sequence or after installation using the **`yum`** command and the Red Hat Network (RHN).

The KVM hypervisor uses the default Fedora kernel with the *kvm* kernel module.

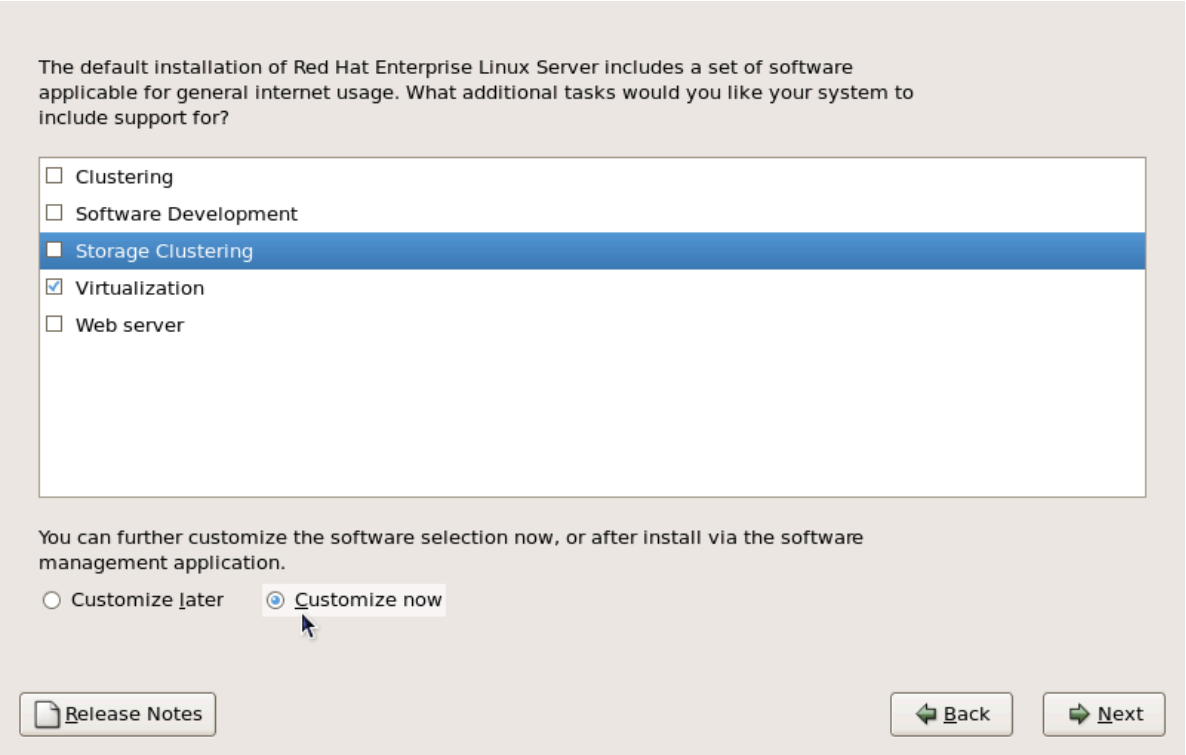
4.1. Installing KVM with a new Fedora installation

This section covers installing virtualization tools and KVM package as part of a fresh Fedora installation.

Need help installing?

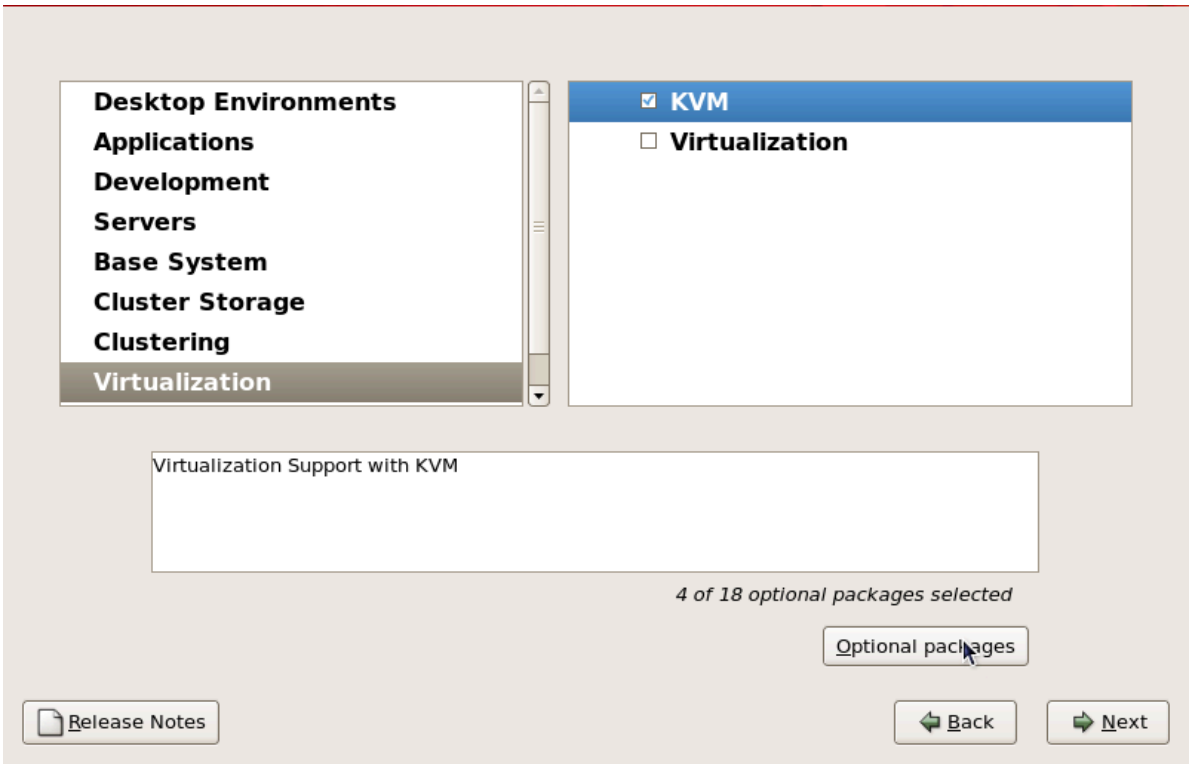
The *Fedora 13 Installation Guide* (available from *http://docs.fedoraproject.org*) covers installing Fedora in detail.

1. Start an interactive Fedora installation from the Fedora 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.

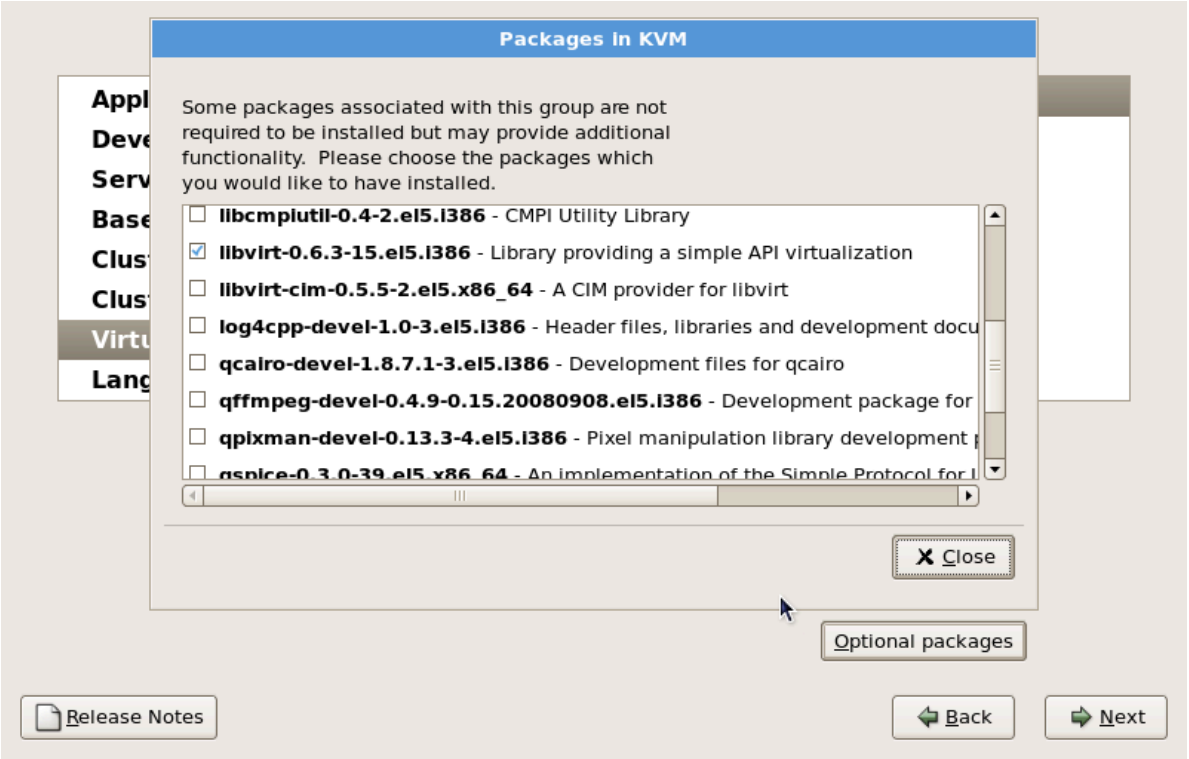


Select the **Virtualization** package group and the **Customize Now** radio button.

4. Select the **KVM** package group. Deselect the **Virtualization** package group. This selects the KVM hypervisor, **`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 **Forward** button to continue the installation.

Installing KVM packages with Kickstart files

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

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

```
%packages
@kvm
```

More information on Kickstart files can be found in the *Fedora 13 Installation Guide*, available from *http://docs.fedoraproject.org*.

4.2. Installing KVM packages on an existing Fedora system

The section describes the steps for installing the KVM hypervisor on a working Fedora or newer system.

Installing the KVM hypervisor with `yum`

To use virtualization on Fedora you require the **`kvm`** package. The **`kvm`** package contains the KVM kernel module providing the KVM hypervisor on the default Fedora kernel.

To install the **`kvm`** package, run:

```
# yum install kvm
```

Now, install additional virtualization management packages.

Recommended virtualization packages:

python-virtinst	Provides the **`virt-install`** command for creating virtual machines.
libvirt	*libvirt* is a cross platform Application Programmers Interface (API) for interacting with hypervisors and host systems. *libvirt* manages systems and controls the hypervisor. The *libvirt* package includes the **`virsh`** command line tool to manage and control virtualized guests and hypervisors from the command line or a special virtualization shell.
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* library as the management API.

Install the other recommended virtualization packages:

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

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 for specific versions of Fedora, other Linux distributions, Solaris and Windows. Refer to the relevant procedure for you guest operating system:

- Red Hat Enterprise Linux 5: *Chapter 6, Installing Red Hat Enterprise Linux 5 as a fully virtualized guest*
- Windows XP: *Chapter 7, Installing Windows XP as a fully virtualized guest*
- Windows Server 2003: *Chapter 8, Installing Windows Server 2003 as a fully virtualized guest*
- Windows Server 2008: *Chapter 9, Installing Windows Server 2008 as a fully virtualized guest*

5.1. Virtualized guest prerequesites and considerations

Various factors should be considered before creating any virtualized guests. Factors include:

- Performance
- Input/output requirements and types of input/output
- Storage
- Networking and network infrastructure

Performance

Virtualization has a performance impact.

I/O requirements and architectures

.

Storage

.

Networking and network infrastructure

.

5.2. 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 one can pass on the command line. To see a complete list of options run:

```
$ 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.

This example creates a Red Hat Enterprise Linux 3 guest, named *rhel3support*, from a CD-ROM, with virtual networking and with a 5 GB file-based block device image. This example uses the KVM hypervisor.

```
# virt-install --accelerate --hvm --connect qemu:///system \
   --network network:default \
   --name rhel3support --ram=756\
   --file=/var/lib/libvirt/images/rhel3support.img \
   --file-size=6 --vnc --cdrom=/dev/sr0
```

Example 5.1. Using virt-install with KVM to create a Red Hat Enterprise Linux 3 guest

```
# virt-install --name fedora11 --ram 512 --file=/var/lib/libvirt/images/fedora11.img \
 --file-size=3 --vnc --cdrom=/var/lib/libvirt/images/fedora11.iso
```

Example 5.2. Using virt-install to create a fedora 11 guest

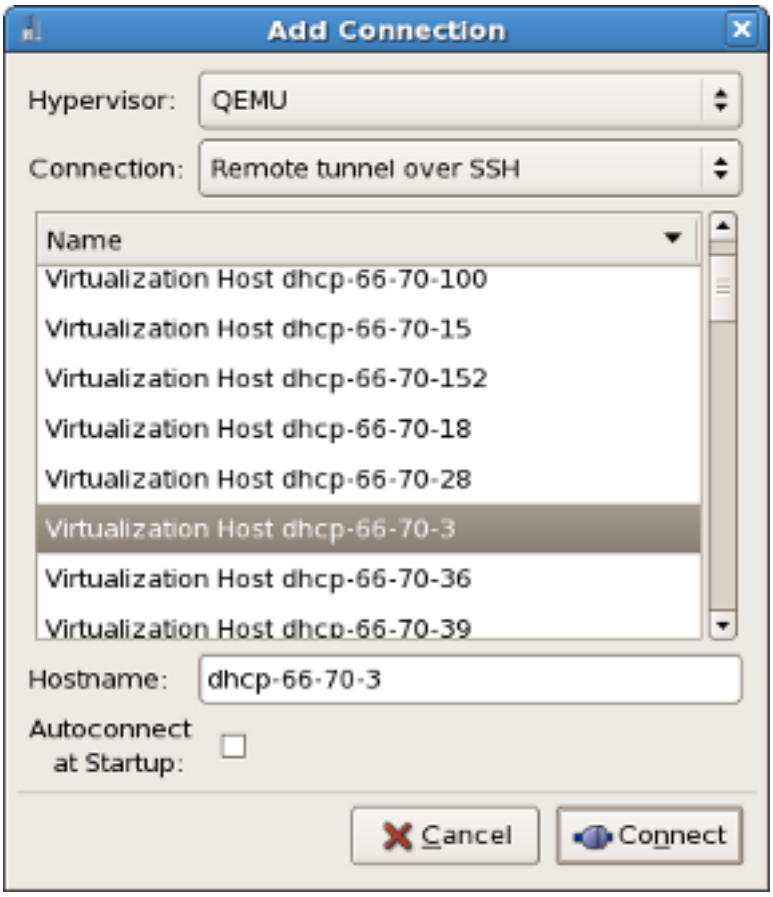
5.3. Creating guests with virt-manager

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

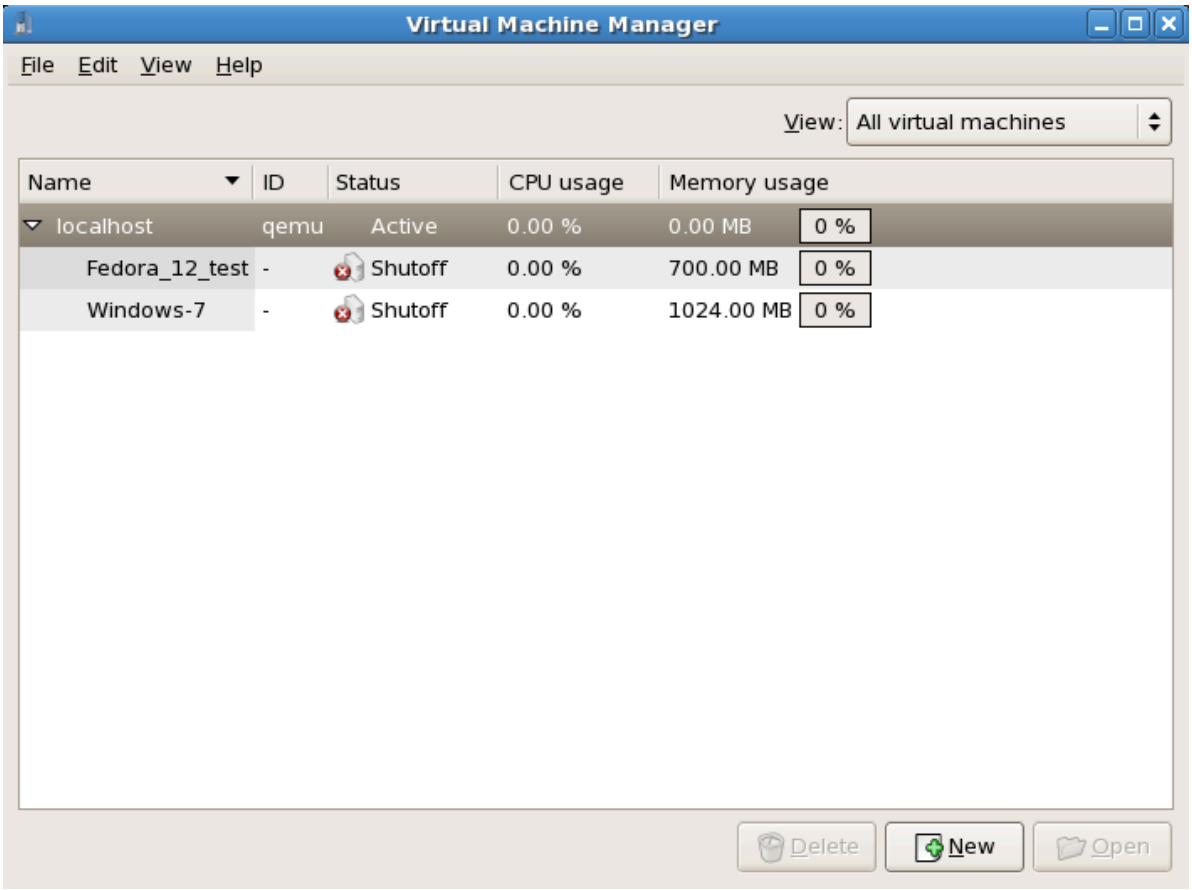
Procedure 5.1. Creating a virtualized guest with **virt-manager**

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**
   Open the **File -> Add Connection**. The dialog box below appears. Select a hypervisor and click the **Connect** button:



3. **Create a new guest**
   The **virt-manager** window allows you to create a new virtual machine. Click the **New** button to create a new guest. This opens the wizard shown in the screenshot.



4. **New guest wizard**
   The **Create a new virtual machine** window provides a summary of the information you must provide in order to create a virtual machine:

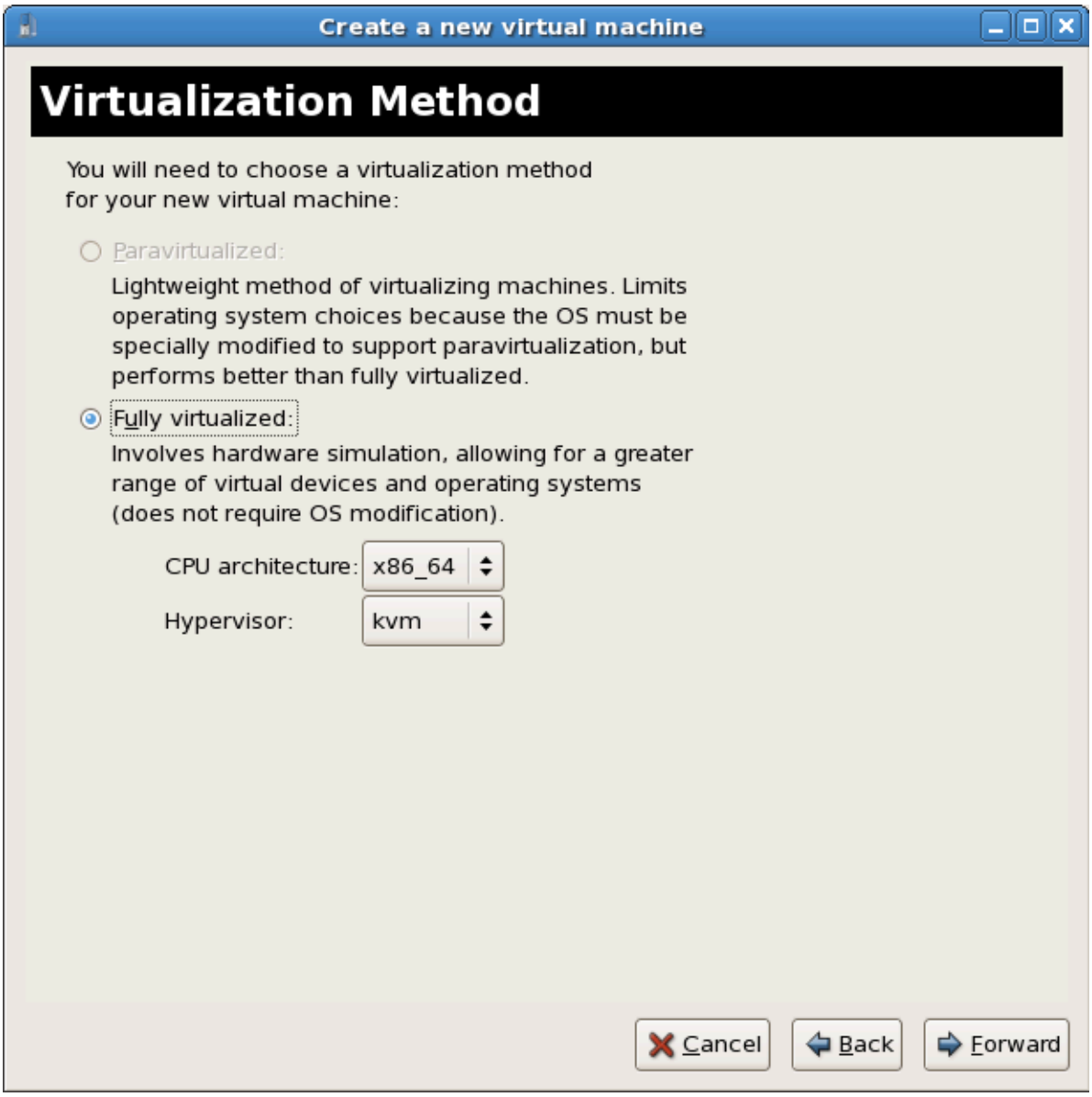
Review the information for your installation and click the **Forward** button.

5. **Name the virtual machine**
   The following characters are allowed in guest names: '_', '**.**' and '**-**' characters.

Press **Forward** to continue.

6. **Choose virtualization method**
   The **Choosing a virtualization method** window appears.

   Full virtualization requires a processor with the AMD 64 and the AMD-V extensions or a processor with the Intel 64 and Intel VT extensions. If the virtualization extensions are not present, KVM will not be avilable.



Choose the virtualization type and click the **Forward** button.

7. **Select the installation method**
   The **Installation Method** window asks for the type of installation you selected.

   Guests can be installed using one of the following methods:

Local media installation	This method uses a CD-ROM or DVD or an image of an installation CD-ROM or DVD (an `.iso` file).
Network installation tree	This method uses a mirrored Fedora installation tree to install guests. The installation tree must be accessible using one of the following network protocols: `HTTP`, `FTP` or `NFS`.

	The network services and files can be hosted using network services on the host or another mirror.
Network boot	This method uses a Preboot eXecution Environment (PXE) server to install the guest. Setting up a PXE server is covered in the *Fedora Deployment Guide*. Using this method requires a guest with a routable IP address or shared network device. Refer to *Chapter 11, Network Configuration* for information on the required networking configuration for PXE installation.

Set the **OS type** and **OS variant**.

Choose the installation method and click **Forward** to procede.

8. **Installation media selection**
   This window is dependent on what was selected in the previous step.

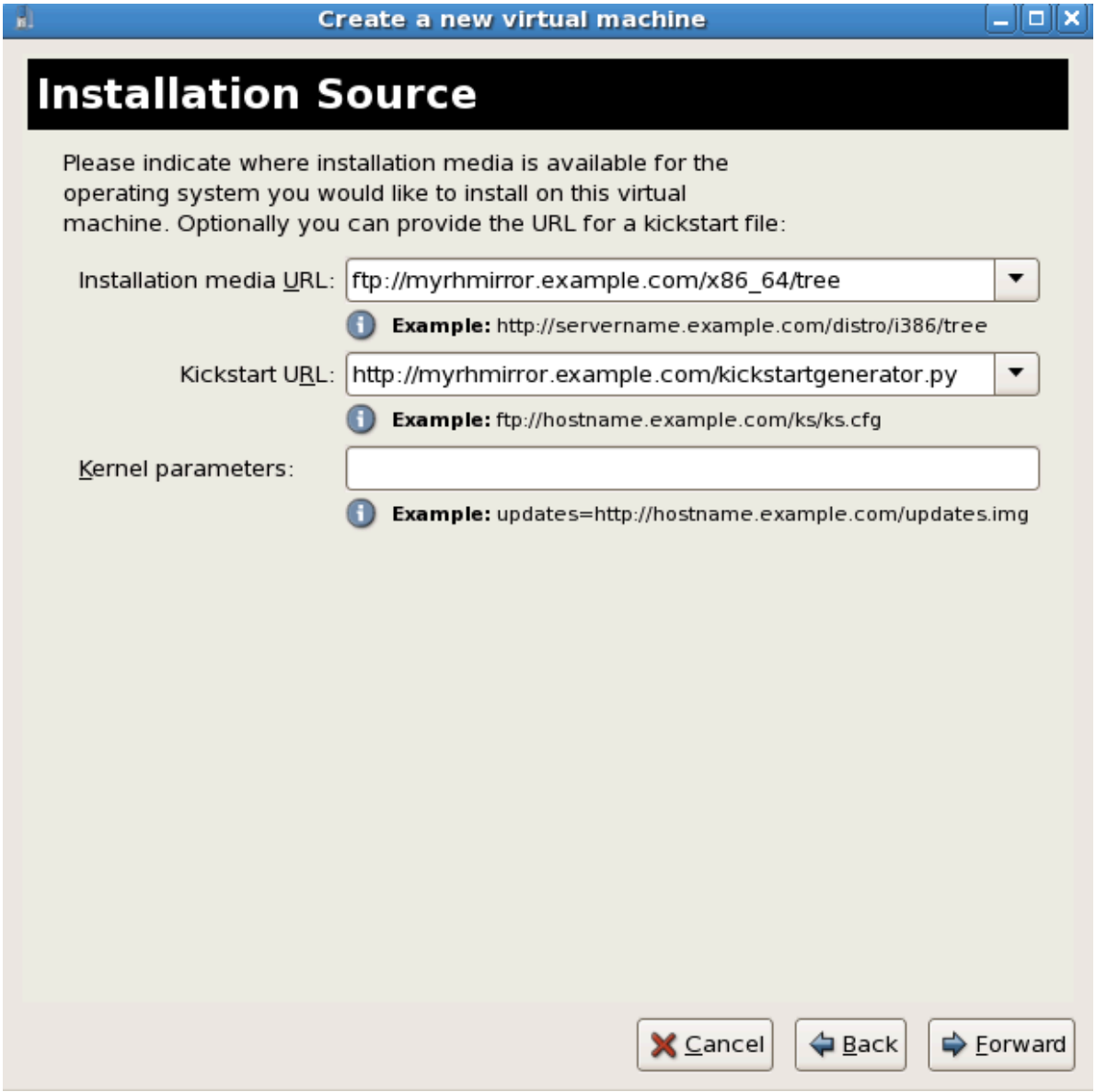
   a. **ISO image or phyiscal media installation**
      If **Local install media** was selected in the previous step this screen is called **Install Media**.

      Select the location of an ISO image or select a DVD or CD-ROM from the dropdown list.

      Click the **Forward** button to procede.

   b. **Network install tree installation**
      If **Network install tree** was selected in the previous step this screen is called **Installation Source**.

Network installation requires the address of a mirror of a Linux installation tree using NFS, FTP or HTTP. Optionally, a kickstart file can be specified to automated the installation. Kernel parameters can also be specified if required.



Click the **Forward** button to procede.

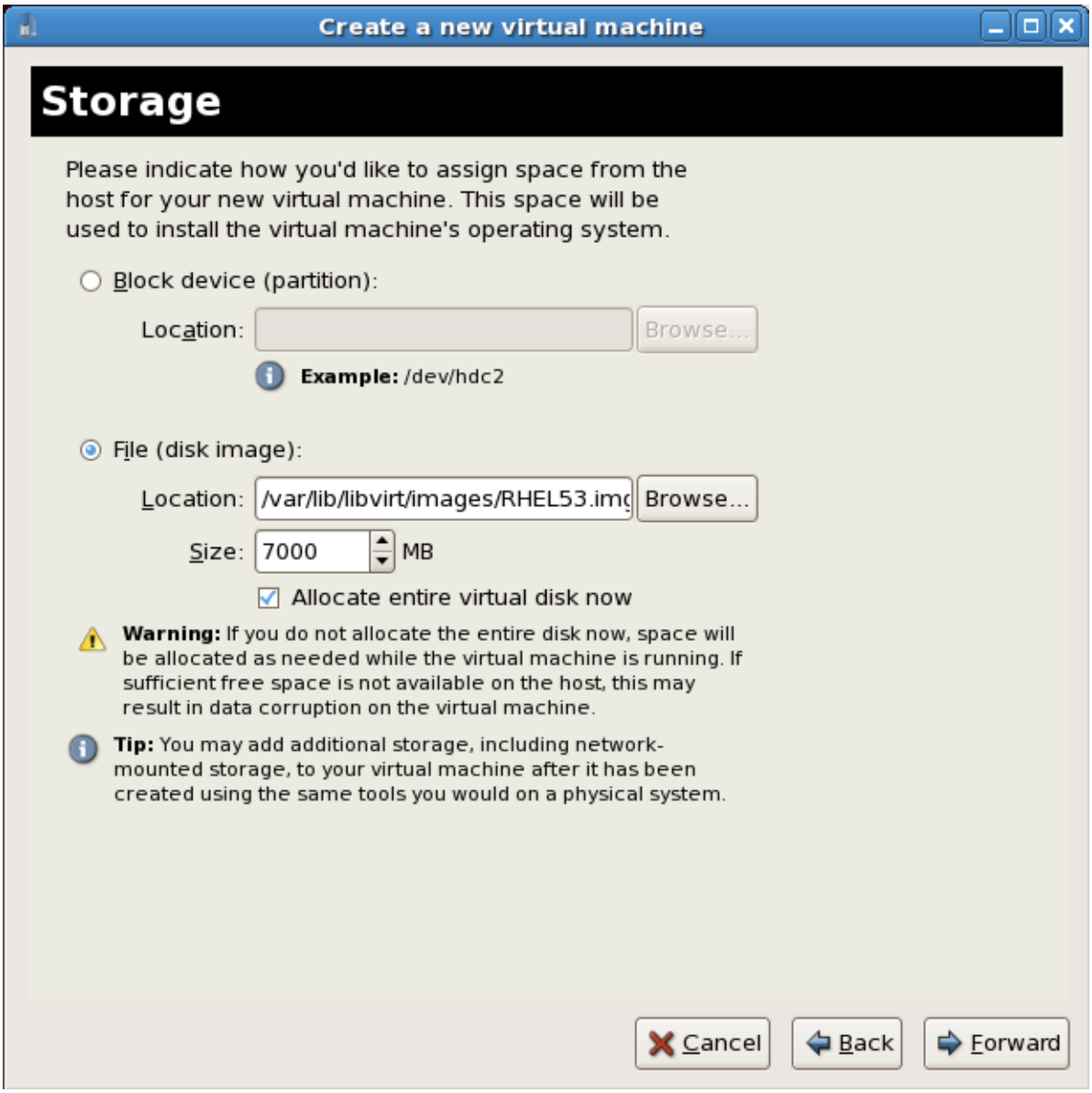
c. **Network boot (PXE)**
PXE installation does not have an additional step.

9. **Storage setup**
The **Storage** window displays. Choose a disk partition, LUN or create a file-based image for the guest storage.

All image files should be stored in the **`/var/lib/libvirt/images/`** directory. Other directory locations for file-based images are prohibited by SELinux. If you run SELinux in enforcing mode, refer to *Section 19.2, "SELinux and virtualization"* for more information on installing guests.

Your guest storage image should be larger than the size of the installation, any additional packages and applications, and the size of the guests swap file. The installation process will choose the size of the guest's swap based on size of the RAM allocated to the guest.

Allocate extra space if the guest needs additional space for applications or other data. For example, web servers require additional space for log files.



Choose the appropriate size for the guest on your selected storage type and click the **Forward** button.

Note

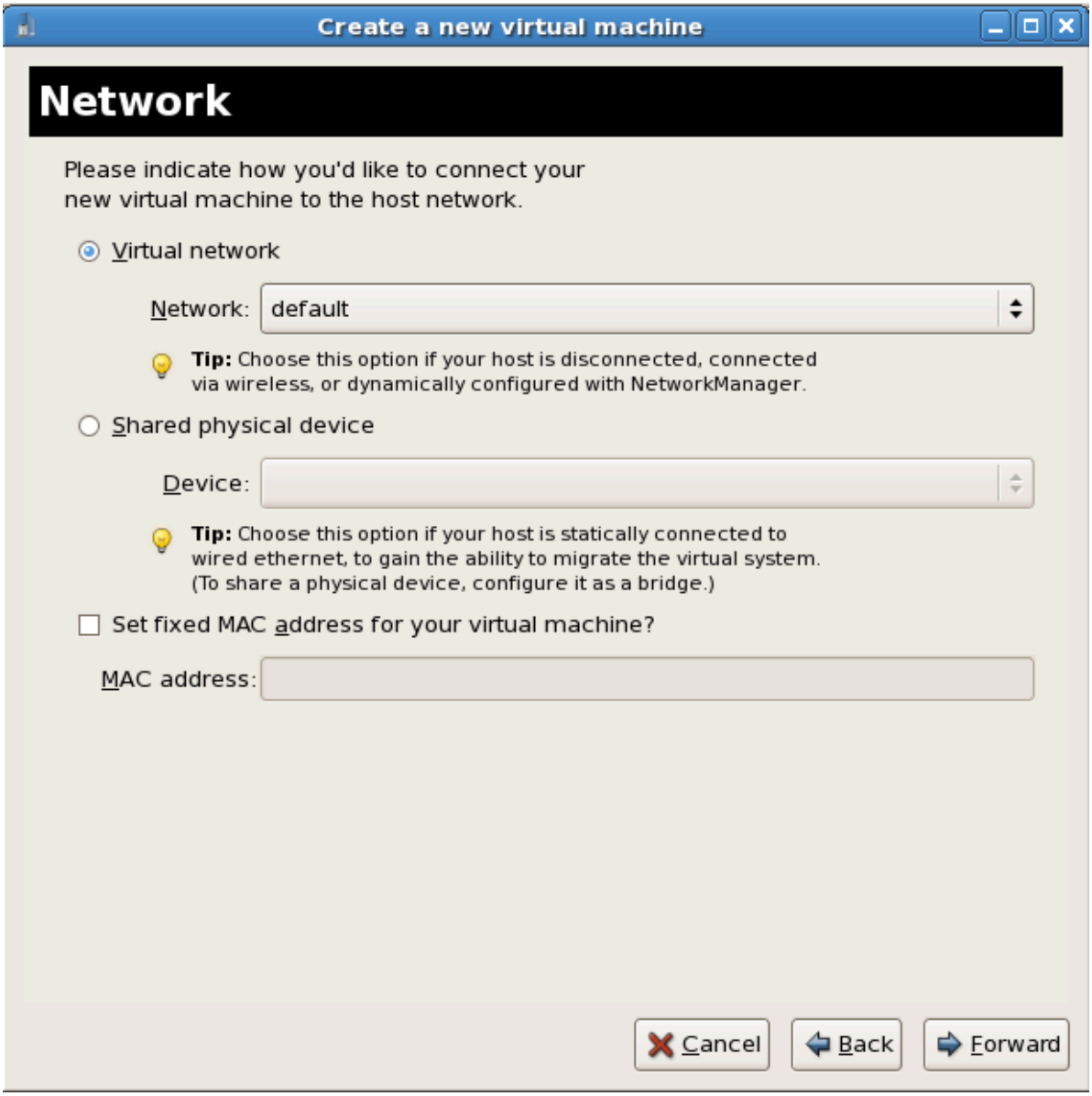
It is recommend that you use the default directory for virtual machine images, `/var/lib/libvirt/images/`. If you are using a different location (such as `/images/` in this example) make sure it is added to your SELinux policy and relabeled before you

continue with the installation (later in the document you will find information on how to modify your SELinux policy).

10. **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. Use the virtual network option for wireless networks.

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

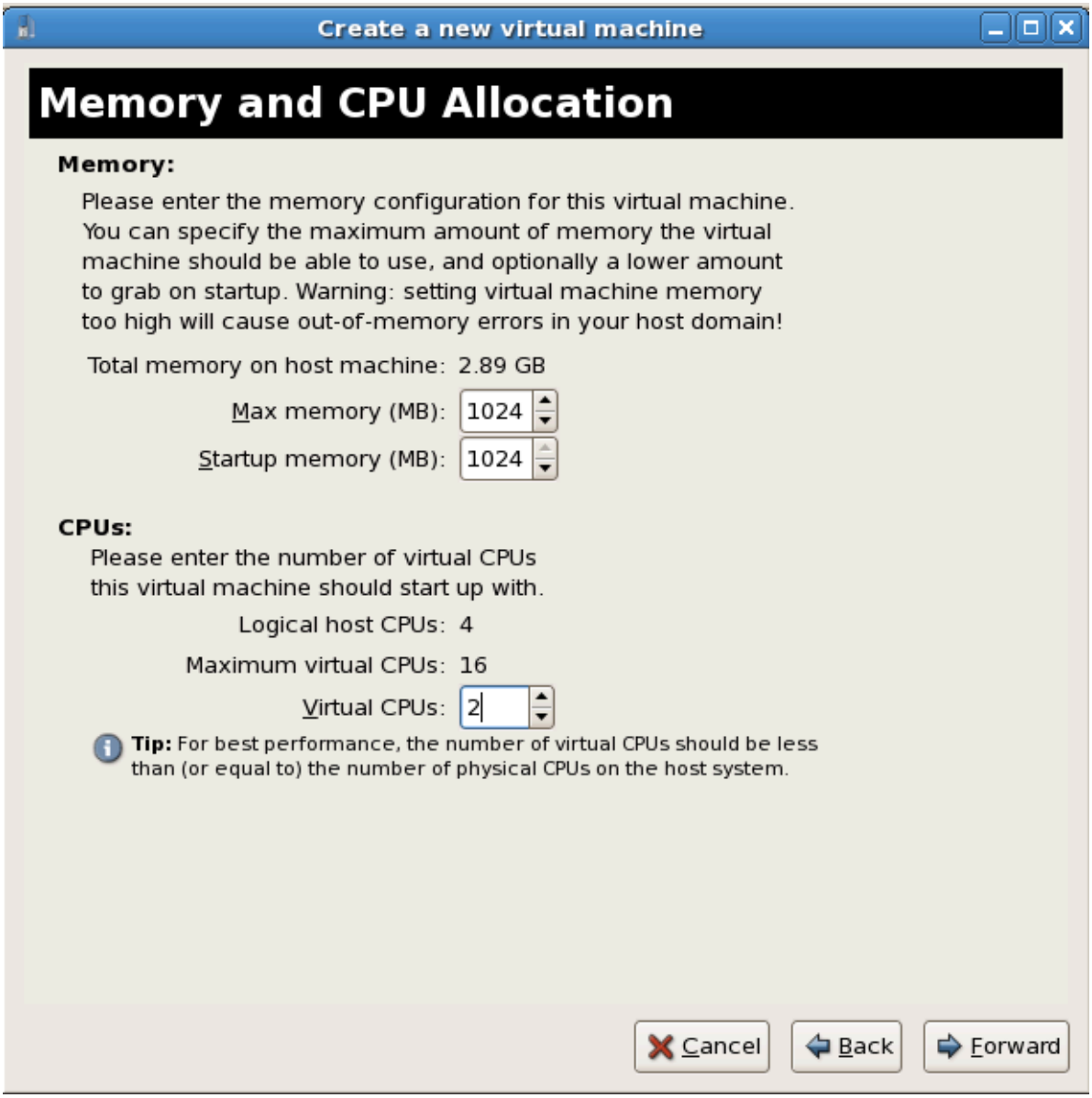


    Press **Forward** to continue.

11. **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.

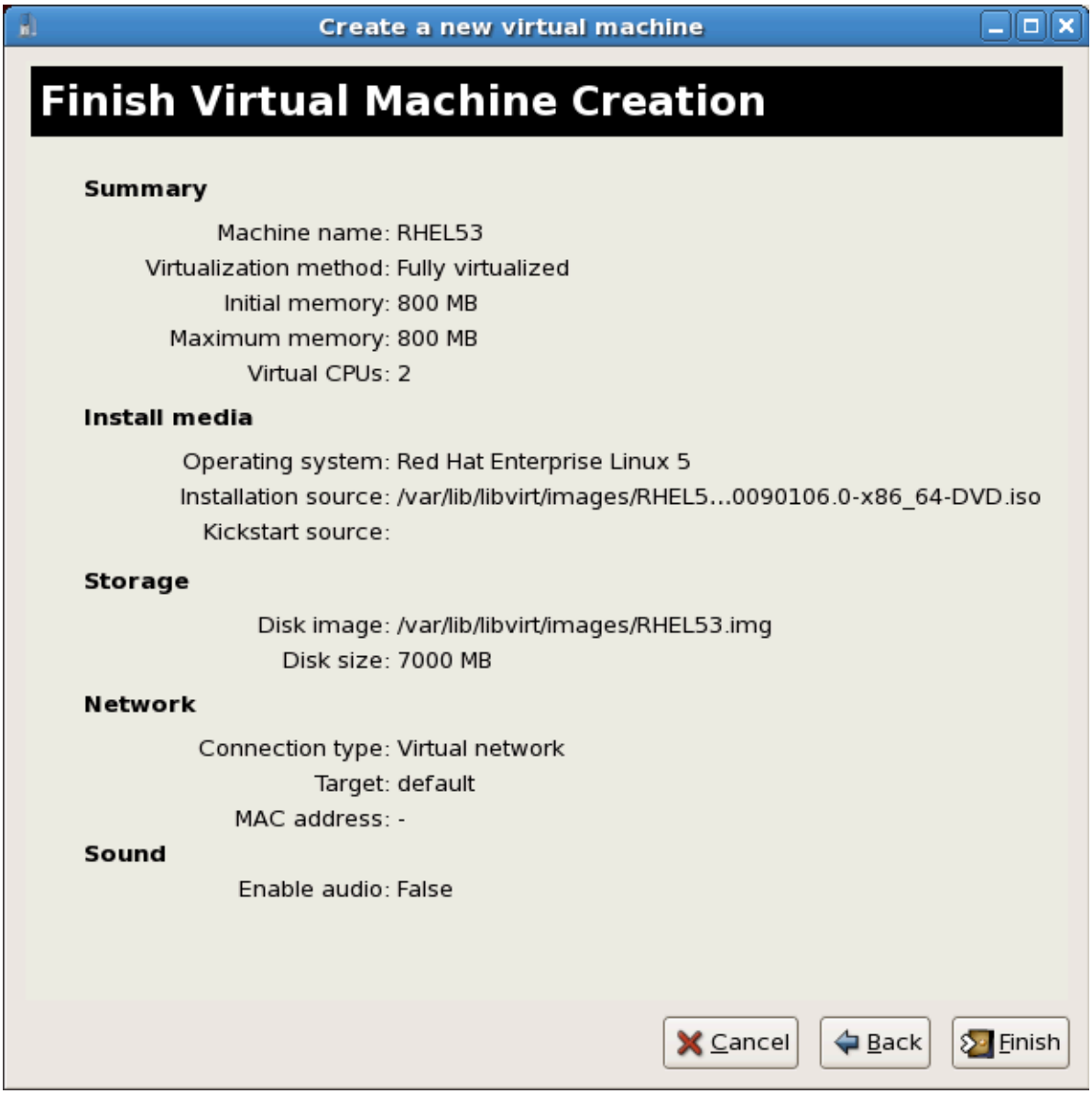
Guests require sufficient physical memory (RAM) to run efficiently and effectively. Choose a memory value which suits your guest operating system and application requirements. Most operating system require at least 512MB of RAM to work responsively. Remember, guests use physical RAM. Running too many guests or leaving insufficient memory for the host system results in significant usage of virtual memory. Virtual memory is significantly slower causing degraded system performance and responsiveness. Ensure to 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 has a significant, negative affect on guest and host performance due to processor context switching overheads.



Press **Forward** to continue.

12. **Verify and start guest installation**
    The **Finish Virtual MAchine Creation** window presents a summary of all configuration information you entered. Review the information presented and use the **Back** button to make changes, if necessary. Once you are satisfied click the **Finish** button and to start the installation process.



    A VNC window opens showing the start of the guest operating system installation process.

This concludes the general process for creating guests with **`virt-manager`**. *Chapter 5, Virtualized guest installation overview* contains step-by-step instructions to installing a variety of common operating systems.

5.4. 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
```

> **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
```

   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.

```
# virt-install --accelerate --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 \
```

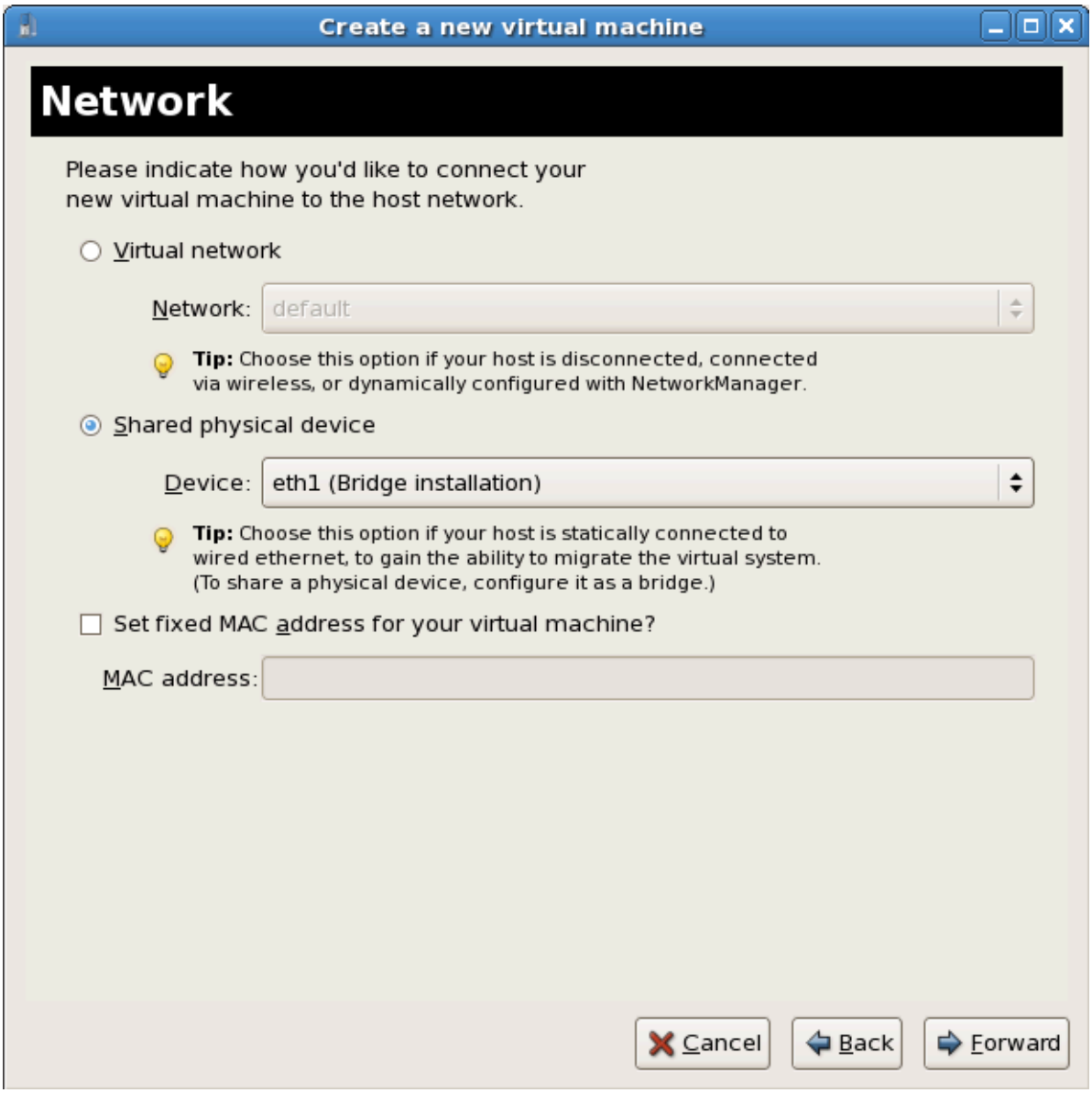
Example 5.3. PXE installation with virt-install

PXE installation with virt-manager

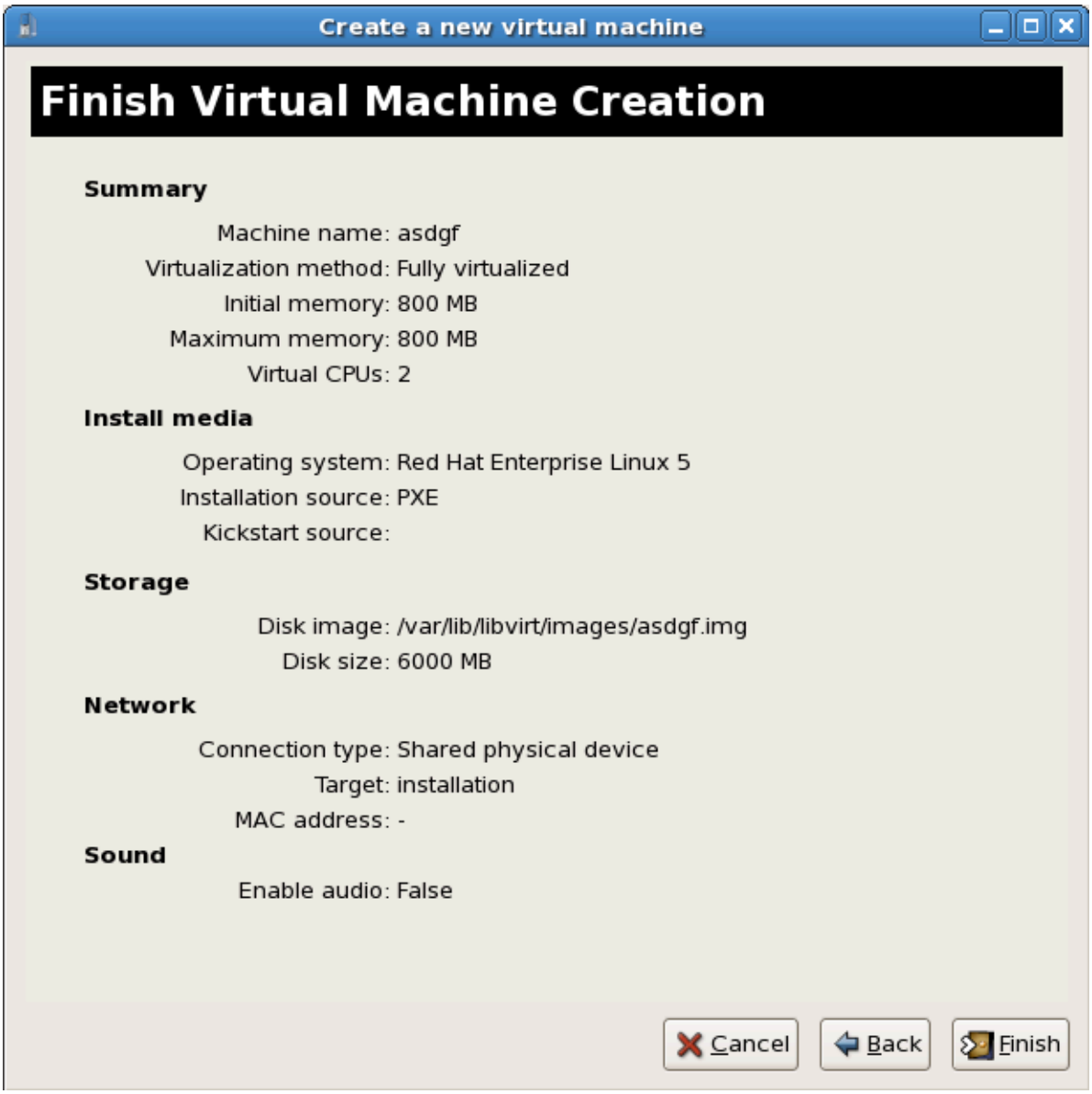
The steps below are the steps that vary from the standard **`virt-manager`** installation procedures.

1. **Select PXE**
   Select PXE as the installation method.

2. **Select the bridge**
   Select **Shared physical device** and select the bridge created in the previous procedure.



3. **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 5 as a fully virtualized guest

This section covers installing a fully virtualized Red Hat Enterprise Linux 5 guest on a Fedora host.

Procedure 6.1. Creating a fully virtualized Red Hat Enterprise Linux 5 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 hypervisor. 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 *Section 29.1, "The Add Connection window"*.

   Once a hypervisor connection is selected the **New** button becomes available. Press the **New** button.

3. **Start the new virtual machine wizard**
   Pressing the **New** button starts the virtual machine creation wizard.

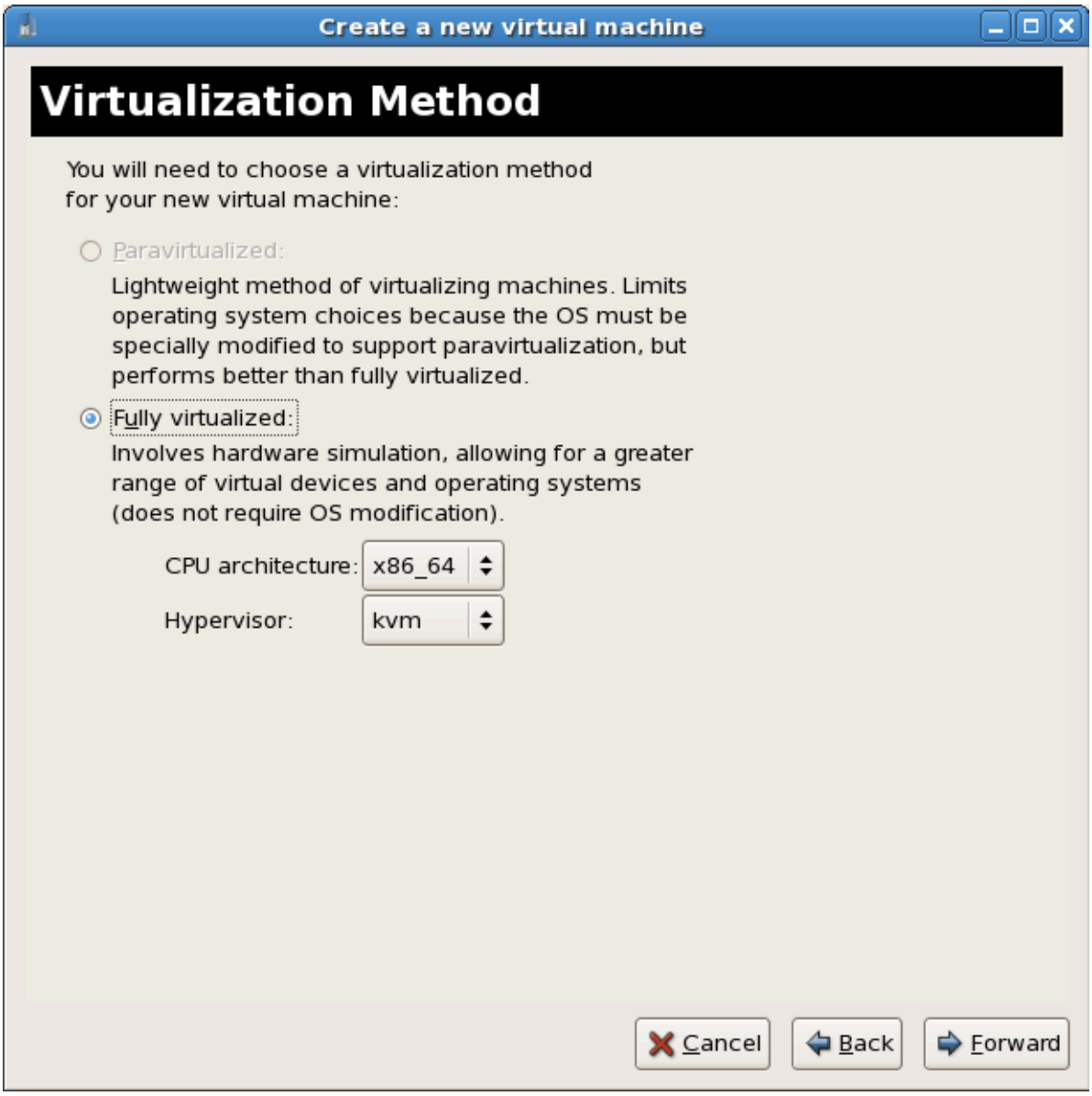
Press **Forward** to continue.

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

Press **Forward** to continue.

5. **Choose a virtualization method**

   Choose the virtualization method for the virtualized guest. Note you can only select between x86_64 (64 bit) and x86 (32 bit).



Press **Forward** to continue.

6. **Select the installation method**
   Red Hat Enterprise Linux can be installed using one of the following methods:

   - **local install media**, either an ISO image or physical optical media.
   - Select **Network install tree** if you have the installation tree for Red Hat Enterprise Linux hosted somewhere on your network via HTTP, FTP or NFS.
   - PXE can be used if you have a PXE server configured for booting Red Hat Enterprise Linux installation media. Configuring a sever to PXE boot a Red Hat Enterprise Linux installation is not covered by this guide. However, most of the installation steps are the same after the media boots.

   Set **OS Type** to **Linux** and **OS Variant** to **Red Hat Enterprise Linux 5** as shown in the screenshot.

Press **Forward** to continue.

7. **Locate installation media**

   Select ISO image location or CD-ROM or DVD device. This example uses an ISO file image of the Red Hat Enterprise Linux installation DVD.

   a. Press the **Browse** button.

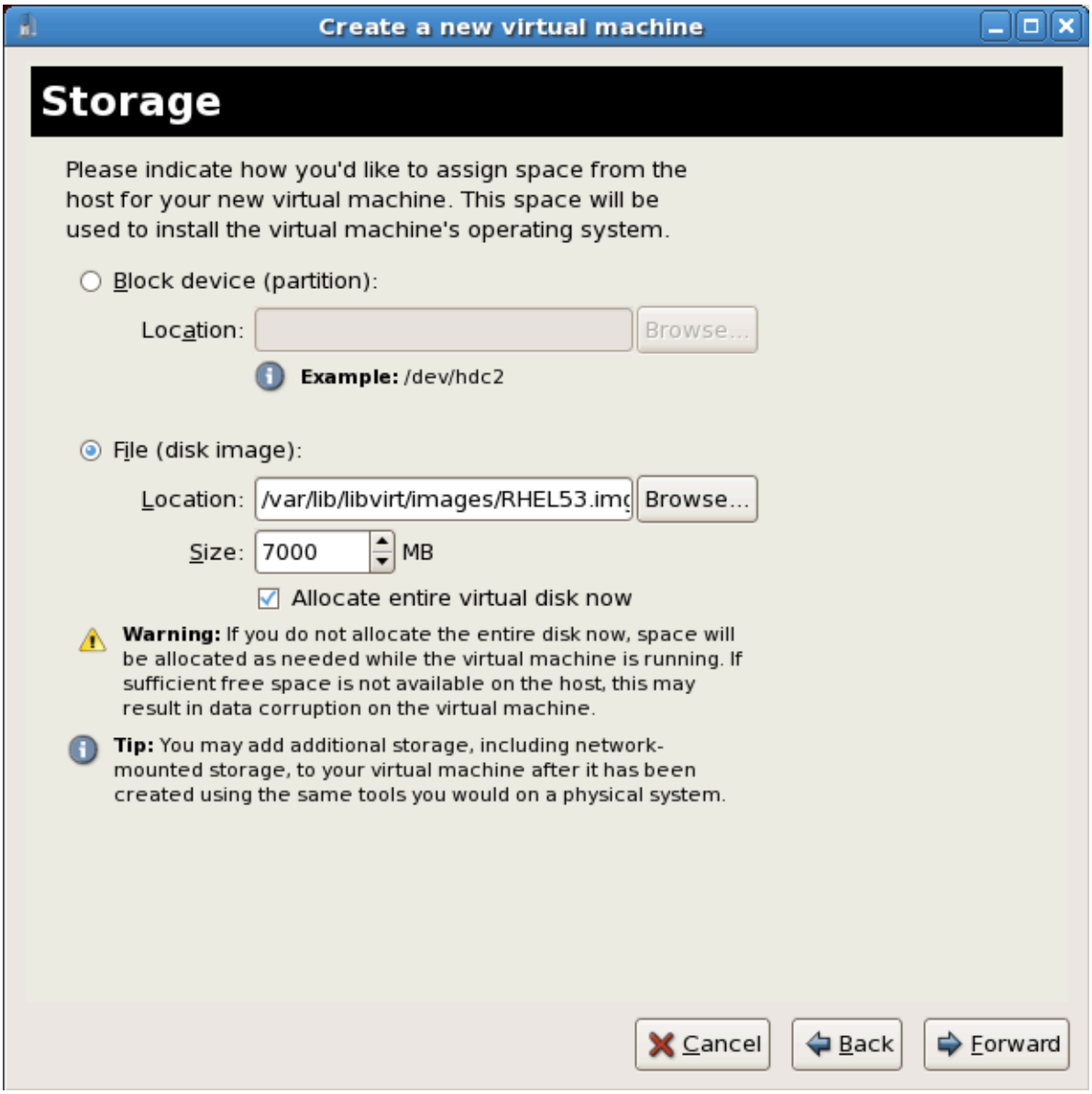
   b. Search to the location of the ISO file and select the ISO image. Press **Open** to confirm your selection.

   c. The file is selected and ready to install.

Press **Forward** to continue.

> **Image files and SELinux**
> For ISO image files and guest storage images the recommended to use the `/var/lib/libvirt/images/` directory. Any other location may require additional configuration for SELinux, refer to *Section 19.2, "SELinux and virtualization"* for details.

8. **Storage setup**
   Assign a physical storage device (**Block device**) or a file-based image (**File**). File-based images must be stored in the `/var/lib/libvirt/images/` directory. Assign sufficient space for your virtualized guest and any applications the guest requires.



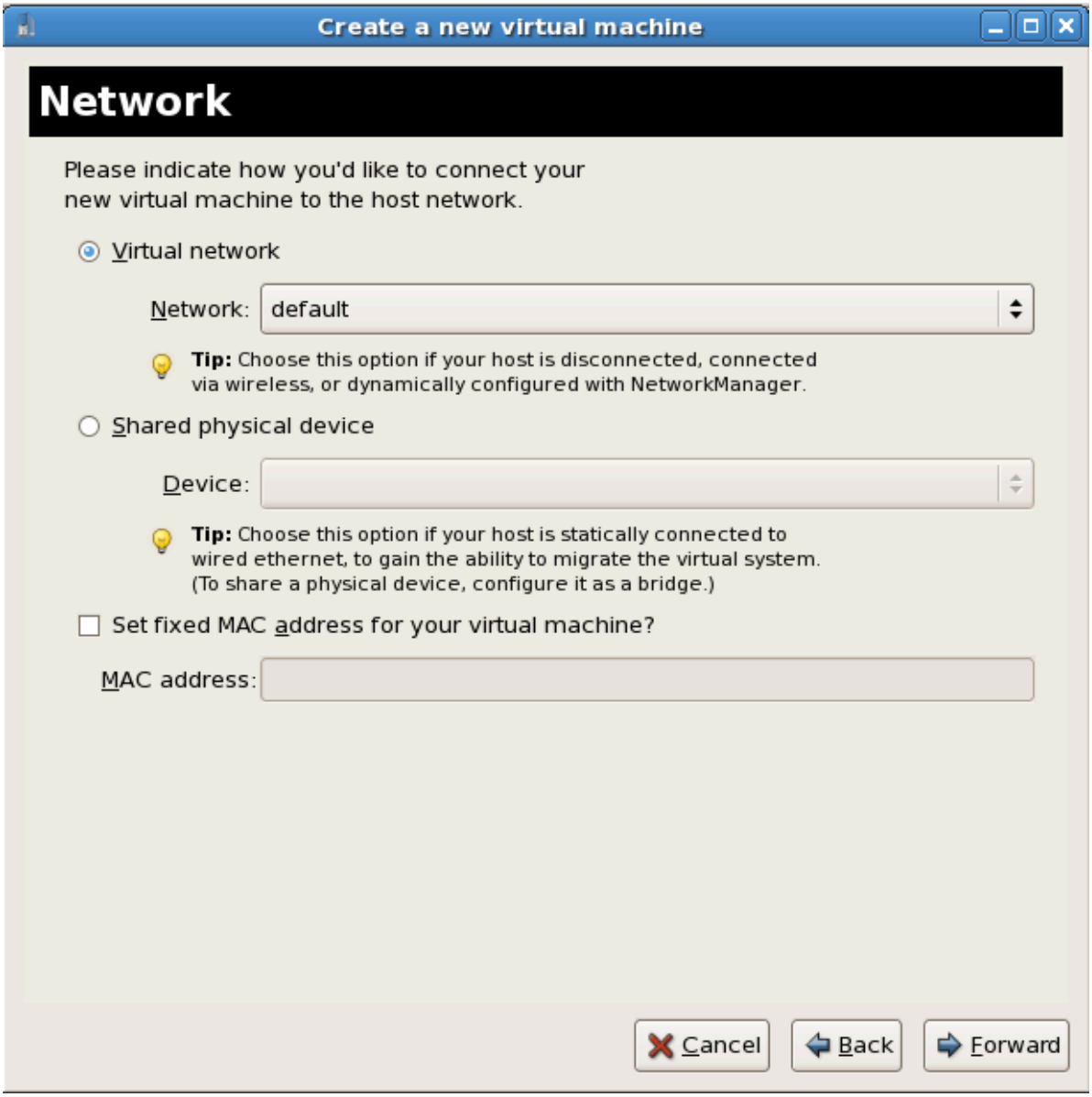
Press **Forward** to continue.

> **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 *Part V, "Virtualization storage topics"*.

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. Use the virtual network option for wireless networks.

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



Press **Forward** to continue.

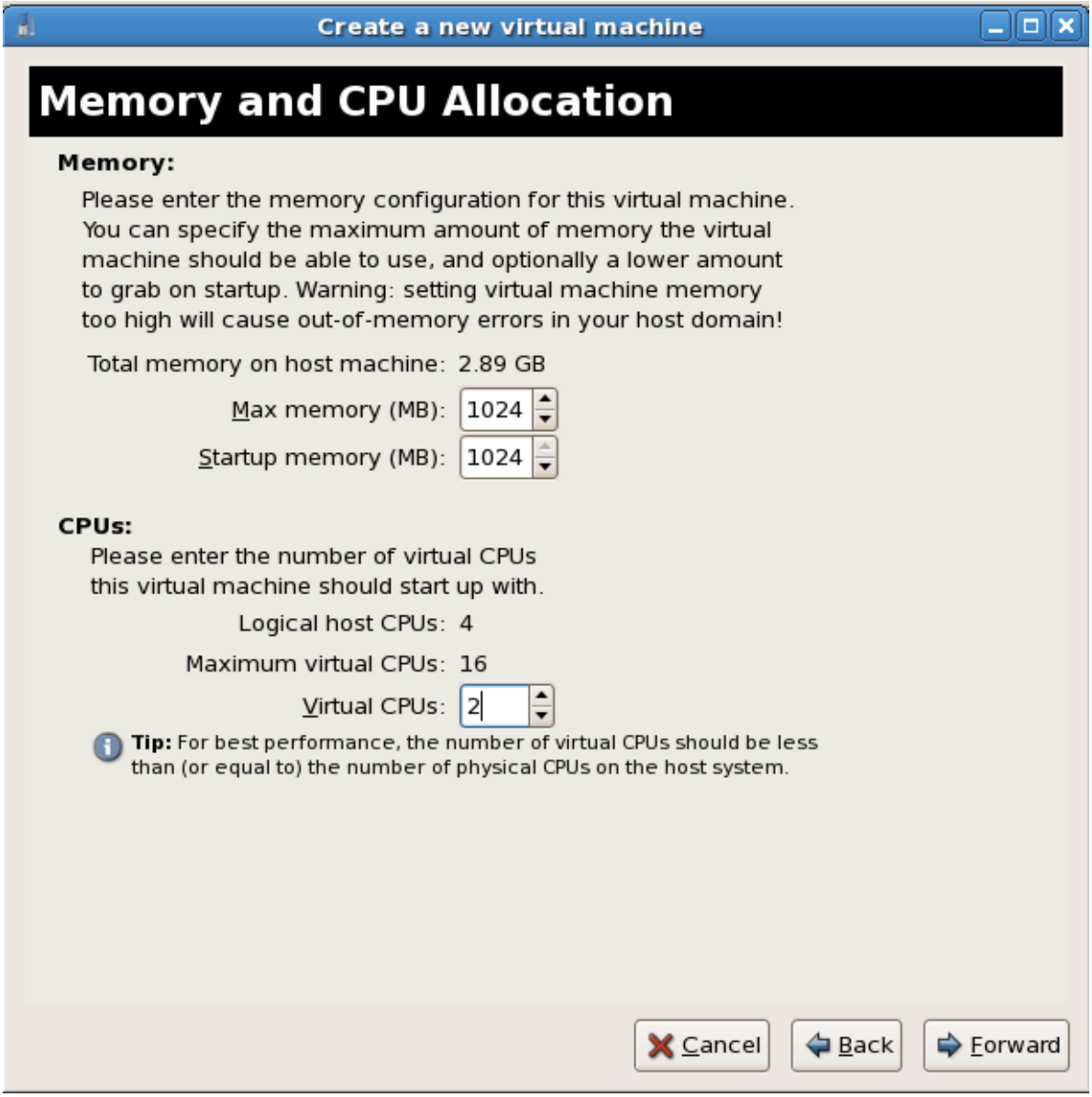
10. **Memory and CPU allocation**

    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, 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 has

a significant, negative effect on guest and host performance due to processor context switching overheads.



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 *Installation Guide*, refer to *Red Hat Documentation*[1] for the Red Hat Enterprise Linux *Installation Guide*.

A fully virtualized Red Hat Enterprise Linux 5 guest is now ready to install.

Installing Windows XP as a fully virtualized guest

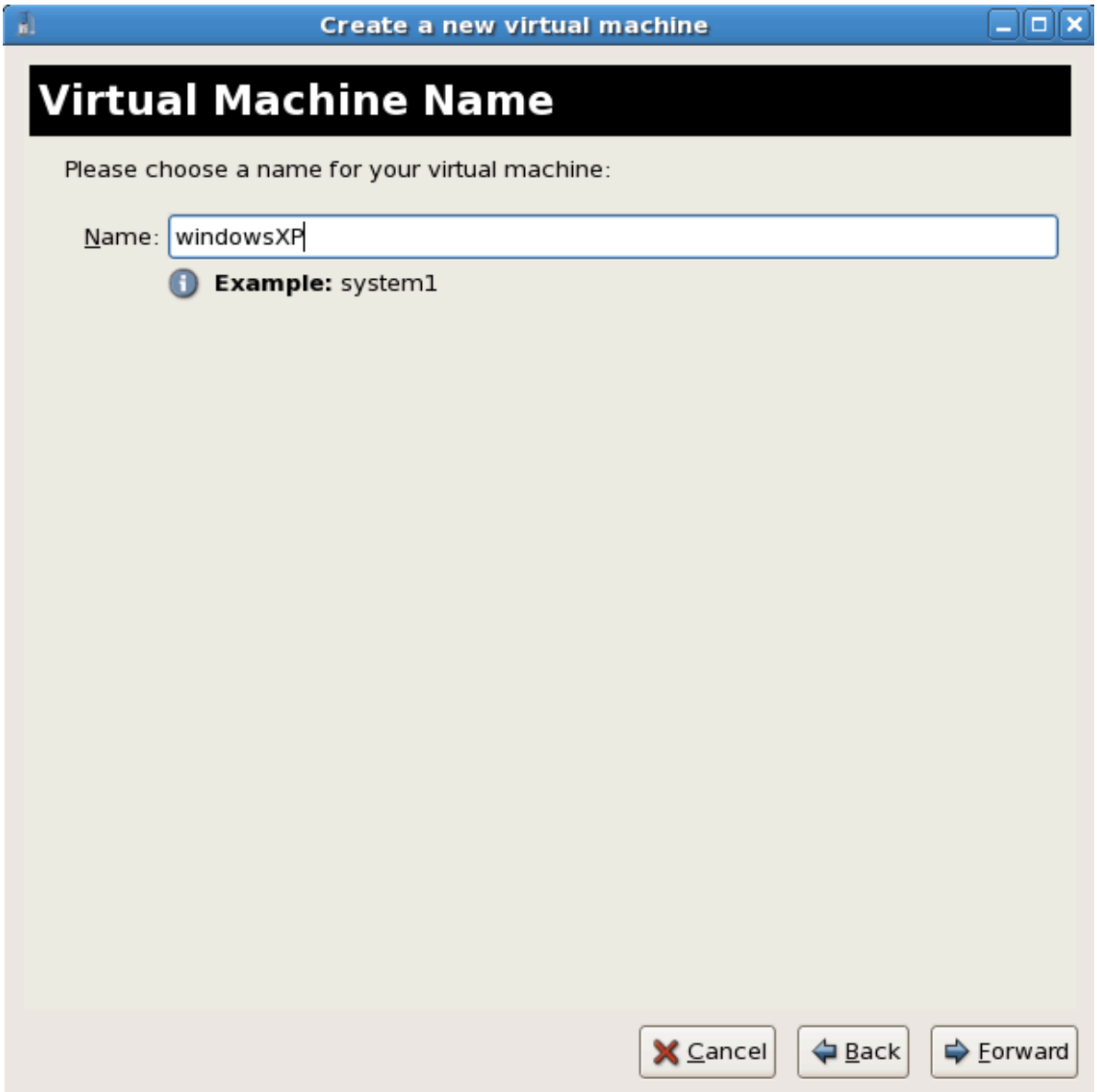
Windows XP can be installed as a fully virtualized guest. This section describes how to install Windows XP as a fully virtualized guest on Fedora.

Before commencing this procedure ensure you must have root access.

1. **Starting virt-manager**

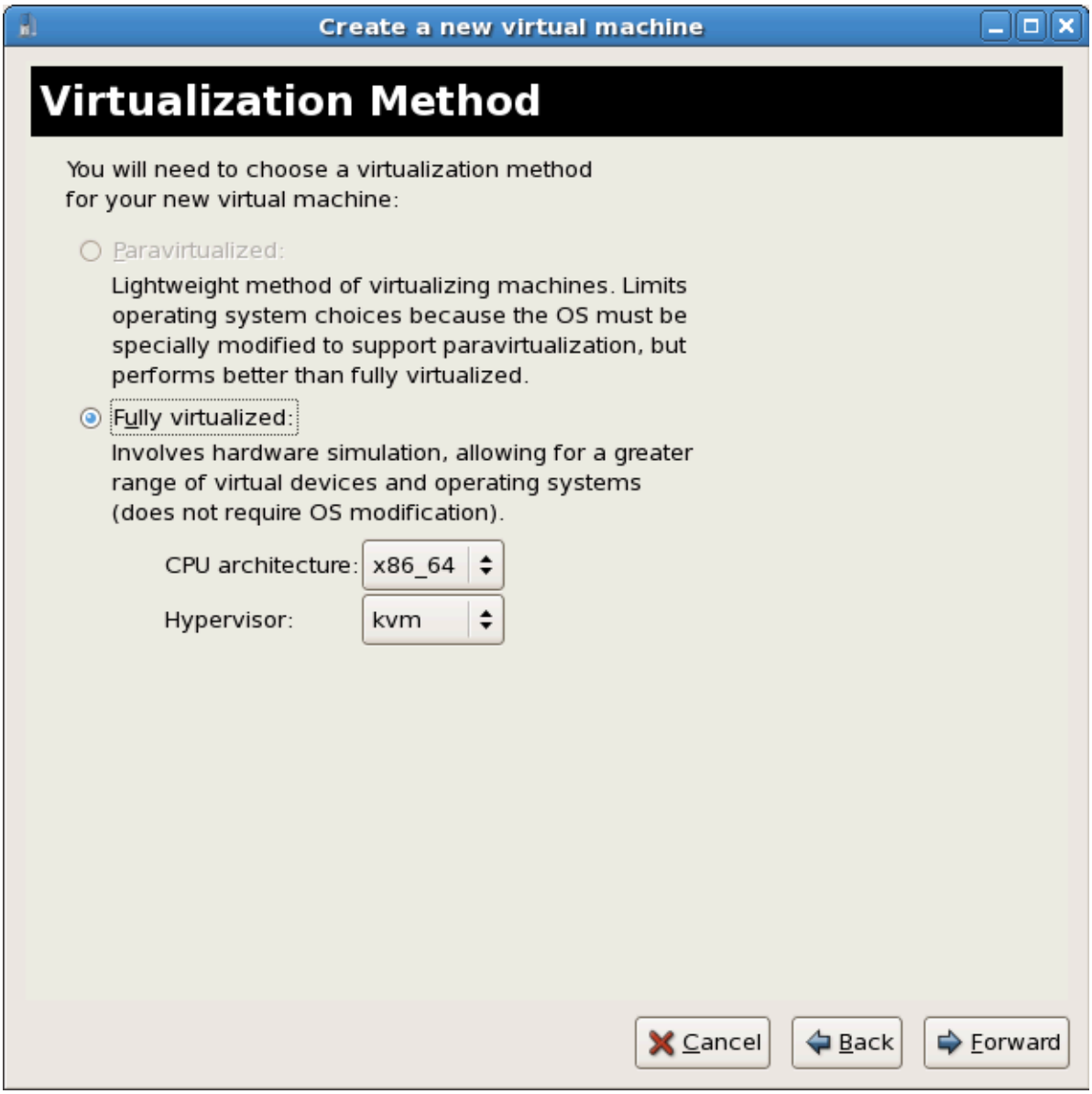
   Open **Applications > System Tools > Virtual Machine Manager**. Open a connection to a host (click **File > Add Connection**). Click the **New** button to create a new virtual machine.

2. **Naming your guest**
   Enter the **System Name** and click the **Forward** button.



3. **Choosing a virtualization method**
   The **Choosing a virtualization method** window appears.

   Full virtualization requires a processor with the AMD 64 and the AMD-V extensions or a processor with the Intel 64 and Intel VT extensions. If the virtualization extensions are not present, KVM will not be avilable.



   Press **Forward** to continue.

4. **Choosing an installation method**
   This screen enables you to specify the installation method and the type of operating system.

   Select **Windows** from the **OS Type** list and **Microsoft Windows XP** from the **OS Variant** list.

   PXE installation is not covered by this chapter.

Image files and SELinux

For ISO image files and guest storage images the recommended to use the `/var/lib/libvirt/images/` directory. Any other location may require additional configuration for SELinux, refer to *Section 19.2, "SELinux and virtualization"* for details.

Press **Forward** to continue.

5. **Choose installation image**

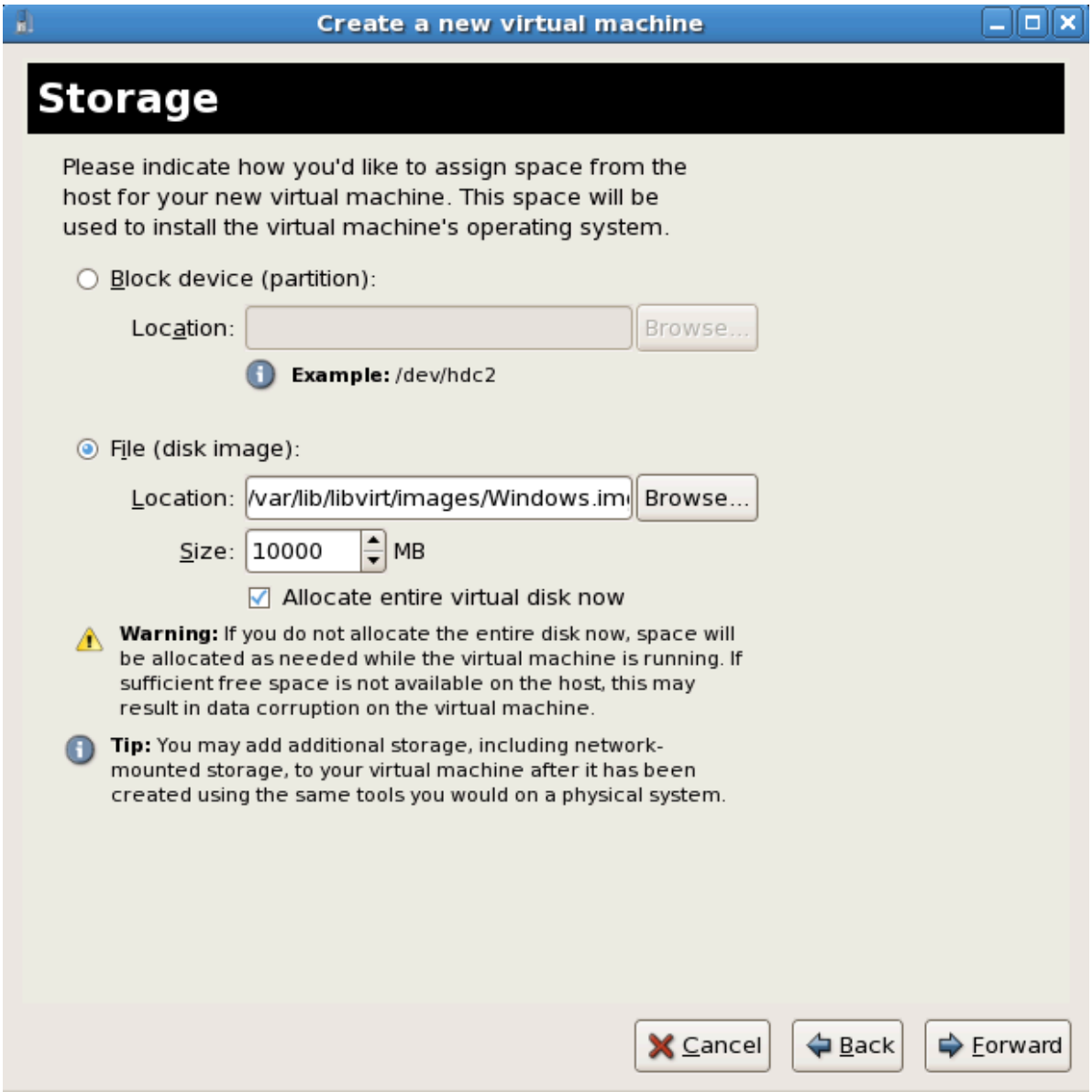
   Choose the installation image or CD-ROM. For CD-ROM or DVD installation select the device with the Windows installation disc in it. If you chose **ISO Image Location** enter the path to a Windows installation .iso image.

Press **Forward** to continue.

6. The **Storage** window displays. Choose a disk partition, LUN or create a file-based image for the guest's storage.

   All image files should be stored in the **`/var/lib/libvirt/images/`** directory. Other directory locations for file-based images are prohibited by SELinux. If you run SELinux in enforcing mode, refer to *Section 19.2, "SELinux and virtualization"* for more information on installing guests.

   Allocate extra space if the guest needs additional space for applications or other data. For example, web servers require additional space for log files.



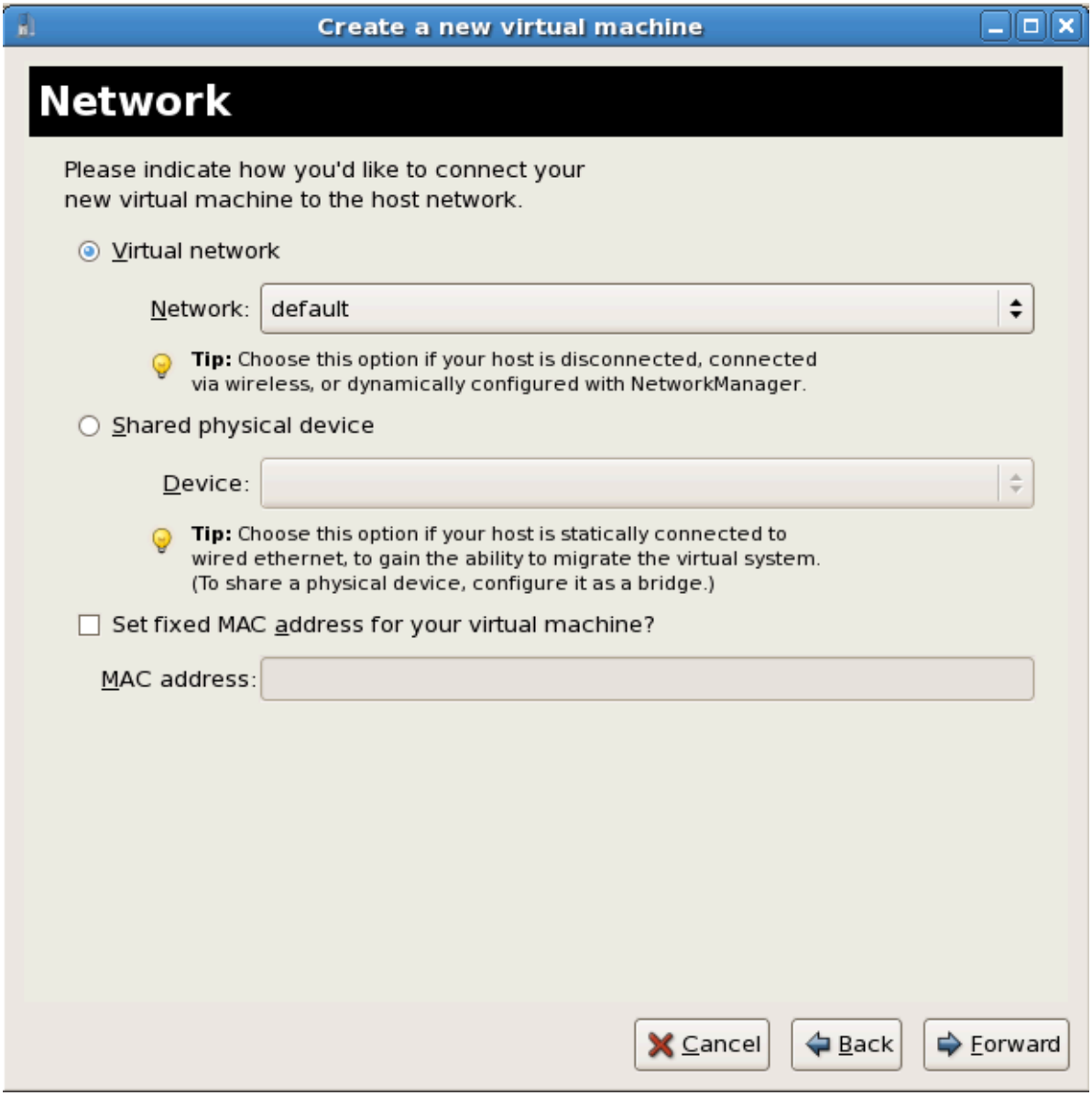
Choose the appropriate size for the guest on your selected storage type and click the **Forward** button.

> **Note**
> It is recommend that you use the default directory for virtual machine images, `/var/lib/libvirt/images/`. If you are using a different location (such as `/images/` in this example) make sure it is added to your SELinux policy and relabeled before you continue with the installation (later in the document you will find information on how to modify your SELinux policy)

7. **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. Use the virtual network option for wireless networks.

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



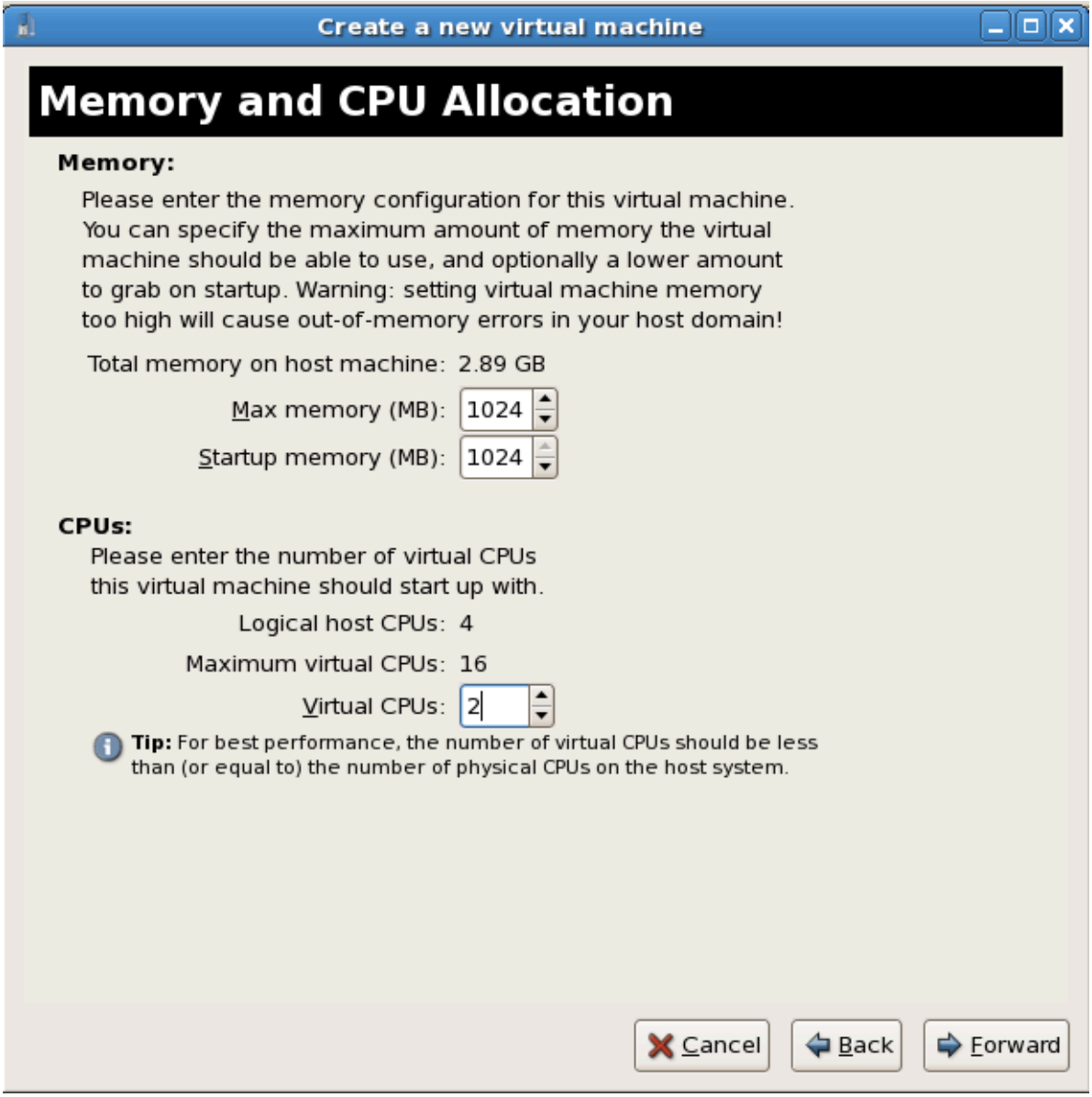
Press **Forward** to continue.

8. 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. Most operating system require at least 512MB of RAM to work responsively. Remember, 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

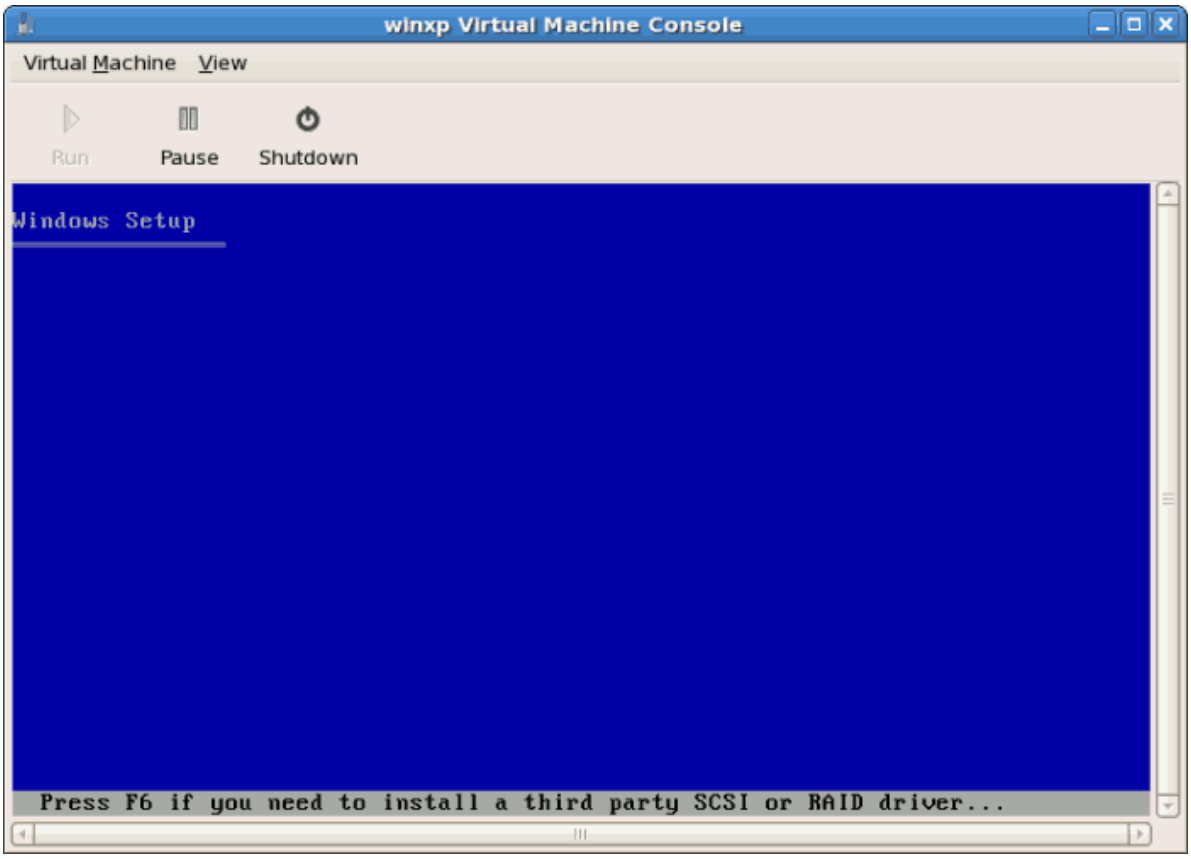
causing degraded system performance and responsiveness. Ensure to allocate sufficient memory for all guests and the host to operate effectively.

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 has a significant, negative effect on guest and host performance due to processor context switching overheads.

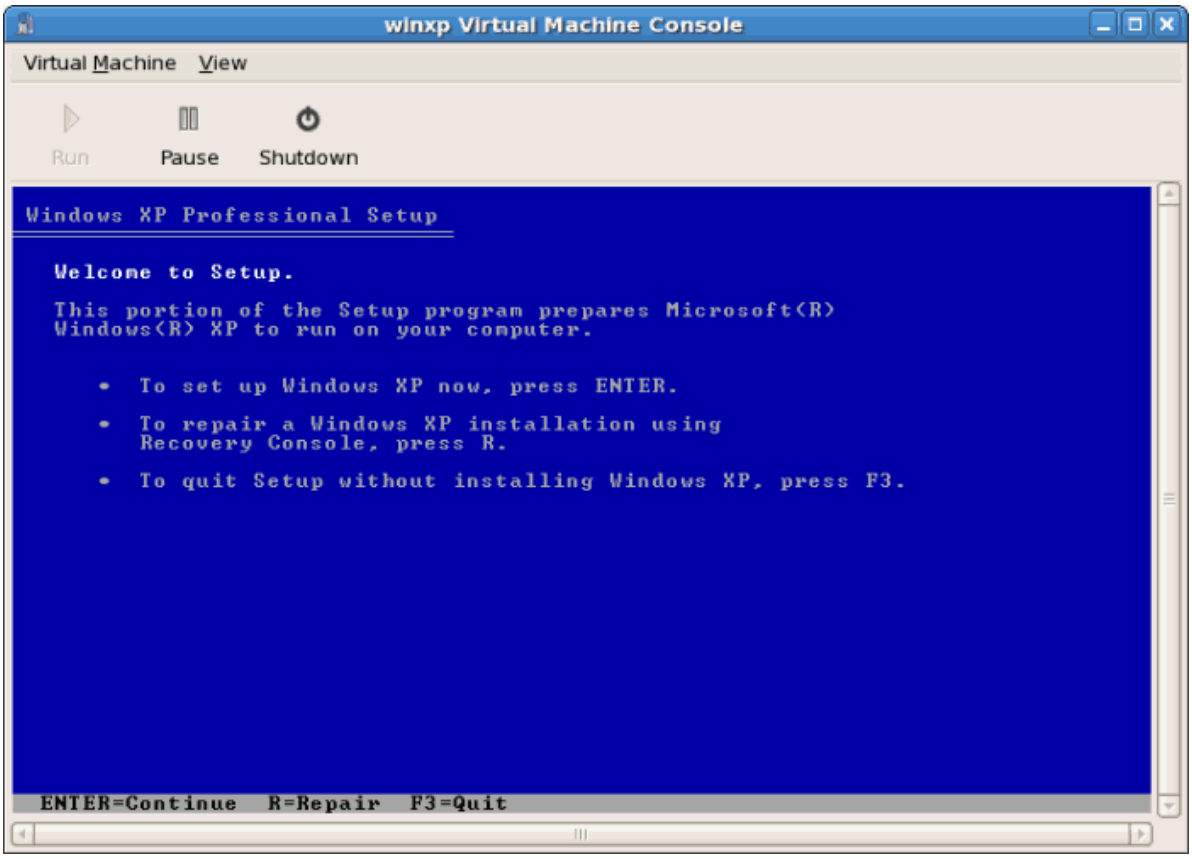


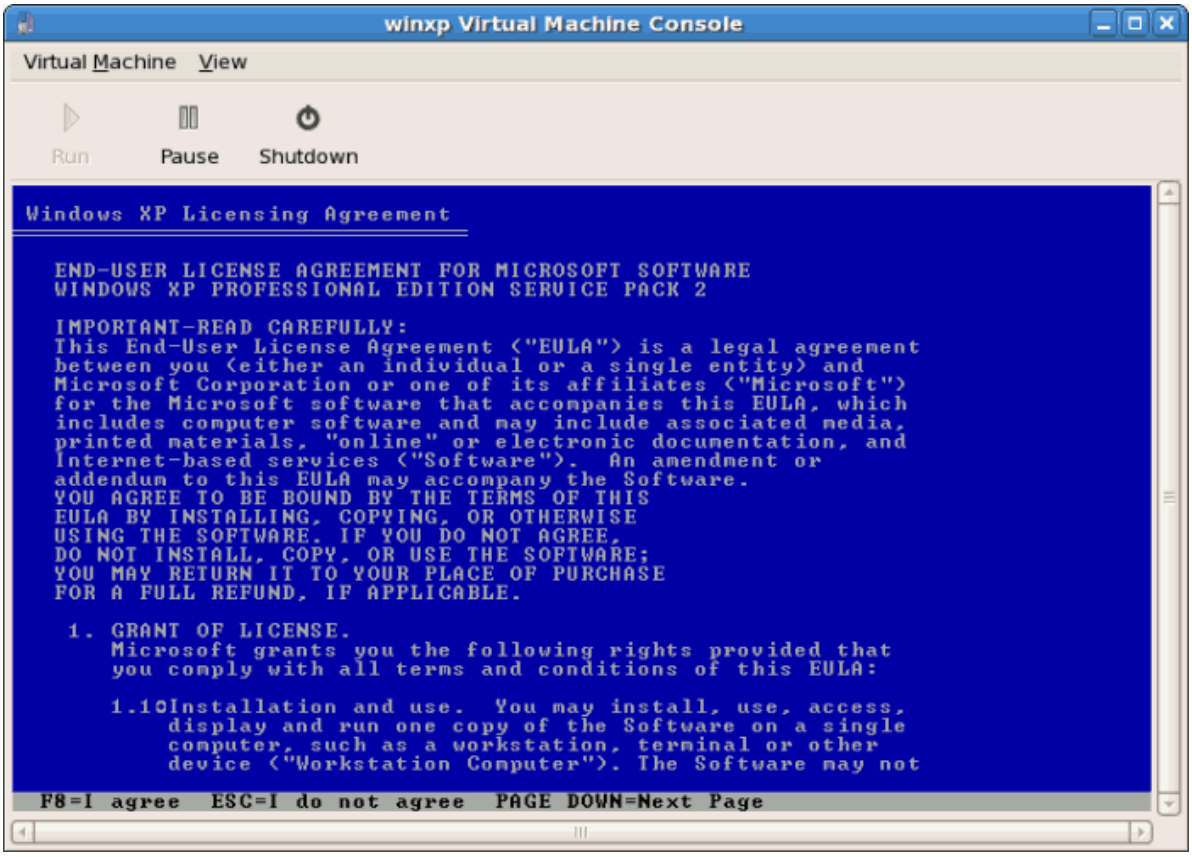
9. Before the installation continues you will see the summary screen. Press **Finish** to proceed to the guest installation:

10. You must make a hardware selection so open a console window quickly after the installation starts. Click **Finish** then switch to the **virt-manager** summary window and select your newly started Windows guest. Double click on the system name and the console window opens. Quickly and repeatedly press **F5** to select a new `HAL`, once you get the dialog box in the Windows install select the '`Generic i486 Platform`' tab. Scroll through selections with the **`Up`** and **`Down`** arrows.

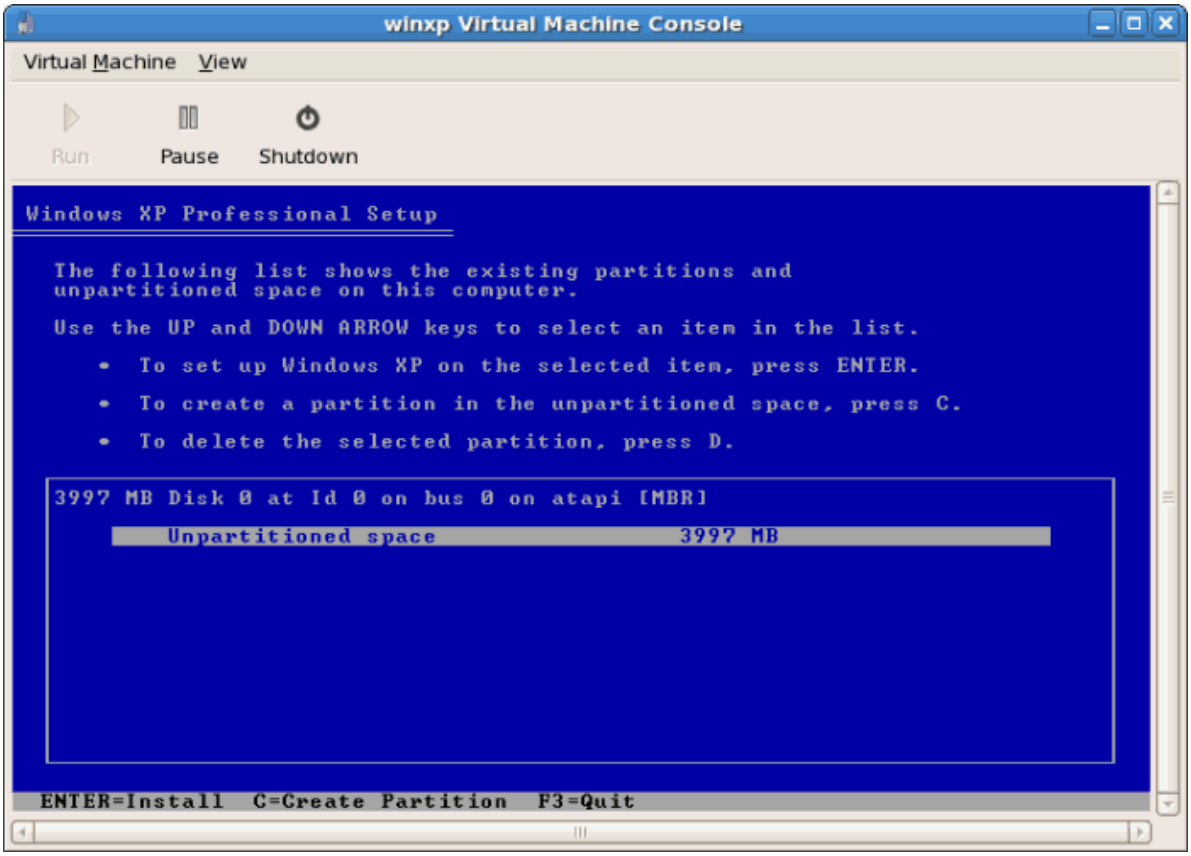


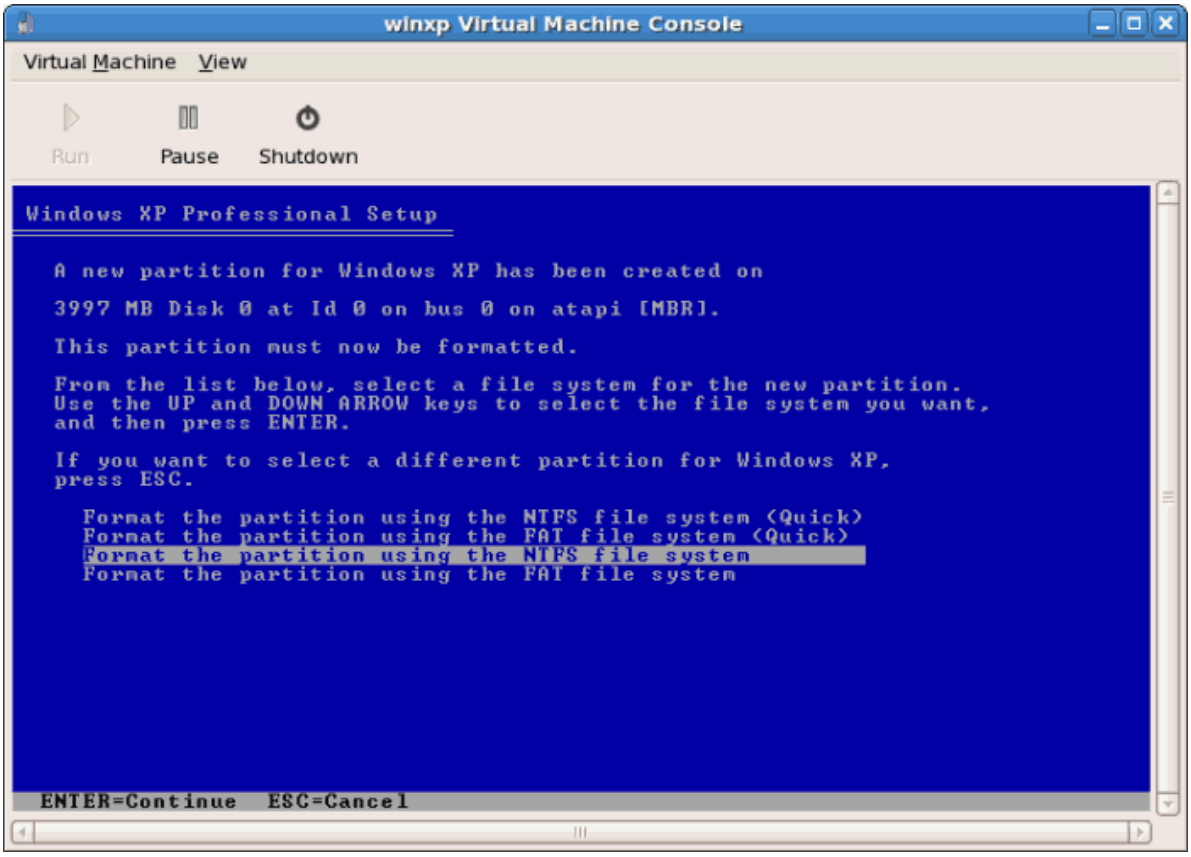
11. The installation continues with the standard Windows installation.


winxp Virtual Machine Console
Virtual Machine View
Run
Pause
Shutdown
Windows XP Professional Setup
Welcome to Setup.
This portion of the Setup program prepares Microsoft(R) Windows(R) XP to run on your computer.
• To set up Windows XP now, press ENTER.
• To repair a Windows XP installation using Recovery Console, press R.
• To quit Setup without installing Windows XP, press F3.
ENTER=Continue R=Repair F3=Quit

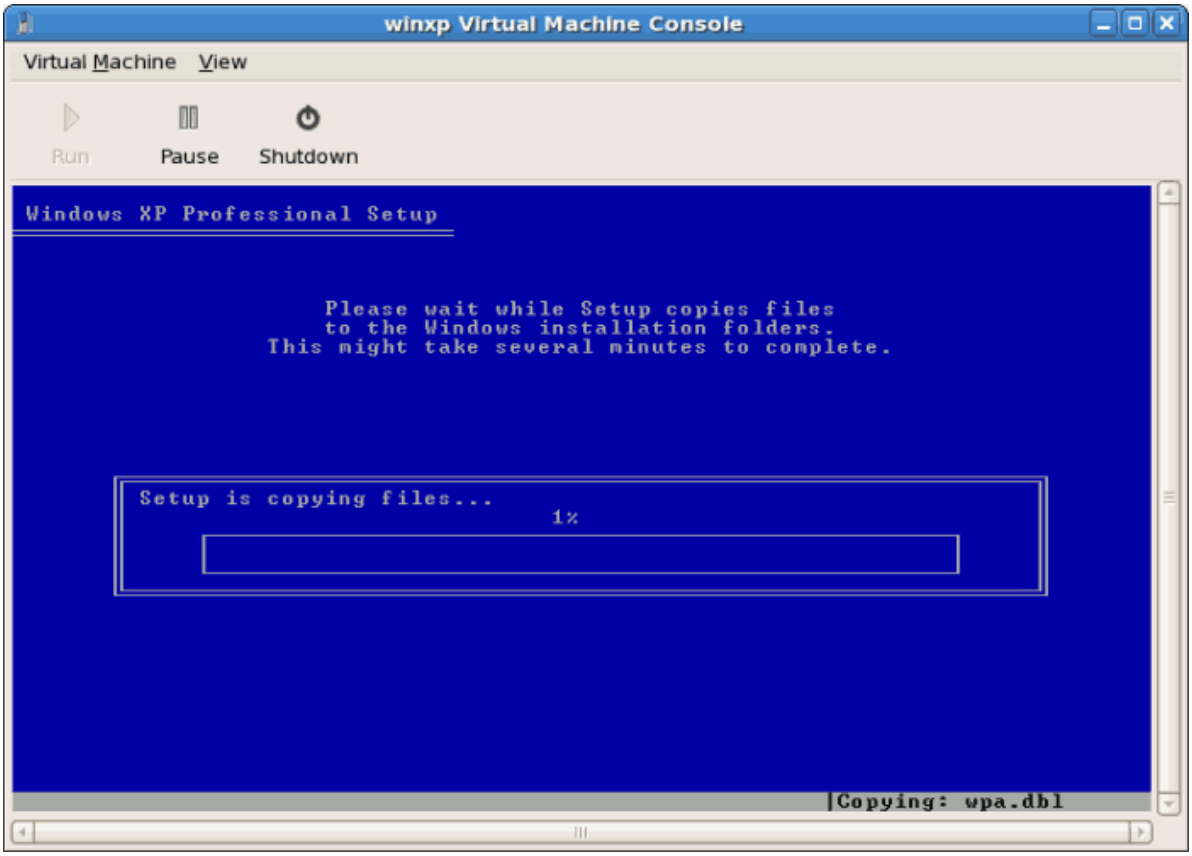

winxp Virtual Machine Console
Virtual Machine View
Run
Pause
Shutdown
Windows XP Licensing Agreement
END-USER LICENSE AGREEMENT FOR MICROSOFT SOFTWARE
WINDOWS XP PROFESSIONAL EDITION SERVICE PACK 2
IMPORTANT-READ CAREFULLY:
This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Microsoft Corporation or one of its affiliates ("Microsoft") for the Microsoft software that accompanies this EULA, which includes computer software and may include associated media, printed materials, "online" or electronic documentation, and Internet-based services ("Software"). An amendment or addendum to this EULA may accompany the Software.
YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE. IF YOU DO NOT AGREE, DO NOT INSTALL, COPY, OR USE THE SOFTWARE; YOU MAY RETURN IT TO YOUR PLACE OF PURCHASE FOR A FULL REFUND, IF APPLICABLE.
1. GRANT OF LICENSE.
Microsoft grants you the following rights provided that you comply with all terms and conditions of this EULA:
1.1 Installation and use. You may install, use, access, display and run one copy of the Software on a single computer, such as a workstation, terminal or other device ("Workstation Computer"). The Software may not
F8=I agree ESC=I do not agree PAGE DOWN=Next Page

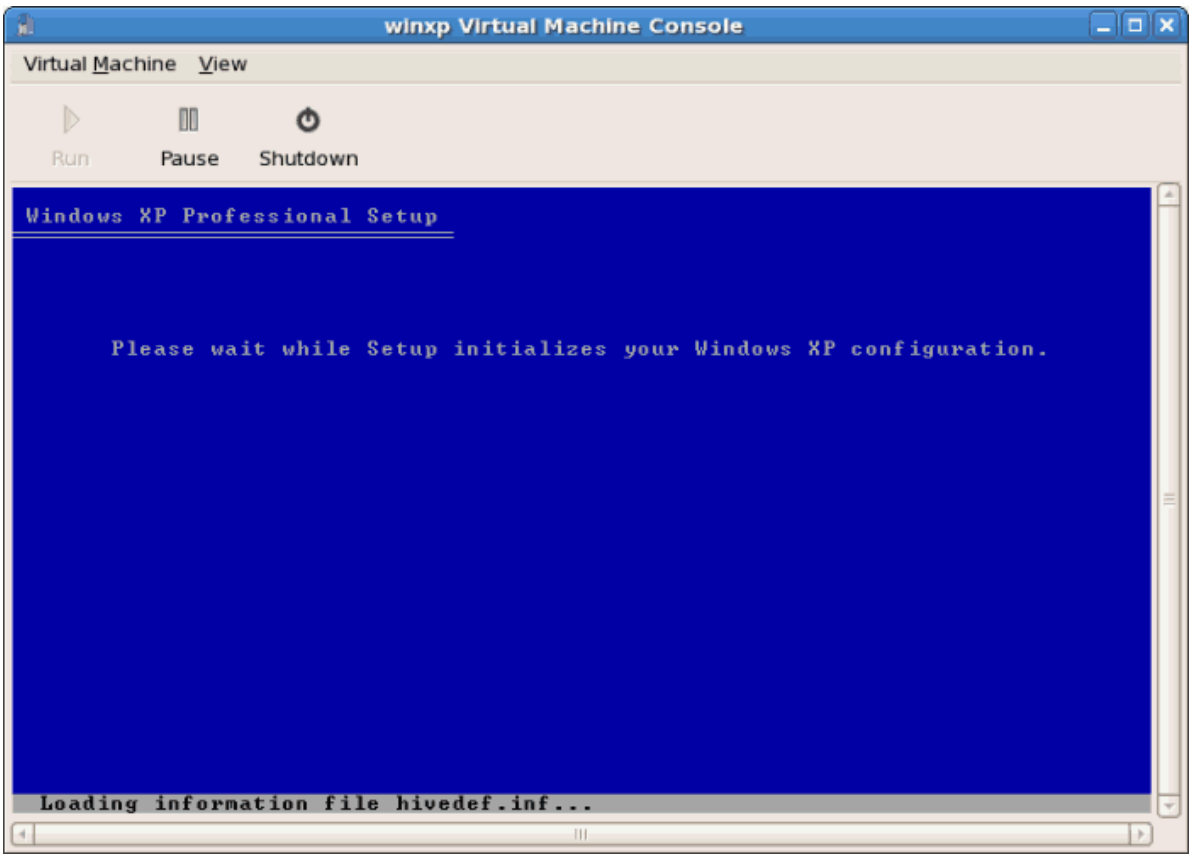
12. Partition the hard drive when prompted.



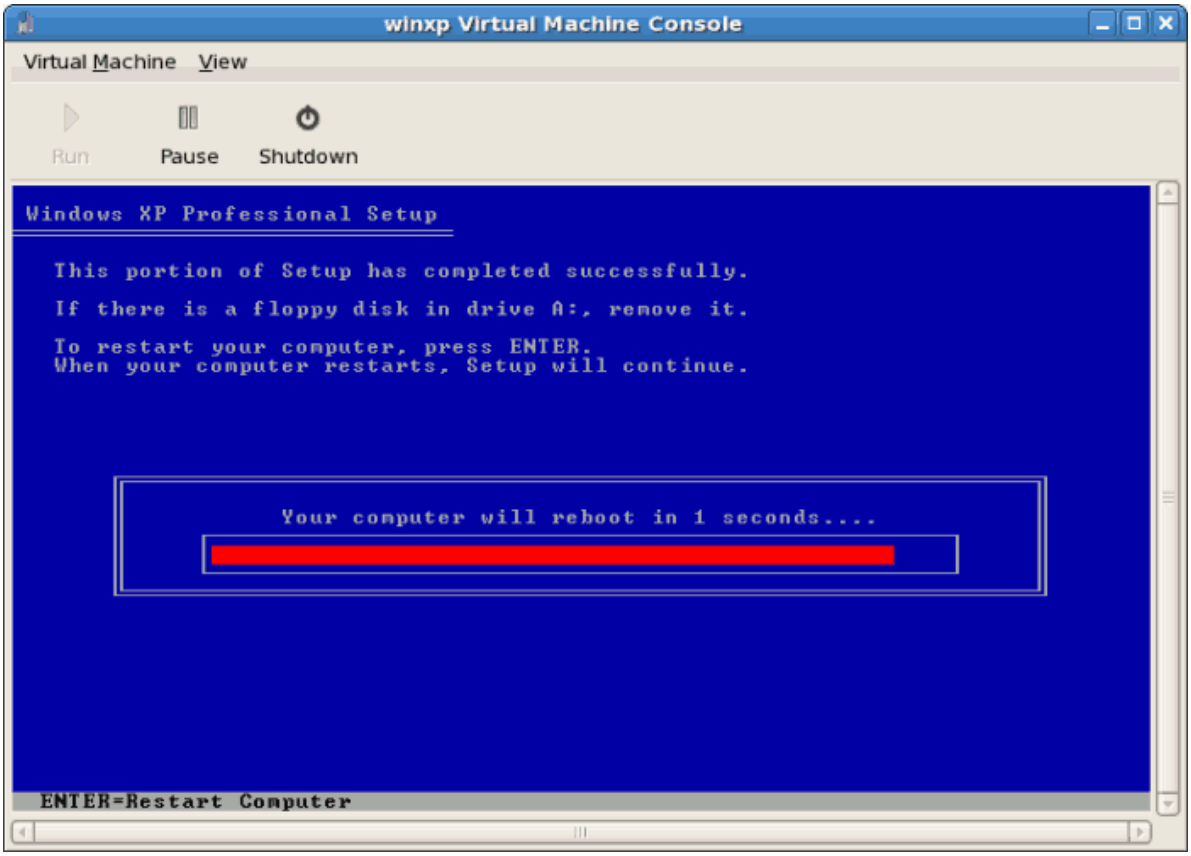


13. After the drive is formatted, Windows starts copying the files to the hard drive.





14. The files are copied to the storage device, Windows now reboots.

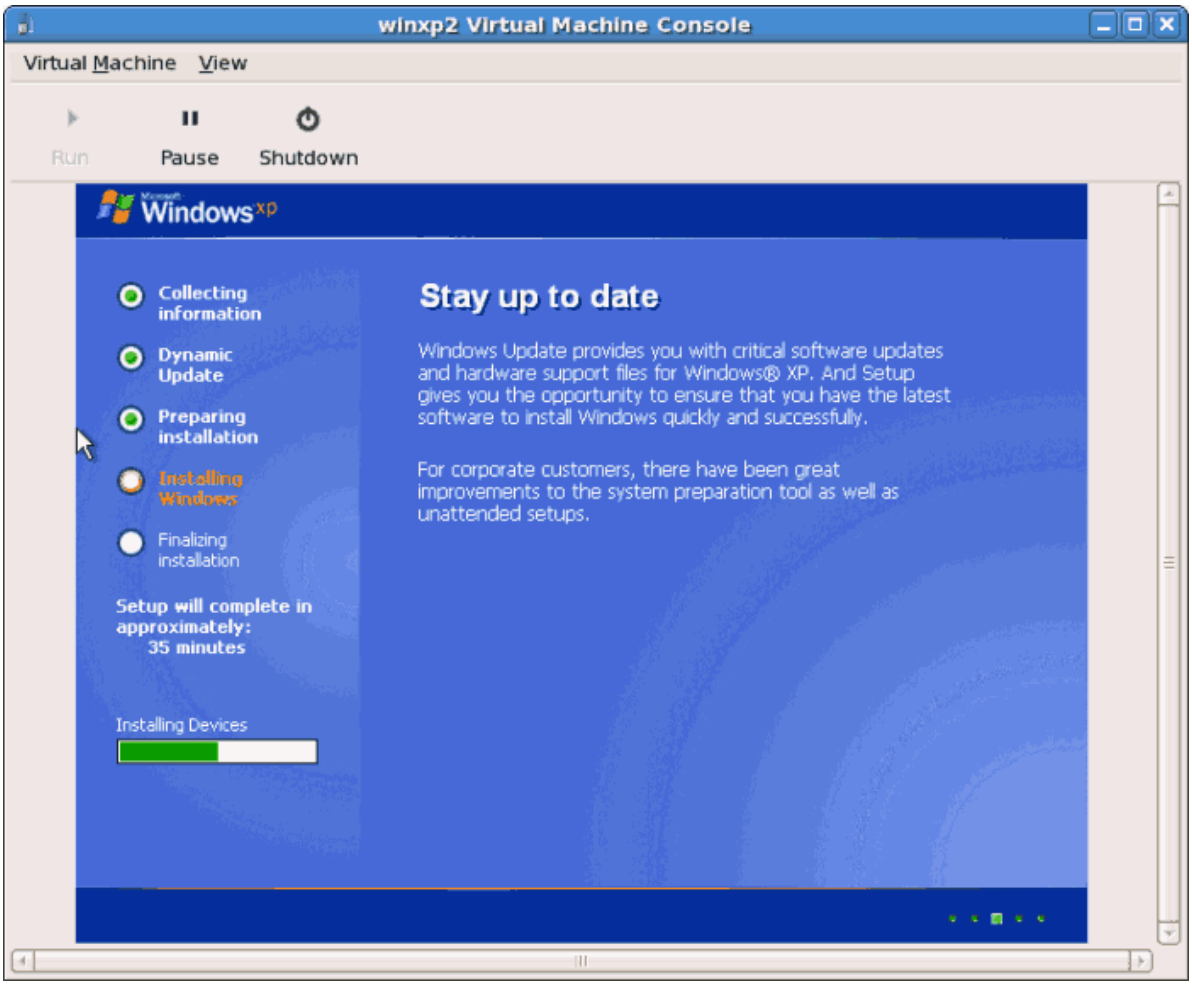


15. Restart your Windows guest:

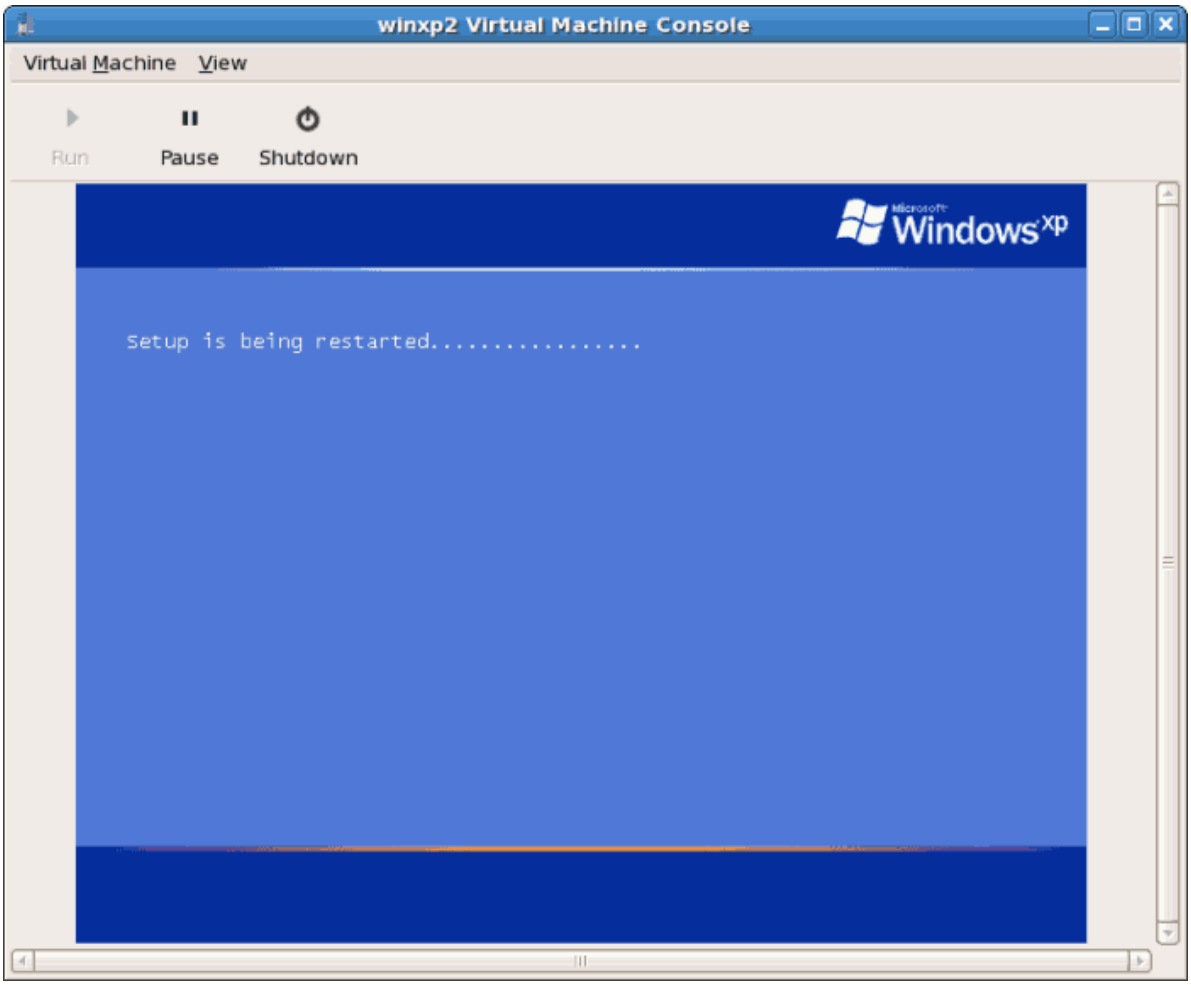
```
# virsh start WindowsGuest
```

Where *WindowsGuest* is the name of your virtual machine.

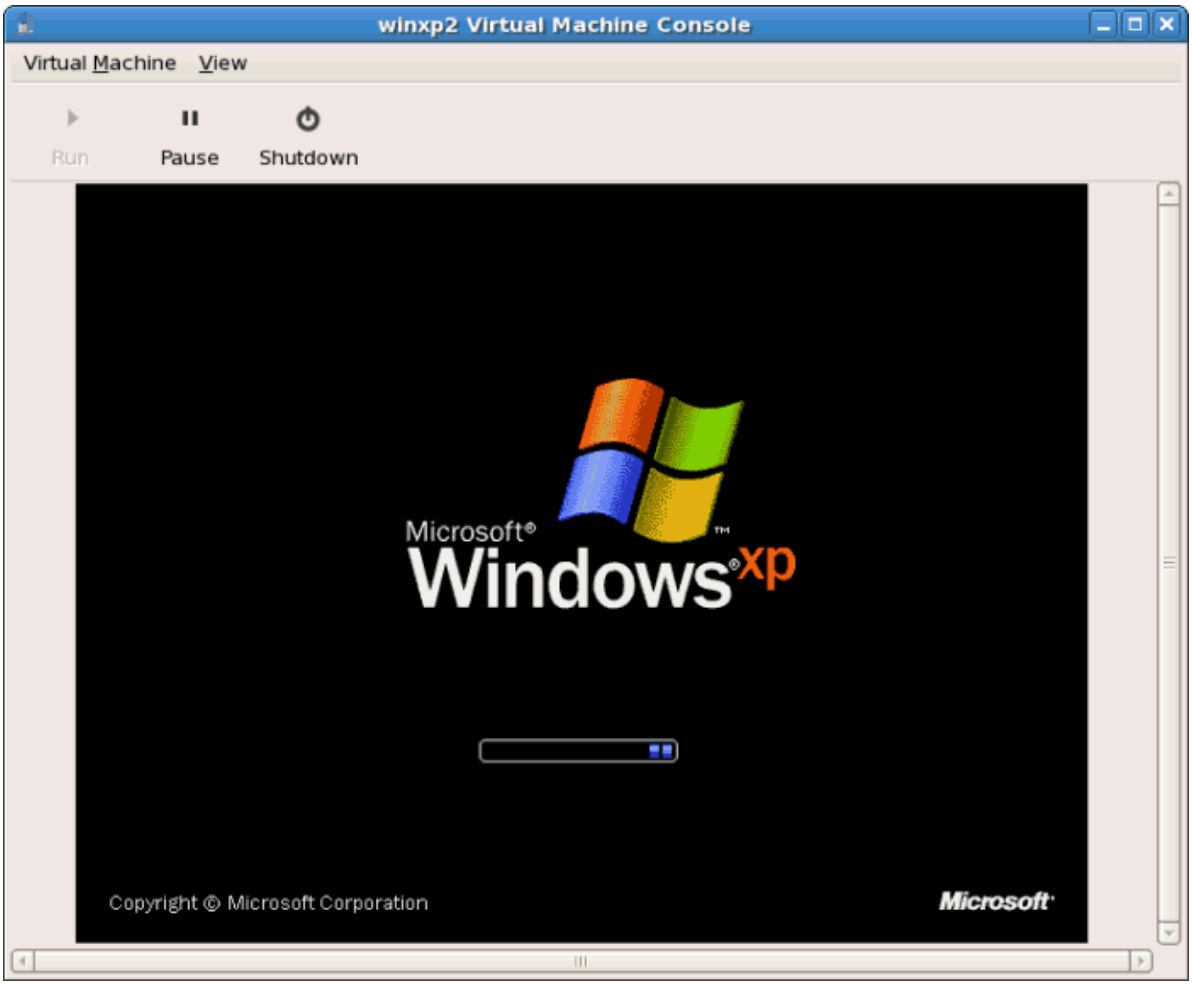
16. When the console window opens, you will see the setup phase of the Windows installation.



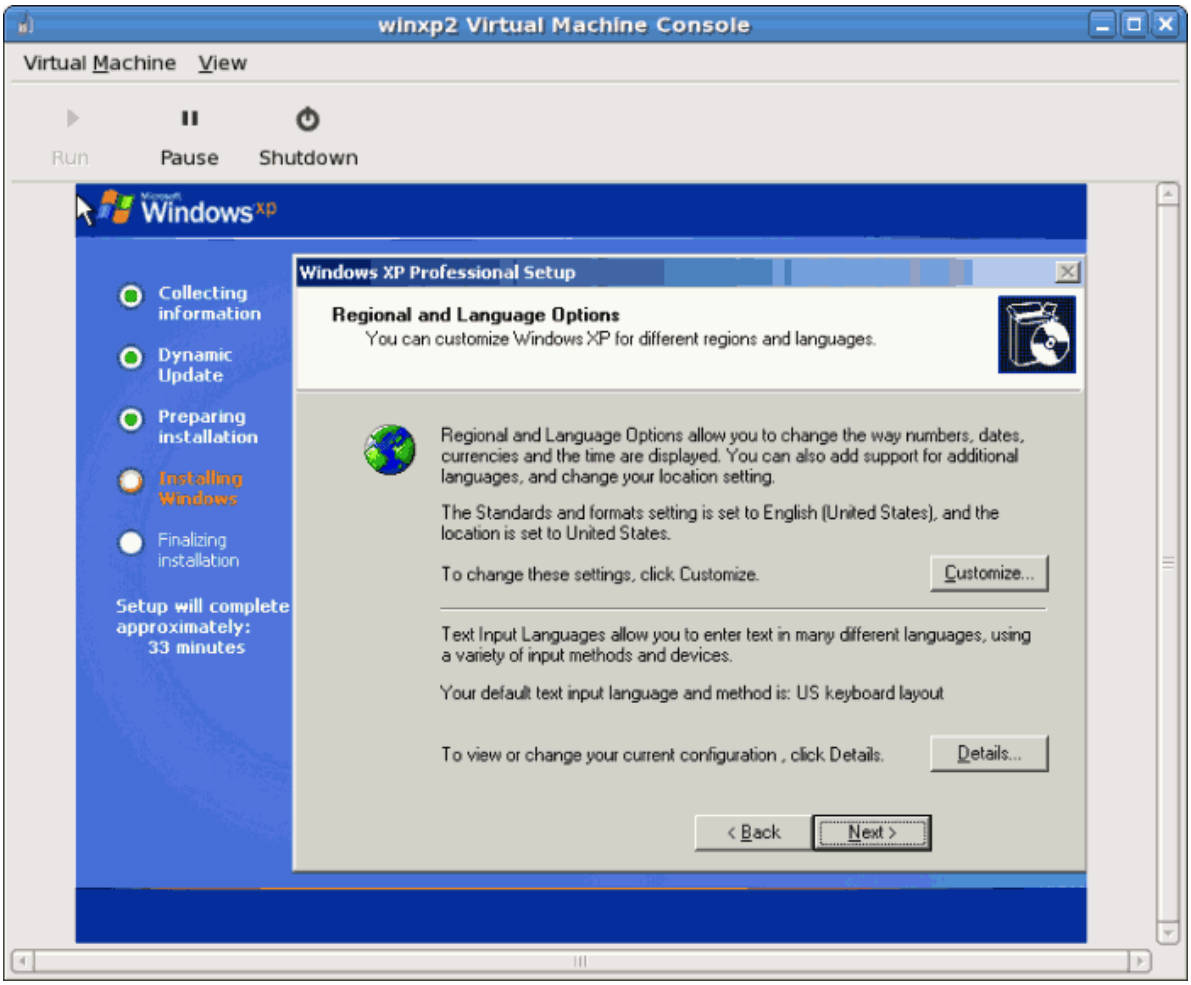
17. If your installation seems to get stuck during the setup phase, restart the guest with **`virsh reboot WindowsGuestName`**. When you restart the virtual machine, the **`Setup is being restarted`** message displays:



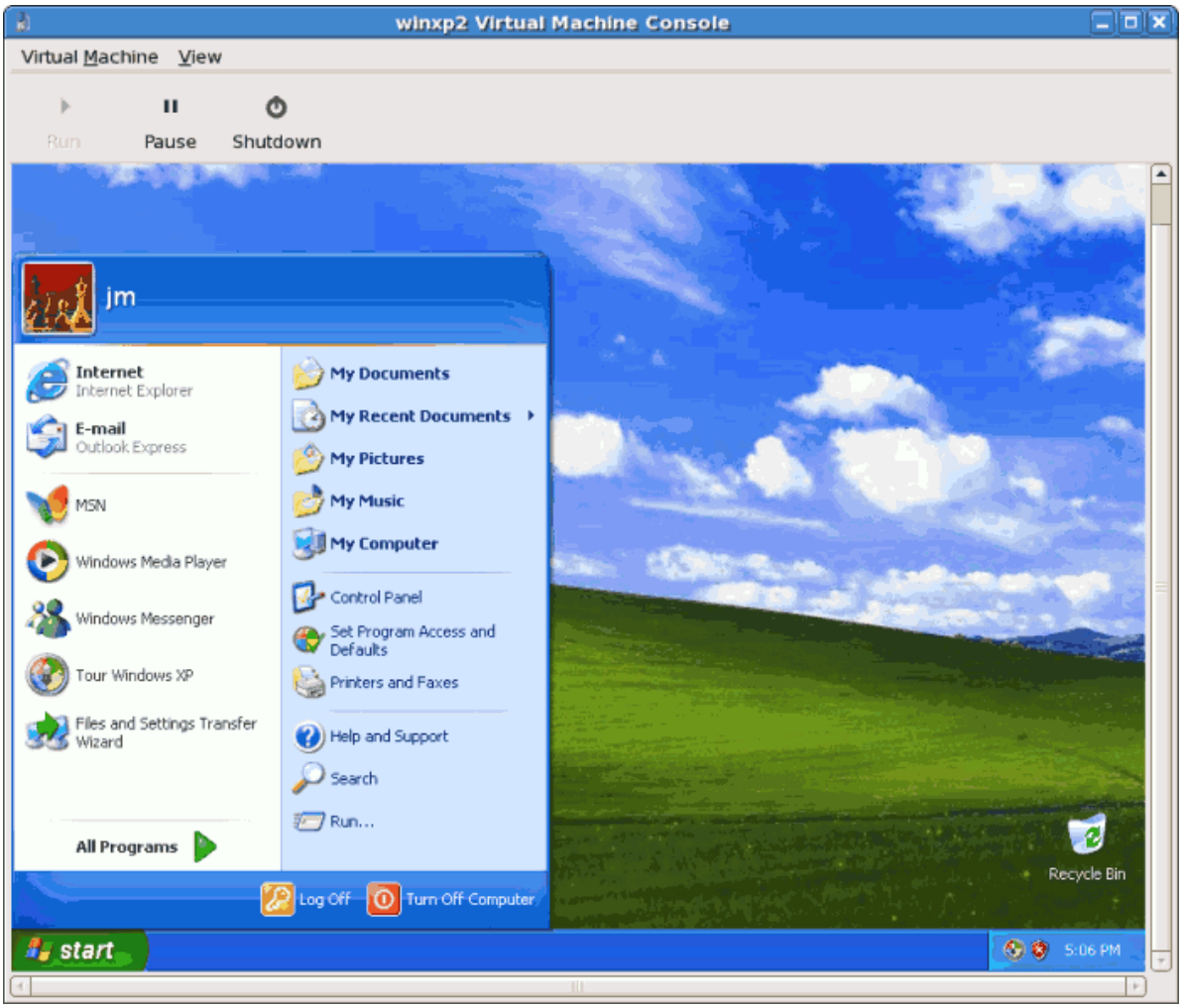
18. After setup has finished you will see the Windows boot screen:



19. Now you can continue with the standard setup of your Windows installation:



20. The setup process is complete.


winxp2 Virtual Machine Console
Virtual Machine View
Run
Pause
Shutdown
jm
Internet
Internet Explorer
E-mail
Outlook Express
MSN
Windows Media Player
Windows Messenger
Tour Windows XP
Files and Settings Transfer Wizard
All Programs
My Documents
My Recent Documents
My Pictures
My Music
My Computer
Control Panel
Set Program Access and Defaults
Printers and Faxes
Help and Support
Search
Run...
Log Off
Turn Off Computer
Recycle Bin
start
5:06 PM

Installing Windows Server 2003 as a fully virtualized guest

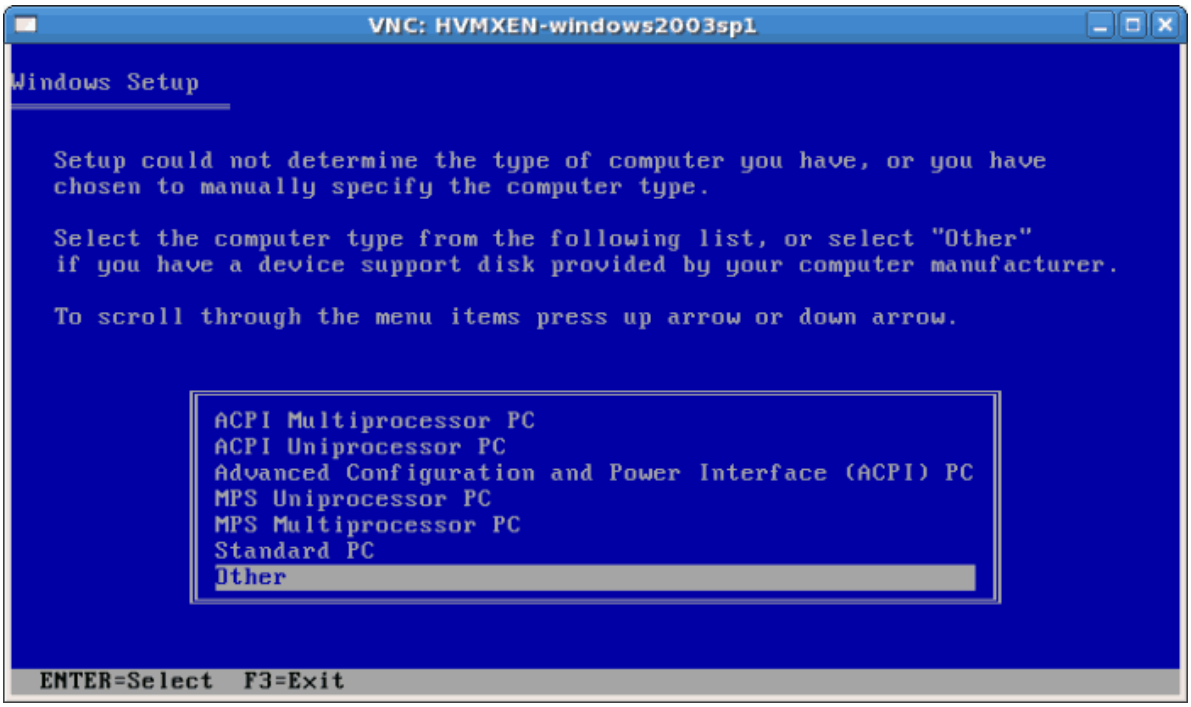
This chapter describes installing a fully virtualized Windows Server 2003 guest with the **virt-install** command. **virt-install** can be used instead of **virt-manager** This process is similar to the Windows XP installation covered in *Chapter 7, Installing Windows XP as a fully virtualized guest*.

1. Using **virt-install** for installing Windows Server 2003 as the console for the Windows guest opens the **virt-viewer** window promptly. The examples below installs a Windows Server 2003 guest with the **virt-install** command.

```
# virt-install --accelerate --hvm --connect qemu:///system \
   --name rhel3support  \
   --network network:default \
   --file=/var/lib/libvirt/images/windows2003sp2.img \
   --file-size=6 \
   --cdrom=/var/lib/libvirt/images/ISOs/WIN/en_windows_server_2003_sp1.iso \
   --vnc --ram=1024
```

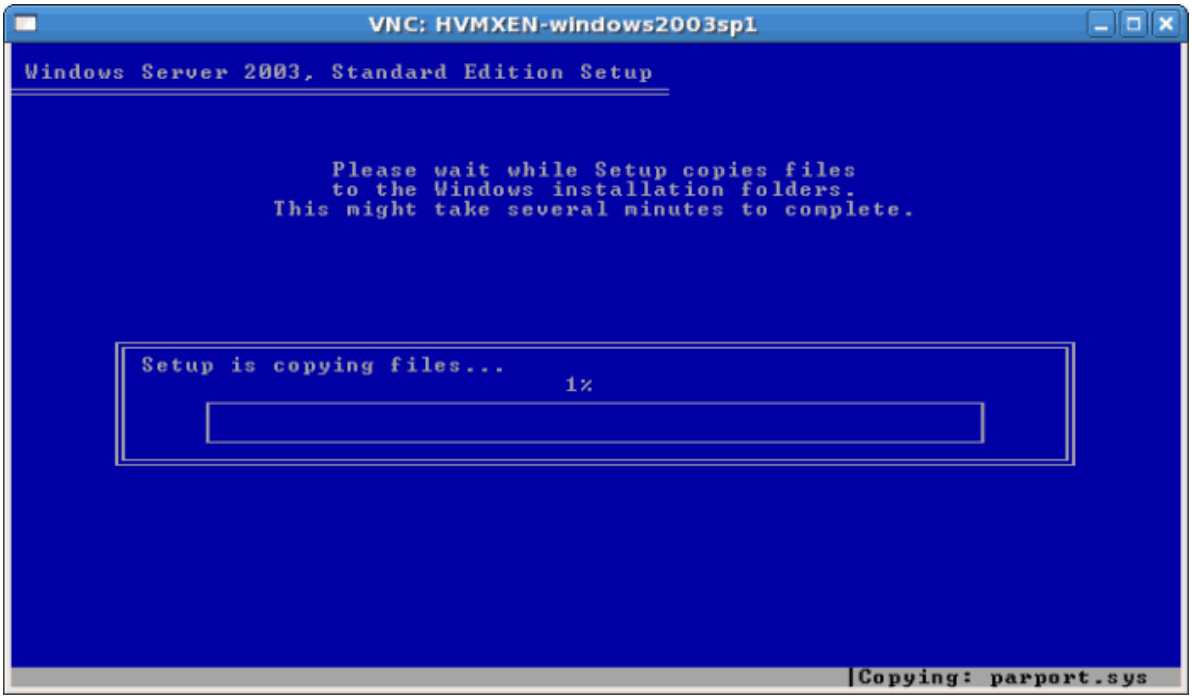
Example 8.1. KVM virt-install

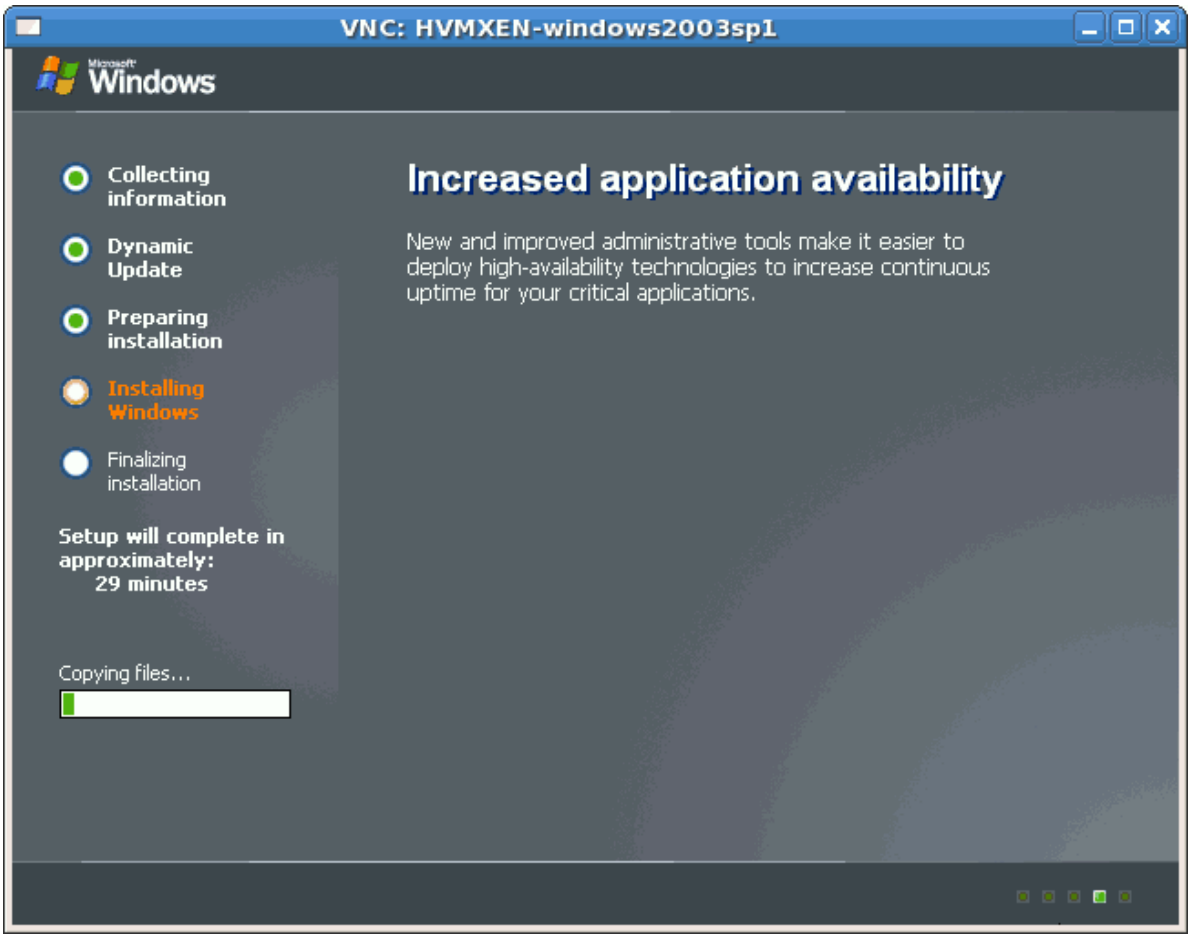
2. Once the guest boots into the installation you must quickly press **F5**. If you do not press **F5** at the right time you will need to restart the installation. Pressing **F5** allows you to select different **HAL** or **Computer Type**. Choose **Standard PC** as the **Computer Type**. Changing the **Computer Type** is required for Windows Server 2003 virtualized guests.



3. Complete the rest of the installation.

VNC: HVMXEN-windows2003sp1
Windows Server 2003, Standard Edition Setup
Welcome to Setup.
This portion of the Setup program prepares Microsoft(R) Windows(R) to run on your computer.
• To set up Windows now, press ENTER.
• To repair a Windows installation using Recovery Console, press R.
• To quit Setup without installing Windows, press F3.
ENTER=Continue R=Repair F3=Quit


VNC: HVMXEN-windows2003sp1
Windows Server 2003, Standard Edition Setup
Please wait while Setup copies files to the Windows installation folders.
This might take several minutes to complete.
Setup is copying files...
1%
Copying: parport.sys



4. Windows Server 2003 is now installed as a fully virtualized guest.

Installing Windows Server 2008 as a fully virtualized guest

This section covers installing a fully virtualized Windows Server 2008 guest on Fedora.

Procedure 9.1. Installing Windows Server 2008 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 hypervisor. Note that presently the KVM hypervisor is named **`qemu`**.

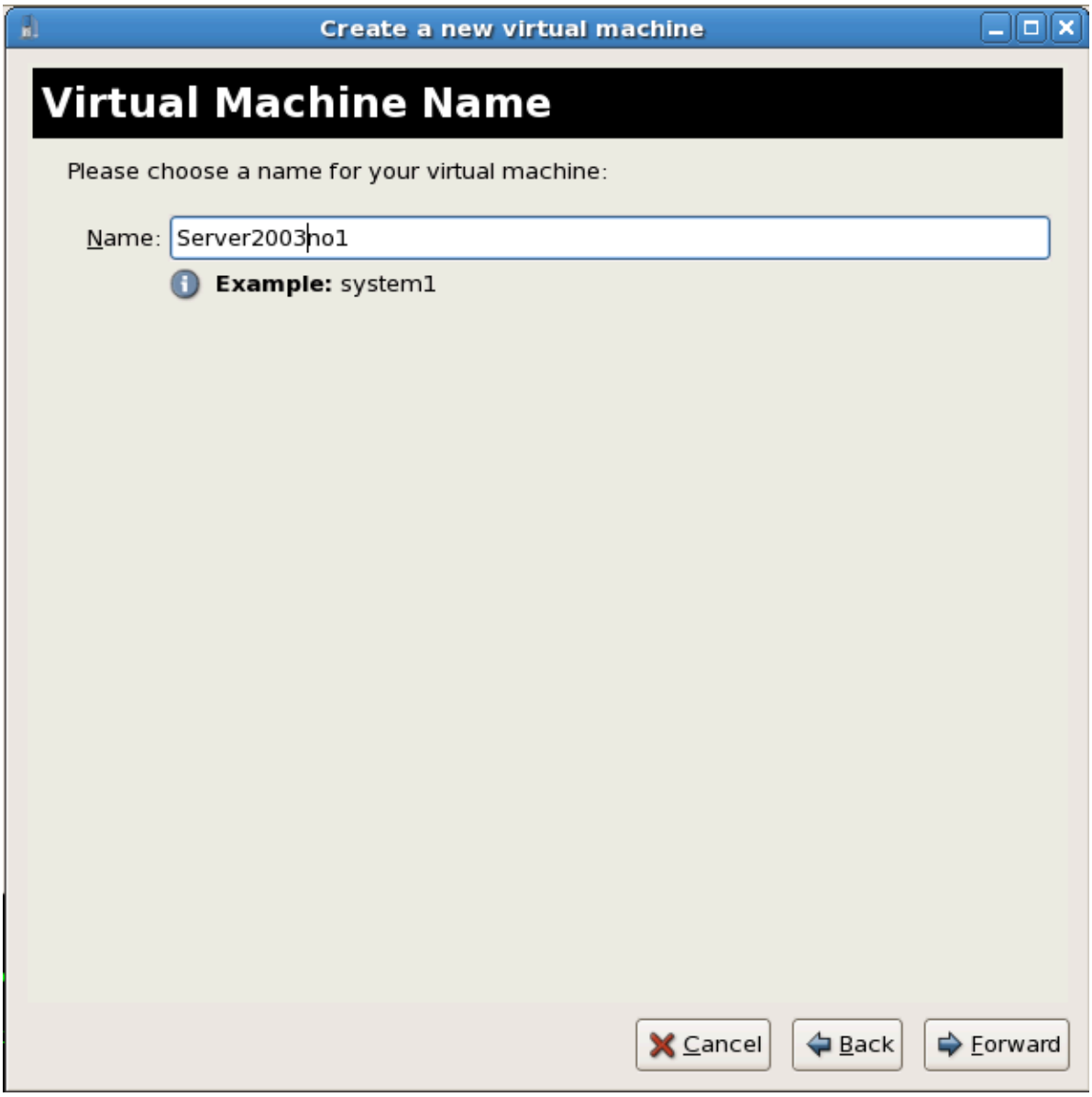
   Once the option is selected the **New** button becomes available. Press the **New** button.

3. **Start the new virtual machine wizard**
   Pressing the **New** button starts the virtual machine creation wizard.

Press **Forward** to continue.

4. **Name the guest**

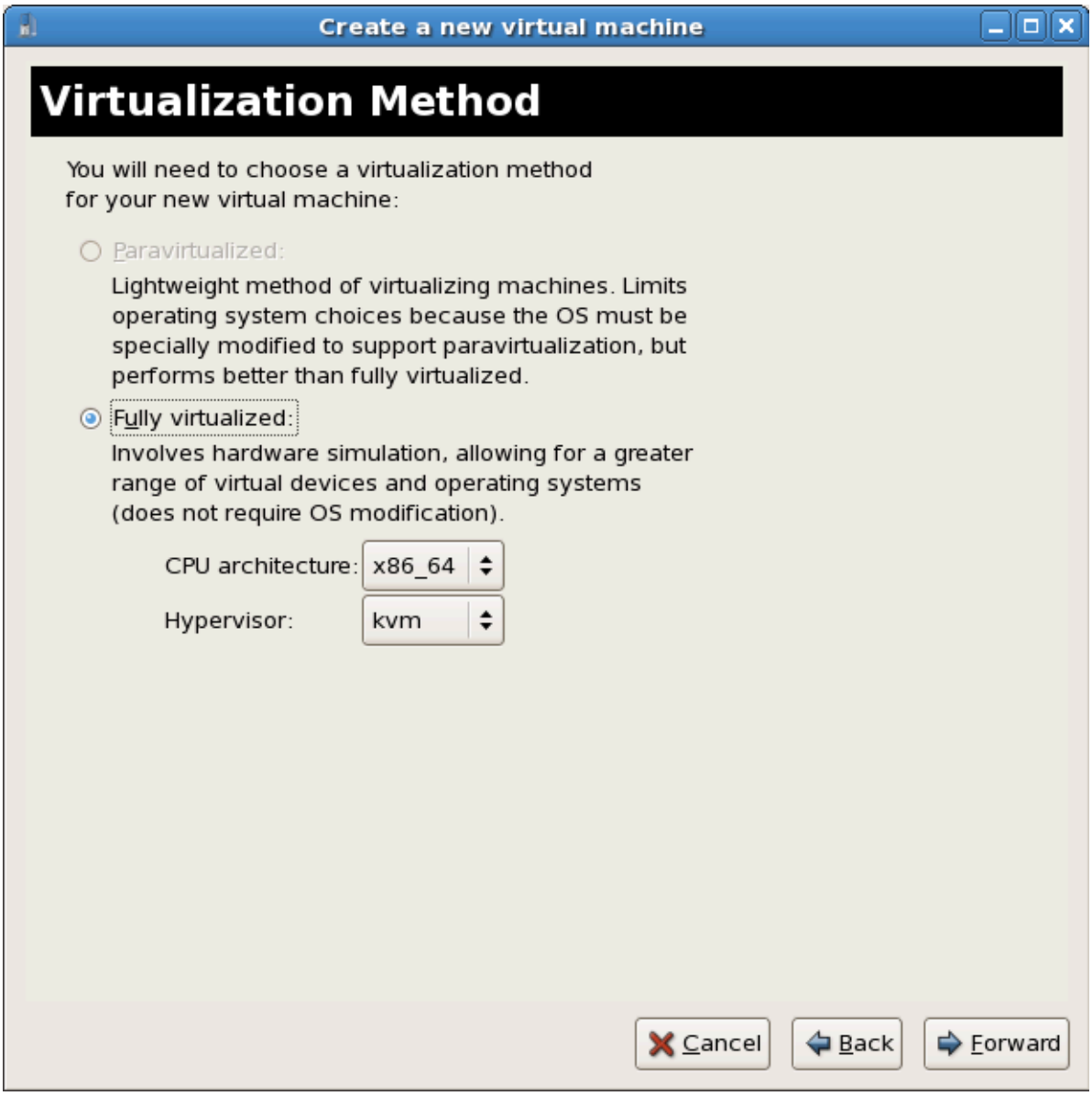
   The following characters are allowed in guest names: '_', '**.**' and '**-**' characters.



Press **Forward** to continue.

5. **Choose a virtualization method**
   The **Choosing a virtualization method** window appears.

   Full virtualization requires a processor with the AMD 64 and the AMD-V extensions or a processor with the Intel 64 and Intel VT extensions. If the virtualization extensions are not present, KVM will not be avilable.



Press **Forward** to continue.

6. **Select the installation method**
   For all versions of Windows you must use **local install media**, either an ISO image or physical optical media.

   PXE may be used if you have a PXE server configured for Windows network installation. PXE Windows installation is not covered by this guide.

   Set **OS Type** to **Windows** and **OS Variant** to **Microsoft Windows 2008** as shown in the screenshot.

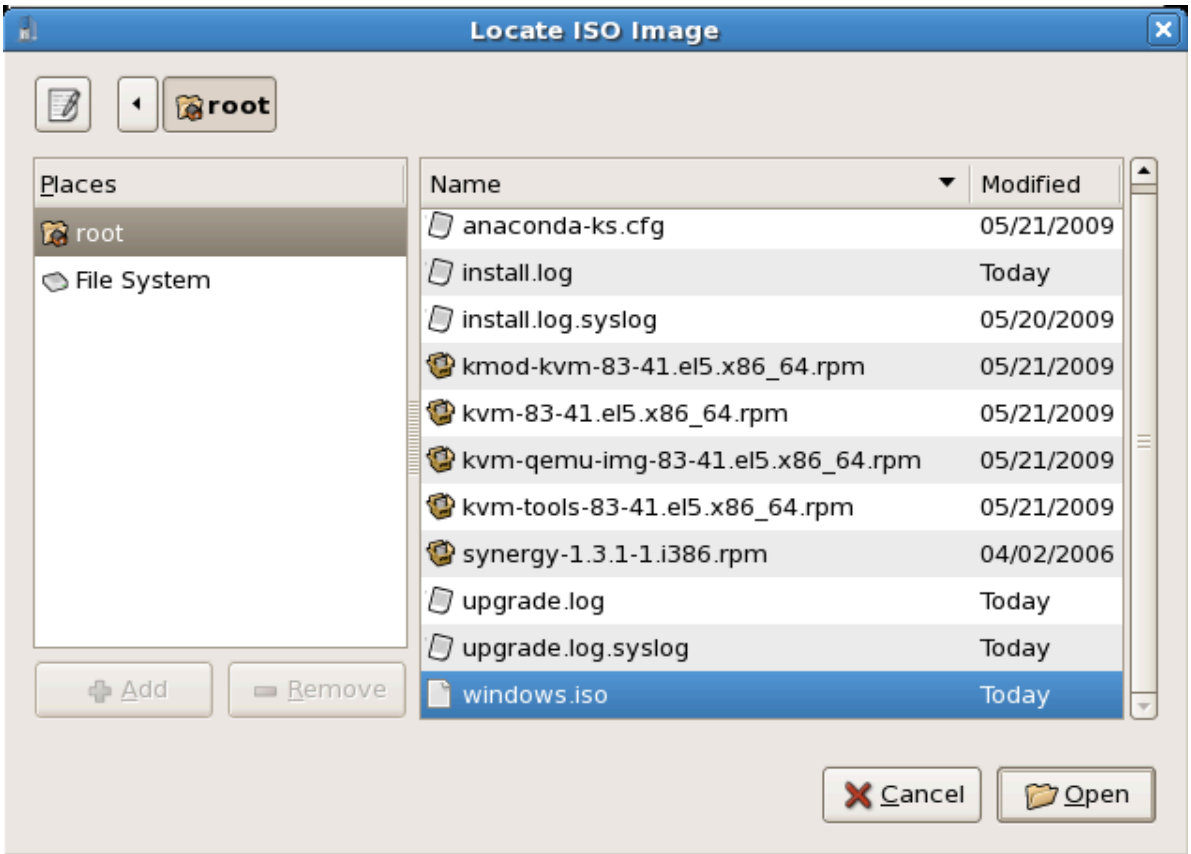
Press **Forward** to continue.

7. **Locate installation media**

   Select ISO image location or CD-ROM or DVD device. This example uses an ISO file image of the Windows Server 2008 installation CD.

   a. Press the **Browse** button.

   b. Search to the location of the ISO file and select it.



Press **Open** to confirm your selection.

c. The file is selected and ready to install.

Press **Forward** to continue.

> **Image files and SELinux**
> For ISO image files and guest storage images, the recommended directory to use is the **`/var/lib/libvirt/images/`** directory. Any other location may require additional configuration for SELinux, refer to *Section 19.2, "SELinux and virtualization"* for details.

8. **Storage setup**
   Assign a physical storage device (**Block device**) or a file-based image (**File**). File-based images must be stored in the **`/var/lib/libvirt/images/`** directory. Assign sufficient space for your virtualized guest and any applications the guest requires.

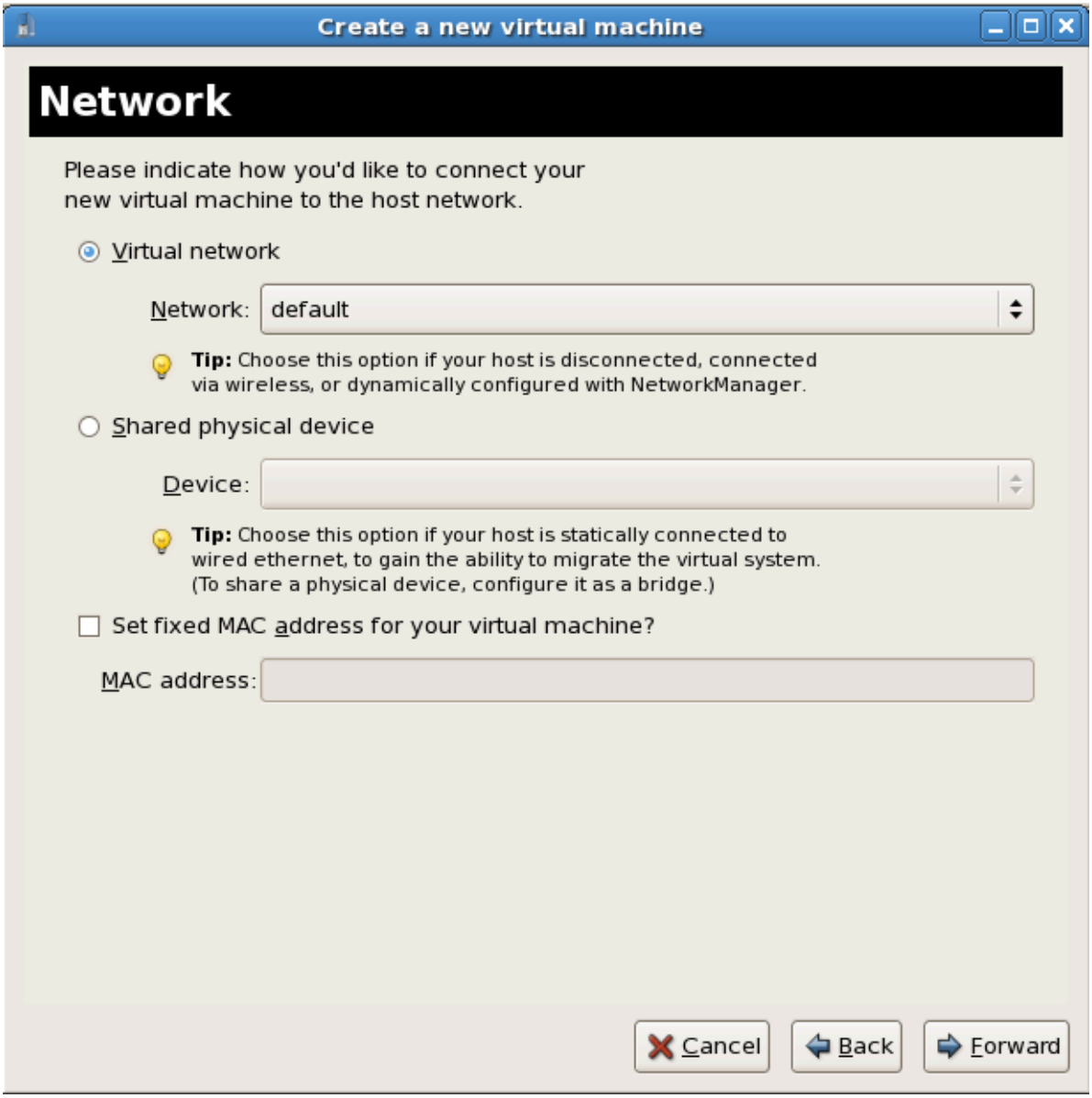
Press **Forward** to continue.

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. Use the virtual network option for wireless networks.

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



Press **Forward** to continue.

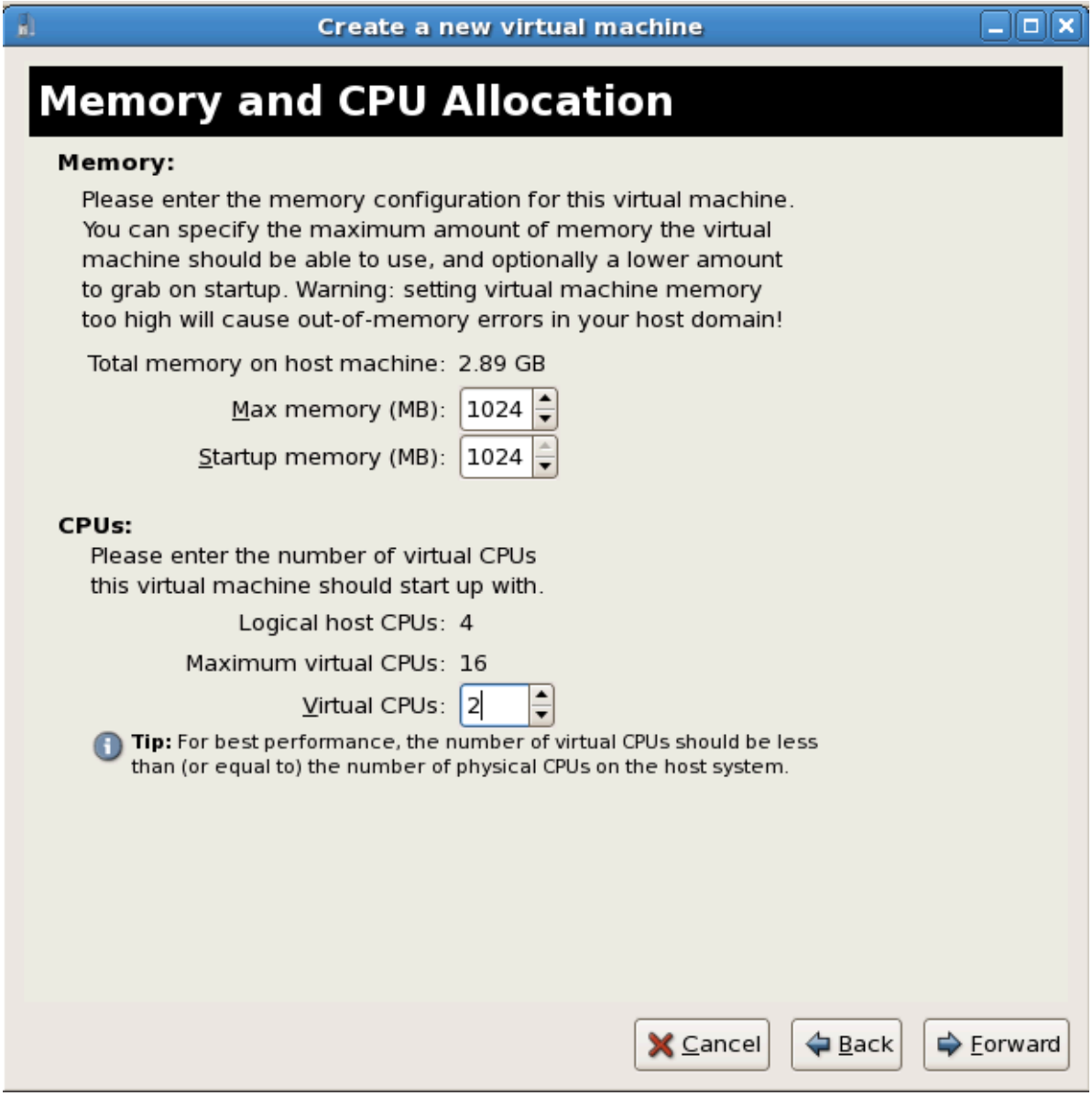
10. **Memory and CPU allocation**

    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, 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 has

a significant, negative effect on guest and host performance due to processor context switching overheads.

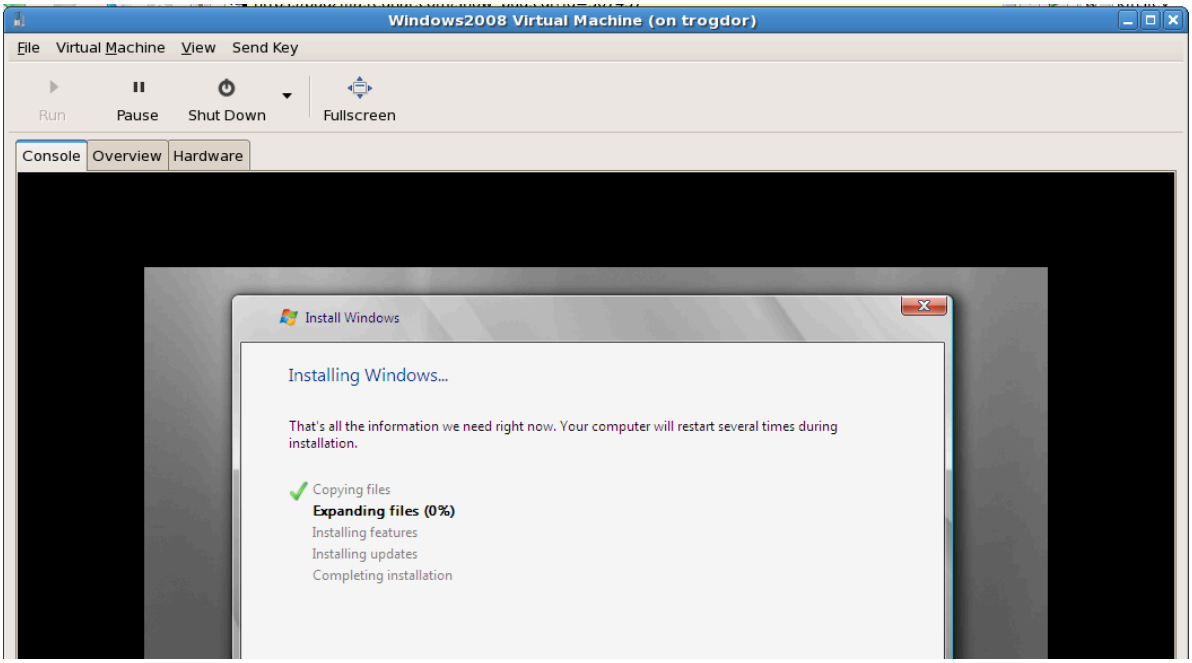


Press **Forward** to continue.

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

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

12. **Installing Windows**



Complete the Windows Server 2008 installation sequence. The installation sequence is not covered by this guide, refer to Microsoft's *documentation*[1] for information on installing Windows.

Part III. Configuration

Configuring virtualization in Fedora

These chapters cover configuration procedures for various advanced virtualization tasks. These tasks include adding network and storage devices, enhancing security, improving performance, and using the *para-virtualized drivers* on fully virtualized guests.

Virtualized storage devices

This chapter covers installing and configuring storage devices in virtualized guests. The term block devices refers to various forms of storage devices. All the procedures in this chapter work with both Xen and KVM hypervisors.

Valid disk targets

The target variable in libvirt configuration files accepts only the following device names:

- **/dev/xvd[a to z][1 to 15]**

  Example: **/dev/xvdb13**

- **/dev/xvd[a to i][a to z][1 to 15]**

  Example: **/dev/xvdbz13**

- **/dev/sd[a to p][1 to 15]**

  Example: **/dev/sda1**

- **/dev/hd[a to t][1 to 63]**

  Example: **/dev/hdd3**

10.1. Creating a virtualized floppy disk controller

Floppy disk controllers are required for a number of older operating systems, especially for installing drivers. Presently, physical floppy disk devices cannot be accessed from virtualized guests. However, creating and accessing floppy disk images from virtualized floppy drives is should work. This section covers creating a virtualized floppy device.

An image file of a floppy disk is required. Create floppy disk image files with the **dd** command. Replace **/dev/fd0** with the name of a floppy device and name the disk appropriately.

```
# dd if=/dev/fd0 of=~/legacydrivers.img
```

This example uses a guest created with **virt-manager** running a fully virtualized Fedora installation with an image located in **/var/lib/libvirt/images/Fedora.img**. The Xen hypervisor is used in the example.

1. Create the XML configuration file for your guest image using the **virsh** command on a running guest.

   ```
   # virsh dumpxml Fedora > rhel5FV.xml
   ```

   This saves the configuration settings as an XML file which can be edited to customize the operations and devices used by the guest. For more information on using the virsh XML configuration files, refer to *Chapter 31, Creating custom libvirt scripts*.

2. Create a floppy disk image for the guest.

```
# dd if=/dev/zero of=/var/lib/libvirt/images/Fedora-floppy.img bs=512 count=2880
```

3. Add the content below, changing where appropriate, to your guest's configuration XML file. This example is an emulated floppy device using a file-based image.

```
<disk type='file' device='floppy'>
 <source file='/var/lib/libvirt/images/Fedora-floppy.img'/>
 <target dev='fda'/>
</disk>
```

4. Force the guest to stop. To shut down the guest gracefully, use the **`virsh shutdown`** command instead.

```
# virsh destroy Fedora
```

5. Restart the guest using the XML configuration file.

```
# virsh create Fedora.xml
```

The floppy device is now available in the guest and stored as an image file on the host.

10.2. Adding storage devices to guests

This section covers adding storage devices to virtualized guest. Additional storage can only be added after guests are created. The might work storage devices and protocol include:

- local hard drive partitions,
- logical volumes,
- Fibre Channel or iSCSI directly connected to the host.
- File containers residing in a file system on the host.
- **NFS** file systems mounted directly by the virtual machine.
- iSCSI storage directly accessed by the guest.
- Cluster File Systems (**GFS**).

Adding file-based storage to a guest

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

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

a. Create a sparse file using the **dd** command. Sparse files are not recommended due to data integrity and performance issues. Sparse files are created much faster and can used for testing but should not be used in production environments.

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

b. Non-sparse, pre-allocated files are recommended for file-based storage images. Create a non-sparse file, execute:

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

Both commands create a 400MB file which can be used as additional storage for a virtualized guest.

2. Dump the configuration for the guest. In this example the guest is called *Guest1* and the file is saved in the users home directory.

```
# virsh dumpxml Guest1 > ~/Guest1.xml
```

3. Open the configuration file (*Guest1.xml* in this example) in a text editor. Find the **<disk>** elements, these elements describe storage devices. The following is an example disk element:

```
<disk type='file' device='disk'>
    <driver name='tap' type='aio'/>
    <source file='/var/lib/libvirt/images/Guest1.img'/>
    <target dev='xvda'/>
</disk>
```

4. Add the additional storage by duplicating or writing a new **<disk>** element. Ensure you specify a device name for the virtual block device attributes. These attributes must be unique for each guest configuration file. The following example is a configuration file section which contains an additional file-based storage container named **FileName.img**.

```
<disk type='file' device='disk'>
    <driver name='tap' type='aio'/>
    <source file='/var/lib/libvirt/images/Guest1.img'/>
    <target dev='xvda'/>
</disk>
<disk type='file' device='disk'>
    <driver name='tap' type='aio'/>
    <source file='/var/lib/libvirt/images/FileName.img'/>
    <target dev='hda'/>
</disk>
```

5. Restart the guest from the updated configuration file.

```
# virsh create Guest1.xml
```

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

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

   a. Press *n* for a new partition.

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

   b. Press *p* for a primary partition.

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

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

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

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

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

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

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

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

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

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

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

   h. Enter *83* for a linux partition.

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

i. write changes to disk and quit.

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

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

```
# mke2fs -j /dev/hdb
```

7. Mount the disk on the guest.

```
# mount /dev/hdb1 /myfiles
```

The guest now has an additional virtualized file-based storage device.

Adding hard drives and other block devices to a guest

System administrators use additional hard drives for to provide more storage space or to separate system data from user data. This procedure, *Procedure 10.1, "Adding physical block devices to virtualized guests"*, describes how to add a hard drive on the host to a virtualized guest.

The procedure works for all physical block devices, this includes CD-ROM, DVD and floppy devices.

Block device security

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

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.

Procedure 10.1. Adding physical block devices to virtualized guests

1. Physically attach the hard disk device to the host. Configure the host if the drive is not accessible by default.

2. Configure the device with **`multipath`** and persistence on the host if required.

3. Use the **`virsh attach`** command. Replace: *`myguest`* with your guest's name, *`/dev/hdb1`* with the device to add, and *hdc* with the location for the device on the guest. The *hdc* must be an unused device name. Use the *hd** notation for Windows guests as well, the guest will recognize the device correctly.

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

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

```
# virsh attach-disk myguest
     /dev/hdb1
     hdc --driver tap --mode readonly
```

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

10.3. Configuring persistent storage in Fedora

This section is for systems with external or networked storage; that is, Fibre Channel or iSCSI based storage devices. It is recommended that those systems have persistent device names configured for your hosts. This assists live migration as well as providing consistent device names and storage for multiple virtualized systems.

Universally Unique Identifiers(UUIDs) are a standardized method for identifying computers and devices in distributed computing environments. This sections uses UUIDs to identify iSCSI or Fibre Channel LUNs. UUIDs persist after restarts, disconnection and device swaps. The UUID is similar to a label on the device.

Systems which are not running **multipath** must use *Single path configuration*. Systems running **multipath** can use *Multiple path configuration*.

Single path configuration

This procedure implements *LUN* device persistence using **udev**. Only use this procedure for hosts which are not using **multipath**.

1. Edit the **/etc/scsi_id.config** file.

   a. Ensure the **options=-b** is line commented out.

   ```
   # options=-b
   ```

   b. Add the following line:

   ```
   options=-g
   ```

   This option configures **udev** to assume all attached SCSI devices return a UUID.

2. To display the UUID for a given device run the **scsi_id -g -s /block/sd*** command. For example:

   ```
   # scsi_id -g -s /block/sd*
   3600a0b8000132751000000015427b625e
   ```

   The output may vary from the example above. The output displays the UUID of the device `/dev/sdc`.

3. Verify the UUID output by the **scsi_id -g -s /block/sd*** command is identical from computer which accesses the device.

4. Create a rule to name the device. Create a file named **20-names.rules** in the **/etc/udev/rules.d** directory. Add new rules to this file. All rules are added to the same file using the same format. Rules follow this format:

```
KERNEL=="sd[a-z]", BUS=="scsi", PROGRAM="/sbin/scsi_id -g -s /block/%k", RESULT="UUID",
 NAME="devicename"
```

   Replace *UUID* and *devicename* with the UUID retrieved above, and a name for the device. This is a rule for the example above:

```
KERNEL="sd*", BUS="scsi", PROGRAM="/sbin/scsi_id -g -s",
 RESULT="3600a0b8000132751000000015427b625e", NAME="rack4row16"
```

   The **udev** daemon now searches all devices named **/dev/sd*** for the UUID in the rule. Once a matching device is connected to the system the device is assigned the name from the rule. In the a device with a UUID of 3600a0b8000132751000000015427b625e would appear as **/dev/rack4row16**.

5. Append this line to **/etc/rc.local**:

```
/sbin/start_udev
```

6. Copy the changes in the **/etc/scsi_id.config**, **/etc/udev/rules.d/20-names.rules**, and **/etc/rc.local** files to all relevant hosts.

```
/sbin/start_udev
```

Networked storage devices with configured rules now have persistent names on all hosts where the files were updated This means you can migrate guests between hosts using the shared storage and the guests can access the storage devices in their configuration files.

Multiple path configuration

The **multipath** package is used for systems with more than one physical path from the computer to storage devices. **multipath** provides fault tolerance, fail-over and enhanced performance for network storage devices attached to Fedora systems.

Implementing LUN persistence in a **multipath** environment requires defined alias names for your multipath devices. Each storage device has a UUID which acts as a key for the aliased names. Identify a device's UUID using the **scsi_id** command.

```
# scsi_id -g -s /block/sdc
```

The multipath devices will be created in the **/dev/mpath** directory. In the example below 4 devices are defined in **/etc/multipath.conf**:

```
multipaths {
 multipath {
 wwid  3600805f300159870000000000768a0019
 alias  oramp1
 }
 multipath {
 wwid  3600805f30015987000000000d643001a
 alias  oramp2
 }
 mulitpath {
 wwid  3600805f300159870000000000086fc001b
 alias  oramp3
 }
 mulitpath {
 wwid  3600805f3001598700000000000984001c
 alias  oramp4
 }
}
```

This configuration will create 4 LUNs named **/dev/mpath/oramp1**, **/dev/mpath/oramp2**, **/dev/mpath/oramp3** and **/dev/mpath/oramp4**. Once entered, the mapping of the devices' WWID to their new names are now persistent after rebooting.

10.4. Add a virtualized CD-ROM or DVD device to a guest

To attach an ISO file to a guest while the guest is online use **virsh** with the *attach-disk* parameter.

```
# virsh attach-disk [domain-id] [source] [target] --driver file --type cdrom --mode readonly
```

The *source* and *target* parameters are paths for the files and devices, on the host and guest respectively. The *source* parameter can be a path to an ISO file or the device from the **/dev** directory.

Network Configuration

This page provides an introduction to the common networking configurations used by libvirt based applications. For additional information consult the libvirt network architecture documentation.

The two common setups are "virtual network" or "shared physical device". The former is identical across all distributions and available out-of-the-box. The latter needs distribution specific manual configuration.

Network services on virtualized guests are not accessible by default from external hosts. You must enable either Network address translation (NAT) ir a network Bridge 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 out of the box. This is the so called '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, since it uses NAT and IP forwarding to connect to outside world. 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 XML can be used in 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 is useful in certain situations.

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

11.2. 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.

Creating network initscripts

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

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

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

Open the network script for the device you are adding to the bridge. In this example, **ifcfg-eth0** defines 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
```

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.

```
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'.

After configuring, restart networking or reboot.

```
# service network restart
```

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
```

Restart the **libvirt** daemon.

```
# service libvirtd reload
```

You should now have a "shared physical device", which guests can be attached and have full LAN access. Verify your new bridge:

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

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.

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:

- Any 2.6.27 or newer kernel.
- Newer Ubuntu, CentOS, Red Hat Enterprise Linux.

Those versions of Linux detect and install the drivers so additional installation steps are not required.

Note

PCI devices are limited by the virtualized system architecture. Out of the 32 available PCI devices for a guest 2 are not removable. This means there are up to 30 PCI slots available for additional devices per guest. Each PCI device can have up to 8 functions; some PCI devices have multiple functions and only use one slot. Para-virtualized network, para-virtualized disk devices, or other PCI devices using VT-d all use slots or functions. The exact number of devices available is difficult to calculate due to the number of available devices. Each guest can use up to 32 PCI devices with each device having up to 8 functions.

The following Microsoft Windows versions have should work KVM para-virtualized drivers:

- 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)

12.1. 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 should work Windows guests.

   Download the *virtio-win* package with the **yum** command.

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

   The drivers are also from Microsoft (*windowsservercatalog.com*[1]).

   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.

Installing drivers with a virtualized CD-ROM

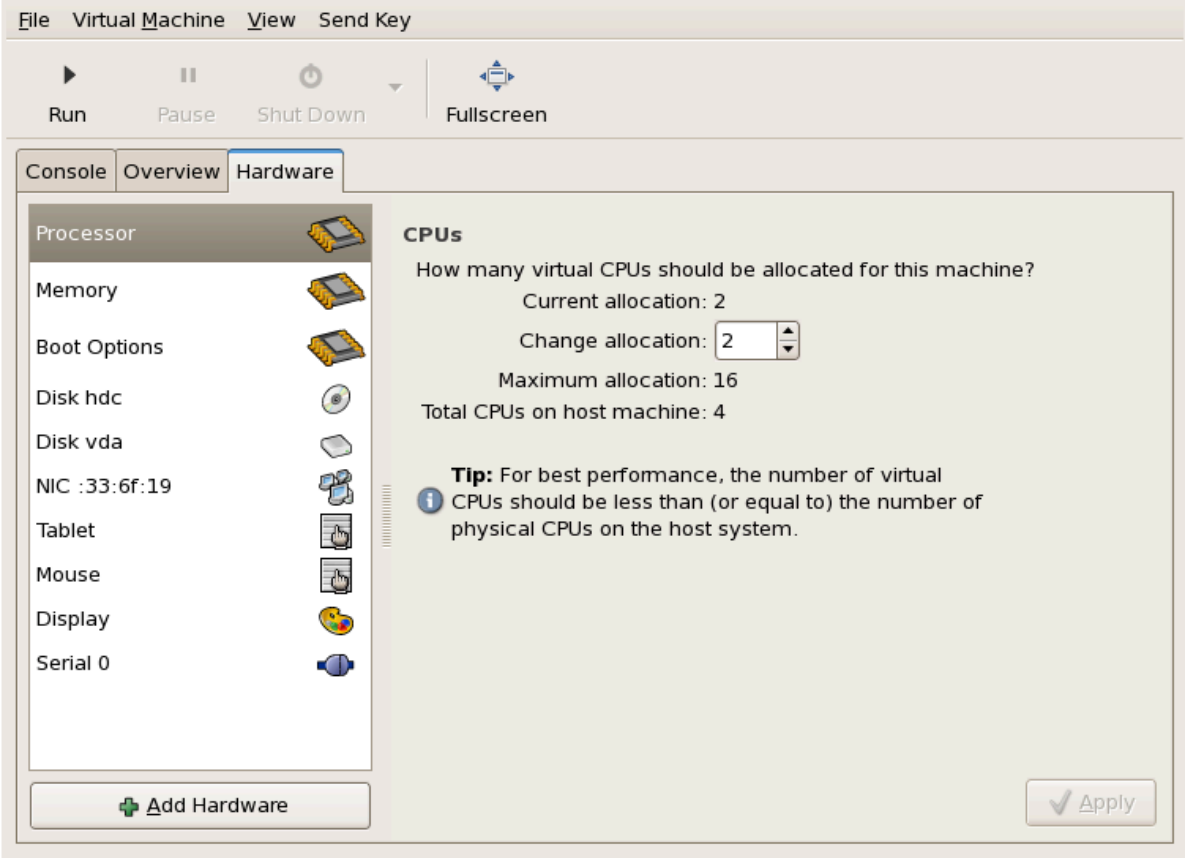
This procedure covers installing the para-virtualized drivers with a virtualized CD-ROM after Windows is installed.

Follow *Procedure 12.1, "Using **virt-manager** to mount a CD-ROM image for a Windows guest"* to add a CD-ROM image with **virt-manager** and then install the drivers.

Procedure 12.1. Using **virt-manager** to mount a CD-ROM image for a Windows guest

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

2. **Open the hardware tab**
   Click the **Add Hardware** button in the **Hardware** tab.



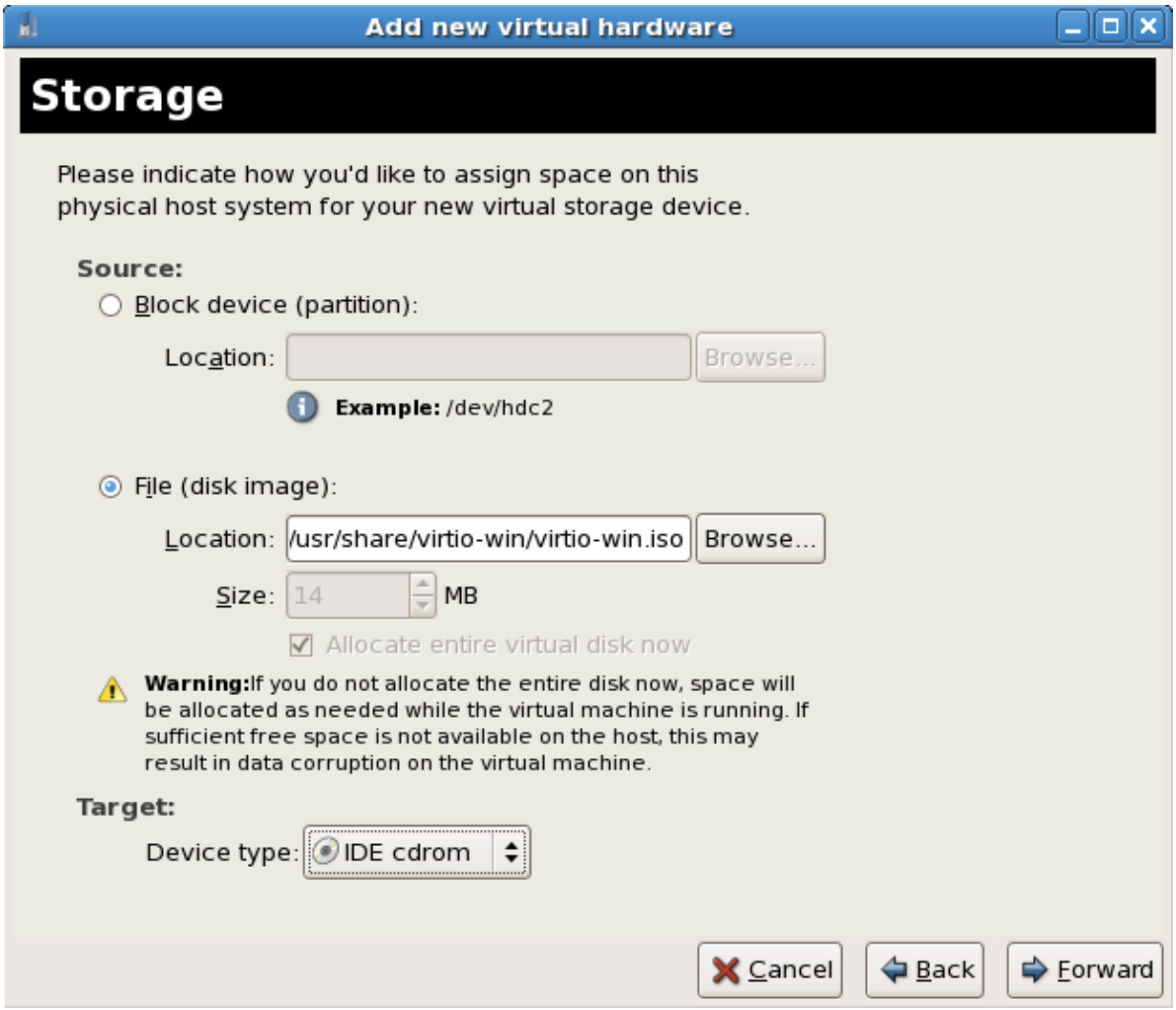
3. **Select the device type**
   This opens a wizard for adding the new device. Select **Storage** from the dropdown menu.

Click the **Forward** button to proceed.

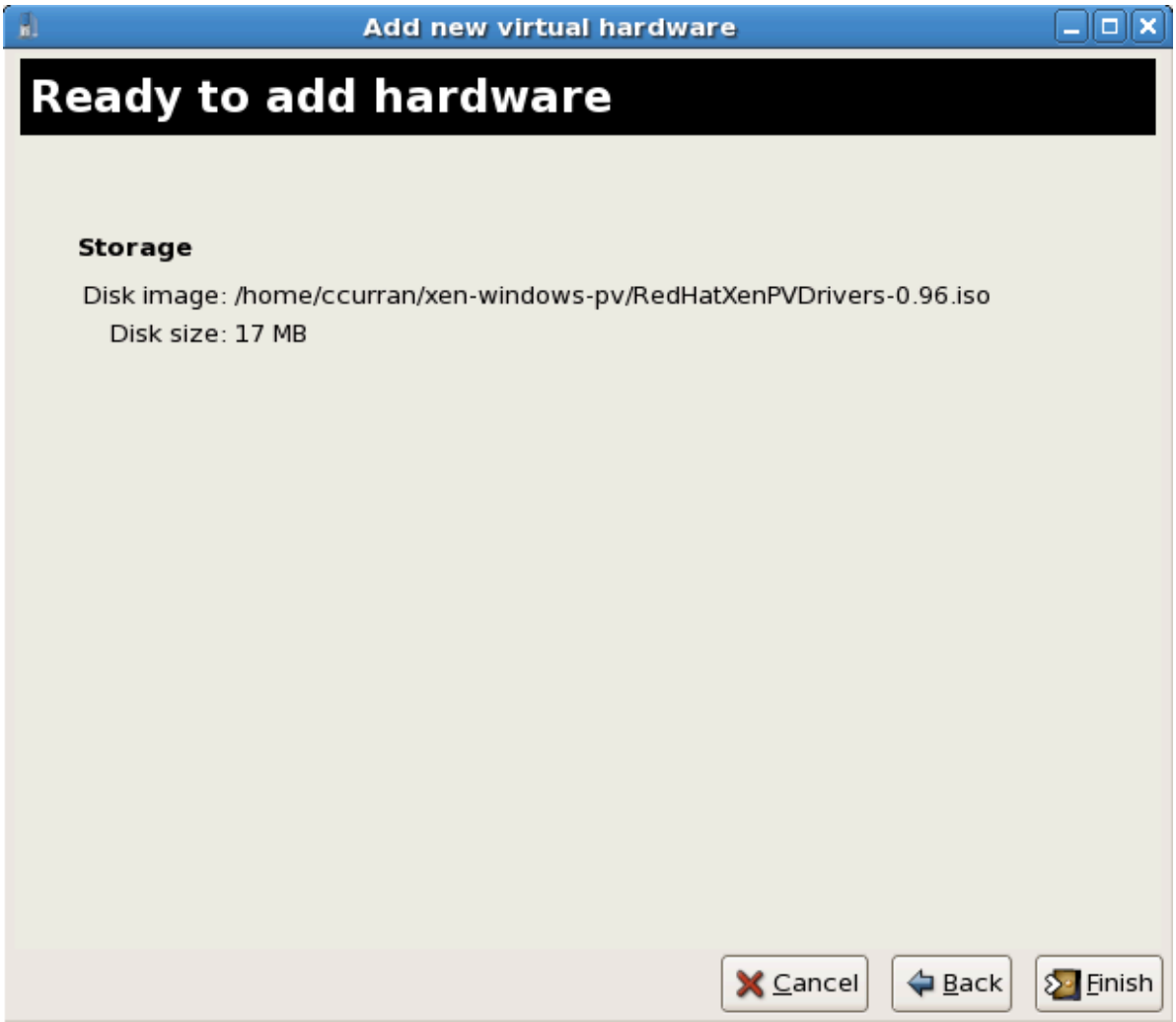
4. **Select the ISO file**
   Choose the **File (disk image)** option and set the file location of the para-virtualized drivers .iso image file. The location file is named **`/usr/share/virtio-win/virtio-win.iso`**.

   If the drivers are stored on a physical CD-ROM, use the **Normal Disk Partition** option.

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



5. **Disc assigned**
   The disk has been assigned and is available for the guest once the guest is started. Click **Finish** to close the wizard or back if you made a mistake.

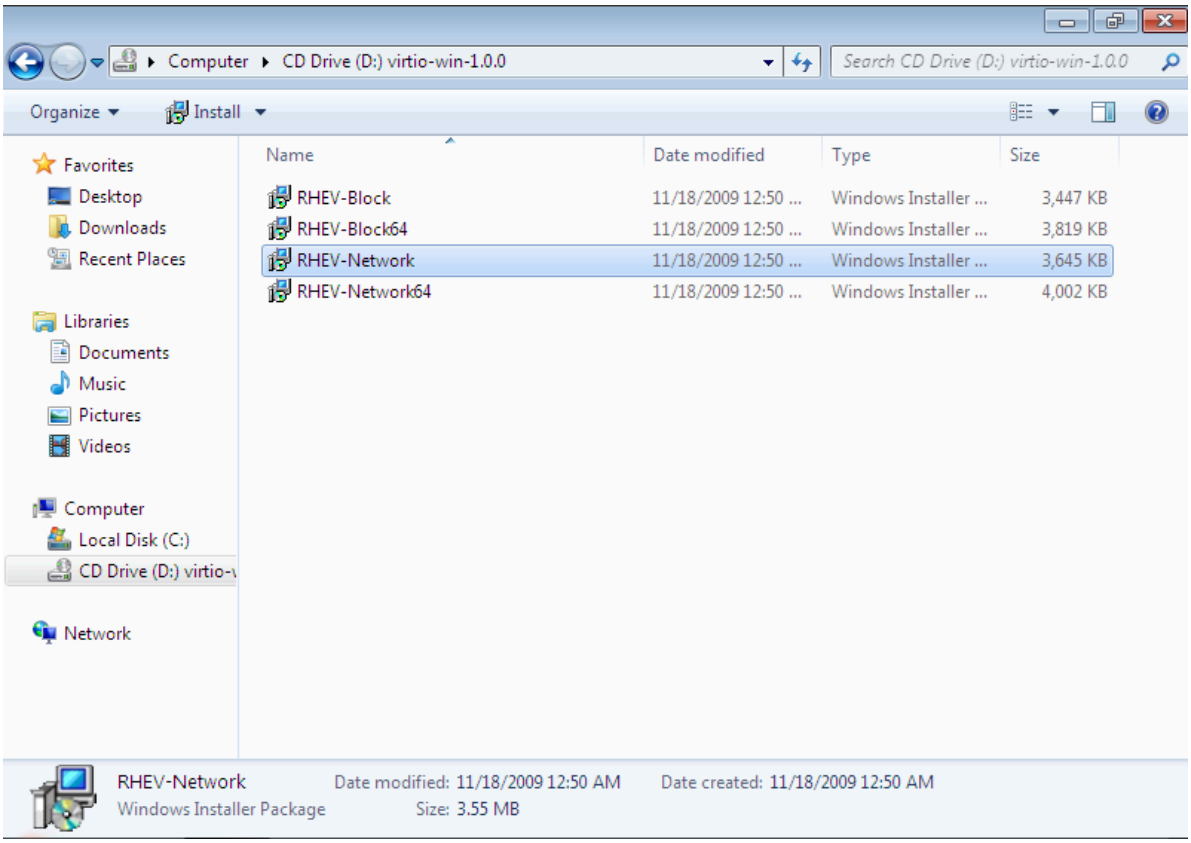


6. **Reboot**
   Reboot or start the guest to add the new device. Virtualized IDE devices require a restart before they can be recognized by guests.

Once the CD-ROM with the drivers is attached and the guest has started, proceed with *Procedure 12.2, "Windows installation"*.

Procedure 12.2. Windows installation

1. **Open My Computer**
   On the Windows guest, open **My Computer** and select the CD-ROM drive.

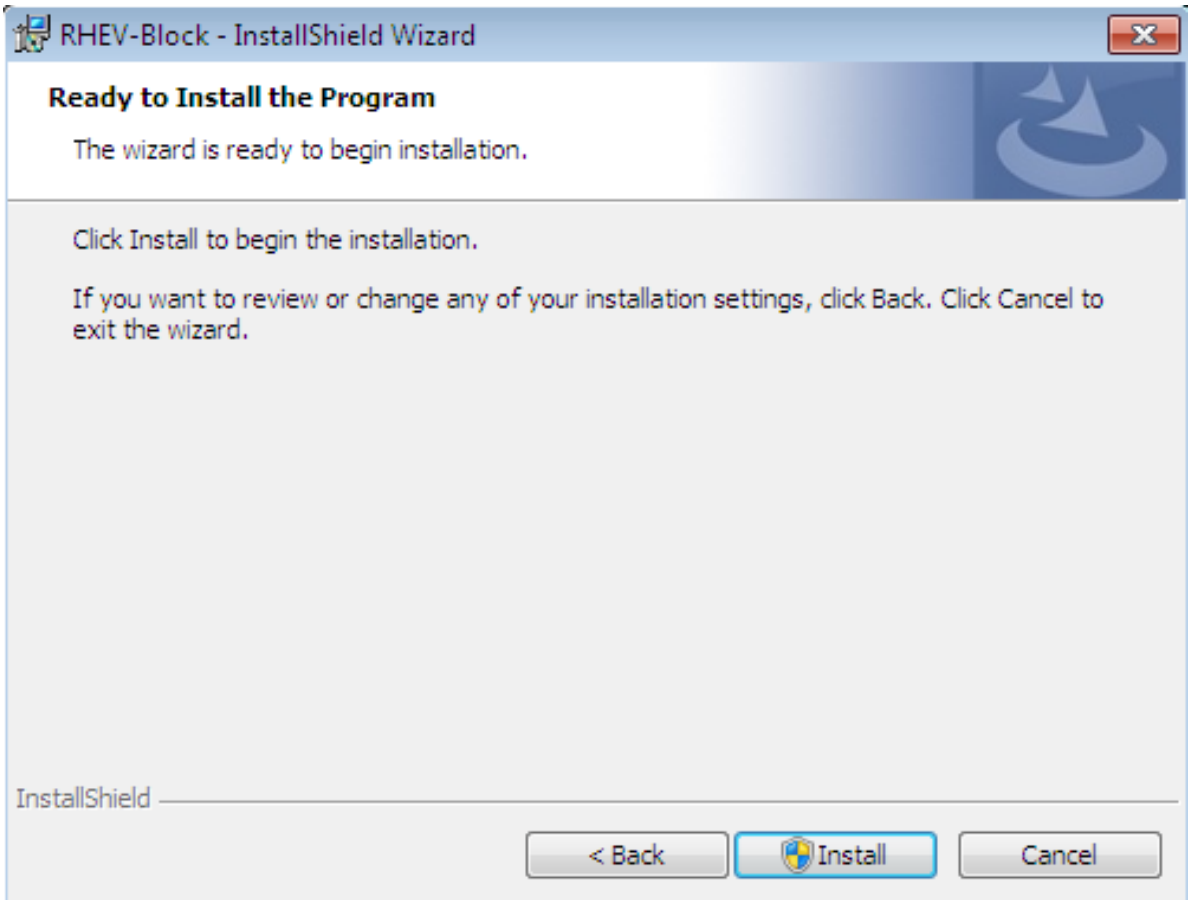


2. **Select the correct installation files**
   There are four files available on the disc. Select the drivers you require for your guest's architecture:

   - the para-virtualized block device driver (**`RHEV-Block.msi`** for 32-bit guests or **`RHEV-Block64.msi`** for 64-bit guests),
   - the para-virtualized network device driver (**`RHEV-Network.msi`** for 32-bit guests or **`RHEV-Block64.msi`** for 64-bit guests),
   - or both the block and network device drivers.

   Double click the installation files to install the drivers.

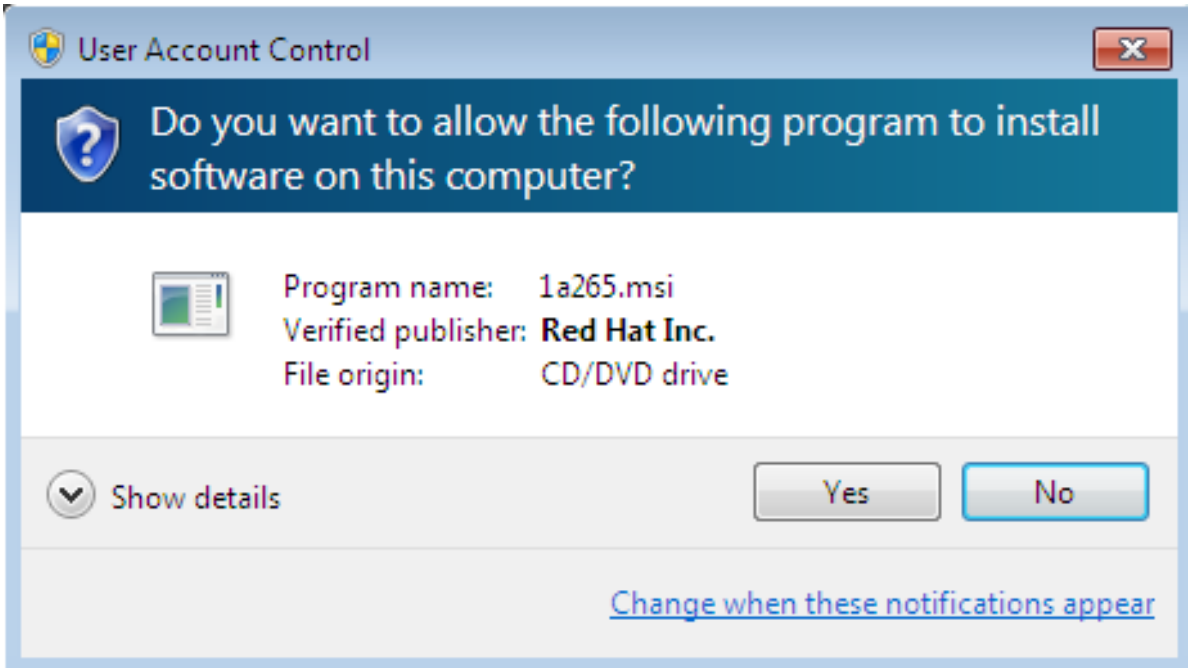
3. **Install the block device driver**
   a. **Start the block device driver installation**
      Double click **`RHEV-Block.msi`** or **`RHEV-Block64.msi`**.



Press **Next** to continue.

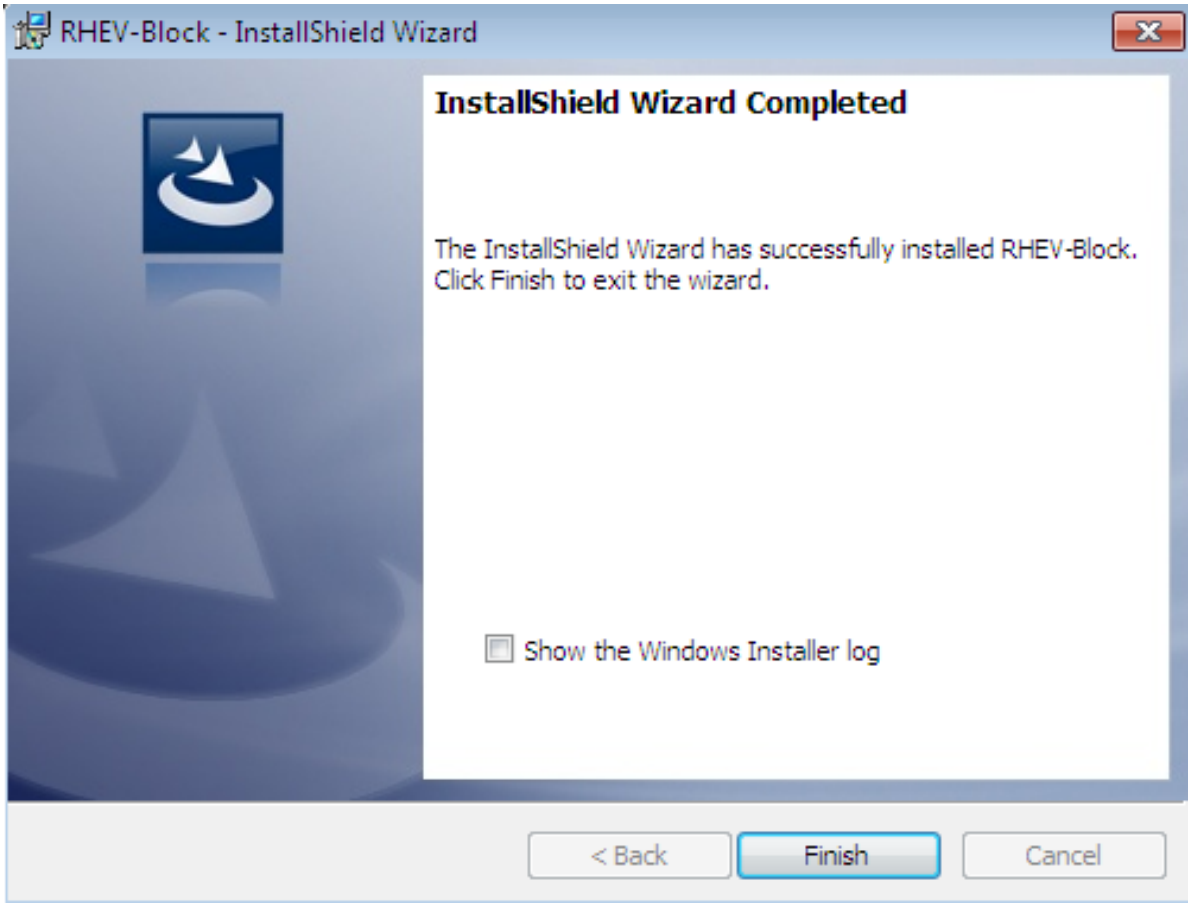
b. **Confirm the exception**

Windows may prompt for a security exception.



Press **Yes** if it is correct.

c. **Finish**

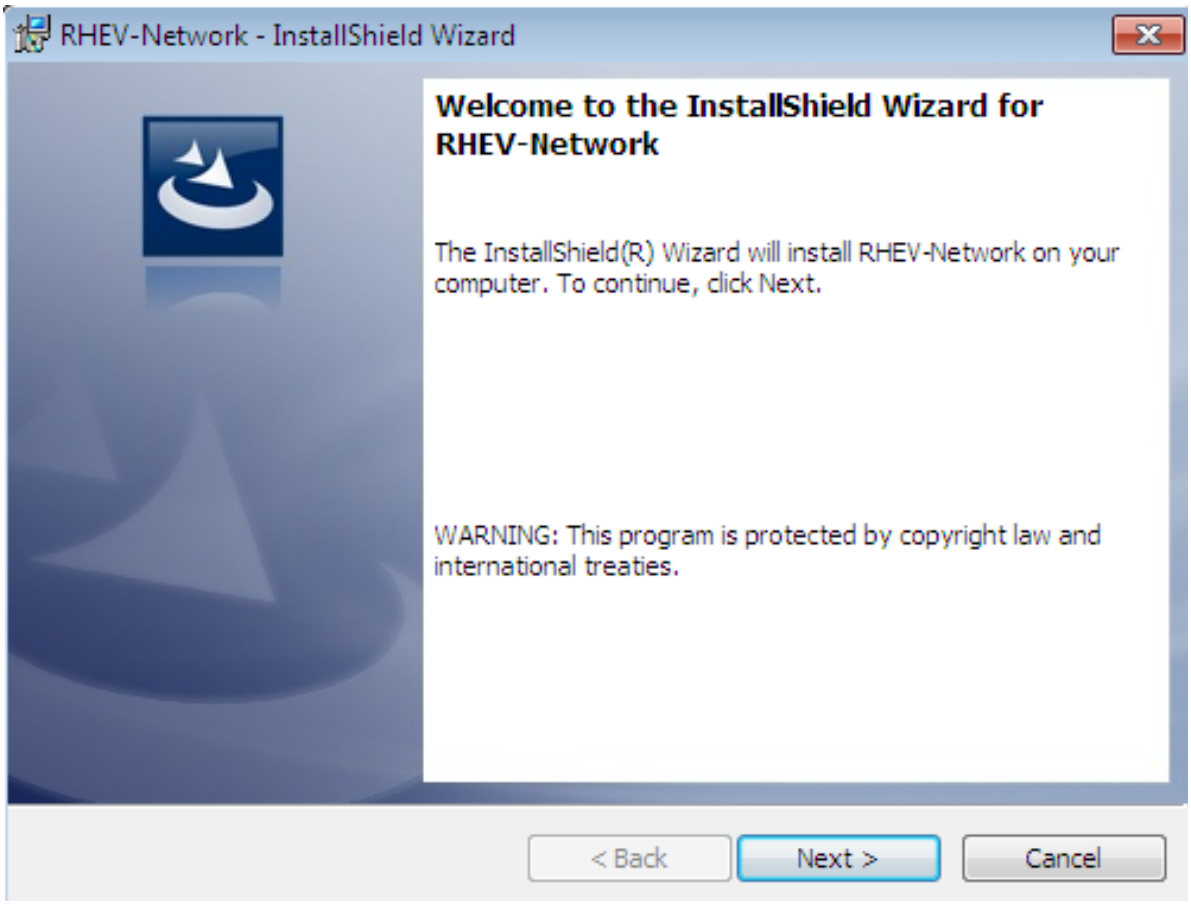


Press **Finish** to complete the installation.

4. **Install the network device driver**

   a. **Start the network device driver installation**

   Double click **`RHEV-Network.msi`** or **`RHEV-Network64.msi`**.



Press **Next** to continue.

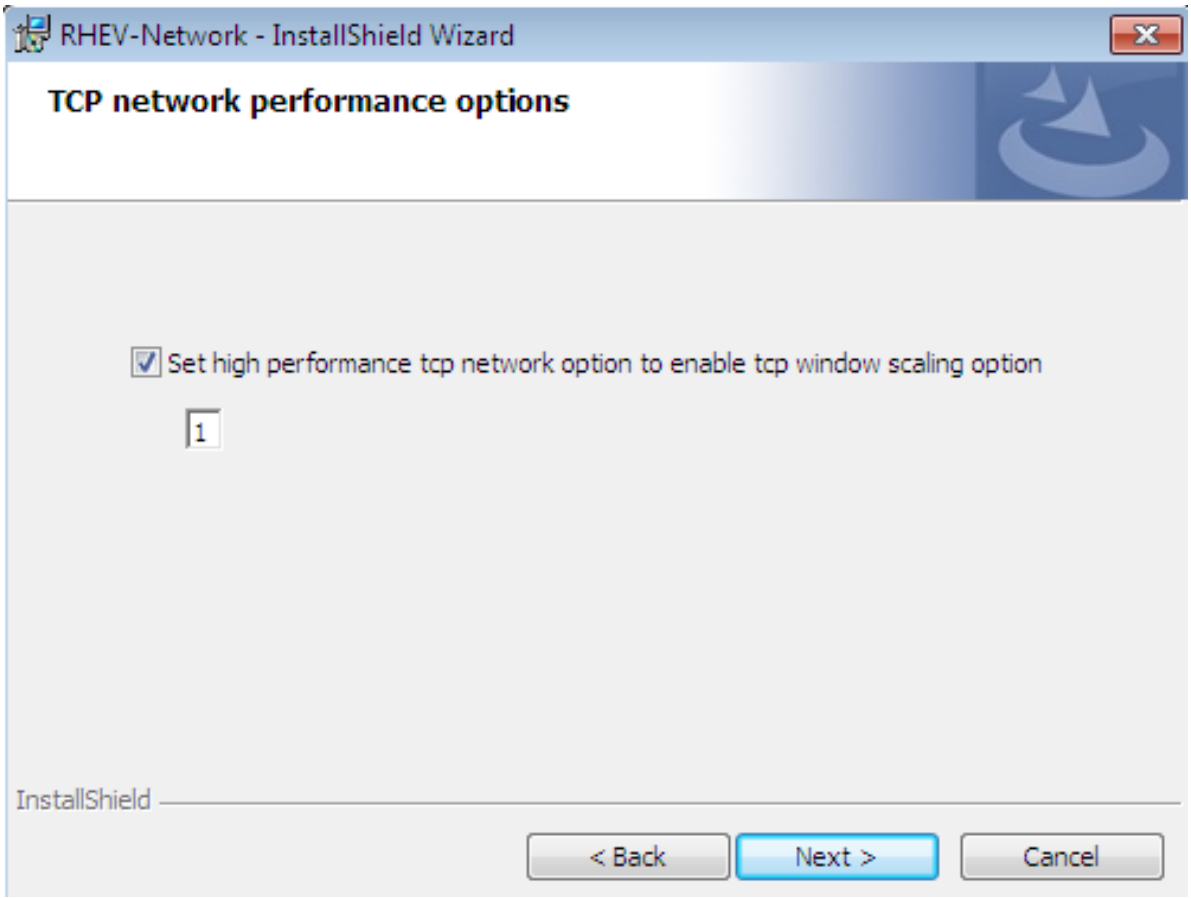
b. **Performance setting**

This screen configures advanced TCP settings for the network driver. TCP timestamps and TCP window scaling can be enabled or disabled. The default is, 1, for window scaling to be enabled.

TCP window scaling is covered by *IETF RFC 1323*[2]. The RFC defines a method of increasing the receive window size to a size greater than the default maximum of 65,535 bytes up to a new maximum of 1 gigabyte (1,073,741,824 bytes). TCP window scaling allows networks to transfer at closer to theoretical network bandwidth limits. Larger receive windows may not be should work by some networking hardware or operating systems.

TCP timestamps are also defined by *IETF RFC 1323*[3]. TCP timestamps are used to better calculate Return Travel Time estimates by embedding timing information is embedded in packets. TCP timestamps help the system to adapt to changing traffic levels and avoid congestion issues on busy networks.

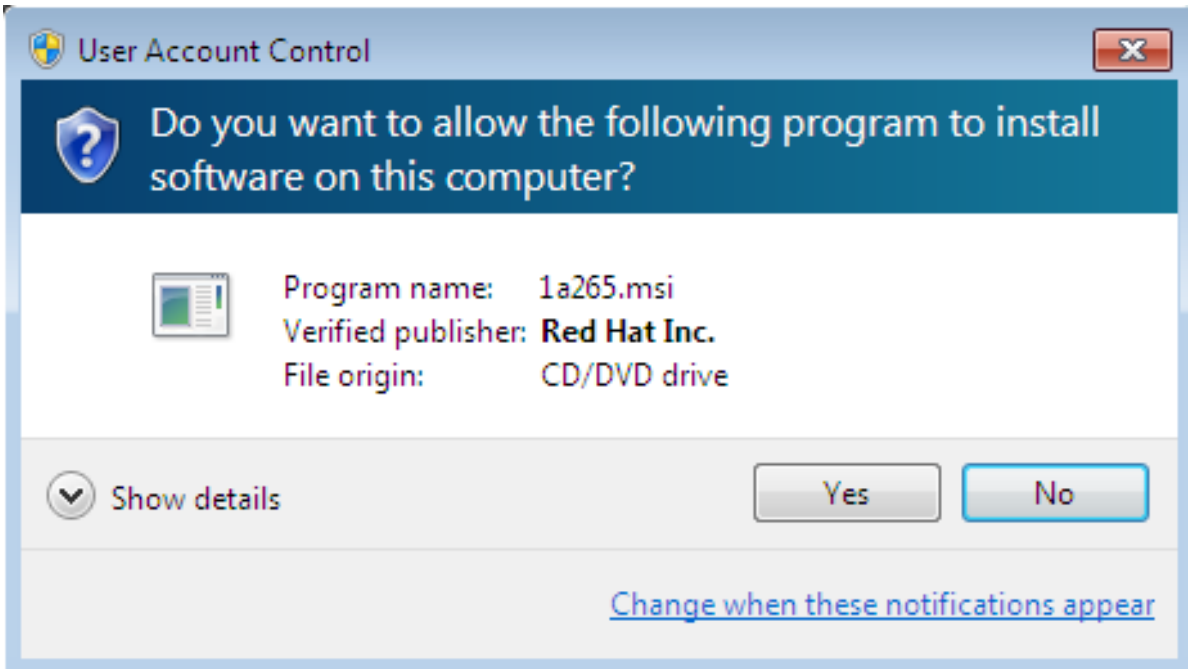
Value	Action
0	Disable TCP timestamps and window scaling.
1	Enable TCP window scaling.
2	Enable TCP timestamps.
3	Enable TCP timestamps and window scaling.



Press **Next** to continue.

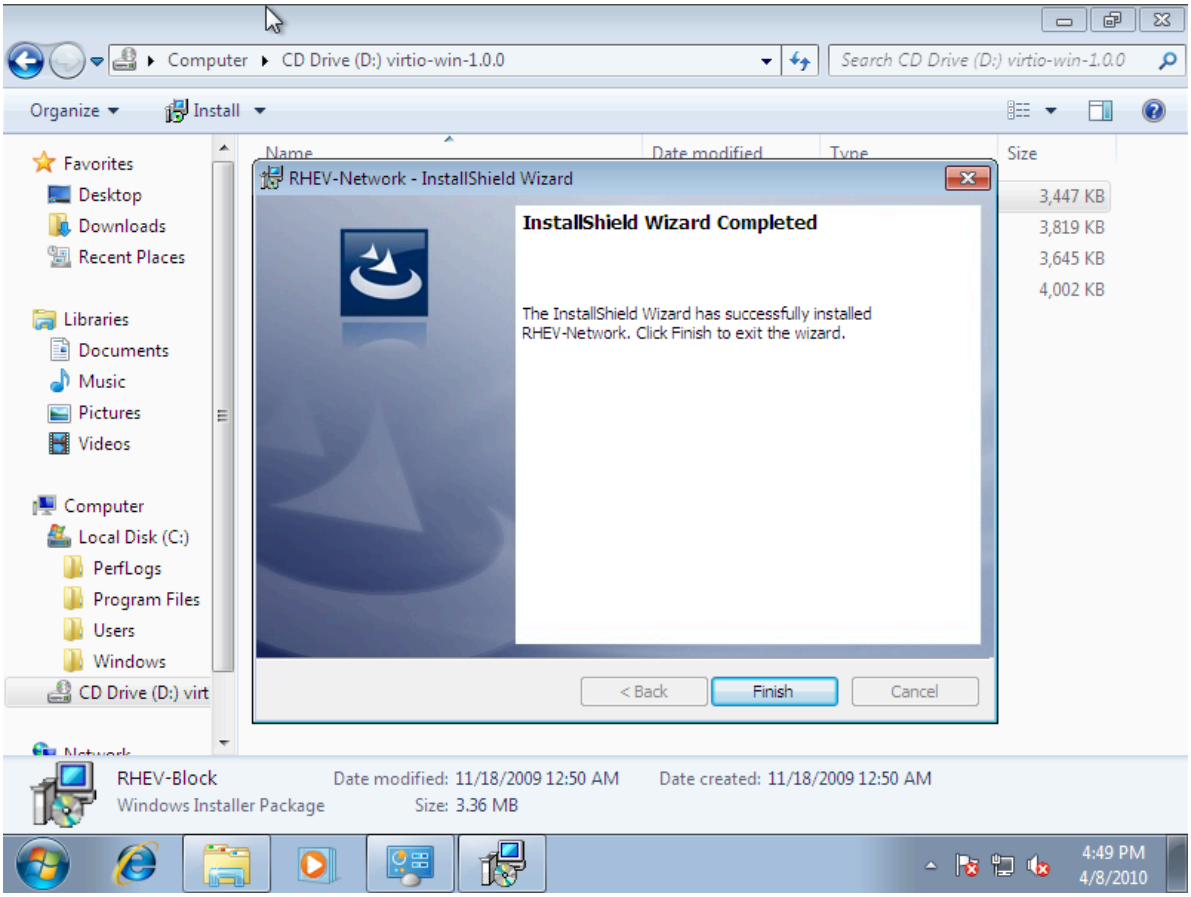
c. **Confirm the exception**

Windows may prompt for a security exception.



Press **Yes** if it is correct.

d. **Finish**



Press **Finish** to complete the installation.

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

Change the device configuration to use the para-virtualized drivers (*Section 12.3, "Using KVM para-virtualized drivers for existing devices"*) or install a new device which uses the para-virtualized drivers (*Section 12.4, "Using KVM para-virtualized drivers for new devices"*).

12.2. Installing drivers with a virtualized floppy disk

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

- Upon installing the Windows VM for the first time using the run-once menu attach **`viostor.vfd`** as a floppy

  a. **Windows Server 2003**
     When windows prompts to press F6 for third party drivers, do so and follow the onscreen instructions.

  b. **Windows Server 2008**
     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.

12.3. Using KVM para-virtualized drivers for existing devices

Modify an existing hard disk device attached to the guest to use the **`virtio`** driver instead of virtualized IDE driver. This example edits libvirt configuration files. Alternatively, **`virt-manager`**, **`virsh attach-disk`** or **`virsh attach-interface`** can add a new device using the para-virtualized drivers *Section 12.4, "Using KVM para-virtualized drivers for new devices"*.

1. 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>
```

2. Change the entry to use the para-virtualized device by modifying the **`bus=`** entry to **`virtio`**.

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

12.4. Using KVM para-virtualized drivers for new devices

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

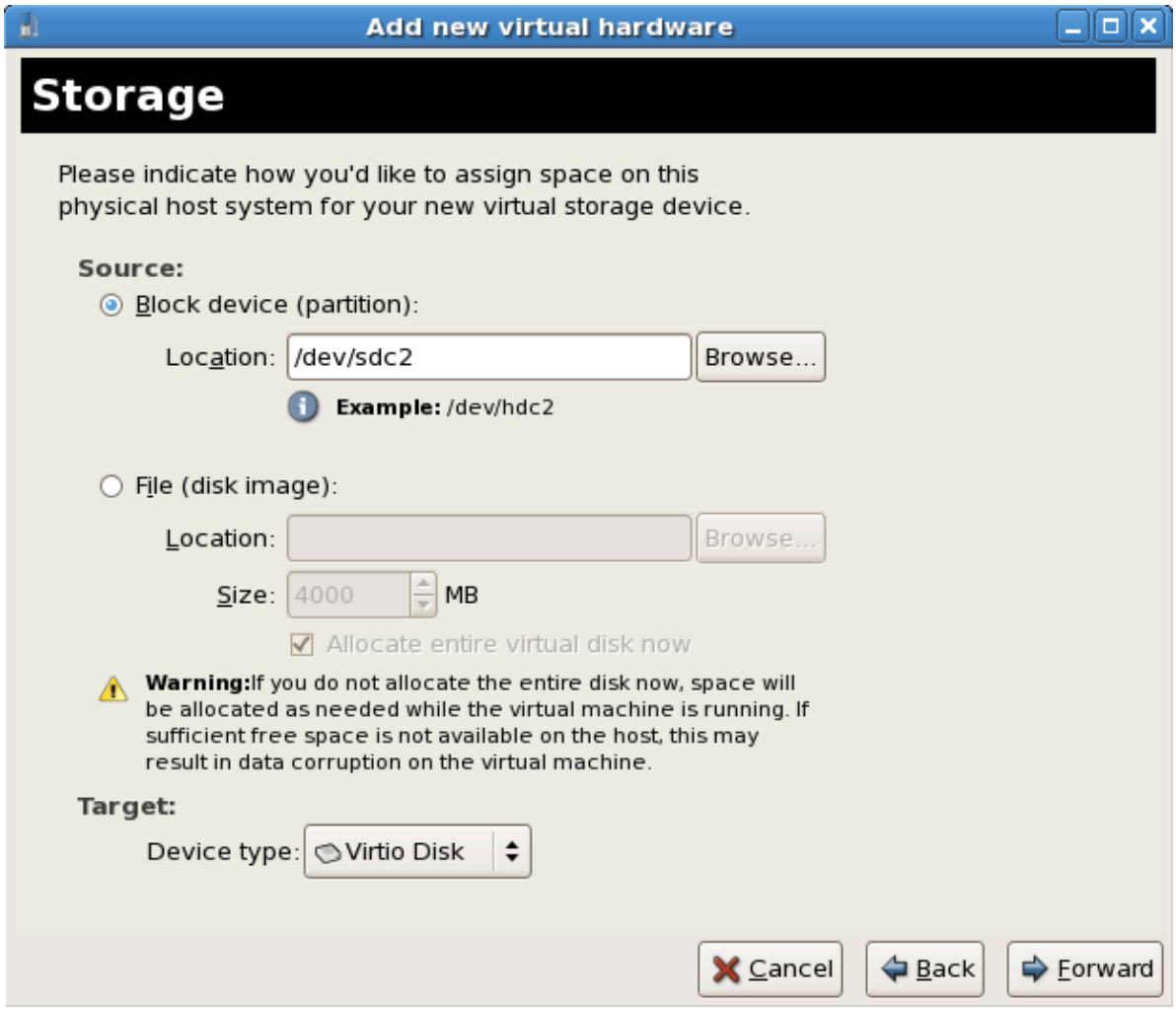
Alternatively, the **`virsh attach-disk`** or **`virsh attach-interface`** commands can be used to attach devices using the para-virtualized drivers.

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.

1. Open the virtualized guest by double clicking on the name of the guest in **`virt-manager`**.
2. Open the **Hardware** tab.
3. Press the **Add Hardware** button.
4. In the Adding Virtual Hardware tab select **Storage** or **Network** for the type of device.
   1. **New disk devices**

      Select the storage device or file-based image. Select **Virtio Disk** as the **Device type** and press **Forward**.



2. **New network devices**

   Select **Virtual network** or **Shared physical device**. Select **virtio** as the **Device type** and press **Forward**.

5. Press **Finish** to save the device.

6. Reboot the guest. The device may not be recognized until the Windows guest restarts.

PCI passthrough

This chapter covers using PCI passthrough with KVM.

The KVM hypervisor supports attaching PCI devices on the host system to virtualized guests. PCI passthrough allows guests to have exclusive access to PCI devices for a range of tasks. PCI passthrough allows PCI devices to appear and behave as if they were physically attached to the guest operating system.

PCI devices are limited by the virtualized system architecture. Out of the 32 available PCI devices for a guest 2 are not removable. This means there are up to 30 PCI slots available for additional devices per guest. Each PCI device can have up to 8 functions; some PCI devices have multiple functions and only use one slot. Para-virtualized network, para-virtualized disk devices, or other PCI devices using VT-d all use slots or functions. The exact number of devices available is difficult to calculate due to the number of available devices. Each guest can use up to 32 PCI devices with each device having up to 8 functions.

The VT-d or AMD IOMMU extensions must be enabled in BIOS.

Procedure 13.1. Preparing an Intel system for PCI passthrough

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

   The Intel VT-d extensions provides hardware support for directly assigning a physical devices to guest. The main benefit of the feature is to improve the performance as native for device access.

   The VT-d extensions are required for PCI passthrough with Fedora. 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. **Activate Intel VT-d in the kernel**

   Activate Intel VT-d in the kernel by appending the *intel_iommu=on* parameter to the kernel line of the kernel line in the **/boot/grub/grub.conf** file.

   The example below is a modified **grub.conf** file with Intel VT-d activated.

   ```
   default=0
   timeout=5
   splashimage=(hd0,0)/grub/splash.xpm.gz
   hiddenmenu
   title Fedora Server (2.6.18-190.el5)
           root (hd0,0)
           kernel /vmlinuz-2.6.18-190.el5 ro root=/dev/VolGroup00/LogVol00 rhgb
    quiet intel_iommu=on
           initrd /initrd-2.6.18-190.el5.img
   ```

3. **Ready to use**

   Reboot the system to enable the changes. Your system is now PCI passthrough capable.

Procedure 13.2. Preparing an AMD system for PCI passthrough

- **Enable AMD IOMMU extensions**

  The AMD IOMMU extensions are required for PCI passthrough with Fedora. The extensions must be enabled in the BIOS. Some system manufacturers disable these extensions by default.

AMD systems only require that the IOMMU is enabled in the BIOS. The system is ready for PCI passthrough once the IOMMU is enabled.

13.1. Adding a PCI device with virsh

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

Important

The VT-d or AMD IOMMU extensions must be enabled in BIOS.

This example uses a USB controller device with the PCI identifier code, **`pci_8086_3a6c`**, and a fully virtualized guest named *`win2k3`*.

1. **Identify the device**

   Identify the PCI device designated for passthrough to the guest. The **`virsh nodedev-list`** command lists all devices attached to the system. The **`--tree`** option is useful for identifying devices attached to the PCI device (for example, disk controllers and USB controllers).

   ```
   # virsh nodedev-list --tree
   ```

   For a list of only PCI devices, run the following command:

   ```
   # virsh nodedev-list | grep pci
   ```

   Each PCI device is identified by a string in the following format (Where *`****`* is a four digit hexadecimal code):

   ```
   pci_8086_****
   ```

   Tip: determining the PCI device

   Comparing **`lspci`** output to **`lspci -n`** (which turns off name resolution) output can assist in deriving which device has which device identifier code.

   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_8086_3a6c
   <device>
     <name>pci_8086_3a6c</name>
     <parent>computer</parent>
     <capability type='pci'>
       <domain>0</domain>
       <bus>0</bus>
       <slot>26</slot>
       <function>7</function>
   ```

```
    <id='0x3a6c'>82801JD/DO (ICH10 Family) USB2 EHCI Controller #2</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
  </capability>
</device>
```

3. Detach the device from the system. Attached devices cannot be used and may cause various errors if connected to a guest without detaching first.

```
# virsh nodedev-dettach pci_8086_3a6c
Device pci_8086_3a6c dettached
```

4. 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'
```

5. Run **`virsh edit`** (or virsh attach device) and added a device entry in the **`<devices>`** section to attach the PCI device to the guest. Only run this command on offline guests. Fedora does not support hotplugging PCI devices at this time.

```
# virsh edit win2k3
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
      <address domain='0x0000' bus='0x00' slot='0x1a' function='0x7'/>
  </source>
</hostdev>
```

6. Once the guest system is configured to use the PCI address, we need to tell the host system to stop using it. The **`ehci`** driver is loaded by default for the USB PCI controller.

```
$ readlink /sys/bus/pci/devices/0000\:00\:1d.7/driver
../../../bus/pci/drivers/ehci_hcd
```

7. Detach the device:

```
$ virsh nodedev-dettach pci_8086_3a6c
```

8. Verify it is now under the control of pci_stub:

```
$ readlink /sys/bus/pci/devices/0000\:00\:1d.7/driver
../../../bus/pci/drivers/pci-stub
```

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

```
$ setsebool -P virt_manage_sysfs 1
```

10. Start the guest system :

```
# virsh start win2k3
```

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

13.2. 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. **Identify the device**
   Identify the PCI device designated for passthrough to the guest. The **`virsh nodedev-list`** command lists all devices attached to the system. The **`--tree`** option is useful for identifying devices attached to the PCI device (for example, disk controllers and USB controllers).

   ```
   # virsh nodedev-list --tree
   ```

   For a list of only PCI devices, run the following command:

   ```
   # virsh nodedev-list | grep pci
   ```

   Each PCI device is identified by a string in the following format (Where `****` is a four digit hexadecimal code):

   ```
   pci_8086_****
   ```

   Tip: determining the PCI device
   Comparing **`lspci`** output to **`lspci -n`** (which turns off name resolution) output can assist in deriving which device has which device identifier code.

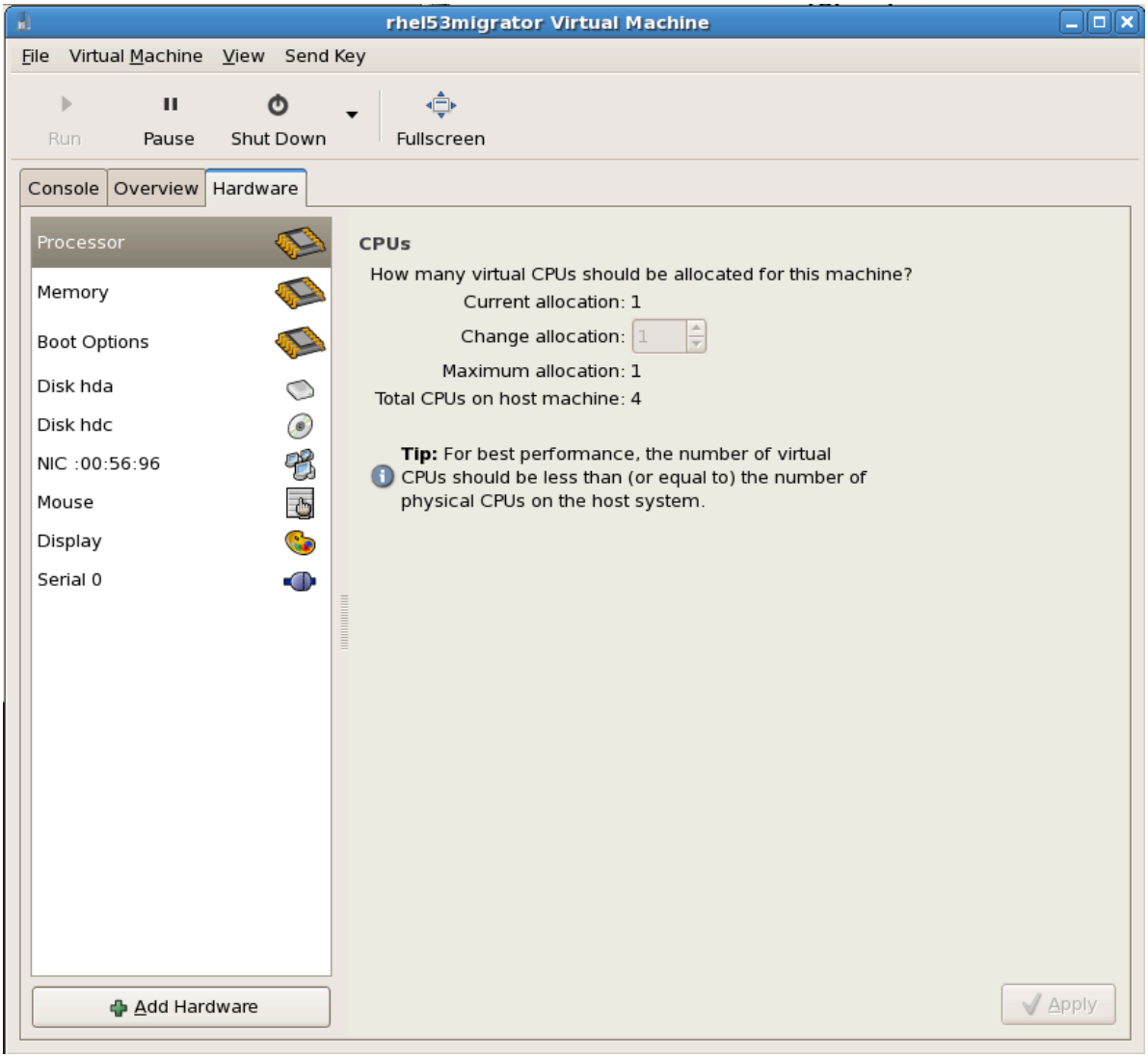
   Record the PCI device number; the number is needed in other steps.

2. **Detach the PCI device**
   Detach the device from the system.

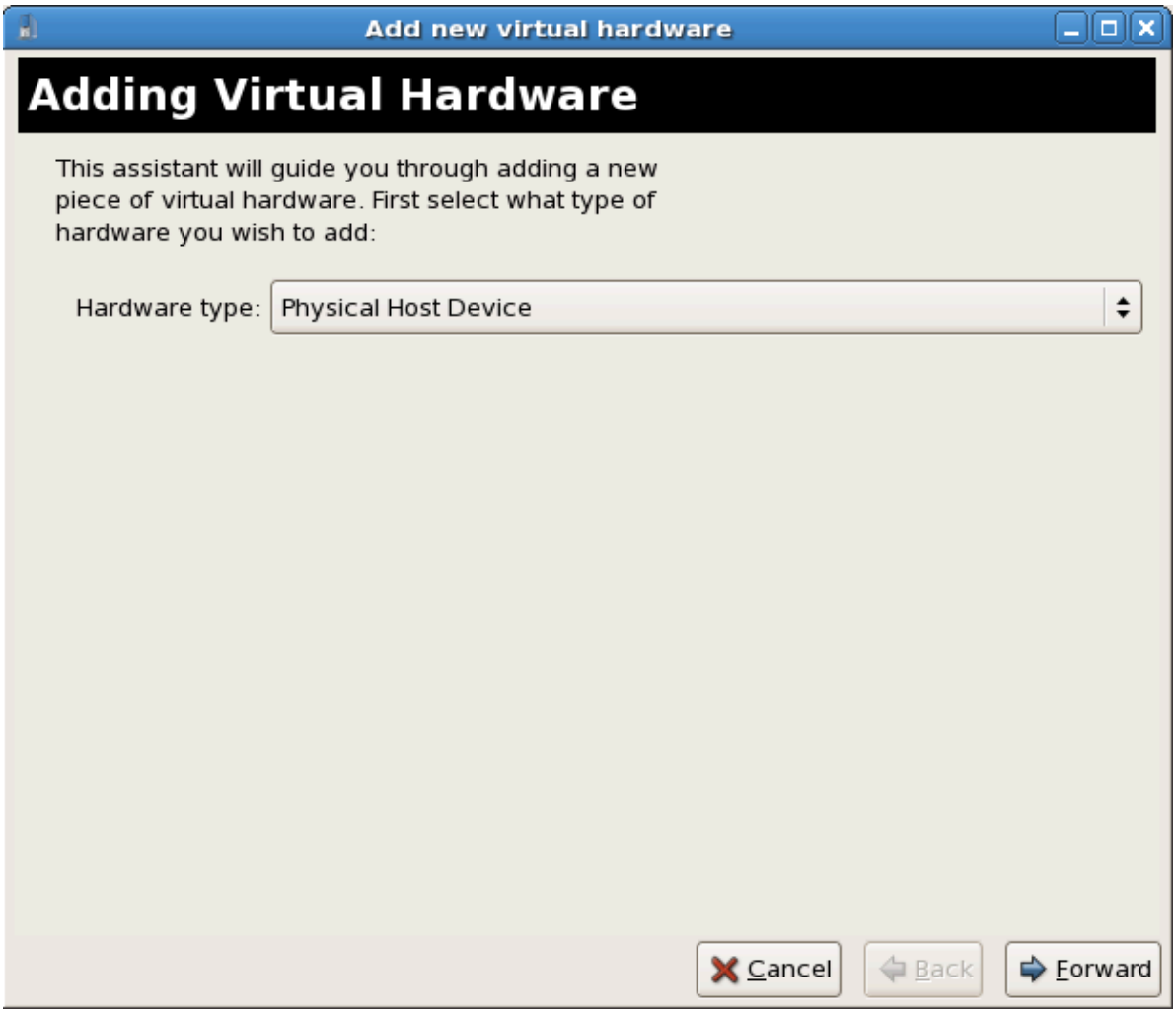
   ```
   # virsh nodedev-dettach pci_8086_3a6c
   Device pci_8086_3a6c dettached
   ```

3. **Power off the guest**
   Power off the guest. Hotplugging PCI devices into guests is presently experimental and may fail or crash.

4. **Open the hardware settings**
   Open the virtual machine and select the **Hardware** tab. Click the **Add Hardware** button to add a new device to the guest.



5. **Add the new device**
   Select **Physical Host Device** from the **Hardware type** list. The **Physical Host Device** represents PCI devices. Click **Forward** to continue.



6. **Select a PCI device**
   Select an unused PCI device. Note taht selecting PCI devices presently in use on the host causes errors. In this example a PCI to USB interface device is used.

7. **Confirm the new device**
   Click the **Finish** button to confirm the device setup and add the device to the guest.

The setup is complete and the guest can now use the PCI device.

13.3. PCI passthrough with virt-install

To use PCI passthrough with the virt-install parameter, use the additional `--host-device` parameter.

1. **Identify the PCI device**
   Identify the PCI device designated for passthrough to the guest. The **`virsh nodedev-list`** command lists all devices attached to the system. The **`--tree`** option is useful for identifying devices attached to the PCI device (for example, disk controllers and USB controllers).

   ```
   # virsh nodedev-list --tree
   ```

   For a list of only PCI devices, run the following command:

   ```
   # virsh nodedev-list | grep pci
   ```

   Each PCI device is identified by a string in the following format (Where `****` is a four digit hexadecimal code):

```
pci_8086_****
```

Tip: determining the PCI device

Comparing **`lspci`** output to **`lspci -n`** (which turns off name resolution) output can assist in deriving which device has which device identifier code.

2. **Add the device**

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

```
# virt-install \
 -n hostdev-test -r 1024 --vcpus 2 \
 --os-variant fedora11 -v --accelerate \
 -l http://download.fedoraproject.org/pub/fedora/linux/development/x86_64/os \
 -x 'console=ttyS0 vnc' --nonetworks --nographics  \
 --disk pool=default,size=8 \
 --debug --host-device=pci_8086_10bd
```

3. **Complete the installation**

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

SR-IOV

14.1. Introduction

The PCI-SIG (PCI Special Interest Group) developed the Single Root I/O Virtualization (SR-IOV) specification. The SR-IOV specification is a standard for a type of PCI passthrough which natively shares a single device to multiple guests. SR-IOV does not require hypervisor involvement in data transfer and management by providing an independent memory space, interrupts, and DMA streams for virtualized guests.

SR-IOV enables a Single Root Function (for example, a single Ethernet port), to appear as multiple, separate, physical devices. PCI 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 new 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 mapped to the configuration space presented to the virtualized guest by the hypervisor.

Each Virtual Function can only be mapped once as Virtual Functions require real hardware. 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.

Disadvantages of SR-IOV

Live migration is presently experimental. As with PCI passthrough, identical device configurations are required for live (and offline) migrations. Without identical device configurations, guest's cannot access the passed-through devices after migrating.

14.2. 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 14.1. 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 13.1, "Preparing an Intel system for PCI passthrough"* 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 should work 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 should work by your device.

```
# modprobe igb max_vfs=1
```

5. **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
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)
03:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
03:10.1 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.

```
# lspci -n | grep 03:00.0
03:00.0 0200: 8086:10c9 (rev 01)
# lspci -n | grep 03:10.0
03:10.0 0200: 8086:10ca (rev 01)
```

   The Physical Function corresponds to **`8086:10c9`** and the Virtual Function to **`8086:10ca`**.

6. **Find the devices with virsh**
   The libvirt service must find the device to add a device to a guest. Use the **`virsh nodedev-list`** command to list available host devices.

```
# virsh nodedev-list | grep 8086
pci_8086_10c9
pci_8086_10c9_0
pci_8086_10ca
pci_8086_10ca_0
[output truncated]
```

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

7. **Get advanced details**
   The **`pci_8086_10c9`** is one of the Physical Functions and **`pci_8086_10ca_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_8086_10ca
# virsh nodedev-dumpxml pci_8086_10ca_0
<device>
  <name>pci_8086_10ca_0</name>
  <parent>pci_8086_3408</parent>
  <driver>
    <name>igbvf</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>3</bus>
    <slot>16</slot>
```

```
    <function>1</function>
    <product id='0x10ca'>82576 Virtual Function</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
  </capability>
</device>
```

This example adds the Virtual Function `pci_8086_10ca_0` to the guest in *Step 9*. Note the **`bus`**, **`slot`** and **`function`** parameters of the Virtual Function, these are required for adding the device.

8. **Detach the Virtual Functions**

   Devices attached to a host cannot be attached to guests. 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.

```
# virsh nodedev-dettach pci_8086_10ca
Device pci_8086_10ca dettached
# virsh nodedev-dettach pci_8086_10ca_0
Device pci_8086_10ca_0 dettached
```

9. **Add the Virtual Function to the guest**

   a. Shut down the guest.

   b. Use the output from the **`virsh nodedev-dumpxml pci_8086_10ca_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='0x03'
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'>
      <source>
        <address bus='0x03' slot='0x10' function='0x01'/>
      </source>
</hostdev>
```

e. Save the configuration.

10. **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.

14.3. Troubleshooting SR-IOV

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

Error starting the guest

Start the configured vm , an error reported 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 which is already assigned to another guest or to the host itself.

USB device passthrough

placeholder

N_Port ID Virtualization (NPIV)

Coming soon.

KVM guest timing management

Virtualization poses various challenges for guest time keeping. Guests which use the Time Stamp Counter (TSC) as a clock source may suffer timing issues as some CPUs do not have a constant Time Stamp Counter. Guests without accurate timekeeping may have issues with some networked applications and processes as the guest will run faster or slower than the actual time and fall out of synchronization.

KVM works around this issue by providing guests with a para-virtualized clock. Alternatively, some guests may use other x86 clock sources for their timing in future versions of those operating systems.

Guests can have several problems caused by inaccurate clocks and counters:

- Clocks can fall out of synchronization with the actual time which invalidates sessions and affects networks.
- Guests with slower clocks may have issues migrating.

These problems exist on other virtualization platforms and timing should always be tested.

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 should minimize the affects of clock skew in all cases.

Determining if your CPU has the constant Time Stamp Counter

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 constant time stamp counters 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:

```
term Red Hat Enterprise Linux Server (2.6.18-159.el5)
        root (hd0,0)
 kernel /vmlinuz-2.6.18-159.el5 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.

Using the para-virtualized clock with 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 for guests on systems without a constant Time Stamp Counter.

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

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

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 clocksource (the PMTIMER usually uses the TSC) add the following line to the Windows boot settings. Windows boot settings are stored in the boot.ini file. Add the following line to the **`boot.ini`** file:

```
/use pmtimer
```

For more information on Windows boot settings and the pmtimer option, refer to *Available switch options for the Windows XP and the Windows Server 2003 Boot.ini files*[2].

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.

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

Part IV. Administration

Administering virtualized systems

These chapters contain information for administering host and virtualized guests using tools included in Fedora.

Server best practices

The following tasks and tips can assist you with securing and ensuring reliability of your Fedora host.

- Run SELinux in enforcing mode. You can do this by executing the command below.

```
# setenforce 1
```

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

Security for virtualization

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

- Run only necessary services on hosts. The fewer processes and services running on the host, the higher the level of security and performance.
- Enable *SELinux* on the hypervisor. Read *Section 19.2, "SELinux and virtualization"* for more information on using SELinux and virtualization.
- Use a firewall to restrict traffic to dom0. You can setup a firewall with default-reject rules that will help secure attacks on dom0. It is also important to limit network facing services.
- Do not allow normal users to access dom0. If you do permit normal users dom0 access, you run the risk of rendering dom0 vulnerable. Remember, dom0 is privileged, and granting unprivileged accounts may compromise the level of security.

19.1. Storage security issues

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

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

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

19.2. SELinux and virtualization

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

SELinux prevents guest images from loading if SELinux is enabled and the images are not in the correct directory. SELinux requires that all guest images are stored in **`/var/lib/libvirt/images`**.

Adding LVM based storage with SELinux in enforcing mode

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

Procedure 19.1. Creating and mounting a logical volume on a virtualized guest with SELinux enabled

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

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

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

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

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

```
# mkdir /virtstorage
```

4. Mount the logical volume.

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

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

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

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

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

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

```
# restorecon -R -v /virtualization
```

Testing new attributes

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

```
# touch /virtualization/newfile
```

Verify the file has been relabeled using the following command:

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

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

19.3. SELinux

This sections contains topics to consider when using SELinux with your virtualization deployment. When you deploy system changes or add devices, you must update your SELinux policy accordingly. To configure an LVM volume for a guest, you must modify the SELinux context for the respective underlying block device and volume group.

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

KVM and SELinux

There are several SELinux booleans which affect KVM. These booleans are listed below for your convenience.

KVM SELinux Booleans

SELinux Boolean	Description
allow_unconfined_qemu_transition	Default: off. This boolean controls whether KVM guests can be transistioned to unconfined users.
qemu_full_network	Default: on. This boolean controls full network access to KVM guests.
qemu_use_cifs	Default: on. This boolean controls KVM's access to CIFS or Samba file systems.
qemu_use_comm	Default: off. This boolean controls whether KVM can access serial or paralell communications ports.
qemu_use_nfs	Default: on. This boolean controls KVM's access to NFS file systems.
qemu_use_usb	Default: on. This boolean allows KVM to access USB devices.

19.4. Virtualization firewall information

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

Guest network services

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

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

KVM live migration

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

Migration is name for the process of moving a virtualized guest from one host to another. Migration is a key feature of virtualization as software is completely separated from hardware. Migration is useful for:

- Load balancing - guests can be moved to hosts with lower usage when a host becomes overloaded.
- Hardware failover - when hardware devices on the host start to fail, guests can be safely relocated so the host can be powered down and repaired.
- Energy saving - guests can be redistributed to other hosts and host systems powered off to save energy and cut costs in low usage periods.
- Geographic migration - guests can be moved to another location for lower latency or in serious circumstances.

Migrations can be performed live or offline. To migrate guests the storage must be shared. Migration works by sending the guests memory to the destination host. The shared storage stores the guest's default file system. The file system image is not sent over the network from the source host to the destination host.

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

The time an offline migration takes depends network bandwidth and latency. A guest with 2GB of memory should take an average of ten or so seconds on a 1 Gbit Ethernet link.

A live migration keeps the guest running on the source host and begins moving the memory without stopping the guest. All modified memory pages are monitored for changes and sent to the destination while the image is sent. The memory is updated with the changed pages. The process continues until the amount of pause time allowed for the guest equals the predicted time for the final few pages to be transfer. KVM estimates the time remaining and attempts to transfer the maximum amount of page files from the source to the destination until KVM predicts the amount of remaining pages can be transferred during a very brief time while the virtualized guest is paused. The registers are loaded on the new host and the guest is then resumed on the destination host. If the guest cannot be merged (which happens when guests are under extreme loads) the guest is paused and then an offline migration is started instead.

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

20.1. Live migration requirements

Migrating guests requires the following:

Migration requirements

- A virtualized guest installed on shared networked storage using one of the following protocols:
  - Fibre Channel

- iSCSI
- NFS
- GFS2

- Two or more Fedora systems of the same version with the same updates.
- Both system must have the appropriate ports open.
- Both systems must have identical network configurations. All bridging and network configurations must be exactly the same on both hosts.
- Shared storage must mount at the same location on source and destination systems. The mounted directory name must be identical.

Configuring network storage

Configure shared storage and install a guest on the shared storage. For shared storage instructions, refer to *Part V, "Virtualization storage topics"*.

Alternatively, use the NFS example in *Section 20.2, "Share storage example: NFS for a simple migration"*.

20.2. Share storage example: NFS for a simple migration

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

For advanced and more robust shared storage instructions, refer to *Part V, "Virtualization storage topics"*

1. **Export your libvirt image directory**
   Add the default image directory to the **`/etc/exports`** file:

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

   Change the hosts parameter as required for your environment.

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

   ```
   # yum install nfs
   ```

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

   c. Start the NFS service:

   ```
   # service nfs start
   ```

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

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

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

20.3. Live KVM migration with virsh

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

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

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

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

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

Example: live migration with virsh

This example migrates from `test1.example.com` to `test2.example.com`. Change the host names for your environment. This example migrates a virtual machine named **RHEL4test**.

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

1. **Verify the guest is running**
   From the source system, `test1.example.com`, verify `RHEL4test` is running:

```
[root@test1 ~]# virsh list
Id Name                 State
----------------------------------
 10 RHEL4                running
```

2. **Migrate the guest**
   Execute the following command to live migrate the guest to the destination, `test2.example.com`. Append **/system** to the end of the destination URL to tell libvirt that you need full access.

```
# virsh migrate --live RHEL4test qemu+ssh://test2.example.com/system
```

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

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

4. **Verify the guest has arrived at the destination host**
   From the destination system, `test2.example.com`, verify `RHEL4test` is running:

```
[root@test2 ~]# virsh list
Id Name                 State
----------------------------------
 10 RHEL4                 running
```

The live migration is now complete.

Other networking methods

libvirt supports a variety of networking methods including TLS/SSL, unix sockets, SSH, and unencrypted TCP. Refer to *Chapter 21, Remote management of virtualized guests* for more information on using other methods.

20.4. Migrating with virt-manager

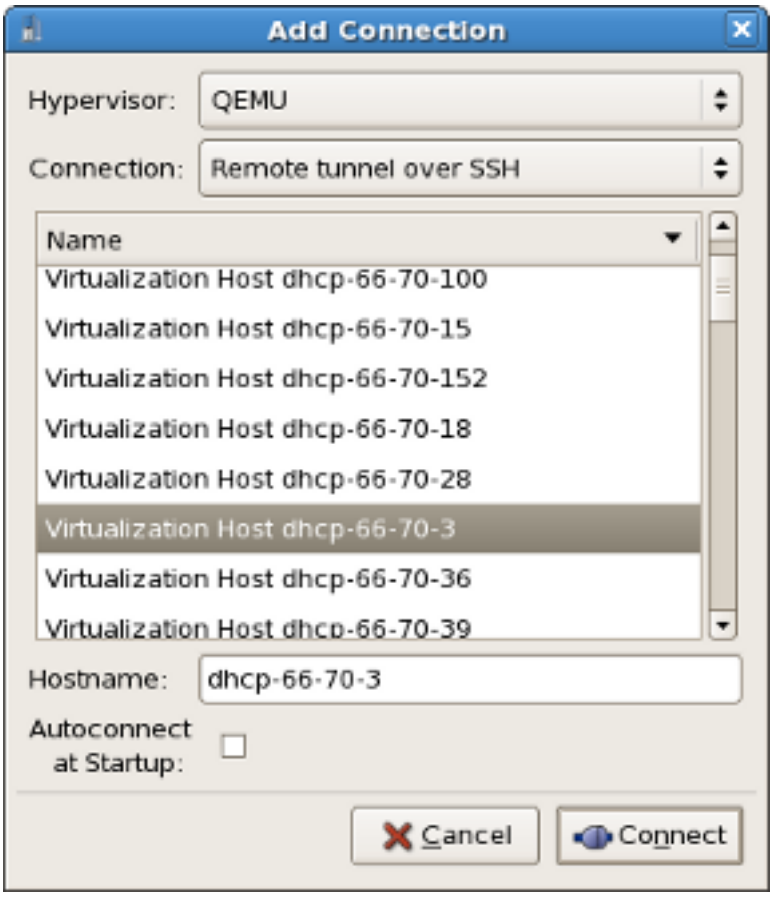
This section covers migrating KVM based guests with **`virt-manager`**.

1. Connect to the source and target hosts. On the **File** menu, click **Add Connection**, the **Add Connection** window appears.

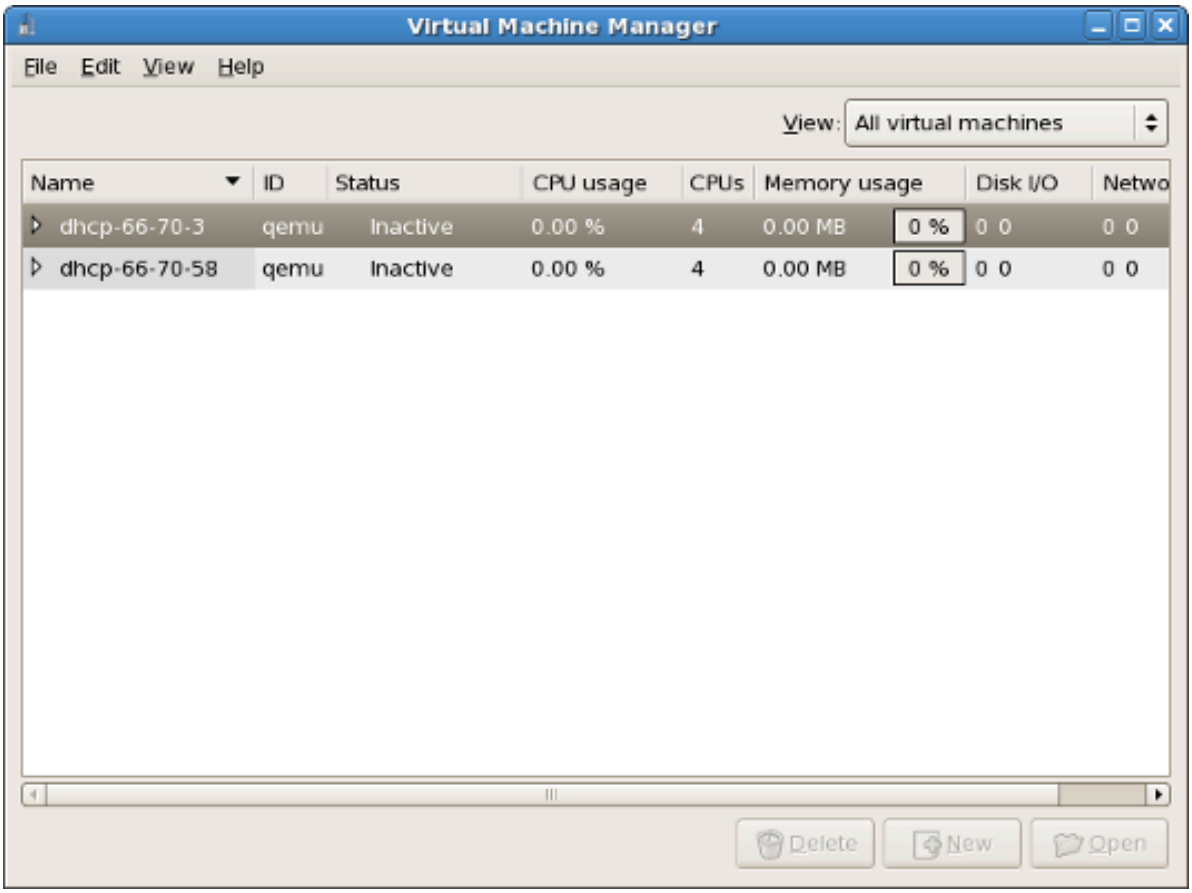
   Enter the following details:

   - **Hypervisor**: Select **QEMU**.
   - **Connection**: Select the connection type.
   - **Hostname**: Enter the hostname.

   Click **Connect**.



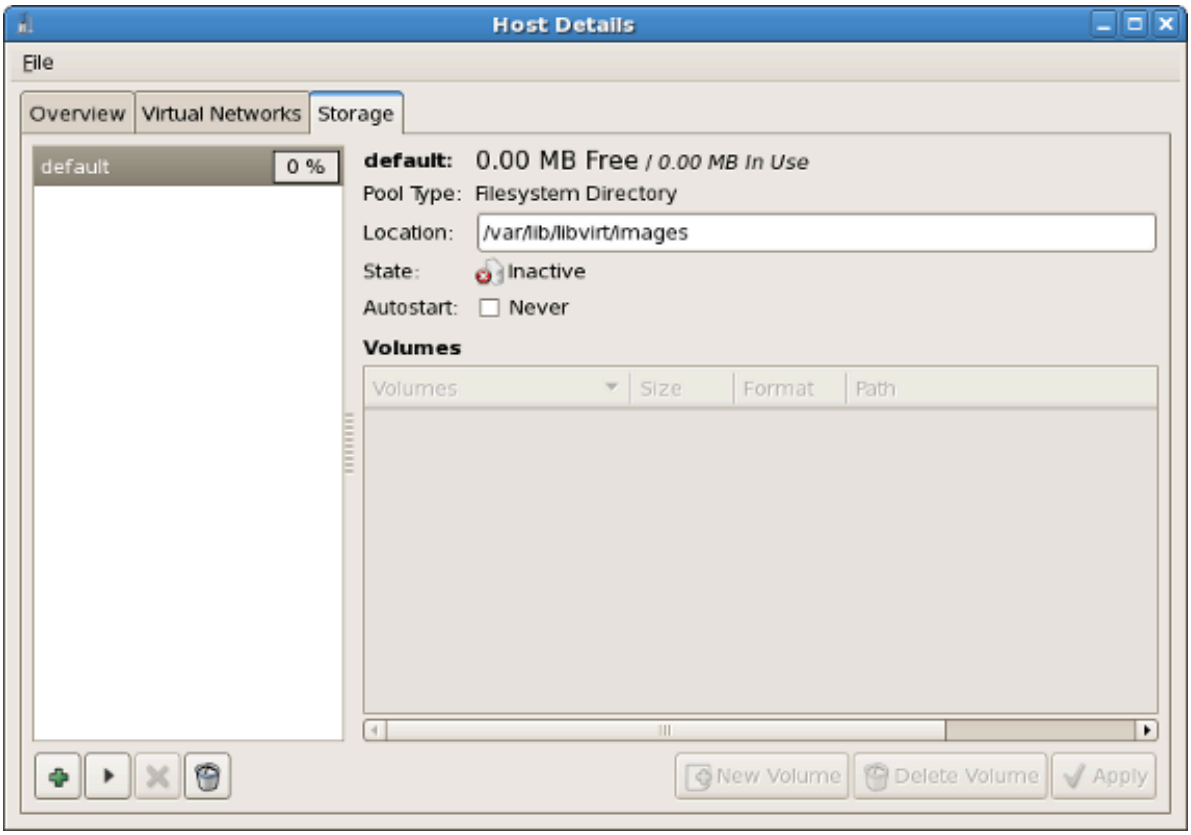
The Virtual Machine Manager displays a list of connected hosts.



2. Add a storage pool with the same NFS to the source and target hosts.

   On the **Edit** menu, click **Host Details**, the Host Details window appears.

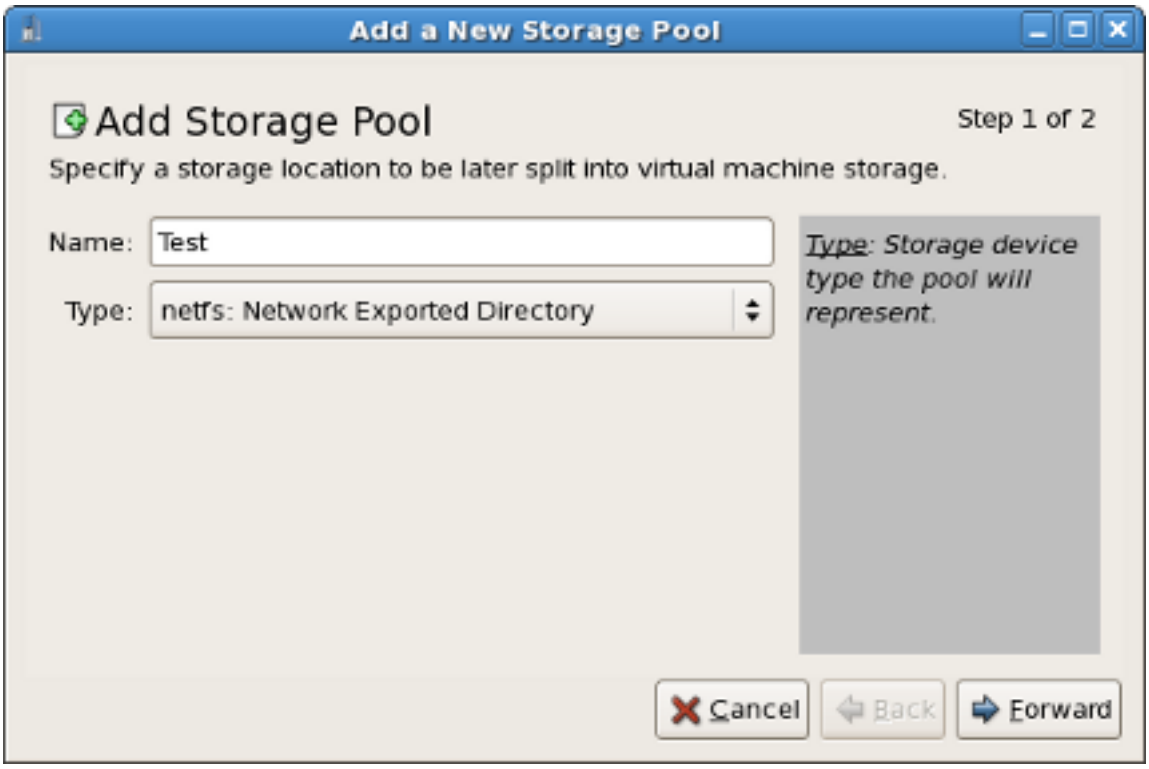
   Click the **Storage** tab.



3. Add a new storage pool. In the lower left corner of the window, click the **+** button. The Add a New Storage Pool window appears.

   Enter the following details:

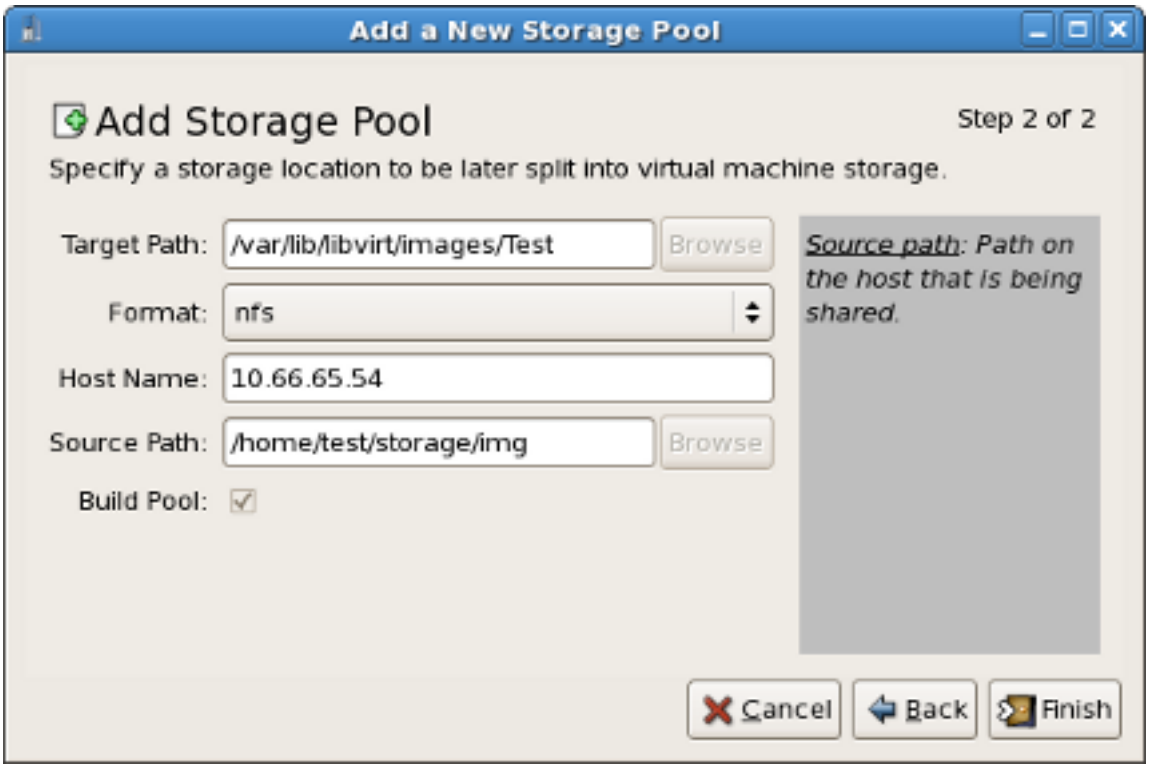
   - **Name**: Enter the name of the storage pool.
   - **Type**: Select **netfs: Network Exported Directory**.



Click **Forward**.

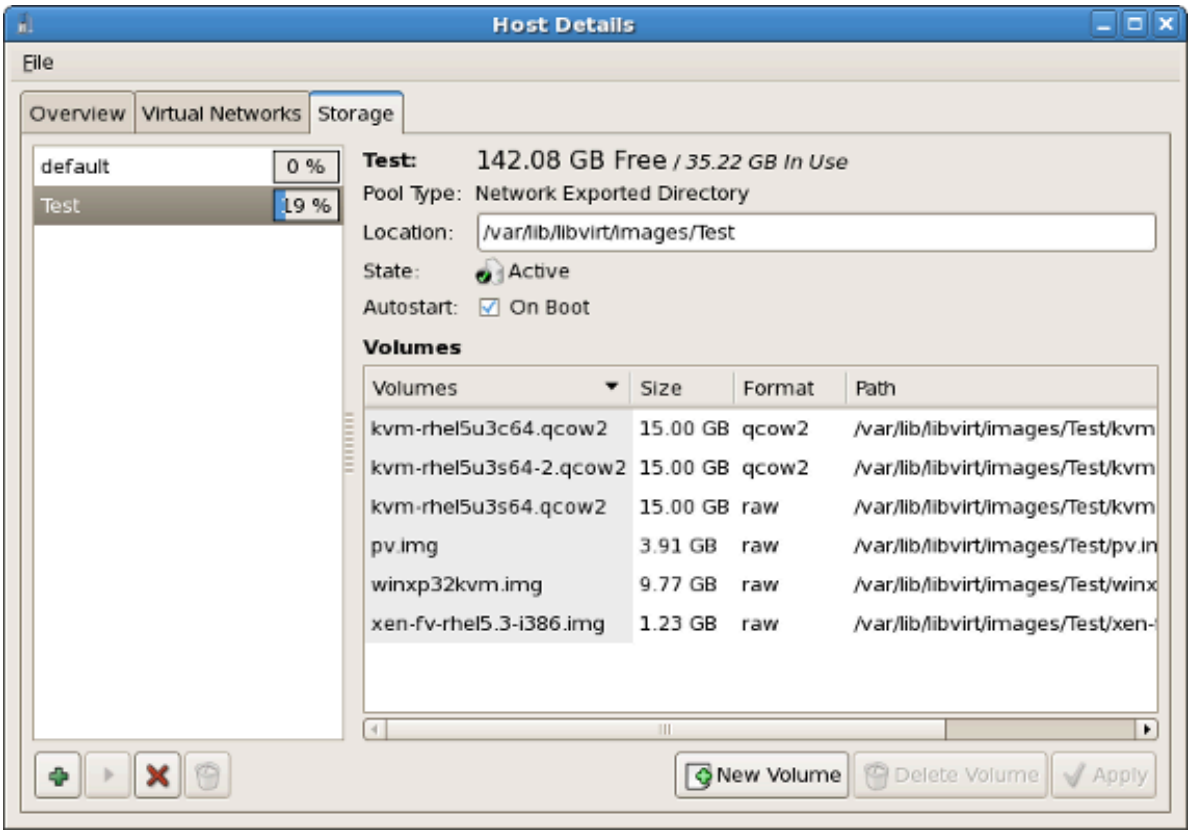
4. Enter the following details:

   - **Format**: Select the storage type. This must be NFS or iSCSI for live migrations.
   - **Host Name**: Enter the IP address or fully-qualified domain name of the storage server.

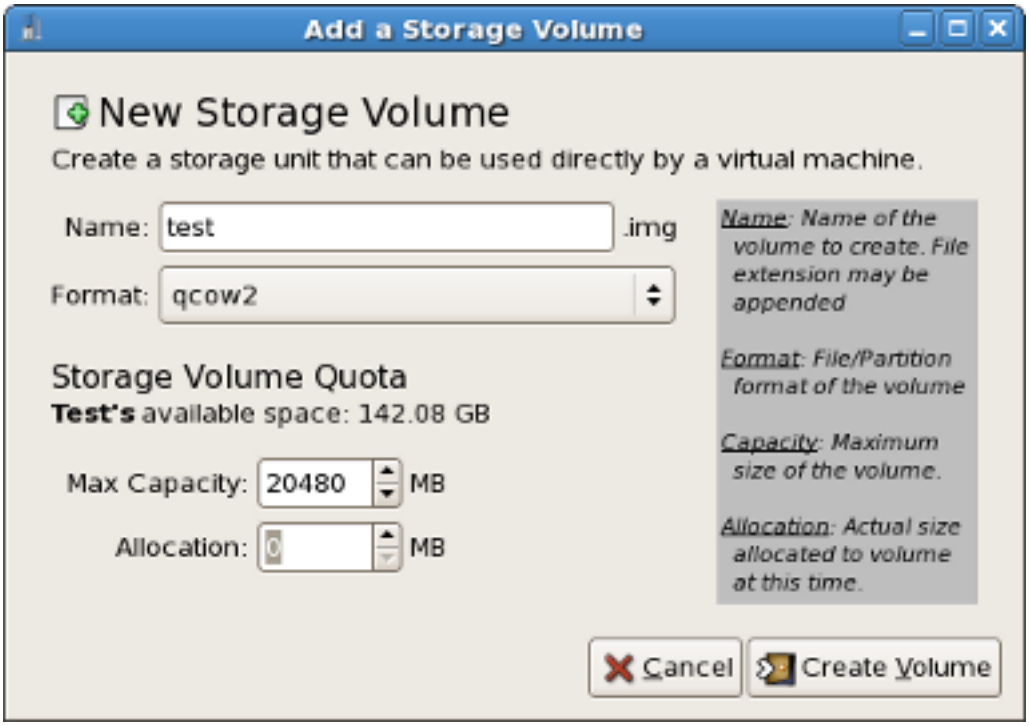


Click **Finish**.

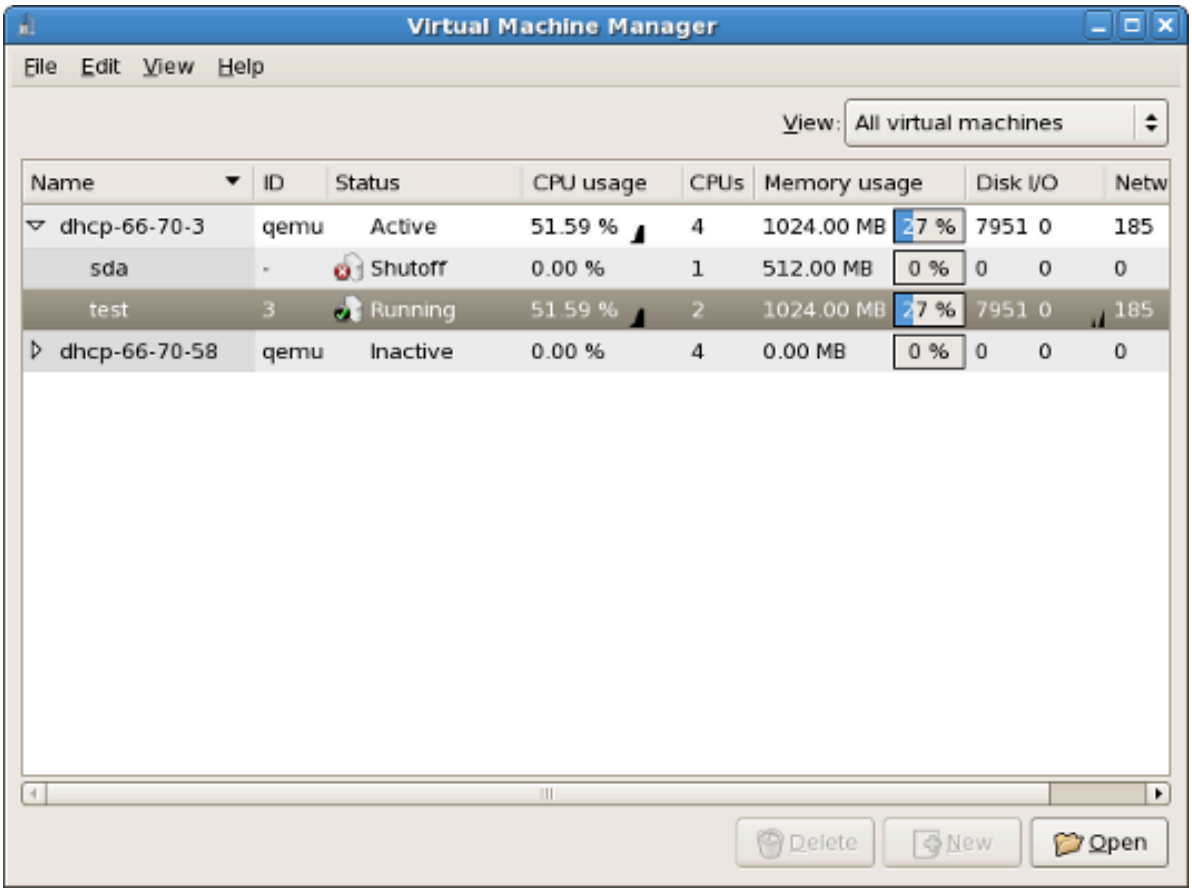
5. Create a new volume in the shared storage pool, click **New Volume**.



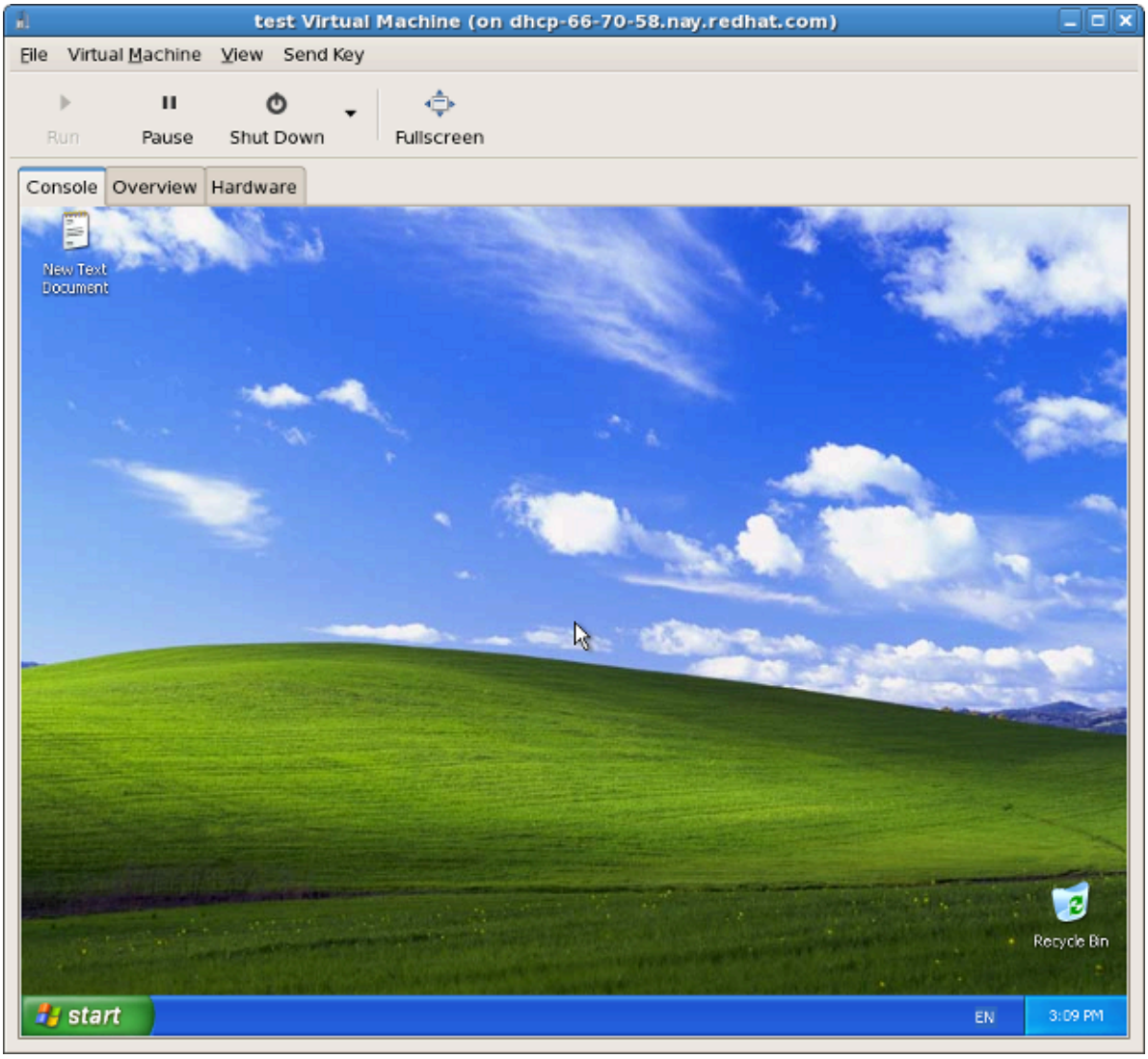
6. Enter the details, then click **Create Volume**.



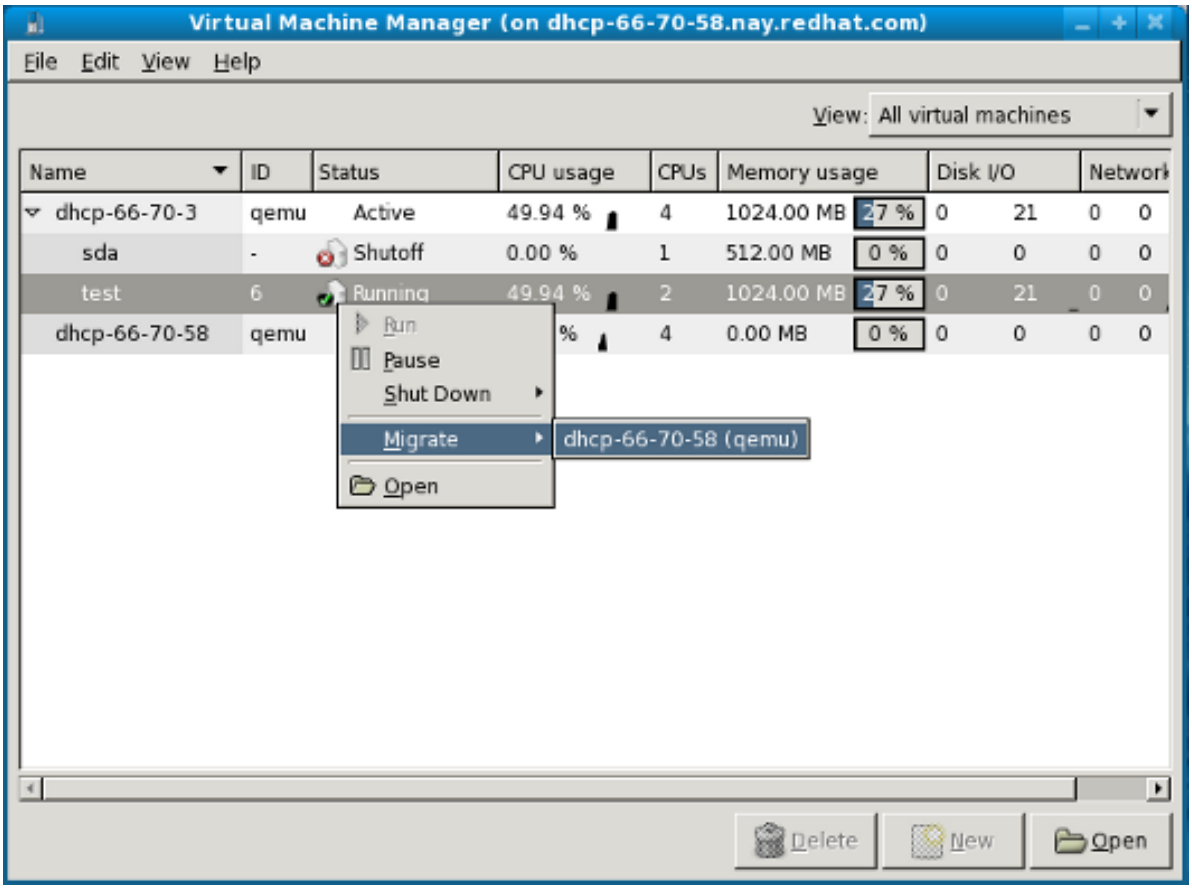
7. Create a virtual machine with the new volume, then run the virtual machine.



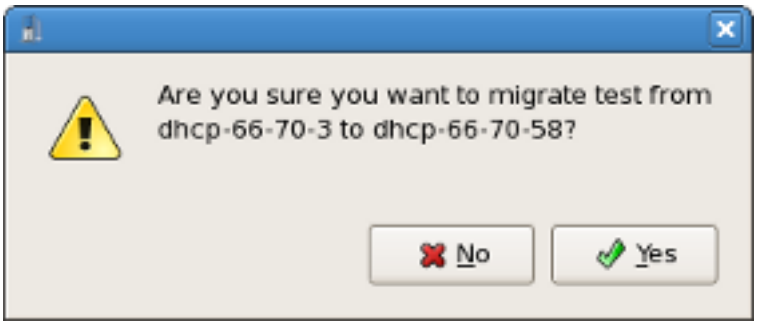
The Virtual Machine window appears.



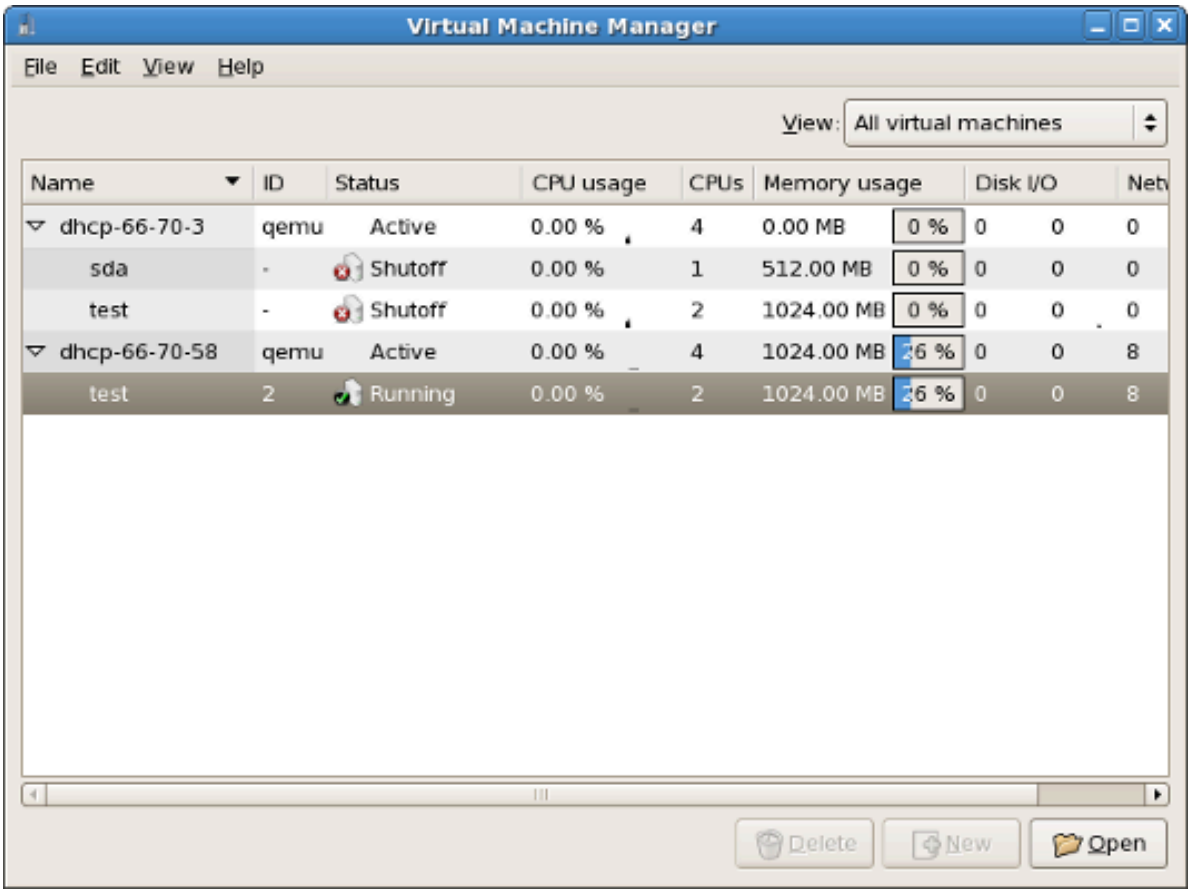
8. In the Virtual Machine Manager window, right-click on the virtual machine, select **Migrate**, then click the migration location.



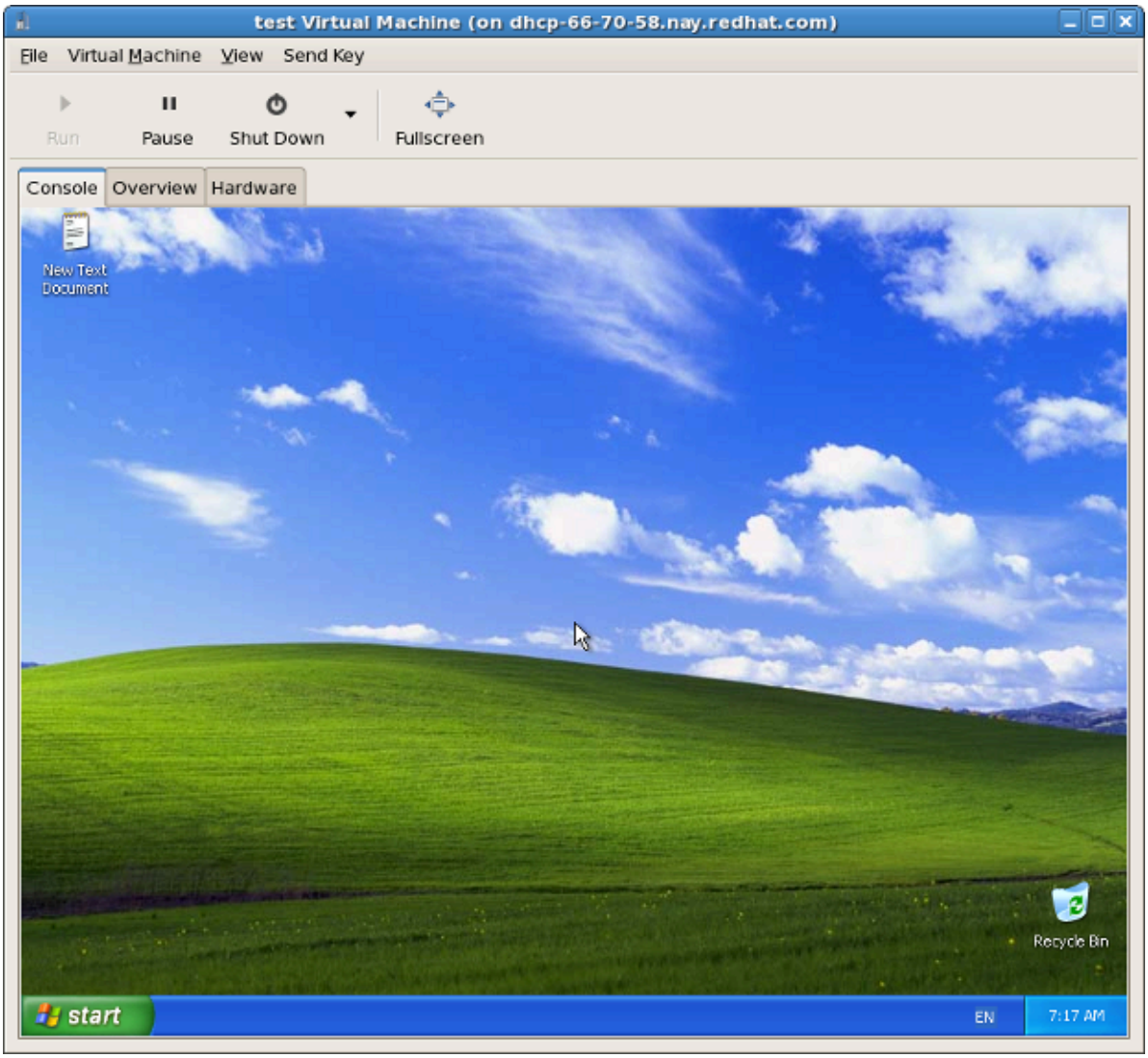
9. Click **Yes** to confirm migration.



The Virtual Machine Manger displays the virtual machine in its new location.



The VNC connection displays the remote host's address in its title bar.


test Virtual Machine (on dhcp-66-70-58.nay.redhat.com)
File
Virtual Machine
View
Send Key
Run
Pause
Shut Down
Fullscreen
Console
Overview
Hardware
New Text Document
Recycle Bin
start
EN
7:17 AM

Remote management of virtualized guests

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

21.1. Remote management with SSH

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

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

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

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

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

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

The user is important for remote management

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

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

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

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

   ```
   $ su -
   ```

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

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

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

   ```
   # ssh-copy-id -i ~/.ssh/id_rsa.pub root@example.com root@example.com's password: Now
    try logging into the machine, with "ssh 'root@example.com'", and check in: .ssh/
   authorized_keys to make sure we haven't added extra keys that you weren't expecting
   ```

   Repeat for other systems, as required.

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

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

The SSH key was added to the remote system.

The `libvirt` daemon (`libvirtd`)

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

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

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

Accessing remote hosts with virt-manager

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

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

21.2. Remote management over TLS and SSL

You can manage virtual machines using TLS and SSL. TLS and SSL provides greater scalability but is more complicated than ssh (refer to *Section 21.1, “Remote management with SSH”*). TLS and SSL

is the same technology used by web browsers for secure connections. The **libvirt** management connection opens a TCP port for incoming connections, which is securely encrypted and authenticated based on x509 certificates. In addition the VNC console for each guest virtual machine will be setup to use TLS with x509 certificate authentication.

This method does not require shell accounts on the remote machines being managed. However, extra firewall rules are needed to access the management service or VNC console. Certificate revocation lists can revoke users' access.

Steps to setup TLS/SSL access for virt-manager

The following short guide assuming you are starting from scratch and you do not have any TLS/SSL certificate knowledge. If you are lucky enough to have a certificate management server you can probably skip the first steps.

libvirt server setup	For more information on creating certificates, refer to the **libvirt** website, *http://libvirt.org/remote.html*.
virt-manager and **virsh** client setup	The setup for clients is slightly inconsistent at this time. To enable the **libvirt** management API over TLS, the CA and client certificates must be placed in **/etc/pki**. For details on this consult *http://libvirt.org/remote.html* In the **virt-manager** user interface, use the '**SSL/TLS**' transport mechanism option when connecting to a host. For **virsh**, the URI has the following format: • **qemu://hostname.guestname/system** for KVM.

To enable SSL and TLS for VNC, it is necessary to put the certificate authority and client certificates into **$HOME/.pki**, that is the following three files:

- CA or **ca-cert.pem** - The CA certificate.
- **libvirt-vnc** or **clientcert.pem** - The client certificate signed by the CA.
- **libvirt-vnc** or **clientkey.pem** - The client private key.

21.3. Transport modes

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

Transport Layer Security (TLS)

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

UNIX sockets

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

SSH

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

ext

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

tcp

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

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

Remote URIs

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

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

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

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

Examples of remote management parameters

- Connect to a remote KVM host named `server7`, using SSH transport and the SSH username `ccurran`.

  ```
  qemu+ssh://ccurran@server7/
  ```

- Connect to a remote KVM hypervisor on the host named `server7` using TLS.

  ```
  qemu://server7/
  ```

- Connect to a remote KVM hypervisor on host `server7` using TLS. The *`no_verify=1`* instructs libvirt not to verify the server's certificate.

```
qemu://server7/?no_verify=1
```

Testing examples

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

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

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

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

Extra URI parameters

Extra parameters can be appended to remote URIs. The table below *Table 21.1, "Extra URI parameters"* covers the recognized parameters. All other parameters are ignored. Note that parameter values must be URI-escaped (that is, a question mark (?) is appended before the parameter and special characters are converted into the URI format).

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

Name	Transport mode	Description	Example usage
		For ssh transport, this is passed to the remote netcat command (see netcat).	
netcat	ssh	The **`netcat`** command can be used to connect to remote systems. The default netcat parameter uses the **nc** command. For SSH transport, libvirt constructs an SSH command using the form below: *`command`* -p *`port`* [-l *`username`*] *`hostname`* *`netcat`* -U socket The *`port`*, *`username`* and *`hostname`* parameters can be specified as part of the remote URI. The *`command`*, *`netcat`* and *`socket`* come from other extra parameters.	netcat=/opt/netcat/bin/nc
no_verify	tls	If set to a non-zero value, this disables client checks of the server's certificate. Note that to disable server checks of the client's certificate or IP address you must change the libvirtd configuration.	no_verify=1
no_tty	ssh	If set to a non-zero value, this stops ssh from asking for a password if it cannot log in to the remote machine automatically (for using ssh-agent or similar). Use this when you do not have access to a terminal - for example	no_tty=1

Name	Transport mode	Description	Example usage
		in graphical programs which use libvirt.	

Table 21.1. Extra URI parameters

KSM

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

KSM is a new Linux feature which uses this concept in reverse. KSM enables the kernel to examine two or more already running programs and compare their memory. If any memory regions are exactly identical, the kernel can combine the two regions into one and mark those regions for copy on write as above.

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

The benefits of KSM are speed and utilization. With KSM, common data is kept in cache or in main memory. This reduces cache misses for the KVM guests which can improve performace for some applications and operating systems. Secondly, sharing memory reduces the overall memory usage of guests which allows for higher densities and greater utilization of resources.

Activating KSM

To be done

Deactivating KSM

To be done

Tuning KSM

To be done

Advanced virtualization administration

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

Note

This chapter is a work in progress. Refer back to this document at a later date.

23.1. Guest scheduling

KVM guests function as Linux processes. By default, KVM guests are prioritised and scheduled with the Linux Completely Fair Scheduler. Tuning the schedule for guest processes may be required for some environments or to prioritize certain guests.

23.2. Advanced memory management

Capping memory available to guests and preventing overcommit on certain guests.

23.3. Guest block I/O throttling

Limit guest block I/O throughput.

23.4. Guest network I/O throttling

Limit guest network activity.

Xen to KVM migration

No longer possible.

24.1. Xen to KVM

This chapter covers migrating Fedora Xen hypervisor guests to Fedora with the KVM hypervisor.

24.2. Older versions of KVM to KVM

This chapter covers migrating Fedora KVM hypervisor guests to Fedora with the KVM hypervisor.

Miscellaneous administration tasks

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

25.1. Automatically starting guests

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

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

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

The guest now automatically starts with the host.

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

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

The guest no longer automatically starts with the host.

25.2. Using qemu-img

The **qemu-img** command line tool is used for formatting various file systems used by KVM. **qemu-img** should be used for formatting virtualized guest images, additional storage devices and network storage. **qemu-img** options and usages are listed below.

Formatting and creating new images or devices

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

```
# qemu-img create [-6] [-e] [-b base_image] [-f format] filename [size]
```

If base_image is specified, then the image will record only the differences from base_image. No size needs to be specified in this case. base_image will never be modified unless you use the "commit" monitor command.

Convert an existing image to another format

The *convert* option is used for converting a recognized format to another image format.

Command format:

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

Convert the disk image *filename* to disk image *output_filename* using format *output_format*. The disk image can be optionally encrypted with the *-e* option or compressed with the *-c* option.

Only the **qcow** format supports encryption or compression. the compression is read-only. it means that if a compressed sector is rewritten, then it is rewritten as uncompressed data.

The encryption uses the AES format with very secure 128-bit keys. Use a long password (over 16 characters) to get maximum protection.

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

getting image information

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

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

give information about the disk image filename. use it in particular to know the size reserved on disk which can be different from the displayed size. if vm snapshots are stored in the disk image, they are displayed too.

Supported formats

The format of an image is usually guessed automatically. The following formats are supported:

raw Raw disk image format (default). This format has the advantage of being simple and easily exportable to all other emulators. If your file system supports holes (for example in ext2 or ext3 on Linux or NTFS on Windows), then only the written sectors will reserve space. Use **qemu-img info** to know the real size used by the image or **ls -ls** on Unix/Linux.

qcow2 QEMU image format, the most versatile format. Use it to have smaller images (useful if your file system does not supports holes, for example: on Windows), optional AES encryption, zlib based compression and support of multiple VM snapshots.

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

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

vmdk VMware 3 and 4 compatible image format.

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

25.3. Overcommitting with KVM

The KVM hypervisor supports overcommitting CPUs and overcommitting memory. Overcommitting is allocating more virtualized CPUs or memory than there are physical resources on the system. With CPU overcommit, under-utilized virtualized servers or desktops can run on fewer servers which saves power and money.

Overcommitting memory

Most operating systems and applications do not use 100% of the available RAM all the time. This behavior can be exploited with KVM to use more memory for virtualized guests than what is physically available.

With KVM, virtual machines are Linux processes. Guests on the KVM hypervisor do not have blocks of physical RAM assigned to them instead they function as processes. Each process is allocated memory when it requests more memory. KVM uses this to allocate memory for guests when the guest operating system requests more or less memory. The guest only uses slightly more physical memory than the virtualized operating system appears to use.

When physical memory is nearly completely used or a process is inactive for some time, Linux moves the process's memory to swap. Swap is usually a partition on a hard disk drive or solid state drive which Linux uses to extend virtual memory. Swap is significantly slower than RAM.

As KVM virtual machines are Linux processes, memory used by virtualized guests can be put into swap if the guest is idle or not in heavy use. Memory can be committed over the total size of the swap and physical RAM. This can cause issues if virtualized guests use their total RAM. Without sufficient swap space for the virtual machine processes to be swapped to the **`pdflush`** process, the cleanup process, starts. **`pdflush`** kills processes to free memory so the system does not crash. **`pdflush`** may destroy virtualized guests or other system processes which may cause file system errors and may leave virtualized guests unbootable.

Warning

If sufficient swap is not available guest operating systems will be forcibly shut down. This may leave guests inoperable. Avoid this by never overcommitting more memory than there is swap available.

The swap partition is used for swapping underused memory to the hard drive to speed up memory performance. The default size of the swap partition is calculated from amount of RAM and overcommit ratio. It is recommended to make your swap partition larger if you intend to overcommit memory with KVM. A recommended overcommit ratio is 50% (0.5). The formula used is:

```
(0.5 * RAM) + (overcommit ratio * RAM) = Recommended swap size
```

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

It is possible to run with an overcommit ratio of ten times the number of virtualized guests over the amount of physical RAM in the system. This only works with certain application loads (for example desktop virtualization with under 100% usage). Setting overcommit ratios is not a hard formula, you must test and customize the ratio for your environment.

Overcommitting virtualized CPUs

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

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

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

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

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

Always test first

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

25.4. Verifying virtualization extensions

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

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

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

2. Analyze the output.

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

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

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

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

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

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

The virtualization extensions may be disabled in the BIOS. If the extensions do not appear or full virtualization does not work refer to *Procedure 32.1, "Enabling virtualization extensions in BIOS"*.

3. **For users of the KVM hypervisor**
   If the kvm package is installed. I As an additional check, verify that the **kvm** modules are loaded in the kernel:

   ```
   # lsmod | grep kvm
   ```

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

Additional output

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

25.5. Accessing data from a guest disk image

There are various methods for accessing the data from guest image files. One common method is to use the **kpartx** tool, covered by this section, to mount the guest file system as a loop device which can then be accessed.

The **kpartx** command creates device maps from partition tables. Each guest storage image has a partition table embedded in the file.

The *libguestfs* and *guestfish* packages, avilable from the *EPEL*[2] repository, allow advanced modification and access to guest file systems. The *libguestfs* and *guestfish* packages are not covered in this section at this time.

Warning

Guests must be offline before their files can be read. Editing or reading files of an active guest is not possible and may cause data loss or damage.

Procedure 25.1. Accessing guest image data

1. Install the *kpartx* package.

   ```
   # yum install kpartx
   ```

2. Use kpartx to list partition device mappings attached to a file-based storage image. This example uses a image file named *guest1.img*.

   ```
   # kpartx -l /var/lib/libvirt/images/guest1.img
   loop0p1 : 0 409600 /dev/loop0 63
   ```

[2] http://fedoraproject.org/wiki/EPEL

```
loop0p2 : 0 10064717 /dev/loop0 409663
```

guest1 is a Linux guest. The first partition is the boot partition and the second partition is an EXT3 containing the root partition.

3. Add the partition mappings to the recognized devices in **/dev/mapper/**.

```
# kpartx -a /var/lib/libvirt/images/guest1.img
```

   - Test that the partition mapping worked. There should be new devices in the /dev/mapper/ directory

```
# ls /dev/mapper/
loop0p1
loop0p2
```

     The mappings for the image are named in the format **loopXpY**.

4. Mount the loop device which to a directory. If required, create the directory. This example uses **/mnt/guest1** for mounting the partition.

```
# mkdir /mnt/guest1
# mount /dev/mapper/loop0p1 /mnt/guest1 -o loop,ro
```

5. The files are now avialable for reading in the **/mnt/guest1** directory. Read or copy the files.

6. Unmount the device so the guest image can be reused by the guest. If the device is mounted the guest cannot access the image and therefore cannot start.

```
# umount /mnt/tmp
```

7. Disconnnect the image file from the partition mappings.

```
# kpartx -d /var/lib/libvirt/images/guest1.img
```

The guest can now be restarted.

Accessing data from guest LVM volumes

Many Linux guests use Logical Volume Management (LVM) volumes. Additional steps are required to read data on LVM volumes on virtual storage images.

1. Add the partition mappings for the *guest1.img* to the recognized devices in the **/dev/mapper/** directory.

```
# kpartx -a /var/lib/libvirt/images/guest1.img
```

2. In this example the LVM volumes are on a second partition. The volumes require a rescan with the **vgscan** command to find the new volume groups.

```
# vgscan
Reading all physical volumes . This may take a while...
Found volume group "VolGroup00" using metadata type lvm2
```

3. Activate the volume group on the partition (called **VolGroup00** by default) with the **vgchange -ay** command.

```
# vgchange -ay VolGroup00
2 logical volumes in volume group VolGroup00 now active.
```

4. Use the **lvs** command to display information about the new volumes. The volume names (the **LV** column) are required to mount the volumes.

```
# lvs
LV VG Attr Lsize Origin Snap% Move Log Copy%
LogVol00 VolGroup00 -wi-a- 5.06G
LogVol01 VolGroup00 -wi-a- 800.00M
```

5. Mount **/dev/VolGroup00/LogVol00** in the **/mnt/guestboot/** directory.

```
# mount /dev/VolGroup00/LogVol00 /mnt/guestboot
```

6. The files are now avialable for reading in the **/mnt/guestboot** directory. Read or copy the files.

7. Unmount the device so the guest image can be reused by the guest. If the device is mounted the guest cannot access the image and therefore cannot start.

```
# umount /mnt/
```

8. Disconnect the volume group *VolGroup00*

```
# vgchange -an VolGroup00
```

9. Disconnnect the image file from the partition mappings.

```
# kpartx -d /var/lib/libvirt/images/guest1.img
```

The guest can now be restarted.

25.6. Setting KVM processor affinities

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

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

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

Identifying CPU and NUMA topology

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

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

This system has eight CPUs, in two sockets, each processor has four cores.

The output shows that that the system has a NUMA architecture. NUMA is more complex and requires more data to accurately interpret. Use the **`virsh capabilities`** to get additional output data on the CPU configuration.

```
# virsh capabilities
<capabilities>
  <host>
    <cpu>
      <arch>x86_64</arch>
    </cpu>
    <migration_features>
      <live/>
      <uri_transports>
        <uri_transport>tcp</uri_transport>
      </uri_transports>
    </migration_features>
    <topology>
      <cells num='2'>
        <cell id='0'>
          <cpus num='4'>
            <cpu id='0'/>
            <cpu id='1'/>
            <cpu id='2'/>
            <cpu id='3'/>
          </cpus>
        </cell>
        <cell id='1'>
          <cpus num='4'>
            <cpu id='4'/>
            <cpu id='5'/>
            <cpu id='6'/>
            <cpu id='7'/>
          </cpus>
        </cell>
      </cells>
    </topology>
    <secmodel>
      <model>selinux</model>
      <doi>0</doi>
```

```
    </secmodel>
  </host>

 [ Additional XML removed ]

</capabilities>
```

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

If a guest requires eight virtual CPUs, as each NUMA node only has four physical CPUs, a better utilization may be obtained by running a pair of four virtual CPU guests and splitting the work between them, rather than using a single 8 CPU guest. Running across multiple NUMA nodes significantly degrades performance for physical and virtualized tasks.

Decide which NUMA node can run the guest

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

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

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

Lock a guest to a NUMA node or physical CPU set

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

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

```
<topology>
  <cells num='2'>
    <cell id='0'>
    <cpus num='4'>
      <cpu id='0'/>
      <cpu id='1'/>
      <cpu id='2'/>
      <cpu id='3'/>
    </cpus>
  </cell>
  <cell id='1'>
    <cpus num='4'>
      <cpu id='4'/>
      <cpu id='5'/>
      <cpu id='6'/>
      <cpu id='7'/>
    </cpus>
  </cell>
  </cells>
```

```
</topology>
```

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

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

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

   b. Locate where the guest's virtual CPU count is specified. Find the **`vcpus`** element.

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

   The guest in this example has four CPUs.

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

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

4. Save the configuration file and restart the guest.

The guest has been locked to CPUs 4 to 7.

Automatically locking guests to CPUs with virt-install

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

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

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

Tuning CPU affinity on running guests

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

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

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

```
# virsh vcpuinfo guest1
VCPU:           0
CPU:            3
State:          running
CPU time:       0.5s
CPU Affinity:   yyyyyyyy
VCPU:           1
CPU:            1
State:          running
CPU Affinity:   yyyyyyyy
VCPU:           2
```

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

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

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

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

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

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

Information from the KVM processes can also confirm that the guest is now running on the second NUMA node.

```
# grep pid /var/run/libvirt/qemu/guest1.xml
<domstatus state='running' pid='4907'>
# grep Cpus_allowed_list /proc/4907/task/*/status
/proc/4907/task/4916/status:Cpus_allowed_list: 4
/proc/4907/task/4917/status:Cpus_allowed_list: 5
/proc/4907/task/4918/status:Cpus_allowed_list: 6
/proc/4907/task/4919/status:Cpus_allowed_list: 7
</section>
```

25.7. Generating a new unique MAC address

In some case you will need to generate a new and unique *MAC address* for a guest. There is no command line tool available to generate a new MAC address at the time of writing. The script

provided below can generate a new MAC address for your guests. Save the script to your guest as **macgen.py**. Now from that directory you can run the script using **./macgen.py** and it will generate a new MAC address. A sample output would look like the following:

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

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

Another method to generate a new MAC for your guest

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

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

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

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

25.8. Very Secure ftpd

vsftpd can provide access to installation trees for para-virtualized guests (for example, the Fedora repositories) or other data. If you have not installed vsftpd during the server installation you can grab the RPM package from your **Server** directory of your installation media and install it using the **rpm -ivh vsftpd*.rpm** (note that the RPM package must be in your current directory).

1. To configure vsftpd, edit **/etc/passwd** using **vipw** and change the ftp user's home directory to the directory where you are going to keep the installation trees for your para-virtualized guests. An example entry for the FTP user would look like the following:

   ```
   ftp:x:14:50:FTP User:/installtree/:/sbin/nologin
   ```

2. Verify that `vsftpd` is not enabled using the **`chkconfig --list vsftpd`**:

```
$ chkconfig --list vsftpd
vsftpd          0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

3. Run the **`chkconfig --levels 345 vsftpd on`** to start vsftpd automatically for run levels 3, 4 and 5.

4. Use the **`chkconfig --list vsftpd`** command to verify the **`vsftpd`** daemon is enabled to start during system boot:

```
$ chkconfig --list vsftpd
vsftpd          0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

5. use the **`service vsftpd start vsftpd`** to start the vsftpd service:

```
$service vsftpd start vsftpd
Starting vsftpd for vsftpd:                  [  OK  ]
```

25.9. Configuring LUN Persistence

This section covers how to implement *LUN* persistence in guests and on the host machine with and without multipath.

Implementing LUN persistence without multipath

If your system is not using multipath, you can use **`udev`** to implement LUN persistence. Before implementing LUN persistence in your system, ensure that you acquire the proper UUIDs. Once you acquire these, you can configure LUN persistence by editing the **`scsi_id`** file that resides in the **`/etc`** directory. Once you have this file open in a text editor, you must comment out this line:

```
# options=-b
```

Then replace it with this parameter:

```
# options=-g
```

This tells udev to monitor all system SCSI devices for returning UUIDs. To determine the system UUIDs, use the **`scsi_id`** command:

```
# scsi_id -g -s /block/sdc
*3600a0b8000132751000001542 7b625e*
```

The long string of characters in the output is the UUID. The UUID does not change when you add a new device to your system. Acquire the UUID for each device in order to create rules for the devices.

To create new device rules, edit the **20-names.rules** file in the **/etc/udev/rules.d** directory. The device naming rules follow this format:

```
# KERNEL="sd*",  BUS="scsi",  PROGRAM="sbin/scsi_id", RESULT="UUID", NAME="devicename"
```

Replace your existing *UUID* and *devicename* with the above UUID retrieved entry. The rule should resemble the following:

```
KERNEL="sd*",  BUS="scsi",  PROGRAM="sbin/scsi_id", RESULT="3600a0b8000132751000001542 7b625e",
 NAME="mydevicename"
```

This enables all devices that match the **/dev/sd*** pattern to inspect the given UUID. When it finds a matching device, it creates a device node called */dev/devicename*. For this example, the device node is */dev/mydevice* . Finally, append the **/etc/rc.local** file with this line:

```
/sbin/start_udev
```

Implementing LUN persistence with multipath

To implement LUN persistence in a multipath environment, you must define the alias names for the multipath devices. For this example, you must define four device aliases by editing the **multipath.conf** file that resides in the **/etc/** directory:

```
multipath  {
             wwid       3600a0b8000132751000001542 7b625e
             alias      oramp1
}
multipath  {
             wwid       3600a0b8000132751000001542 7b6
             alias      oramp2
}
multipath  {
             wwid       3600a0b8000132751000001542 7b625e
             alias      oramp3
}
multipath  {
             wwid       3600a0b8000132751000001542 7b625e
             alias      oramp4
}
```

This defines 4 LUNs: **/dev/mpath/oramp1**, **/dev/mpath/oramp2**, **/dev/mpath/oramp3**, and **dev/mpath/oramp4**. The devices will reside in the **/dev/mpath** directory. These LUN names are persistent after reboots as it creates aliased names on the wwid for each of the LUNs.

25.10. Disable SMART disk monitoring for guests

SMART disk monitoring can be disabled as we are running on virtual disks and the physical storage is managed by the host.

```
/sbin/service smartd stop
/sbin/chkconfig --del smartd
```

25.11. Configuring a VNC Server

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

The following steps set up a dedicated VNC server session:

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

2. A sample **`xstartup`** file:

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

Part V. Virtualization storage topics

Introduction to storage administration for virtualization

This part covers using shared, networked storage with virtualization on Fedora.

The following methods are working for virtualization:

- Fibre Channel
- iSCSI
- NFS
- GFS2

Networked storage is essential for live and offline guest migrations. You cannot migrate guests without shared storage.

Using shared storage with virtual disk images

This chapter covers using various types of shared and network storage devices for virtual disks.

26.1. Using iSCSI for storing virtual disk images

This chapter covers using iSCSI-based devices to store virtualized guests.

26.2. Using NFS for storing virtual disk images

This chapter covers using NFS to store virtualized guests.

26.3. Using GFS2 for storing virtual disk images

This chapter covers using the Red Hat Global File System 2 (GFS2) to store virtualized guests.

26.4. Storage Pools

Using storage pools in RHEL

26.4.1. Configuring storage devices for pools

How to set up the device/RHEL for storage pools for iSCSI, GFS and (maybe) Fibre Channel.

26.4.2. Mapping virtualized guests to storage pools

libvirt example for mapping storage pools

Part VI. Virtualization reference guide

Virtualization commands, system tools, applications and additional systems reference

These chapters provide detailed descriptions of virtualization commands, system tools, and applications included in Fedora. These chapters are designed for users requiring information on advanced functionality and other features.

Virtualization tools

The following is a list of tools for virtualization administration, debugging and networking.

System Administration Tools

- **vmstat**
- **iostat**
- **lsof**
- **qemu-img**

Advanced Debugging Tools

- **systemTap**
- **crash**
- **sysrq**
- **sysrq t**
- **sysrq w**
- **sysrq c**

Networking

brtcl

- ```
  # brctl show
  bridge name bridge id  STP enabled
   interfaces
  pan0  8000.000000000000 no
  virbr0  8000.000000000000 yes
  ```
- ```
  # brctl showmacs virbr0
  port no mac addr  is local? ageing timer
  ```
- ```
  # brctl showstp virbr0
  virbr0
   bridge id  8000.000000000000
   designated root 8000.000000000000
   root port     0   path cost     0
   max age    19.99   bridge max age
   19.99
   hello time     1.99   bridge hello time
    1.99
   forward delay     0.00   bridge forward
   delay    0.00
   ageing time   299.95
   hello timer     1.39   tcn timer     0.00
   topology change timer    0.00   gc timer
      0.39
  ```
- **ifconfig**
- **tcpdump**
  ```

KVM tools	• **`ps`** • **`pstree`** • **`top`** • **`kvmtrace`** • **`kvm_stat`**

Managing guests with virsh

virsh is a command line interface tool for managing guests and the hypervisor.

The **virsh** tool is built on the **libvirt** management API and operates as an alternative to the **xm** command and the graphical guest Manager (**virt-manager**). **virsh** can be used in read-only mode by unprivileged users. You can use **virsh** to execute scripts for the guest machines.

virsh command quick reference

The following tables provide a quick reference for all virsh command line options.

Command	Description
help	Prints basic help information.
list	Lists all guests.
dumpxml	Outputs the XML configuration file for the guest.
create	Creates a guest from an XML configuration file and starts the new guest.
start	Starts an inactive guest.
destroy	Forces a guest to stop.
define	Outputs an XML configuration file for a guest.
domid	Displays the guest's ID.
domuuid	Displays the guest's UUID.
dominfo	Displays guest information.
domname	Displays the guest's name.
domstate	Displays the state of a guest.
quit	Quits the interactive terminal.
reboot	Reboots a guest.
restore	Restores a previously saved guest stored in a file.
resume	Resumes a paused guest.
save	Save the present state of a guest to a file.
shutdown	Gracefully shuts down a guest.
suspend	Pauses a guest.
undefine	Deletes all files associated with a guest.
migrate	Migrates a guest to another host.

Table 28.1. Guest management commands

The following **virsh** command options manage guest and hypervisor resources:

Command	Description
setmem	Sets the allocated memory for a guest.
setmaxmem	Sets maximum memory limit for the hypervisor.

Command	Description
setvcpus	Changes number of virtual CPUs assigned to a guest.
vcpuinfo	Displays virtual CPU information about a guest.
vcpupin	Controls the virtual CPU affinity of a guest.
domblkstat	Displays block device statistics for a running guest.
domifstat	Displays network interface statistics for a running guest.
attach-device	Attach a device to a guest, using a device definition in an XML file.
attach-disk	Attaches a new disk device to a guest.
attach-interface	Attaches a new network interface to a guest.
detach-device	Detach a device from a guest, takes the same kind of XML descriptions as command **attach-device**.
detach-disk	Detach a disk device from a guest.
detach-interface	Detach a network interface from a guest.

Table 28.2. Resource management options

These are miscellaneous **virsh** options:

Command	Description
version	Displays the version of **virsh**
nodeinfo	Outputs information about the hypervisor

Table 28.3. Miscellaneous options

Connecting to the hypervisor

Connect to a hypervisor session with **virsh**:

```
# virsh connect {hostname OR URL}
```

Where **<name>** is the machine name of the hypervisor. To initiate a read-only connection, append the above command with **-readonly**.

Creating a virtual machine XML dump (configuration file)

Output a guest's XML configuration file with **virsh**:

```
# virsh dumpxml {guest-id, guestname or uuid}
```

This command outputs the guest's XML configuration file to standard out (**stdout**). You can save the data by piping the output to a file. An example of piping the output to a file called *guest.xml*:

```
# virsh dumpxml GuestID > guest.xml
```

This file **guest.xml** can recreate the guest (refer to *Editing a guest's configuration file*. You can edit this XML configuration file to configure additional devices or to deploy additional guests. Refer to *Section 31.1, "Using XML configuration files with virsh"* for more information on modifying files created with **virsh dumpxml**.

An example of **virsh dumpxml** output:

```
# virsh dumpxml r5b2-mySQL01
<domain type='kvm' id='13'>
    <name>r5b2-mySQL01</name>
    <uuid>4a4c59a7ee3fc78196e4288f2862f011</uuid>
    <bootloader>/usr/bin/pygrub</bootloader>
    <os>
        <type>linux</type>
        <kernel>/var/lib/libvirt/vmlinuz.2dgnU_</kernel>
 <initrd>/var/lib/libvirt/initrd.UQafMw</initrd>
        <cmdline>ro root=/dev/VolGroup00/LogVol00 rhgb quiet</cmdline>
    </os>
    <memory>512000</memory>
    <vcpu>1</vcpu>
    <on_poweroff>destroy</on_poweroff>
    <on_reboot>restart</on_reboot>
    <on_crash>restart</on_crash>
    <devices>
        <interface type='bridge'>
            <source bridge='br0'/>
            <mac address='00:16:3e:49:1d:11'/>
            <script path='bridge'/>
        </interface>
        <graphics type='vnc' port='5900'/>
        <console tty='/dev/pts/4'/>
    </devices>
</domain>
```

Creating a guest from a configuration file

Guests can be created from XML configuration files. You can copy existing XML from previously created guests or use the **dumpxml** option (refer to *Creating a virtual machine XML dump (configuration file)*). To create a guest with **virsh** from an XML file:

```
# virsh create configuration_file.xml
```

Editing a guest's configuration file

Instead of using the **dumpxml** option (refer to *Creating a virtual machine XML dump (configuration file)*) guests can be edited either while they run or while they are offline. The **virsh edit** command provides this functionality. For example, to edit the guest named *softwaretesting*:

```
# virsh edit softwaretesting
```

This opens a text editor. The default text editor is the **$EDITOR** shell parameter (set to **vi** by default).

Suspending a guest

Suspend a guest with **virsh**:

```
# virsh suspend {domain-id, domain-name or domain-uuid}
```

When a guest is in a suspended state, it consumes system RAM but not processor resources. Disk and network I/O does not occur while the guest is suspended. This operation is immediate and the guest can be restarted with the **resume** (*Resuming a guest*) option.

Resuming a guest

Restore a suspended guest with **virsh** using the **resume** option:

```
# virsh resume {domain-id, domain-name or domain-uuid}
```

This operation is immediate and the guest parameters are preserved for **suspend** and **resume** operations.

Save a guest

Save the current state of a guest to a file using the **virsh** command:

```
# virsh save {domain-name, domain-id or domain-uuid} filename
```

This stops the guest you specify and saves the data to a file, which may take some time given the amount of memory in use by your guest. You can restore the state of the guest with the **restore** (*Restore a guest*) option. Save is similar to pause, instead of just pausing a guest the present state of the guest is saved.

Restore a guest

Restore a guest previously saved with the **virsh save** command (*Save a guest*) using **virsh**:

```
# virsh restore filename
```

This restarts the saved guest, which may take some time. The guest's name and UUID are preserved but are allocated for a new id.

Shut down a guest

Shut down a guest using the **virsh** command:

```
# virsh shutdown {domain-id, domain-name or domain-uuid}
```

You can control the behavior of the rebooting guest by modifying the **on_shutdown** parameter in the guest's configuration file.

Rebooting a guest

Reboot a guest using **virsh** command:

```
#virsh reboot {domain-id, domain-name or domain-uuid}
```

You can control the behavior of the rebooting guest by modifying the **on_reboot** element in the guest's configuration file.

Forcing a guest to stop

Force a guest to stop with the **virsh** command:

```
# virsh destroy {domain-id, domain-name or domain-uuid}
```

This command does an immediate ungraceful shutdown and stops the specified guest. Using **virsh destroy** can corrupt guest file systems . Use the **destroy** option only when the guest is unresponsive. For para-virtualized guests, use the **shutdown** option(*Shut down a guest*) instead.

Getting the domain ID of a guest

To get the domain ID of a guest:

```
# virsh domid {domain-name or domain-uuid}
```

Getting the domain name of a guest

To get the domain name of a guest:

```
# virsh domname {domain-id or domain-uuid}
```

Getting the UUID of a guest

To get the Universally Unique Identifier (UUID) for a guest:

```
# virsh domuuid {domain-id or domain-name}
```

An example of **virsh domuuid** output:

```
# virsh domuuid r5b2-mySQL01
4a4c59a7-ee3f-c781-96e4-288f2862f011
```

Displaying guest Information

Using **virsh** with the guest's domain ID, domain name or UUID you can display information on the specified guest:

```
# virsh dominfo {domain-id, domain-name or domain-uuid}
```

This is an example of **virsh dominfo** output:

```
# virsh dominfo r5b2-mySQL01
id:             13
name:           r5b2-mysql01
uuid:           4a4c59a7-ee3f-c781-96e4-288f2862f011
os type:       linux
state:          blocked
cpu(s):         1
cpu time:      11.0s
max memory:     512000 kb
used memory:    512000 kb
```

Displaying host information

To display information about the host:

```
# virsh nodeinfo
```

An example of **`virsh nodeinfo`** output:

```
# virsh nodeinfo
CPU model                    x86_64
CPU (s)                      8
CPU frequency                2895 Mhz
CPU socket(s)                2
Core(s) per socket           2
Threads per core:            2
Numa cell(s)                 1
Memory size:                 1046528 kb
```

This displays the node information and the machines that support the virtualization process.

Displaying the guests

To display the guest list and their current states with **`virsh`**:

```
# virsh list
```

Other options available include:

the **`--inactive`** option to list inactive guests (that is, guests that have been defined but are not currently active), and

the **`--all`** option lists all guests. For example:

```
# virsh list --all
 Id Name                 State
----------------------------------
  0 Domain-0             running
  1 Domain202            paused
  2 Domain010            inactive
  3 Domain9600           crashed
```

The output from **`virsh list`** is categorized as one of the six states (listed below).

- The **`running`** state refers to guests which are currently active on a CPU.
- Guests listed as **`blocked`** are blocked, and are not running or runnable. This is caused by a guest waiting on I/O (a traditional wait state) or guests in a sleep mode.
- The **`paused`** state lists domains that are paused. This occurs if an administrator uses the **pause** button in **`virt-manager`**, **`xm pause`** or **`virsh suspend`**. When a guest is paused it consumes memory and other resources but it is ineligible for scheduling and CPU resources from the hypervisor.

- The **shutdown** state is for guests in the process of shutting down. The guest is sent a shutdown signal and should be in the process of stopping its operations gracefully. This may not work with all guest operating systems; some operating systems do not respond to these signals.
- Domains in the **dying** state are in is in process of dying, which is a state where the domain has not completely shut-down or crashed.
- **crashed** guests have failed while running and are no longer running. This state can only occur if the guest has been configured not to restart on crash.

Displaying virtual CPU information

To display virtual CPU information from a guest with **virsh**:

```
# virsh vcpuinfo {domain-id, domain-name or domain-uuid}
```

An example of **virsh vcpuinfo** output:

```
# virsh vcpuinfo r5b2-mySQL01
VCPU:           0
CPU:            0
State:          blocked
CPU time:       0.0s
CPU Affinity:   yy
```

Configuring virtual CPU affinity

To configure the affinity of virtual CPUs with physical CPUs:

```
# virsh vcpupin domain-id vcpu cpulist
```

The **domain-id** parameter is the guest's ID number or name.

The **vcpu** parameter denotes the number of virtualized CPUs allocated to the guest.The **vcpu** parameter must be provided.

The **cpulist** parameter is a list of physical CPU identifier numbers separated by commas. The **cpulist** parameter determines which physical CPUs the VCPUs can run on.

Configuring virtual CPU count

To modify the number of CPUs assigned to a guest with **virsh**:

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count
```

The new *count* value cannot exceed the count above the amount specified when the guest was created.

Configuring memory allocation

To modify a guest's memory allocation with **virsh** :

```
# virsh setmem {domain-id or domain-name} count
```

You must specify the *count* in kilobytes. The new count value cannot exceed the amount you specified when you created the guest. Values lower than 64 MB are unlikely to work with most guest operating systems. A higher maximum memory value does not affect an active guests. If the new value is lower the available memory will shrink and the guest may crash.

Displaying guest block device information

Use **`virsh domblkstat`** to display block device statistics for a running guest.

```
# virsh domblkstat GuestName block-device
```

Displaying guest network device information

Use **`virsh domifstat`** to display network interface statistics for a running guest.

```
# virsh domifstat GuestName interface-device
```

Migrating guests with virsh

A guest can be migrated to another host with **`virsh`**. Migrate domain to another host. Add --live for live migration. The **`migrate`** command accepts parameters in the following format:

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

The *`--live`* parameter is optional. Add the *`--live`* parameter for live migrations.

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

The *`DestinationURL`* parameter is the URL or hostname of the destination system. The destination system requires:

- Fedora 9 or newer,
- the same hypervisor version, and
- the **`libvirt`** service must be started.

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

Managing virtual networks

This section covers managing virtual networks with the **`virsh`** command. To list virtual networks:

```
# virsh net-list
```

This command generates output similar to:

```
# virsh net-list
Name                 State      Autostart
-----------------------------------------
```

```
default              active     yes
vnet1              active     yes
vnet2              active     yes
```

To view network information for a specific virtual network:

```
# virsh net-dumpxml NetworkName
```

This displays information about a specified virtual network in XML format:

```
# virsh net-dumpxml vnet1
<network>
  <name>vnet1</name>
  <uuid>98361b46-1581-acb7-1643-85a412626e70</uuid>
  <forward dev='eth0'/>
  <bridge name='vnet0' stp='on' forwardDelay='0' />
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254' />
    </dhcp>
  </ip>
</network>
```

Other **virsh** commands used in managing virtual networks are:

- **virsh net-autostart *network-name*** — Autostart a network specified as *network-name*.
- **virsh net-create *XMLfile*** — generates and starts a new network using an existing XML file.
- **virsh net-define *XMLfile*** — generates a new network device from an existing XML file without starting it.
- **virsh net-destroy *network-name*** — destroy a network specified as *network-name*.
- **virsh net-name *networkUUID*** — convert a specified *networkUUID* to a network name.
- **virsh net-uuid *network-name*** — convert a specified *network-name* to a network UUID.
- **virsh net-start *nameOfInactiveNetwork*** — starts an inactive network.
- **virsh net-undefine *nameOfInactiveNetwork*** — removes the definition of an inactive network.

Managing guests with the Virtual Machine Manager (virt-manager)

This section describes the Virtual Machine Manager (**`virt-manager`**) windows, dialog boxes, and various GUI controls.

`virt-manager` provides a graphical view of hypervisors and guest on your system and on remote machines. You can use **`virt-manager`** to define both para-virtualized and fully virtualized guests. **`virt-manager`** can perform virtualization management tasks, including:

- assigning memory,
- assigning virtual CPUs,
- monitoring operational performance,
- saving and restoring, pausing and resuming, and shutting down and starting virtualized guests,
- links to the textual and graphical consoles, and
- live and offline migrations.

29.1. The Add Connection window

This window appears first and prompts the user to choose a hypervisor session. Non-privileged users can initiate a read-only session. Root users can start a session with full blown read-write status. For normal use, select the QEMU option for KVM.

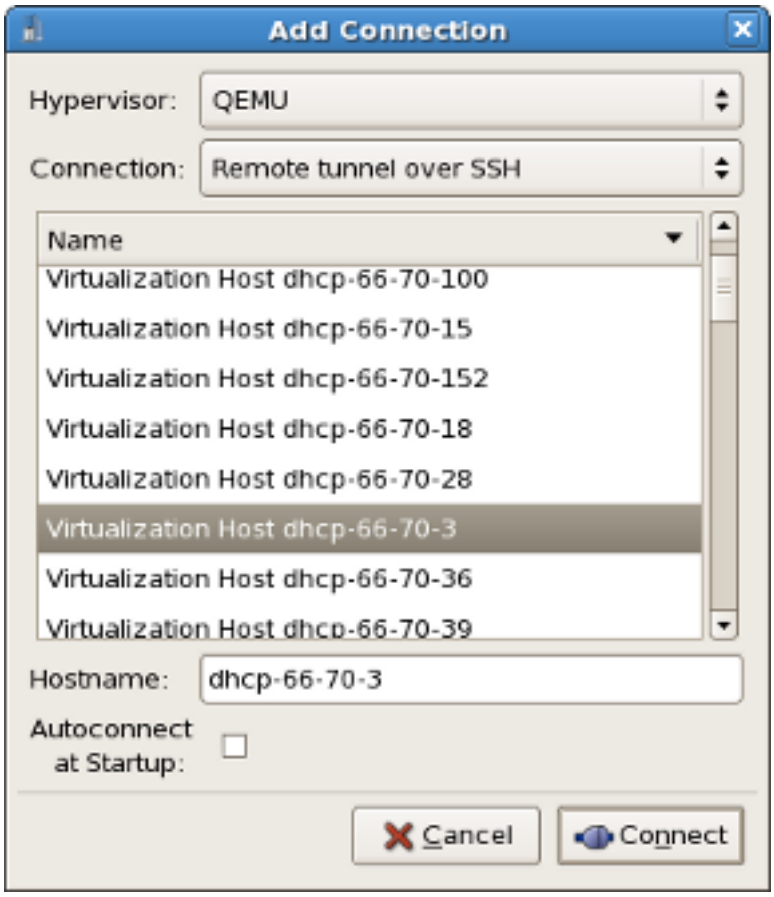


Figure 29.1. Virtual Machine Manager connection window

29.2. The Virtual Machine Manager main window

This main window displays all the running guests and resources used by guests. Select a virtualized guest by double clicking the guest's name.

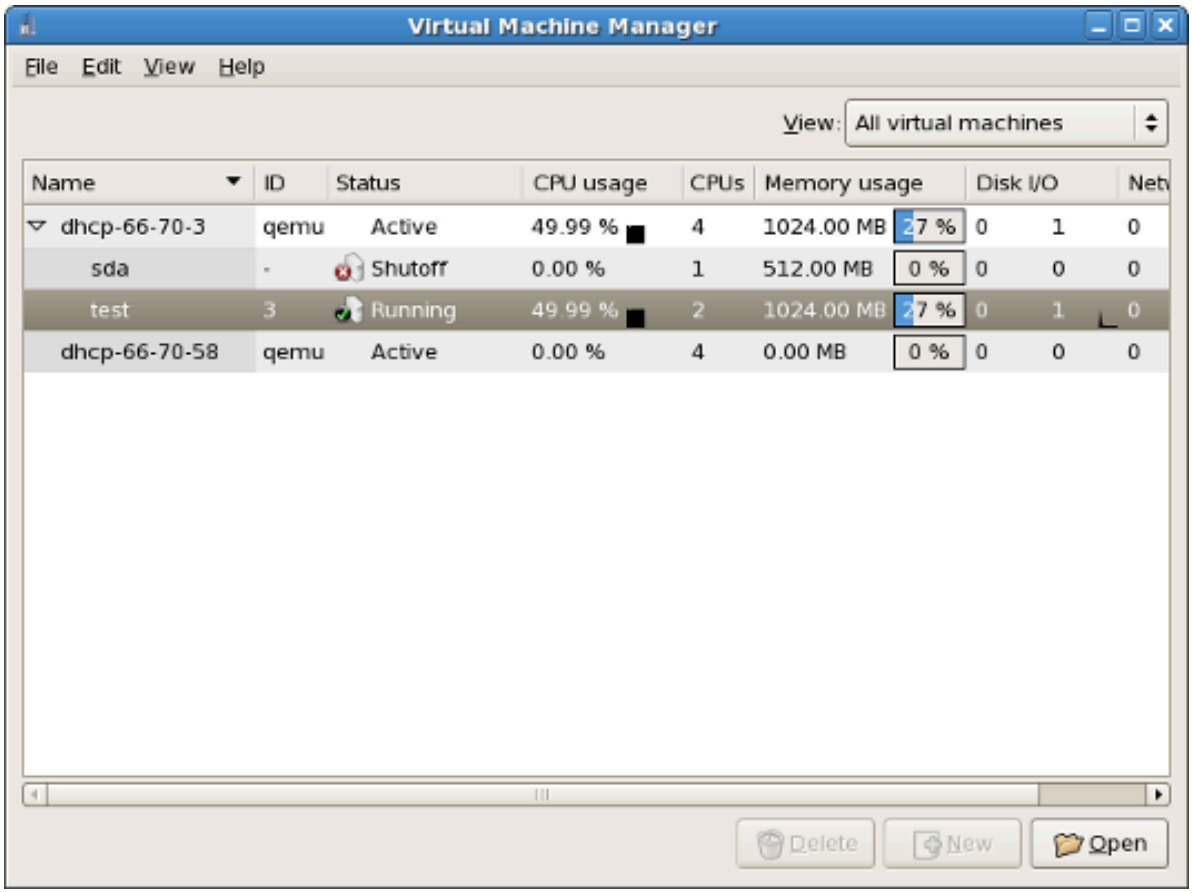


Figure 29.2. Virtual Machine Manager main window

29.3. The guest Overview tab

The **Overview** tab displays graphs and statistics of a guest's live resource utilization data available from **`virt-manager`**. The UUID field displays the globally unique identifier for the virtual machines.

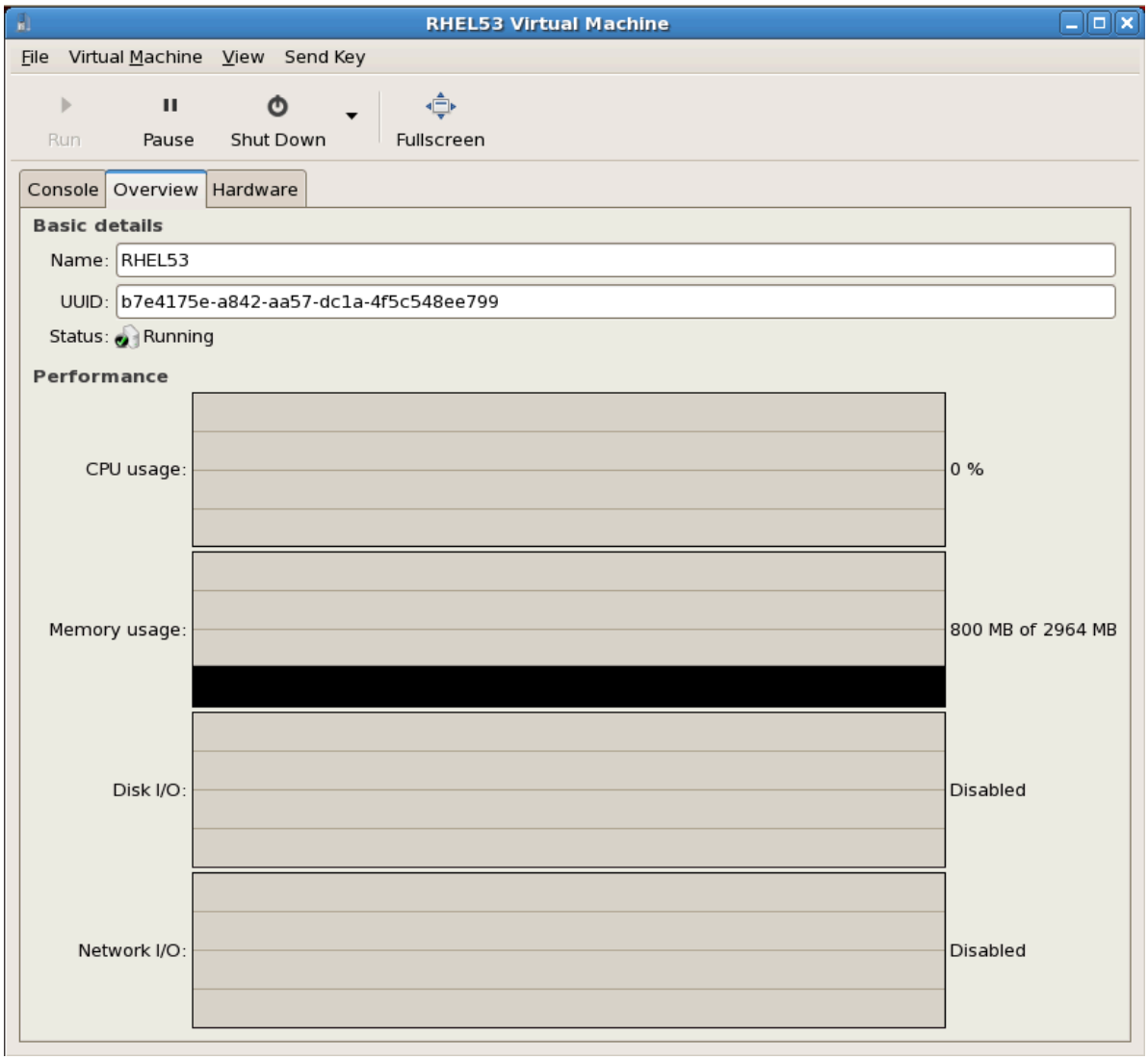


Figure 29.3. The **Overview** tab

29.4. Virtual Machine graphical console

This window displays a virtual machine's graphical console. Para-virtualized and fully virtualized guests use different techniques to export their local virtual framebuffers, but both technologies use **VNC** to make them available to the Virtual Machine Manager's console window. If your virtual machine is set to require authentication, the Virtual Machine Graphical console prompts you for a password before the display appears.

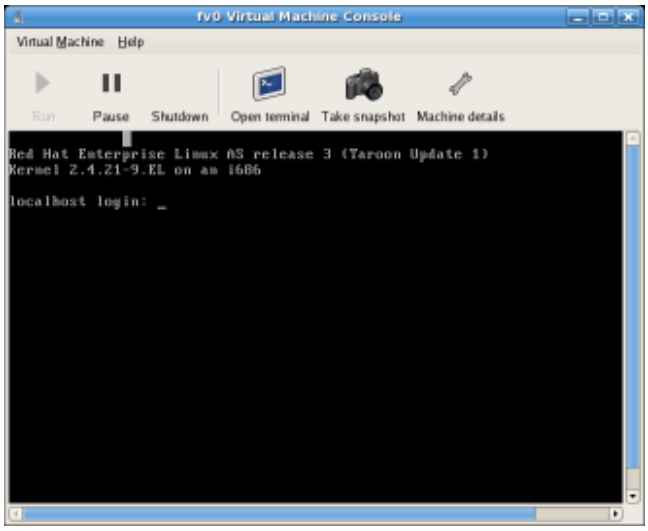


Figure 29.4. Graphical console window

A note on security and VNC

VNC is considered insecure by many security experts, however, several changes have been made to enable the secure usage of VNC for virtualization. The guest machines only listen to the local host (`dom0`)'s loopback address (`127.0.0.1`). This ensures only those with shell privileges on the host can access virt-manager and the virtual machine through VNC.

Remote administration can be performed following the instructions in *Chapter 21, Remote management of virtualized guests*. TLS can provide enterprise level security for managing guest and host systems.

Your local desktop can intercept key combinations (for example, Ctrl+Alt+F11) to prevent them from being sent to the guest machine. You can use **`virt-manager`**sticky key' capability to send these sequences. You must press any modifier key (Ctrl or Alt) 3 times and the key you specify gets treated as active until the next non-modifier key is pressed. Then you can send Ctrl-Alt-F11 to the guest by entering the key sequence 'Ctrl Ctrl Ctrl Alt+F1'.

Try Open SPICE.

29.5. Starting virt-manager

To start **`virt-manager`** session open the **Applications** menu, then the **System Tools** menu and select **Virtual Machine Manager** (**`virt-manager`**).

The **`virt-manager`** main window appears.

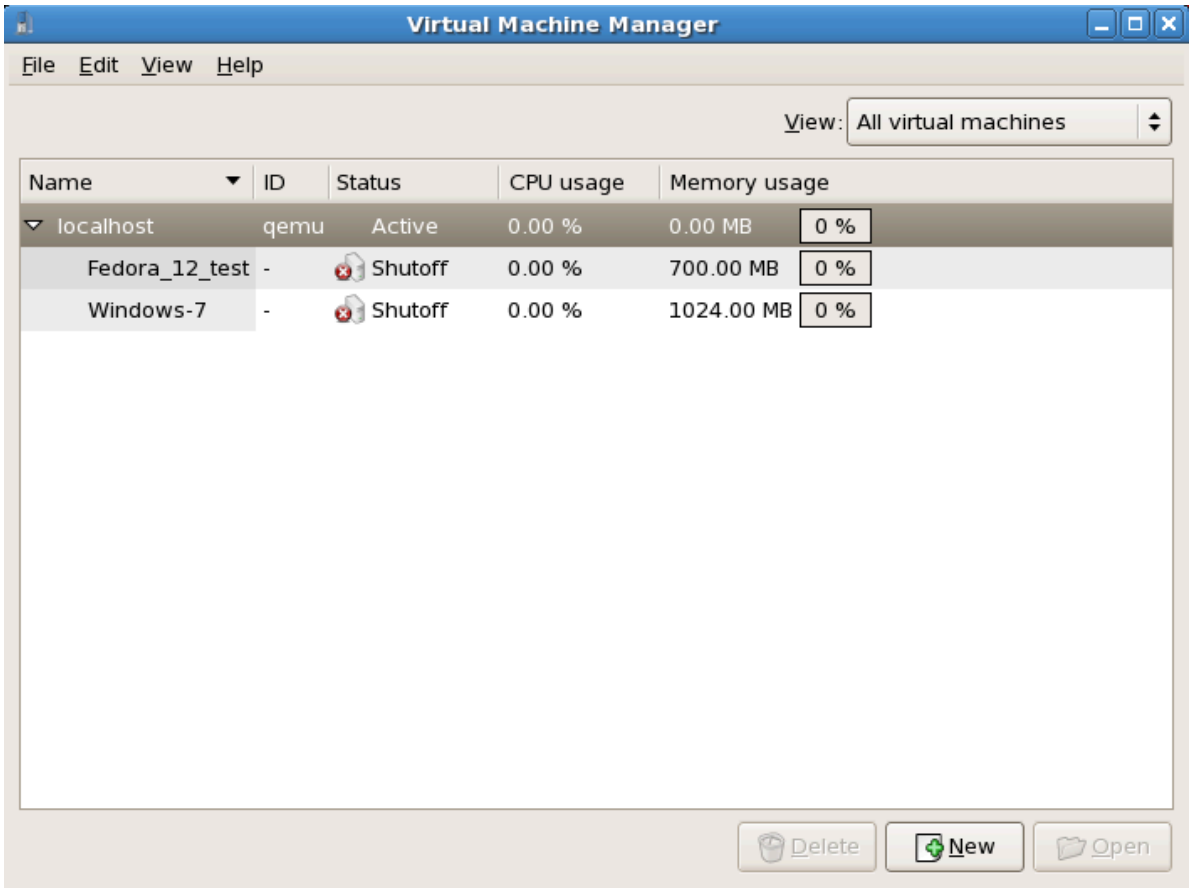


Figure 29.5. Starting **virt-manager**

Alternatively, **virt-manager** can be started remotely using ssh as demonstrated in the following command:

```
ssh -X host's address[remotehost]# virt-manager
```

Using **ssh** to manage virtual machines and hosts is discussed further in *Section 21.1, "Remote management with SSH"*.

29.6. Restoring a saved machine

After you start the Virtual Machine Manager, all virtual machines on your system are displayed in the main window. Domain0 is your host system. If there are no machines present, this means that currently there are no machines running on the system.

To restore a previously saved session:

1. From the **File** menu, select **Restore a saved machine**.

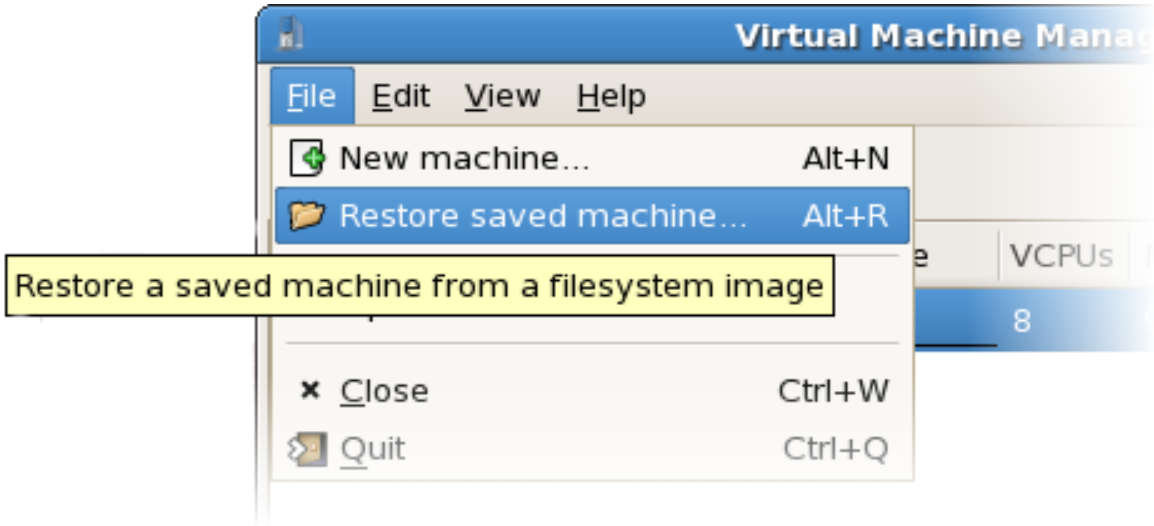


Figure 29.6. Restoring a virtual machine

2. The **Restore Virtual Machine** main window appears.
3. Navigate to correct directory and select the saved session file.
4. Click **Open**.

The saved virtual system appears in the Virtual Machine Manager main window.

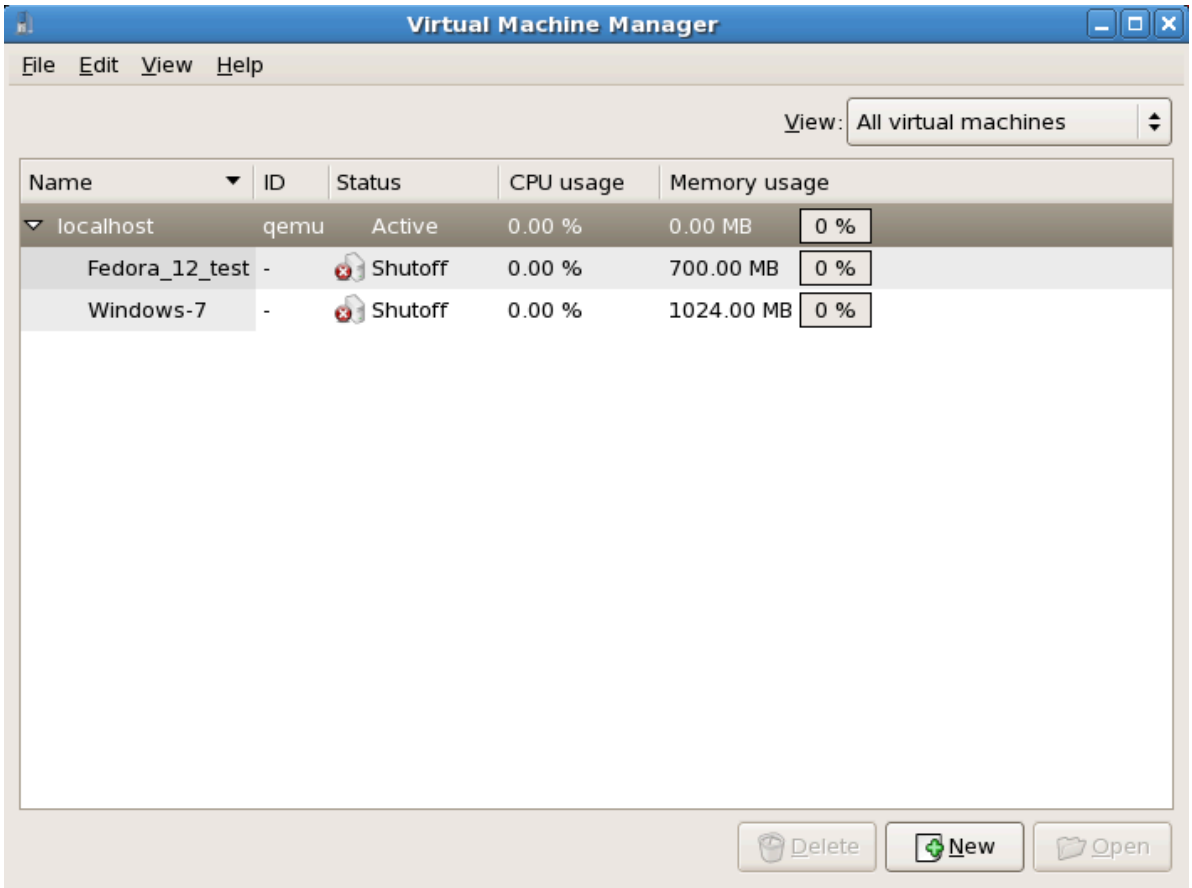


Figure 29.7. A restored virtual machine manager session

29.7. Displaying guest details

You can use the Virtual Machine Monitor to view activity data information for any virtual machines on your system.

To view a virtual system's details:

1. In the Virtual Machine Manager main window, highlight the virtual machine that you want to view.

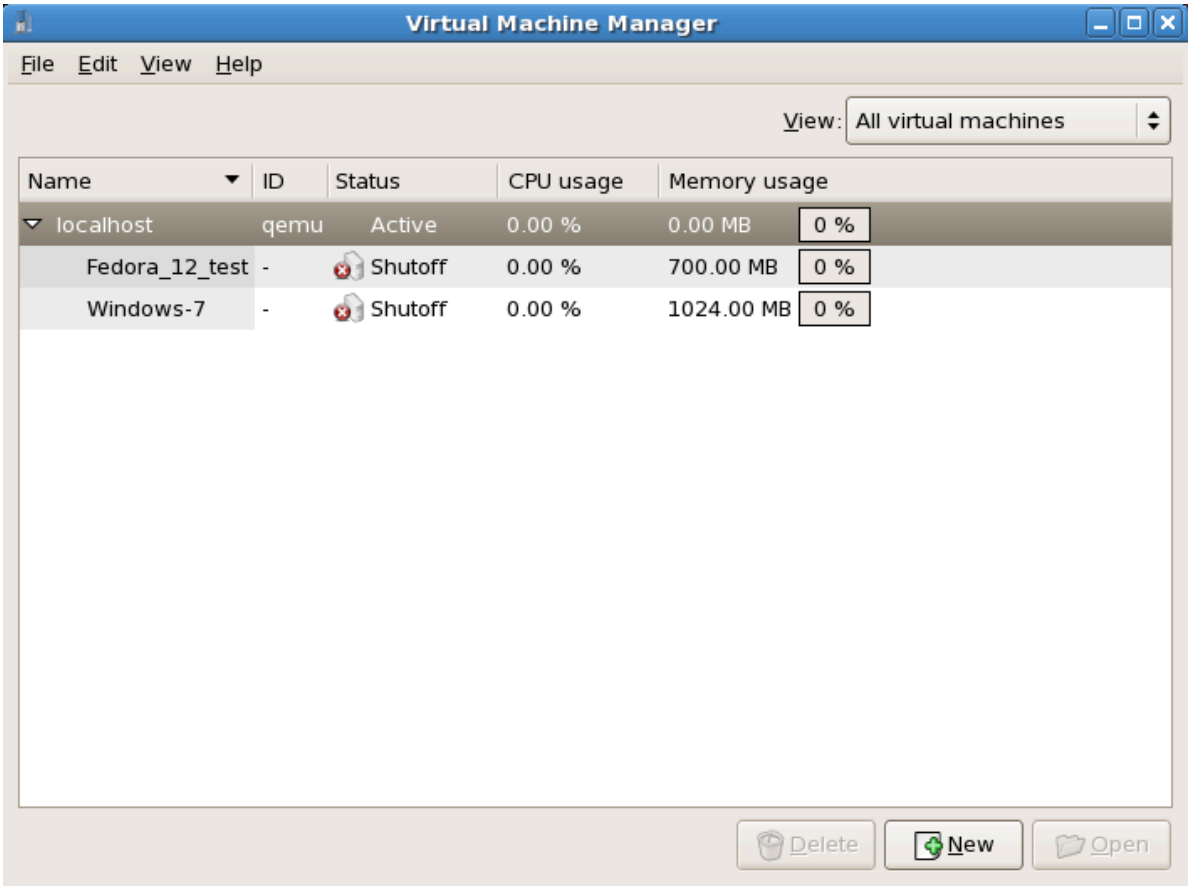


Figure 29.8. Selecting a virtual machine to display

2. From the Virtual Machine Manager **Edit** menu, select **Machine Details** (or click the **Details** button on the bottom of the Virtual Machine Manager main window).

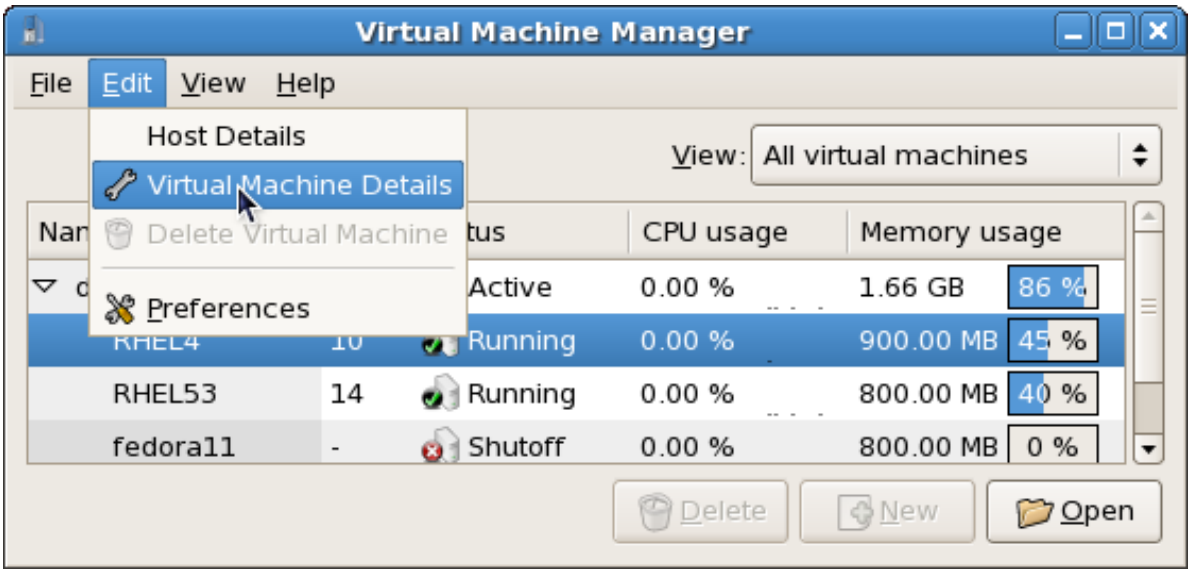


Figure 29.9. Displaying the overview window

On the **Virtual Machine** window, click the **Overview** tab.

The **Overview** tab summarizes CPU and memory usage for the virtualized guest you specified.

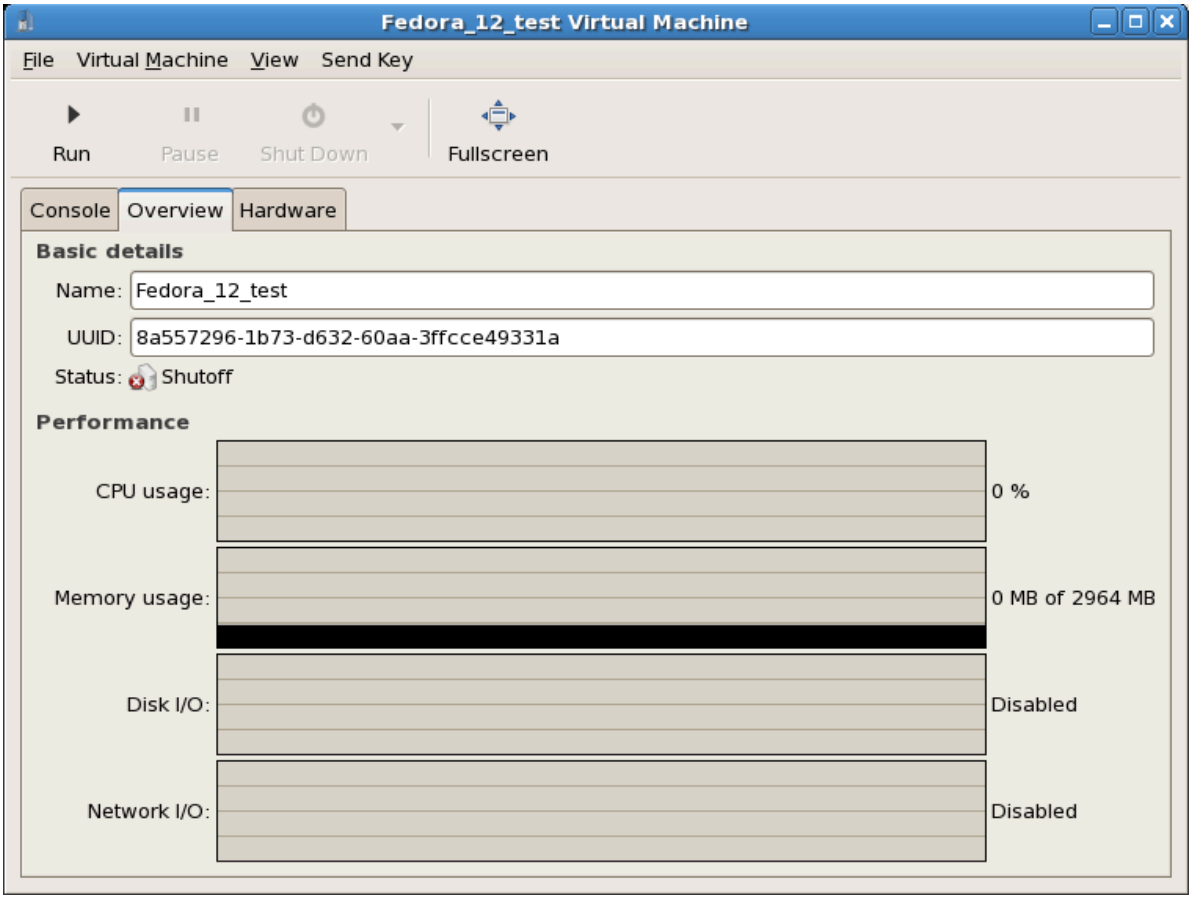


Figure 29.10. Displaying guest details overview

3. On the **Virtual Machine** window, click the **Hardware**tab.

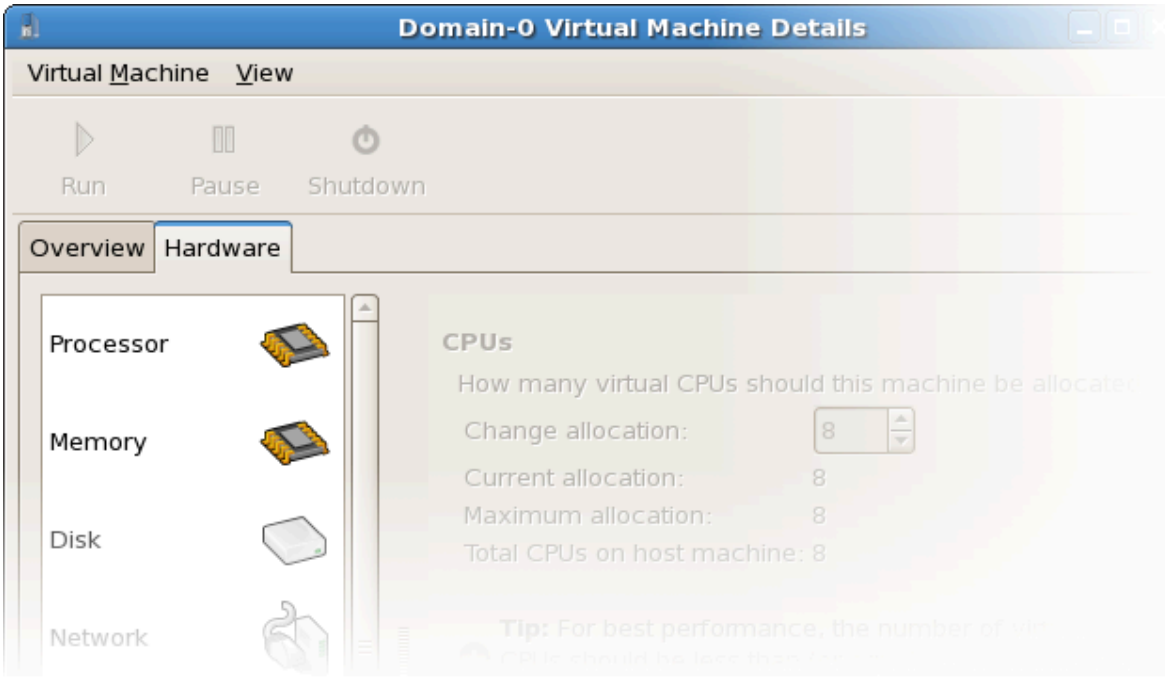


Figure 29.11. Displaying guest hardware details

4. On the **Hardware** tab, click on **Processor** to view or change the current processor allocation.

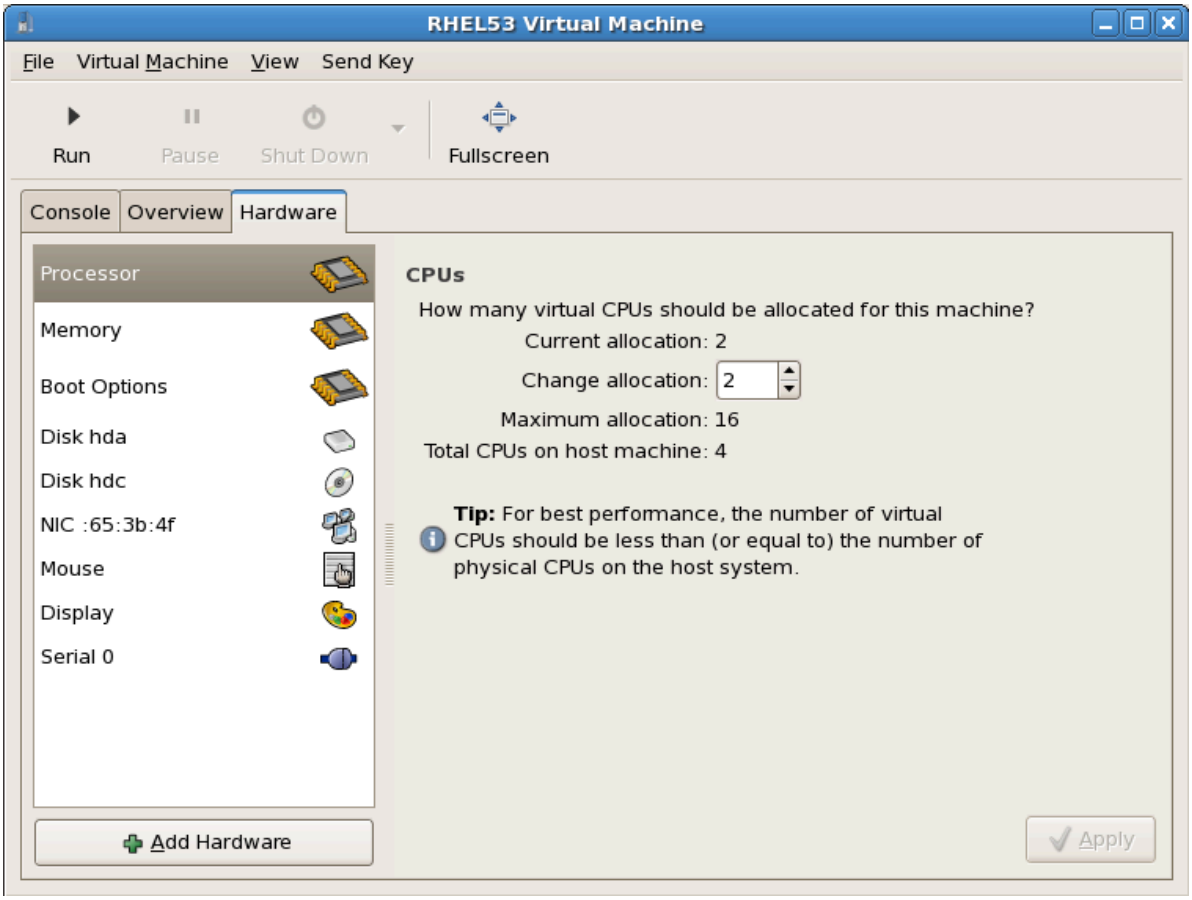


Figure 29.12. Processor allocation panel

5. On the **Hardware** tab, click on **Memory** to view or change the current RAM memory allocation.

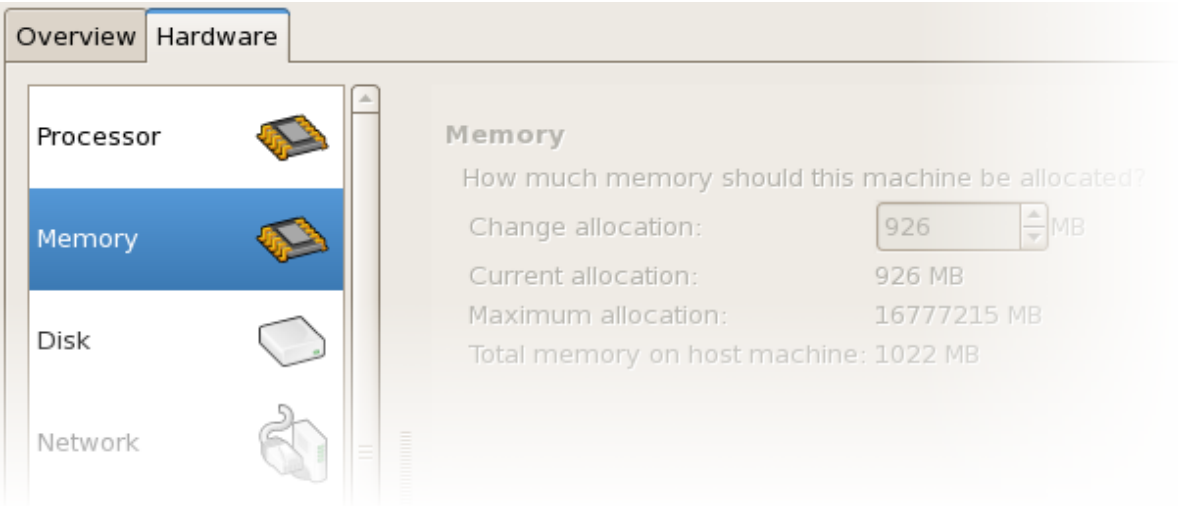


Figure 29.13. Displaying memory allocation

6. On the **Hardware** tab, click on **Disk** to view or change the current hard disk configuration.

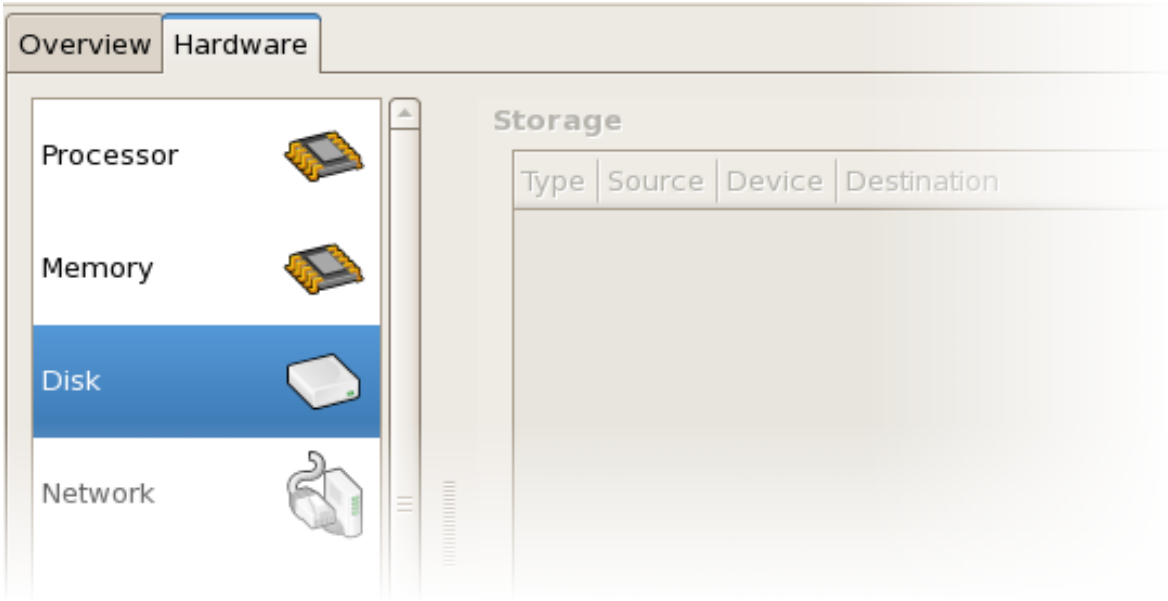


Figure 29.14. Displaying disk configuration

7. On the **Hardware** tab, click on **NIC** to view or change the current network configuration.

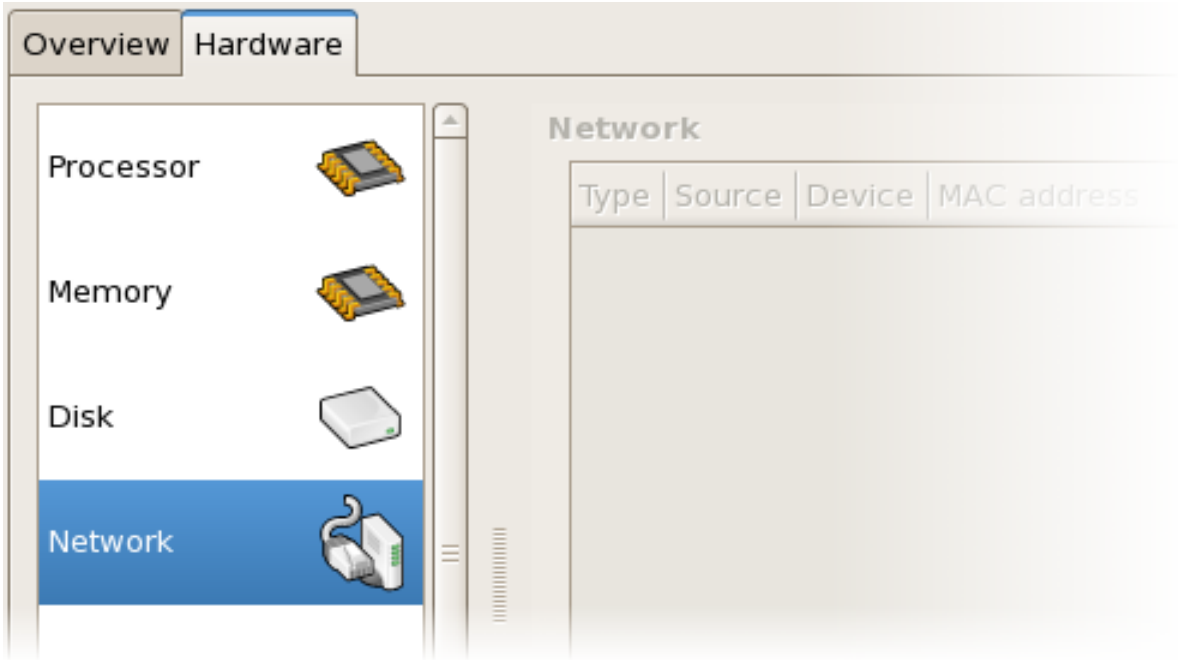


Figure 29.15. Displaying network configuration

29.8. Status monitoring

Status status monitoring preferences can be modified with **`virt-manager`**'s preferences window.

To configure status monitoring:

1. From the **Edit** menu, select **Preferences**.

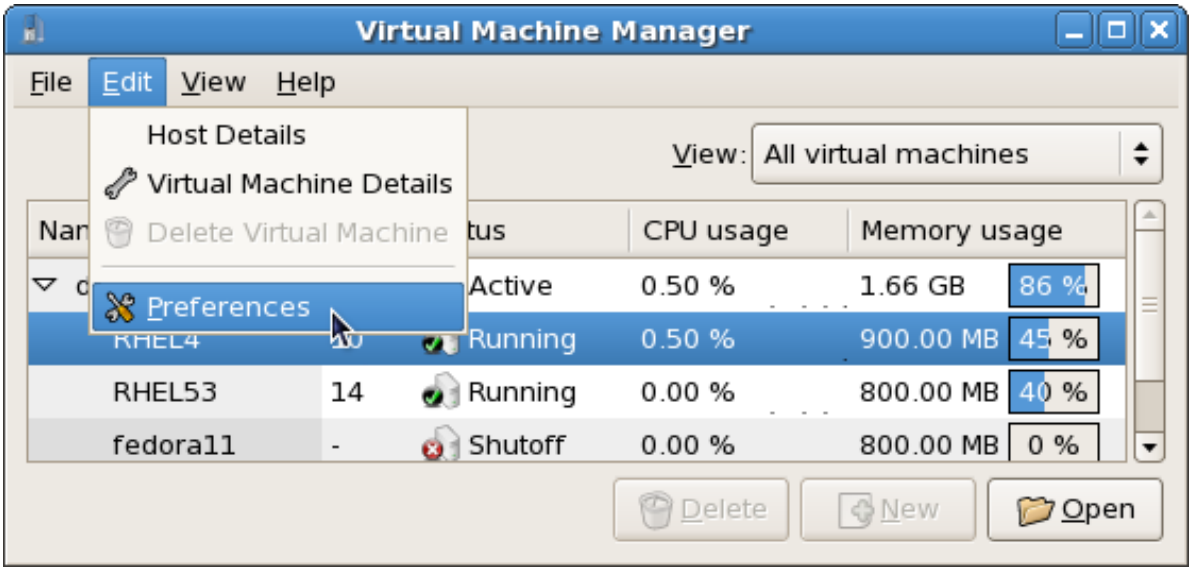


Figure 29.16. Modifying guest preferences

The **Preferences** window appears.

2. From the **Stats** tab specify the time in seconds or stats polling options.

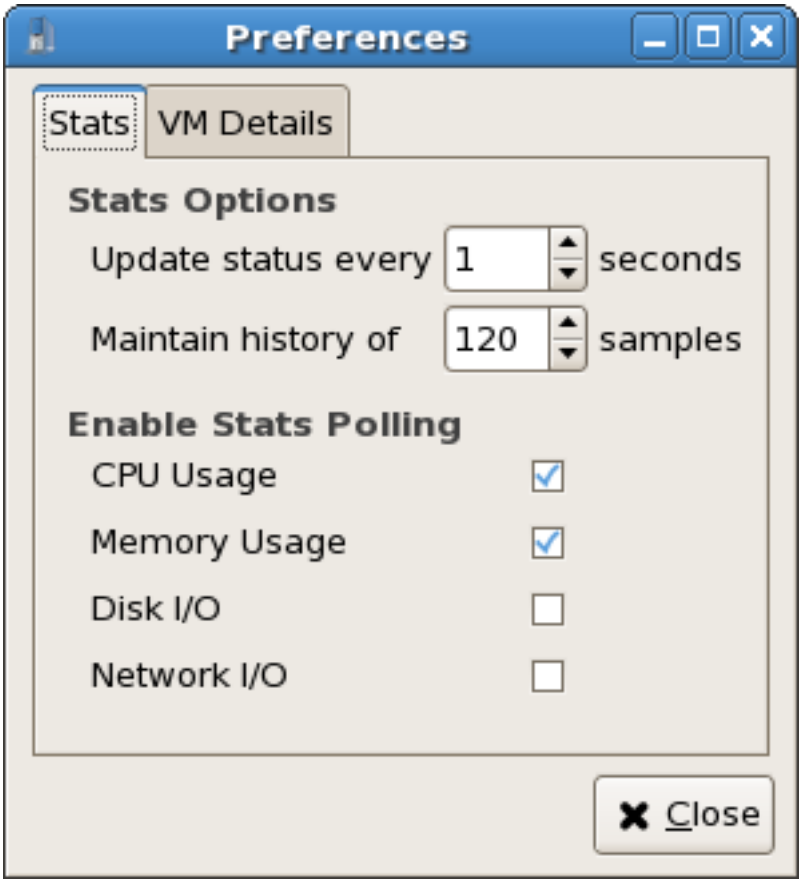


Figure 29.17. Configuring status monitoring

29.9. Displaying guest identifiers

To view the guest IDs for all virtual machines on your system:

1. From the **View** menu, select the **Domain ID** check box.

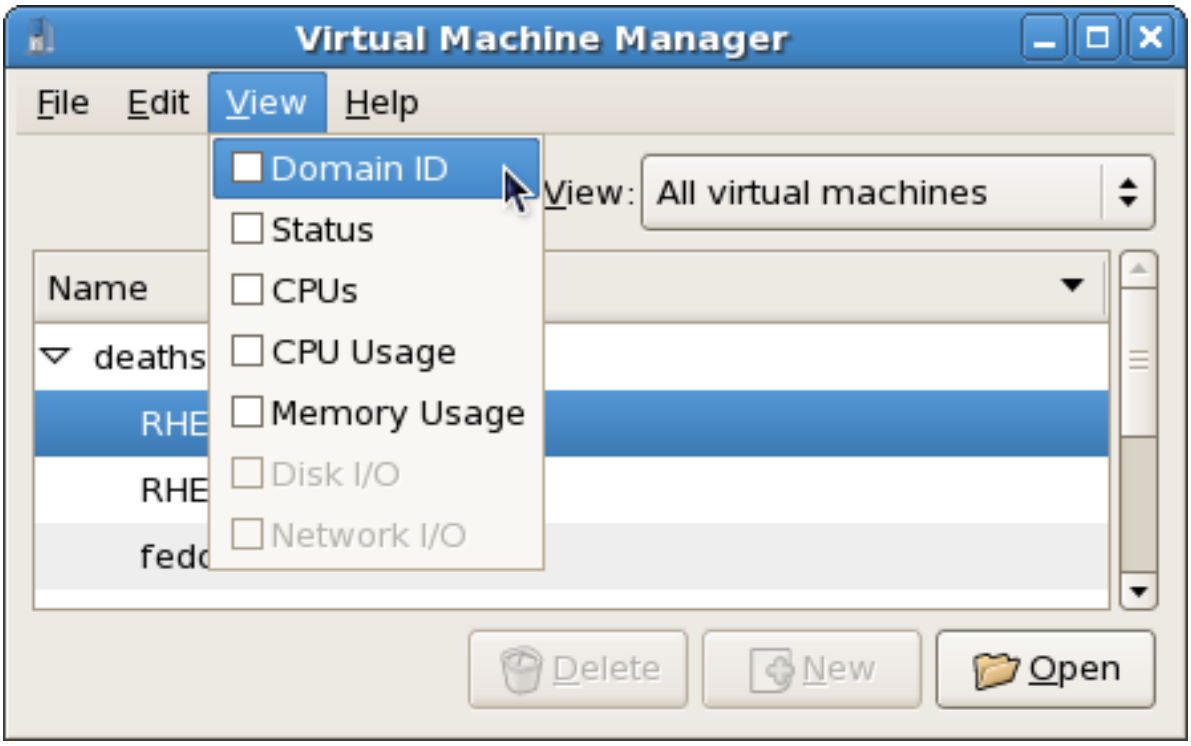


Figure 29.18. Viewing guest IDs

2. The Virtual Machine Manager lists the Domain IDs for all domains on your system.

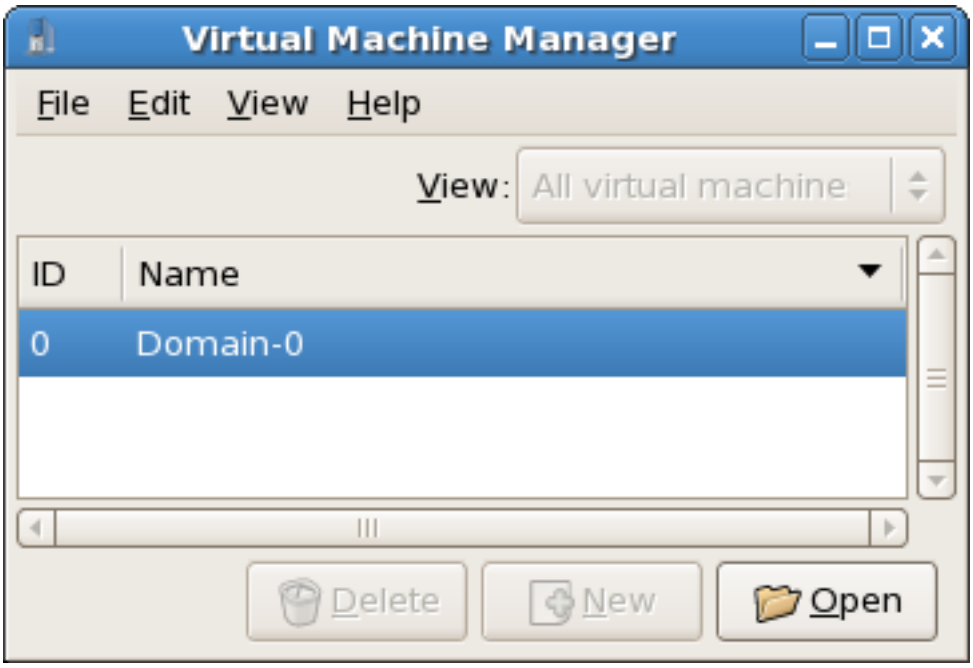


Figure 29.19. Displaying domain IDs

29.10. Displaying a guest's status

To view the status of all virtual machines on your system:

1. From the **View** menu, select the **Status** check box.

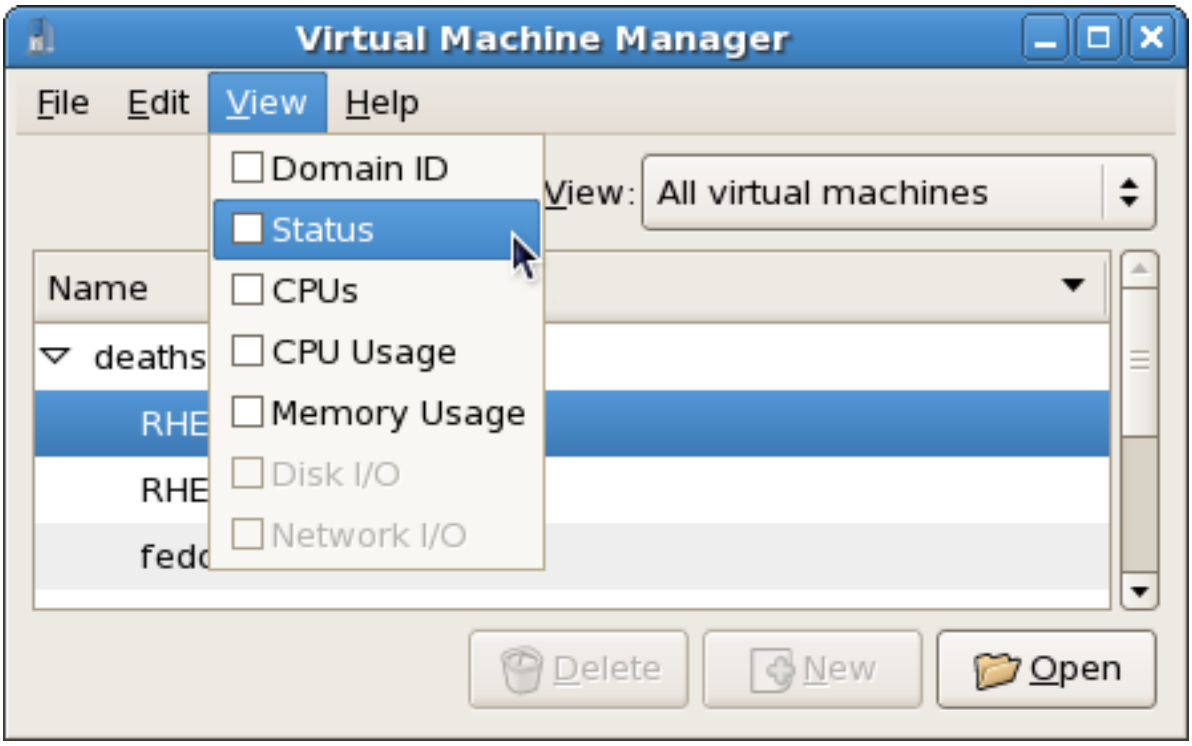


Figure 29.20. Selecting a virtual machine's status

2. The Virtual Machine Manager lists the status of all virtual machines on your system.

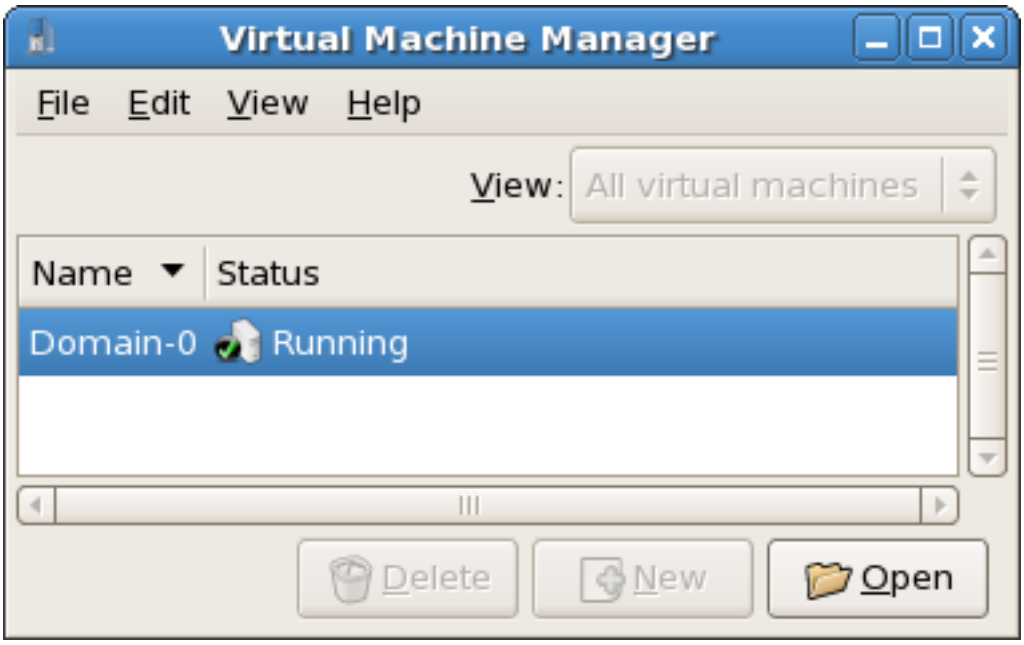


Figure 29.21. Displaying a virtual machine's status

29.11. Displaying virtual CPUs

To view the amount of virtual CPUs for all virtual machines on your system:

1. From the **View** menu, select the **Virtual CPUs** check box.

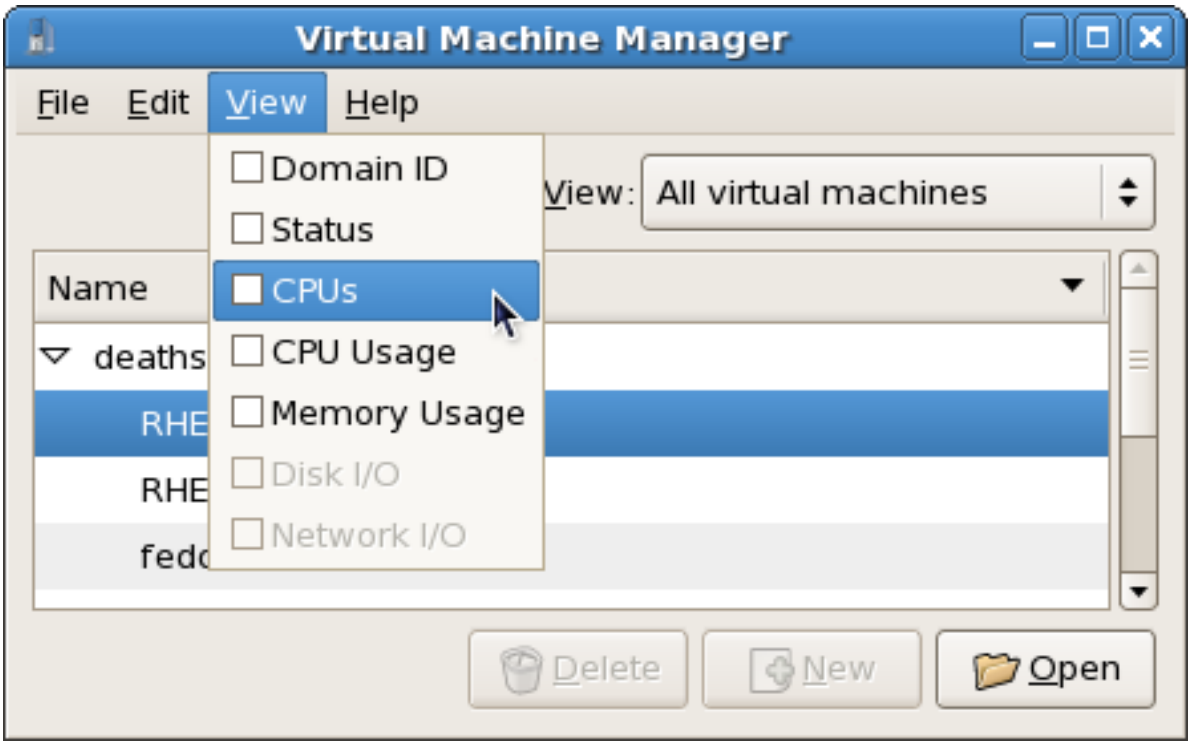


Figure 29.22. Selecting the virtual CPUs option

2. The Virtual Machine Manager lists the Virtual CPUs for all virtual machines on your system.

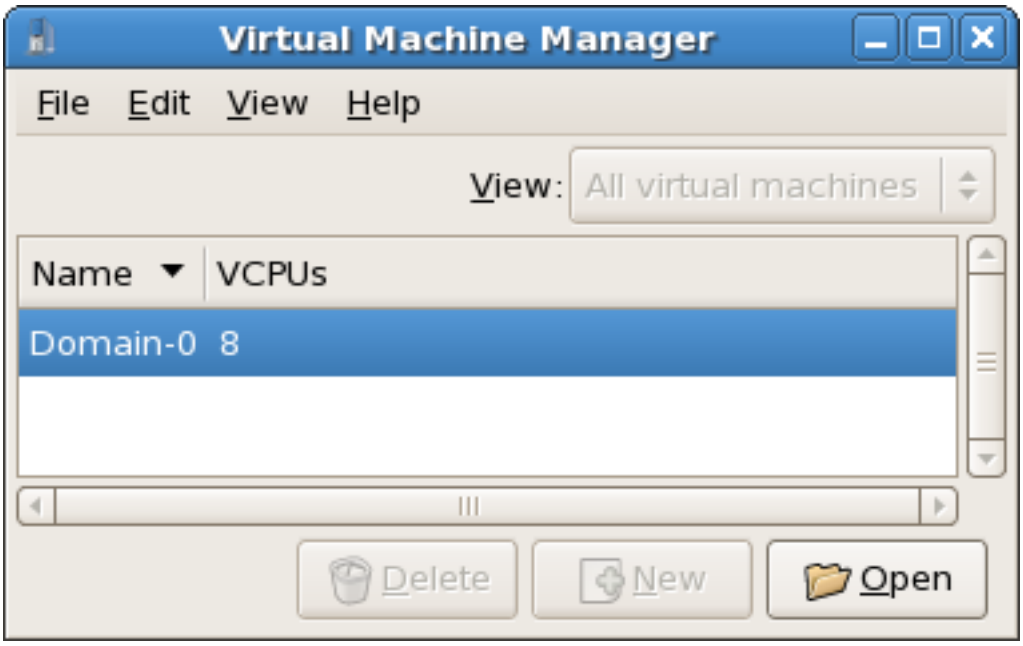


Figure 29.23. Displaying Virtual CPUs

29.12. Displaying CPU usage

To view the CPU usage for all virtual machines on your system:

1. From the **View** menu, select the **CPU Usage** check box.

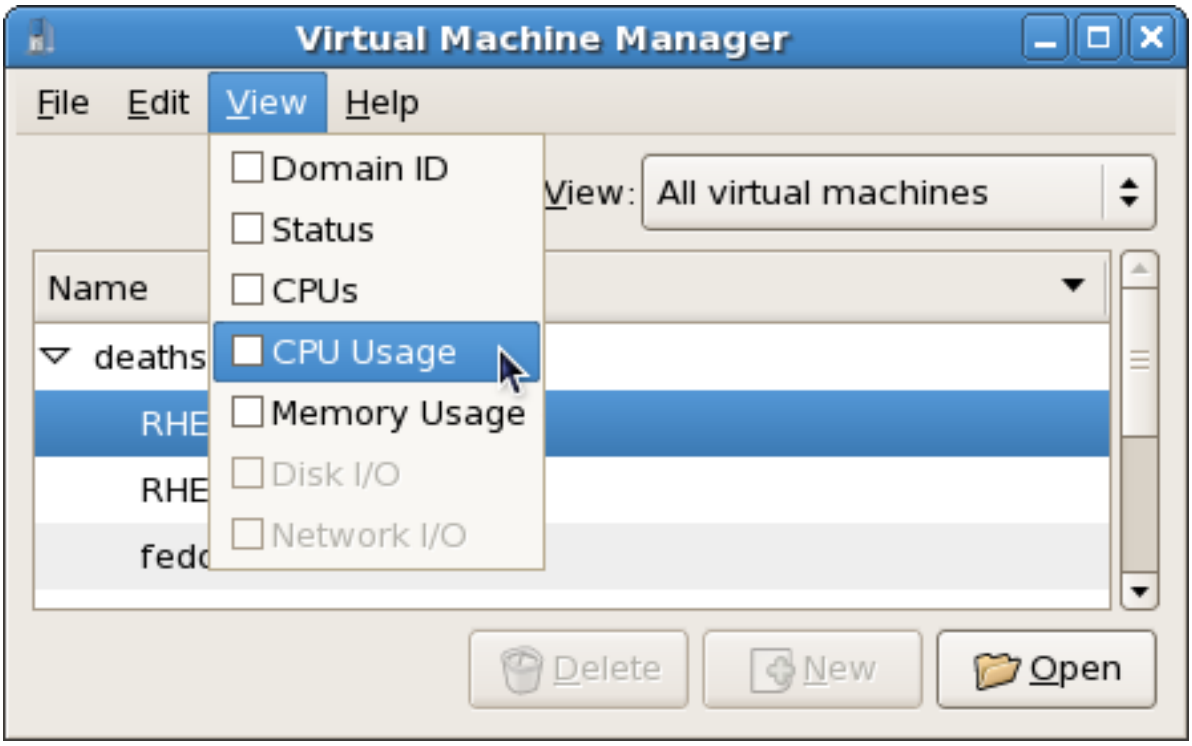


Figure 29.24. Selecting CPU usage

2. The Virtual Machine Manager lists the percentage of CPU in use for all virtual machines on your system.

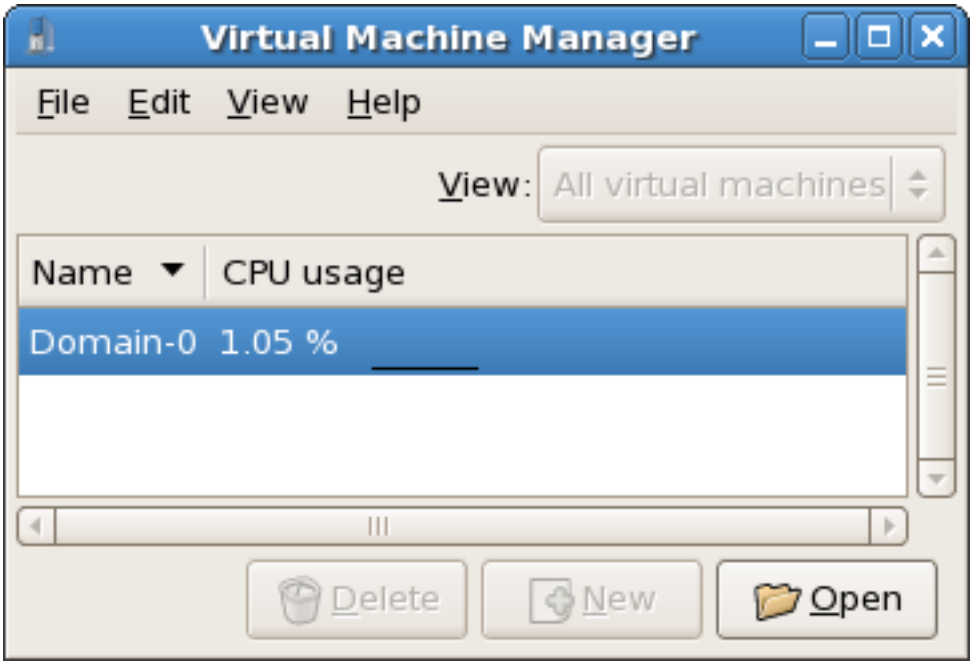


Figure 29.25. Displaying CPU usage

29.13. Displaying memory usage

To view the memory usage for all virtual machines on your system:

1. From the **View** menu, select the **Memory Usage** check box.

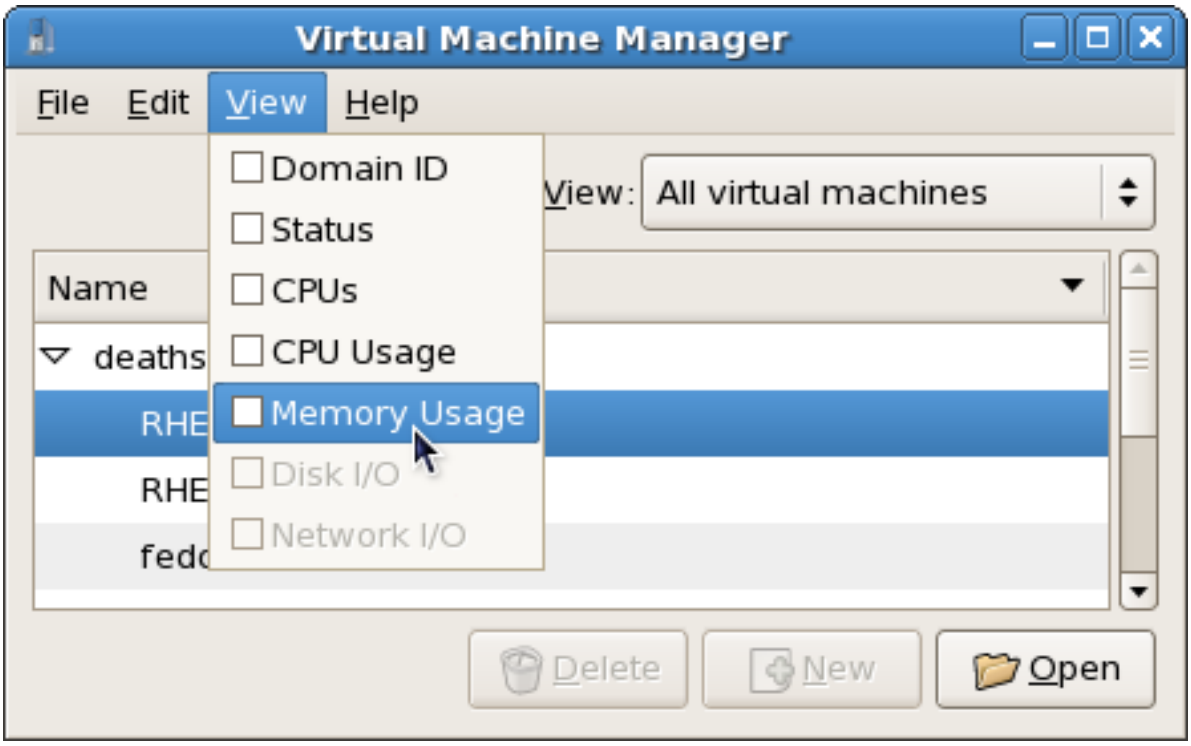


Figure 29.26. Selecting Memory Usage

2. The Virtual Machine Manager lists the percentage of memory in use (in megabytes) for all virtual machines on your system.

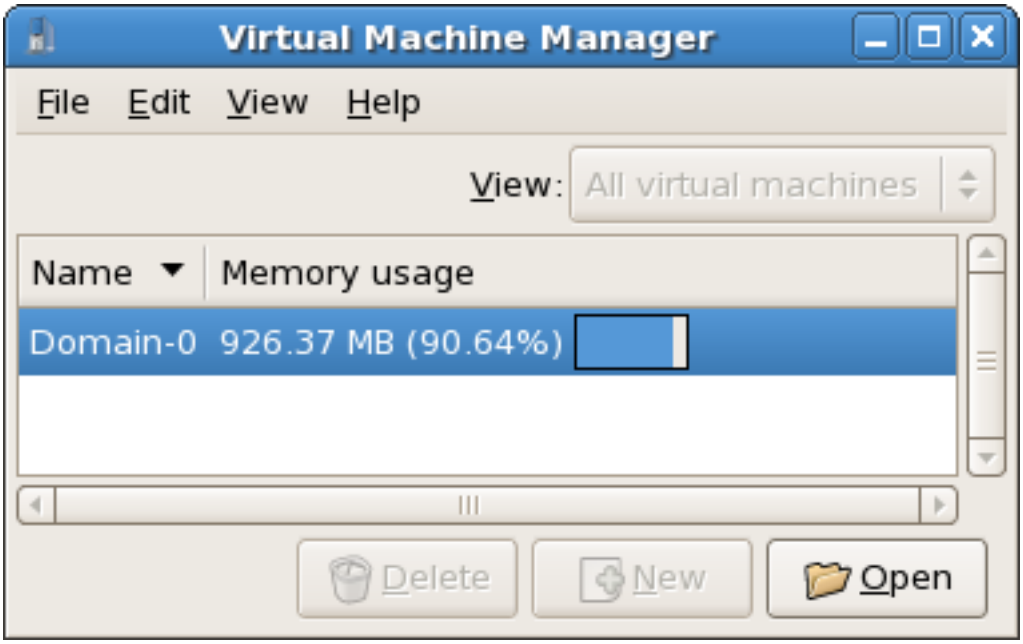


Figure 29.27. Displaying memory usage

29.14. Managing a virtual network

To configure a virtual network on your system:

1. From the **Edit** menu, select **Host Details**.

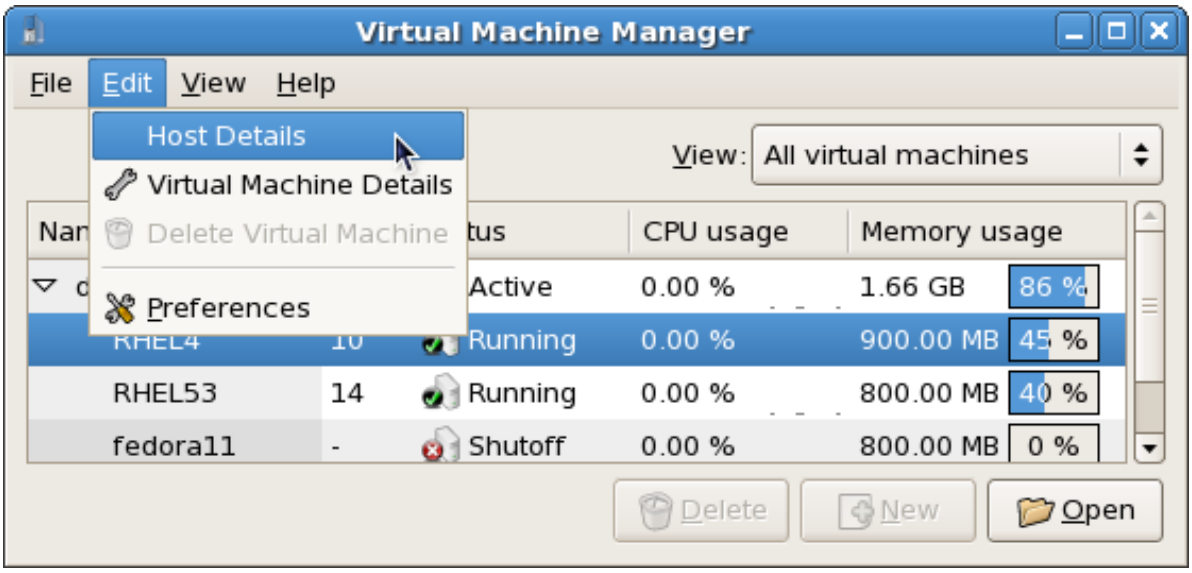


Figure 29.28. Selecting a host's details

2. This will open the **Host Details** menu. Click the **Virtual Networks** tab.

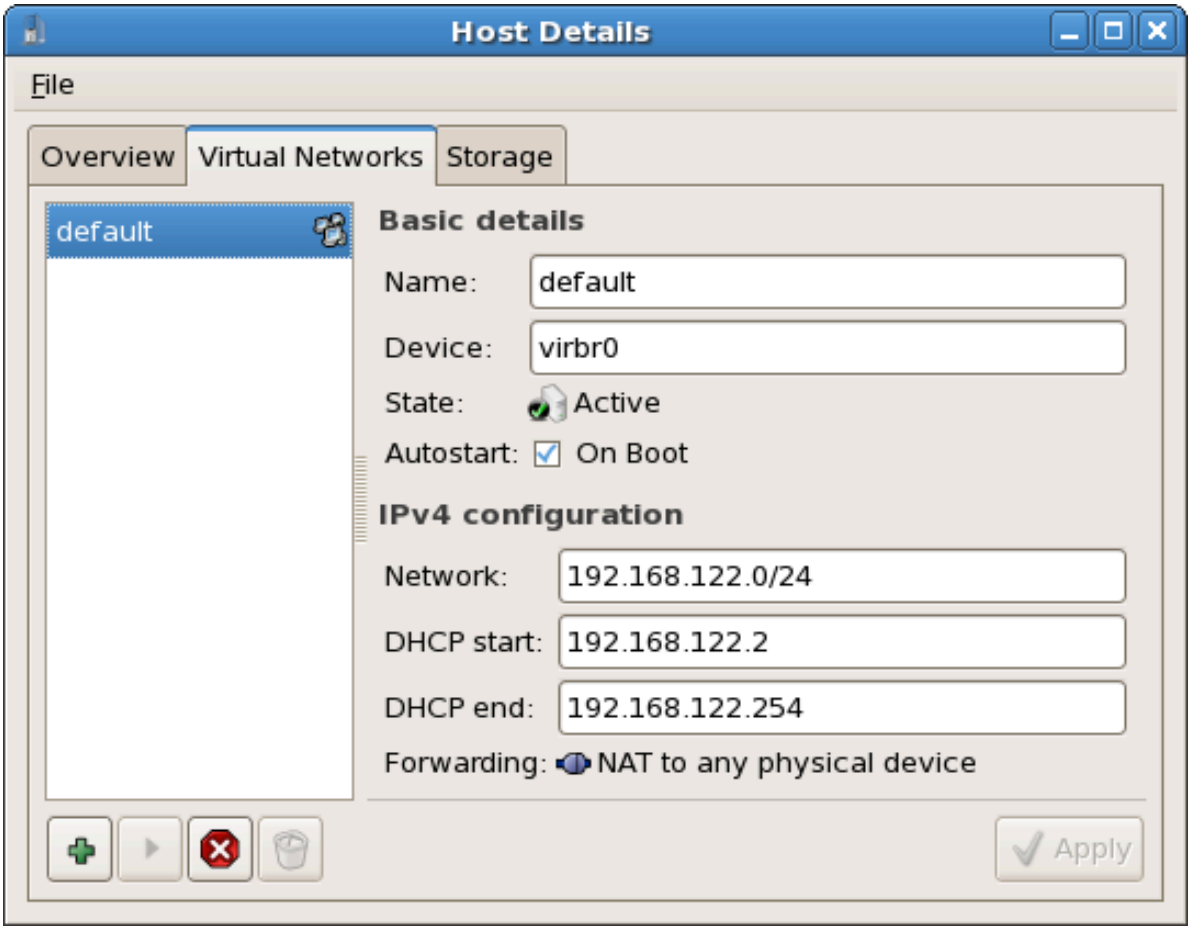


Figure 29.29. Virtual network configuration

3. All available virtual networks are listed on the left-hand box of the menu. You can edit the configuration of a virtual network by selecting it from this box and editing as you see fit.

29.15. Creating a virtual network

To create a virtual network on your system:

1. Open the **Host Details** menu (refer to *Section 29.14, "Managing a virtual network"*) and click the **Add** button.

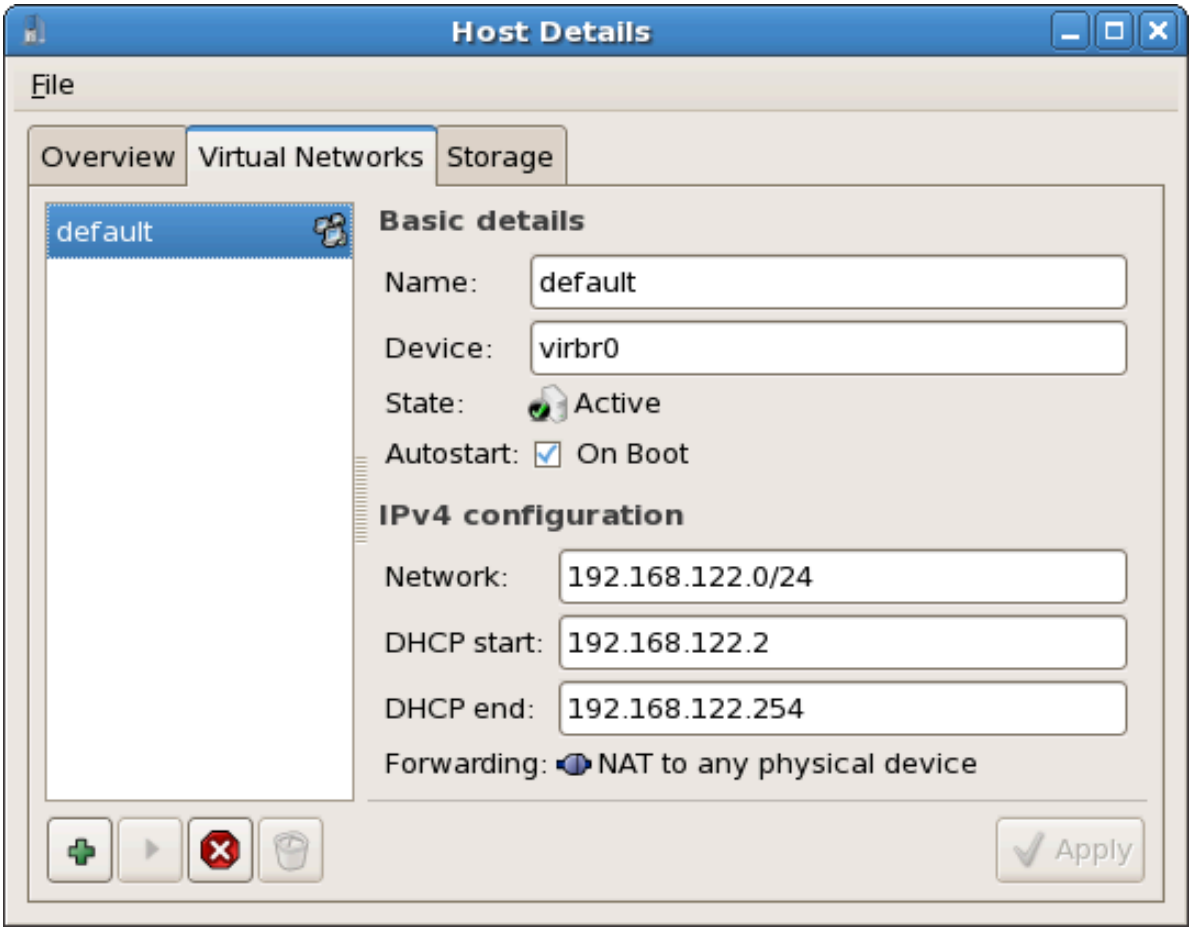


Figure 29.30. Virtual network configuration

This will open the **Create a new virtual network** menu. Click **Forward** to continue.

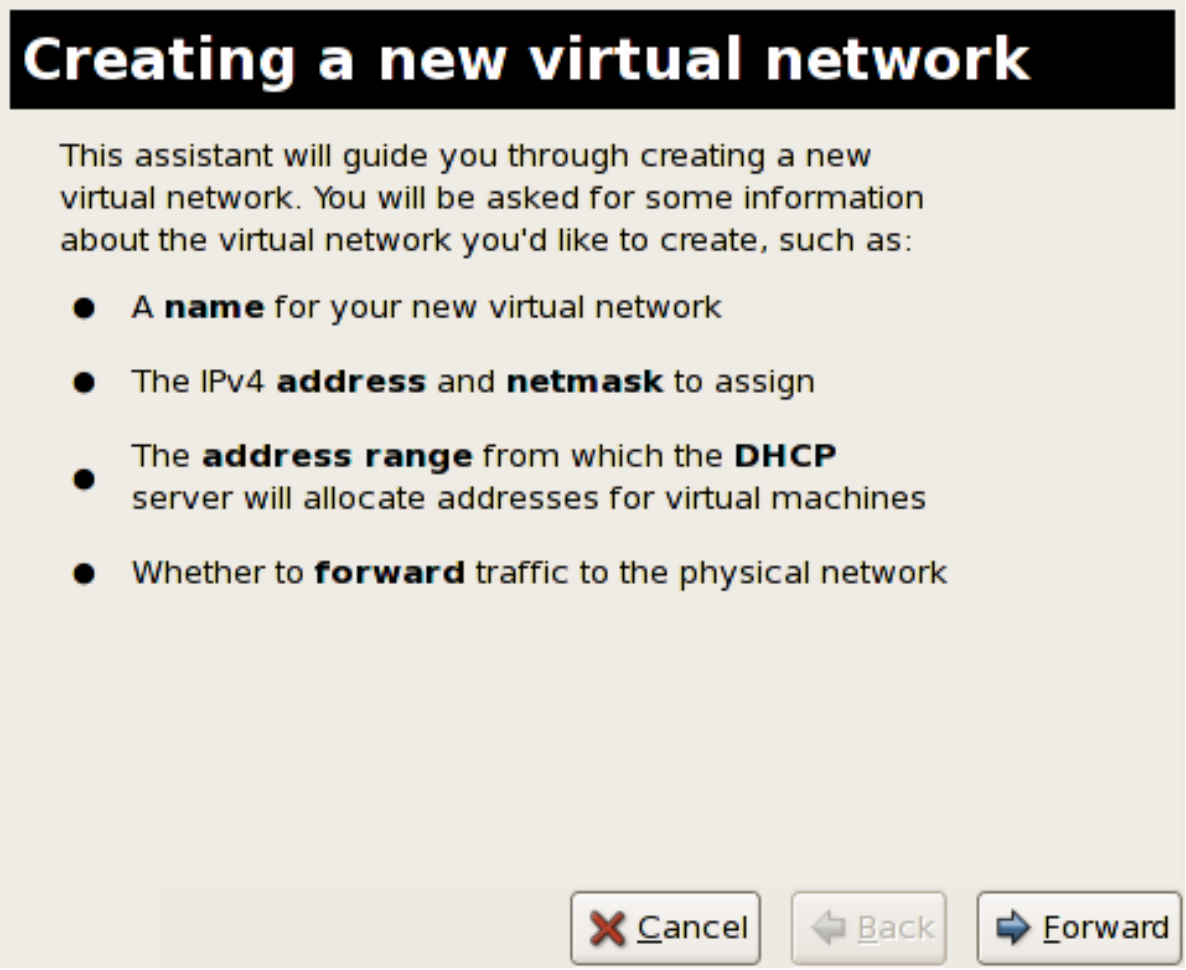


Figure 29.31. Creating a new virtual network

2. Enter an appropriate name for your virtual network and click **Forward**.

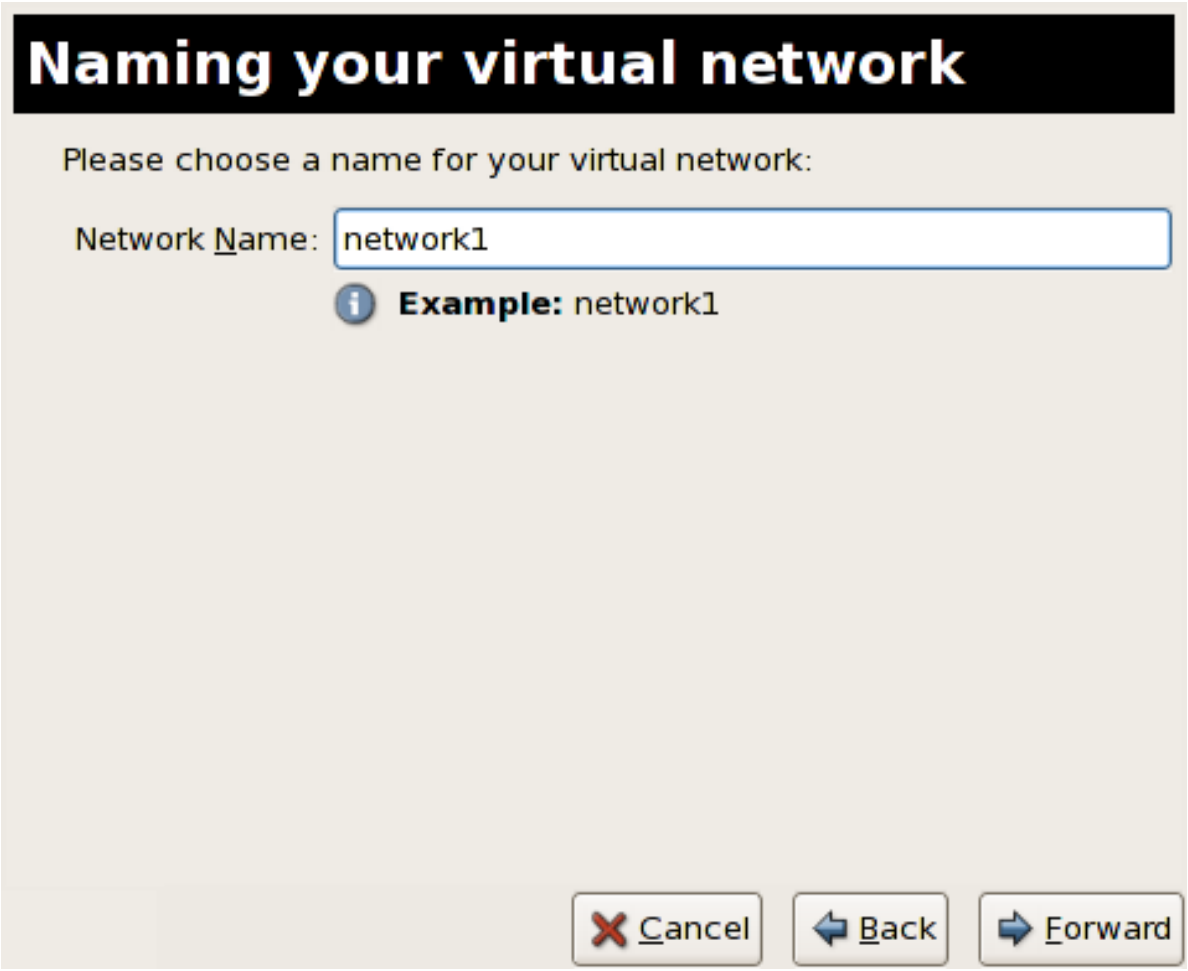


Figure 29.32. Naming your virtual network

3. Enter an IPv4 address space for your virtual network and click **Forward**.

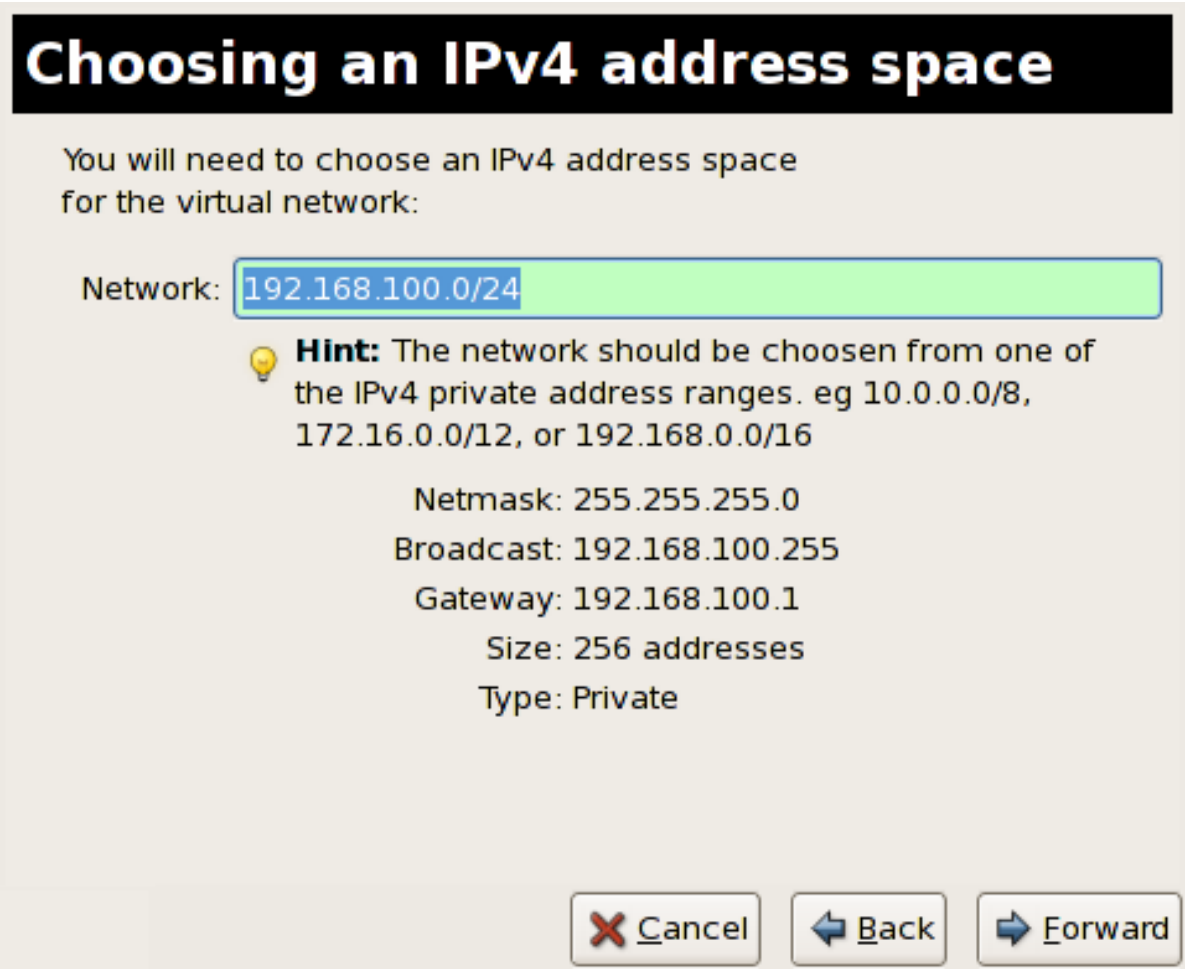


Figure 29.33. Choosing an IPv4 address space

4. Define the DHCP range for your virtual network by specifying a **Start** and **End** range of IP addresses. Click **Forward** to continue.

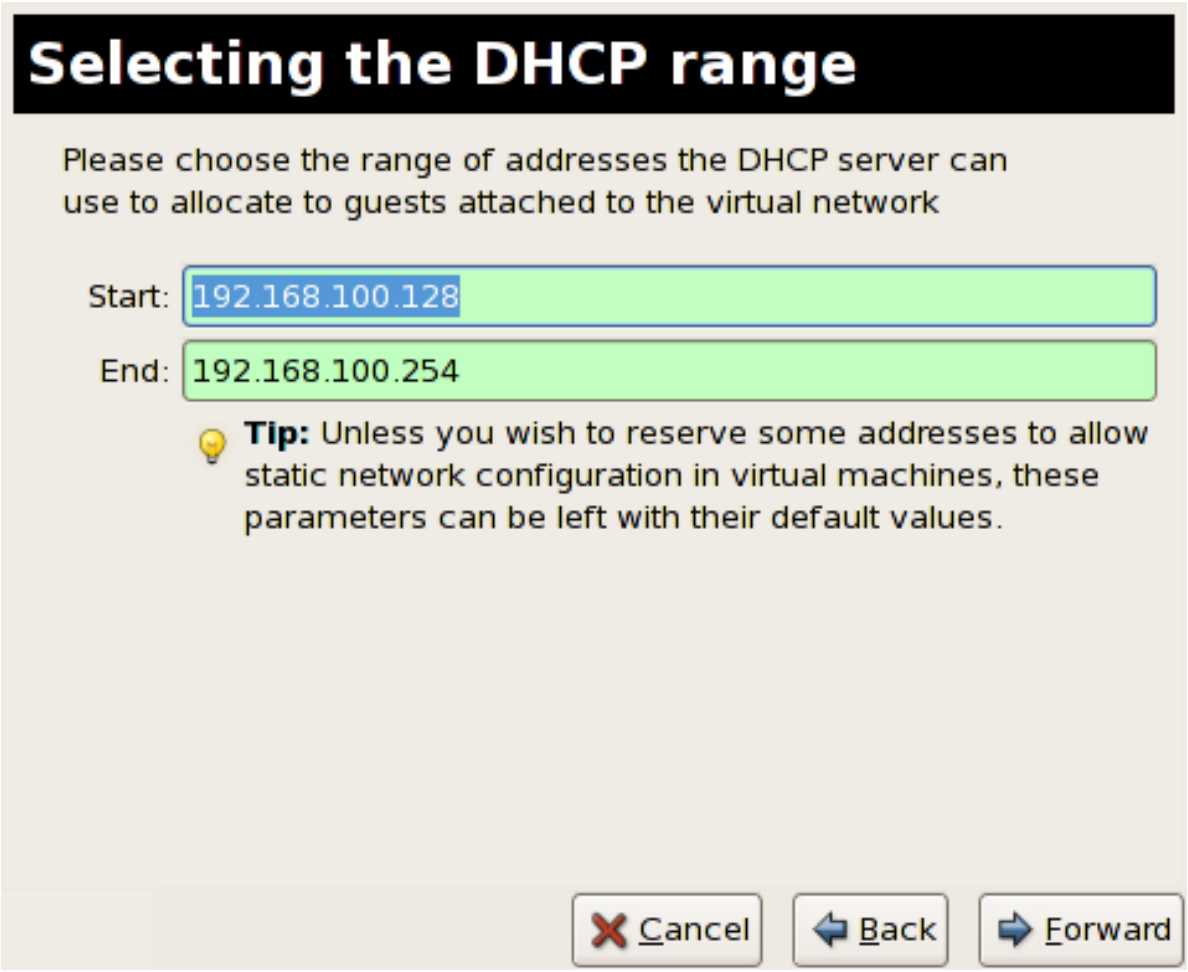


Figure 29.34. Selecting the DHCP range

5. Select how the virtual network should connect to the physical network.

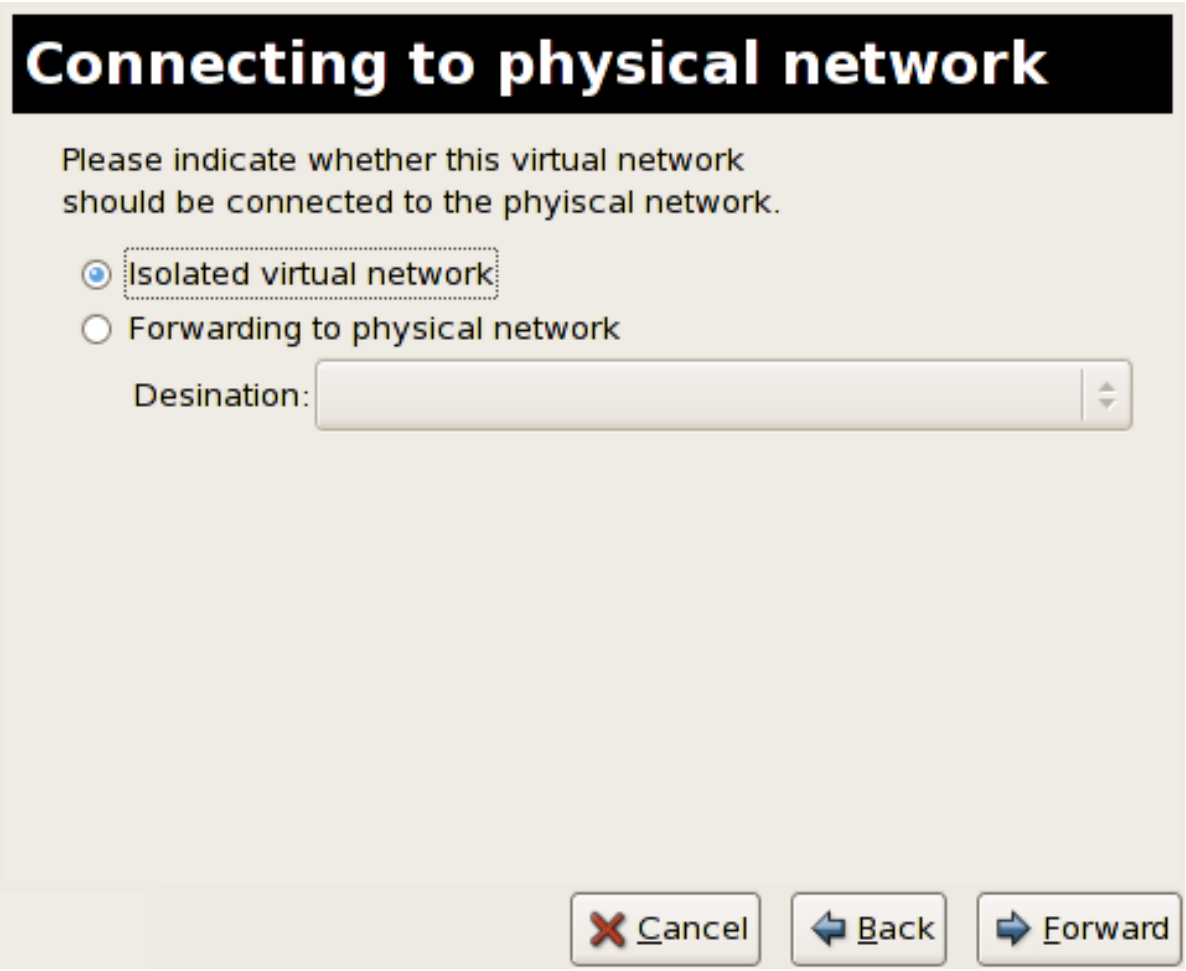


Figure 29.35. Connecting to physical network

If you select **Forwarding to physical network**, choose whether the **Destination** should be **NAT to any physical device** or **NAT to physical device eth0**.

Click **Forward** to continue.

6. You are now ready to create the network. Check the configuration of your network and click **Finish**.

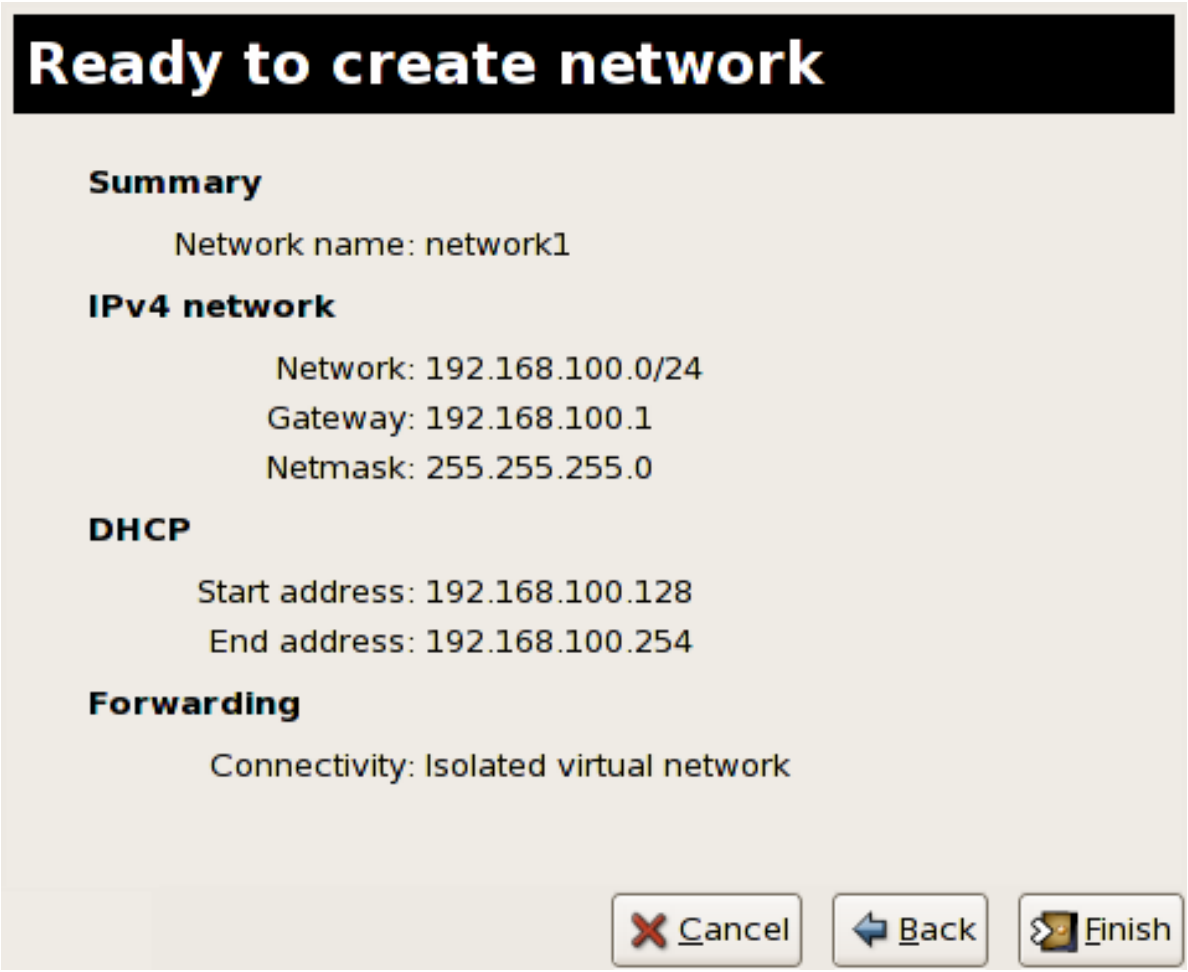


Figure 29.36. Ready to create network

7. The new virtual network is now available in the **Virtual Network** tab of the **Host Details** menu.

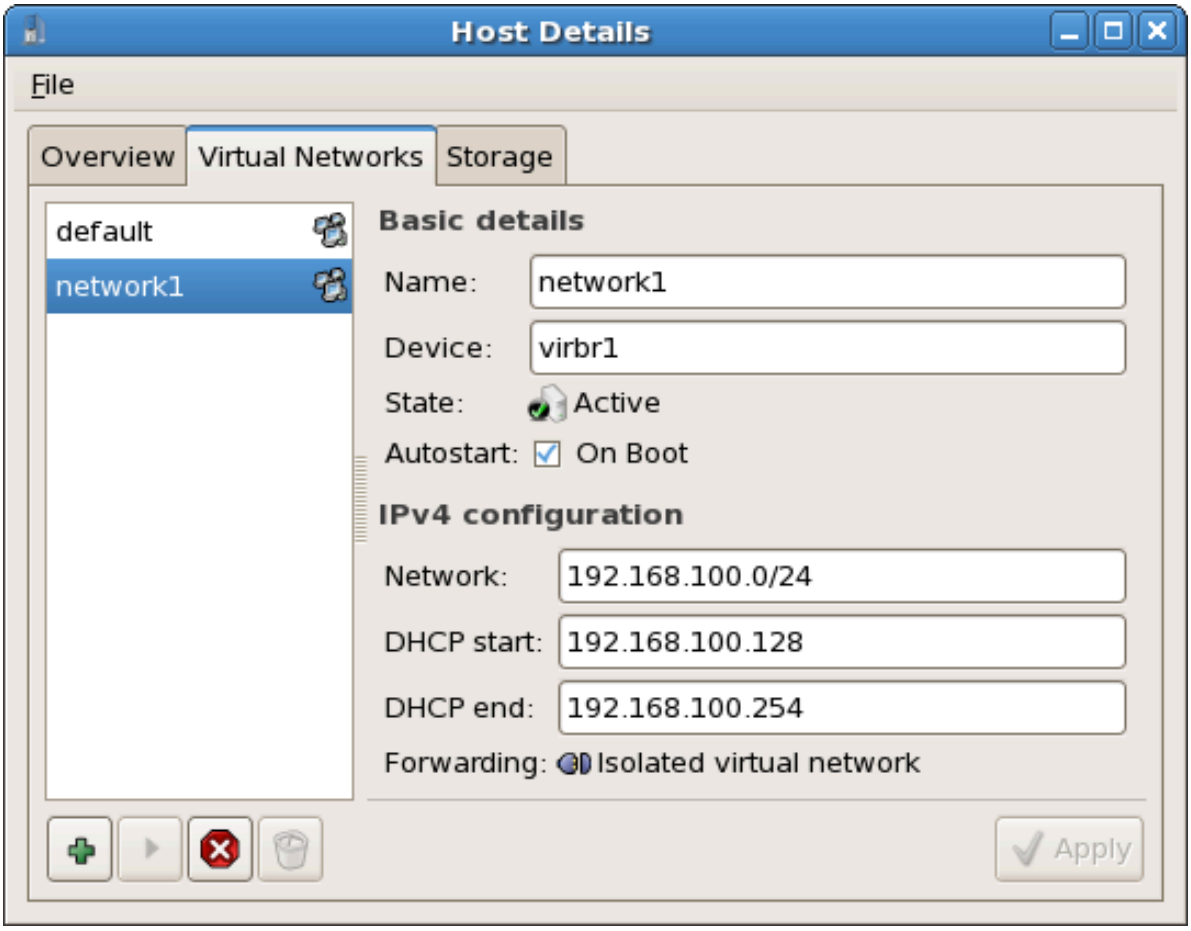


Figure 29.37. New virtual network is now available

libvirt configuration reference

This chapter provides is a references for various parameters of libvirt XML configuration files

Item	Description
pae	Specifies the physical address extension configuration data.
apic	Specifies the advanced programmable interrupt controller configuration data.
memory	Specifies the memory size in megabytes.
vcpus	Specifies the numbers of virtual CPUs.
console	Specifies the port numbers to export the domain consoles to.
nic	Specifies the number of virtual network interfaces.
vif	Lists the randomly-assigned MAC addresses and bridges assigned to use for the domain's network addresses.
disk	Lists the block devices to export to the domain and exports physical devices to domain with read only access.
dhcp	Enables networking using DHCP.
netmask	Specifies the configured IP netmasks.
gateway	Specifies the configured IP gateways.
acpi	Specifies the advanced configuration power interface configuration data.

Table 30.1. libvirt configuration files

Creating custom libvirt scripts

This section provides some information which may be useful to programmers and system administrators intending to write custom scripts to make their lives easier by using **libvirt**.

Chapter 25, Miscellaneous administration tasks is recommended reading for programmers thinking of writing new applications which use **libvirt**.

31.1. Using XML configuration files with virsh

virsh can handle XML configuration files. You may want to use this to your advantage for scripting large deployments with special options. You can add devices defined in an XML file to a running para-virtualized guest. For example, to add a ISO file as **hdc** to a running guest create an XML file:

```
# cat satelliteiso.xml
<disk type="file" device="disk">
 <driver name="file"/>
 <source file="/var/lib/libvirt/images/rhn-satellite-5.0.1-11-redhat-linux-as-i386-4-embedded-
oracle.iso"/>
 <target dev="hdc"/>
 <readonly/>
</disk>
```

Run **virsh attach-device** to attach the ISO as **hdc** to a guest called "satellite" :

```
# virsh attach-device satellite satelliteiso.xml
```

Part VII. Troubleshooting

Introduction to troubleshooting and problem solving

The following chapters provide information to assist you in troubleshooting issues you may encounter using virtualization.

Troubleshooting

This chapter covers common problems and solutions with Fedora virtualization.

This chapter is to give you, the reader, a background to identify where problems with virtualization technologies are. Troubleshooting takes practice and experience which are difficult to learn from a book. It is recommended that you experiment and test virtualization on Fedora to develop your troubleshooting skills.

If you cannot find the answer in this document there may be an answer online from the virtualization community. Refer to *Section A.1, "Online resources"* for a list of Linux virtualization websites.

32.1. Debugging and troubleshooting tools

This section summarizes the System Administrator applications, the networking utilities, and debugging tools. You can employ these standard System administration tools and logs to assist with troubleshooting:

- **`vmstat`**
- **`iostat`**
- **`lsof`**
- **`systemtap`**
- **`crash`**
- **`sysrq`**
- **`sysrq t`**
- **`sysrq w`**

These networking tools can assist with troubleshooting virtualization networking problems:

- **`ifconfig`**
- **`tcpdump`**

  The **`tcpdump`** command 'sniffs' network packets. **`tcpdump`** is useful for finding network abnormalities and problems with network authentication. There is a graphical version of **`tcpdump`** named **`wireshark`**.

- **`brctl`**

  `brctl` is a networking tool that inspects and configures the Ethernet bridge configuration in the Virtualization linux kernel. You must have root access before performing these example commands:

```
# brctl show
bridge-name    bridge-id          STP  enabled  interfaces
-----------------------------------------------------------------------------
virtbr0             8000.feffffff       yes        eth0

# brctl showmacs virtbr0
port-no           mac-addr                  local?       aging timer
```

```
1                 fe:ff:ff:ff:ff:           yes            0.00
2                 fe:ff:ff:fe:ff:           yes            0.00
# brctl showstp virtbr0
virtbr0
bridge-id              8000.fefffffffff
designated-root        8000.fefffffffff
root-port              0                    path-cost             0
max-age                20.00                bridge-max-age        20.00
hello-time             2.00                 bridge-hello-time     2.00
forward-delay          0.00                 bridge-forward-delay  0.00
aging-time            300.01
hello-timer            1.43                 tcn-timer             0.00
topology-change-timer  0.00                 gc-timer              0.02
```

Listed below are some other useful commands for troubleshooting virtualization.

- **strace** is a command which traces system calls and events received and used by another process.
- **vncviewer**: connect to a VNC server running on your server or a virtual machine. Install **vncviwer** using the **`yum install vnc`** command.
- **vncserver**: start a remote desktop on your server. Gives you the ability to run graphical user interfaces such as virt-manager via a remote session. Install **vncserver** using the **`yum install vnc-server`** command.

32.2. Log files

KVM uses various log files. All the log files are standard ASCII files, and accessible with a text editor.

- The default directory for all file-based images is the **`/var/lib/libvirt/images`** directory.
- **`qemu-kvm.[PID].log`** is the log file created by the **`qemu-kvm`** process for each fully virtualized guest. When using this log file, you must retrieve the given **`qemu-kvm`** process PID, by using the **`ps`** command to examine process arguments to isolate the **`qemu-kvm`** process on the virtual machine. Note that you must replace the [PID] symbol with the actual PID **`qemu-kvm`** process.

If you encounter any errors with the Virtual Machine Manager, you can review the generated data in the **`virt-manager.log`** file that resides in the **`/.virt-manager`** directory. Note that every time you start the Virtual Machine Manager, it overwrites the existing log file contents. Make sure to backup the **`virt-manager.log`** file, before you restart the Virtual Machine manager after a system error.

32.3. Troubleshooting with serial consoles

Linux kernels can output information to serial ports. This is useful for debugging kernel panics and hardware issues with video devices or headless servers. The subsections in this section cover setting up serial console output for machines running Fedora virtualization kernels and their virtualized guests.

This section covers how to enable serial console output for fully virtualized guests.

Fully virtualized guest serial console output can be viewed with the **`virsh console`** command.

Be aware fully virtualized guest serial consoles have some limitations. Present limitations include:

- output data may be dropped or scrambled.

The serial port is called `ttyS0` on Linux or `COM1` on Windows.

You must configure the virtualized operating system to output information to the virtual serial port.

To output kernel information from a fully virtualized Linux guest into the domain modify the **`/boot/grub/grub.conf`** file by inserting the line *`console=tty0 console=ttys0,115200`*.

```
title Fedora Server (2.6.18-92.el5)
 root (hd0,0)
 kernel /vmlinuz-2.6.18-92.el5 ro root=/dev/volgroup00/logvol00
 console=tty0 console=ttys0,115200
 initrd /initrd-2.6.18-92.el5.img
```

Reboot the guest.

On the host, access the serial console with the following command:

```
# virsh console
```

You can also use **`virt-manager`** to display the virtual text console. In the guest console window, select **Serial Console** from the **View** menu.

32.4. Virtualization log files

- /var/log/libvirt/qemu/*`GuestName`*.log

If you encounter any errors with the Virtual Machine Manager, you can review the generated data in the **`virt-manager.log`** file that resides in the **`/.virt-manager`** directory. Note that every time you start the Virtual Machine Manager, it overwrites the existing log file contents. Make sure to backup the **`virt-manager.log`** file, before you restart the Virtual Machine manager after a system error.

32.5. Loop device errors

If file-based guest images are used you may have to increase the number of configured loop devices. The default configuration allows up to eight active loop devices. If more than eight file-based guests or loop devices are needed the number of loop devices configured can be adjusted in **`/etc/modprobe.conf`**. Edit **`/etc/modprobe.conf`** and add the following line to it:

```
options loop max_loop=64
```

This example uses 64 but you can specify another number to set the maximum loop value. You may also have to implement loop device backed guests on your system. To employ loop device backed guests for a para-virtualized guest, use the **`phy: block device`** or **`tap:aio`** commands. To employ loop device backed guests for a full virtualized system, use the **`phy: device`** or **`file: file`** commands.

32.6. Enabling Intel VT and AMD-V virtualization hardware extensions in BIOS

This section describes how to identify hardware virtualization extensions and enable them in your BIOS if they are disabled.

The Intel VT extensions can be disabled in the BIOS. Certain laptop vendors have disabled the Intel VT extensions by default in their CPUs.

The virtualization extensions can not be disabled in the BIOS for AMD-V.

The virtualization extensions are sometimes disabled in BIOS, usually by laptop manufacturers. Refer to *Section 32.6, "Enabling Intel VT and AMD-V virtualization hardware extensions in BIOS"* for instructions on enabling disabled virtualization extensions.

Verify the virtualization extensions are enabled in BIOS. The BIOS settings for Intel® VT or AMD-V are usually in the **Chipset** or **Processor** menus. The menu names may vary from this guide, the virtualization extension settings may be found in **Security Settings** or other non standard menu names.

Procedure 32.1. Enabling virtualization extensions in BIOS

1. Reboot the computer and open the system's BIOS menu. This can usually be done by pressing the **delete** key, the **`F1`** key or **`Alt`** and **`F4`** keys depending on the system.

2. Select **Restore Defaults** or **Restore Optimized Defaults**, and then select **Save & Exit**.

3. Power off the machine and disconnect the power supply.

4. **Enabling the virtualization extensions in BIOS**

   Note: BIOS steps
   Many of the steps below may vary depending on your motherboard, processor type, chipset and OEM. Refer to your system's accompanying documentation for the correct information on configuring your system.

   a. Power on the machine and open the BIOS (as per Step 1).

   b. Open the **Processor** submenu The processor settings menu may be hidden in the **Chipset**, **Advanced CPU Configuration** or **Northbridge**.

   c. Enable **Intel Virtualization Technology** (also known as Intel VT) or **AMD-V** depending on the brand of the processor. The virtualization extensions may be labeled **Virtualization Extensions**, **Vanderpool** or various other names depending on the OEM and system BIOS.

   d. Enable Intel VTd or AMD IOMMU, if the options are available. Intel VTd and AMD IOMMU are used for *PCI passthrough*.

   e. Select **Save & Exit**.

5. Power off the machine and disconnect the power supply.

6. Run **`cat /proc/cpuinfo | grep vmx svm`**. If the command outputs, the virtualization extensions are now enabled. If there is no output your system may not have the virtualization extensions or the correct BIOS setting enabled.

32.7. KVM networking performance

By default, KVM virtual machines are assigned a virtual Realtek 8139 (rtl8139) NIC (network interface controller).

The rtl8139 virtualized NIC works fine in most environments. However, this device can suffer from performance degradation problems on some networks, for example, a 10 Gigabit Ethernet network.

A workaround is to switch to a different type of virtualized NIC. For example, Intel PRO/1000 (e1000) or virtio (the para-virtualized network driver).

To switch to the e1000 driver:

1. Shutdown the guest operating system.
2. Edit the guest's configuration file with the **`virsh`** command (where *`GUEST`* is the guest's name):

   ```
   # virsh edit GUEST
   ```

   The **`virsh edit`** command uses the **`$EDITOR`** shell variable to determine which editor to use.
3. Find the network interface section of the configuration. This section resembles the snippet below:

   ```
   <interface type='network'>
     [output truncated]
     <model type='rtl8139' />
   </interface>
   ```
4. Change the type attribute of the model element from *`'rtl8139'`* to *`'e1000'`*. This will change the driver from the rtl8139 driver to the e1000 driver.

   ```
   <interface type='network'>
     [output truncated]
     <model type='e1000' />
   </interface>
   ```
5. Save the changes and exit the text editor
6. Restart the guest operating system.

Alternatively, you can install new virtualized guests with a different network driver. This may be required if you are having difficulty installing guests over a network connection. This method requires you to have at least one virtual machine already created (possibly installed from CD or DVD) to use as a template.

1. Create an XML template from an existing virtual machine:

   ```
   # virsh dumpxml GUEST > /tmp/guest.xml
   ```
2. Copy and edit the XML file and update the unique fields: virtual machine name, UUID, disk image, MAC address, and any other unique parameters. Note that you can delete the UUID and MAC address lines and virsh will generate a UUID and MAC address.

```
# cp /tmp/guest.xml /tmp/new-guest.xml
# vi /tmp/new-guest.xml
```

Add the model line in the network interface section:

```
 <interface type='network'>
  [output truncated]
  <model type='e1000' />
</interface>
```

3. Create the new virtual machine:

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

The network performance should be better with the e1000 or virtio driver. (*BZ#517181*[1])

[1] https://bugzilla.redhat.com/show_bug.cgi?id=517181

Appendix A. Additional resources

To learn more about virtualization and Fedora, refer to the following resources.

A.1. Online resources

- *http://www.libvirt.org/* is the official website for the **`libvirt`** virtualization API.
- *http://virt-manager.et.redhat.com/* is the project website for the **Virtual Machine Manager** (virt-manager), the graphical application for managing virtual machines.
- Open Virtualization Center

  http://www.openvirtualization.com[1]
- Red Hat Documentation

  http://www.redhat.com/docs/
- Virtualization technologies overview

  http://virt.kernelnewbies.org[2]
- Red Hat Emerging Technologies group

  http://et.redhat.com[3]

A.2. Installed documentation

- **`man virsh`** and **`/usr/share/doc/libvirt-<version-number>`** — Contains sub commands and options for the **`virsh`** virtual machine management utility as well as comprehensive information about the **`libvirt`** virtualization library API.
- **`/usr/share/doc/gnome-applet-vm-<version-number>`** — Documentation for the GNOME graphical panel applet that monitors and manages locally-running virtual machines.
- **`/usr/share/doc/libvirt-python-<version-number>`** — Provides details on the Python bindings for the **`libvirt`** library. The **`libvirt-python`** package allows python developers to create programs that interface with the **`libvirt`** virtualization management library.
- **`/usr/share/doc/python-virtinst-<version-number>`** — Provides documentation on the **`virt-install`** command that helps in starting installations of Fedora and Red Hat Enterprise Linux related distributions inside of virtual machines.
- **`/usr/share/doc/virt-manager-<version-number>`** — Provides documentation on the Virtual Machine Manager, which provides a graphical tool for administering virtual machines.

Glossary

This glossary is intended to define the terms used in this Installation Guide.

Bare-metal	The term bare-metal refers to the underlying physical architecture of a computer. Running an operating system on bare-metal is another way of referring to running an unmodified version of the operating system on the physical hardware. An example of operating system running on bare metal is a normally installed operating system.
Full virtualization	KVM uses full, hardware-assisted virtualization. Full virtualization uses hardware features of the processor to provide total abstraction of the underlying physical system (*Bare-metal*) and creates a new virtual machine in which the guest operating systems can run. No modifications are needed in the guest operating system. The guest operating system and any applications on the guest are not aware of the virtualized environment and run normally. Para-virtualization requires a modified version of the Linux operating system.
Fully virtualized	See *Full virtualization*.
Guest system	Also known as guests, virtual machines, virtual servers or domains.
Hardware Virtual Machine	See *Full virtualization*
Host	The host operating system runs virtualized guests.
Hypervisor	The hypervisor is the software layer that abstracts the hardware from the operating system permitting multiple operating systems to run on the same hardware. The hypervisor runs on a host operating system allowing other virtualized operating systems to run on the host's hardware. The *Kernel-based Virtual Machine* hypervisor is provided with Fedora.
I/O	Short for input/output (pronounced "eye-oh"). The term I/O describes any program, operation or device that transfers data to or from a computer and to or from a peripheral device. Every transfer is an output from one device and an input into another. Devices such as keyboards and mouses are input-only devices while devices such as printers are output-only. A writable CD-ROM is both an input and an output device.
Kernel SamePage Merging	The Kernel SamePage Merging (KSM) module is used by the KVM hypervisor to allow KVM guests to share identical memory pages. The pages shared are usually common libraries or other identical, high-use data. KSM can increase the performance of certain guests by keeping these libraries in cache for various guests as well as increasing guest density.
Kernel-based Virtual Machine	KVM (Kernel-based Virtual Machine) is a *Full virtualization* solution for Linux on AMD64 and Intel 64 hardware. VM is a Linux kernel module built for the standard Red Hat Enterprise Linux kernel. KVM can run multiple, unmodified virtualized guest Windows and Linux operating

systems. KVM is a hypervisor which uses the libvirt virtualization tools (virt-manager and virsh).

KVM is a set of Linux kernel modules which manage devices, memory and management APIs for the Hypervisor module itself. Virtualized guests are run as Linux processes and threads which are controlled by these modules.

LUN

A Logical Unit Number (LUN) is a number assigned to a logical unit (a SCSI protocol entity).

MAC Addresses

The Media Access Control Address is the hardware address for a Network Interface Controller. In the context of virtualization MAC addresses must be generated for virtual network interfaces with each MAC on your local domain being unique.

Migration

Migration is name for the process of moving a virtualized guest from one host to another. Migration can be conducted offline (where the guest is suspended and then moved) or live (where a guest is moved without suspending). KVM fully virtualized guests can be migrated.

Migration is a key feature of virtualization as software is completely separated from hardware. Migration is useful for:

- Load balancing - guests can be moved to hosts with lower usage when a host becomes overloaded.
- Hardware failover - when hardware devices on the host start to fail, guests can be safely relocated so the host can be powered down and repaired.
- Energy saving - guests can be redistributed to other hosts and host systems powered off to save energy and cut costs in low usage periods.
- Geographic migration - guests can be moved to another location for lower latency or in serious circumstances.

Shared, networked storage is used for storing guest images. Without shared storage migration is not possible.

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

The time an offline migration takes depends network bandwidth and latency. A guest with 2GB of memory should take several seconds on a 1 Gbit Ethernet link.

A live migration keeps the guest running on the source host and begins moving the memory without stopping the guest. All modified memory pages are monitored for changes and sent to the destination while the image is sent. The memory is updated with the changed pages. The process continues until the amount of pause time allowed

for the guest equals the predicted time for the final few pages to be transfer. KVM estimates the time remaining and attempts to transfer the maximum amount of page files from the source to the destination until KVM predicts the amount of remaining pages can be transferred during a very brief time while the virtualized guest is paused. The registers are loaded on the new host and the guest is then resumed on the destination host. If the guest cannot be merged (which happens when guests are under extreme loads) the guest is paused and then an offline migration is started instead.

The time an offline migration takes depends network bandwidth and latency as well as activity on the guest. If the guest is using significant I/O or CPU the migration will take much longer.

Para-virtualization

Para-virtualization uses a special kernel, sometimes referred to as the Xen kernel or the *kernel-xen* package. Para-virtualized guest kernels are run concurrently on the host while using the host's libraries and devices. A para-virtualized installation can have complete access to all devices on the system which can be limited with security settings (SELinux and file controls). Para-virtualization is faster than full virtualization. Para-virtualization can effectively be used for load balancing, provisioning, security and consolidation advantages.

As of Fedora 9 a special kernel will no longer be needed. Once this patch is accepted into the main Linux tree all Linux kernels after that version will have para-virtualization enabled or available.

Para-virtualized

See *Para-virtualization*,

Para-virtualized drivers

Para-virtualized drivers are device drivers that operate on fully virtualized Linux guests. These drivers greatly increase performance of network and block device I/O for fully virtualized guests.

PCI passthrough

The KVM hypervisor supports attaching PCI devices on the host system to virtualized guests. PCI passthrough allows guests to have exclusive access to PCI devices for a range of tasks. PCI passthrough allows PCI devices to appear and behave as if they were physically attached to the guest operating system.

phy device

The phy device parameter allows guest's to access physical disks. Physical disks includes:

- LVM volumes (for example, **`/dev/VolGroup00/LogVol02`**),
- disk partitions (for example, **`/dev/sda5`**), and
- whole block devices (for example, **`/dev/sda`**).

Physical mode provides the best performance as the hypervisor bypasses extra layers of software on the host at the price of slightly less flexibility in managing the device.

Security Enhanced Linux

Short for Security Enhanced Linux, SELinux uses Linux Security Modules (LSM) in the Linux kernel to provide a range of minimum privilege required security policies.

Universally Unique Identifier	A Universally Unique Identifier (UUID) is a standardized numbering method for devices, systems and certain software objects in distributed computing environments. Types of UUIDs in virtualization include: `ext2` and `ext3` file system identifiers, RAID device identifiers, iSCSI and LUN device identifiers, MAC addresses and virtual machine identifiers.
Virtual machines	A virtual machine is a software implementation of a physical machine or programming language (for example the Java Runtime Environment or LISP). Virtual machines in the context of virtualization are operating systems running on virtualized hardware.
Virtualization	Virtualization is a broad computing term for running software, usually operating systems, concurrently and isolated from other programs on one system. Most existing implementations of virtualization use a hypervisor, a software layer on top of an operating system, to abstract hardware. The hypervisor allows multiple operating systems to run on the same physical system by giving the guest operating system virtualized hardware. There are various methods for virtualizing operating systems: • Hardware-assisted virtualization is the technique used for full virtualization with KVM (definition: *Full virtualization*) • Para-virtualization is a technique used by Xen to run Linux guests. (definition: *Para-virtualization*) • Software virtualization or emulation. Software virtualization uses binary translation and other emulation techniques to run unmodified operating systems. Software virtualization is significantly slower than hardware-assisted virtualization or para-virtualization. Software virtualization, in the form of QEMU or BORCH, works in Fedora, it's just slow. Fedora supports hardware-assisted, full virtualization with the KVM hypervisor.
Virtualized CPU	A system has a number of virtual CPUs (VCPUs) relative to the number of physical processor cores. The number of virtual CPUs is finite and represents the total number of virtual CPUs that can be assigned to guest virtual machines.

Appendix B. Revision History

Revision 13 Wed Apr 23 2010 Christopher Curran ccurran@redhat.com

Fedora update and port.

Appendix C. Colophon

This manual was written in the DocBook XML v4.3 format.

This book is based on the work of Jan Mark Holzer and Chris Curran.

Other writing credits go to:

- Don Dutile contributed technical editing for the para-virtualized drivers section.
- Barry Donahue contributed technical editing for the para-virtualized drivers section.
- Rick Ring contributed technical editing for the Virtual Machine Manager Section.
- Michael Kearey contributed technical editing for the sections on using XML configuration files with virsh and virtualized floppy drives.
- Marco Grigull contributed technical editing for the software compatibility and performance section.
- Eugene Teo contributed technical editing for the Managing Guests with virsh section.

Publican, the publishing tool which produced this book, was written by Jeffrey Fearn.

Translators

Due to technical limitations, the translators credited in this section are those who worked on previous versions of the *Red Hat Enterprise Linux Virtualization Guide* and the *Fedora Virtualization Guide*.

To find out who translated the current version of the guide, visit *https://fedoraproject.org/wiki/Fedora_13_Documentation_Translations_-_Contributors*. These translators will receive credit in subsequent versions of this guide.

- Simplified Chinese
  - Leah Wei Liu
- Traditional Chinese
  - Chester Cheng
  - Terry Chuang
- Japanese
  - Kiyoto Hashida
- Korean
  - Eun-ju Kim
- Dutch
  - Geert Warrink
- French
  - Sam Friedmann

- German
  - Hedda Peters
- Greek
  - Nikos Charonitakis
- Italian
  - Silvio Pierro
  - Francesco Valente
- Brazilian Portuguese
  - Glaucia de Freitas
  - Leticia de Lima
- Spanish
  - Domingo Becker
  - Héctor Daniel Cabrera
  - Angela Garcia
  - Gladys Guerrero
- Russian
  - Yuliya Poyarkova