



Red Hat Enterprise Linux 5 Virtualization Guide

Virtualization Documentation

Scott Radvan
Jan Mark Holzer

Dayle Parker

Christopher Curran

Red Hat Enterprise Linux 5 Virtualization Guide

Virtualization Documentation

Scott Radvan
Red Hat Engineering Content Services
sradvan@redhat.com

Dayle Parker
Red Hat Engineering Content Services
dayleparker@redhat.com

Christopher Curran
Red Hat Engineering Content Services
ccurran@redhat.com

Jan Mark Holzer
Red Hat Emerging Technology Group

Legal Notice

Copyright © 2008–2014 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

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

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

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

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

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

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

Node.js® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

The Red Hat Enterprise Linux Virtualization Guide contains information on installation, configuring, administering, and troubleshooting virtualization technologies included with Red Hat Enterprise Linux.

Table of Contents

Preface	6
1. About this book	6
1.1. Overview	6
2. Document Conventions	6
2.1. Typographic Conventions	6
2.2. Pull-quote Conventions	8
2.3. Notes and Warnings	8
3. We need your feedback	9
3.1. Technical review requests	9
4. What is Virtualization?	9
5. Types of Virtualization	10
5.1. Full Virtualization	10
5.2. Para-Virtualization	10
5.3. Para-virtualized drivers	10
6. How CIOs should think about virtualization	10
Part I. Requirements and Limitations for Virtualization with Red Hat Enterprise Linux	13
System requirements, support restrictions and limitations	13
Chapter 1. System requirements	14
Chapter 2. Xen restrictions and support	16
Chapter 3. KVM restrictions and support	17
Chapter 4. Hyper-V restrictions and support	18
4.1. Hyper-V drivers	18
Chapter 5. Virtualization limitations	19
5.1. General limitations for virtualization	19
5.2. KVM limitations	19
5.3. Xen limitations	20
5.4. Application limitations	22
Part II. Installation	23
Virtualization installation topics	23
Chapter 6. Installing the virtualization packages	24
6.1. Installing Xen with a new Red Hat Enterprise Linux installation	24
6.2. Installing Xen packages on an existing Red Hat Enterprise Linux system	27
6.3. Installing KVM with a new Red Hat Enterprise Linux installation	28
6.4. Installing KVM packages on an existing Red Hat Enterprise Linux system	30
Chapter 7. Guest installation overview	32
7.1. Creating guests with virt-install	32
7.2. Creating guests with virt-manager	32
7.3. Installing guests with PXE	44
Chapter 8. Guest operating system installation procedures	50
8.1. Installing Red Hat Enterprise Linux 5 as a para-virtualized guest	50
8.2. Installing Red Hat Enterprise Linux as a fully virtualized guest	89
8.3. Installing Windows XP as a fully virtualized guest	98
8.4. Installing Windows Server 2003 as a fully virtualized guest	114

8.5. Installing Windows Server 2008 as a fully virtualized guest	116
Part III. Configuration	128
Configuring Virtualization in Red Hat Enterprise Linux	128
Chapter 9. Virtualized storage devices	129
9.1. Creating a virtualized floppy disk controller	129
9.2. Adding storage devices to guests	130
9.3. Configuring persistent storage in Red Hat Enterprise Linux 5	133
9.4. Add a virtualized CD-ROM or DVD device to a guest	135
Chapter 10. Network Configuration	136
10.1. Network Address Translation (NAT) with libvirt	136
10.2. Bridged networking with libvirt	137
Chapter 11. Pre-Red Hat Enterprise Linux 5.4 Xen networking	140
11.1. Configuring multiple guest network bridges to use multiple Ethernet cards	140
11.2. Red Hat Enterprise Linux 5.0 laptop network configuration	141
Chapter 12. Xen Para-virtualized Drivers	146
12.1. System requirements	147
12.2. Para-virtualization Restrictions and Support	148
12.3. Installing the Para-virtualized Drivers	150
12.3.1. Common installation steps	151
12.3.2. Installation and Configuration of Para-virtualized Drivers on Red Hat Enterprise Linux 3	
12.3.3. Installation and Configuration of Para-virtualized Drivers on Red Hat Enterprise Linux 4	152
12.3.4. Xen Para-virtualized Drivers on Red Hat Enterprise Linux 5	156
12.3.5. Xen Para-virtualized Drivers on Red Hat Enterprise Linux 6	158
12.4. Para-virtualized Network Driver Configuration	161
12.5. Additional Para-virtualized Hardware Configuration	164
12.5.1. Virtualized Network Interfaces	164
12.5.2. Virtual Storage Devices	165
Chapter 13. KVM Para-virtualized Drivers	167
13.1. Installing the KVM Windows para-virtualized drivers	167
13.2. Installing drivers with a virtualized floppy disk	177
13.3. Using KVM para-virtualized drivers for existing devices	177
13.4. Using KVM para-virtualized drivers for new devices	177
Chapter 14. Installing Red Hat Enterprise Linux 6 as a Xen guest on Red Hat Enterprise Linux 5	
14.1. Installing Red Hat Enterprise Linux 6 as a Xen para-virtualized guest on Red Hat Enterprise Linux 5	181
14.1.1. Using virt-install	181
14.1.2. Using virt-manager	182
14.2. Installing Red Hat Enterprise Linux 6 as a Xen fully virtualized guest on Red Hat Enterprise Linux 5	191
Chapter 15. PCI passthrough	192
15.1. Adding a PCI device with virsh	193
15.2. Adding a PCI device with virt-manager	195
15.3. PCI passthrough with virt-install	200
15.4. Removing a PCI passthrough device for host re-use	201
15.5. PCI passthrough for para-virtualized Xen guests on Red Hat Enterprise Linux	202
Chapter 16. SR-IOV	204

16.1. Introduction	204
16.2. Using SR-IOV	204
16.3. Troubleshooting SR-IOV	208
Chapter 17. KVM guest timing management	209
Part IV. Administration	213
Administering virtualized systems	213
Chapter 18. Server best practices	214
Chapter 19. Security for virtualization	215
19.1. Storage security issues	215
19.2. SELinux and virtualization	215
19.3. SELinux	216
19.4. Virtualization firewall information	217
Chapter 20. Managing guests with xend	219
Chapter 21. Xen live migration	221
21.1. A live migration example	222
21.2. Configuring guest live migration	229
Chapter 22. KVM live migration	230
22.1. Live migration requirements	230
22.2. Share storage example: NFS for a simple migration	231
22.3. Live KVM migration with virsh	232
22.4. Migrating with virt-manager	233
Chapter 23. Remote management of guests	241
23.1. Remote management with SSH	241
23.2. Remote management over TLS and SSL	242
23.3. Transport modes	244
Part V. Virtualization Storage Topics	248
Introduction to storage administration for virtualization	248
Chapter 24. Using shared storage with virtual disk images	249
24.1. Using iSCSI for storing virtual disk images	249
24.1.1. How to set up an iSCSI target on Red Hat Enterprise Linux	249
24.1.2. How to configure iSCSI on a libvirt KVM host and provision a guest using virt-install	251
Part VI. Virtualization Reference Guide	255
Virtualization commands, system tools, applications and additional systems reference	255
Chapter 25. Virtualization tools	256
Chapter 26. Managing guests with virsh	258
Chapter 27. Managing guests with the Virtual Machine Manager (virt-manager)	267
27.1. The Add Connection window	267
27.2. The Virtual Machine Manager main window	268
27.3. The guest Overview tab	268
27.4. Virtual Machine graphical console	269
27.5. Starting virt-manager	270
27.6. Restoring a saved machine	271

27.6. Restoring a saved machine	271
27.7. Displaying guest details	272
27.8. Status monitoring	276
27.9. Displaying guest identifiers	277
27.10. Displaying a guest's status	278
27.11. Displaying virtual CPUs	279
27.12. Displaying CPU usage	280
27.13. Displaying memory usage	281
27.14. Managing a virtual network	282
27.15. Creating a virtual network	283
Chapter 28. The xm command quick reference	292
Chapter 29. Configuring the Xen kernel boot parameters	294
Chapter 30. Configuring ELILO	296
Chapter 31. libvirt configuration reference	299
Chapter 32. Xen configuration files	300
Part VII. Tips and Tricks	307
Tips and Tricks to Enhance Productivity	307
Chapter 33. Tips and tricks	308
33.1. Automatically starting guests	308
33.2. Changing between the KVM and Xen hypervisors	308
33.2.1. Xen to KVM	308
33.2.2. KVM to Xen	310
33.3. Using qemu-img	311
33.4. Overcommitting Resources	313
33.5. Modifying /etc/grub.conf	314
33.6. Verifying virtualization extensions	315
33.7. Accessing data from a guest disk image	316
33.8. Setting KVM processor affinities	318
33.9. Generating a new unique MAC address	322
33.10. Limit network bandwidth for a Xen guest	323
33.11. Configuring Xen processor affinities	324
33.12. Modifying the Xen hypervisor	324
33.13. Very Secure ftpd	325
33.14. Configuring LUN Persistence	325
33.15. Disable SMART disk monitoring for guests	327
33.16. Cleaning up old Xen configuration files	327
33.17. Configuring a VNC Server	327
33.18. Cloning guest configuration files	328
33.19. Duplicating an existing guest and its configuration file	328
Chapter 34. Creating custom libvirt scripts	330
34.1. Using XML configuration files with virsh	330
Part VIII. Troubleshooting	331
Introduction to Troubleshooting and Problem Solving	331
Chapter 35. Troubleshooting Xen	332
35.1. Debugging and troubleshooting Xen	332
35.1. Log files overview	333

35.2. Log files overview	333
35.3. Log file descriptions	334
35.4. Important directory locations	334
35.5. Troubleshooting with the logs	335
35.6. Troubleshooting with the serial console	335
35.7. Para-virtualized guest console access	336
35.8. Fully virtualized guest console access	336
35.9. Common Xen problems	336
35.10. Guest creation errors	337
35.11. Troubleshooting with serial consoles	337
35.11.1. Serial console output for Xen	337
35.11.2. Xen serial console output from para-virtualized guests	338
35.11.3. Serial console output from fully virtualized guests	339
35.12. Xen configuration files	339
35.13. Interpreting Xen error messages	340
35.14. The layout of the log directories	343
Chapter 36. Troubleshooting	344
36.1. Identifying available storage and partitions	344
36.2. After rebooting Xen-based guests the console freezes	344
36.3. Virtualized Ethernet devices are not found by networking tools	344
36.4. Loop device errors	344
36.5. Failed domain creation caused by a memory shortage	345
36.6. Wrong kernel image error	345
36.7. Wrong kernel image error - non-PAE kernel on a PAE platform	346
36.8. Fully-virtualized 64 bit guest fails to boot	346
36.9. A missing localhost entry causes virt-manager to fail	346
36.10. Microcode error during guest boot	347
36.11. Python depreciation warning messages when starting a virtual machine	347
36.12. Enabling Intel VT and AMD-V virtualization hardware extensions in BIOS	347
36.13. KVM networking performance	348
Chapter 37. Troubleshooting the Xen para-virtualized drivers	351
37.1. Red Hat Enterprise Linux 5 Virtualization log file and directories	351
37.2. Para-virtualized guest fail to load on a Red Hat Enterprise Linux 3 guest operating system	
37.3. A warning message is displayed while installing the para-virtualized drivers on Red Hat Enterprise Linux 3	352
37.4. Manually loading the para-virtualized drivers	353
37.5. Verifying the para-virtualized drivers have successfully loaded	354
37.6. The system has limited throughput with para-virtualized drivers	354
Additional resources	355
A.1. Online resources	355
A.2. Installed documentation	355
Colophon	357

Preface

The Red Hat Enterprise Linux Virtualization Guide covers all aspects of using and managing virtualization products included with Red Hat Enterprise Linux.

1. About this book

1.1. Overview

This book is divided into 8 parts:

- » Requirements and Limitations
- » Installation
- » Configuration
- » Administration
- » Storage
- » Reference
- » Tips and Tricks
- » Troubleshooting

2. Document Conventions

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

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 keys 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 key, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from an individual key by the plus sign that connects each part of a key combination. For example:

Press **Enter** to execute the command.

Press **Ctrl+Alt+F2** to switch to a virtual terminal.

The first example highlights a particular key to press. The second example highlights a key combination: a set of three keys 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 submenu titles. For example:

Choose **System → Preferences → Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, select 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:

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                                         struct kvm_assigned_pci_dev *assigned_dev)
{
    int r = 0;
    struct kvm_assigned_dev_kernel *match;

    mutex_lock(&kvm->lock);

    match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
                                  assigned_dev->assigned_dev_id);
    if (!match) {
        printk(KERN_INFO "%s: device hasn't been assigned before, "
               "so cannot be deassigned\n", __func__);
        r = -EINVAL;
        goto out;
    }

    kvm_deassign_device(kvm, match);

    kvm_free_assigned_device(kvm, match);

out:
    mutex_unlock(&kvm->lock);
    return r;
}
```

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” will not cause data loss but may cause irritation and frustration.



Warning

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

3. We need your 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. Submit a report in Bugzilla: <http://bugzilla.redhat.com/> against **Red Hat Enterprise Linux 5** with the **Virtualization_Guide** component.

When submitting a bug report, be sure to mention the manual's identifier: **Virtualization_Guide** and the Edition number: **5.11**.

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, include the section number and some of the surrounding text so we can find it easily.

3.1. Technical review requests

All review requests are classified into one of the following five categories:

New content

content documented for the first time — an entirely new feature, procedure, or concept. For example: "Section now describes the new procedure for creating bootable USB devices."

Correction

a factual error previously present in the text has been corrected. For example: "Section previously stated (incorrectly) that IPv4 and IPv6 were both supported; section now states that IPv6 has never been supported."

Clarification

material that was already factually correct but is now better explained. Clarifications are usually in response to reader feedback that the previous content was confusing or misleading in some way. For example: "Paths described in Example 1.2.3 now better reflect the directory structure of an actual installed system."

Obsoletion

a description of a feature or a procedure has been dropped. Material might be obsolete because of a feature that is no longer supported, a known issue that has been corrected, or hardware that is now obsolete. For example, "Section no longer describes how to update kernel modules using a floppy disk."

Verification

a request to check a fact, procedure, or whether material should be obsoleted. For example, "Section describes how to connect to a generic iSCSI storage device. Please verify this on your hardware" or "Section still describes how to update kernel modules using a LS-120 SuperDisk; please verify that we still need to tell readers about this obsolete hardware."

4. What is 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 that controls hardware and provides guest operating systems with access to underlying hardware devices. The hypervisor allows multiple operating systems to run on the same physical system by offering virtualized hardware to a guest operating system.

5. Types of Virtualization

5.1. Full Virtualization

Red Hat Enterprise Linux contains virtualization packages and tools to run fully virtualized, unmodified, operating system guests. This provides companies with the ability to consolidate older systems onto newer, more efficient hardware, and reduces physical space and operating costs involved with powering and cooling older, less efficient systems. Full virtualization offers lower I/O performance than native (or bare-metal) installations of operating systems.

5.2. Para-Virtualization

Para-virtualization is a virtualization technique which involves running modified versions of operating systems. The para-virtualized operating system is modified to be aware that it is being virtualized, offering an increased ability for optimization as the guest is more aware of its environment. Performance is generally very close to running bare-metal, non-virtualized operating systems.

5.3. Para-virtualized drivers

Para-virtualization and full virtualization can be combined to allow unmodified operating systems to receive near native I/O performance by using para-virtualized drivers on fully virtualized operating systems.

The para-virtualized drivers contain storage and network device drivers for fully virtualized Microsoft Windows® guests. The drivers provide Microsoft Windows® guests running on Red Hat Enterprise Linux with enhanced disk and network I/O performance.

6. How CIOs should think about virtualization

by Lee Congdon, Chief Information Officer, Red Hat, Inc.

You may already be heavily invested in the rapidly emerging technology of virtualization. If so, consider some of the ideas below for further exploiting the technology. If not, now is the right time to get started.

Virtualization provides a set of tools for increasing flexibility and lowering costs, things that are important in every enterprise and Information Technology organization. Virtualization solutions are becoming increasingly available and rich in features.

Since virtualization can provide significant benefits to your organization in multiple areas, you should be establishing pilots, developing expertise and putting virtualization technology to work now.

Virtualization for Innovation

In essence, virtualization increases flexibility by decoupling an operating system and the services and applications supported by that system from a specific physical hardware platform. It allows the establishment of multiple virtual environments on a shared hardware platform.

Organizations looking to innovate find that the ability to create new systems and services without installing additional hardware (and to quickly tear down those systems and services when they are no longer needed) can be a significant boost to innovation.

Among possible approaches are the rapid establishment of development systems for the creation of custom software, the ability to quickly set up test environments, the capability to provision alternate software solutions and compare them without extensive hardware investments, support for rapid prototyping and agile development environments, and the ability to quickly establish new production services on demand.

These environments can be created in house or provisioned externally, as with Amazon's EC2 offering. Since the cost to create a new virtual environment can be very low, and can take advantage of existing hardware, innovation can be facilitated and accelerated with minimal investment.

Virtualization can also excel at supporting innovation through the use of virtual environments for training and learning. These services are ideal applications for virtualization technology. A student can start course work with a known, standard system environment. Class work can be isolated from the production network. Learners can establish unique software environments without demanding exclusive use of hardware resources.

As the capabilities of virtual environments continue to grow, we're likely to see increasing use of virtualization to enable portable environments tailored to the needs of a specific user. These environments can be moved dynamically to an accessible or local processing environment, regardless of where the user is located. The user's virtual environments can be stored on the network or carried on a portable memory device.

A related concept is the Appliance Operating System, an application package oriented operating system designed to run in a virtual environment. The package approach can yield lower development and support costs as well as insuring the application runs in a known, secure environment. An Appliance Operating System solution provides benefits to both application developers and the consumers of those applications.

How these applications of virtualization technology apply in your enterprise will vary. If you are already using the technology in more than one of the areas noted above, consider an additional investment in a solution requiring rapid development. If you haven't started with virtualization, start with a training and learning implementation to develop skills, then move on to application development and testing. Enterprises with broader experience in virtualization should consider implementing portable virtual environments or application appliances.

Virtualization for Cost Savings

Virtualization can also be used to lower costs. One obvious benefit comes from the consolidation of servers into a smaller set of more powerful hardware platforms running a collection of virtual environments. Not only can costs be reduced by reducing the amount of hardware and reducing the amount of unused capacity, but application performance can actually be improved since the virtual guests execute on more powerful hardware.

Further benefits include the ability to add hardware capacity in a non-disruptive manner and to dynamically migrate workloads to available resources.

Depending on the needs of your organization, it may be possible to create a virtual environment for disaster recovery. Introducing virtualization can significantly reduce the need to replicate identical hardware environments and can also enable testing of disaster scenarios at lower cost.

Virtualization provides an excellent solution for addressing peak or seasonal workloads. If you have

complementary workloads in your organization, you can dynamically allocate resources to the applications which are currently experiencing the greatest demand. If you have peak workloads that you are currently provisioning inside your organization, you may be able to buy capacity on demand externally and implement it efficiently using virtual technology.

Cost savings from server consolidation can be compelling. If you are not exploiting virtualization for this purpose, you should start a program now. As you gain experience with virtualization, explore the benefits of workload balancing and virtualized disaster recovery environments.

Virtualization as a Standard Solution

Regardless of the specific needs of your enterprise, you should be investigating virtualization as part of your system and application portfolio as the technology is likely to become pervasive. We expect operating system vendors to include virtualization as a standard component, hardware vendors to build virtual capabilities into their platforms, and virtualization vendors to expand the scope of their offerings.

If you don't have plans to incorporate virtualization in your solution architecture, now is a very good time to identify a pilot project, allocate some underutilized hardware platforms, and develop expertise with this flexible and cost-effective technology. Then, extend your target architectures to incorporate virtual solutions. Although substantial benefits are available from virtualizing existing services, building new applications with an integrated virtualization strategy can yield further benefits in both manageability and availability.

You can learn more about Red Hat's virtualization solutions at <http://www.redhat.com/products/>

Part I. Requirements and Limitations for Virtualization with Red Hat Enterprise Linux

System requirements, support restrictions and limitations

These chapters outline the system requirements, support restrictions, and limitations of virtualization on Red Hat Enterprise Linux.

Chapter 1. System requirements

This chapter lists system requirements for successfully running virtualization with Red Hat Enterprise Linux.

The requirements for virtualization vary depending on the type of hypervisor. The Kernel-based Virtual Machine (KVM) and Xen hypervisors are provided with Red Hat Enterprise Linux 5. Both support Full virtualization. The Xen hypervisor also supports Para-virtualization.

For information on installing the virtualization packages, read [Chapter 6, *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 thread for each virtualized CPU and one for the hypervisor.
- » 2GB of RAM plus additional RAM for guests.

Note

KVM can overcommit physical resources for guests. Overcommitting resources means the total virtualized RAM and processor cores used by the guests can exceed the physical RAM and processor cores on the host. For information on safely overcommitting resources with KVM, see [Section 33.4, “Overcommitting Resources”](#).

Xen para-virtualization requirements

Para-virtualized guests require a Red Hat Enterprise Linux 5 installation tree available over the network using the **NFS**, **FTP** or **HTTP** protocols.

Xen full virtualization requirements

Full virtualization with the Xen Hypervisor requires:

- » an Intel processor with the Intel VT extensions, or
- » an AMD processor with the AMD-V extensions, or
- » an Intel Itanium processor.

See [Section 33.6, “Verifying virtualization extensions”](#) to determine if your processor has the virtualization extensions.

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.

See [Section 33.6, “Verifying virtualization extensions”](#) to determine if your processor has the virtualization extensions.

Storage support

The supported guest storage methods are:

- » Files on local storage
- » Physical disk partitions
- » Locally connected physical LUNs
- » LVM partitions
- » iSCSI and Fibre Channel based LUNs



Important

File-based guest images are stored in the `/var/lib/libvirt/images/` directory by default. If you choose to use a different directory you must label the new directory according to SELinux policy. See [Section 19.2, “SELinux and virtualization”](#) for details.

Chapter 2. Xen restrictions and support

Red Hat Enterprise Linux 5 supports various combinations for hosts and guests. Processor and memory limitations exist and can be viewed at the following URLs:

- » For host systems: <http://www.redhat.com/products/enterprise-linux/server/compare.html>
- » For hypervisors: <http://www.redhat.com/resourcelibrary/articles/virtualization-limits-rhel-hypervisors>

The following URL shows a complete matrix of supported operating systems and host and guest combinations:

- » <http://www.redhat.com/resourcelibrary/articles/enterprise-linux-virtualization-support>

Note

To utilize para-virtualization on Red Hat Enterprise Linux 5, your processor must have the Physical Address Extension (PAE) instruction set.

Note

Virtualization with the Xen hypervisor on the Intel Itanium architecture requires the guest firmware image package. See [Installing the Xen hypervisor with yum](#) for more information.

Chapter 3. KVM restrictions and support

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

To verify the presence of the extensions in your system, see [Section 33.6, “Verifying virtualization extensions”](#).

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

- » For host systems: <http://www.redhat.com/products/enterprise-linux/server/compare.html>
- » For hypervisors: <http://www.redhat.com/resource/library/articles/virtualization-limits-rhel-hypervisors>

The following URL shows a complete matrix of supported operating systems and host and guest combinations:

- » <http://www.redhat.com/resource/library/articles/enterprise-linux-virtualization-support>

Chapter 4. Hyper-V restrictions and support

Certification of guests running under the Microsoft Hyper-V server is conducted by Microsoft. Red Hat Enterprise Linux 5 is fully certified to run under the Microsoft Hyper-V server.

4.1. Hyper-V drivers

For enhanced performance, Red Hat Enterprise Linux 5 provides support for Hyper-V para-virtualized drivers. These drivers (and their kernel module name) are described in the following list:

- » **Hyper-V vmbus driver (hv_vmbus)** - Provides the infrastructure for other Hyper-V drivers to communicate with the hypervisor.
- » **Utility driver (hv_utils)** - Provides Hyper-V integration services such as shutdown, time synchronization, heartbeat and Key-Value Pair Exchange.
- » **Network driver (hv_netvsc)** - Provides network performance improvements.
- » **Storage driver (hv_storvsc)** - Increases performance when accessing storage (IDE and SCSI) devices.
- » **Mouse driver (hid_hyperv)** - Improves user experience by allowing mouse focus changes for a virtualized guest.
- » **Clocksource driver** - This driver provides a stable clock source for Red Hat Enterprise Linux 5.11 running within the Hyper-V platform.

Note

There is no Clocksource module, instead it is built directly into the kernel.

The Hyper-V kernel modules for Red Hat Enterprise Linux 5.11 are shown in the following output:

```
# /sbin/lsmod | grep hv
hv_netvsc          57153  0
hv_utils           41841  0
hv_storvsc         47681  2
hv_vmbus           66105  4 hv_netvsc,hid_hyperv,hv_utils,hv_storvsc
```

For further information on configuring network devices, storage controllers, and for details on Linux Integration Services when running Red Hat Enterprise Linux 5 under Hyper-V, see the following document: <http://technet.microsoft.com/en-us/library/dn531030.aspx>

Chapter 5. Virtualization limitations

This chapter covers additional limitations of the virtualization packages in Red Hat Enterprise Linux.

5.1. General limitations for virtualization

Converting between hypervisors

There is no support for converting Xen-based guests to KVM or KVM-based guests to Xen.

Other limitations

See the *Red Hat Enterprise Linux Release Notes* at <https://access.redhat.com/site/documentation/> for your version. The 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 system and network load before deploying heavy I/O applications such as database servers. Load testing and planning are important as virtualization performance can suffer under high I/O.

5.2. KVM limitations

The following limitations apply to the KVM hypervisor:

Constant TSC bit

Systems without a Constant Time Stamp Counter require additional configuration. See [Chapter 17, KVM guest timing management](#) to determine whether you have a Constant Time Stamp Counter and what additional configuration may be required.

Memory overcommit

KVM supports memory overcommit and can store the memory of guests in swap space. A guest will run slower if it is swapped frequently. When Kernel SamePage Merging (KSM) is used, make sure that the swap size is equivalent to the size of the overcommit ratio.

CPU overcommit

No support exists for having more than 10 virtual CPUs per physical processor core. A CPU overcommit configuration exceeding this limitation is unsupported and can cause problems with some guests.

Overcommitting CPUs has some risk and can lead to instability. See [Section 33.4, “Overcommitting Resources”](#) for tips and recommendations on overcommitting CPUs.

Virtualized SCSI devices

SCSI emulation is presently not supported. Virtualized SCSI devices are disabled in KVM.

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.

Hence, of the 32 available PCI devices for a guest, 4 are not removable. This means there are 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.

See [Chapter 21, Xen live migration](#) and [Chapter 22, KVM live migration](#) for more details on live migration.

Storage limitations

The host should not use disk labels to identify file systems in the **/etc/fstab** file, the **initrd** file or in the kernel command line. A security weakness exists if less privileged users or guests have write access to entire partitions or LVM volumes.

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

5.3. Xen limitations

Note

All limitations in this chapter are for Red Hat Enterprise Linux 5.11 except where noted.

Xen host (dom0) limitations

- ✖ A limit of 254 para-virtualized block devices per host exists. The total number of block devices attached to guests cannot exceed 254.



Note

There are two methods for working around the para-virtualized device limit: using phy devices (devices using the physical access mode) or using LVM on the guest.

A host has no limit to the number of **phy** devices it can have if it has sufficient resources.

LVM, or a similar logical partitioning tool, can be used on a block device to create additional logical partitions on a single para-virtualized block device.

Xen Para-virtualization limitations

- » For x86 guests, a maximum of 16GB memory per guest.
- » For x86_64 guests, a maximum of 168GB memory per guest.
- » A maximum of 254 devices per guest.
- » A maximum of 15 network devices per guest.

Xen full virtualization limitations

- » For x86 guests, a maximum of 16GB memory per guest.
- » A maximum of four virtualized (emulated) IDE devices per guest.

Devices using the para-virtualized drivers for fully-virtualized guests do not have this limitation.

- » Virtualized (emulated) IDE devices are limited by the total number of loopback devices supported by the system. The default number of available loopback devices on Red Hat Enterprise Linux 5.11 is 8. That is, by default, all guests on the system can each have no more than 8 virtualized (emulated) IDE devices.

For more information on loopback devices, their creation and use, see the [Red Hat Knowledge Solution 1721](#).



Note

The number of available loopback devices can be raised by modifying the kernel limit.

In the **/etc/modprobe.conf** file, add the following line:

```
options loop max_loop=64
```

Reboot the machine or run the following commands to update the kernel with this new limit:

```
# rmmod loop
# modprobe loop
```

- » A limit of 254 para-virtualized block devices per host. The total number of block devices (using the tap:aio driver) attached to guests cannot exceed 254 devices.
- » A maximum of 254 block devices using the para-virtualized drivers per guest.
- » A maximum of 15 network devices per guest.
- » A maximum of 15 virtualized SCSI devices per guest.

PCI passthrough limitations

- » PCI passthrough (attaching PCI devices to guests) is presently only supported on the following architectures:
 - 32 bit (x86) systems.
 - Intel 64 systems.
 - Intel Itanium 2 systems.

5.4. 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 high I/O load.

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

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

You should carefully evaluate database applications before running them on a 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 in these cases.

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 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 describe setting up the host and installing guests with Red Hat Enterprise Linux.

Chapter 6. Installing the virtualization packages

Before you can use virtualization, the virtualization packages must be installed on Red Hat Enterprise Linux. These can be installed either during installation, or after installation using the **yum** command.

You can install both the KVM and Xen hypervisors on a single system. The Xen hypervisor uses the *kernel-xen* package and the KVM hypervisor uses the default Red Hat Enterprise Linux kernel with the *kvm* kernel module. Xen and KVM each use a different kernel and only one hypervisor can be active at any given time.

To change hypervisor from Xen to KVM or KVM to Xen, see [Section 33.2, “Changing between the KVM and Xen hypervisors”](#).

6.1. Installing Xen with a new Red Hat Enterprise Linux installation

This section covers installing Xen packages and virtualization tools on a new Red Hat Enterprise Linux installation.

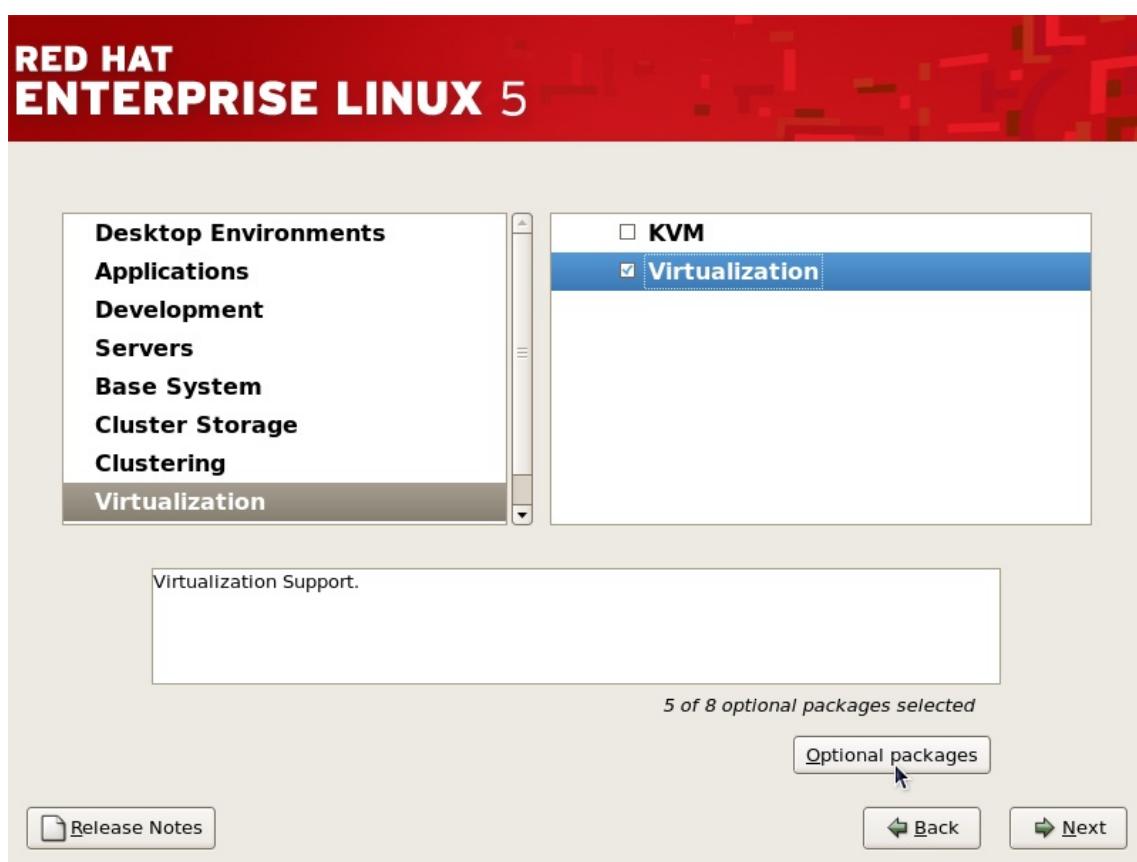
Note

The *Installation Guide* (available from access.redhat.com) details the installation of Red Hat Enterprise Linux.

1. Start an interactive Red Hat Enterprise Linux installation from the Red Hat Enterprise Linux Installation CD-ROM, DVD or PXE.
2. You must enter a valid installation number when prompted to receive access to the virtualization packages. Installation numbers can be obtained from Red Hat Customer Service.
3. Complete all steps until you see the package selection step.



- Select the **Virtualization** package group and the **Customize Now** radio button.
4. Select the **Virtualization** package group. The **Virtualization** package group selects the Xen hypervisor, **virt-manager**, **libvirt** and **virt-viewer** and all dependencies 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.



Important

You require a valid Red Hat account with the virtualization entitlement to receive updates for the virtualization packages.

Installing Xen packages with Kickstart files

This section describes how to use a Kickstart file to install Red Hat Enterprise Linux with the Xen 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 Red Hat Enterprise Linux with the virtualization packages.

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

```
%packages
@xen
```



Note

Fully virtualized guests on the Itanium® architecture require the guest firmware image package (*xen-ia64-guest-firmware*). Append the following package to your kickstart file:

```
xen-ia64-guest-firmware
```

More information on Kickstart is available in the *Installation Guide* at access.redhat.com

6.2. Installing Xen packages on an existing Red Hat Enterprise Linux system

The section describes the steps necessary to install the virtualization packages on a working Red Hat Enterprise Linux system.

Registering your system with the subscription-manager command

Your machines must be registered on your Red Hat account to receive packages and updates. To register an unregistered installation of Red Hat Enterprise Linux, run the **subscription-manager register** command and follow the prompts.

If you do not have a valid Red Hat subscription, visit the [Red Hat online store](#).

Installing the Xen hypervisor with yum

To use virtualization on Red Hat Enterprise Linux you need the **xen** and **kernel-xen** packages. The **xen** package contains the hypervisor and basic virtualization tools. The **kernel-xen** package contains a modified Linux kernel which runs as a virtual machine guest on the hypervisor.

To install the **xen** and **kernel-xen** packages, run:

```
# yum install xen kernel-xen
```

Additional virtualization packages are available for management and configuration.

Recommended virtualization packages:

python-virtinst

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

libvirt

libvirt is an API library for interacting with hypervisors. **libvirt** uses the **xm** virtualization framework and the **virsh** command line tool to manage and control virtual machines.

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
```

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

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



Note

The *Installation Guide* (available from access.redhat.com) covers installing Red Hat Enterprise Linux in detail.



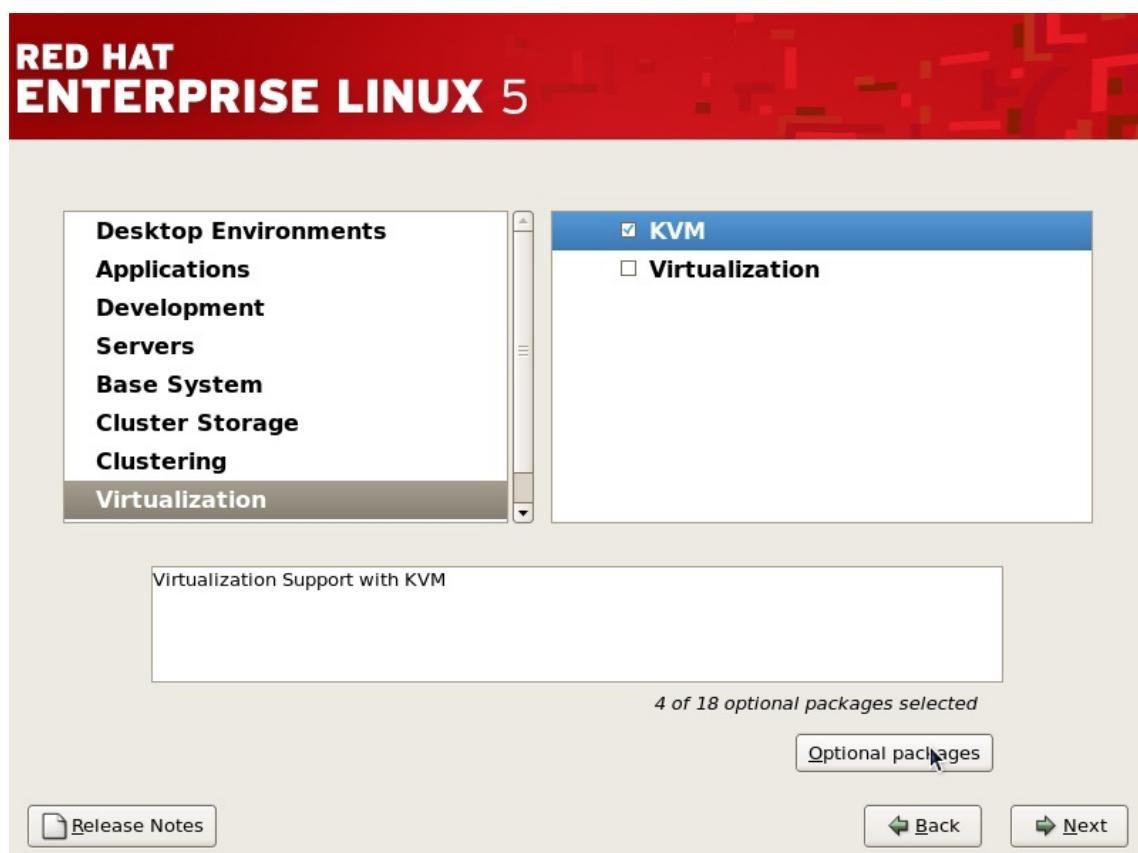
Important

You cannot select the virtualization packages during the installation without a valid installation number.

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



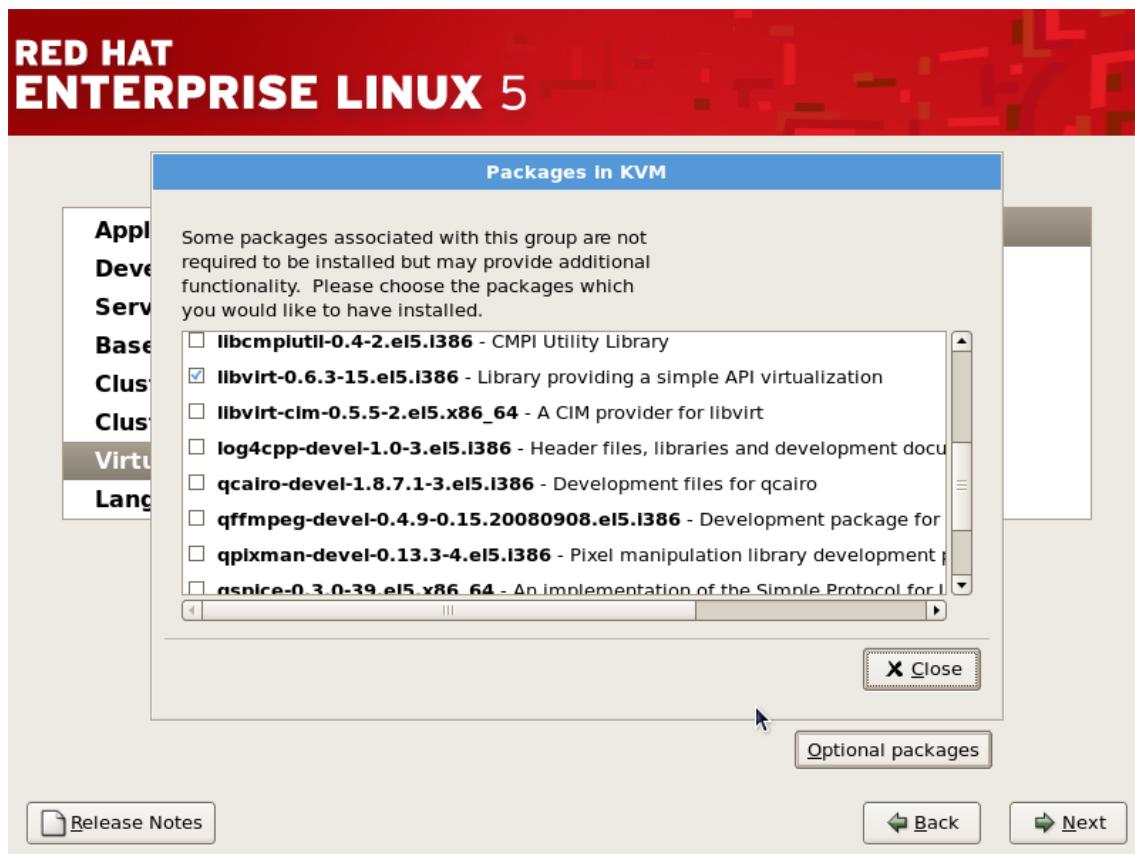
- Select the **Virtualization** package group and the **Customize Now** radio button.
4. Select the **KVM** package group. Deselect the **Virtualization** package group. This selects the KVM hypervisor, **virt-manager**, **libvirt** and **virt-viewer** for installation.



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.



Important

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

Installing KVM packages with Kickstart files

This section describes how to use a Kickstart file to install Red Hat Enterprise Linux with the 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 Red Hat Enterprise Linux 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 on Red Hat's website, access.redhat.com, in the *Installation Guide*.

6.4. Installing KVM packages on an existing Red Hat Enterprise Linux system

The section describes the steps for installing the KVM hypervisor on a working Red Hat Enterprise Linux 5 system.

Adding packages to your list of Red Hat account entitlements

This section describes how to enable entitlements in your Red Hat account for the virtualization packages. You need these entitlements enabled to install and update the virtualization packages on Red Hat Enterprise Linux. You require a valid Red Hat account in order to install virtualization packages on Red Hat Enterprise Linux.

If you do not have a valid Red Hat subscription, visit the [Red Hat online store](#).

Installing the KVM hypervisor with yum

To use virtualization on Red Hat Enterprise Linux you require the **kvm** package. The **kvm** package contains the KVM kernel module providing the KVM hypervisor on the default Red Hat Enterprise Linux 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 an API library for interacting with hypervisors. **libvirt** uses the **xm** virtualization framework and the **virsh** command line tool to manage and control virtual machines.

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
```

Chapter 7. Guest installation overview

After you have installed the virtualization packages on the host system you can create guest operating systems. 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 Red Hat Enterprise Linux, other Linux distributions and Windows. See [Chapter 8, Guest operating system installation procedures](#) for those procedures.

7.1. Creating guests with virt-install

You can use the **virt-install** command to create 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.

Example 7.1. Using virt-install with KVM to create a Red Hat Enterprise Linux 3 guest

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 7.2. Using virt-install to create a fedora 11 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
```

7.2. Creating guests with virt-manager

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

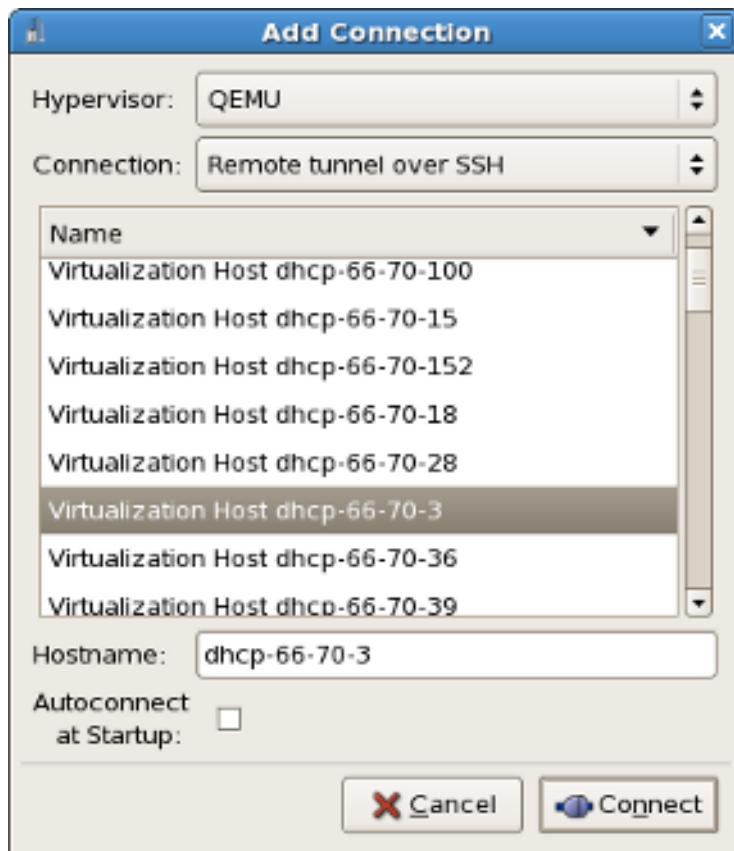
Procedure 7.1. Creating a guest with virt-manager

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.

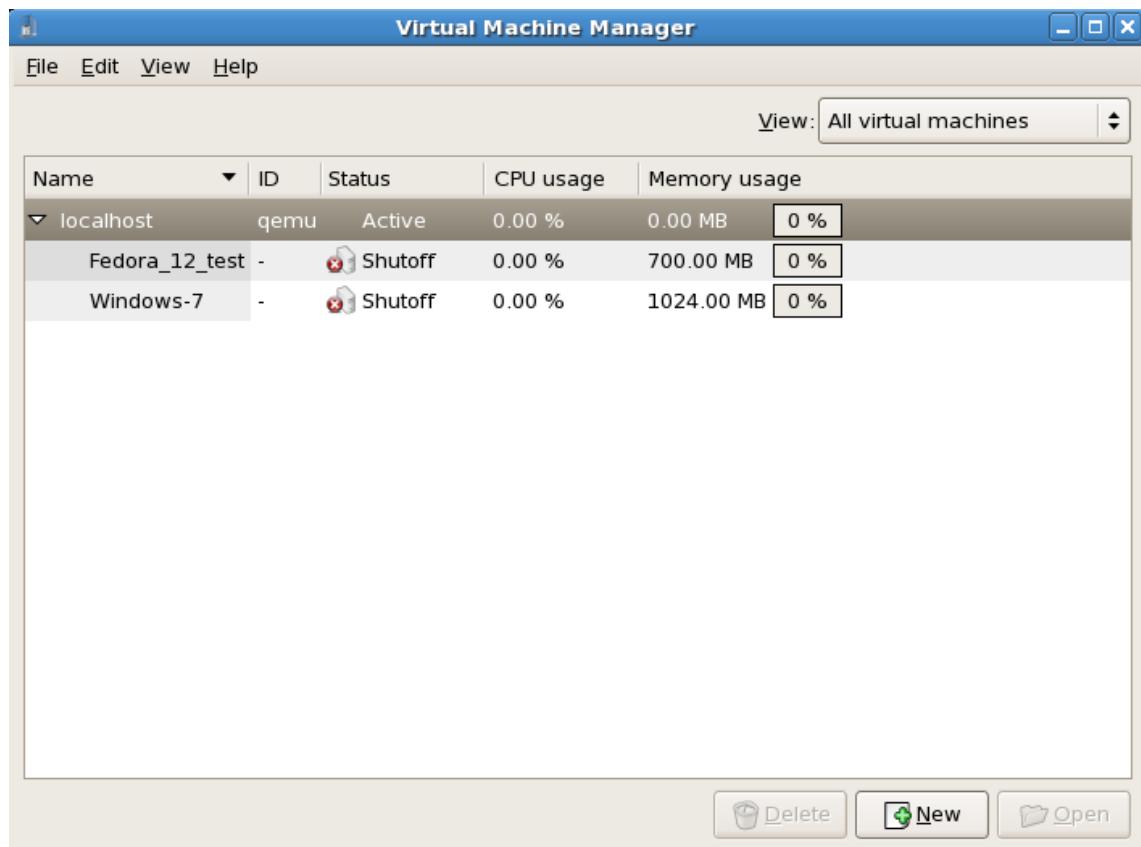
2. Optional: Open a remote hypervisor

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

Provide a name for your guest. Punctuation and whitespace characters are not permitted in versions before Red Hat Enterprise Linux 5.5. Red Hat Enterprise Linux 5.5 adds support for '_', '.', and '-' characters.



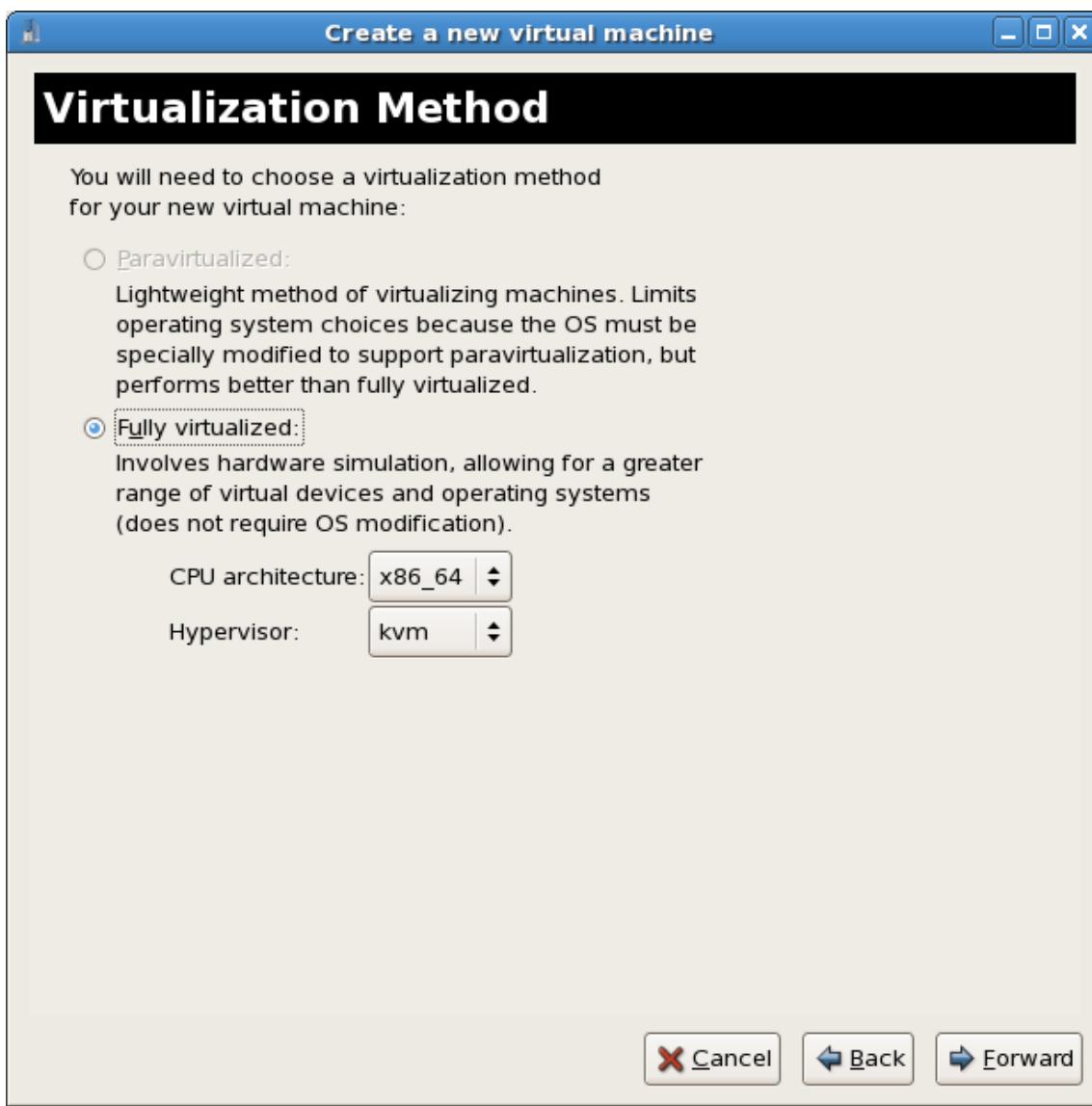
Press **Forward** to continue.

6. Choose virtualization method

The **Choosing a virtualization method** window appears. Choose between **Para-virtualized** or **Fully virtualized**.

Full virtualization requires a system with Intel® VT or AMD-V processor. If the virtualization extensions are not present the **fully virtualized** radio button or the **Enable kernel/hardware acceleration** will not be selectable. The **Para-virtualized** option will be grayed out if **kernel-xen** is not the kernel running presently.

If you connected to a KVM hypervisor, only full virtualization is available.



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 Red Hat Enterprise Linux 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 *Red Hat Enterprise Linux Deployment Guide*. Using this method requires a guest with a routable IP address or shared network device. See [Chapter 10, 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 proceed.



Important

Para-virtualized installation must be installed with a network installation tree. The installation tree must be accessible using one of the following network protocols: **HTTP**, **FTP** or **NFS**. The installation media URL must contain a Red Hat Enterprise Linux installation tree. This tree is hosted using **NFS**, **FTP** or **HTTP**.

8. Installation media selection

This window is dependent on what was selected in the previous step.

a. ISO image or physical 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 proceed.

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 proceed.

c. Network boot (PXE)

PXE installation does not have an additional step.

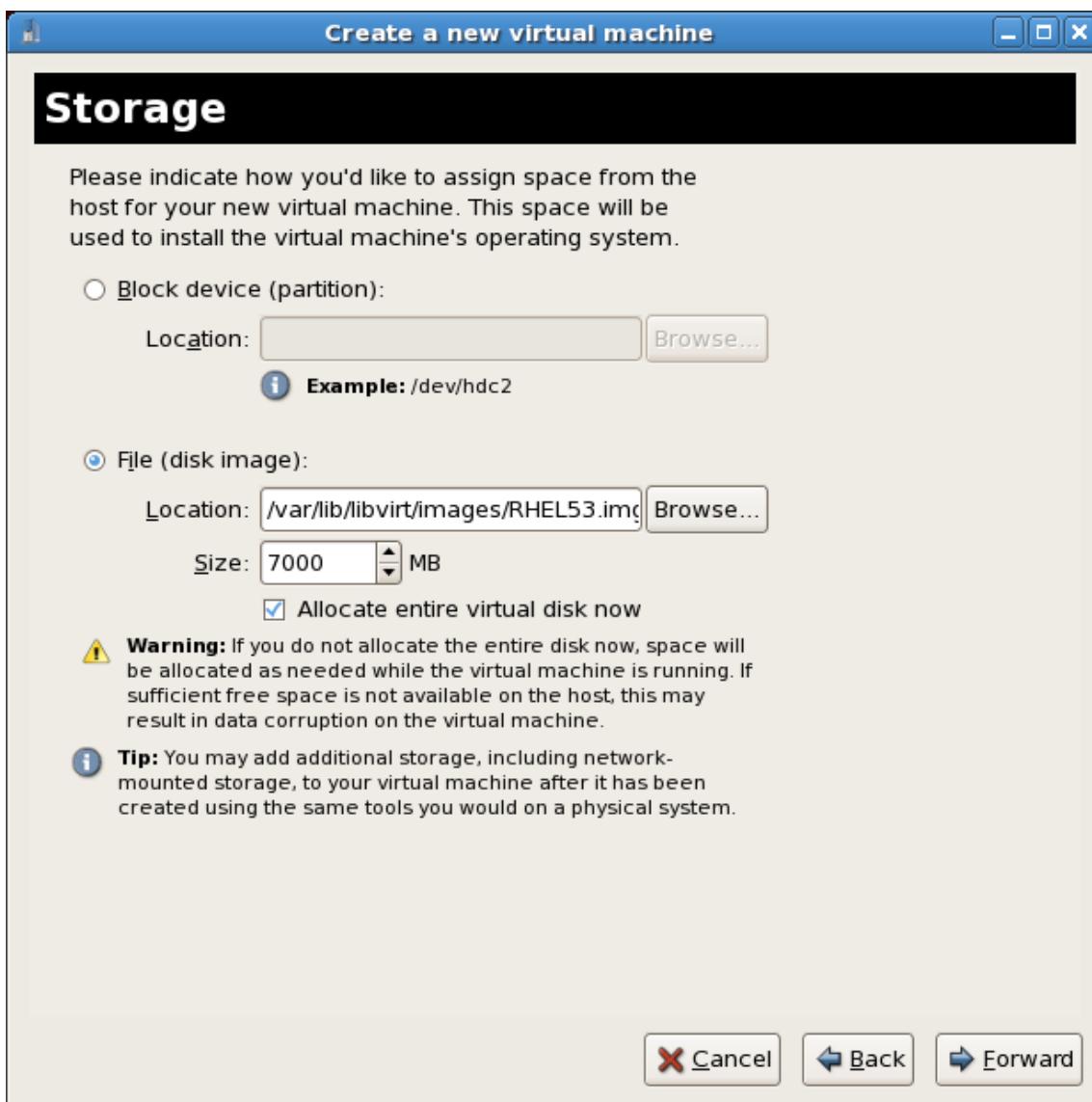
9. Storage setup

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

All image files are stored in the **/var/lib/libvirt/images/** directory by default. In the default configuration, other directory locations for file-based images are prohibited by SELinux. If you use a different directory you must label the new directory according to SELinux policy. See [Section 19.2, “SELinux and virtualization”](#) for details.

Your guest storage image should be larger than the size of the installation, any additional packages and applications, and the size of the guest's 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

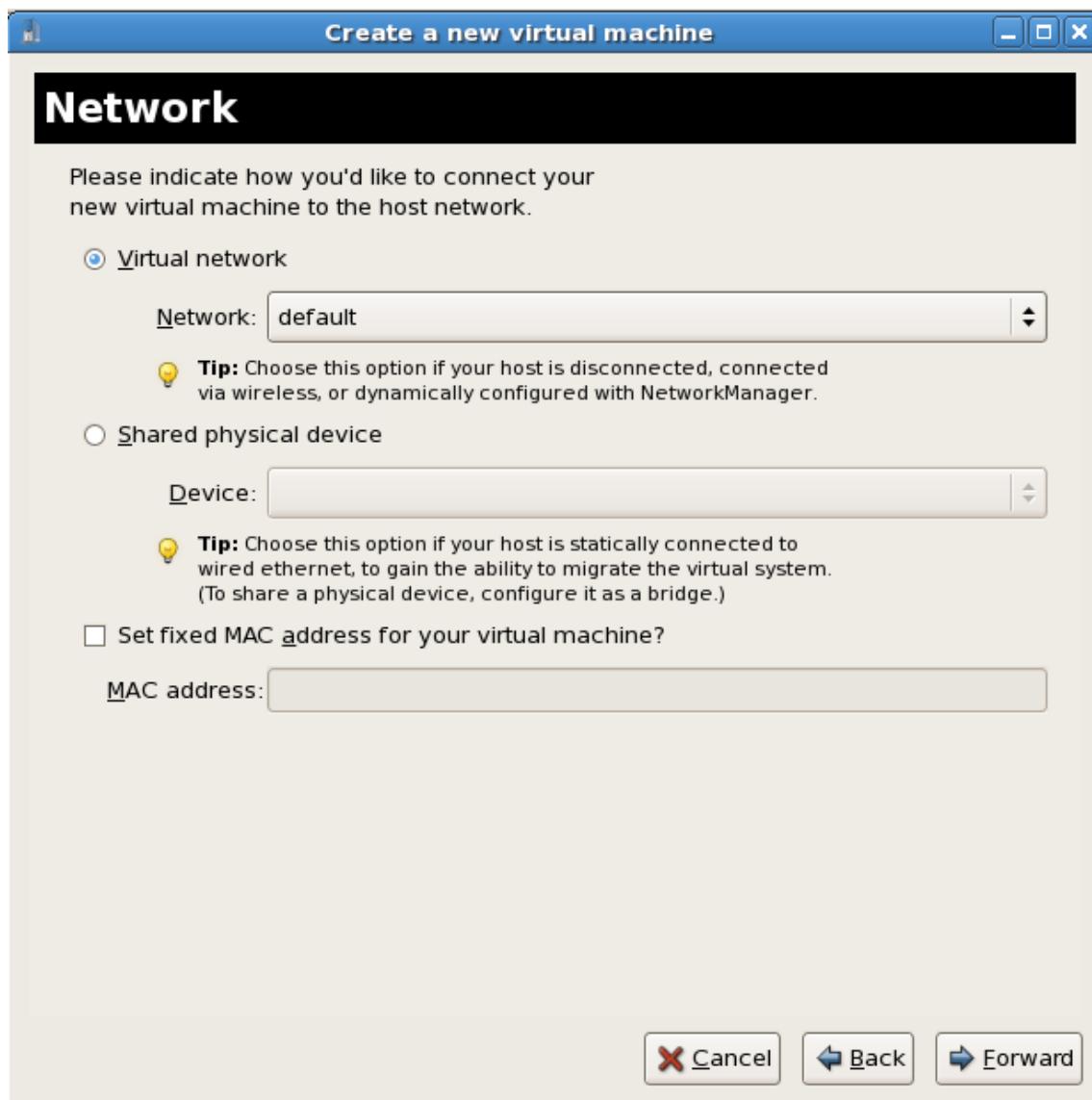
It is recommended that you use the default directory for virtual machine images, `/var/lib/libvirt/images/`. If you are using a different location (such as `/images/` in this example) make sure it is labeled according to SELinux policy before you continue with the installation. See [Section 19.2, “SELinux and virtualization”](#) for details.

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 guest. Use the virtual network option for wireless networks.

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



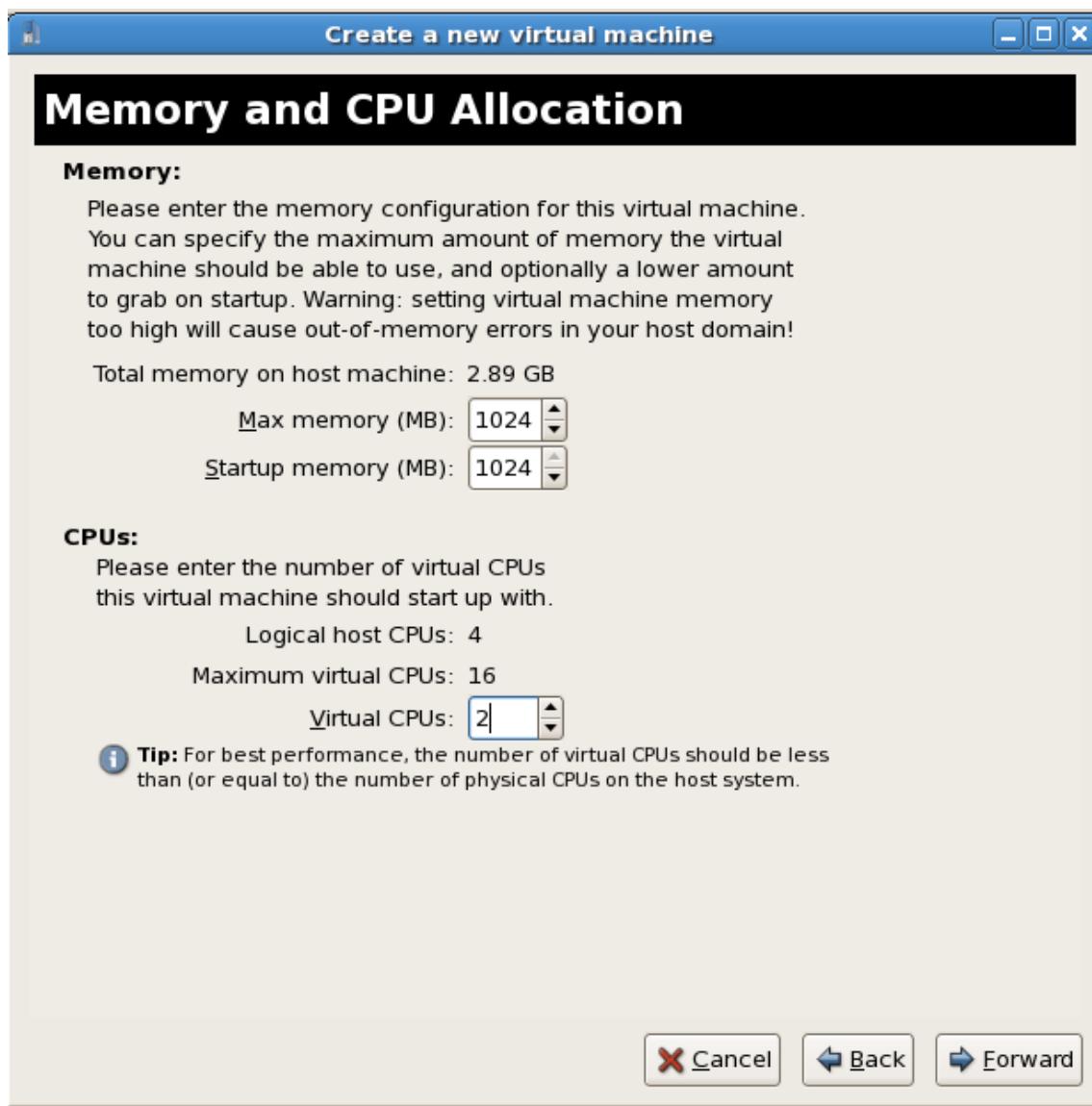
Press **Forward** to continue.

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 systems 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 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 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.

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 8, Guest operating system installation procedures](#) contains step-by-step instructions to installing a variety of common operating systems.

7.3. 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'.



Important

Prior to the release of Red Hat Enterprise Linux 5.9, a segmentation fault can occur when the bridge name contains only uppercase characters. Please upgrade to 5.9 or newer if uppercase names are required.

- 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
installation     8000.001320f76e8e      STP enabled
virbr0          8000.000000000000      no
                                         interfaces
                                         eth1
```

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
```

Note

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 restart
```

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

PXE installation with virt-install

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

Example 7.3. PXE installation with virt-install

```
# 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 \
```

PXE installation with virt-manager

The steps below are the steps that vary from the standard virt-manager installation procedures. For the standard installations see [Chapter 8, Guest operating system installation procedures](#).

1.

Select PXE

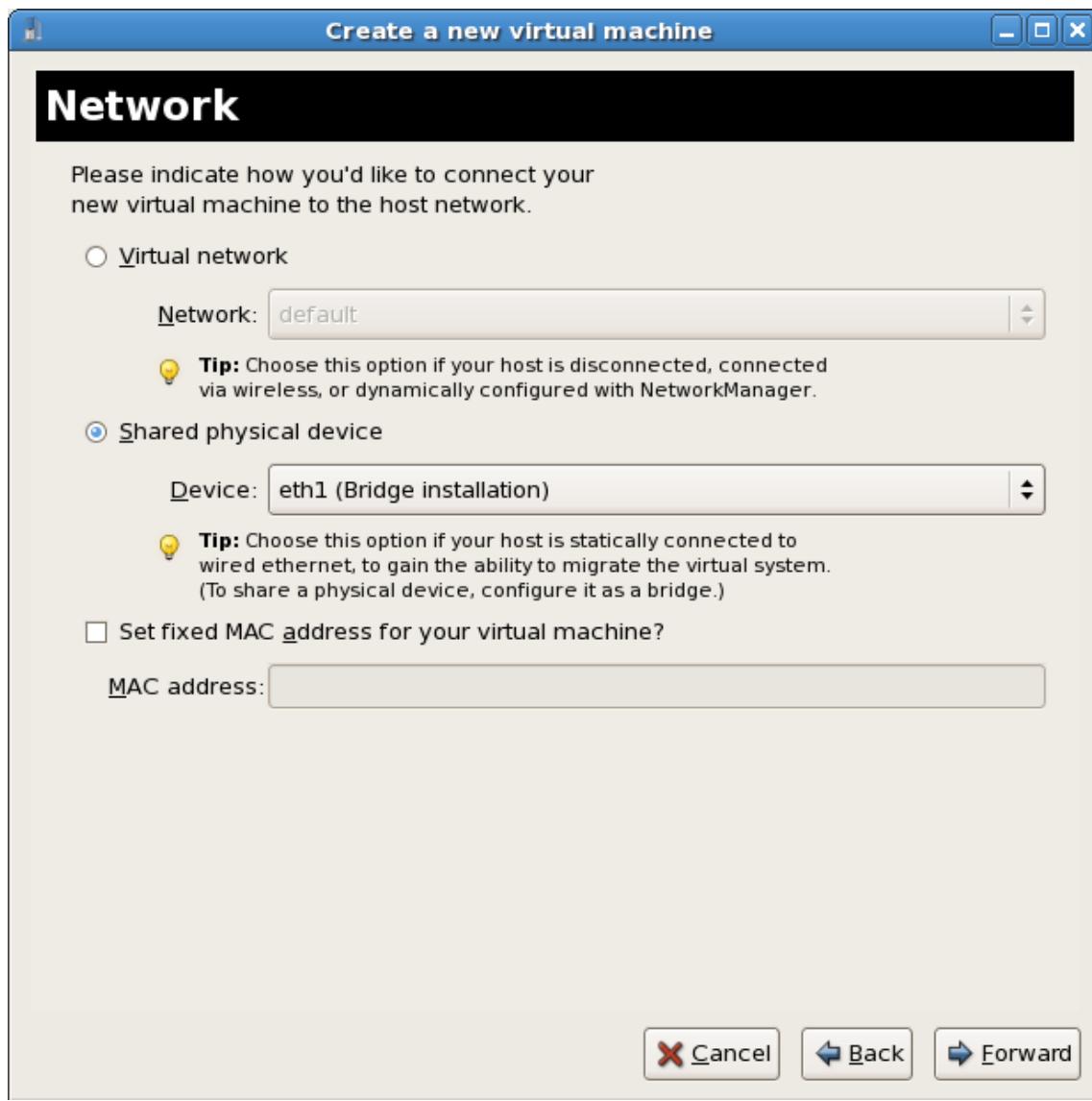
Select PXE as the installation method.



2.

Select the bridge

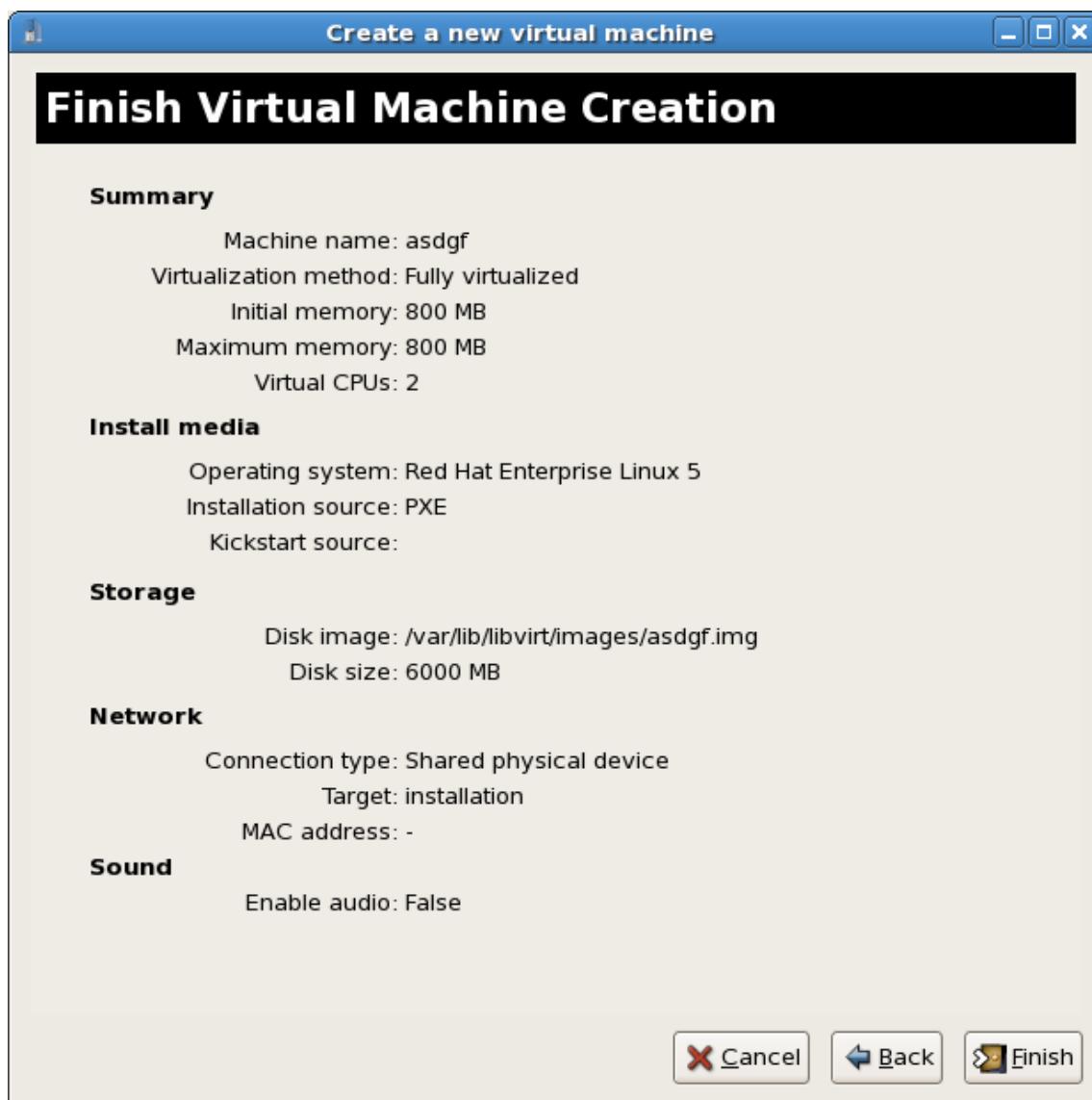
Select **Shared physical device** and select the bridge created in the previous procedure.



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.

Chapter 8. Guest operating system installation procedures

This chapter covers how to install various guest operating systems in a virtualized environment on Red Hat Enterprise Linux. To understand the basic processes, see [Chapter 7, Guest installation overview](#).



Important

When installing a Red Hat Enterprise Linux guest, the installer will ask to perform an integrity check on your installation source (CD/DVD media, or ISO file). If you select to perform the check, once the media is tested and the installation continues, you may encounter a message that states: **The Red Hat Enterprise Linux Server CD was not found in any of your CDROM drives. Please insert the Red Hat Enterprise Linux Server CD and press OK to retry.**

This behavior is intentional, provided as a convenience to make sure media is ejected in case a CD install (requiring multiple discs/images) is being performed as opposed to a DVD install.

To proceed past this message, make sure you either insert the next CD, or edit the guest's XML file specifying the next ISO file (or re-insert the DVD media). Next, run **virsh update-device Guest1 ~/Guest1.xml** (substituting your guest's name and XML file), and select OK to continue past this step.

8.1. Installing Red Hat Enterprise Linux 5 as a para-virtualized guest

This section describes how to install Red Hat Enterprise Linux 5 as a para-virtualized guest. Para-virtualization is a faster than full virtualization and supports all of the advantages of full virtualization. Para-virtualization requires a special, supported kernel, the **kernel-xen** kernel.



Important

Para-virtualization only works with the Xen hypervisor. Para-virtualization does not work with the KVM hypervisor.

Ensure you have root access before starting the installation.

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

Create para-virtualized Red Hat Enterprise Linux 5 guests using virt-manager or virt-install. For instructions on **virt-manager**, see [Section 7.2, “Creating guests with virt-manager”](#).

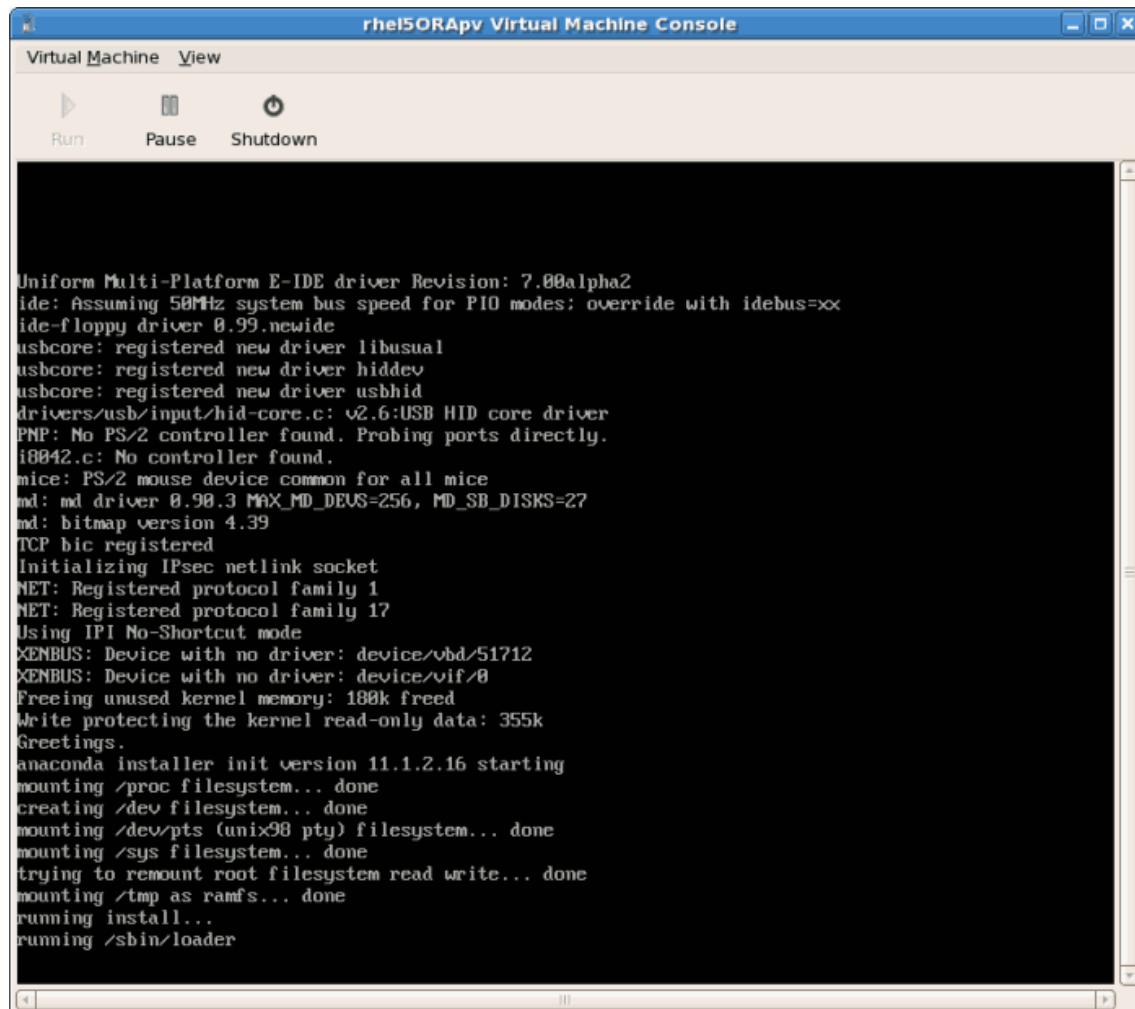
Create a para-virtualized guest with the command line based **virt-install** tool. The **--vnc** option shows the graphical installation. The name of the guest in the example is *rhel5PV*, the disk image file is *rhel5PV.dsk* and a local mirror of the Red Hat Enterprise Linux 5 installation tree is *ftp://10.1.1.1/trees/RHEL5-B2-Server-i386/*. Replace those values with values accurate for your system and network.

```
# virt-install -n rhel5PV -r 500 \
-f /var/lib/libvirt/images/rhel5PV.dsk -s 3 --vnc -p \
-l ftp://10.1.1.1/trees/RHEL5-B2-Server-i386/
```

Note

Red Hat Enterprise Linux can be installed without a graphical interface or manual input. Use Kickstart files to automate the installation process.

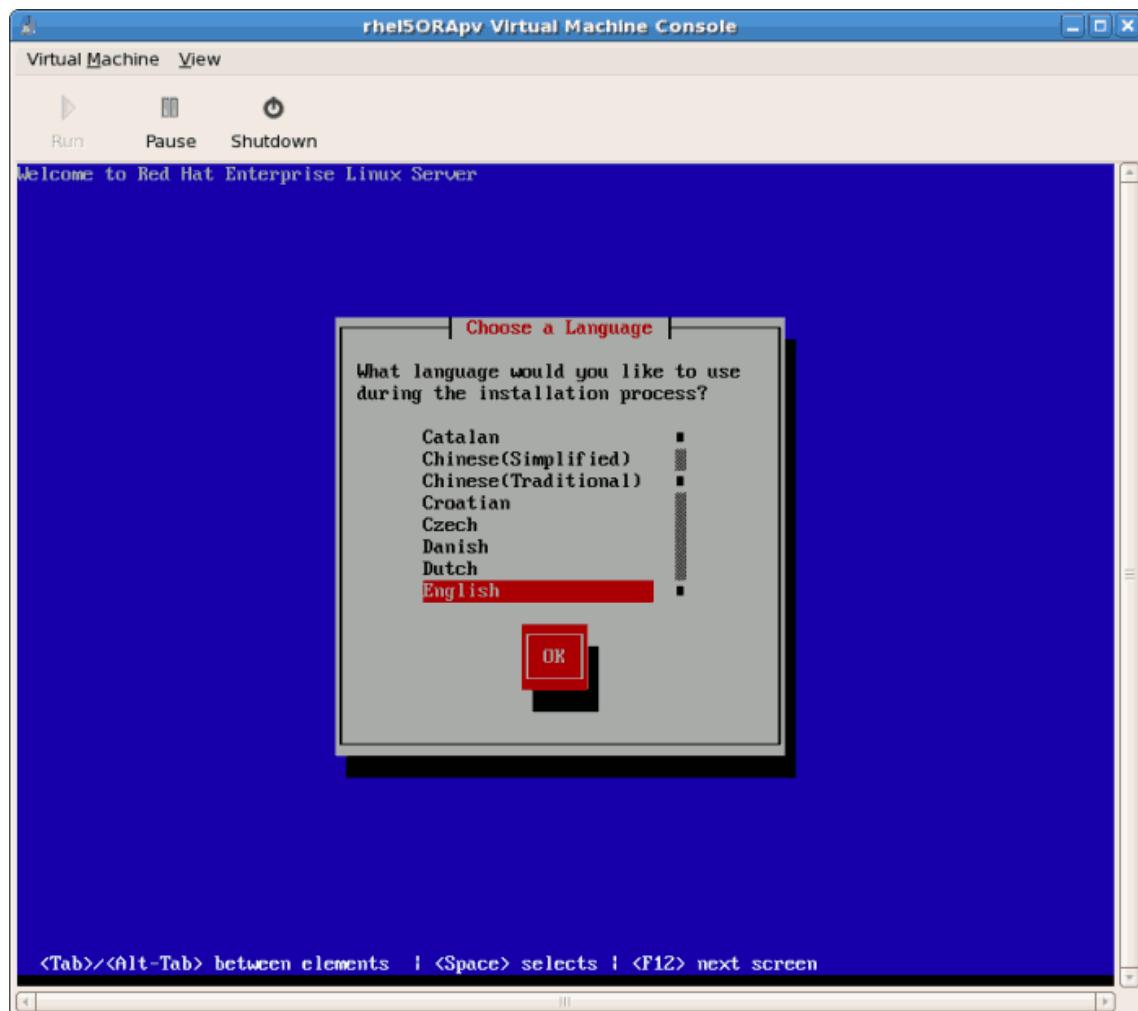
Using either method opens this window, displaying the initial boot phases of your guest:



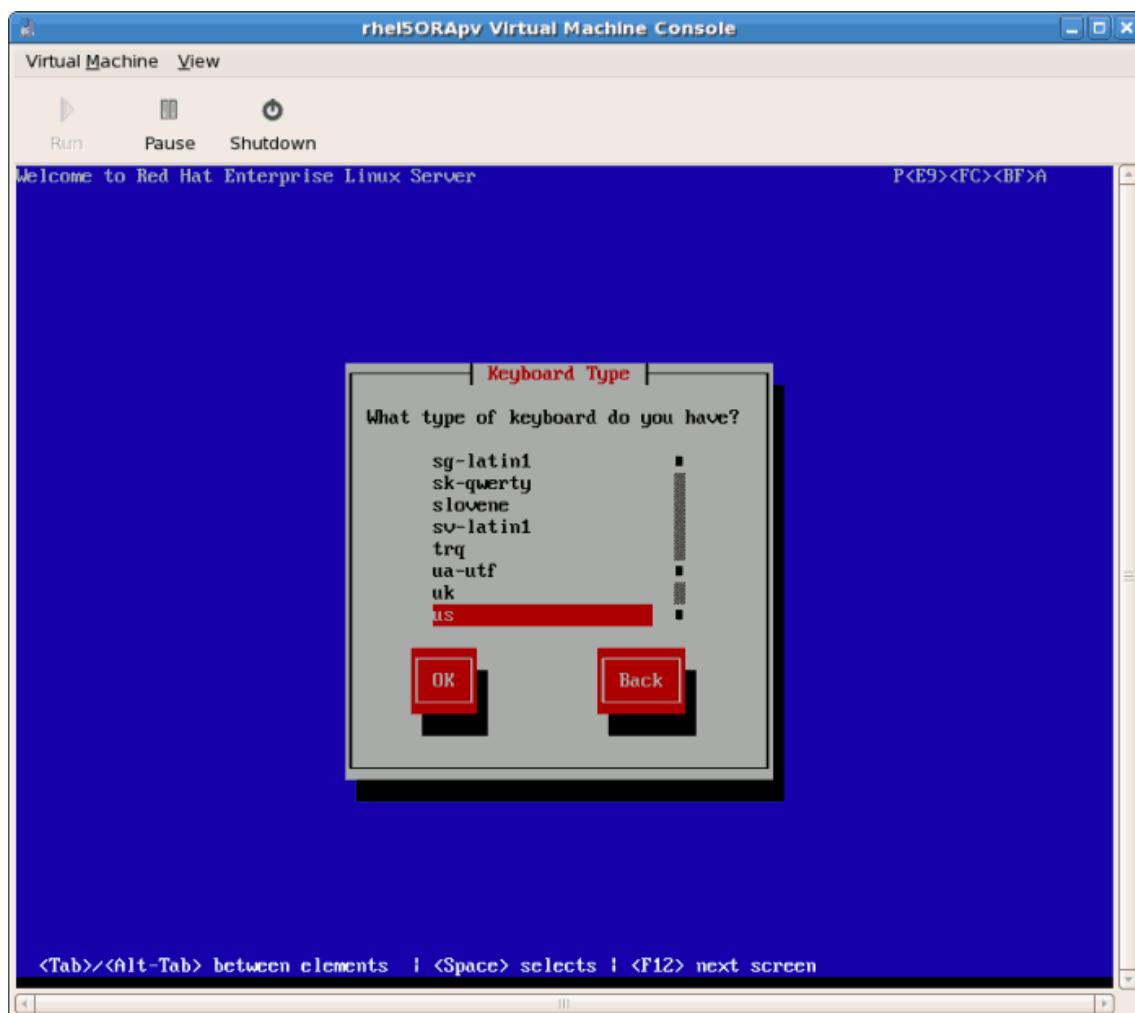
After your guest has completed its initial boot, the standard installation process for Red Hat Enterprise Linux starts. For most systems the default answers are acceptable.

Procedure 8.1. Para-virtualized Red Hat Enterprise Linux guest installation procedure

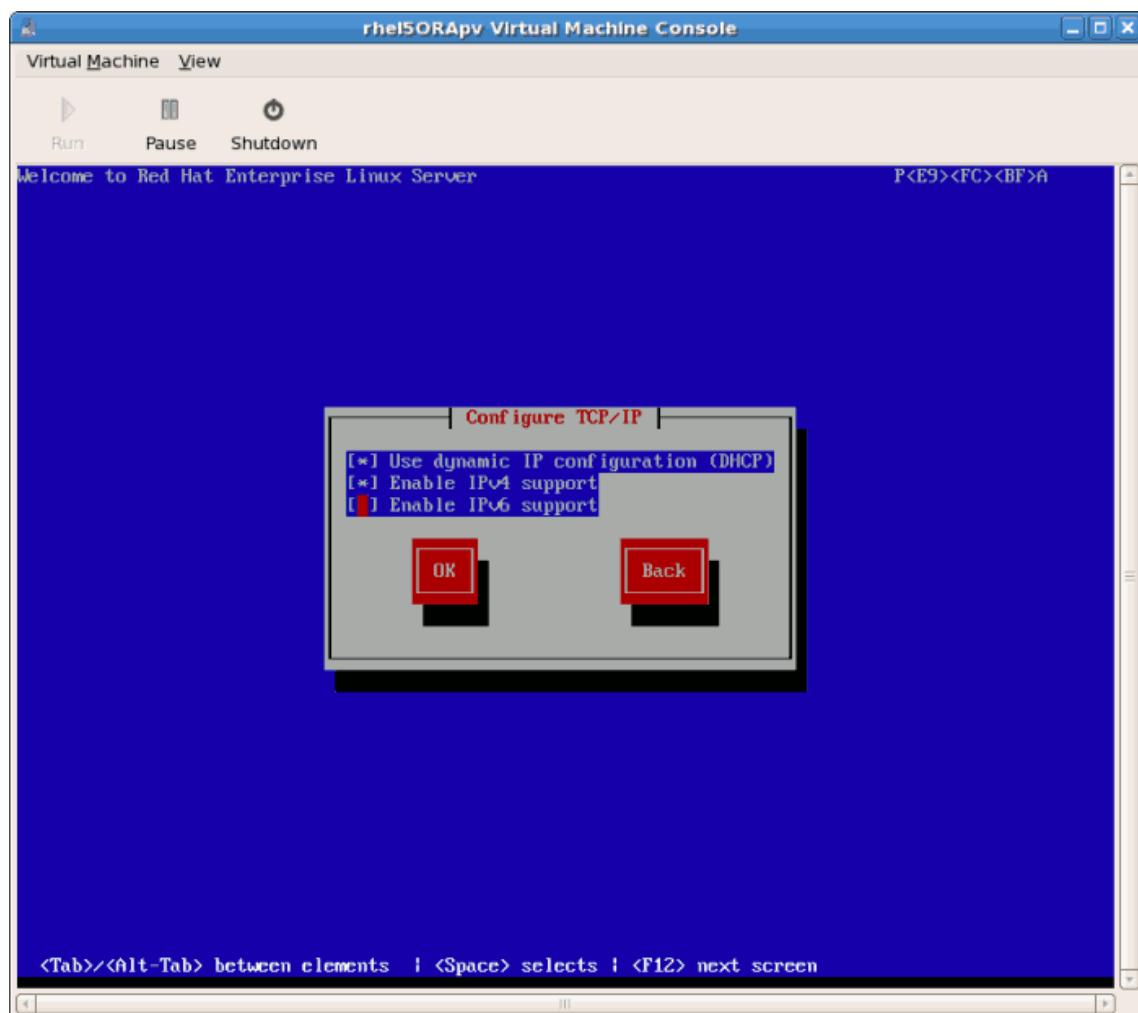
1. Select the language and click **OK**.



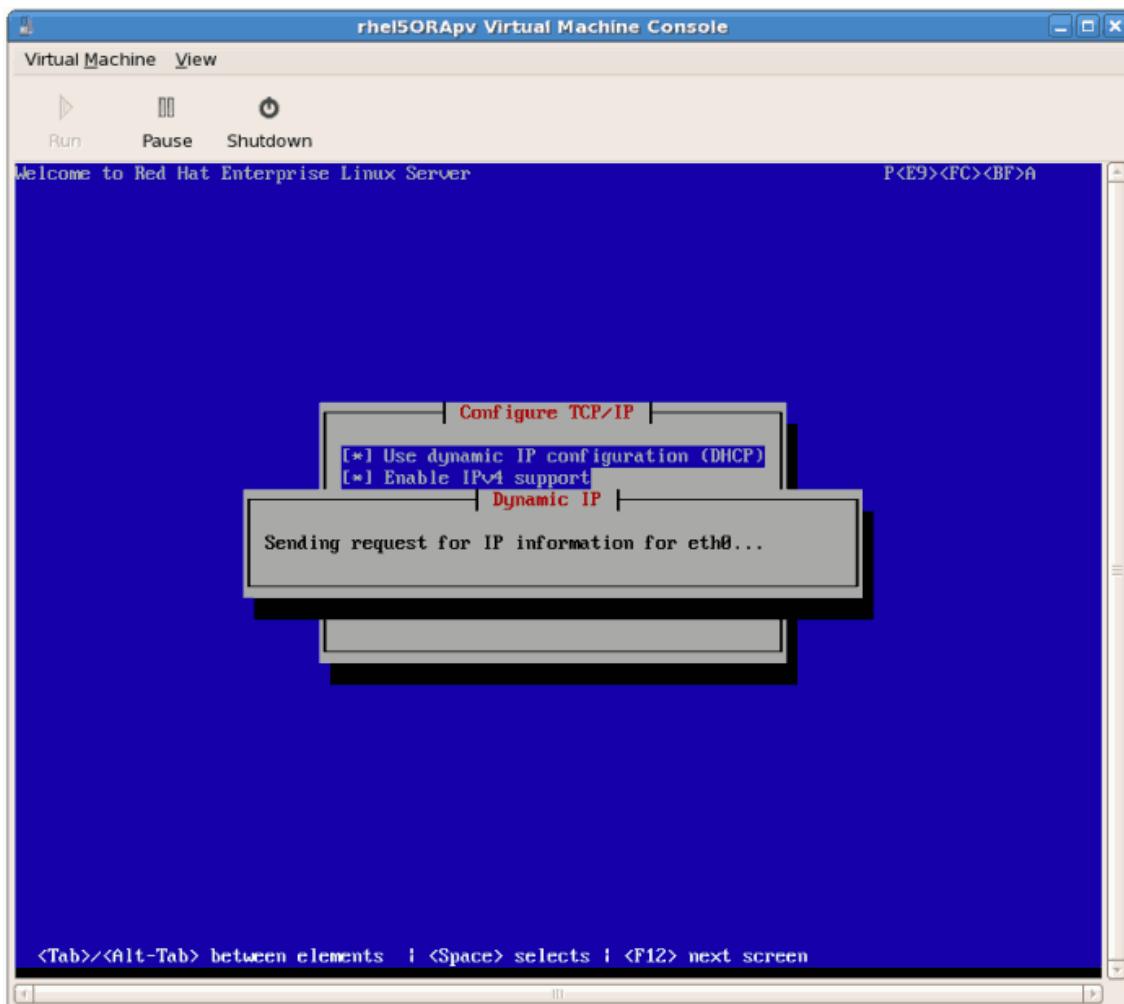
2. Select the keyboard layout and click **OK**.



3. Assign the guest's network address. Choose to use **DHCP** (as shown below) or a static IP address:

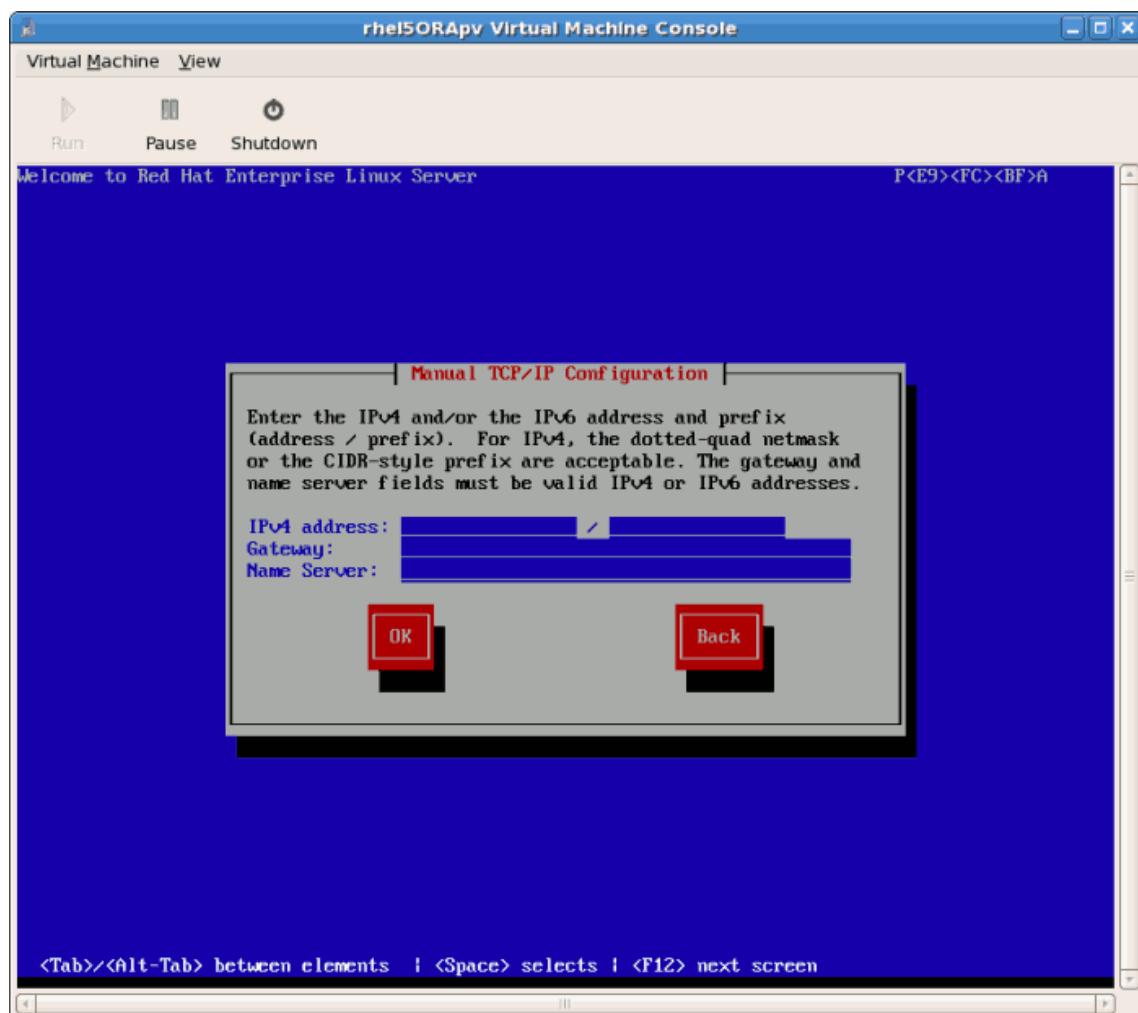


4. If you select DHCP the installation process will now attempt to acquire an IP address:

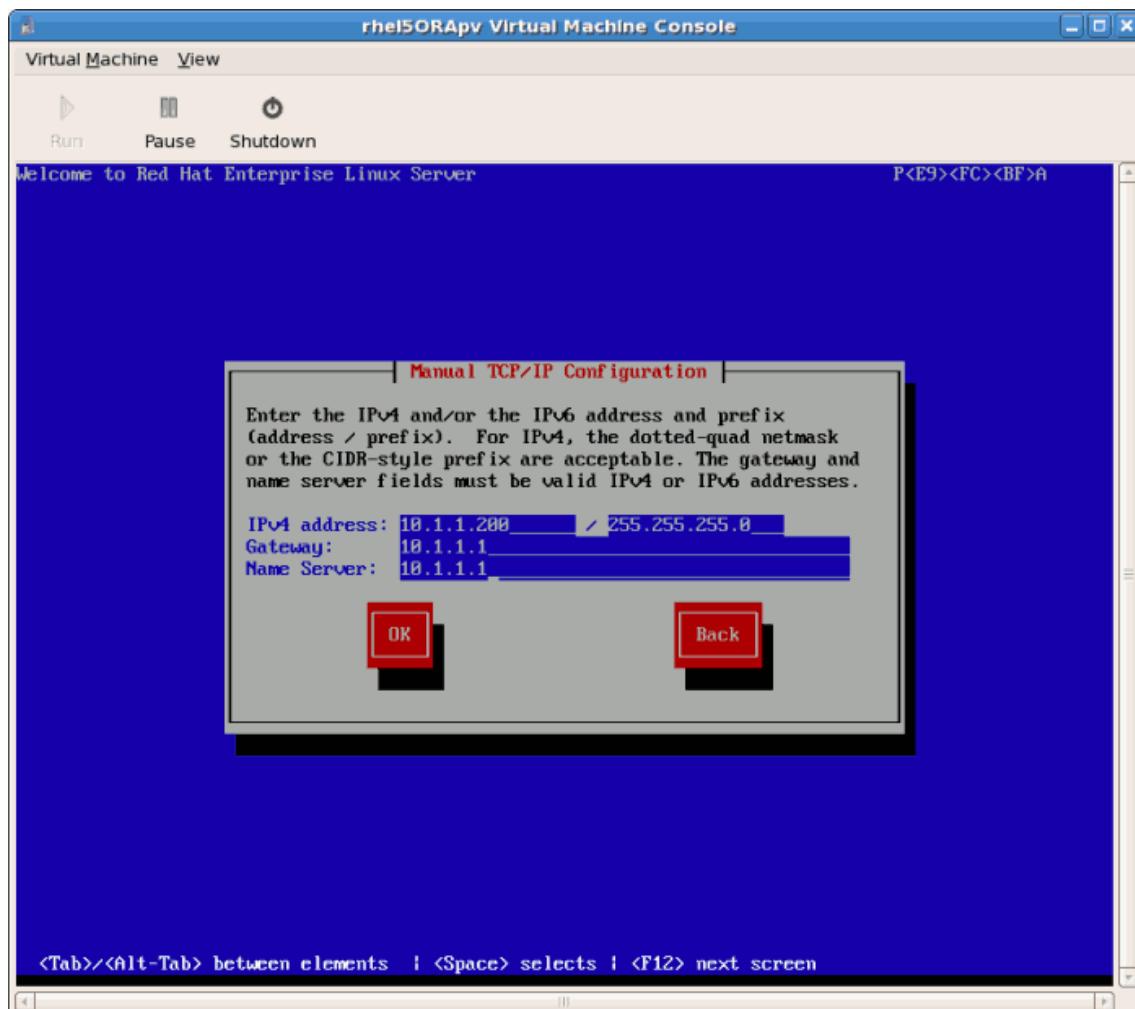


5. If you chose a static IP address for your guest this prompt appears. Enter the details on the guest's networking configuration:
 - a. Enter a valid IP address. Ensure the IP address you enter can reach the server with the installation tree.
 - b. Enter a valid Subnet mask, default gateway and name server address.

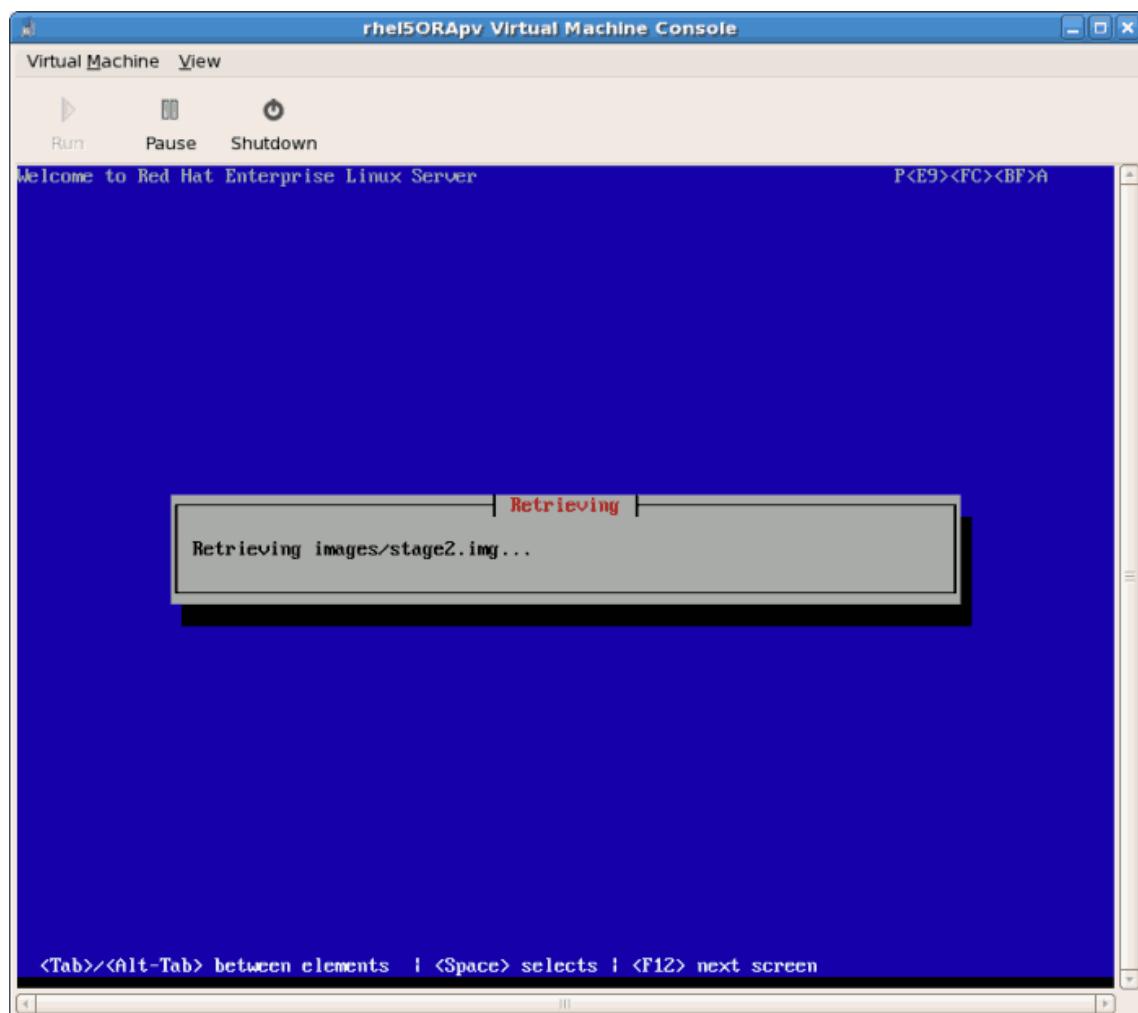
Select the language and click **OK**.



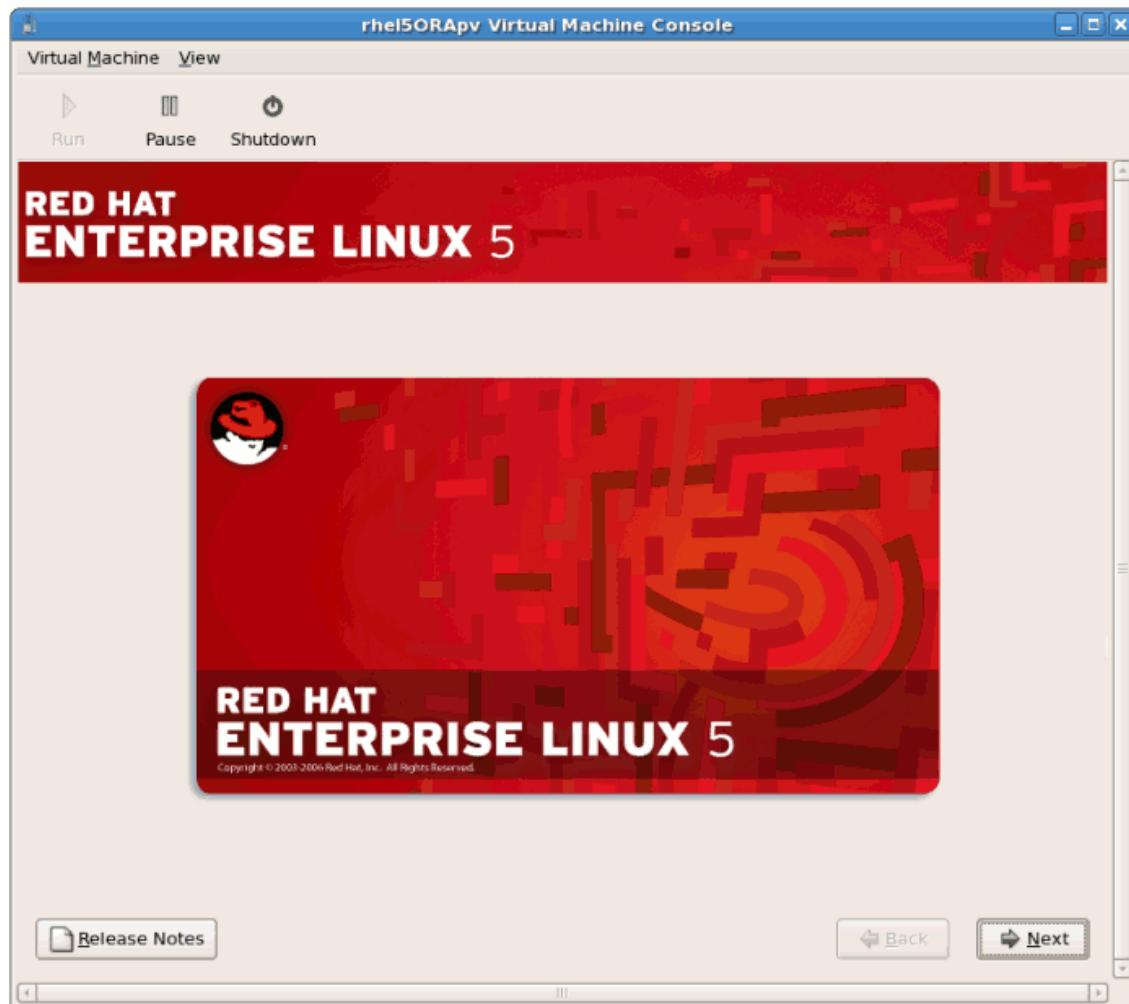
6. This is an example of a static IP address configuration:



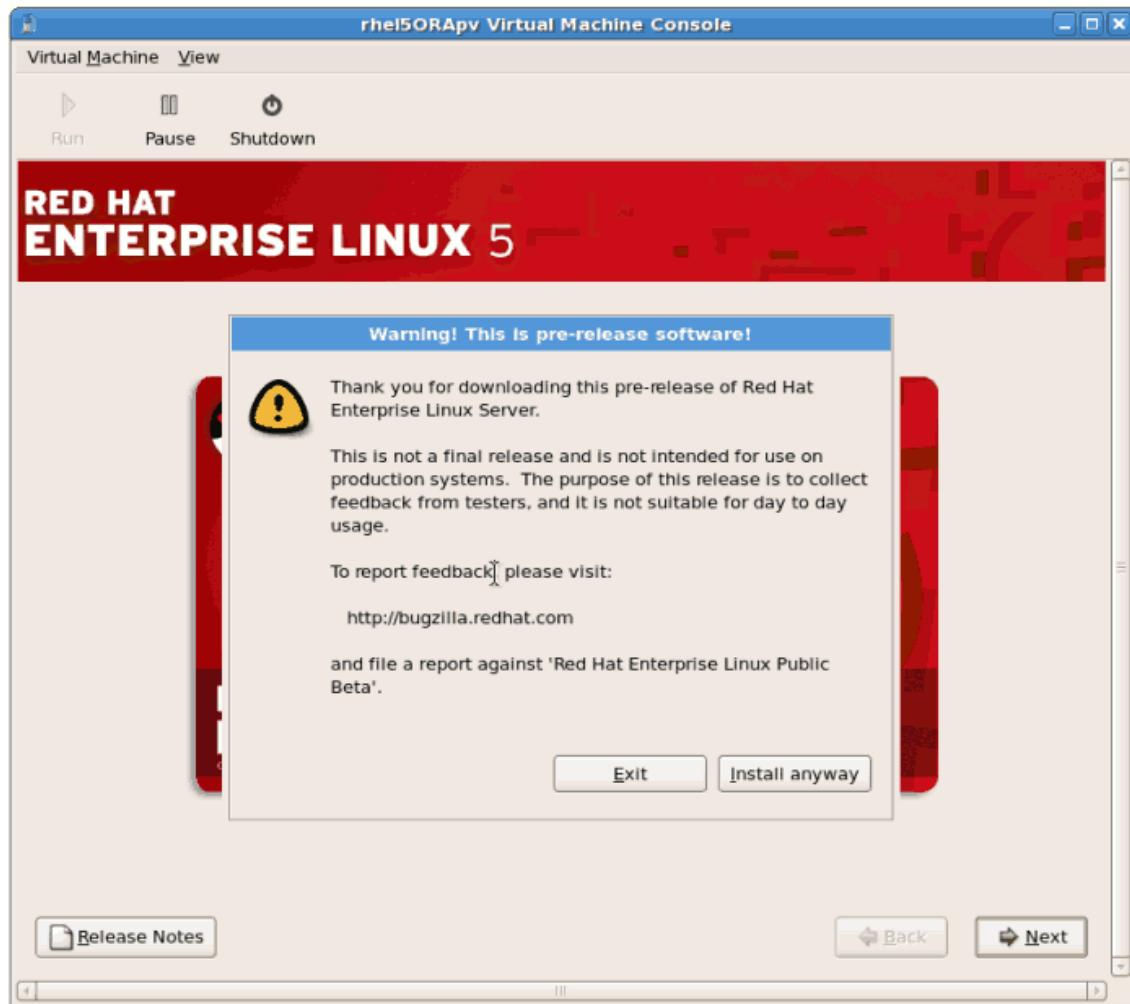
7. The installation process now retrieves the files it needs from the server:



Once the initial steps are complete the graphical installation process starts.

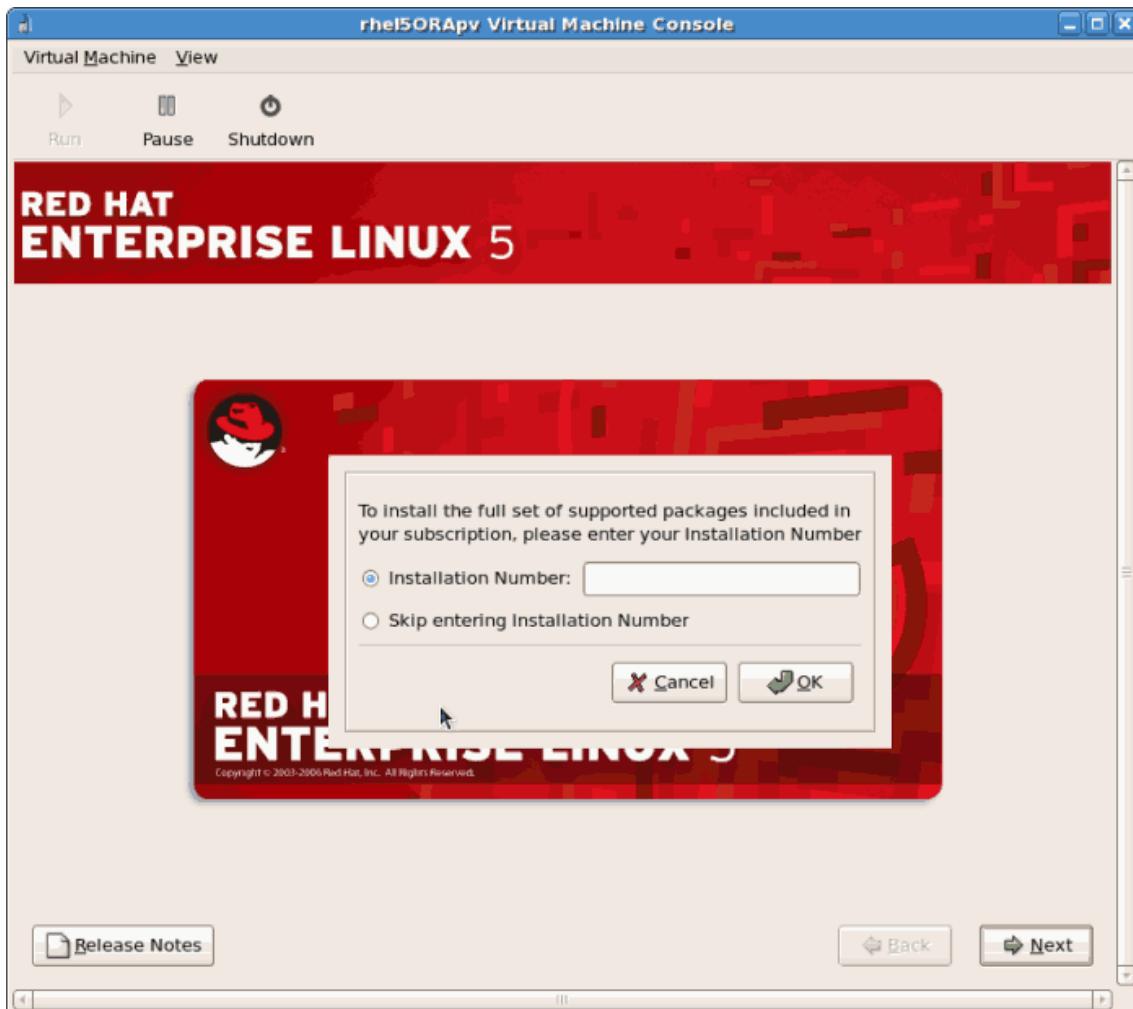


If you are installing a Beta or early release distribution confirm that you want to install the operating system. Click **Install Anyway**, and then click **OK**:



Procedure 8.2. The graphical installation process

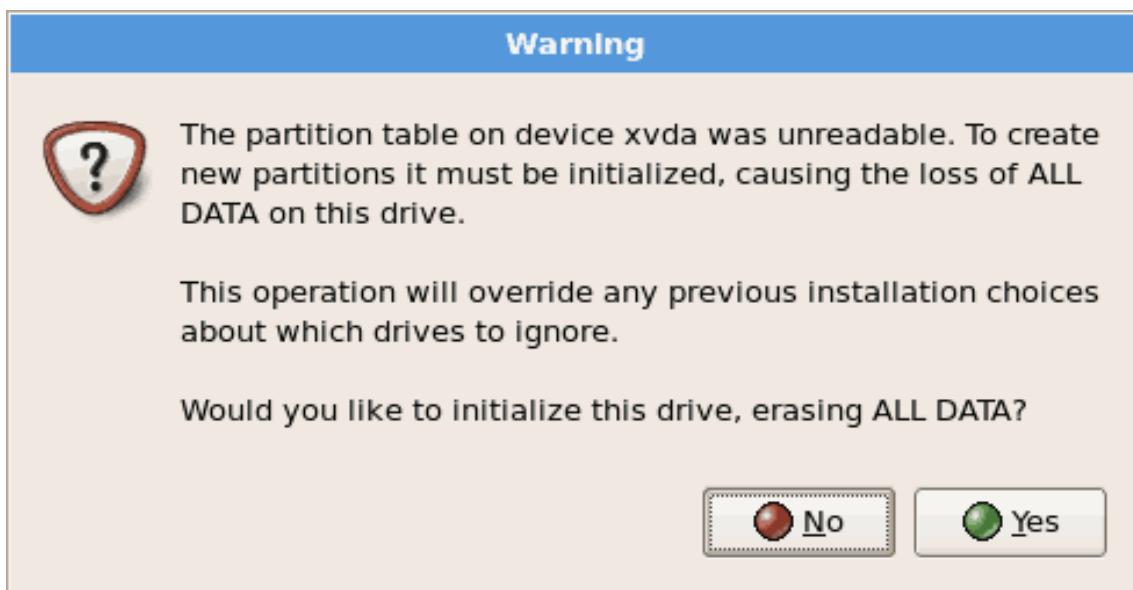
1. Enter a valid registration code. If you have a valid Red Hat subscription key please enter in the **Installation Number** field:



Note

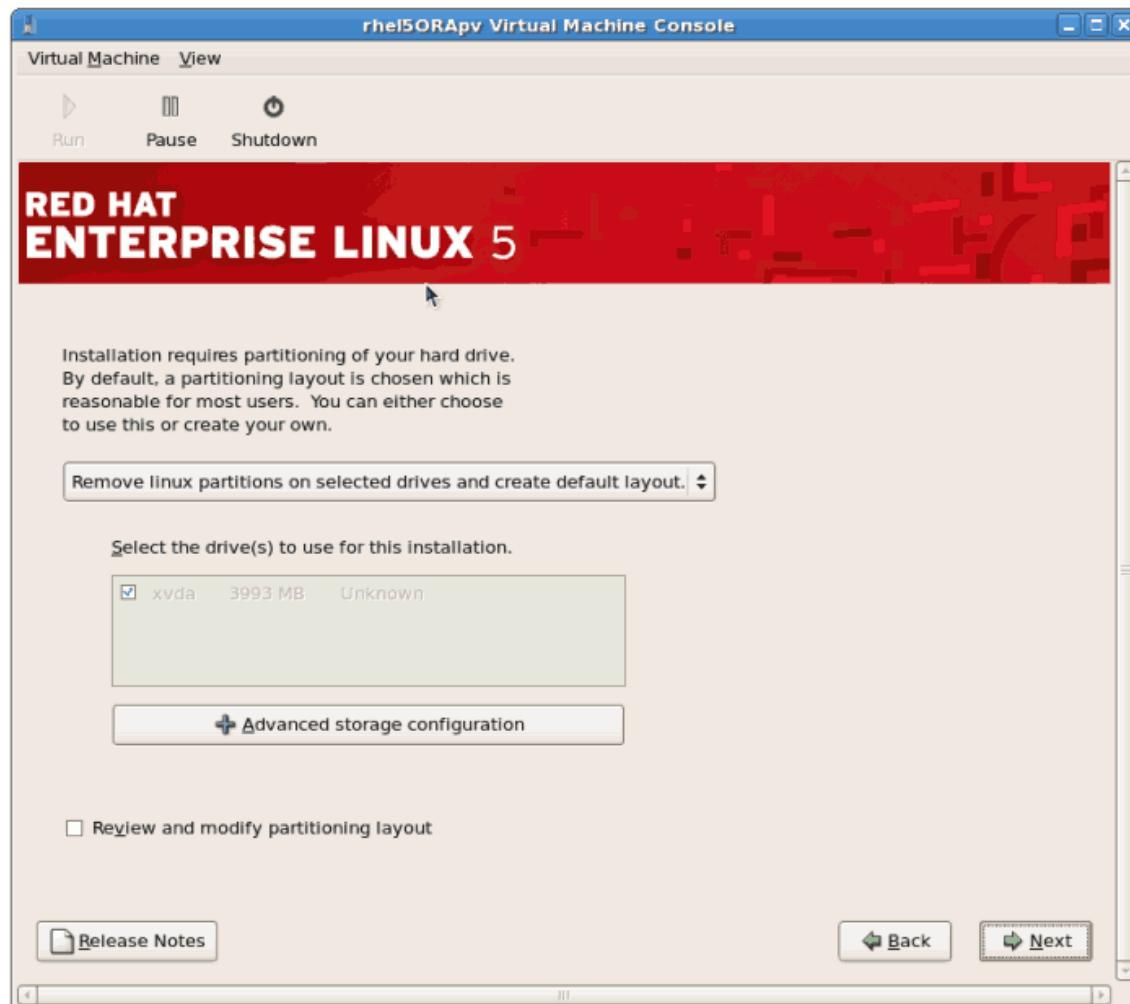
If you skip the registration step, confirm your Red Hat account details after the installation with the **rhn_register** command. The **rhn_register** command requires root access.

2. The installation prompts you to confirm erasure of all data on the storage you selected for the installation:



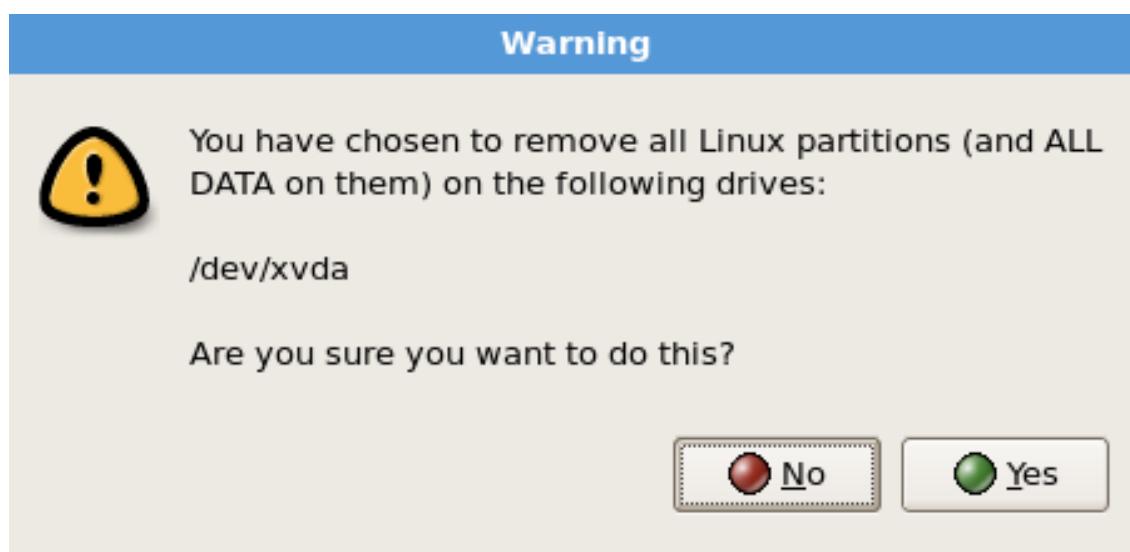
Click **Yes** to continue.

3. Review the storage configuration and partition layout. You can chose to select the advanced storage configuration if you want to use iSCSI for the guest's storage.



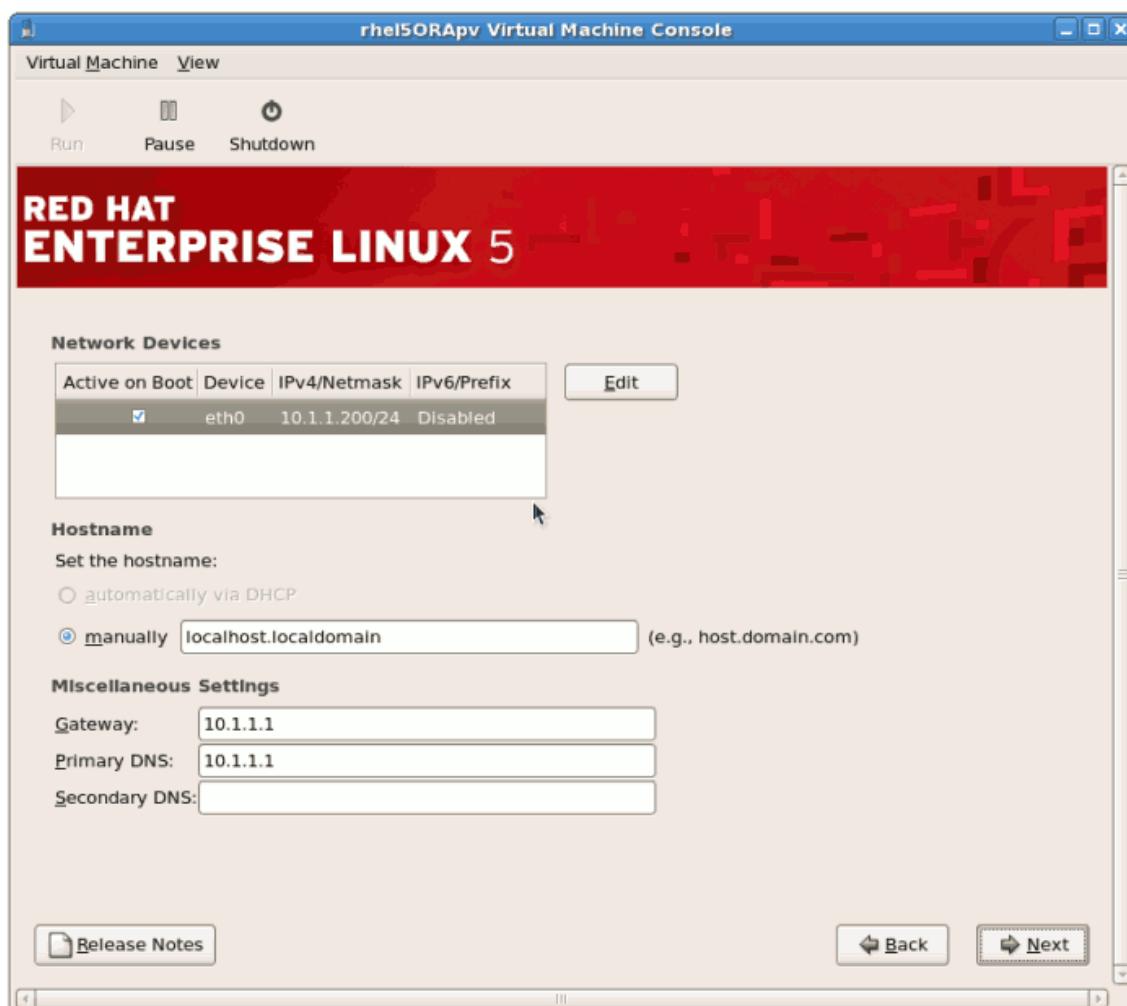
Make your selections then click **Forward**.

4. Confirm the selected storage for the installation.



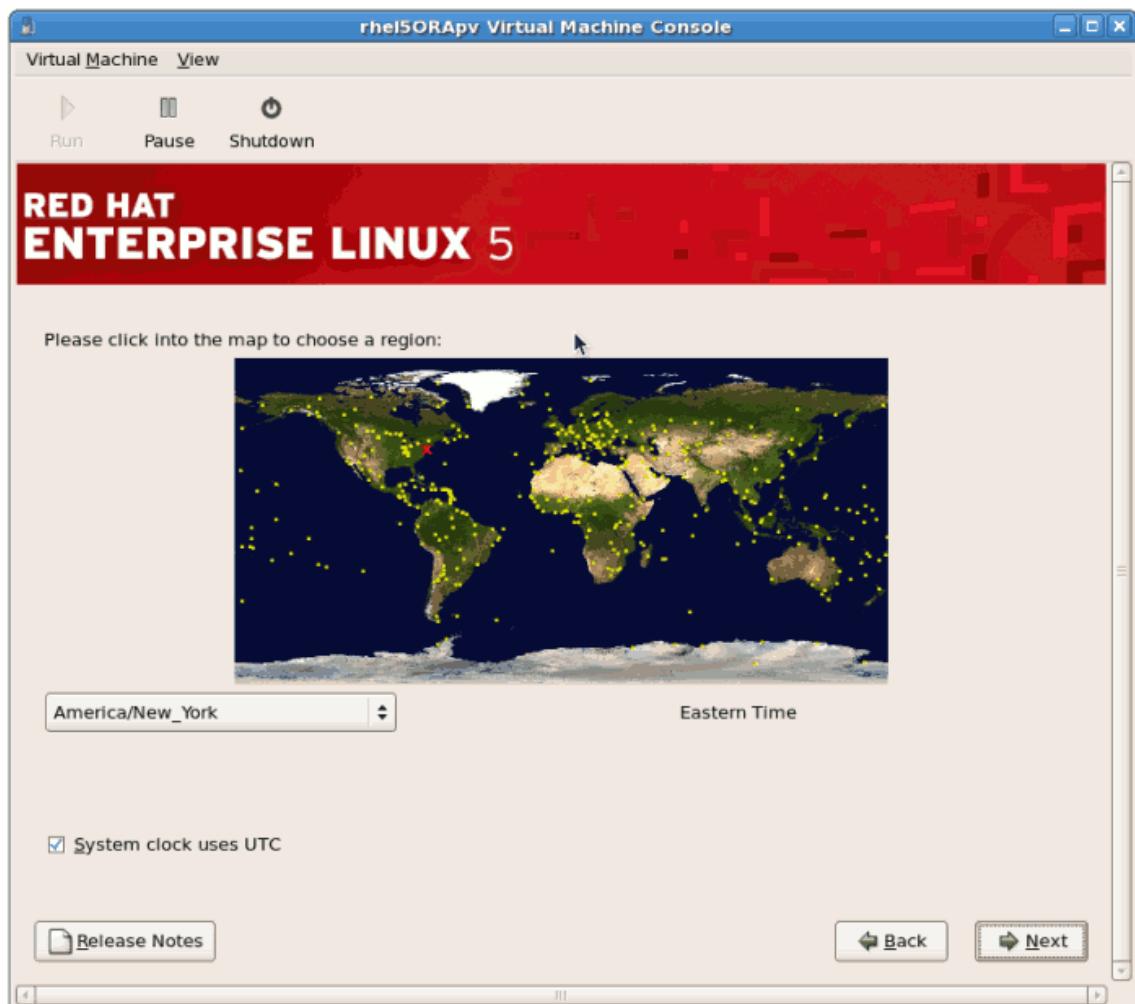
Click **Yes** to continue.

5. Configure networking and hostname settings. These settings are populated with the data entered earlier in the installation process. Change these settings if necessary.

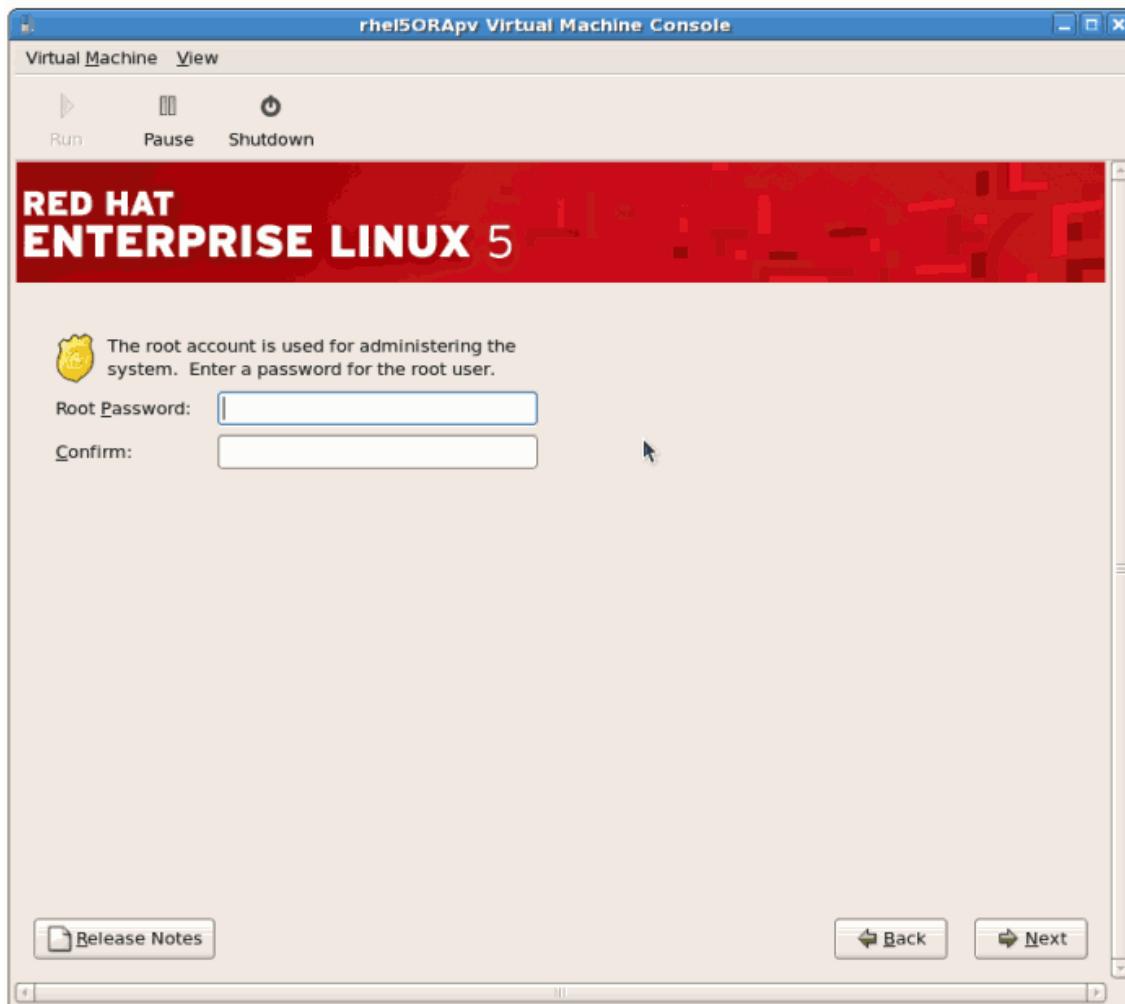


Click **OK** to continue.

6. Select the appropriate time zone for your environment.

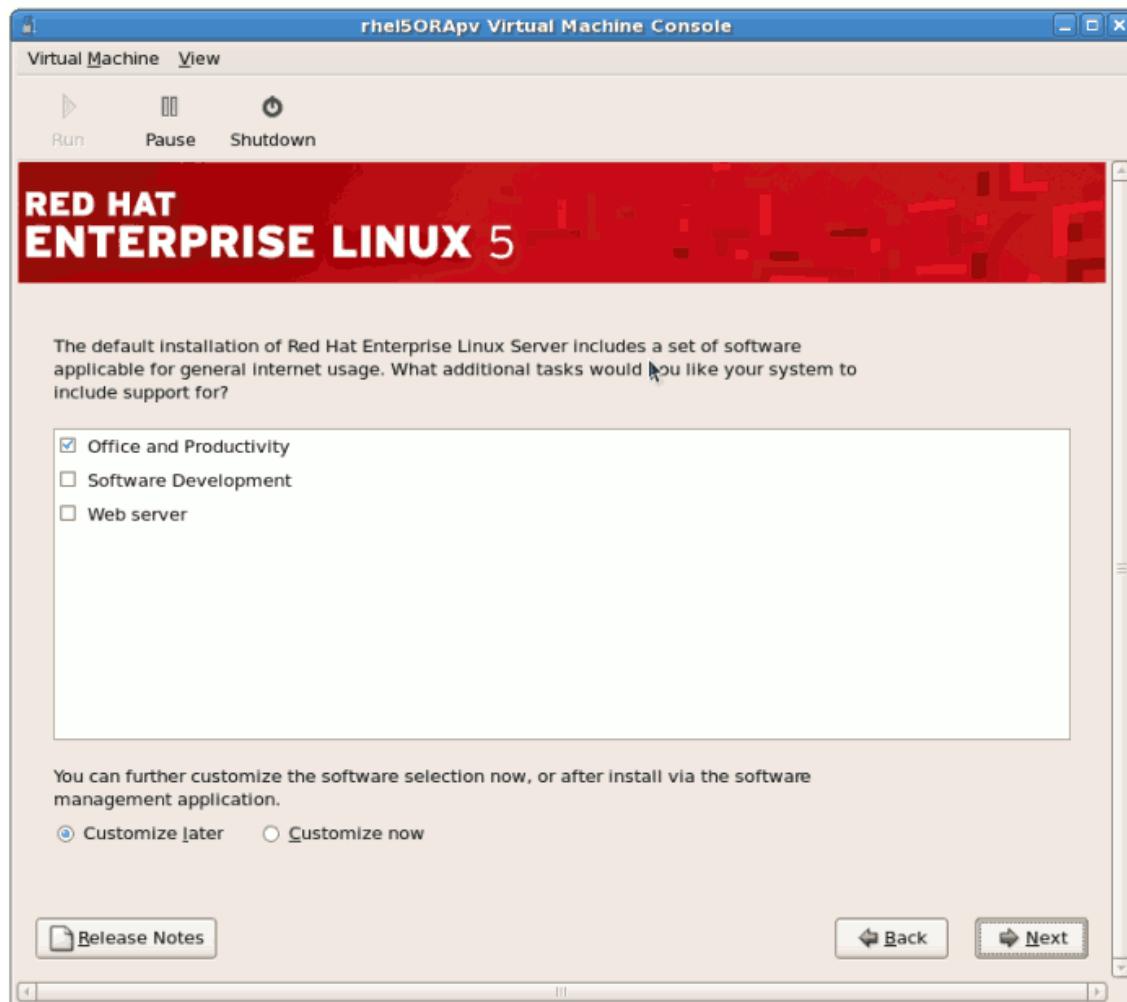


7. Enter the root password for the guest.



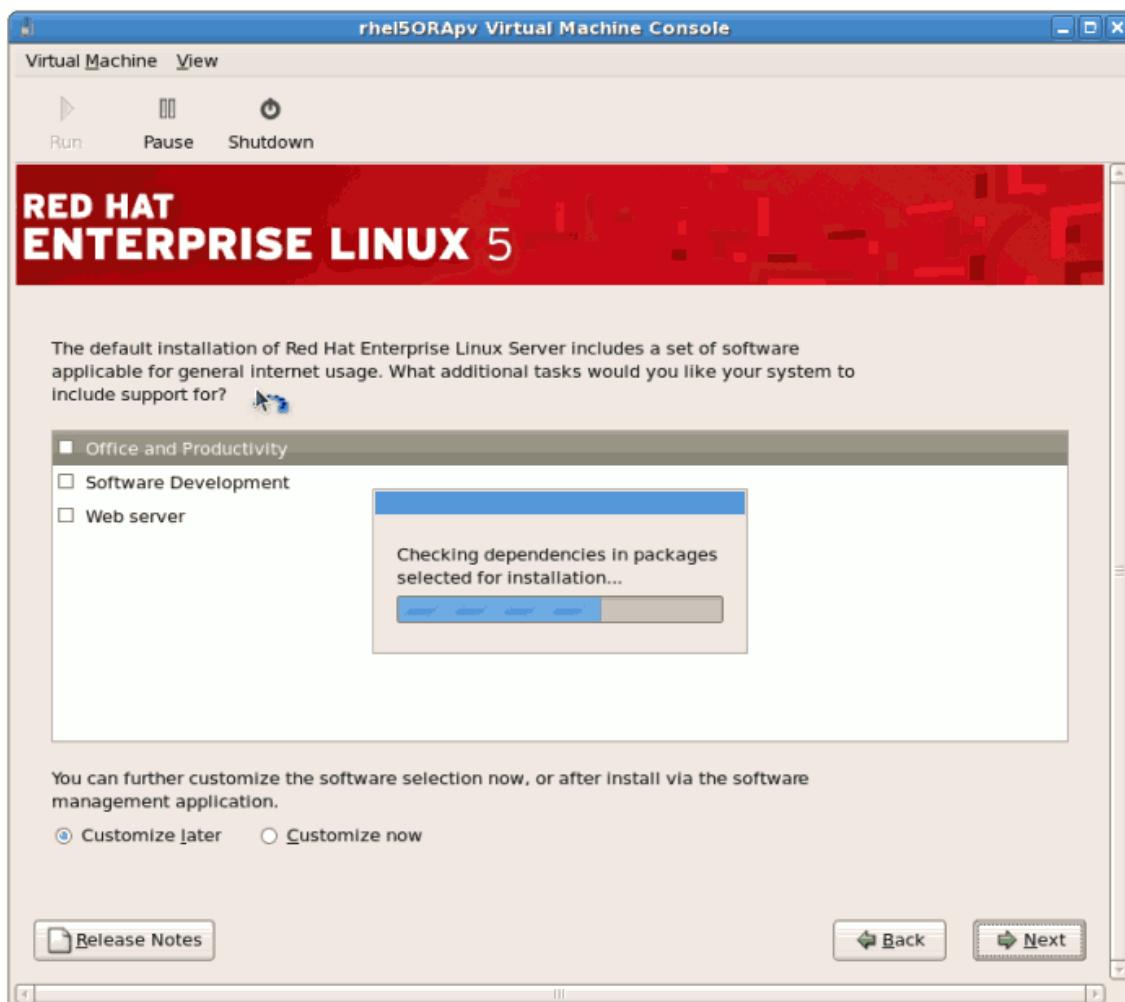
Click **Forward** to continue.

8. Select the software packages to install. Select the **Customize Now** button. You must install the **kernel-xen** package in the **System** directory. The **kernel-xen** package is required for para-virtualization.

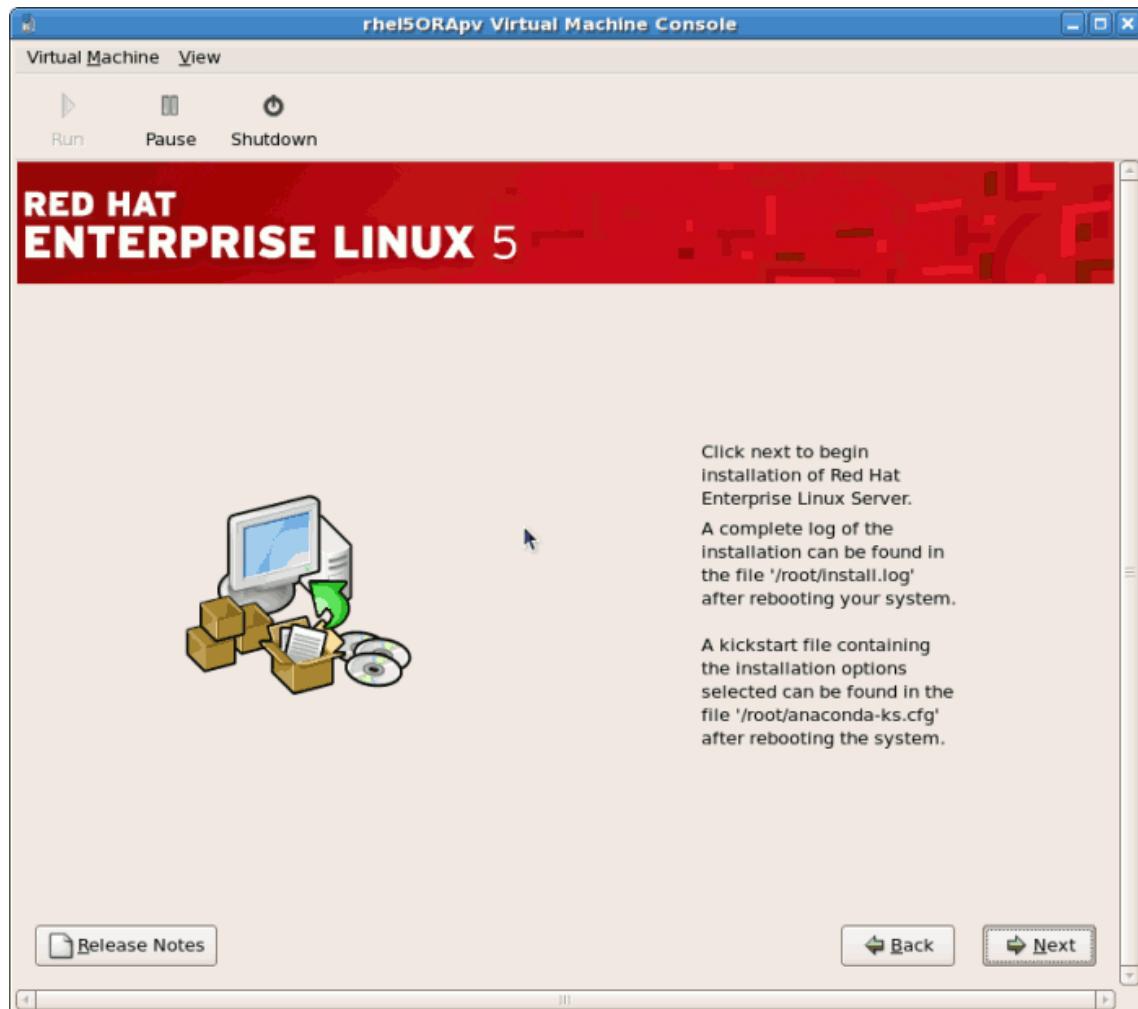


Click **Forward**.

9. Dependencies and space requirements are calculated.



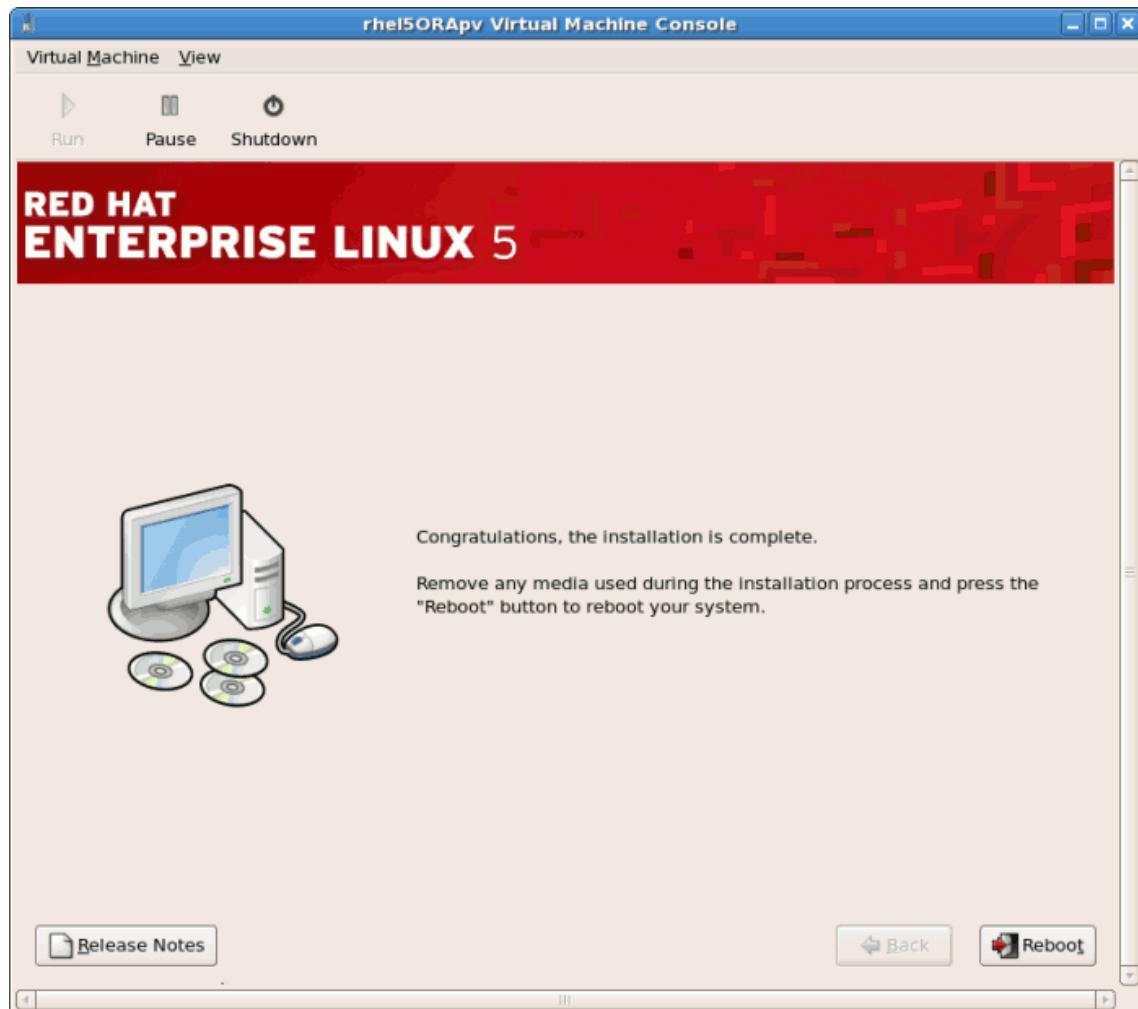
10. After the installation dependencies and space requirements have been verified click **Forward** to start the actual installation.



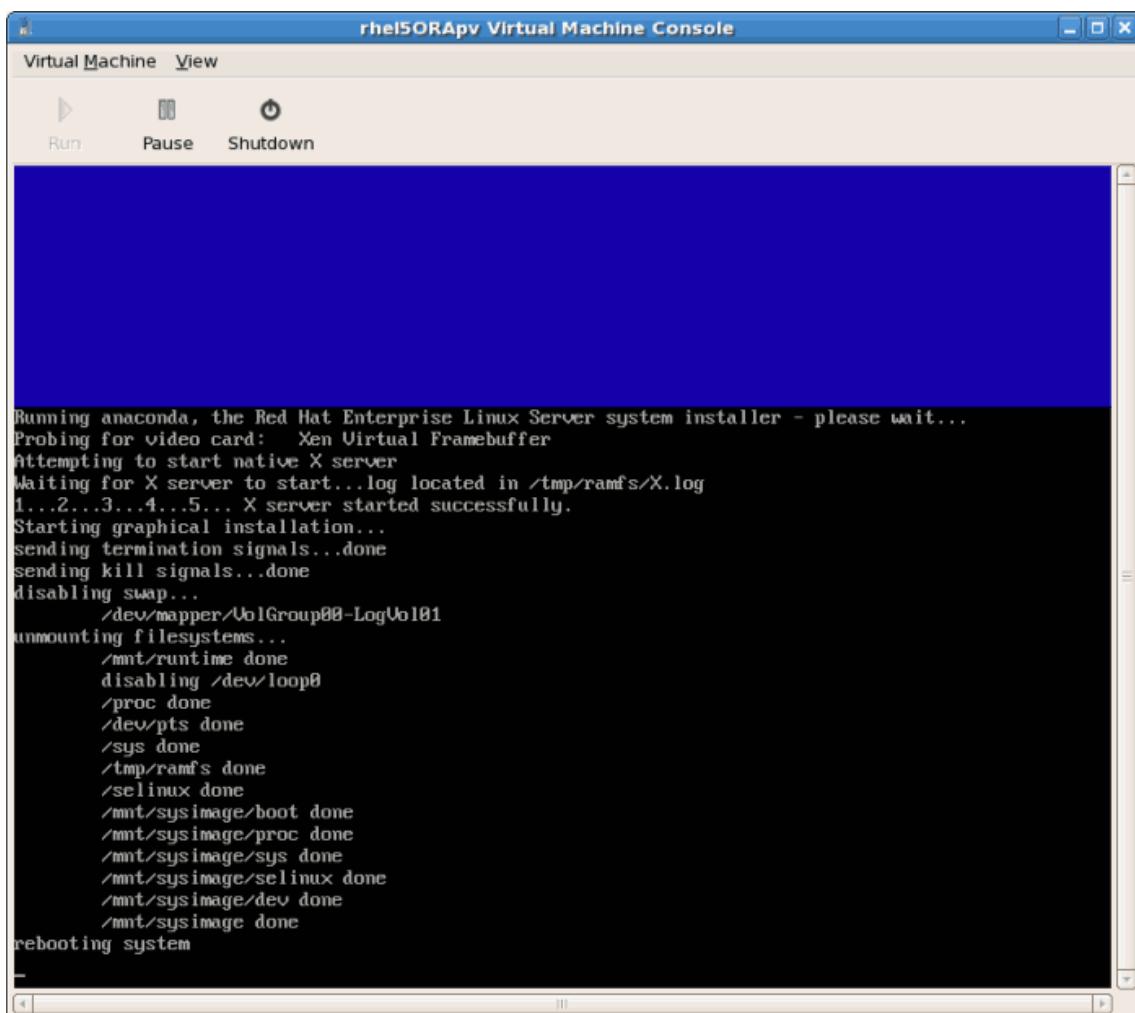
11. All of the selected software packages are installed automatically.



12. After the installation has finished reboot your guest:



13. The guest will not reboot, instead it will shutdown..



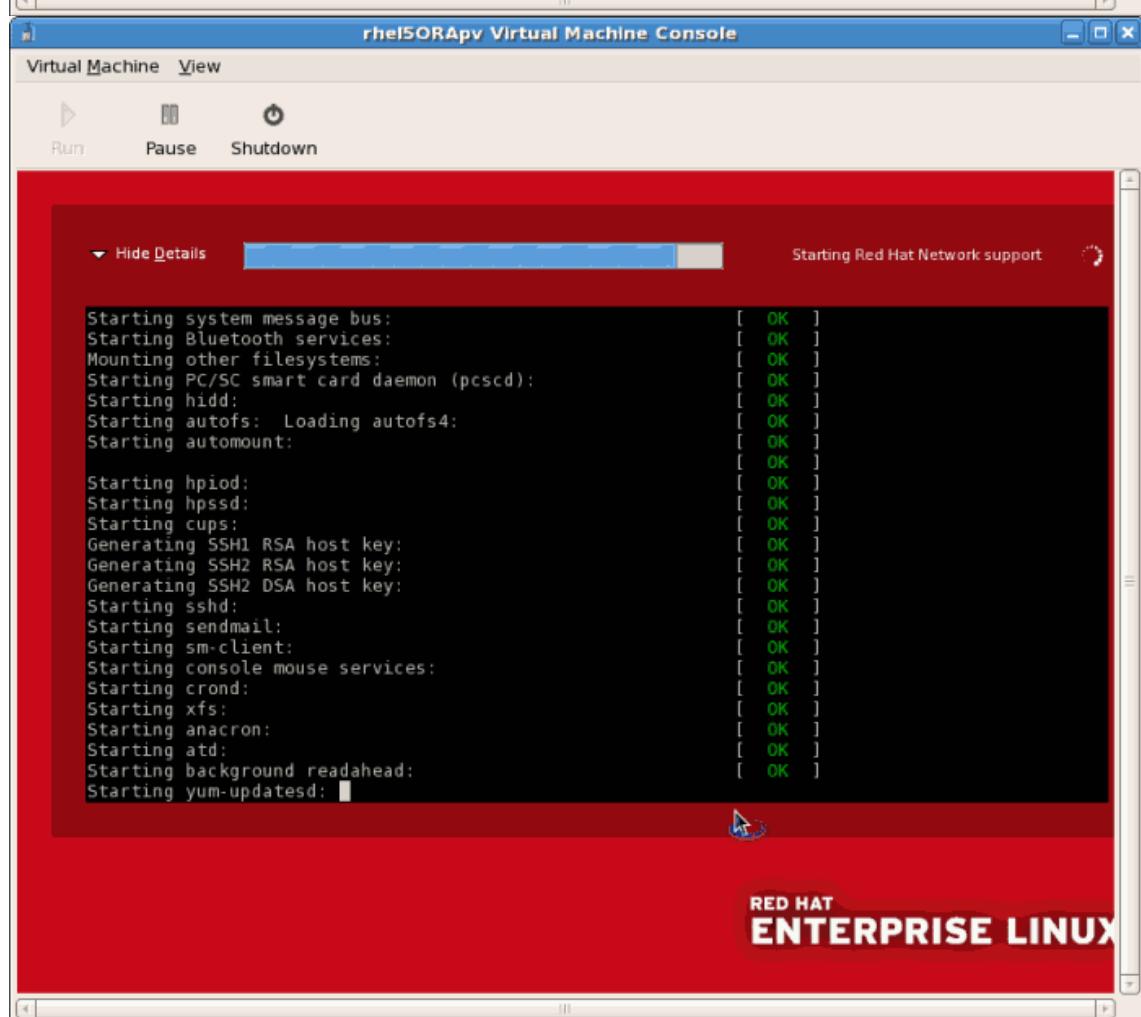
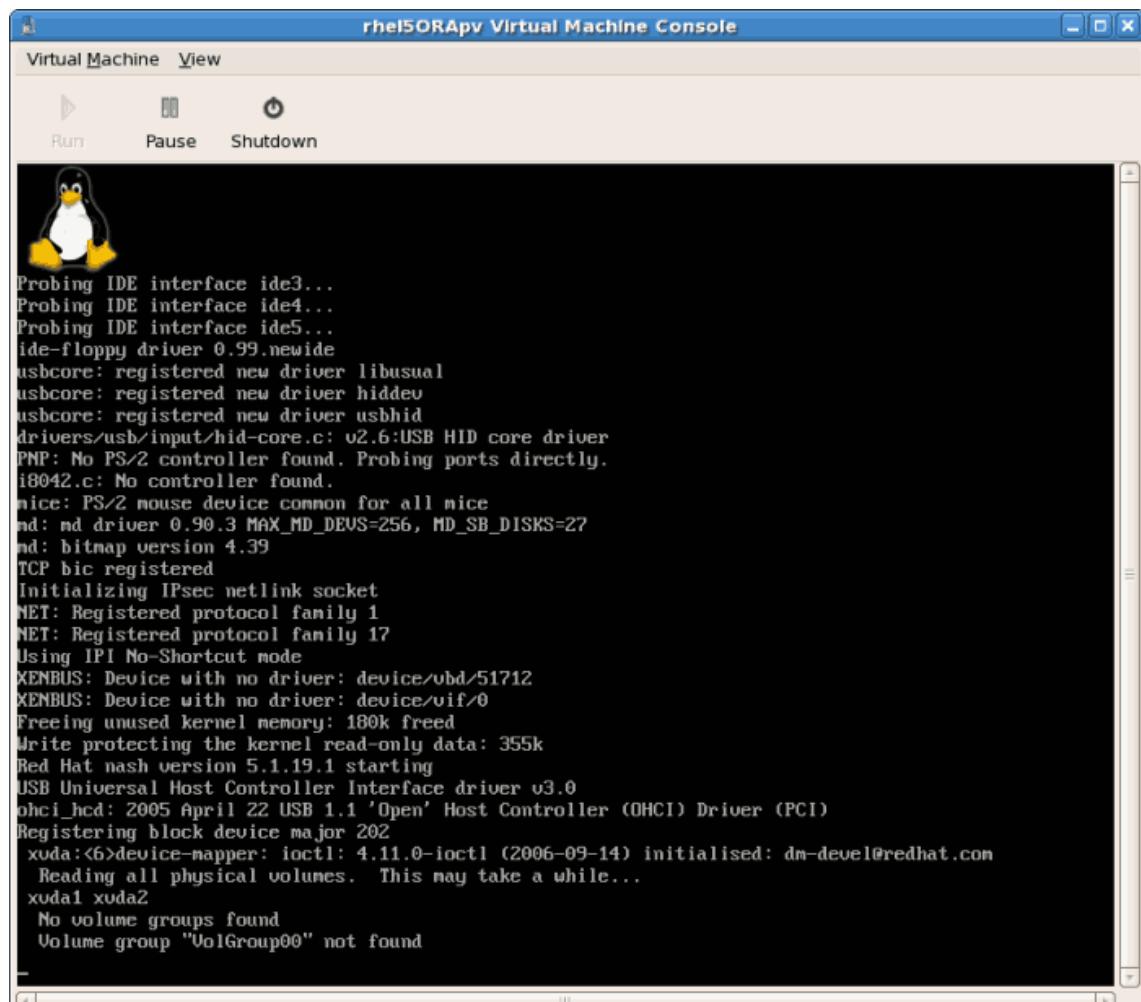
14. Boot the guest. The guest's name was chosen when you used the **virt-install** in [Section 8.1, “Installing Red Hat Enterprise Linux 5 as a para-virtualized guest”](#). If you used the default example the name is *rhel5PV*.

Use **virsh** to reboot the guest:

```
# virsh reboot rhel5PV
```

Alternatively, open **virt-manager**, select the name of your guest, click **Open**, then click **Run**.

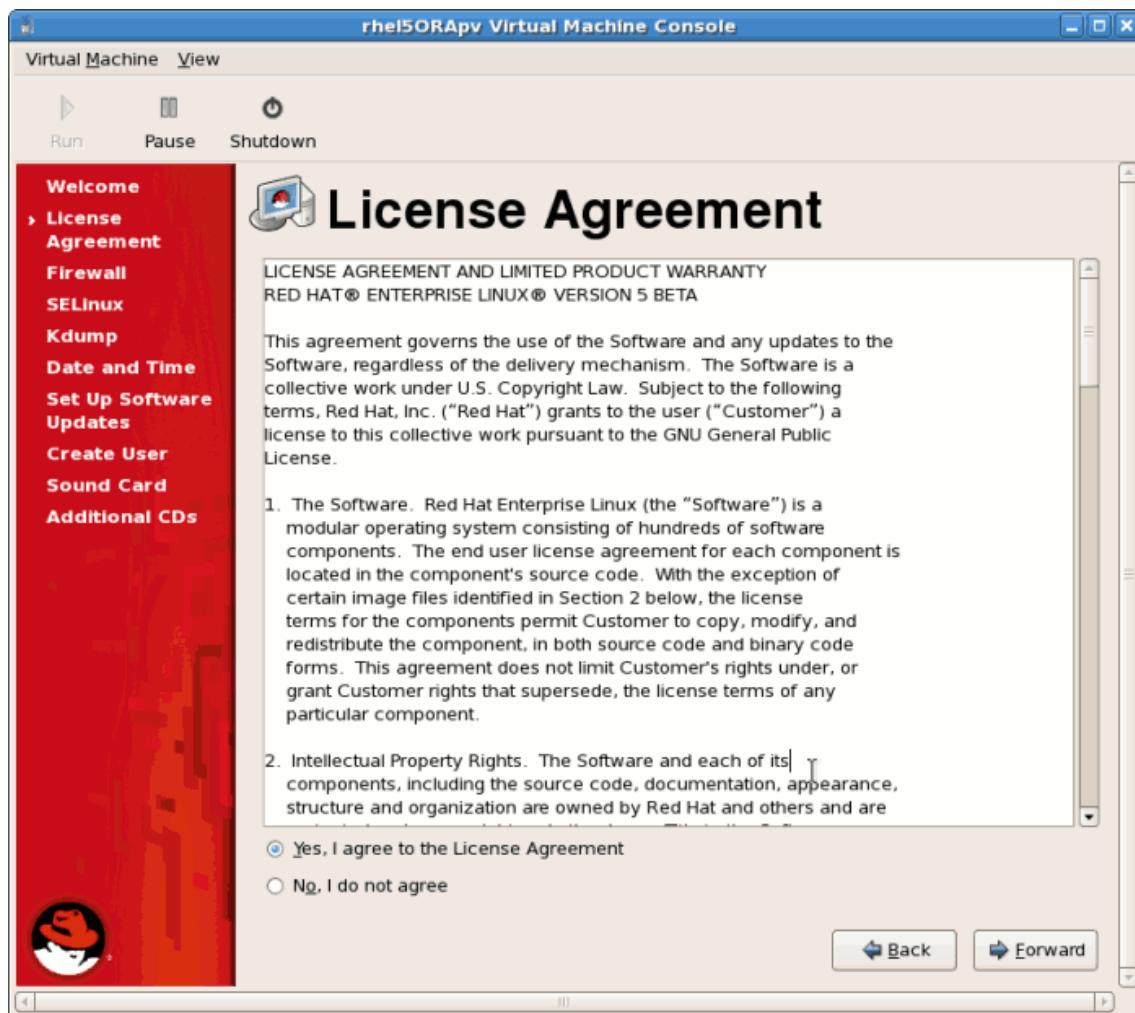
A **VNC** window displaying the guest's boot processes now opens.



15. Booting the guest starts the *First Boot* configuration screen. This wizard prompts you for some basic configuration choices for your guest.



16. Read and agree to the license agreement.



Click **Forward** on the license agreement windows.

17. Configure the firewall.

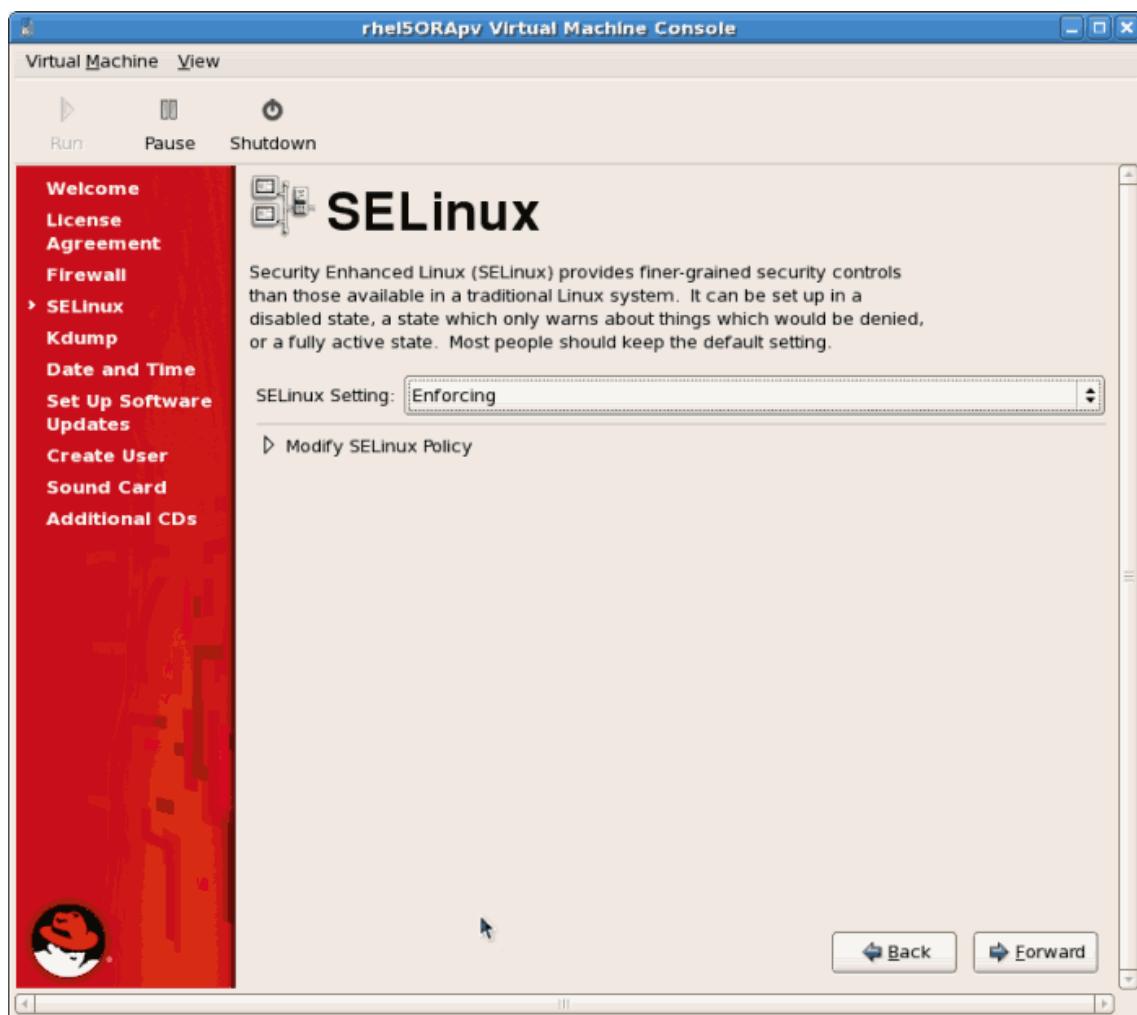


Click **Forward** to continue.

- a. If you disable the firewall you will be prompted to confirm your choice. Click **Yes** to confirm and continue. It is not recommended to disable your firewall.

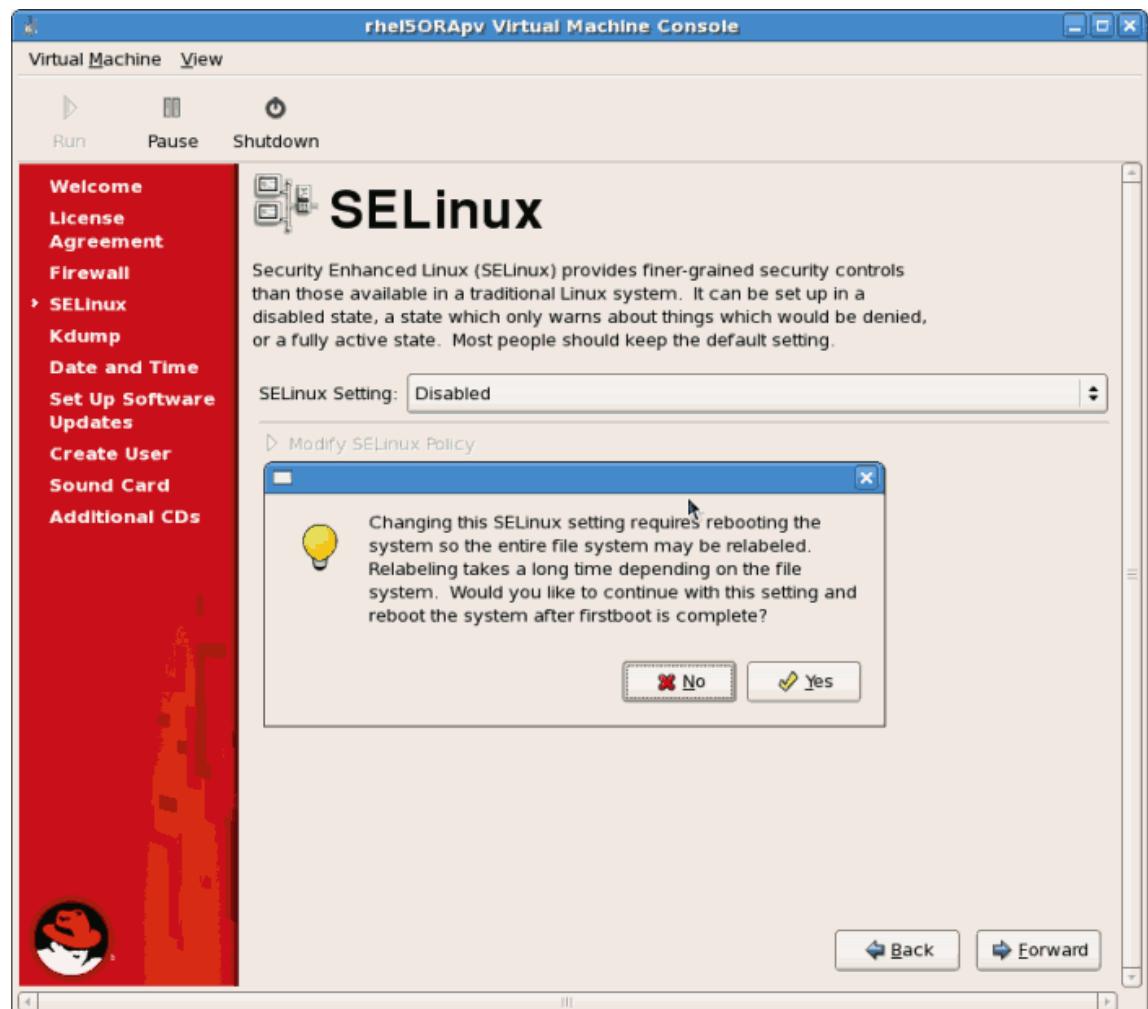


18. Configure SELinux. It is strongly recommended you run SELinux in **enforcing mode**. You can choose to either run SELinux in permissive mode or completely disable it.



Click **Forward** to continue.

- a. If you choose to disable SELinux this warning displays. Click **Yes** to disable SELinux.



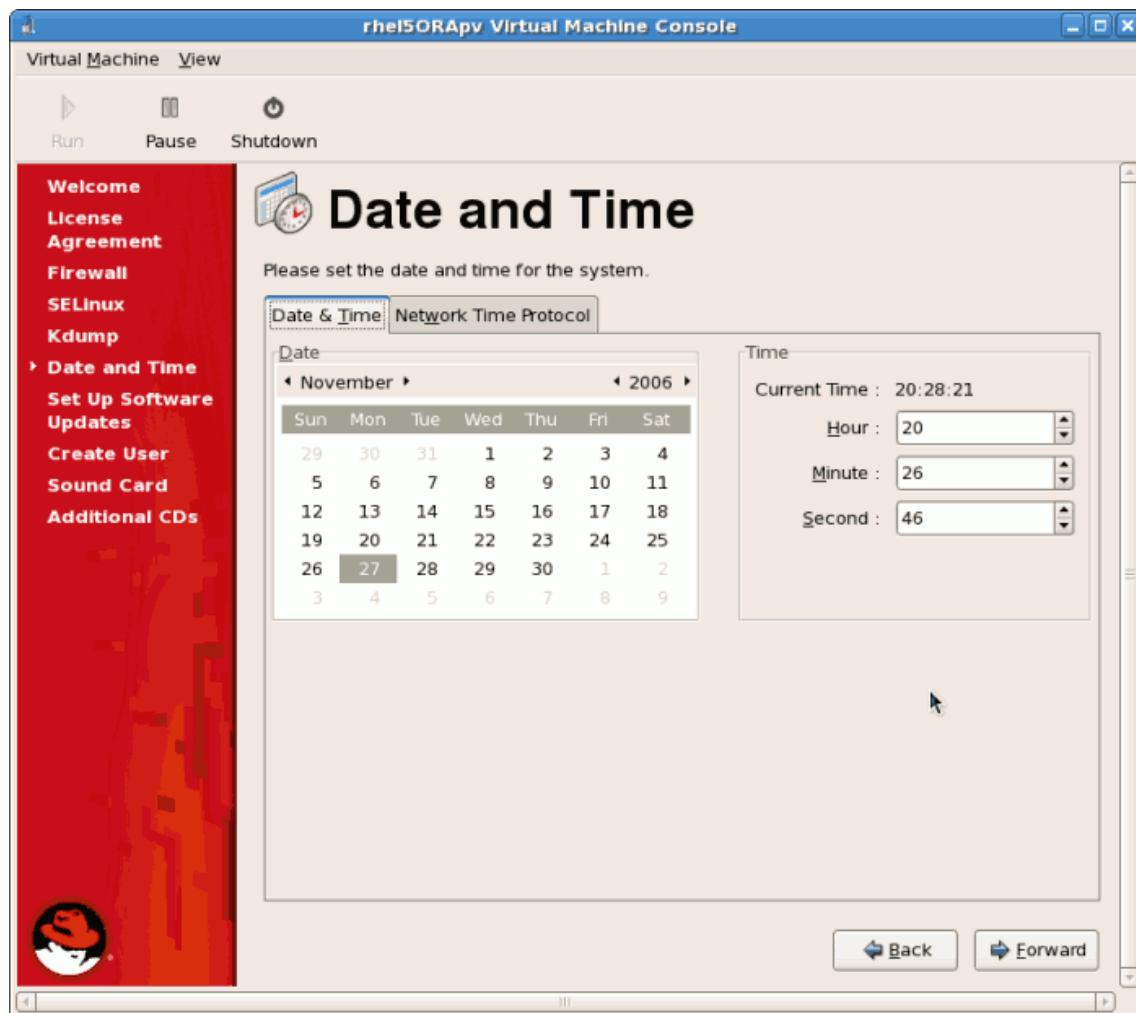
19. Disable **kdump**. The use of **kdump** is unsupported on para-virtualized guests.



Click **Forward** to continue.

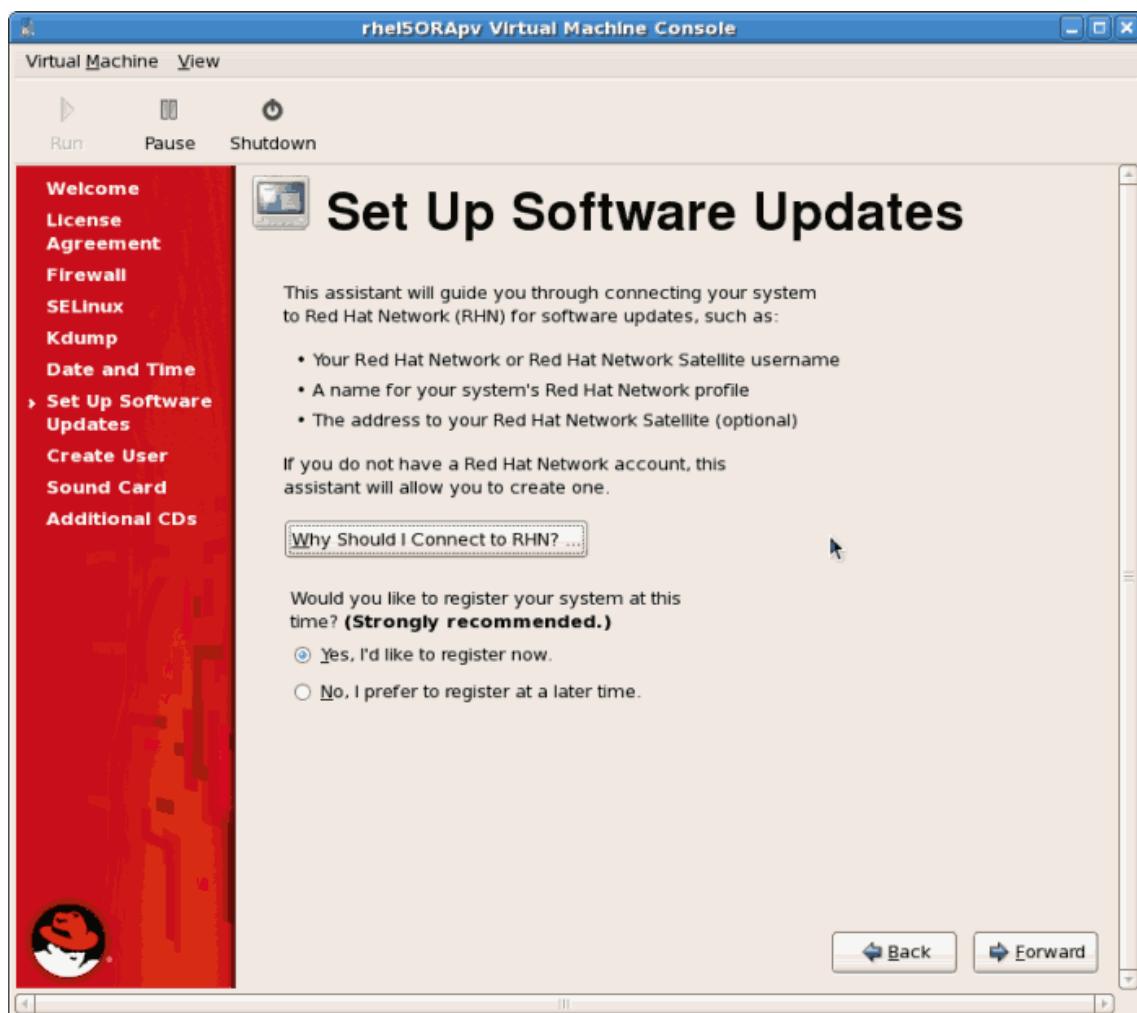
20. Confirm time and date are set correctly for your guest. If you install a para-virtualized guest time and date should synchronize with the hypervisor.

If the user sets the time or date during the installation it is ignored and the hypervisor's time is used.



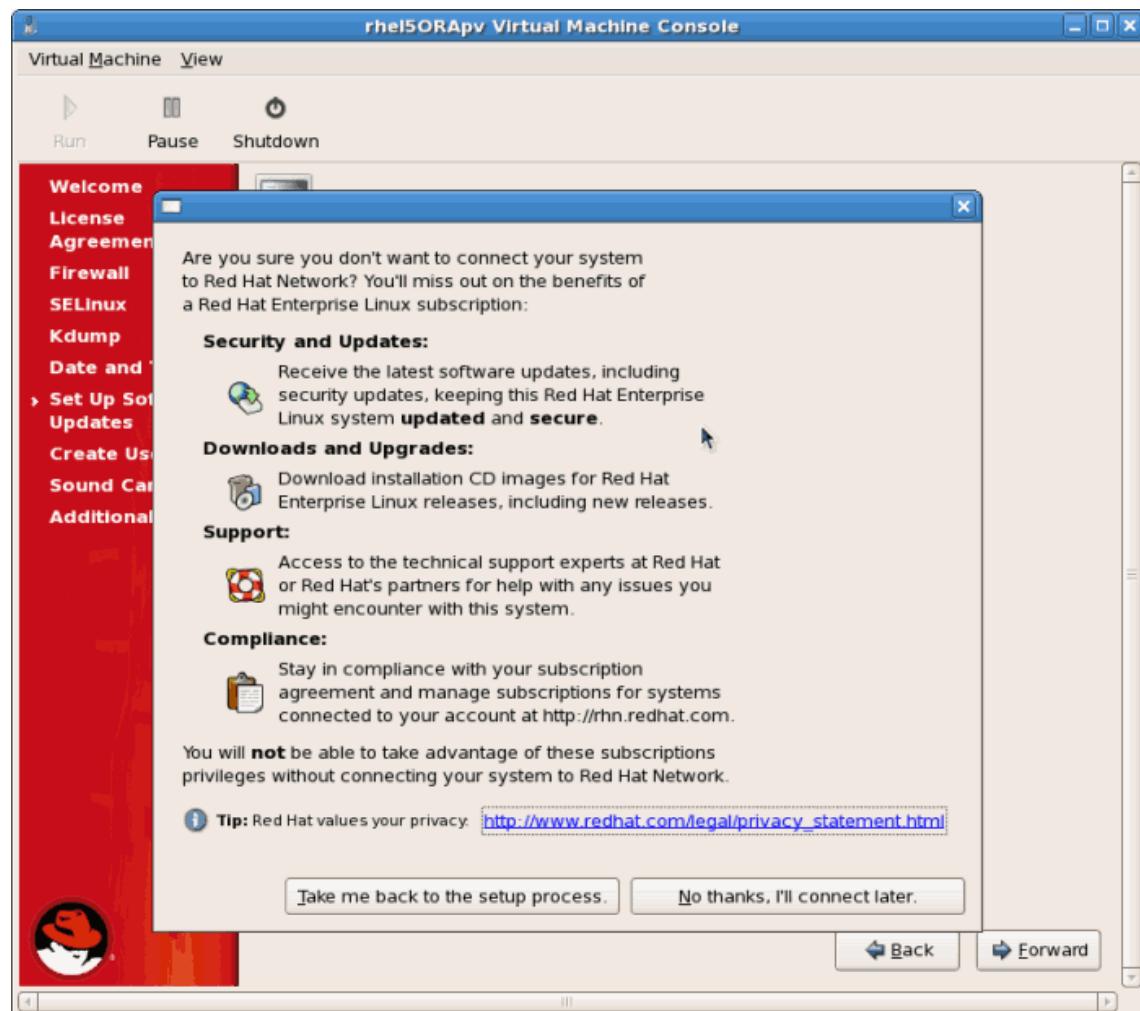
Click **Forward** to continue.

21. Set up software updates. If you have a Red Hat account or want to trial one use the screen below to register your newly installed guest.

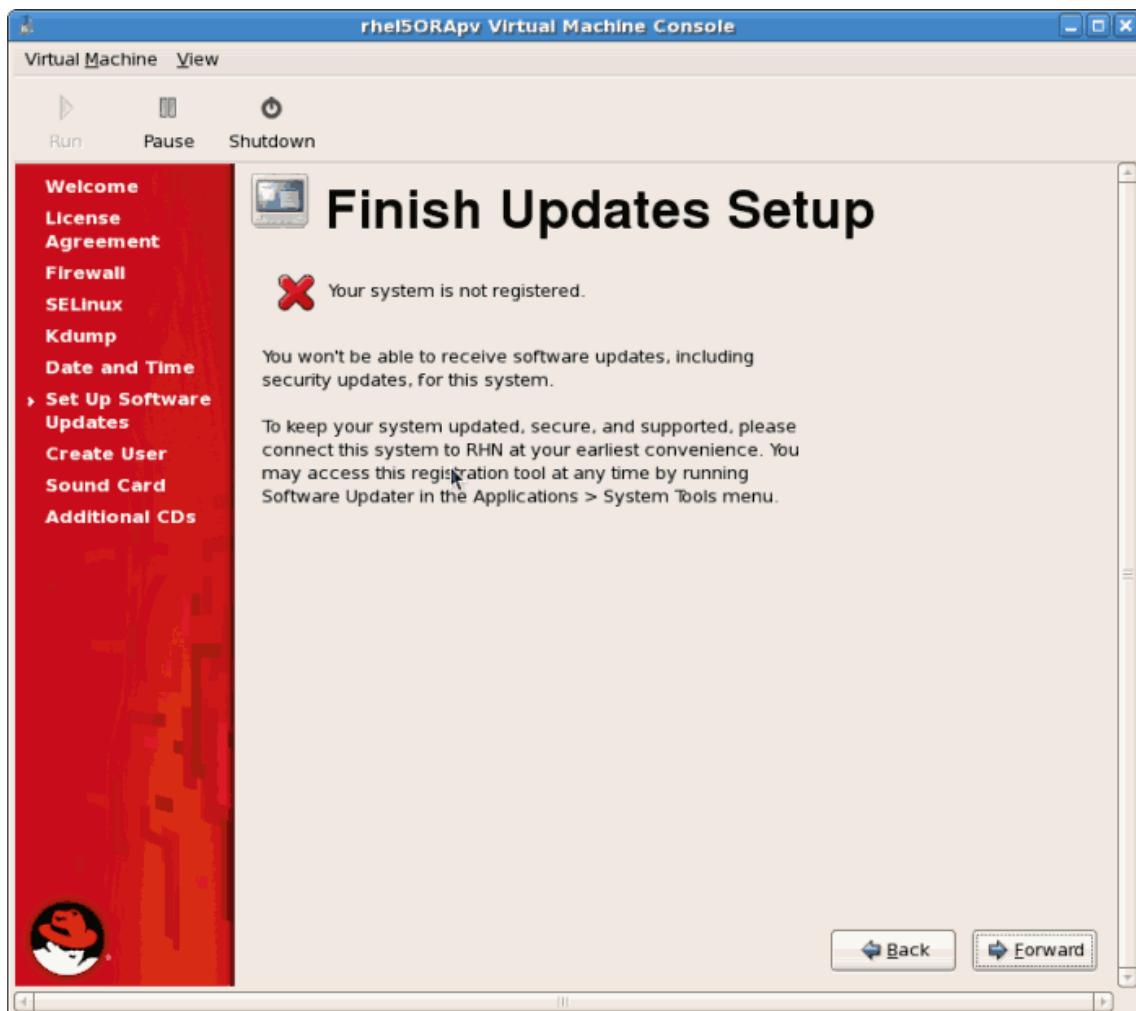


Click **Forward** to continue.

- a. Confirm your choices for RHN.

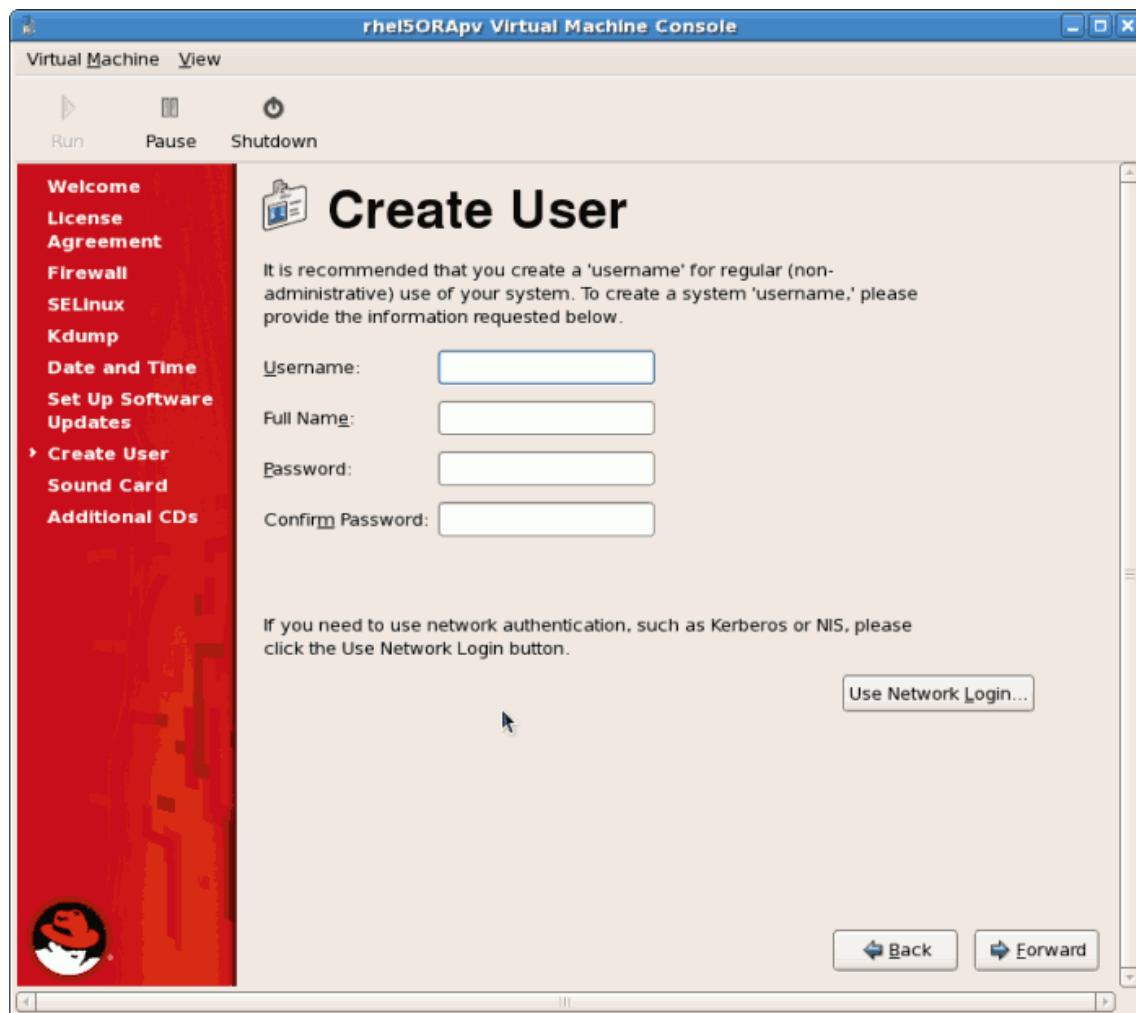


- b. You may see an additional screen if you did not configure RHN access. If RHN access is not enabled, you will not receive software updates.



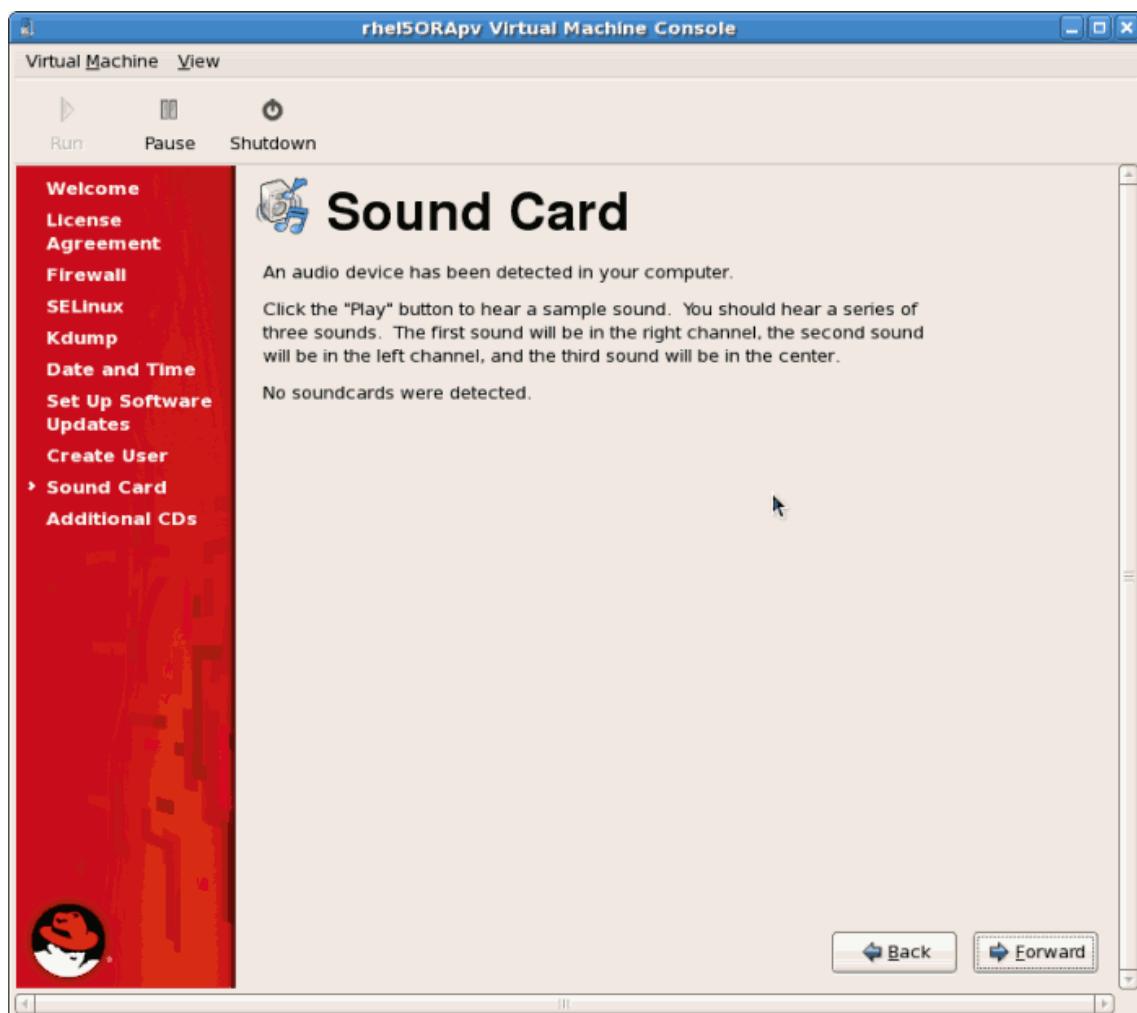
Click the **Forward** button.

22. Create a non root user account. It is advised to create a non root user for normal usage and enhanced security. Enter the Username, Name and password.

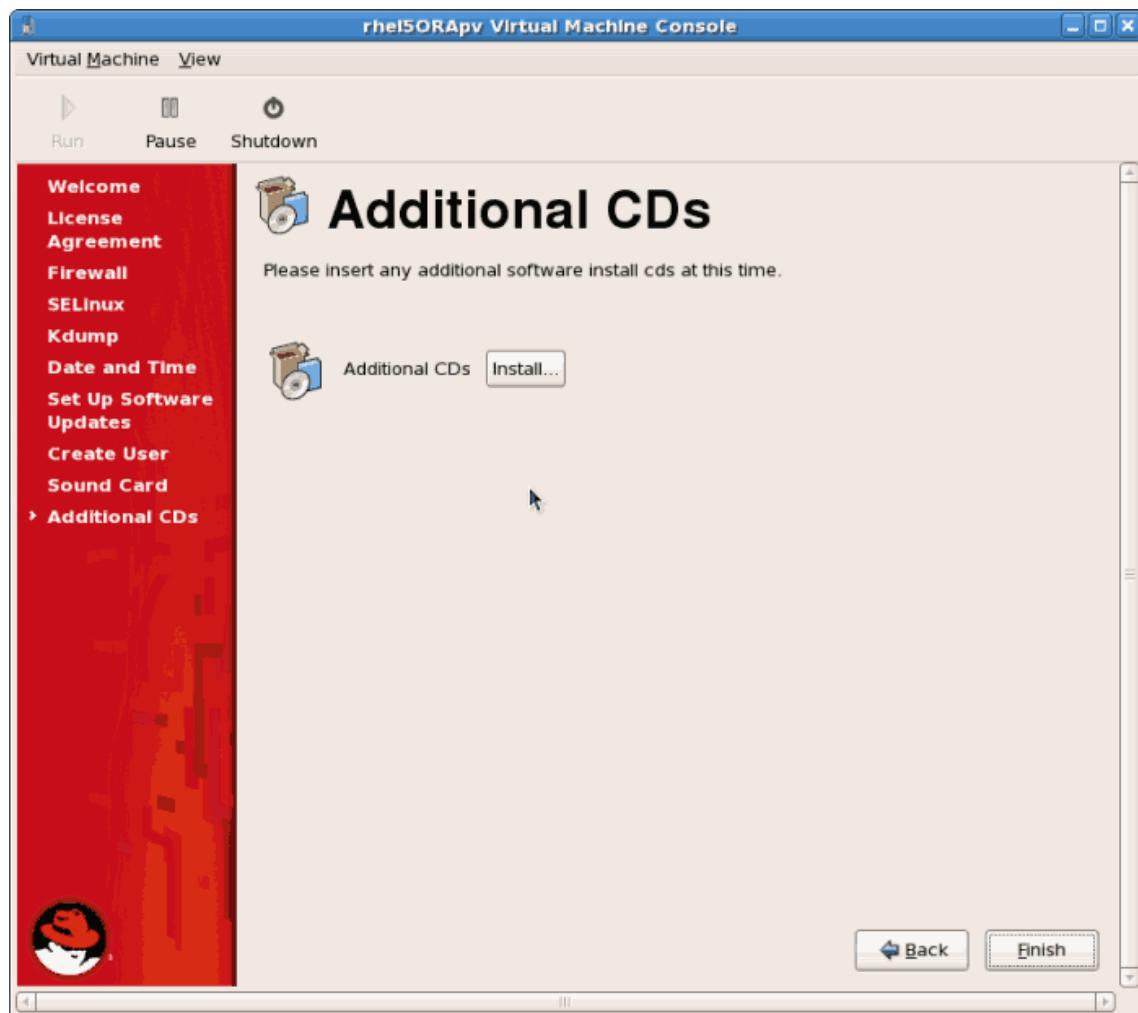


Click the **Forward** button.

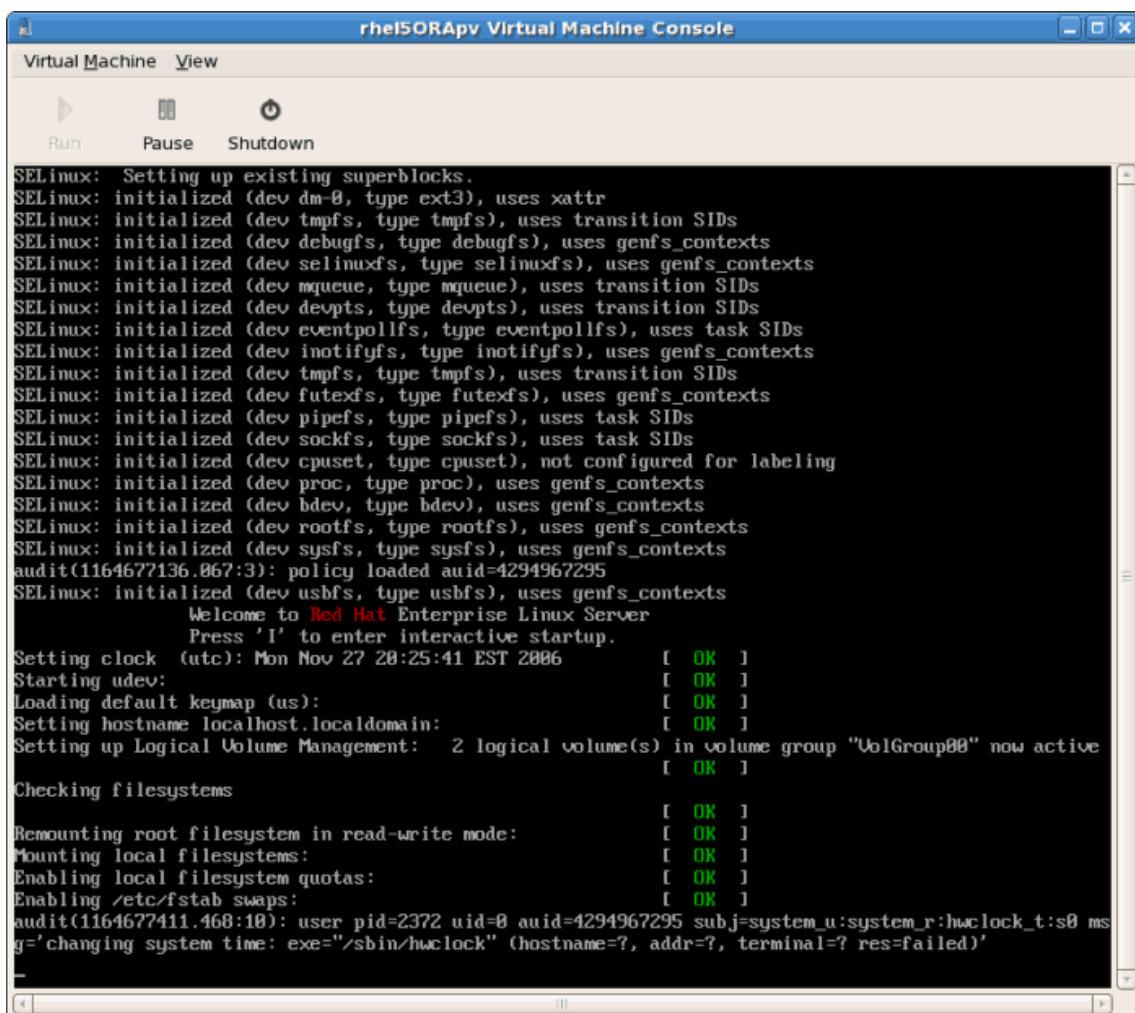
23. If a sound device is detected and you require sound, calibrate it. Complete the process and click **Forward**.



24. You can install additional packages from a CD or another repository using this screen. It is often more efficient to not install any additional software at this point but add packages later using the **yum** command or RHN. Click **Finish**.



25. The guest now configures any settings you changed and continues the boot process.



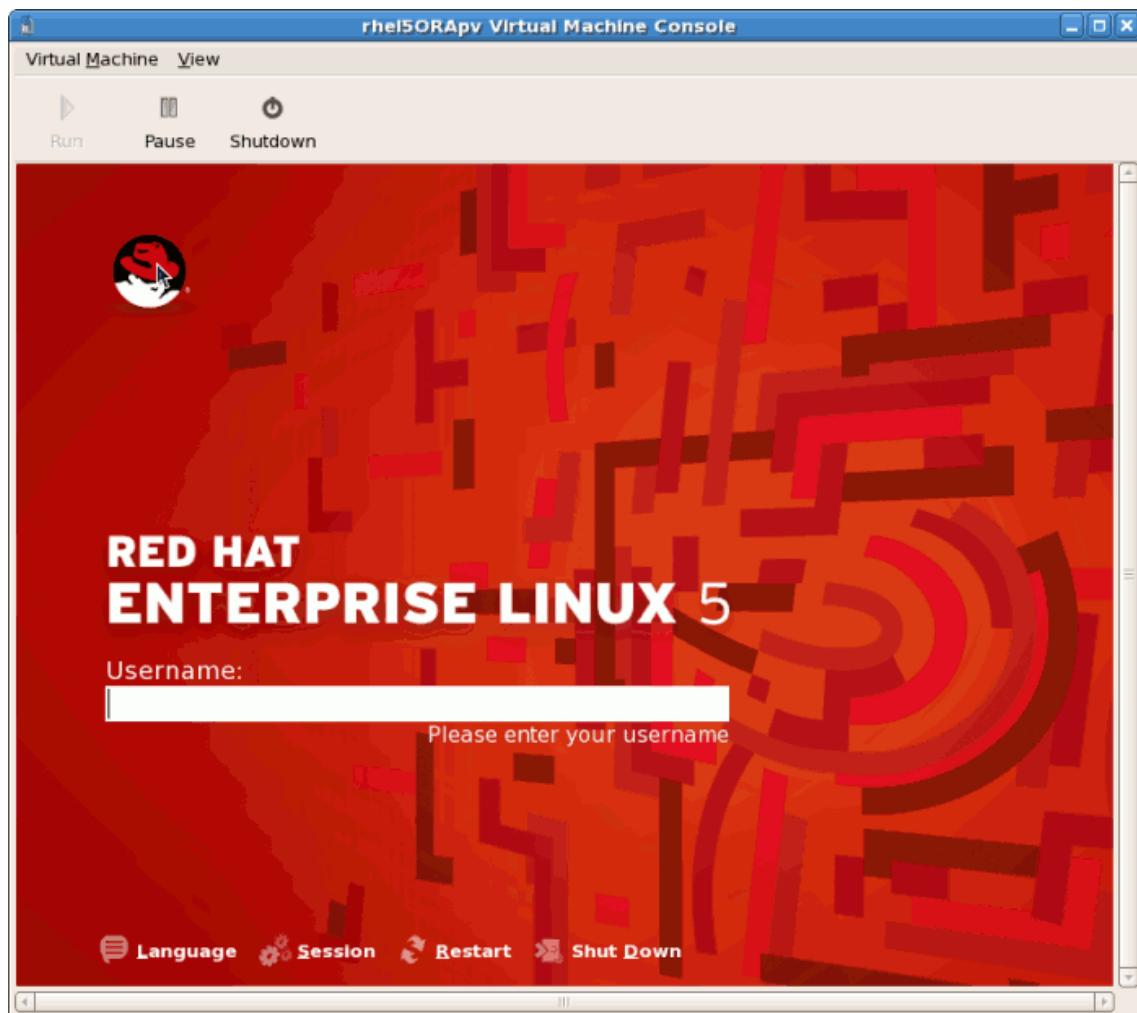
The screenshot shows a terminal window titled "rhel5ORApv Virtual Machine Console". The window has three buttons at the top: "Virtual Machine", "View", and a close button. Below the title bar are three icons: "Run", "Pause", and "Shutdown". The main area of the window displays the boot log of a Red Hat Enterprise Linux 5 system. The log includes messages from SELinux, the kernel, and various initialization scripts. Key messages include:

- SELinux: Setting up existing superblocks.
- SELinux: initialized (dev dm-0, type ext3), uses xattr
- SELinux: initialized (dev tmpfs, type tmpfs), uses transition SIDs
- SELinux: initialized (dev debugfs, type debugfs), uses genfs_contexts
- SELinux: initialized (dev selinuxfs, type selinuxfs), uses genfs_contexts
- SELinux: initialized (dev mqueue, type mqueue), uses transition SIDs
- SELinux: initialized (dev devpts, type devpts), uses transition SIDs
- SELinux: initialized (dev eventpollfs, type eventpollfs), uses task SIDs
- SELinux: initialized (dev inotifyfs, type inotifyfs), uses genfs_contexts
- SELinux: initialized (dev tmpfs, type tmpfs), uses transition SIDs
- SELinux: initialized (dev futexfs, type futexfs), uses genfs_contexts
- SELinux: initialized (dev pipefs, type pipefs), uses task SIDs
- SELinux: initialized (dev sockfs, type sockfs), uses task SIDs
- SELinux: initialized (dev cpuset, type cpuset), not configured for labeling
- SELinux: initialized (dev proc, type proc), uses genfs_contexts
- SELinux: initialized (dev bdev, type bdev), uses genfs_contexts
- SELinux: initialized (dev rootfs, type rootfs), uses genfs_contexts
- SELinux: initialized (dev sysfs, type sysfs), uses genfs_contexts
- audit(1164677136.067:3): policy loaded auid=4294967295
- SELinux: initialized (dev usbfs, type usbfs), uses genfs_contexts

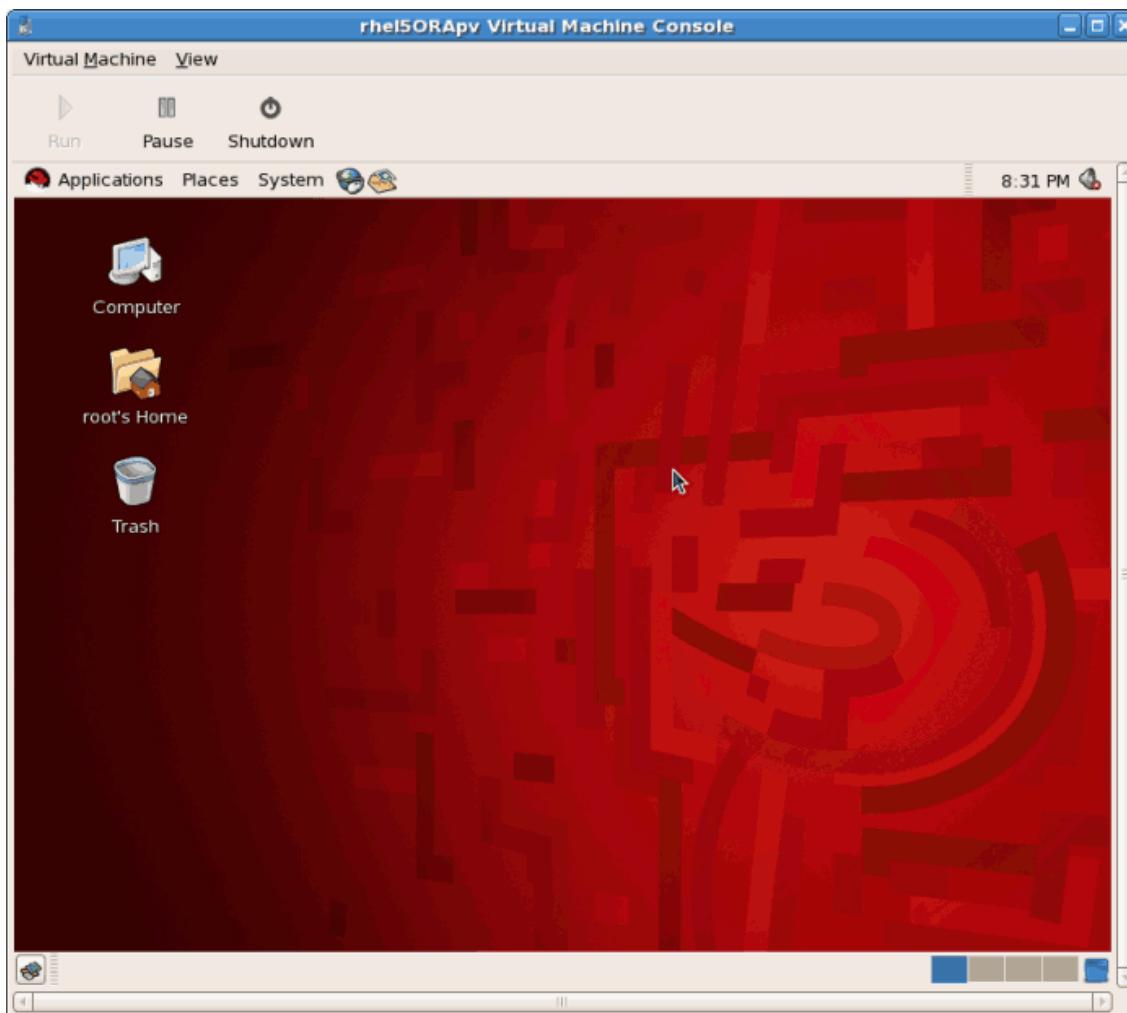
Following the SELinux messages, the log shows the system booting into Red Hat Enterprise Linux Server. It prompts the user to press 'I' for interactive startup. The log continues with standard system initialization steps:

- Welcome to Red Hat Enterprise Linux Server
- Press 'I' to enter interactive startup.
- Setting clock (utc): Mon Nov 27 20:25:41 EST 2006 [OK]
- Starting udev: [OK]
- Loading default keymap (us): [OK]
- Setting hostname localhost.localdomain: [OK]
- Setting up Logical Volume Management: 2 logical volume(s) in volume group "VolGroup00" now active [OK]
- Checking filesystems [OK]
- Remounting root filesystem in read-write mode: [OK]
- Mounting local filesystems: [OK]
- Enabling local filesystem quotas: [OK]
- Enabling /etc/fstab swaps: [OK]
- audit(1164677411.468:10): user pid=2372 uid=0 auid=4294967295 subj=system_u:system_r:hwclock_t:s0 msg='changing system time: exe="/sbin/hwclock" (hostname=?, addr=?, terminal=? res=failed)'

26. The Red Hat Enterprise Linux 5 login screen displays. Log in using the username created in the previous steps.



27. You have now successfully installed a para-virtualized Red Hat Enterprise Linux guest.



8.2. Installing Red Hat Enterprise Linux as a fully virtualized guest

This section covers installing a fully virtualized Red Hat Enterprise Linux 5 guest. This procedure covers both the KVM and the Xen hypervisors; the steps are interchangeable and different steps are noted.

The KVM hypervisor requires Red Hat Enterprise Linux 5.4 or newer.

Procedure 8.3. 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. If installed, select Xen or KVM. For this example, select KVM. Note that presently KVM is named **qemu**.

Connect to a hypervisor if you have not already done so. Open the **File** menu and select the **Add Connection...** option. See [Section 27.1, “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 guest. Punctuation and whitespace characters are not permitted in versions before Red Hat Enterprise Linux 5.5. Red Hat Enterprise Linux 5.5 adds support for '_', '.' and '-' characters.

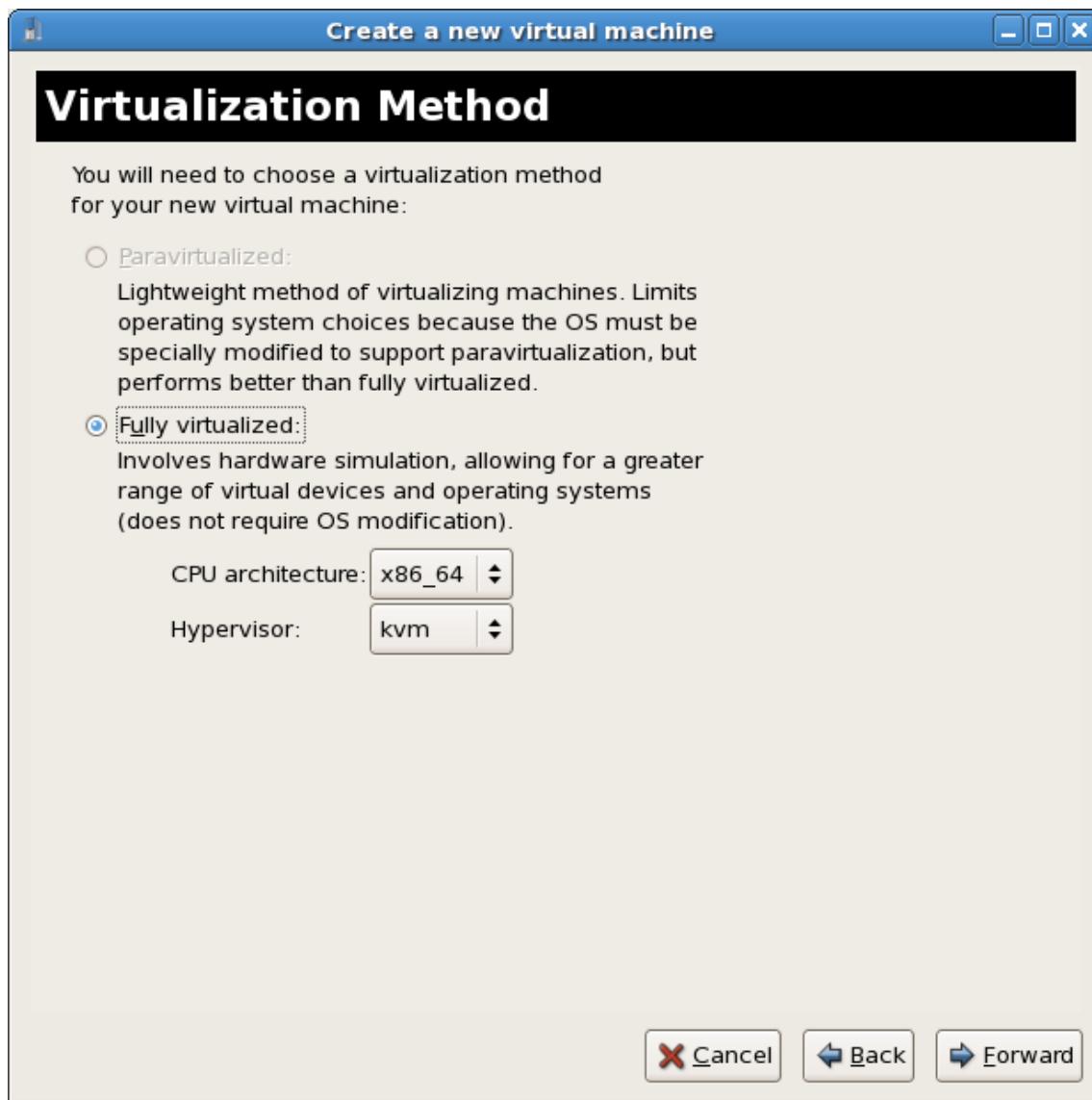


Press **Forward** to continue.

5.

Choose a virtualization method

Choose the virtualization method for the guest. Note you can only select an installed virtualization method. If you selected KVM or Xen earlier ([Step 4](#)) you must use the hypervisor you selected. This example uses the KVM hypervisor.



Press **Forward** to continue.

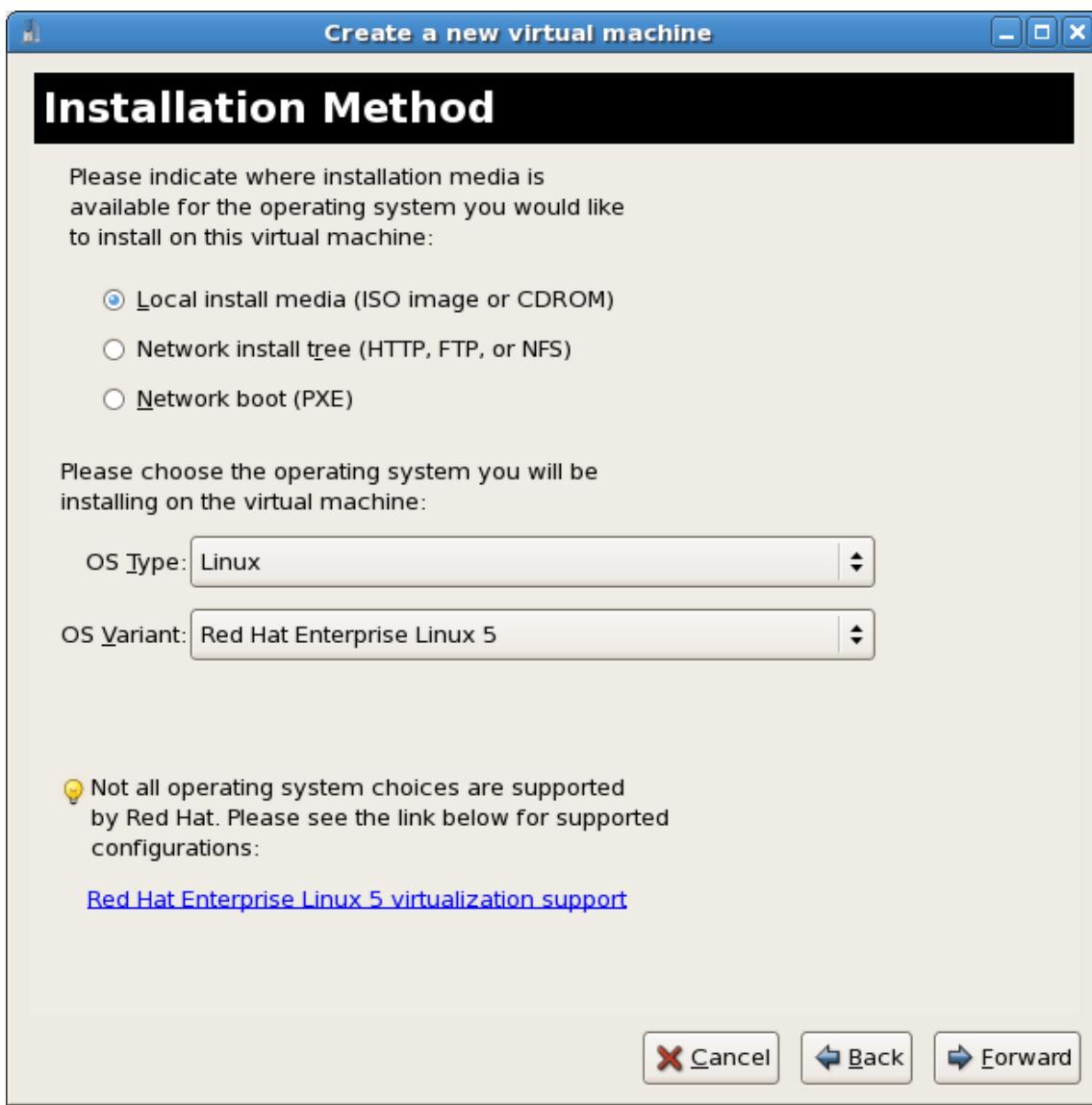
6.

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.



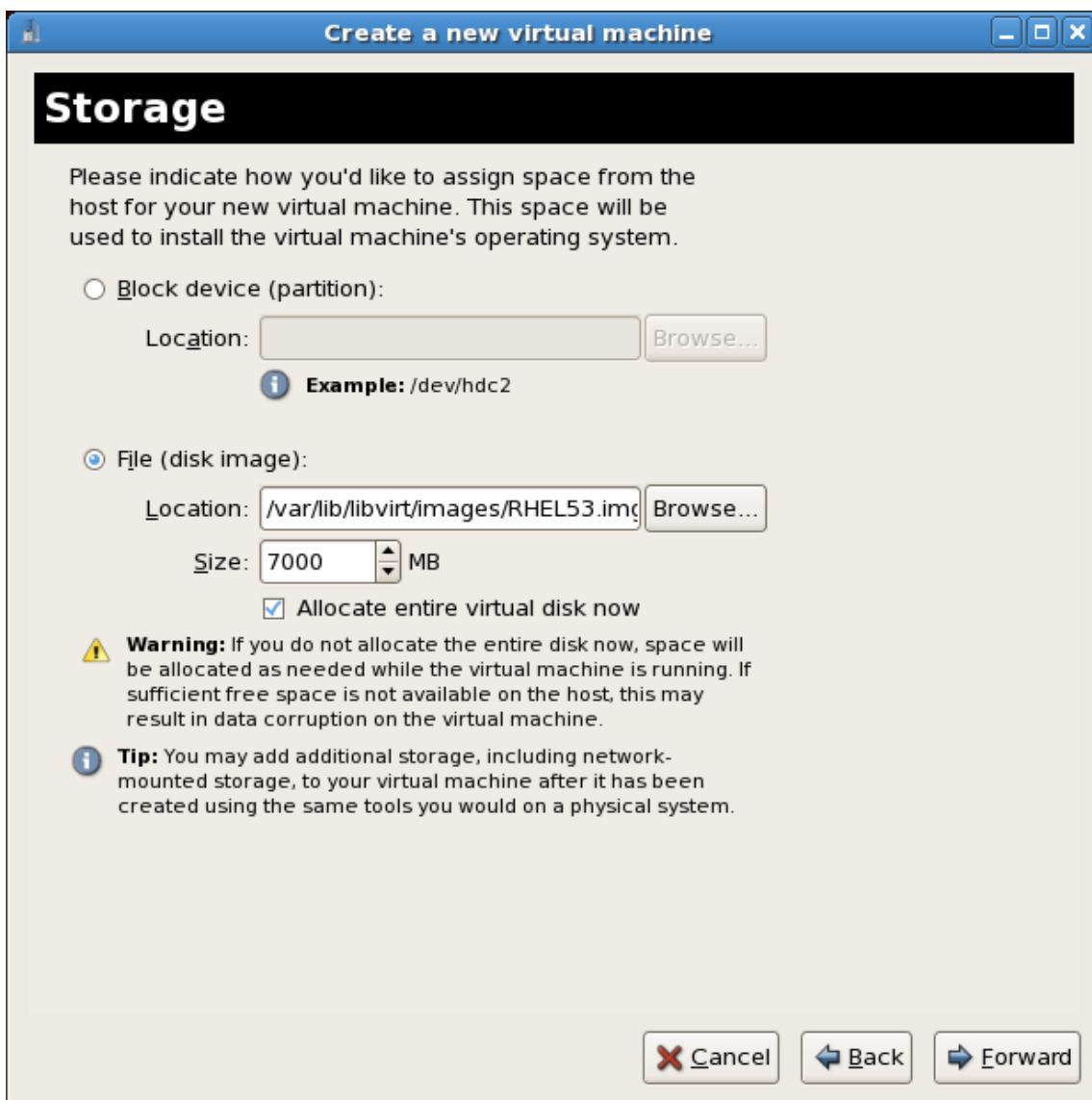
Warning

For ISO image files and guest storage images the recommended directory is **/var/lib/libvirt/images/**. Any other location may require additional configuration for SELinux, see [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 guest and any applications the guest requires.



Press **Forward** to continue.

Note

Live and offline migrations require guests to be installed on shared network storage. For information on setting up shared storage for guests see [Part V, “Virtualization Storage Topics”](#).

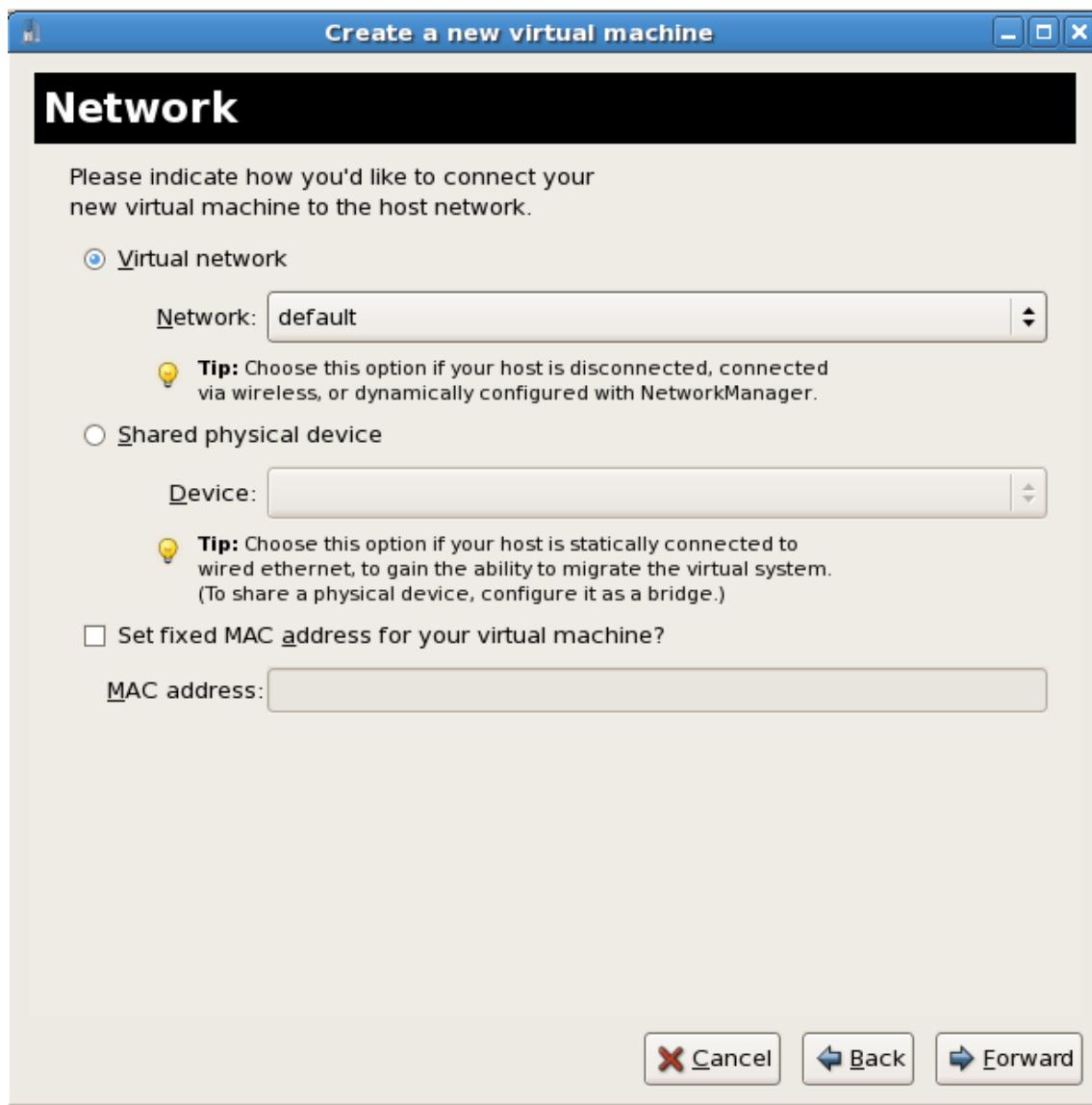
9.

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 guest. Use the virtual network option for wireless networks.

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



Press **Forward** to continue.

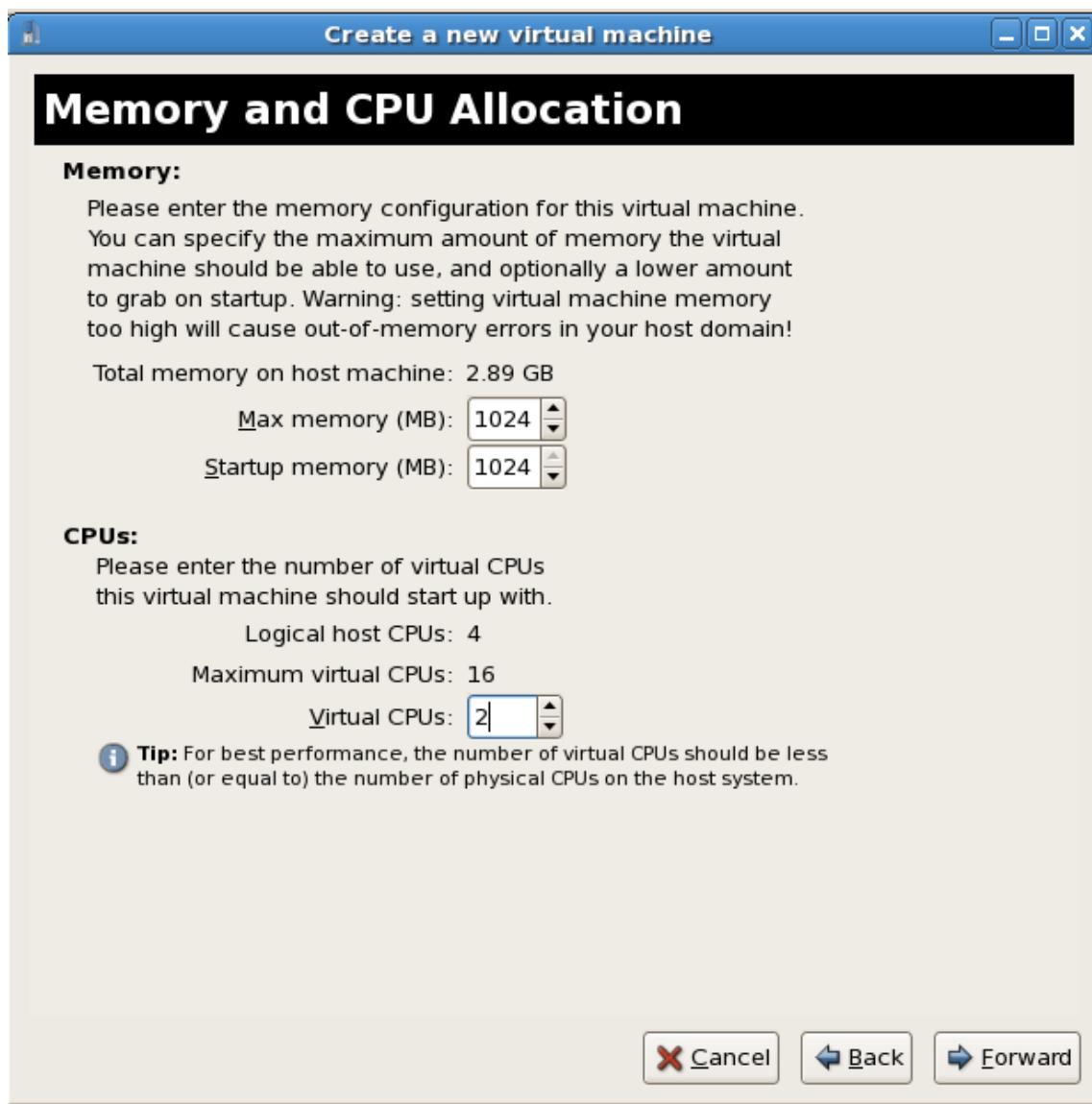
10.

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.

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 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 5 installation sequence. The installation sequence is covered by the *Installation Guide*, see [Red Hat Documentation](#) for the Red Hat Enterprise Linux *Installation Guide*.

A fully virtualized Red Hat Enterprise Linux 5 Guest is now installed.

8.3. 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 Red Hat Enterprise Linux.

This procedure covers both the KVM and the Xen hypervisors; the steps are interchangeable and different steps are noted.

The KVM hypervisor requires Red Hat Enterprise Linux 5.4 or newer.

Before commencing this procedure ensure you must have root access.



Important

Presently, Red Hat Enterprise Linux hosts on the Itanium® architecture does not support fully virtualized Windows XP guests. Only Windows Server 2003 for Itanium-based Systems is supported for Itanium systems.

1.

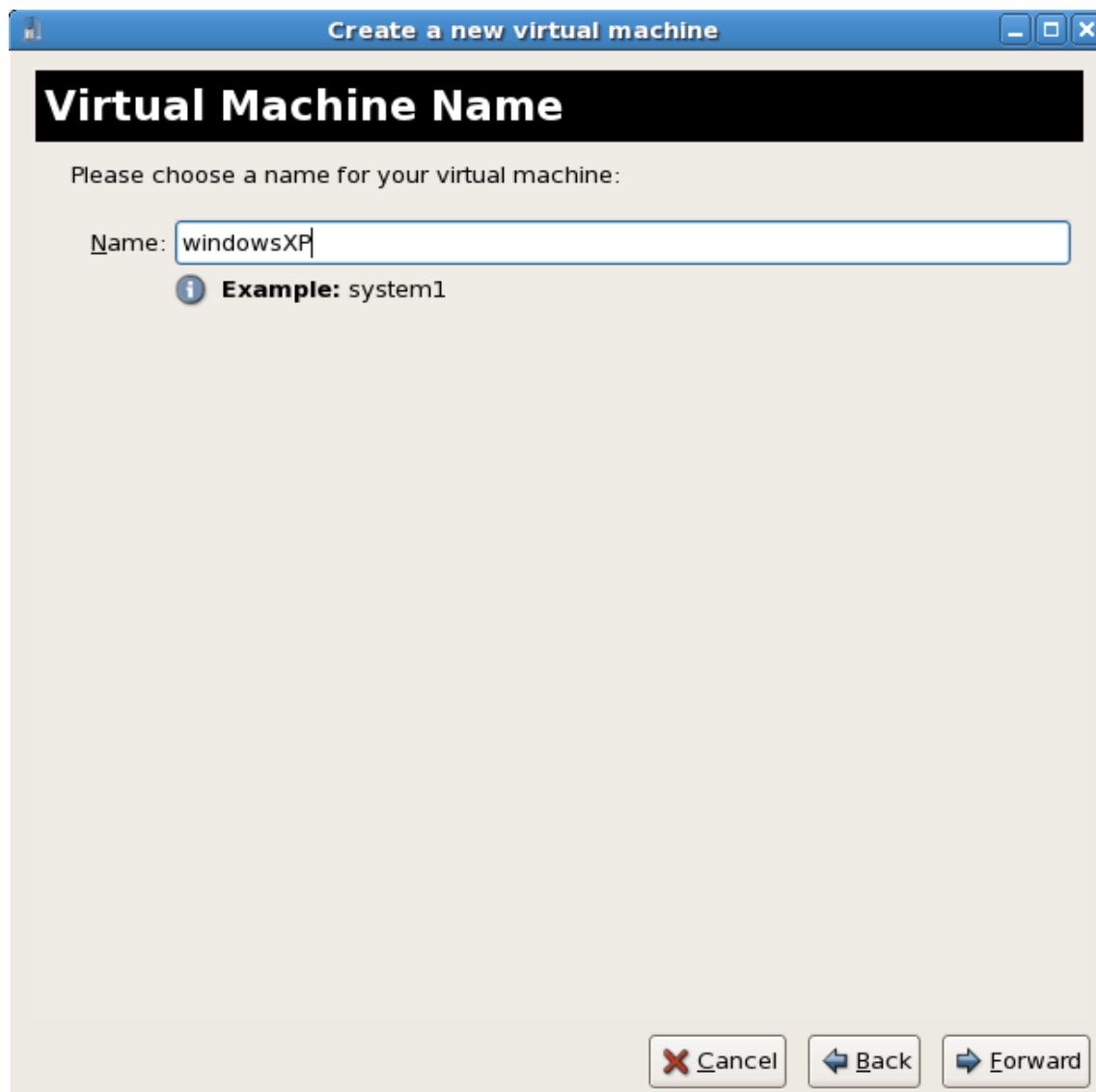
Starting virt-manager

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 virtual system

Enter the **System Name** and click the **Forward** button.



3.

Choosing a virtualization method

If you selected KVM or Xen earlier (step [Step 1](#)) you must use the hypervisor you selected. This example uses the KVM hypervisor.

Windows can only be installed using full virtualization.



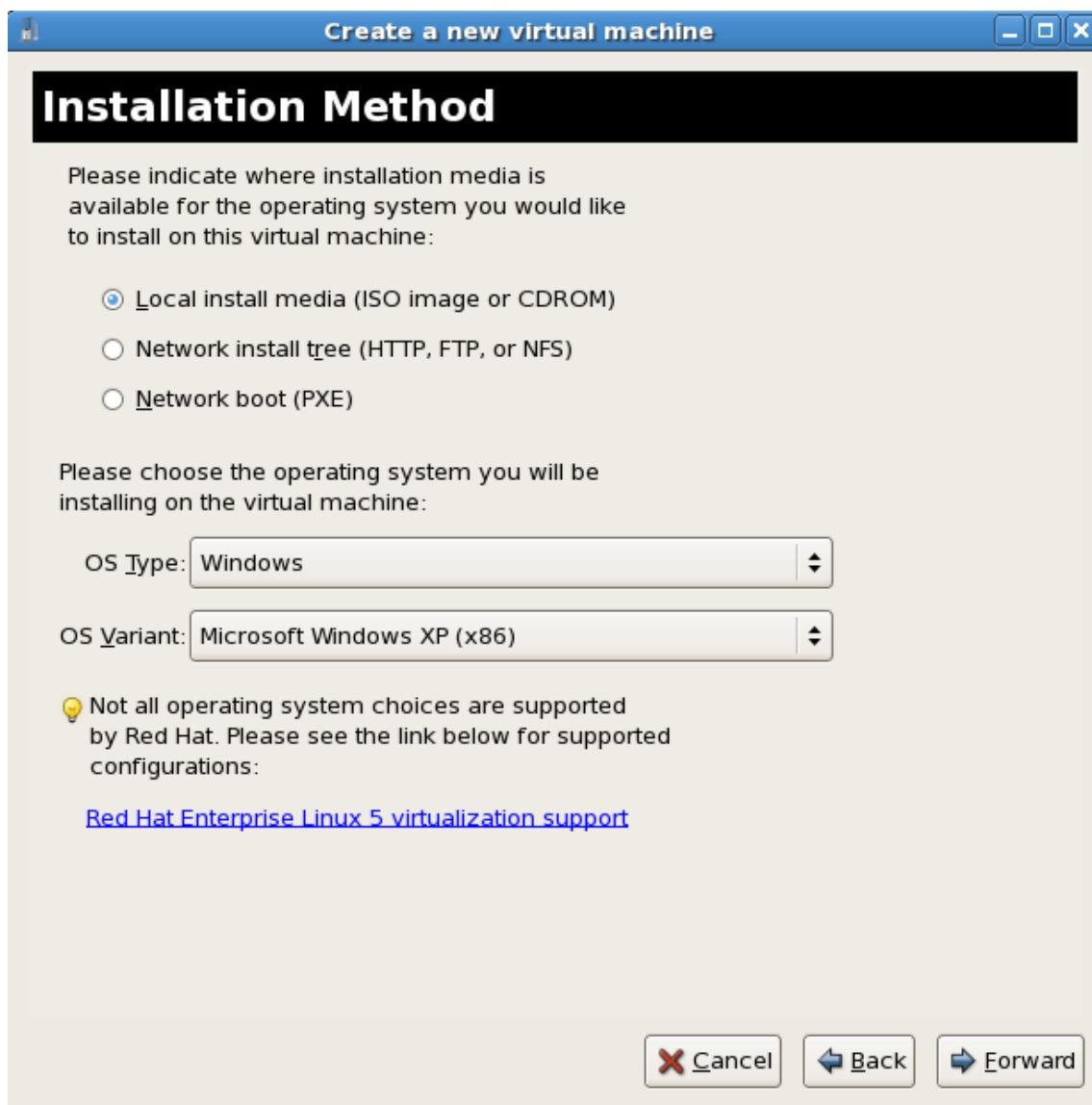
4.

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.

Installing guests with PXE is supported in Red Hat Enterprise Linux 5.2. PXE installation is not covered by this chapter.



Warning

For ISO image files and guest storage images it is recommended to use the **/var/lib/libvirt/images/** directory. Any other location will require additional configuration for SELinux, see [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 are stored in the `/var/lib/libvirt/images/` directory by default. In the default configuration, other directory locations for file-based images are prohibited by SELinux. If you use a different directory you must label the new directory according to SELinux policy. See [Section 19.2, “SELinux and virtualization”](#) for details.

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 recommended 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)

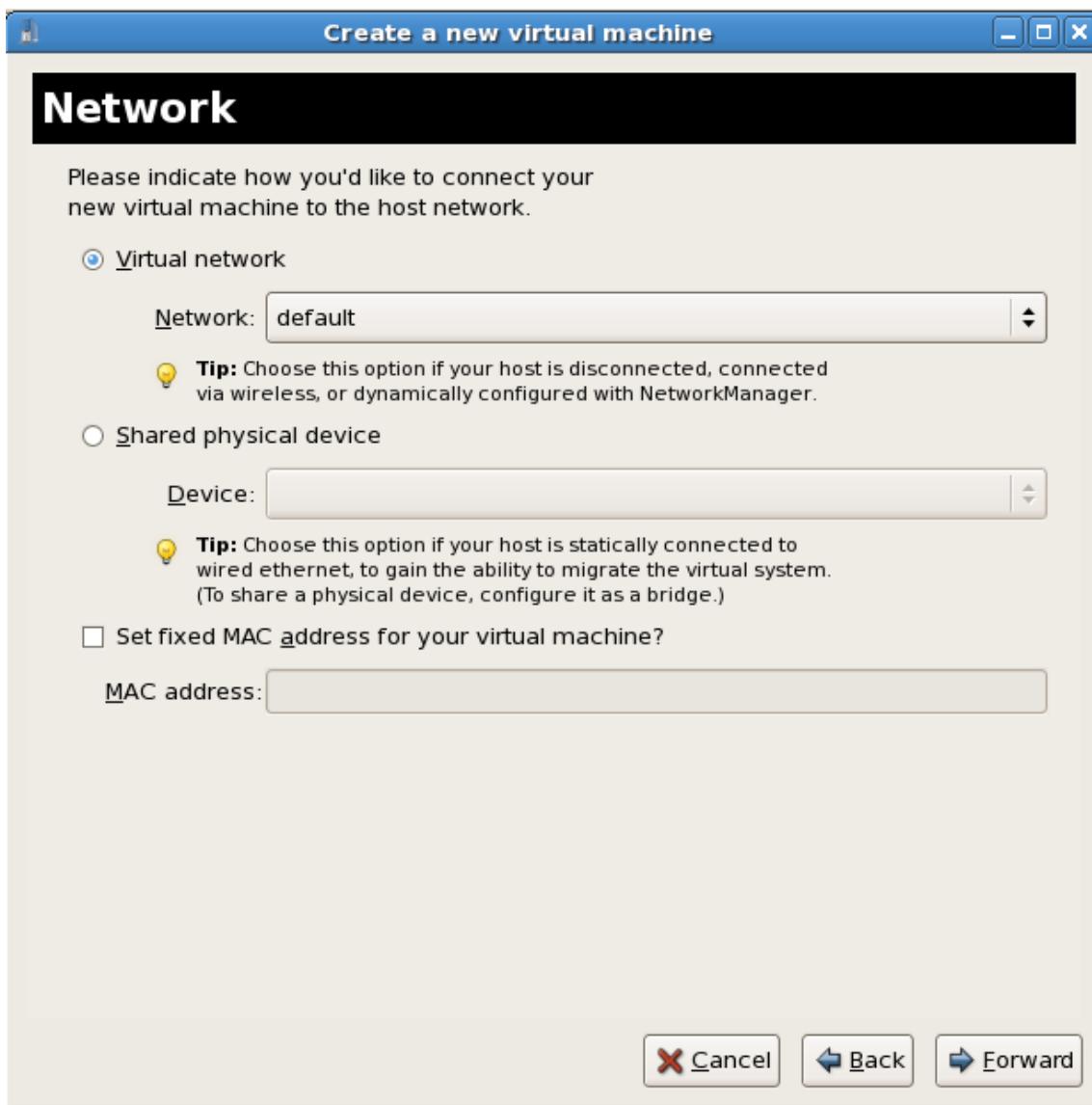
7.

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 guest. Use the virtual network option for wireless networks.

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

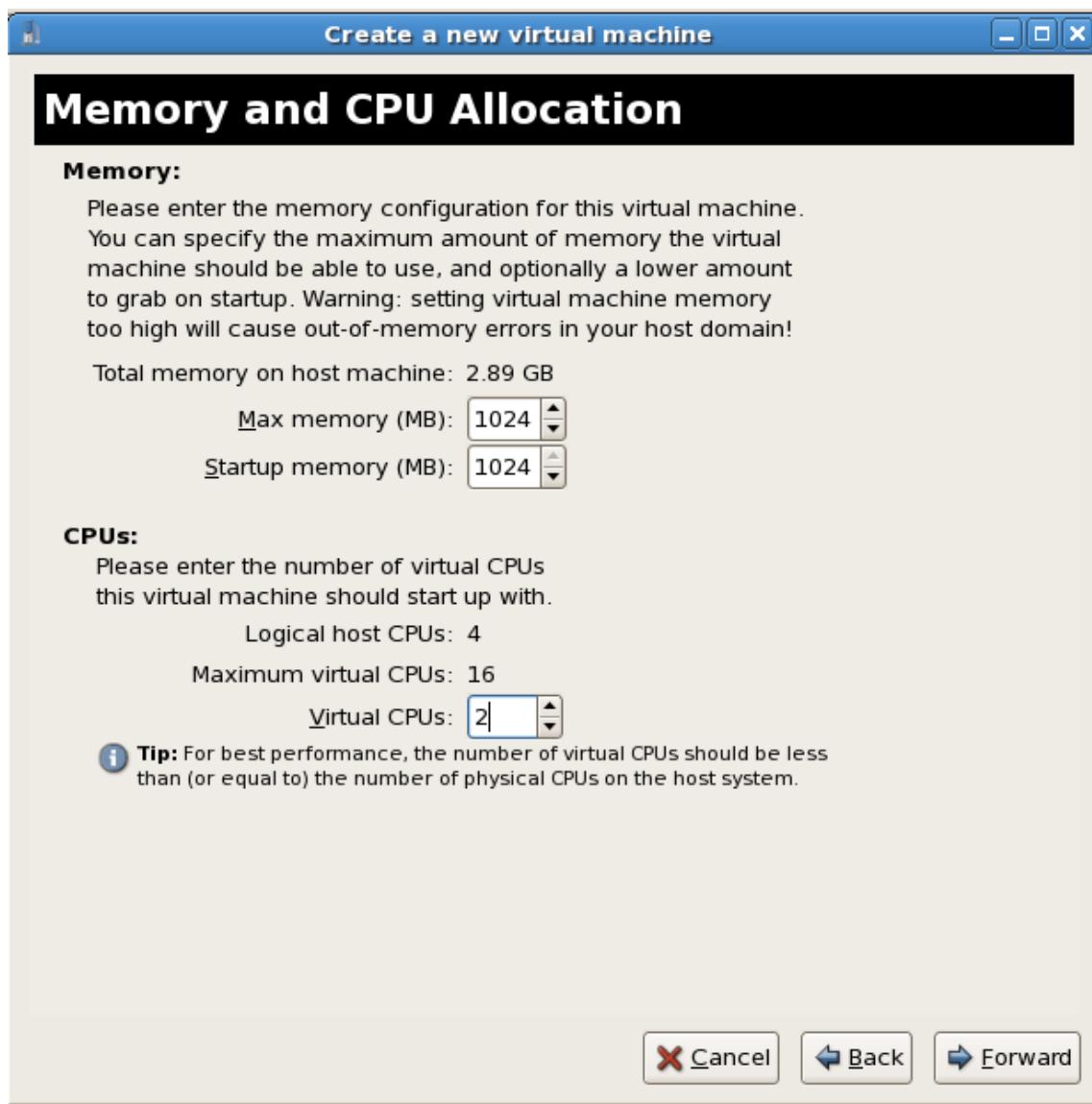


Press **Forward** to continue.

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.

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 systems 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.

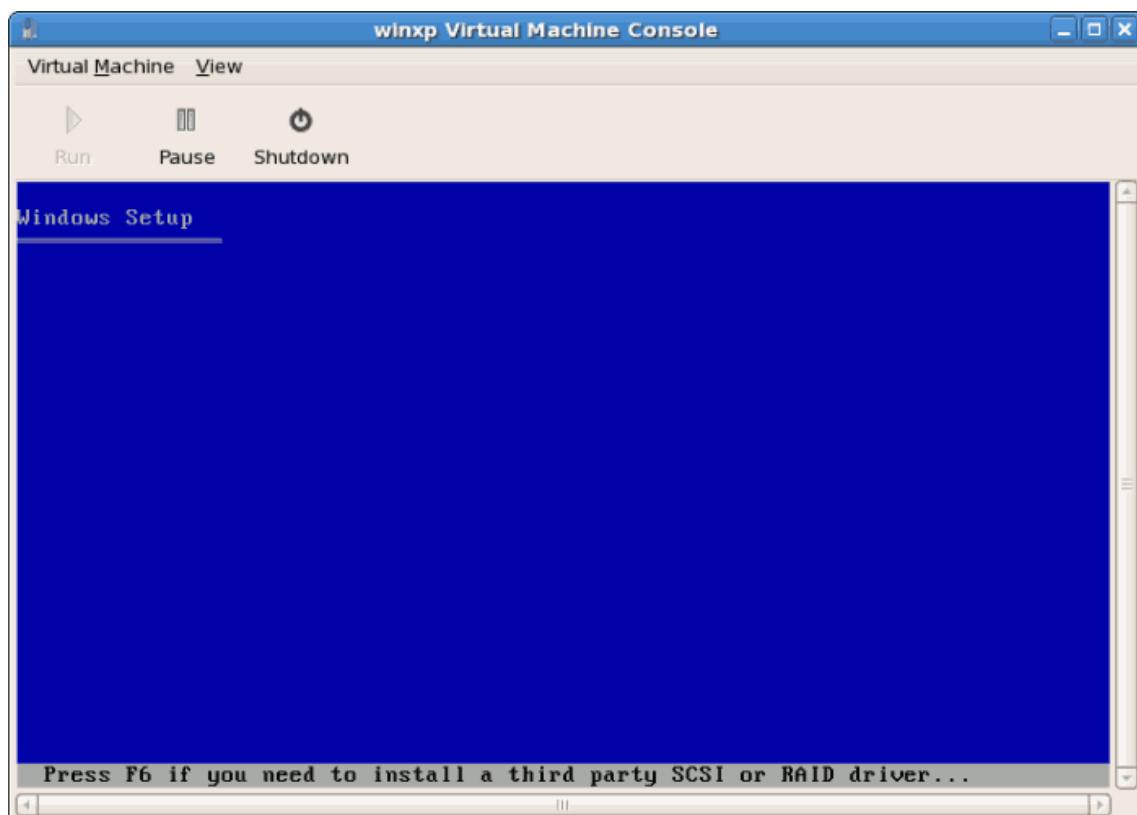
Assign sufficient virtual CPUs for the guest. If the guest runs a multithreaded application, assign the number of virtualized CPUs the guest will require to run efficiently. Do not assign more virtual CPUs than there are physical processors (or hyper-threads) available on the host system. It is possible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



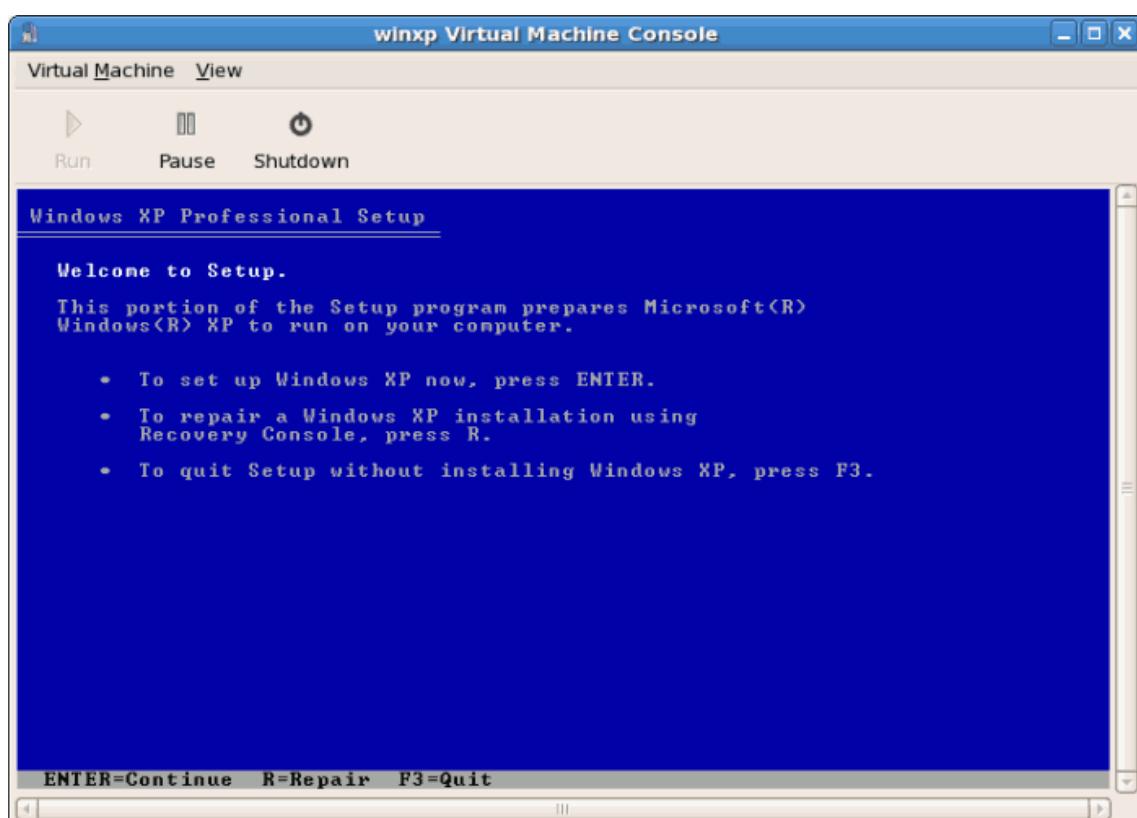
9. Before the installation continues you will see the summary screen. Press **Finish** to proceed to the guest installation:

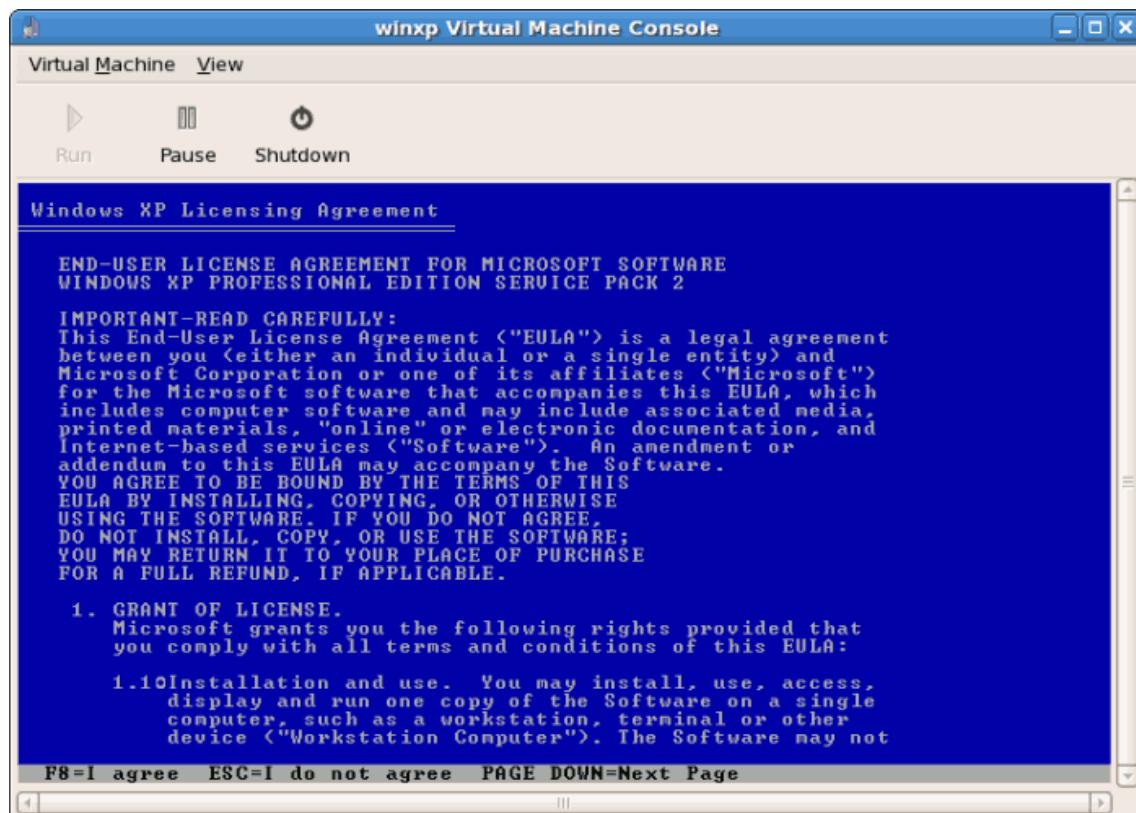


10. You must make a hardware selection so open a console window quickly after the installation starts. Click **Finish** then switch to the **virt-manager** summary window and select your newly started Windows guest. Double click on the system name and the console window opens. Quickly and repeatedly 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.

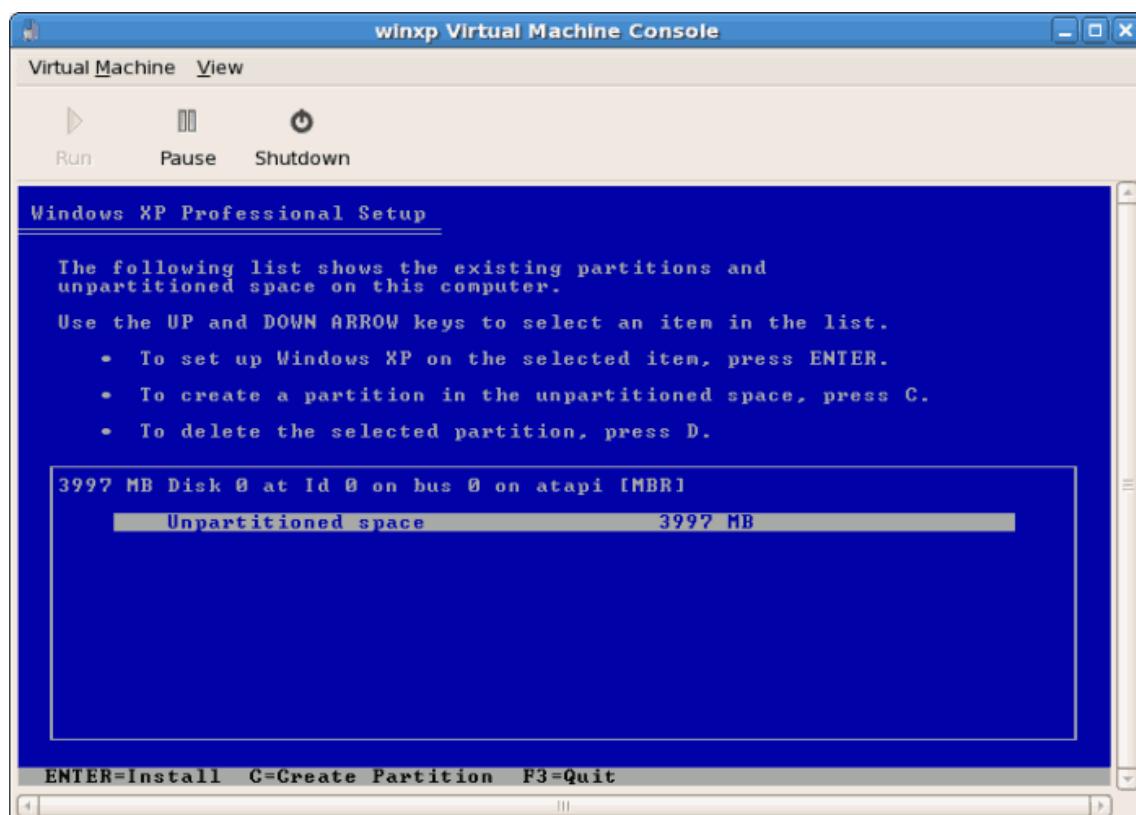


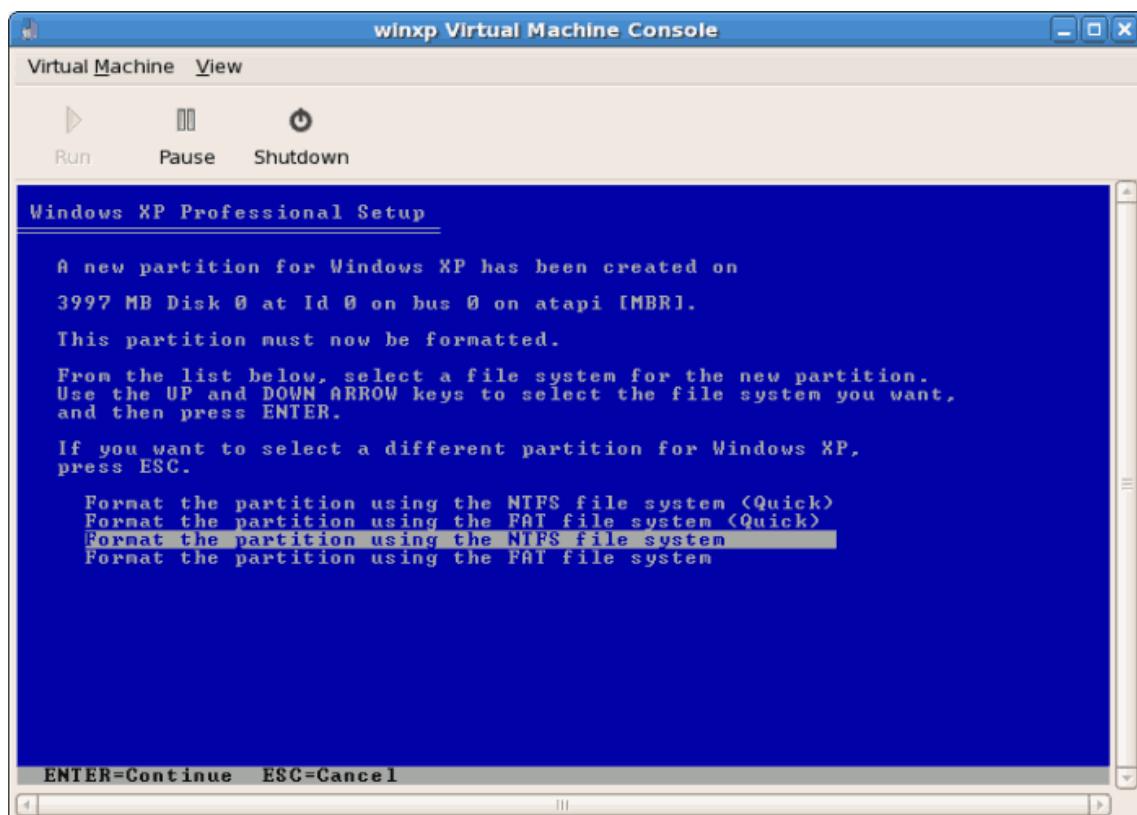
11. The installation continues with the standard Windows installation.



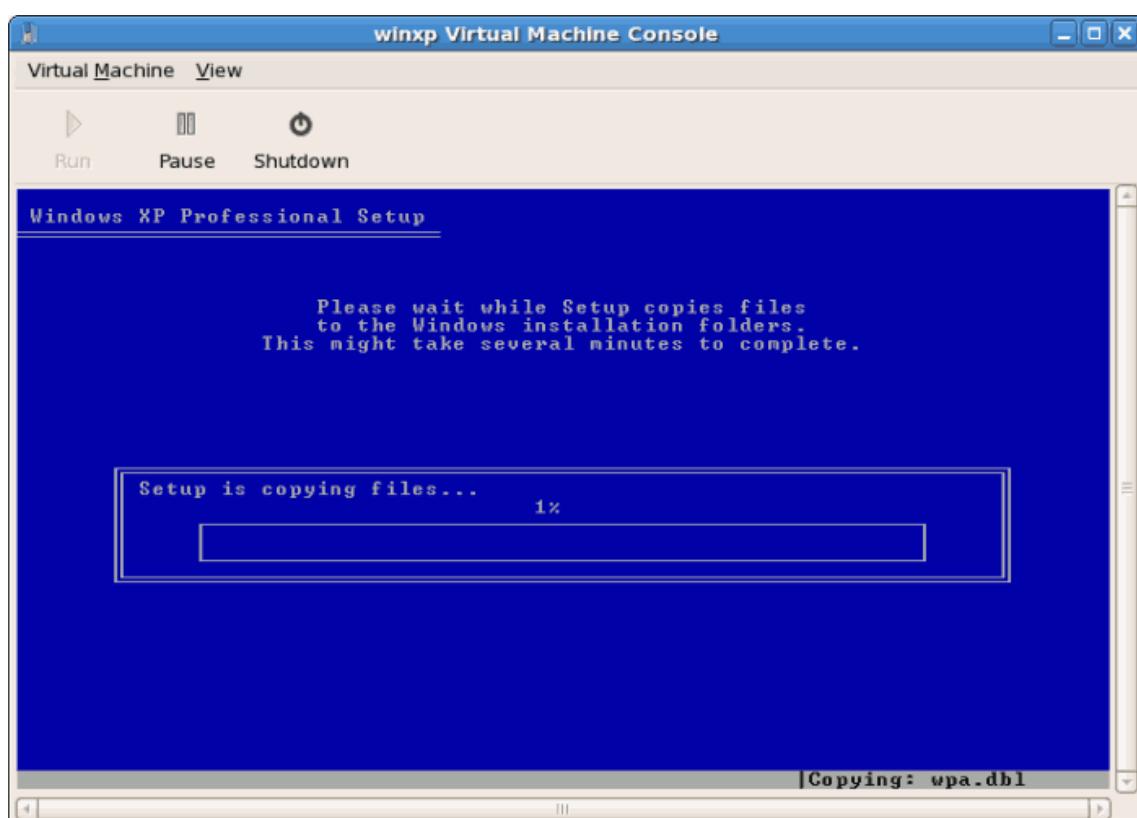


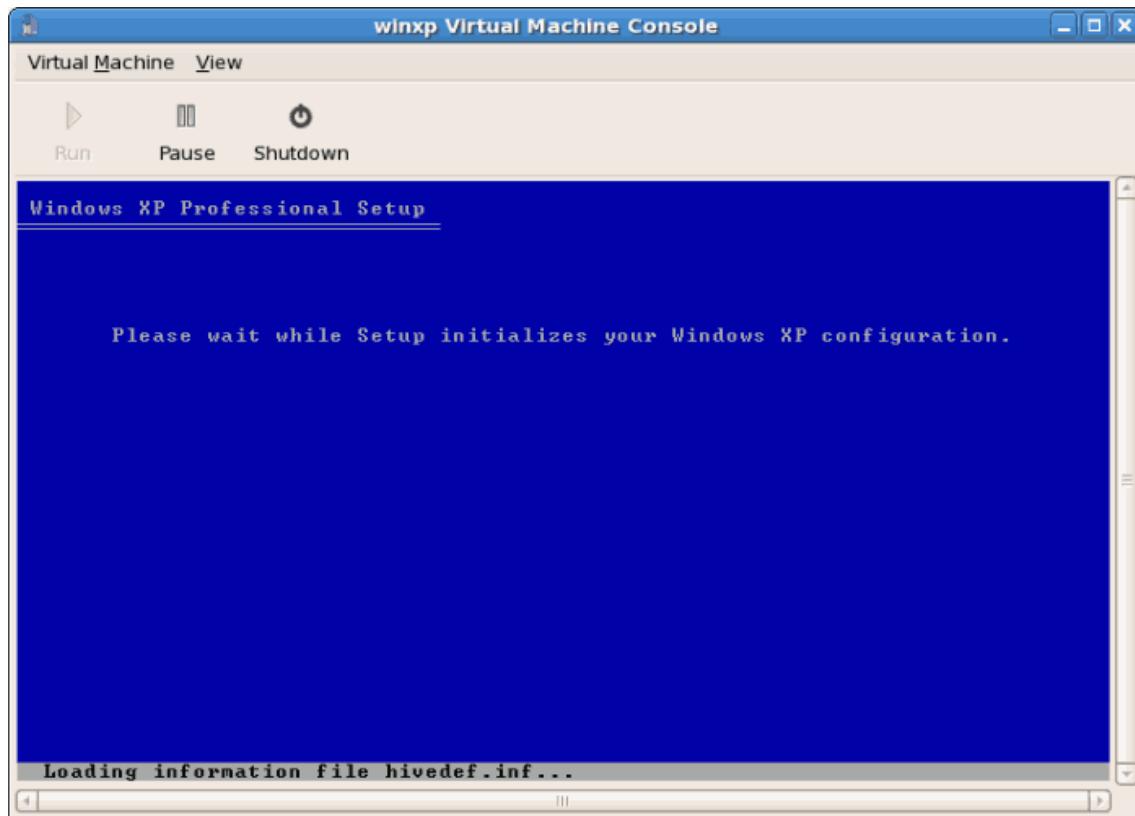
12. Partition the hard drive when prompted.



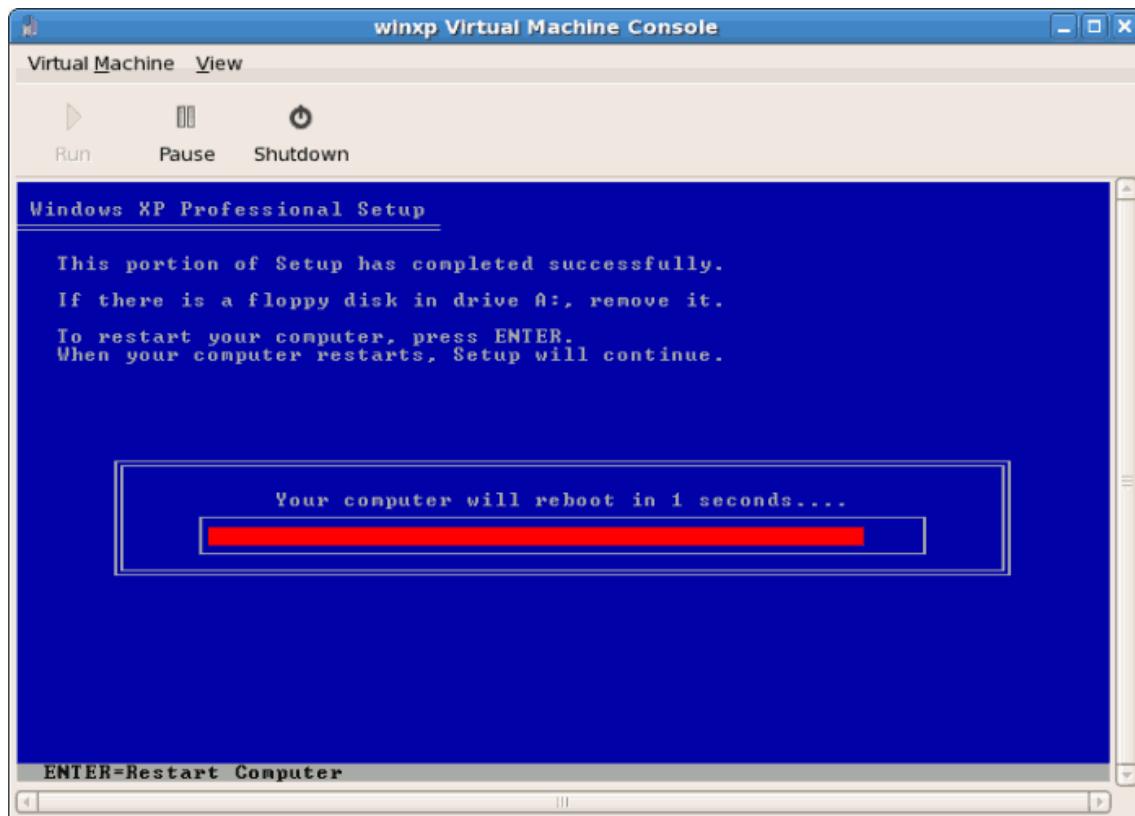


13. After the drive is formatted, Windows starts copying the files to the hard drive.





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

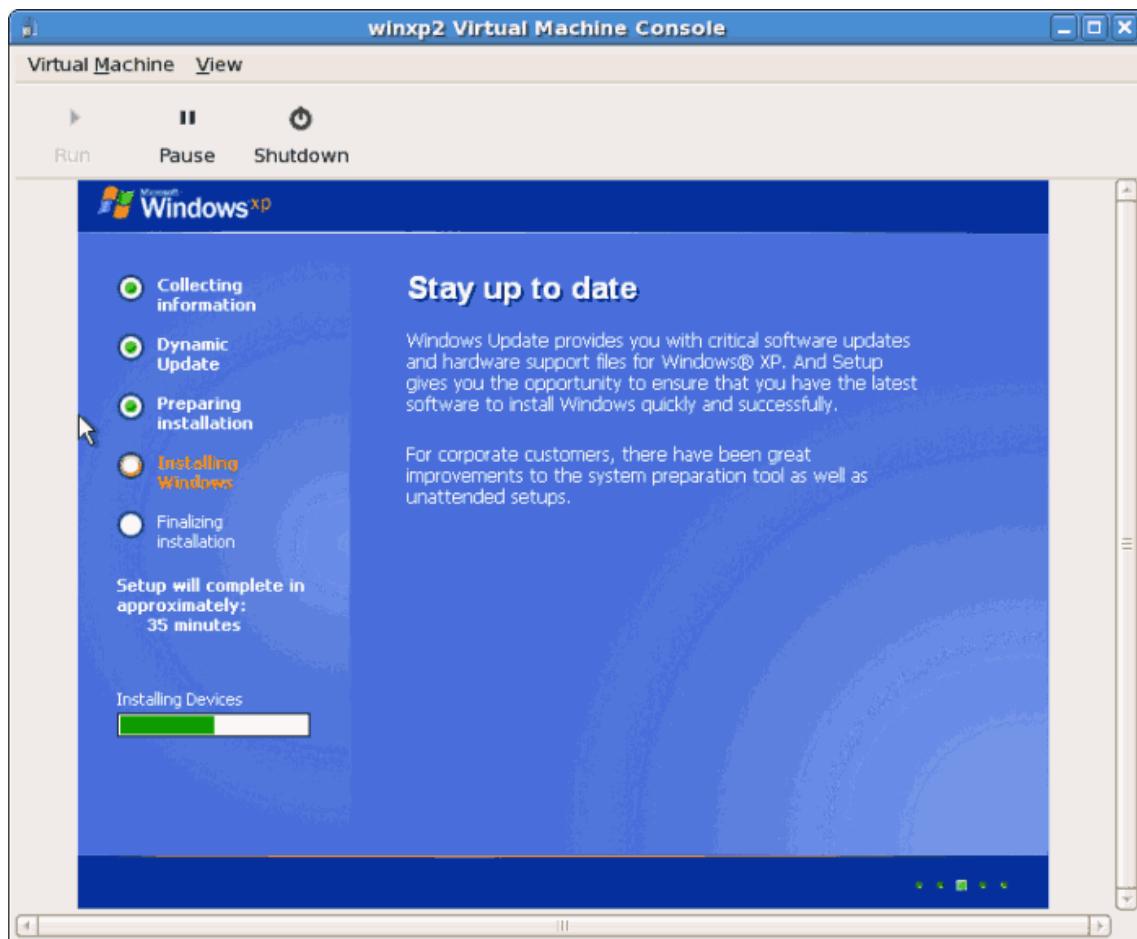


15. Restart your Windows guest:

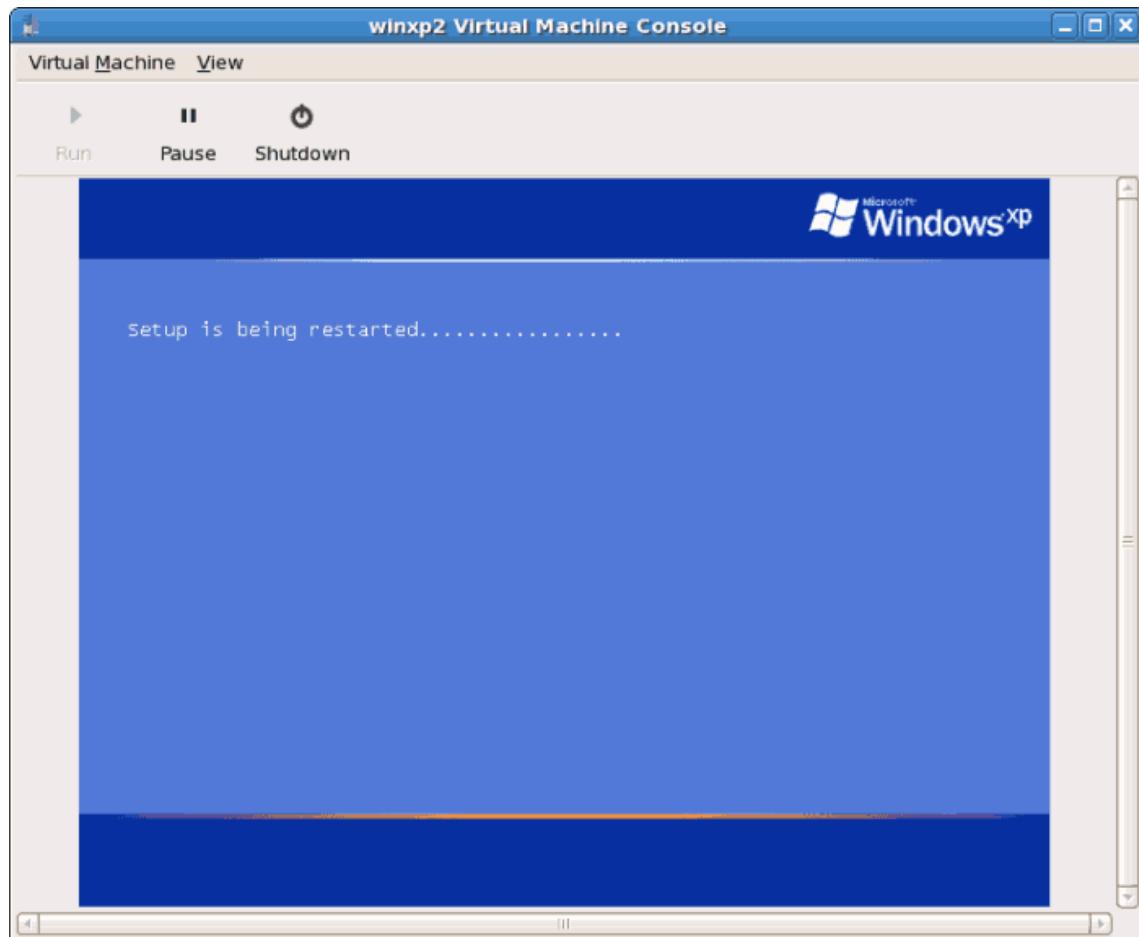
```
# virsh start WindowsGuest
```

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.



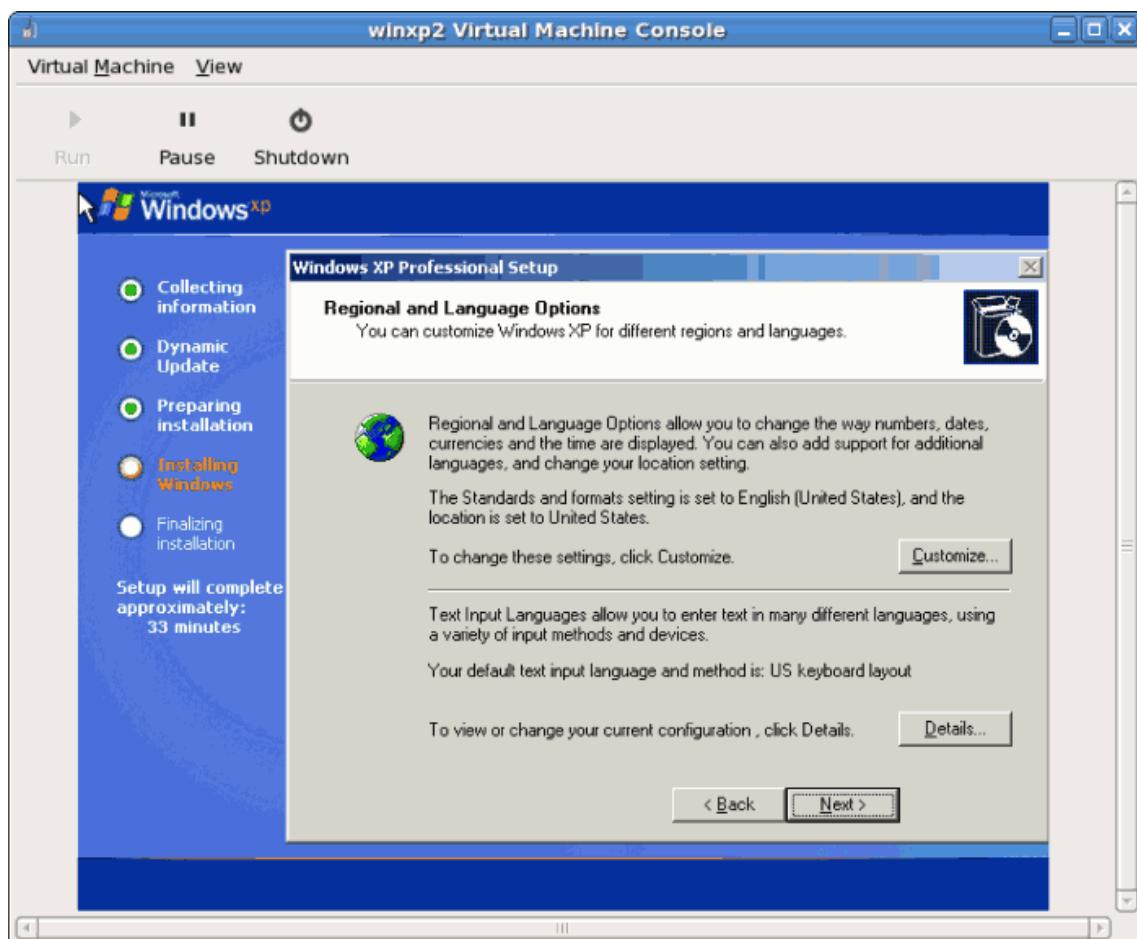
17. If your installation seems to get stuck during the setup phase, restart the guest with **virsh reboot *WindowsGuestName***. When you restart the virtual machine, the **Setup is being restarted** message displays:



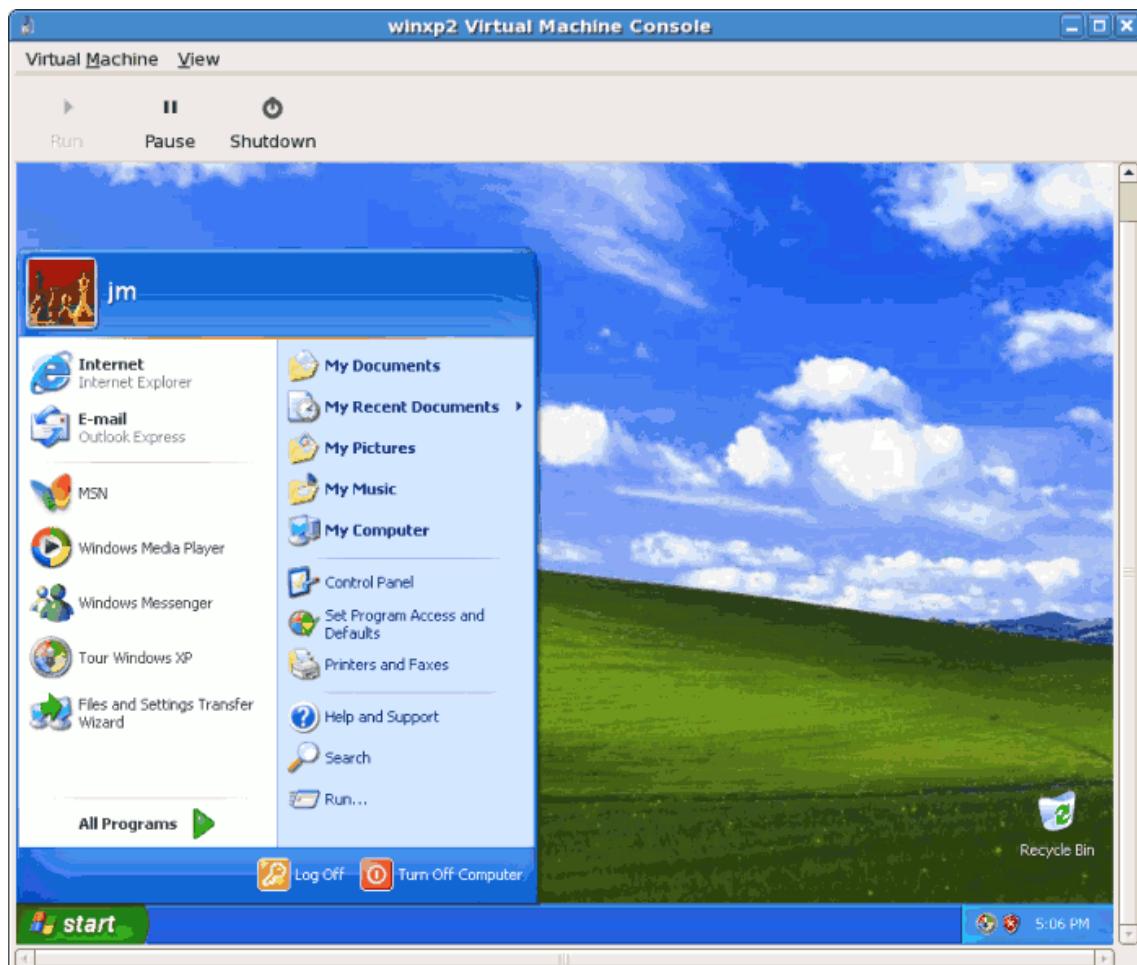
18. After setup has finished you will see the Windows boot screen:



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



20. The setup process is complete.



8.4. 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 [Section 8.3, “Installing Windows XP as a fully virtualized guest”](#).

Note

Presently, Red Hat Enterprise Linux hosts on the Itanium® architecture do not support fully virtualized windows guests. This section only applies to x86 and x86-64 hosts.

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.

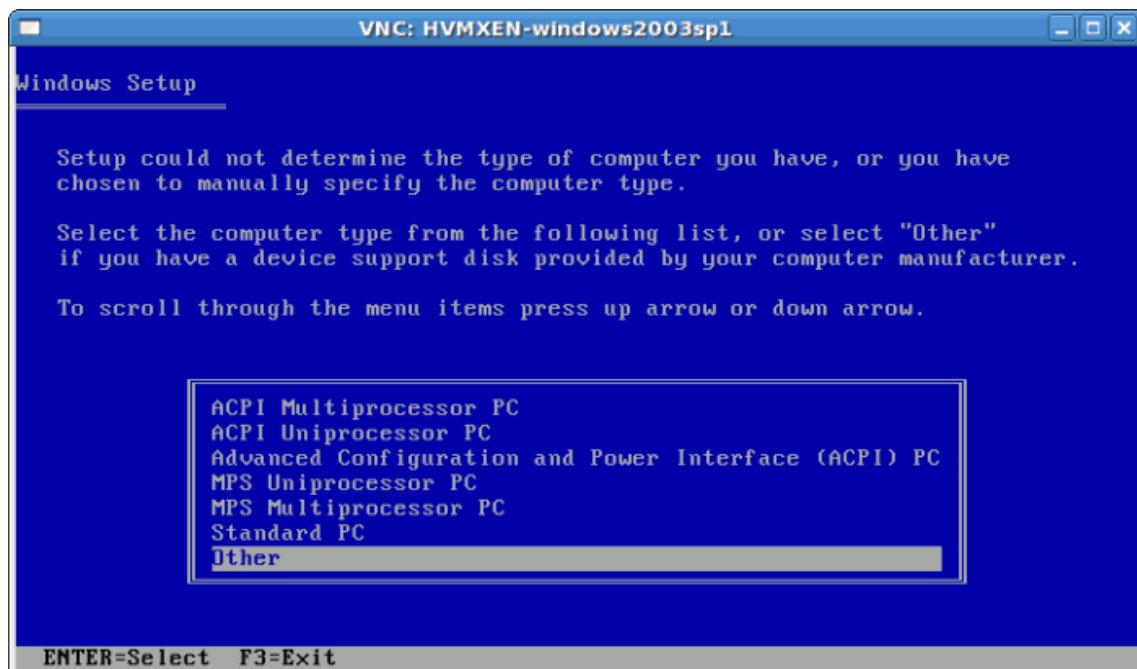
a. Xen `virt-install`

```
# virt-install --virt-type=xen -hvm \
--name windows2003sp1 \
--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
```

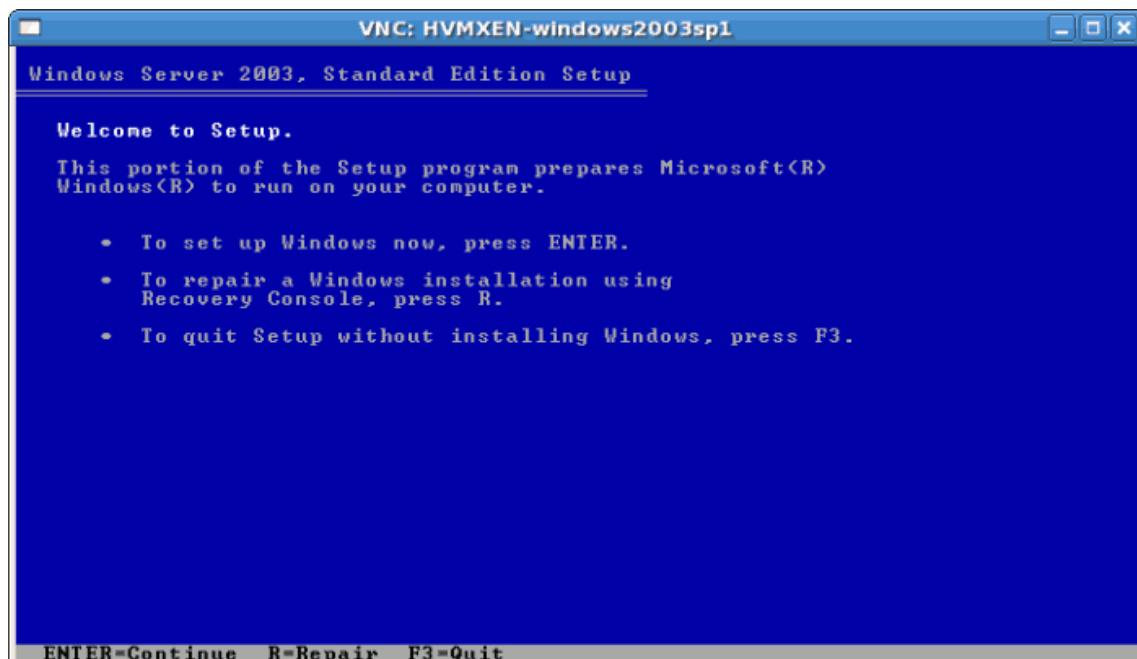
b. KVM `virt-install`

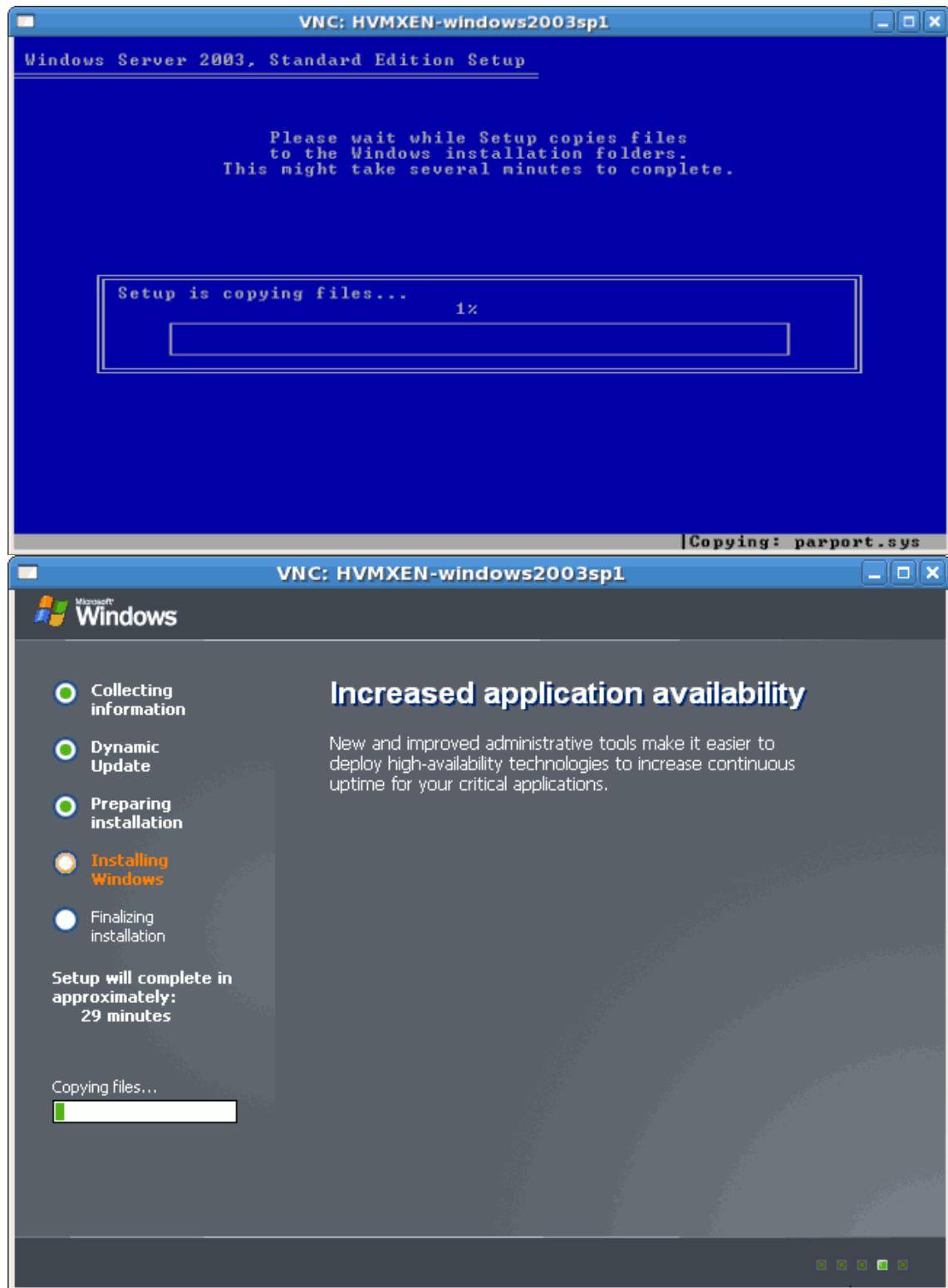
```
# virt-install --accelerate --hvm --connect qemu:///system \
--name rhel3support \
--network network:default \
--file=/var/lib/libvirt/images/windows2003sp2.img \
--file-size=6 \
-- \
cdrom=/var/lib/libvirt/images/ISOs/WIN/en_windows_server_2003_sp1.iso \
--vnc --ram=1024
```

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 guests.



3. Complete the rest of the installation.





- Windows Server 2003 is now installed as a fully guest.

8.5. Installing Windows Server 2008 as a fully virtualized guest

This section covers installing a fully virtualized Windows Server 2008 guest. This procedure covers both the KVM and the Xen hypervisors; the steps are interchangeable and different steps are noted.

The KVM hypervisor requires Red Hat Enterprise Linux 5.4 or newer.

Procedure 8.4. Installing Windows Server 2008 with virt-manager

-

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. If installed, select Xen or KVM. For this example, select KVM. Note that presently KVM 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 virtual machine

Provide a name for your guest. Punctuation and whitespace characters are not permitted in versions before Red Hat Enterprise Linux 5.5. Red Hat Enterprise Linux 5.5 adds support for '.', '.', '-' and '-' characters.

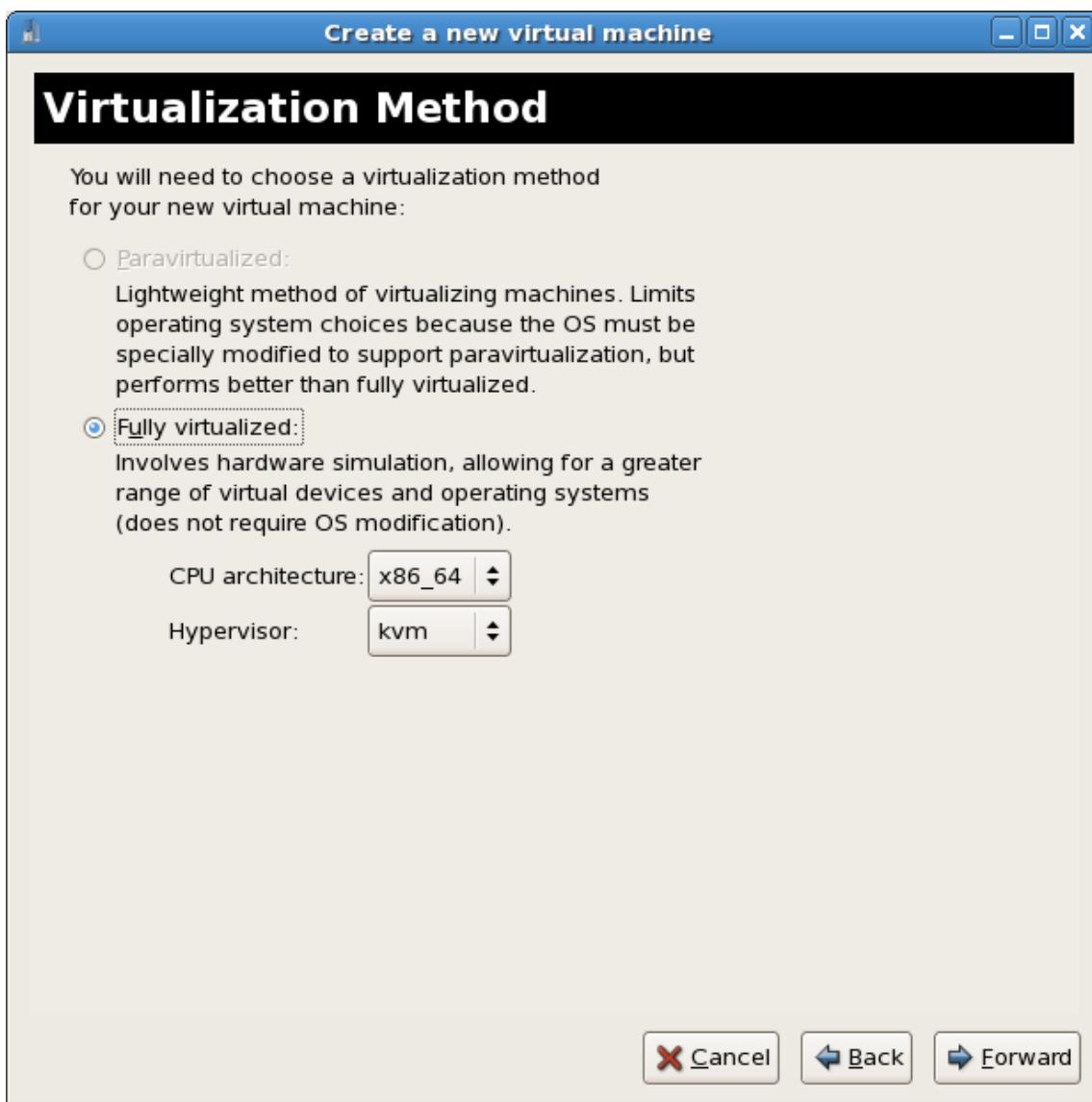


Press **Forward** to continue.

5.

Choose a virtualization method

Choose the virtualization method for the guest. Note you can only select an installed virtualization method. If you selected KVM or Xen earlier (step 2) you must use the hypervisor you selected. This example uses the KVM hypervisor.



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.



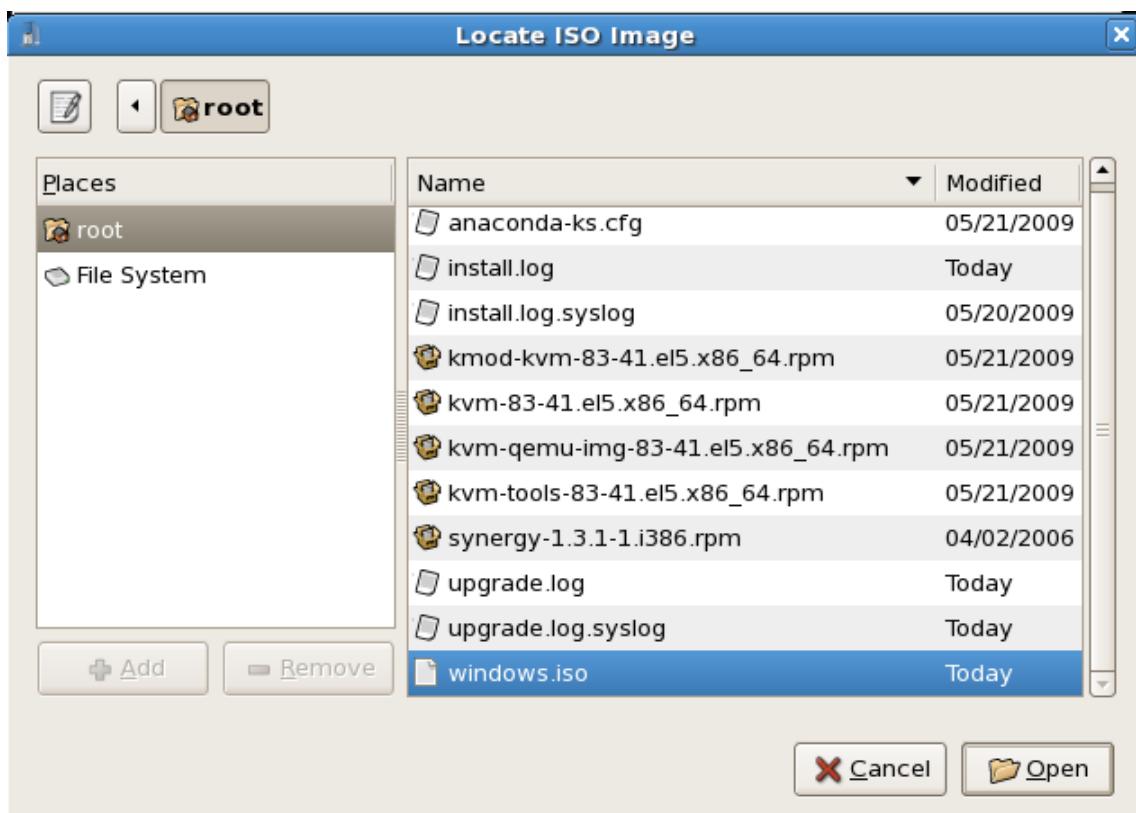
Press **Forward** to continue.

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.



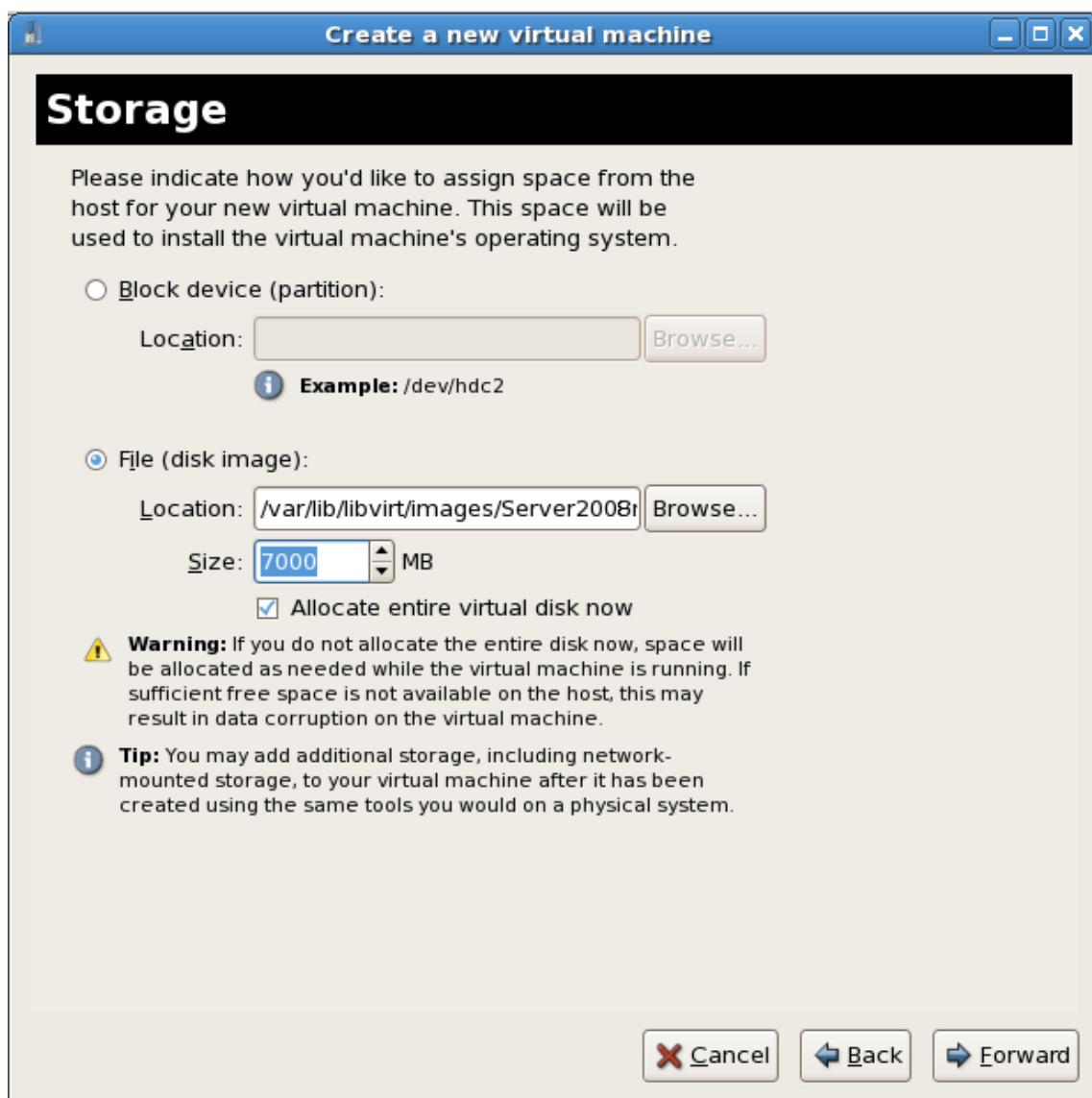
Warning

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, see [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 guest and any applications the guest requires.



Press **Forward** to continue.

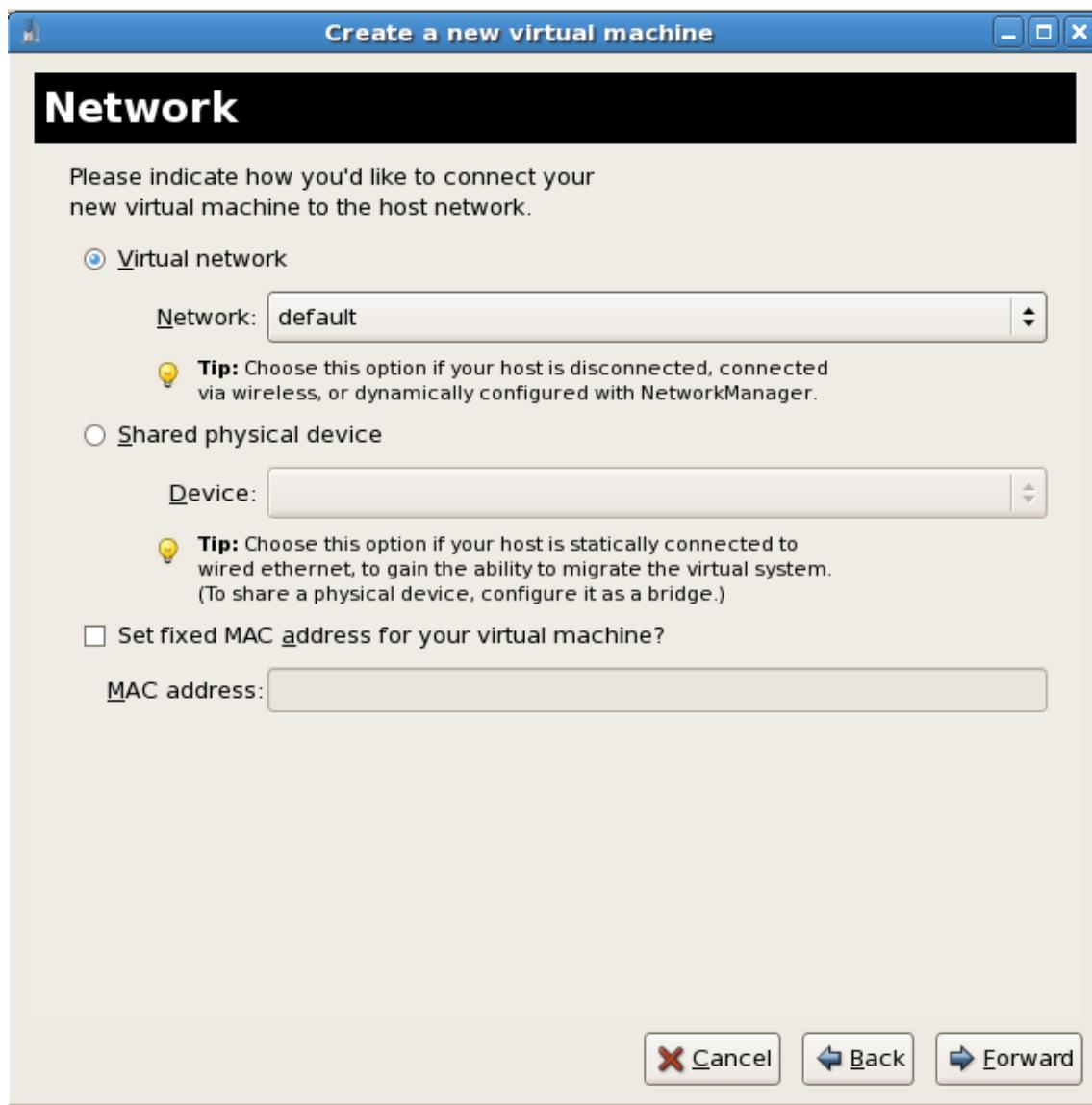
9.

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 guest. Use the virtual network option for wireless networks.

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



Press **Forward** to continue.

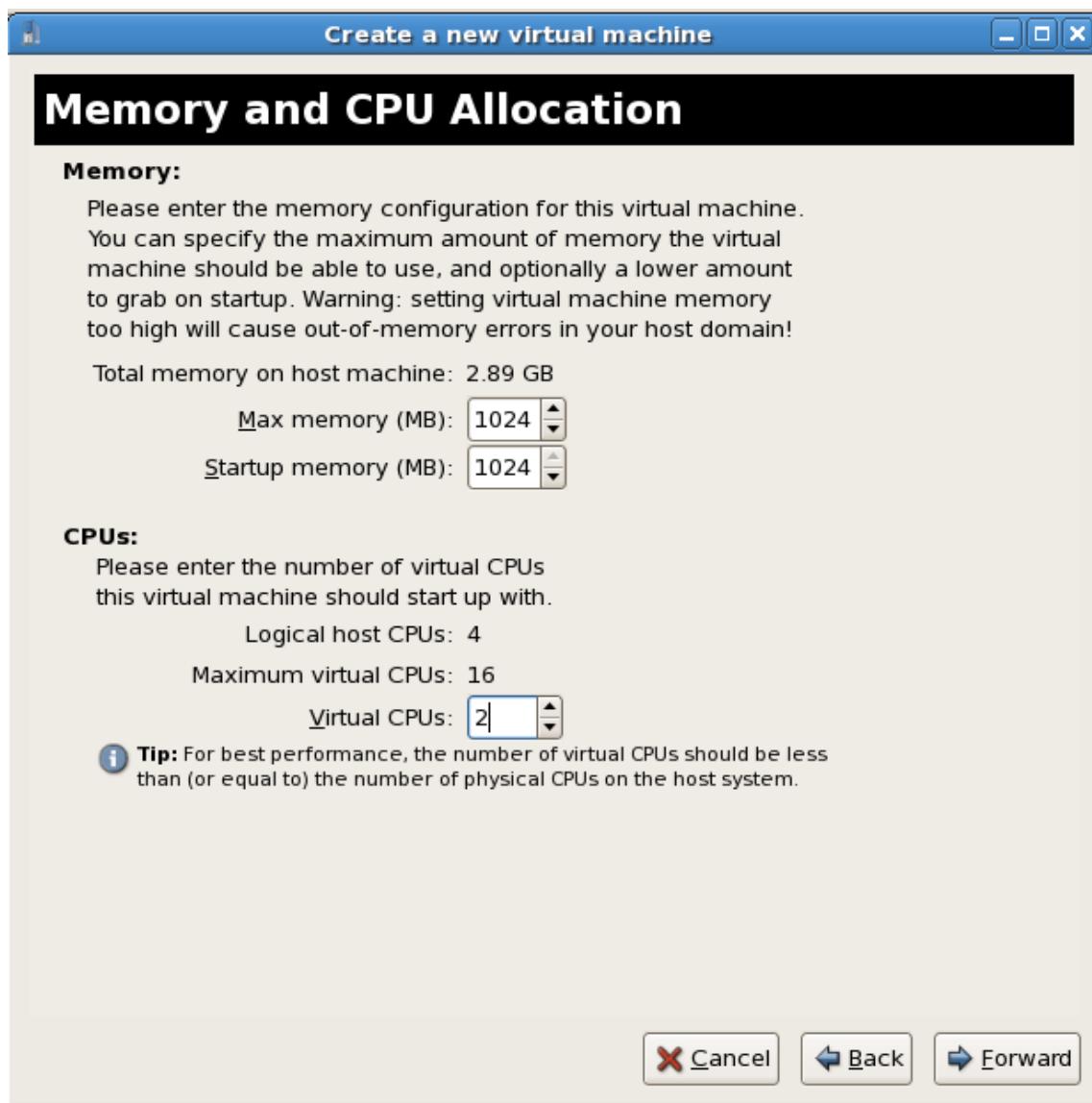
10.

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.

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 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

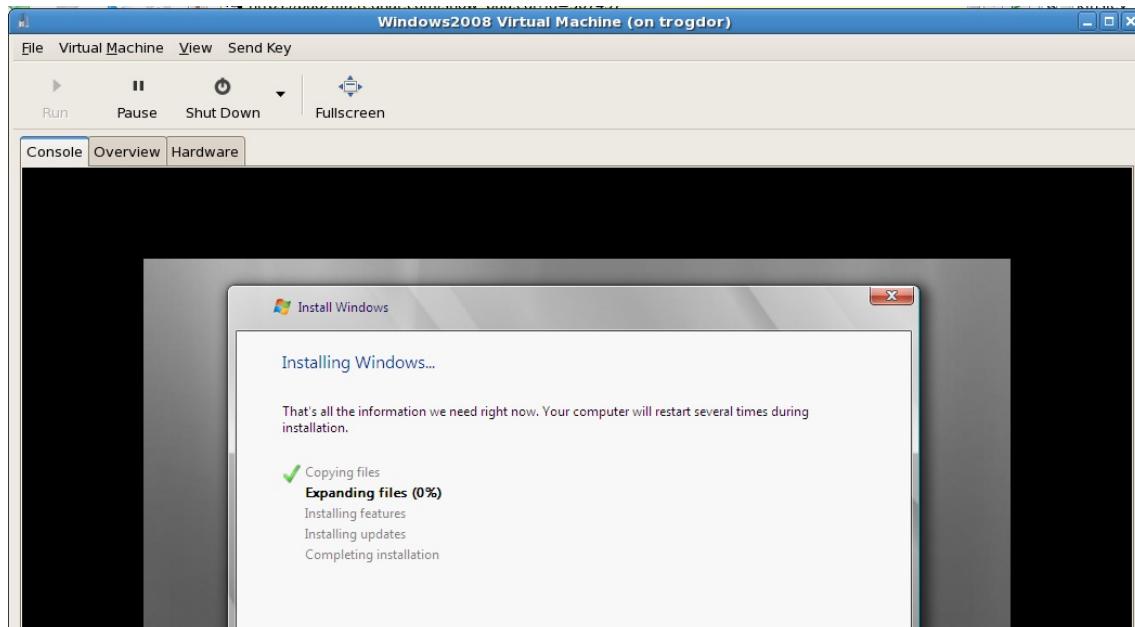
Verify the configuration.



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, see Microsoft's [documentation](#) for information on installing Windows.

Part III. Configuration

Configuring Virtualization in Red Hat Enterprise Linux

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.

Chapter 9. Virtualized storage devices

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



Important

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**

9.1. 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 virtual machines. However, creating and accessing floppy disk images from virtualized floppy drives is supported. This section covers creating a virtualized floppy device.

First, create an image file of the floppy disk with the **dd** command. Replace **/dev/fd0** with the name of a floppy device and name the disk image appropriately.

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



Note

The para-virtualized drivers can map physical floppy devices to fully virtualized guests. For more information on using para-virtualized drivers see [Chapter 13, KVM Para-virtualized Drivers](#).

This example uses a guest created with **virt-manager** running a fully virtualized Red Hat Enterprise Linux installation with an image located in **/var/lib/libvirt/images/rhel5FV.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 rhel5FV > 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, see [Chapter 34, Creating custom libvirt scripts](#).

2. Create a floppy disk image for the guest.

```
# dd if=/dev/zero of=/var/lib/libvirt/images/rhel5FV-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/rhel5FV-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 rhel5FV
```

5. Restart the guest using the XML configuration file.

```
# virsh create rhel5FV.xml
```

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

9.2. Adding storage devices to guests

This section covers adding storage devices to virtual machines. Additional storage can only be added after guests are created. The supported 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 virtual machines. 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).

- 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 be 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
```

- 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 4GB file which can be used as additional storage for a virtual machine.

- 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
```

- 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>
```

- 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>
```

- Restart the guest from the updated configuration file.

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

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

The guest now uses the file **FileName.img** as the device called **/dev/sdb**. 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.

- Press **n** for a new partition.

```
# fdisk /dev/sdb
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/sdb1
```

7. Mount the disk on the guest.

```
# mount /dev/sdb1 /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 9.1, “Adding physical block devices to virtual machines”](#), describes how to add a hard drive on the host to a guest.

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



Warning

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 virtual machines, have write access to whole partitions or LVM volumes the host system could be compromised.

Guests should not be given write access to whole disks or block devices (for example, **/dev/sdb**). Virtual machines 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 9.1. Adding physical block devices to virtual machines

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/sdb1** with the device to add, and *sdc* with the location for the device on the guest. The *sdc* must be an unused device name. Use the *sd** notation for Windows guests as well, the guest will recognize the device correctly.

Append the **--type cdrom** 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/sdb1
sdc --driver tap --mode readonly
```

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

9.3. Configuring persistent storage in Red Hat Enterprise Linux 5

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*
3600a0b800013275100000015427b625e
```

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="3600a0b800013275100000015427b625e", 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 3600a0b800013275100000015427b625e 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 Red Hat Enterprise Linux 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 3600805f30015987000000000768a0019
        alias oramp1
    }
    multipath {
        wwid 3600805f30015987000000000d643001a
        alias oramp2
    }
    multipath {
        wwid 3600805f3001598700000000086fc001b
        alias oramp3
    }
    multipath {
        wwid 3600805f300159870000000000984001c
        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.

9.4. 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.

Chapter 10. Network Configuration

This page provides an introduction to the common networking configurations used by libvirt based applications. This information applies to all hypervisors, whether Xen, KVM or another. 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 guests are not accessible by default from external hosts. You must enable either Network Address Translation (NAT) or a network bridge to allow external hosts access to network services on guests.

10.1. 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 known 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>
```

10.2. 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 Xen network scripts

If your system was using a Xen bridge, it is recommended to disable the default Xen network bridge by editing **/etc/xen/xend-config.sxp** and changing the line:

```
(network-script network-bridge)
```

To:

```
(network-script /bin/true)
```

Disable NetworkManager

NetworkManager does not support bridging. Running NetworkManager will overwrite any manual bridge configuration. Because of this, NetworkManager should be disabled in order to use networking via the network scripts (located in the **/etc/sysconfig/network-scripts/** directory):

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

 **Note**

As an alternative to turning off NetworkManager, add "**NM_CONTROLLED=no**" to the **ifcfg-*** scripts used in the examples. **If you do not either set this parameter or disable NetworkManager entirely, any bridge configuration will be overwritten and lost when NetworkManager next starts.**

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
```

 **Note**

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 name 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
```



Note

IP address configuration, be it dynamic or static, should be configured on the bridge itself (for example, in the **ifcfg-br0** file). Network access will not function as expected if IP address details are configured on the physical interface that the bridge is connected to.



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
```



Note

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            eth0
br0             8000.000e0cb30550  no
```

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.

Chapter 11. Pre-Red Hat Enterprise Linux 5.4 Xen networking

This chapter covers special topics for networking and network configuration with the Xen hypervisor.

Most guest network configuration occurs during the guest initialization and installation process. To learn about configuring networking during the guest installation process, read the relevant sections of the installation process, [Chapter 7, Guest installation overview](#).

Network configuration is also covered in the tool specific reference chapters for **virsh** ([Chapter 26, Managing guests with virsh](#)) and **virt-manager** ([Chapter 27, Managing guests with the Virtual Machine Manager \(virt-manager\)](#)). Those chapters provide a detailed description of the networking configuration tasks using both tools.

Note

Using para-virtualized network drivers improves performance on fully virtualized Linux guests. [Chapter 12, Xen Para-virtualized Drivers](#) explains how to utilize para-virtualized network drivers.

11.1. Configuring multiple guest network bridges to use multiple Ethernet cards

To set up network bridges (with the Xen hypervisor):

1. Configure another network interface using either the **system-config-network** application. Alternatively, create a new configuration file named **ifcfg-ethX** in the **/etc/sysconfig/network-scripts/** directory where **X** is any number not already in use. Below is an example configuration file for a second network interface called **eth1**:

```
$ cat /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
BOOTPROTO=static
ONBOOT=yes
USERCTL=no
IPV6INIT=no
PEERDNS=yes
TYPE=Ethernet
NETMASK=255.255.255.0
IPADDR=10.1.1.1
GATEWAY=10.1.1.254
ARP=yes
```

2. Copy the file **/etc/xen/scripts/network-bridge** to **/etc/xen/scripts/network-bridge.xen**.
3. Comment out any existing network scripts in **/etc/xen/xend-config.sxp** and add the line (**network-xen-multi-bridge**).

A typical **xend-config.sxp** file should have the following line. Comment this line out. Use the **#** symbol to comment out lines.

```
network-script network-bridge
```

Below is the commented out line and the new line, containing the ***network-xen-multi-bridge*** parameter to enable multiple network bridges:

```
#network-script network-bridge
network-script network-xen-multi-bridge
```

4. Create a script to create multiple network bridges. This example creates a script called ***network-xen-multi-bridge.sh*** in the ***/etc/xen/scripts/*** directory. The following example script will create two Xen network bridges (***xenbr0*** and ***xenbr1***); one will be attached to ***eth1*** and the other one to ***eth0***. To create additional bridges, follow the example in the script and copy and paste the lines as required:

```
#!/bin/sh
# network-xen-multi-bridge
# Exit if anything goes wrong.
set -e
# First arg is the operation.
OP=$1
shift
script=/etc/xen/scripts/network-bridge.xen
case ${OP} in
start)
$script start vifnum=1 bridge=xenbr1 netdev=eth1
$script start vifnum=0 bridge=xenbr0 netdev=eth0
;;
stop)
$script stop vifnum=1 bridge=xenbr1 netdev=eth1
$script stop vifnum=0 bridge=xenbr0 netdev=eth0
;;
status)
$script status vifnum=1 bridge=xenbr1 netdev=eth1
$script status vifnum=0 bridge=xenbr0 netdev=eth0
;;
*)
echo 'Unknown command: ' ${OP}
echo 'Valid commands are: start, stop, status'
exit 1
esac
```

5. Make the script executable.

```
# chmod +x /etc/xen/scripts/network-xen-multi-bridge.sh
```

6. Restart networking or restart the system to activate the bridges.

```
# service network restart
```

Multiple bridges should now be configured for guests on the Xen hypervisor.

11.2. Red Hat Enterprise Linux 5.0 laptop network configuration



Important

This section describes manually adding network bridges. This procedure is not required or recommended for all versions of Red Hat Enterprise Linux newer than version 5.0. For newer versions use "**Virtual Network**" adapters when creating guests in **virt-manager**. **NetworkManager** works with virtual network devices by default in Red Hat Enterprise Linux 5.1 and newer.

An example of a virsh XML configuration file for a virtual network device:

```
<interface type='network'>
  <mac address='AA:AA:AA:AA:AA:AA' />
  <source network='default' />
  <target dev='vnet0' />
  <model type='virtio' />
</interface>
```

In **xm** configuration files, virtual network devices are labeled "**vif**".

The challenge in running the Xen hypervisor on a laptop is that most laptops will connect to the network via wireless network or wired connections. Often these connections are switched multiple times a day. In such an environment, the system assumes it has access to the same interface all the time and it also can perform **ifup** or **ifdown** calls to the network interface it is using. In addition wireless network cards do not work well in a virtualization environment due to Xen's (default) bridged network usage.

This setup will also enable you to run Xen in offline mode when you have no active network connection on your laptop. The easiest solution to run Xen on a laptop is to follow the procedure outlined below:

- » You will be configuring a 'dummy' network interface which will be used by Xen. In this example the interface is called **dummy0**. This will also allow you to use a hidden IP address space for your guests.
- » You will need to use static IP address as DHCP will not listen on the dummy interface for DHCP requests. You can compile your own version of DHCP to listen on dummy interfaces, however you may want to look into using dnsmasq for DNS, DHCP and tftpboot services in a Xen environment. Setup and configuration are explained further down in this section/chapter.
- » You can also configure NAT and IP masquerading in order to enable access to the network from your guests.

Configuring a dummy network interface

Perform the following configuration steps on your host:

1. Create a **dummy0** network interface and assign it a static IP address. In our example I selected 10.1.1.1 to avoid routing problems in our environment. To enable dummy device support add the following lines to **/etc/modprobe.conf**:

```
alias dummy0 dummy
options dummy numdummies=1
```

2. To configure networking for **dummy0**, edit or create **/etc/sysconfig/network-scripts/ifcfg-dummy0**:

```
DEVICE=dummy0
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
IPV6INIT=no
PEERDNS=yes
TYPE=Ethernet
NETMASK=255.255.255.0
IPADDR=10.1.1.1
ARP=yes
```

- Bind **xenbr0** to **dummy0**, so you can use networking even when not connected to a physical network. Edit **/etc/xen/xend-config.sxp** to include the **netdev=dummy0** entry:

```
(network-script 'network-bridge bridge=xenbr0 netdev=dummy0')
```

- Open **/etc/sysconfig/network** in the guest and modify the default gateway to point to **dummy0**. If you are using a static IP, set the guest's IP address to exist on the same subnet as **dummy0**.

```
NETWORKING=yes
HOSTNAME=localhost.localdomain
GATEWAY=10.1.1.1
IPADDR=10.1.1.10
NETMASK=255.255.255.0
```

- Setting up NAT in the host will allow the guests access Internet, including with wireless, solving the Xen and wireless card issues. The script below will enable NAT based on the interface currently used for your network connection.

Configuring NAT for guests

Network address translation (NAT) allows multiple network address to connect through a single IP address by intercepting packets and passing them to the private IP addresses. You can copy the following script to **/etc/init.d/xenLaptopNAT** and create a soft link to **/etc/rc3.d/S99xenLaptopNAT**. This automatically starts NAT at boot time.

Note

The script below may not work well with wireless network or **NetworkManager** due to start-up delays. In this case run the script manually once the machine has booted.

```
#!/bin/bash
PATH=/usr/bin:/sbin:/bin:/usr/sbin
export PATH
GATEWAYDEV=`ip route | grep default | awk {'print $5'}`
iptables -F
case "$1" in
start)
  if test -z "$GATEWAYDEV"; then
    echo "No gateway device found"
    else
      echo "Masquerading using $GATEWAYDEV"
      /sbin/iptables -t nat -A POSTROUTING -o $GATEWAYDEV -j MASQUERADE
  fi
  echo "Enabling IP forwarding"
  echo 1 > /proc/sys/net/ipv4/ip_forward
```

```

echo "IP forwarding set to `cat /proc/sys/net/ipv4/ip_forward`"
echo "done."
;;
*)
echo "Usage: $0 {start|restart|status}"
;;
esac

```

Configuring dnsmasq for the DNS, DHCP and tftpboot services

One of the challenges in running virtualization on a laptop (or any other computer which is not connected by a single or stable network connection) is the change in network interfaces and availability. Using a dummy network interface helps to build a more stable environment but it also brings up new challenges in providing DHCP, DNS and tftpboot services to your guest virtual machines. The default DHCP daemon shipped with Red Hat Enterprise Linux and Fedora Core will not listen on dummy interfaces, and your DNS forwarded information may change as you connect to different networks and VPNs.

One solution to the above challenges is to use **dnsmasq**, which can provide all of the above services in a single package, and also allows you to configure services to be available only to requests from your dummy interface. Below is a short write up on how to configure **dnsmasq** on a laptop running virtualization:

- » Get the latest version of **dnsmasq** from [here](#).
- » Documentation for **dnsmasq** can be found [here](#).
- » Copy the other files referenced below from <http://et.redhat.com/~jmh/tools/xen/> and grab the file **dnsmasq.tgz**. The tar archive includes the following files:
 - **nm-dnsmasq** can be used as a dispatcher script for **NetworkManager**. It will be run every time **NetworkManager** detects a change in connectivity and force a restart or reload of **dnsmasq**. It should be copied to **/etc/NetworkManager/dispatcher.d/nm-dnsmasq**
 - **xenDNSmasq** can be used as the main startup or shutdown script for **/etc/init.d/xenDNSmasq**
 - **dnsmasq.conf** is a sample configuration file for **/etc/dnsmasq.conf**
 - **dnsmasq** is the binary image for **/usr/local/sbin/dnsmasq**
- » Once you have unpacked and built **dnsmasq** (the default installation will be the binary into **/usr/local/sbin/dnsmasq**) you need to edit your **dnsmasq** configuration file. The file is located in **/etc/dnsmasq.conf**.
- » Edit the configuration to suit your local needs and requirements. The following parameters are likely the ones you want to modify:
 - The **interface** parameter allows **dnsmasq** to listen for **DHCP** and **DNS** requests only on specified interfaces (such as dummy interfaces). Note that **dnsmasq** cannot listen to public interfaces as well as the local loopback interface simultaneously. Add another **interface** line for more than one interface. **interface=dummy0** is an example which listens on the **dummy0** interface.
 - Modify **dhcp-range** to enable the integrated **DHCP** server. You will need to supply the range of addresses available for lease and optionally a lease time. If you have more than one network, you will need to repeat this for each network on which you want to supply **DHCP** service. An example would be (for network 10.1.1.* and a lease time of 12 hours): **dhcp-**

```
range=10.1.1.10,10.1.1.50,255.255.255.0,12h
```

- Modify **dhcp-option** to override the default route supplied by **dnsmasq**, which assumes the router is the same machine as the one running **dnsmasq**. An example would be **dhcp-option=3,10.1.1.1**
- » After configuring **dnsmasq** you can copy the script below as **xenDNSmasq** to **/etc/init.d**
- » If you want to automatically start **dnsmasq** during system boot, you should register it using **chkconfig(8)**:

```
chkconfig --add xenDNSmasq
```

Enable it for automatic startup:

```
chkconfig --levels 345 xenDNSmasq on
```

- » To configure **dnsmasq** to restart every time **NetworkManager** detects a change in connectivity you can use the supplied script **nm-dnsmasq**.
- Copy the **nm-dnsmasq** script to **/etc/NetworkManager/dispatcher.d**
- The **NetworkManager** dispatcher will execute the script (in alphabetical order if you have other scripts in the same directory) every time there is a change in connectivity.
- » **dnsmasq** will also detect changes in your **/etc/resolv.conf** and automatically reload them (if you start up a VPN session, for example).
- » Both the **nm-dnsmasq** and **xenDNSmasq** script will also set up NAT if you have your guests on a hidden network to allow them access to the public network.

Chapter 12. Xen Para-virtualized Drivers

Para-virtualized drivers provide increased performance for fully virtualized Red Hat Enterprise Linux guests. Use these drivers if you are using fully virtualized Red Hat Enterprise Linux guests and require better performance.

Note

There are other para-virtualized drivers for Windows for both Xen and KVM hypervisors.

For Windows guests on Xen hosts, see the *Red Hat Enterprise Linux 5 Para-virtualized Windows Drivers Guide*, which covers installation and administration.

For Windows guests on KVM hosts, see [Chapter 13, KVM Para-virtualized Drivers](#).

The RPM packages for the para-virtualized drivers include the modules for storage and networking para-virtualized drivers for the supported Red Hat Enterprise Linux guest operating systems. These drivers enable high performance throughput of I/O operations in unmodified Red Hat Enterprise Linux guest operating systems on top of a Red Hat Enterprise Linux 5.1 (or greater) host.

The supported guest operating systems are:

- » Red Hat Enterprise Linux 3
- » Red Hat Enterprise Linux 4
- » Red Hat Enterprise Linux 5
- » Red Hat Enterprise Linux 6

Note

The minimum guest operating system requirements are architecture dependent. Only x86 and x86-64 guests are supported.

The drivers are not supported on Red Hat Enterprise Linux guest operating systems prior to Red Hat Enterprise Linux 3 .

Using Red Hat Enterprise Linux 5 as the virtualization platform allows system administrators to consolidate Linux and Windows workloads onto newer, more powerful hardware with increased power and cooling efficiency. Red Hat Enterprise Linux 4 (as of update 6) and Red Hat Enterprise Linux 5 guest operating systems are aware of the underlying virtualization technology and can interact with it efficiently using specific interfaces and capabilities. This approach can achieve similar throughput and performance characteristics compared to running on the bare metal system.

As para-virtualization requires a modified guest operating system, not all operating systems can use para-virtualization. For operating systems which can not be modified, full virtualization is required. Full virtualization, by default, uses emulated disk, network, video and other hardware devices. Emulated I/O devices can be very slow and are not suited for applications requiring high disk and/or network throughput. The majority of the performance loss with virtualization occurs through the use of emulated devices.

The para-virtualized device drivers are included in the Red Hat Enterprise Linux packages. The

drivers bring many of the performance advantages of para-virtualized guest operating systems to unmodified operating systems because only the para-virtualized device driver (but not the rest of the operating system) is aware of the underlying virtualization platform.

After installing the para-virtualized device drivers, a disk device or network card will continue to appear as a normal, physical disk or network card to the operating system. However, now the device driver interacts directly with the virtualization platform (with no emulation) to efficiently deliver disk and network access, allowing the disk and network subsystems to operate at near native speeds even in a virtualized environment, without requiring changes to existing guest operating systems.

The para-virtualized drivers have certain host requirements. 64 bit hosts can run:

- » 32 bit guests.
- » 64 bit guests.
- » a mixture of 32 bit and 64 bit guests.

12.1. System requirements

This section provides the requirements for para-virtualized drivers with Red Hat Enterprise Linux.

Installation

Before you install the para-virtualized drivers the following requirements (listed below) must be met.



Important

All versions of Red Hat Enterprise Linux from 4.7 and 5.3 have the kernel module for the para-virtualized drivers, the **pv-on-hvm** module, in the default kernel package. That means the para-virtualized drivers are available for Red Hat Enterprise Linux 4.7 and newer or 5.3 and newer guests.



Note

The para-virtualized guest drivers are enabled by default for guests running Red Hat Enterprise Linux 6.1 and newer on Xen hosts. Using the **xen_pv_hvm=enable** kernel boot parameter is no longer required for these guests.

You will need the following RPM packages for para-virtualized drivers for each guest operating system installation.

Minimum host operating system version:

- » Red Hat Enterprise Linux 5.1 or newer.

Minimum guest operating system version:

- » Red Hat Enterprise Linux 5.1 or newer.
- » Red Hat Enterprise Linux 4 Update 6 or newer.
- » Red Hat Enterprise Linux 3 Update 9 or newer.

Red Hat Enterprise Linux 5 requires:

- » **kmod-xenpv.**

Red Hat Enterprise Linux 4 requires:

- » **kmod-xenpv,**
- » **modules-init-tools** (for versions prior to Red Hat Enterprise Linux 4.6z you require **modules-init-tools-3.1-0.pre5.3.4.el4_6.1** or greater), and
- » **modversions.**

Red Hat Enterprise Linux 3 requires:

- » **kmod-xenpv.**

You require at least 50MB of free disk space in the **/lib** file system.

12.2. Para-virtualization Restrictions and Support

This section outlines support restrictions and requirements for using para-virtualized drivers on Red Hat Enterprise Linux.

Supported Guest Operating Systems

Support for para-virtualized drivers is available for the following operating systems and versions:

- » Red Hat Enterprise Linux 5.1 and newer.
- » Red Hat Enterprise Linux 4 Update 6 and newer.
- » Red Hat Enterprise Linux 3 Update 9 and newer.

You are supported for running a 32 bit guest operating system with para-virtualized drivers on 64 bit Red Hat Enterprise Linux 5 Virtualization.

The table below indicates the kernel variants supported with the para-virtualized drivers. You can use the command shown below to identify the exact kernel revision currently installed on your host. Compare the output against the table to determine if it is supported.

```
# rpm -q --queryformat '%{NAME}-%{VERSION}-%{RELEASE}.%{ARCH}\n' kernel
```

The Red Hat Enterprise Linux 5 i686 and x86_64 kernel variants include Symmetric Multiprocessing(SMP), no separate SMP kernel RPM is required.

Take note of processor specific kernel requirements for Red Hat Enterprise Linux 3 Guests in the table below.

Table 12.1. Supported guest kernel architectures for para-virtualized drivers

Kernel Architecture	Red Hat Enterprise Linux 3	Red Hat Enterprise Linux 4	Red Hat Enterprise Linux 5
athlon	Supported (AMD)		
athlon-SMP	Supported (AMD)		
i32e	Supported (Intel)		
i686	Supported (Intel)	Supported	Supported

Kernel Architecture	Red Hat Enterprise Linux 3	Red Hat Enterprise Linux 4	Red Hat Enterprise Linux 5
i686-PAE			Supported
i686-SMP	Supported (Intel)	Supported	
i686-HUGEMEM	Supported (Intel)	Supported	
x86_64	Supported (AMD)	Supported	Supported
x86_64-SMP	Supported (AMD)	Supported	
x86_64-LARGESMP		Supported	
Itanium (IA64)			Supported



Important

The host system requires Red Hat Enterprise Linux 5.1 or newer.



Note

Write the output of the command below down or remember it. This is the value that determines which packages and modules you need to download.

```
# rpm -q --queryformat '%{NAME}- %{VERSION}-%{RELEASE}.%{ARCH}\n'
kernel
```

Your output should appear similar to this:

kernel-PAE-2.6.18-53.1.4.el5.i686

The name of the kernel is PAE (an abbreviation of Physical Address Extensions), kernel version is 2.6.18, the release is 53.1.4.el5 and the architecture is i686. The kernel RPM should always be in the format **kernel-name-version-release.arch.rpm**.

Important Restrictions

Para-virtualized device drivers can be installed after successfully installing a guest operating system. You will need a functioning host and guest before you can install these drivers.



Note

GRUB cannot presently access para-virtualized block devices. Therefore, a guest can not be booted from a device that uses the para-virtualized block device drivers: specifically, the disk that contains the Master Boot Record (MBR), a disk containing a boot loader (**GRUB**), or a disk that contains the kernel **initrd** images. Any disk which contains the **/boot** directory or partition cannot use the para-virtualized block device drivers.

Red Hat Enterprise Linux 3 kernel variant architecture dependencies

For Red Hat Enterprise Linux 3 based guest operating systems you must use the processor specific kernel and para-virtualized driver RPMs as seen in the tables below. If you fail to install the matching para-virtualized driver package, loading of the **xen-pci-platform** module will fail.

The table below shows which host kernel is required to run a Red Hat Enterprise Linux 3 guest, if the guest was compiled for an Intel processor.

Table 12.2. Required host kernel architecture for guests using para-virtualized drivers on Red Hat Enterprise Linux 3 for Intel processors

Guest kernel type	Required host kernel type
ia32e (UP and SMP)	x86_64
i686	i686
i686-SMP	i686
i686-HUGEMEM	i686

The table below shows which host kernel is required to run a Red Hat Enterprise Linux 3 guest, if the guest was compiled for an AMD processor.

Table 12.3. Required host kernel architectures for guests using para-virtualized drivers on Red Hat Enterprise Linux 3 for AMD processors

Guest kernel type	Required host kernel type
athlon	i686
athlon-SMP	i686
x86_64	x86_64
x86_64-SMP	x86_64

12.3. Installing the Para-virtualized Drivers

The following sections describe how to install and configure your fully virtualized guests to run on Red Hat Enterprise Linux 5.1 or above with para-virtualized drivers.



Important

Para-virtualized drivers are only supported on certain hardware and version combinations. Verify your hardware and operating system requirements are met before proceeding to install para-virtualized drivers.



Note

If you are installing a new guest system, in order to gain maximal benefit from the para-virtualized block device drivers, you should create the guest with at least two disks.

Using the para-virtualized drivers for the disk that contains the **MBR** and the boot loader (**GRUB**), and for the **/boot** partition. This partition can be very small, as it only needs to have enough capacity to hold the **/boot** partition.

Use the second disk and any additional disks for all other partitions (for example, **/**, **/usr**) or logical volumes.

Using this installation method, when the para-virtualized block device drivers are later installed after completing the install of the guest, only booting the guest and accessing the **/boot** partition will use the virtualized block device drivers.

12.3.1. Common installation steps

The list below covers the high level steps common across all guest operating system versions.

1. Copy the RPMs for your hardware architecture to a suitable location in your guest operating system. Your home directory is sufficient. If you do not know which RPM you require verify against the table at [Section 12.2, “Para-virtualization Restrictions and Support”](#).
2. Use the **rpm** command or the **yum** command to install the packages. The **rpm** utility will install the following four new kernel modules into **/lib/modules/[%kversion] [%kvariant]/extra/xenpv/%release**:
 - » the PCI infrastructure module, **xen_platform_pci.ko**,
 - » the ballooning module, **xen_balloon.ko**,
 - » the virtual block device module, **xen_vbd.ko**,
 - » and the virtual network device module, **xen_vnif.ko**.
3. If the guest operating does not support automatically loading the para-virtualized drivers (for example, Red Hat Enterprise Linux 3) perform the required post-install steps to copy the drivers into the operating system specific locations.
4. Shut down your guest operating system.
5. Reconfigure the guest operating system's configuration file on the host to use the installed para-virtualized drivers.
6. Remove the “type=ioemu” entry for the network device.
7. Add any additional disk partitions, volumes or LUNs to the guest so that they can be accessed via the para-virtualized (**xen-vbd**) disk driver.
8. For each physical device, LUN, partition or volume you want to use the para-virtualized drivers you must edit the disk entry for that device in the libvirt configuration file.
9. A typical disk entry resembles the following:

```
<disk type='file' device='disk'>
  <driver name='file' />
  <source file='/dev/hda6' />
  <target dev='hda' />
</disk>
```

Modify each disk entry, as desired, to use the para-virtualized by changing the driver elements as shown below.

```
<disk type='file' device='disk'>
  <driver name='tap' type='aio' />
  <source file='/dev/hda6' />
  <target dev='xvda' />
</disk>
```

10. Add any additional storage entities you want to use for the para-virtualized block device driver.
11. Restart your guest:

```
# xm start YourGuestName
```

Where *YourGuestName* is the name of the configuration file or the guest operating system's name as defined in its configuration file in the **name = "os_name"** parameter.

12. Reconfigure the guest network.

12.3.2. Installation and Configuration of Para-virtualized Drivers on Red Hat Enterprise Linux 3

This section contains detailed instructions for the para-virtualized drivers in a Red Hat Enterprise 3 guest operating system.



Note

These packages do not support booting from a para-virtualized disk. Booting the guest operating system kernel still requires the use of the emulated IDE driver, while any other (non-system) user-space applications and data can use the para-virtualized block device drivers.

Driver Installation

The list below covers the steps to install a Red Hat Enterprise Linux 3 guest with para-virtualized drivers.

1. Install the latest kernel version. The para-virtualized drivers require at least Red Hat Enterprise Linux 3.9 kernel version **kernel-2.4.21-60.EL** for all the required headers.
2. Copy the **kmod-xenpv** rpm for your hardware architecture and kernel variant to your guest operating system.
3. Use the **rpm** utility to install the RPM packages. Ensure you have correctly identified which package you need for your guest operating system variant and architecture.

```
[root@rhel3]# rpm -ivh kmod-xenpv*
```

4. Use the commands below load the para-virtualized driver modules. *%kvariant* is the kernel variant the para-virtualized drivers have been build against and *%release* corresponds to the release version of the para-virtualized drivers.

```
[root@rhel3]# mkdir -p /lib/modules/'uname -r'/extra/xenpv
[root@rhel3]# cp -R /lib/modules/2.4.21-52.EL[%kvariant]/extra/xenpv/%release
\ /lib/modules/'uname -r'/extra/xenpv
[root@rhel3]# depmod -ae
[root@rhel3]# modprobe xen-vbd
[root@rhel3]# modprobe xen-vnif
```

Note

Warnings will be generated by **insmod** when installing the binary driver modules due to Red Hat Enterprise Linux 3 having **MODVERSIONS** enabled. These warnings can be ignored.

5. Verify **/etc/modules.conf** and make sure you have an alias for **eth0** like the one below. If you are planning to configure multiple interfaces add an additional line for each interface.

```
alias eth0 xen-vnif
```

Edit **/etc/rc.local** and add the line:

```
insmod /lib/modules/'uname -r'/extra/xenpv/%release/xen-vbd.o
```

Note

Substitute “%release” with the actual release version (for example 0.1-5.el) for the para-virtualized drivers. If you update the para-virtualized driver RPM package make sure you update the release version to the appropriate version.

6. Shutdown the virtual machine (use “**#shutdown -h now**” inside the guest).
7. Edit the guest configuration file in **/etc/xen/YourGuestName** with a text editor, performing the following changes:
 - Remove the “**type=ioemu**” entry from the “**vif=**” entry.
 - Add any additional disk partitions, volumes or LUNs to the guest so that they can be accessed via the para-virtualized (**xen-vbd**) disk driver.
 - For each physical device, LUN, partition or volume you want to use the para-virtualized drivers you must edit the disk entry for that device in the libvirt configuration file.
 - A typical disk entry resembles the following:

```
<disk type='file' device='disk'>
  <driver name='file' />
  <source file='/dev/hda6' />
  <target dev='hda' />
</disk>
```

Modify each disk entry, as desired, to use the para-virtualized by changing the driver elements as shown below.

```
<disk type='file' device='disk'>
  <driver name='tap' type='aio' />
  <source file='/dev/hda6' />
  <target dev='xvda' />
</disk>
```

- » Once complete, save the modified configuration file and restart the guest.

8. Boot the virtual machine:

```
# xm start YourGuestName
```

Where *YourGuestName* is the name of the configuration file or the guest operating system's name as defined in its configuration file in the **name = "os_name"** parameter.



Warning

The para-virtualized drivers are not automatically added and loaded to the system because **weak-modules** and **modversions** support is not provided in Red Hat Enterprise Linux 3. To insert the module execute the command below.

```
insmod xen_vbd.ko
```

Red Hat Enterprise Linux 3 requires the manual creation of the special files for the block devices which use **xen-vbd**. The steps below will cover how to create and register para-virtualized block devices.

Use the following script to create the special files after the para-virtualized block device driver is loaded.

```
#!/bin/sh
module="xvd"
mode="664"
major=`awk "\$2==\"$module\" {print \$1}" /proc/devices`
# < mknod for as many or few partitions on xvd disk attached to FV guest >
# change/add xvda to xvdb, xvbd, etc. for 2nd, 3rd, etc., disk added in
# in xen config file, respectively.
mknod /dev/xvdb b $major 16
mknod /dev/xvdb1 b $major 17
mknod /dev/xvdb2 b $major 18
chgrp disk /dev/xvd*
chmod 0660 /dev/xvd*
```

For each additional virtual disk, increment the minor number by 16. In the example below an additional device, minor number 16, is created.

```
# mknod /dev/xvdc b $major 16
# mknod /dev/xvdc1 b $major 17
```

This would make the next device 32 which can be created by:

```
# mknod /dev/xvdd b $major 32
# mknod /dev/xvdd1 b $major 33
```

Now you should verify the partitions which you have created are available.

```
[root@rhel3]# cat /proc/partitions
major minor #blocks name

 3      0    10485760  hda
 3      1     104391  hda1
 3      2   10377990  hda2
202    16      64000  xvdb
202    17     32000  xvdb1
202    18     32000  xvdb2
253     0    8257536  dm-0
253     1   2031616  dm-1
```

In the above output, you can observe that the partitioned device “**xvdb**” is available to the system.

The procedure below adds the new device to the guest and makes it persistent after rebooting. All these commands are executed on the guest.

1. Create directories to mount the block device image in.

```
[root@rhel3]# mkdir /mnt/pvdisk_p1
[root@rhel3]# mkdir /mnt/pvdisk_p2
```

2. Mount the devices to the new folders.

```
[root@rhel3]# mount /dev/xvdb1 /mnt/pvdisk_p1
[root@rhel3]# mount /dev/xvdb2 /mnt/pvdisk_p2
```

3. Verify the devices are mounted correctly.

```
[root@rhel3]# df /mnt/pvdisk_p1
Filesystem      1K-blocks   Used   Available Use% Mounted on
/dev/xvdb1        32000     15     31985   1% /mnt/pvdisk_p1
```

4. Update the **/etc/fstab** file inside the guest to mount the devices during the boot sequence.
Add the following lines:

```
/dev/xvdb1  /mnt/pvdisk_p1  ext3  defaults      1 2
/dev/xvdb2  /mnt/pvdisk_p2  ext3  defaults      1 2
```

Note

Using a Red Hat Enterprise Linux 5.1 host (**dom0**), the “**noapic**” parameter should be added to the kernel boot line in your virtual guest’s **/boot/grub/grub.conf** entry as seen below. Keep in mind your architecture and kernel version may be different.

```
kernel /vmlinuz-2.6.9-67.EL ro root=/dev/VolGroup00/rhel4_x86_64 rhgb
noapic
```

A Red Hat Enterprise Linux 5.2 dom0 will not need this kernel parameter for the guest.



Important

The Itanium (ia64) binary RPM packages and builds are not presently available.

12.3.3. Installation and Configuration of Para-virtualized Drivers on Red Hat Enterprise Linux 4

This section contains detailed instructions for the para-virtualized drivers in a Red Hat Enterprise 4 guest operating system.



Note

These packages do not support booting from a para-virtualized disk. Booting the guest operating system kernel still requires the use of the emulated IDE driver, while any other (non-system) user-space applications and data can use the para-virtualized block device drivers.

Driver Installation

The list below covers the steps to install a Red Hat Enterprise Linux 4 guest with para-virtualized drivers.

1. Copy the **kmod-xenpv**, **modules-init-tools** and **modversions** RPMs for your hardware architecture and kernel variant to your guest operating system.
2. Use the **rpm** utility to install the RPM packages. Make sure you have correctly identified which package you need for your guest operating system variant and architecture. An updated module-init-tools is required for this package, it is available with the Red Hat Enterprise Linux 4-6-z kernel or newer.

```
[root@rhel4]# rpm -ivh modversions
[root@rhel4]# rpm -Uvh module-init-tools
[root@rhel4]# rpm -ivh kmod-xenpv*
```



Note

There are different packages for UP, SMP, Hugemem and architectures so make sure you have the right RPMs for your kernel.

3. Execute **cat /etc/modprobe.conf** to verify you have an alias for **eth0** like the one below. If you are planning to configure multiple interfaces add an additional line for each interface. If it does not look like the entry below change it.

```
alias eth0 xen-vnif
```

4. Shutdown the virtual machine (use “**#shutdown -h now**” inside the guest).
5. Edit the guest configuration file in **/etc/xen/YourGuestsName** in the following ways:
 - Remove the “**type=ioemu**” entry from the “**vif=**” entry.

- Add any additional disk partitions, volumes or LUNs to the guest so that they can be accessed via the para-virtualized (**xen-vbd**) disk driver.
- For each additional physical device, LUN, partition or volume add an entry similar to the one shown below to the “**disk=**” section in the guest configuration file. The original “**disk=**” entry might also look like the entry below.

```
disk = [ "file:/var/lib/libvirt/images/rhel4_64_fv.dsk,hda,w" ]
```

- Once you have added additional physical devices, LUNs, partitions or volumes; the para-virtualized driver entry in your XML configuration file should resemble the entry shown below.

```
disk = [ "file:/var/lib/libvirt/images/rhel3_64_fv.dsk,hda,w",
          "tap:aio:/var/lib/libvirt/images/UserStorage.dsk,xvda,w" ]
```

Note

Use “**tap:aio**” for the para-virtualized device if a file-based image is used.

6. Boot the virtual machine using the **virsh** command:

```
# virsh start YourGuestName
```

On the first reboot of the virtual guest, **kudzu** will ask you to “*Keep or Delete the Realtek Network device*” and “*Configure the xen-bridge device*”. You should configure the **xen-bridge** and delete the Realtek network device.

Note

Using a Red Hat Enterprise Linux 5.1 host (**dom0**), the “**noapic**” parameter should be added to the kernel boot line in your virtual guest’s **/boot/grub/grub.conf** entry as seen below. Keep in mind your architecture and kernel version may be different.

```
kernel /vmlinuz-2.6.9-67.EL ro root=/dev/VolGroup00/rhel4_x86_64 rhgb
noapic
```

A Red Hat Enterprise Linux 5.2 dom0 will not need this kernel parameter for the guest.

Now, verify the partitions which you have created are available.

```
[root@rhel4]# cat /proc/partitions
major    minor   #blocks   name
      3        0     10485760   hda
      3        1       104391   hda1
      3        2     10377990   hda2
     202        0       64000   xvdb
     202        1       32000   xvdb1
     202        2       32000   xvdb2
     253        0     8257536   dm-0
     253        1     2031616   dm-1
```

In the above output, you can see the partitioned device “**xvdb**” is available to the system.

The procedure below adds the new device to the guest and makes it persistent after rebooting. All these commands are executed on the guest.

1. Create directories to mount the block device image in.

```
[root@rhel4]# mkdir /mnt/pvdisk_p1
[root@rhel4]# mkdir /mnt/pvdisk_p2
```

2. Mount the devices to the new folders.

```
[root@rhel4]# mount /dev/xvdb1 /mnt/pvdisk_p1
[root@rhel4]# mount /dev/xvdb2 /mnt/pvdisk_p2
```

3. Verify the devices are mounted correctly.

```
[root@rhel4]# df /mnt/pvdisk_p1
Filesystem      1K-blocks   Used   Available  Use%   Mounted on
/dev/xvdb1        32000     15     31985    1%   /mnt/pvdisk_p1
```

4. Update the **/etc/fstab** file inside the guest to mount the devices during the boot sequence.

Add the following lines:

```
/dev/xvdb1  /mnt/pvdisk_p1  ext3  defaults      1 2
/dev/xvdb2  /mnt/pvdisk_p2  ext3  defaults      1 2
```

Note

This package is not supported for Red Hat Enterprise Linux 4-GA through Red Hat Enterprise Linux 4 update 2 systems and kernels.



Important

IA64 binary RPM packages and builds are not presently available.

Note

The **xen-vbd** driver may not automatically load. Execute the following command on the guest, substituting **%release** with the correct release version for the para-virtualized drivers.

```
# insmod /lib/modules/'uname -r'/weak-
updates/xenpv/%release/xen_vbd.ko
```

12.3.4. Xen Para-virtualized Drivers on Red Hat Enterprise Linux 5

This section contains detailed instructions for the para-virtualized drivers in a Red Hat Enterprise Linux 5 guest operating system.



Note

These packages do not support booting from a para-virtualized disk. Booting the guest operating system kernel still requires the use of the emulated IDE driver, while any other (non-system) user-space applications and data can use the para-virtualized block device drivers.

The procedure below covers the steps to enable the para-virtualized drivers for a Red Hat Enterprise Linux 5 guest.

Procedure 12.1. Enable para-virtualized drivers for a Red Hat Enterprise Linux Guest

1. Shutdown the virtual machine (use “**#shutdown -h now**” inside the guest).
2. Edit the guest configuration file in **/etc/xen/<Your GuestsName>** in the following ways:
 - a. Remove the “**type=ioemu**” entry from the “**vif=**” entry.
 - b. Add any additional disk partitions, volumes or LUNs to the guest so that they can be accessed via the para-virtualized (**xen-vbd**) disk driver.
 - c. For each additional physical device, LUN, partition or volume add an entry similar to the one shown below to the “**disk=**” section in the guest configuration file. The original “**disk=**” entry might also look like the entry below.

```
disk = [ "file:/var/lib/libvirt/images/rhel4_64_fv.dsk,hda,w" ]
```

- d. Once you have added additional physical devices, LUNs, partitions or volumes; the para-virtualized driver entry in your XML configuration file should resemble the entry shown below.

```
disk = [ "file:/var/lib/libvirt/images/rhel3_64_fv.dsk,hda,w",
          "tap:aio:/var/lib/libvirt/images/UserStorage.dsk,xvda,w" ]
```



Note

Use “**tap:aio**” for the para-virtualized device if a file-based image is used.

3. Boot the virtual machine using the **virsh** command:

```
# virsh start YourGuestName
```

To verify the network interface has come up after installing the para-virtualized drivers issue the following command on the guest. It should display the interface information including an assigned IP address

```
[root@rhel5]# ifconfig eth0
```

Now, verify the partitions which you have created are available.

```
[root@rhel5]# cat /proc/partitions
major minor #blocks name
 3        0    10485760  hda
```

```

3      1    104391  hda1
3      2    10377990  hda2
202    0     64000  xvdb
202    1     32000  xvdb1
202    2     32000  xvdb2
253    0    8257536  dm-0
253    1    2031616  dm-1

```

In the above output, you can see the partitioned device “**xvdb**” is available to the system.

The procedure below adds the new device to the guest and makes it persistent after rebooting. All these commands are executed on the guest.

1. Create directories to mount the block device image in.

```
[root@rhel5]# mkdir /mnt/pvdisk_p1
[root@rhel5]# mkdir /mnt/pvdisk_p2
```

2. Mount the devices to the new folders.

```
[root@rhel5]# mount /dev/xvdb1 /mnt/pvdisk_p1
[root@rhel5]# mount /dev/xvdb2 /mnt/pvdisk_p2
```

3. Verify the devices are mounted correctly.

```
[root@rhel5]# df /mnt/pvdisk_p1
Filesystem           1K-blocks      Used   Available  Use%   Mounted on
/dev/xvdb1            32000        15     31985    1%   /mnt/pvdisk_p1
```

4. Update the **/etc/fstab** file inside the guest to mount the devices during the boot sequence. Add the following lines:

```
/dev/xvdb1  /mnt/pvdisk_p1  ext3  defaults  1 2
/dev/xvdb2  /mnt/pvdisk_p2  ext3  defaults  1 2
```

Note

Using a Red Hat Enterprise Linux 5.1 host (**dom0**), the “**noapic**” parameter should be added to the kernel boot line in your virtual guest’s **/boot/grub/grub.conf** entry as seen below. Keep in mind your architecture and kernel version may be different.

```
kernel /vmlinuz-2.6.9-67.EL ro root=/dev/VolGroup00/rhel4_x86_64 rhgb
noapic
```

A Red Hat Enterprise Linux 5.2 dom0 will not need this kernel parameter for the guest.

Hiding fake interfaces

Sometimes, activating the para-virtualized drivers does not delete the old virtualized network interfaces. To remove these interfaces from guests use the following procedure.

1. Add the following lines to the **/etc/modprobe.d/blacklist** file. Blacklist **8139cp** and **8139too** for the RealTek 8139 and **e1000** for the virtualized Intel e1000 NIC.

```
8139cp
8139too
e1000
```

2. Remove the old network scripts from the **/etc/sysconfig/network-scripts** directory.
3. Reboot the guest. The default network interface should now use the para-virtualized drivers.

12.3.5. Xen Para-virtualized Drivers on Red Hat Enterprise Linux 6

This section describes the use of para-virtualized drivers in a Red Hat Enterprise Linux 6 guest.

The para-virtualized drivers are enabled by default for a Red Hat Enterprise Linux 6 guest. The guest will automatically unplug any emulated devices that are presented to it, and will use the para-virtualized drivers instead.

Although the para-virtualized drivers are enabled by default, they can be disabled. Add the following to the guest kernel command line upon boot to disable the para-virtualized drivers:

```
xen_emul_unplug=never
```

12.4. Para-virtualized Network Driver Configuration

Once the para-virtualized network driver is loaded you may need to reconfigure the guest's network interface to reflect the driver and virtual Ethernet card change.

Perform the following steps to reconfigure the network interface inside the guest.

1. In **virt-manager** open the console window for the guest and log in as **root**.
2. On Red Hat Enterprise Linux 4 verify the file **/etc/modprobe.conf** contains the line “**alias eth0 xen-vnif**”.

```
# cat /etc/modprobe.conf
alias eth0 xen-vnif
```

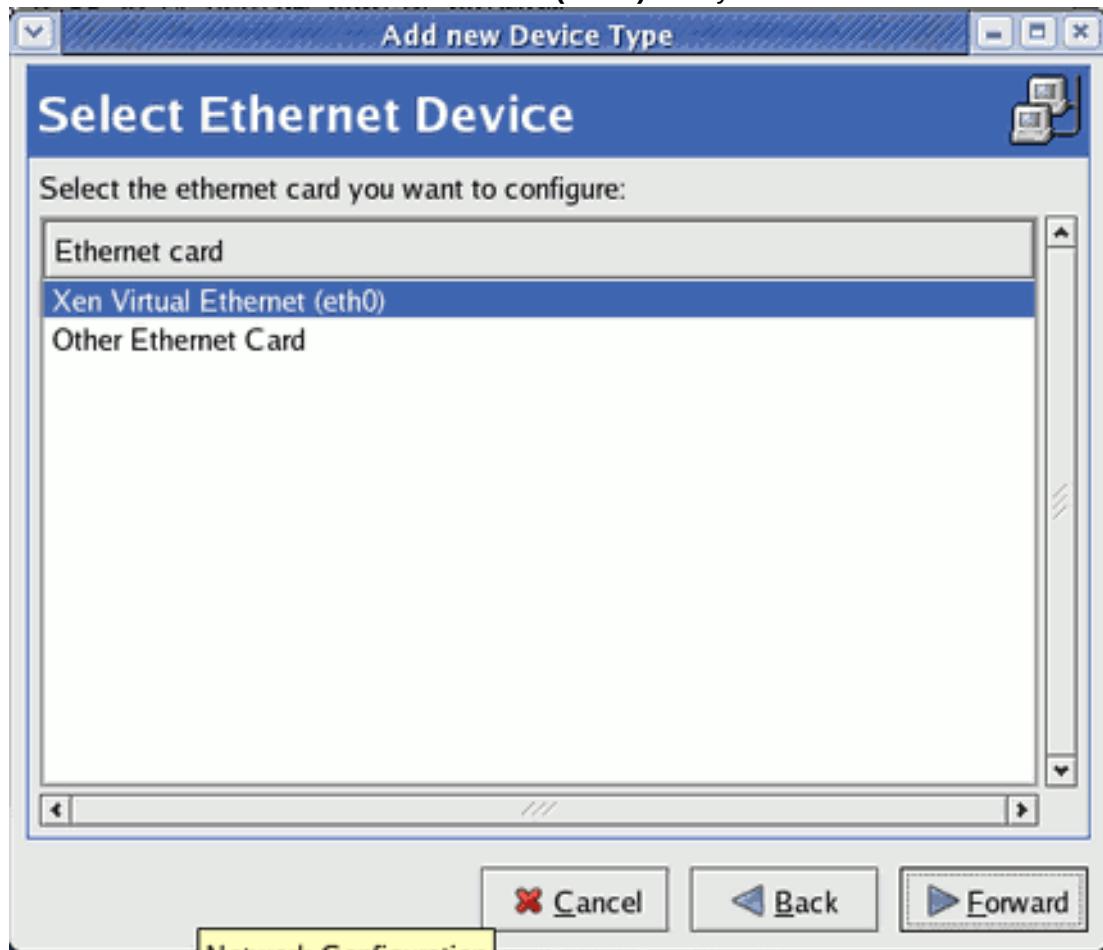
3. To display the present settings for **eth0** execute “**# ifconfig eth0**”. If you receive an error about the device not existing you should load the modules manually as outlined in [Section 37.4, “Manually loading the para-virtualized drivers”](#).

```
ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:00:6A:27:3A
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:630150 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:109336431 (104.2 MiB)  TX bytes:846 (846.0 b)
```

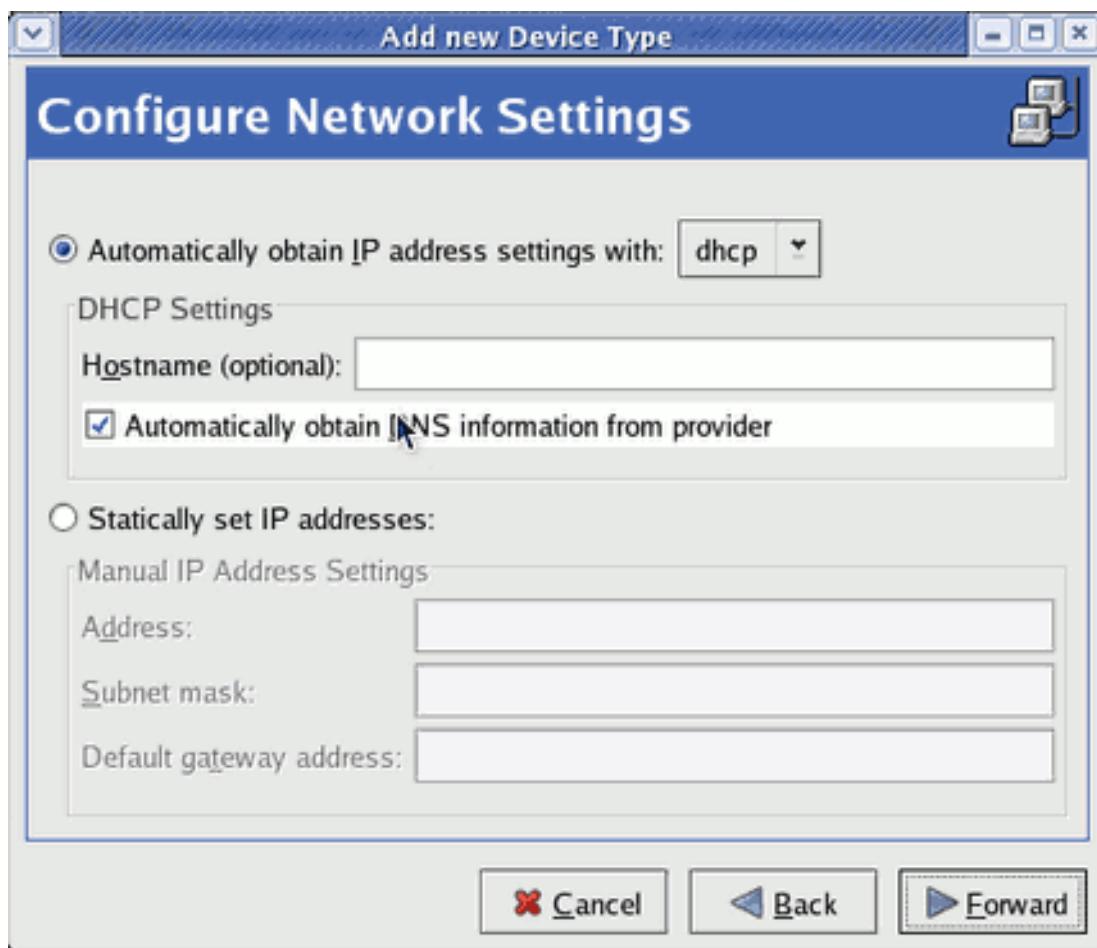
4. Start the network configuration utility(NetworkManager) with the command “**# system-config-network**”. Click on the “**Forward**” button to start the network card configuration.



5. Select the 'Xen Virtual Ethernet Card (eth0)' entry and click 'Forward'.



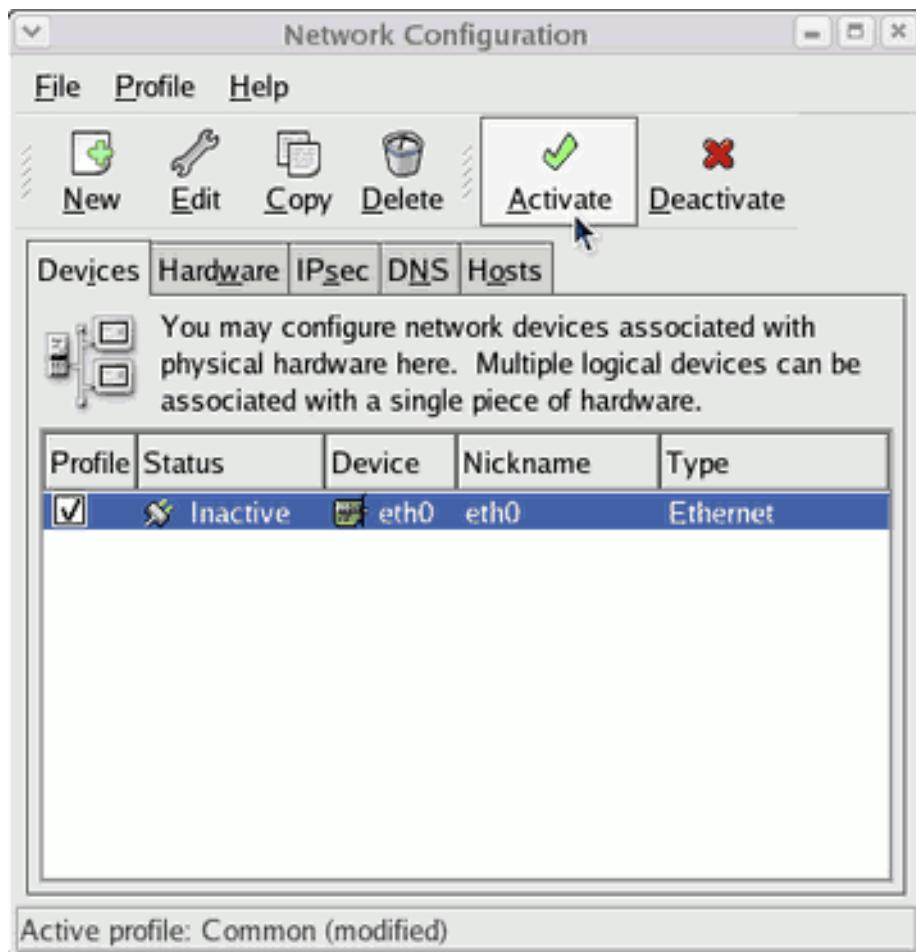
Configure the network settings as required.



6. Complete the configuration by pressing the '**Apply**' button.



7. Press the '**Activate**' button to apply the new settings and restart the network.



8. You should now see the new network interface with an IP address assigned.

```
ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:16:3E:49:E4:E0
          inet addr:192.168.78.180  Bcast:192.168.79.255  Mask:255.255.252.0
                  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                  RX packets:630150 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:501209 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:109336431 (104.2 MiB)  TX bytes:46265452 (44.1 MiB)
```

12.5. Additional Para-virtualized Hardware Configuration

This section will explain how to add additional virtual network or storage to a guest operating system. For more details on configuring network and storage resources on Red Hat Enterprise Linux 5 Virtualization read the document available on [Emerging Technologies, Red Hat.com](#)

12.5.1. Virtualized Network Interfaces

Perform the following steps to configure additional network devices for your guest.

Edit your guest configuration file in **/etc/xen/YourGuestName** replacing **YourGuestName** with the name of your guest.

The original entry may look like the one below.

```
vif = [ "mac=00:16:3e:2e:c5:a9,bridge=xenbr0" ]
```

Add an additional entry to the “**vif=**” section of the configuration file similar to the one seen below.

```
vif = [ "mac=00:16:3e:2e:c5:a9,bridge=xenbr0",
        "mac=00:16:3e:2f:d5:a9,bridge=xenbr0" ]
```

Make sure you generate a unique MAC address for the new interface. You can use the command below.

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

After the guest has been rebooted perform the following step in the guest operating system. Verify the update has been added to your **/etc/modules.conf** in Red Hat Enterprise Linux 3 or **/etc/modprobe.conf** in Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5. Add a new alias for each new interface you added.

```
alias eth1 xen-vnif
```

Now test that each new interface you added make sure it is available inside the guest.

```
# ifconfig eth1
```

The command above should display the properties of **eth1**, repeat the command for **eth2** if you added a third interface, and so on.

Now configure the new network interfaces with **redhat-config-network** on Red Hat Enterprise Linux 3 or **system-config-network** on Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5.

12.5.2. Virtual Storage Devices

Perform the following steps to configure additional virtual storage devices for your guest.

Edit your guest configuration file in **/etc/xen/YourGuestName** replacing **YourGuestName** with the name of your guest. The original entry may look like the one below.

```
disk = [ "file:/var/lib/libvirt/images/rhel5_64_fv.dsk,hda,w" ]
```

Now, add an additional entry for your new physical device, LUN, partition or volume to the “**disk=**” parameter in the configuration file. Storage entities which use the para-virtualized driver resemble the entry below. The “**tap:gio**” parameter instructs the hypervisor to use the para-virtualized driver.

```
disk = [ "file:/var/lib/libvirt/images/rhel5_64_fv.dsk,hda,w",
        "tap:gio:/var/lib/libvirt/images/UserStorage1.dsk,xvda,w" ]
```

If you want to add more entries just add them to the “**disk=**” section as a comma separated list.

Note

You need to increment the letter for the '**xvd**' device, that is for your second storage entity it would be '**xvdb**' instead of '**xvda**'.

```
disk = [ "file:/var/lib/libvirt/images/rhel5_64_fv.dsk,hda,w",
         "tap:aio:/var/lib/libvirt/images/UserStorage1.dsk,xvda,w",
         "tap:aio:/var/lib/libvirt/images/UserStorage2.dsk,xvdb,w" ]
```

Verify the partitions have been created and are available.

```
# cat /proc/partitions
major minor #blocks name
  3      0   10485760 hda
  3      1    104391 hda1
  3      2   10377990 hda2
202     0     64000 xvda
202     1     64000 xvdb
253     0   8257536 dm-0
253     1  2031616 dm-1
```

In the above output you can see the partition or device “**xvdb**” is available to the system.

Mount the new devices and disks to local mount points and update the **/etc/fstab** inside the guest to mount the devices and partitions at boot time.

```
# mkdir /mnt/pvdisk_xvda
# mkdir /mnt/pvdisk_xvdb
# mount /dev/xvda /mnt/pvdisk_xvda
# mount /dev/xvdb /mnt/pvdisk_xvdb
# df /mnt
Filesystem      1K-blocks    Used   Available  Use% Mounted on
/dev/xvda        64000       15    63985      1% /mnt/pvdisk_xvda
/dev/xvdb        64000       15    63985      1% /mnt/pvdisk_xvdb
```

Chapter 13. 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-win* package. The *virtio-win* package supports block (storage) devices and network interface controllers.

As with the KVM module, the *virtio-win* drivers package is only available on hosts running Red Hat Enterprise Linux 5.4 and newer.

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 versions of Red Hat Enterprise Linux:

- » Red Hat Enterprise Linux 4.8 and newer
- » Red Hat Enterprise Linux 5.3 and newer
- » Red Hat Enterprise Linux 6.

Those Red Hat Enterprise Linux versions 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 4 are not removable. This means there are up to 28 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 supported 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)

13.1. Installing the KVM Windows para-virtualized drivers

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

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

- » hosting the installation files on a network accessible to the guest,

- » using a virtualized CD-ROM device of the driver installation disk .iso file, or
- » using a virtualized floppy device to install the drivers during boot time (for Windows guests).

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

1.

Download the drivers

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

If the Red Hat Enterprise Linux Supplementary channel entitlements are not enabled for the system, the download will not be available. Enable the Red Hat Enterprise Linux Supplementary channel to access the *virtio-win* package.

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

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

The drivers are also available on the Red Hat Enterprise Linux Supplementary disc or from Microsoft (windowsservercatalog.com). Note that the Red Hat Enterprise Virtualization Hypervisor and Red Hat Enterprise Linux are created on the same code base so the drivers for the same version (for example, 5.5) are supported for both environments.

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

2.

Install the para-virtualized drivers

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

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

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 13.1, “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.

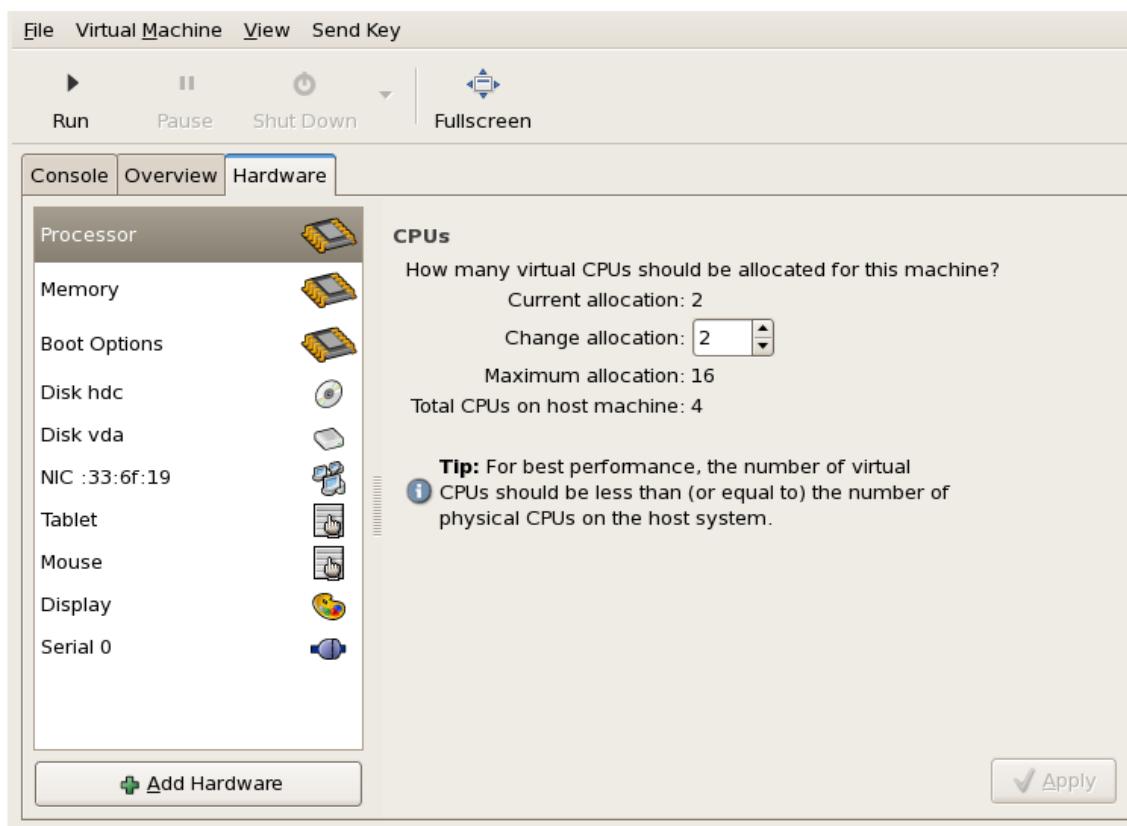
Procedure 13.1. Using virt-manager to mount a CD-ROM image for a Windows guest

1. Open virt-manager and the guest

Open **virt-manager**, select your 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.



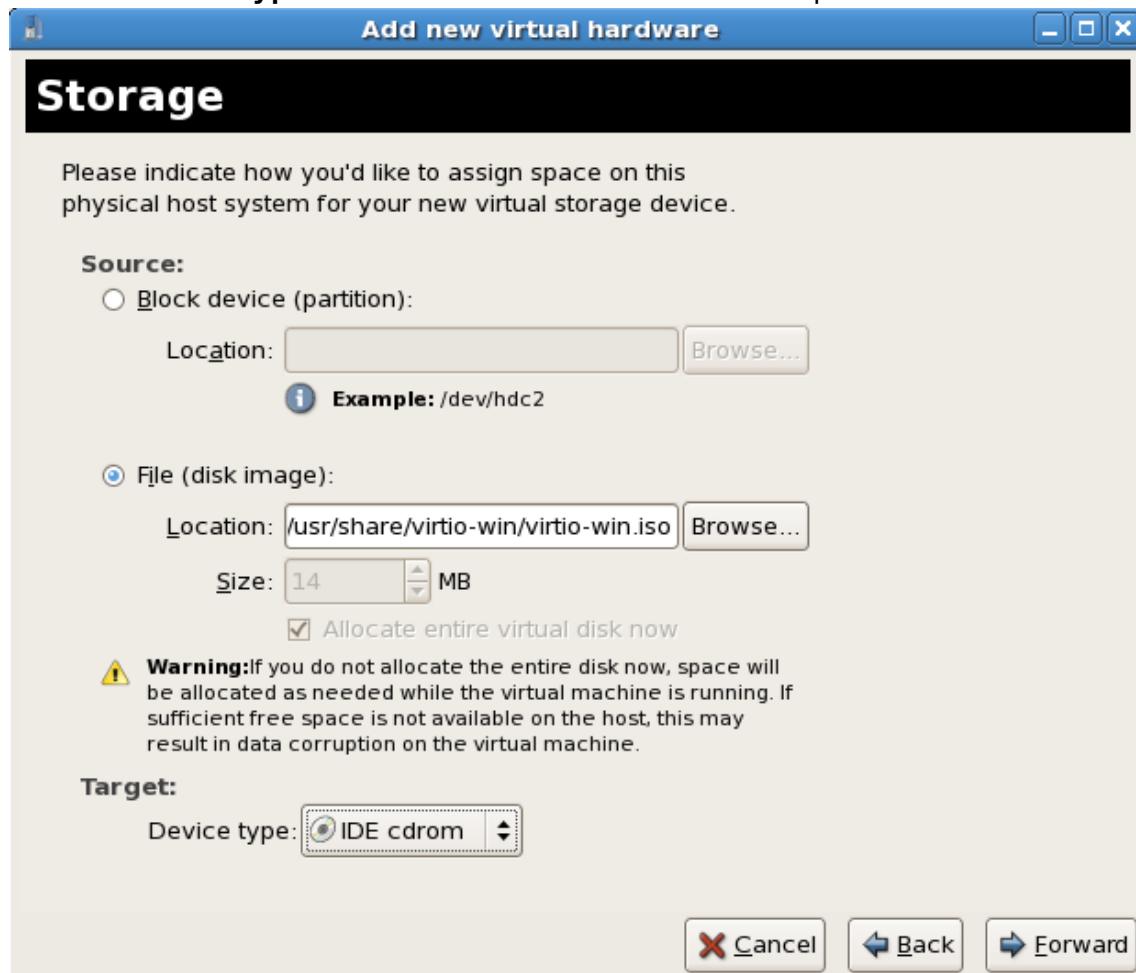
Click the **Forward** button to proceed.

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**.

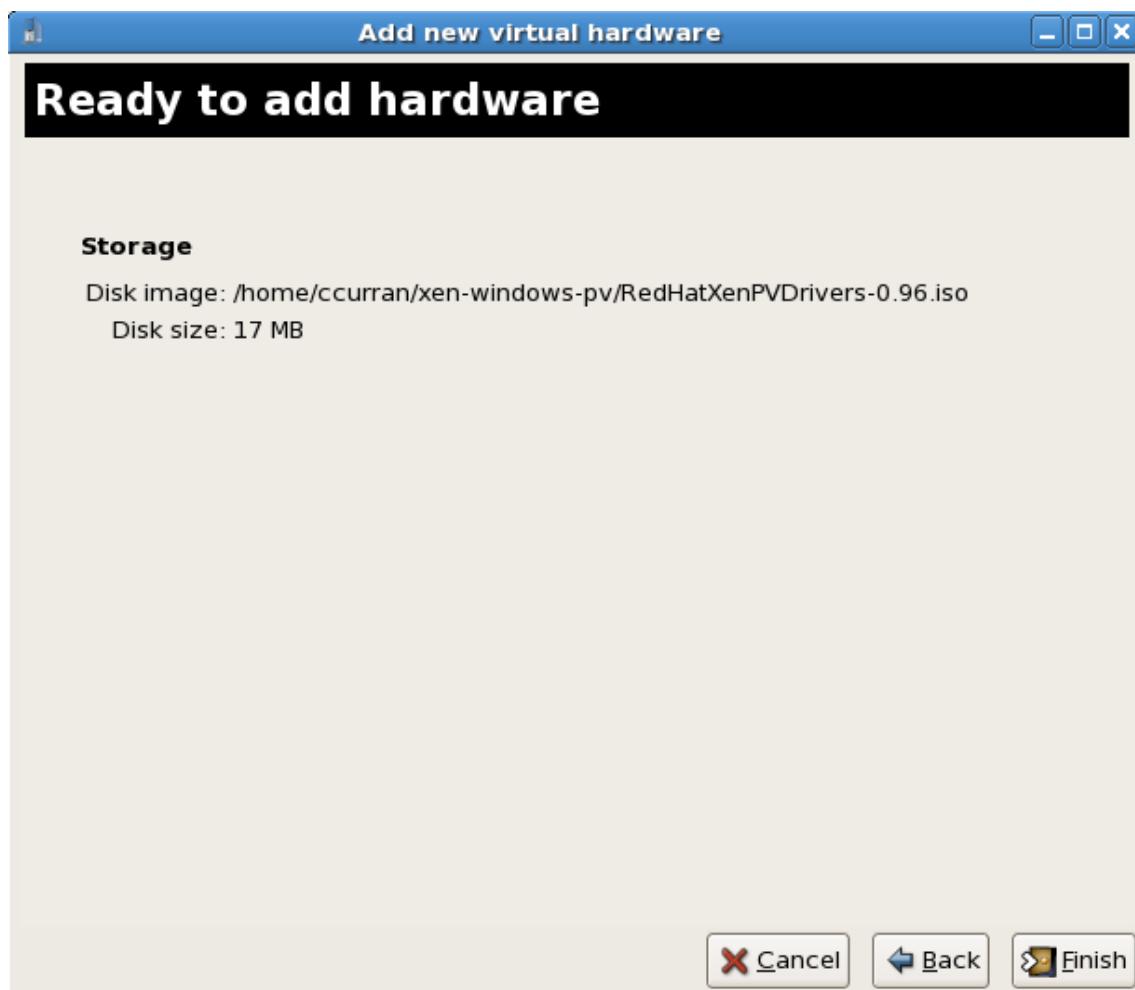
If the drivers are stored on a physical CD-ROM, use the **Normal Disk Partition** option.

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

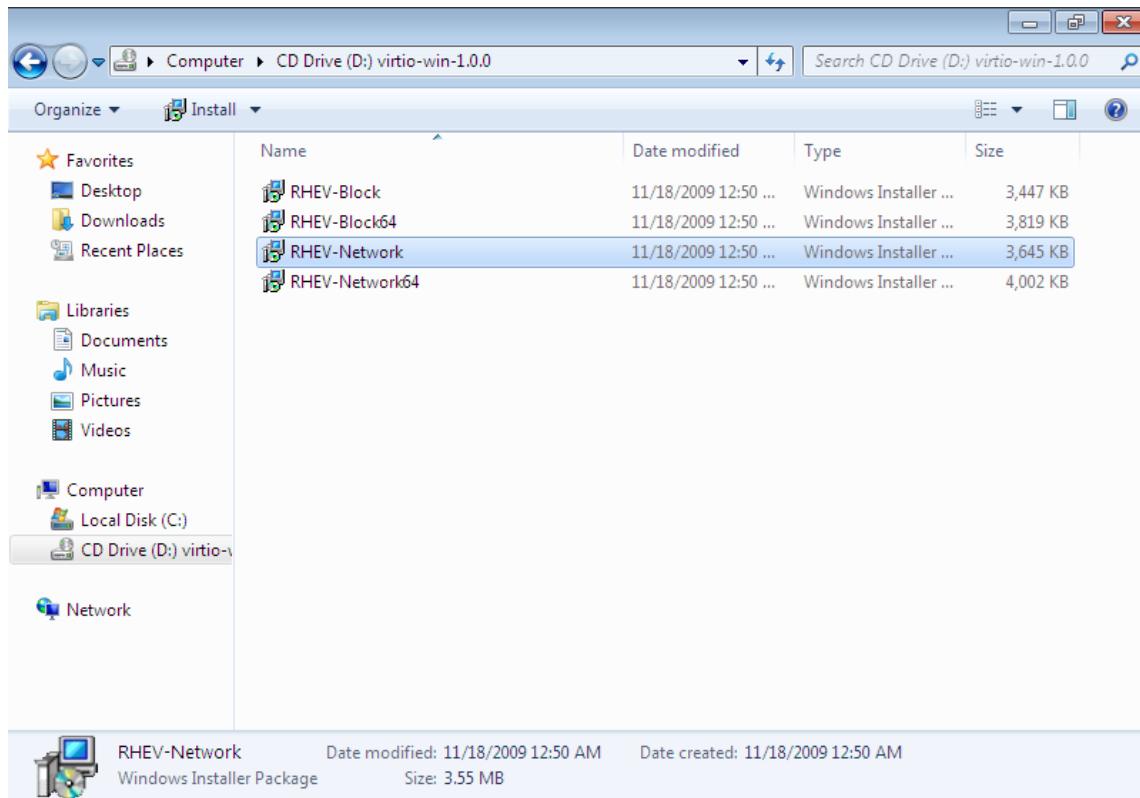
Reboot or start the guest to add the new device. Virtualized IDE devices require a restart before they can be recognized by guests.

Once the CD-ROM with the drivers is attached and the guest has started, proceed with [Procedure 13.2, “Windows installation”](#).

Procedure 13.2. 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:

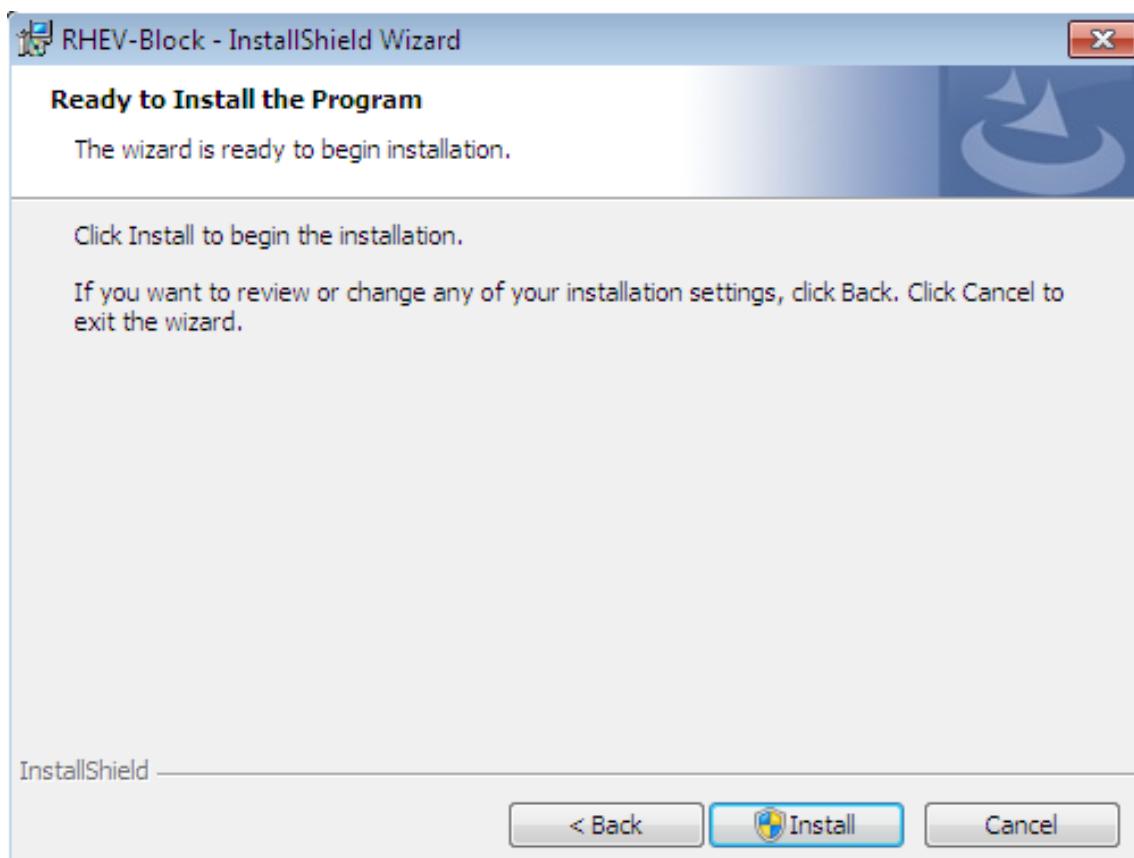
- » the para-virtualized block device driver (**RHEV-Block.msi** for 32-bit guests or **RHEV-Block64.msi** for 64-bit guests),
- » the para-virtualized network device driver (**RHEV-Network.msi** for 32-bit guests or **RHEV-Block64.msi** for 64-bit guests),
- » or both the block and network device drivers.

Double click the installation files to install the drivers.

3. Install the block device driver

a. Start the block device driver installation

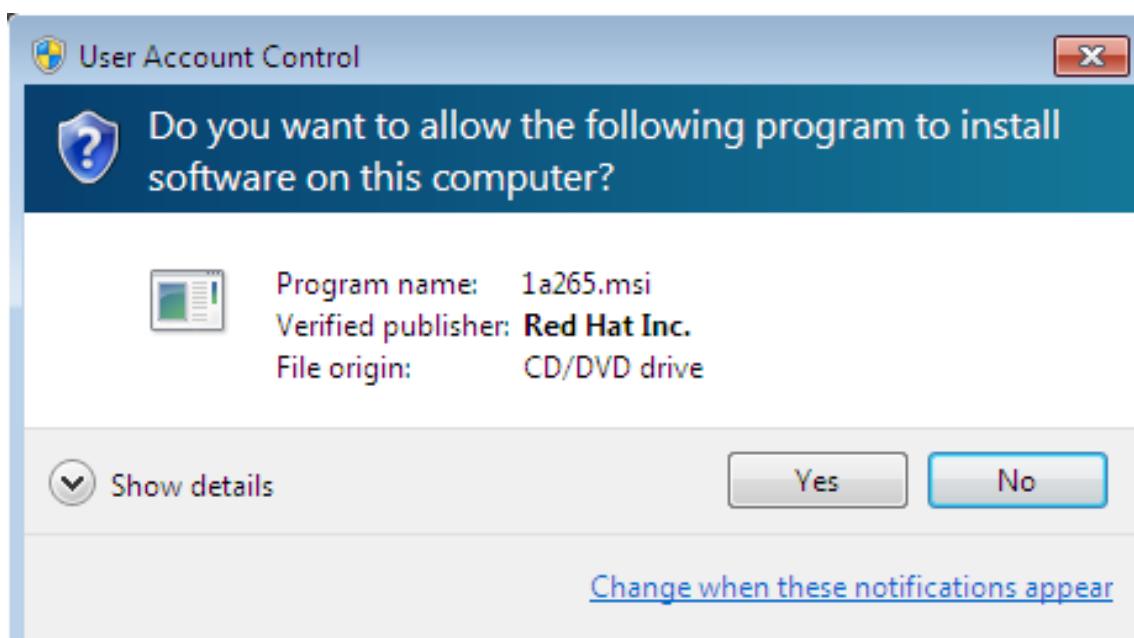
Double click **RHEV-Block.msi** or **RHEV-Block64.msi**.



Press **Next** to continue.

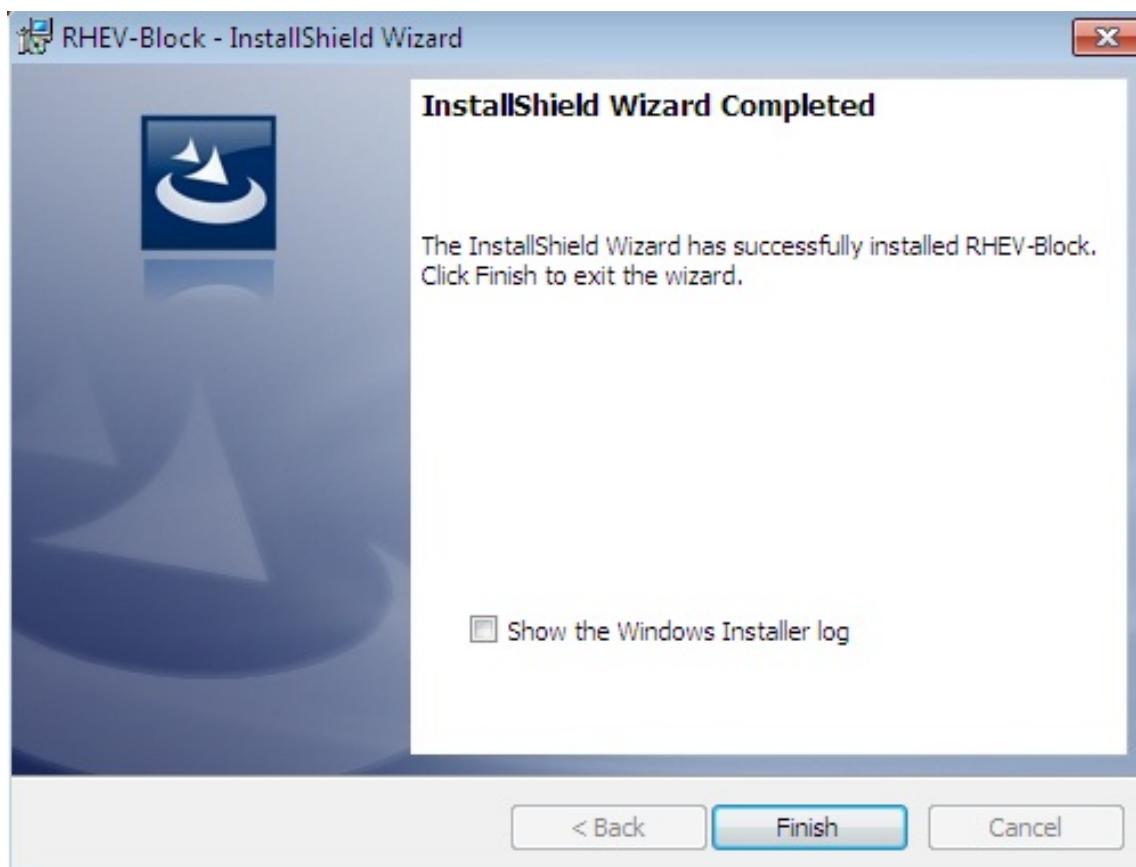
b. Confirm the exception

Windows may prompt for a security exception.



Press **Yes** if it is correct.

c. Finish

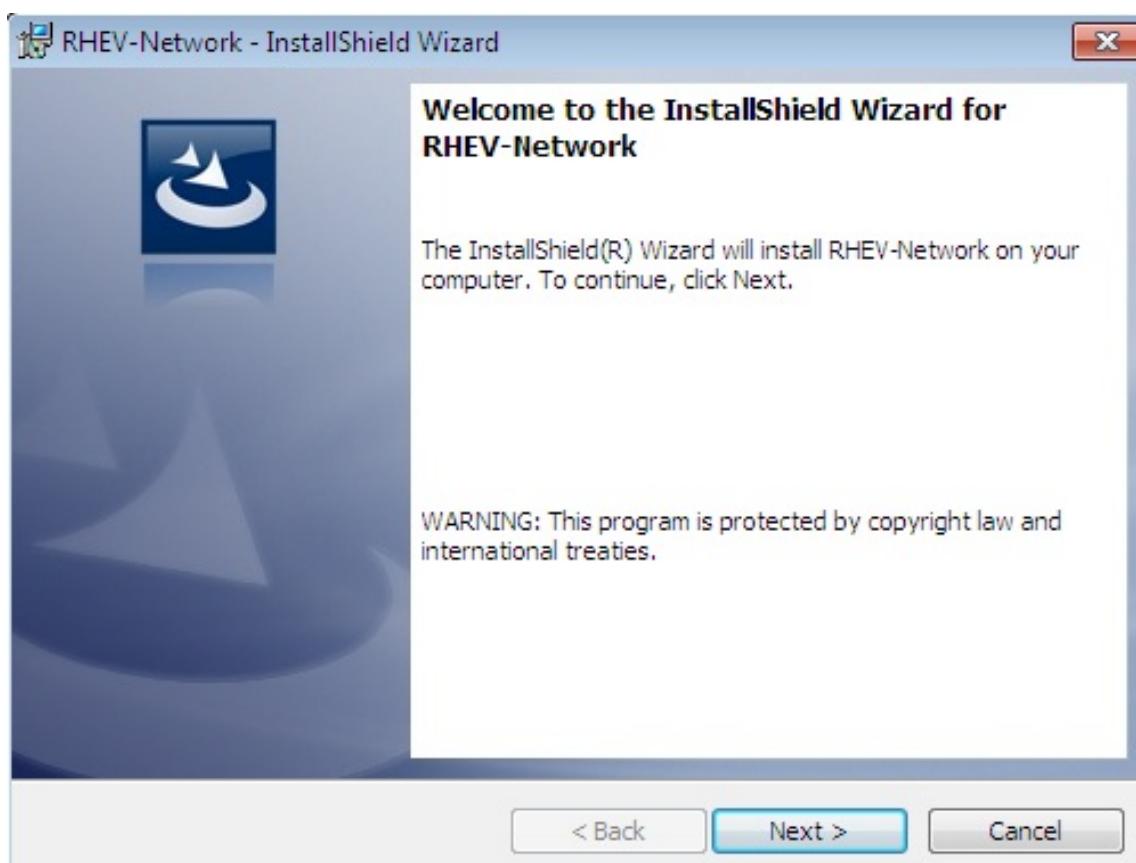


Press **Finish** to complete the installation.

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.

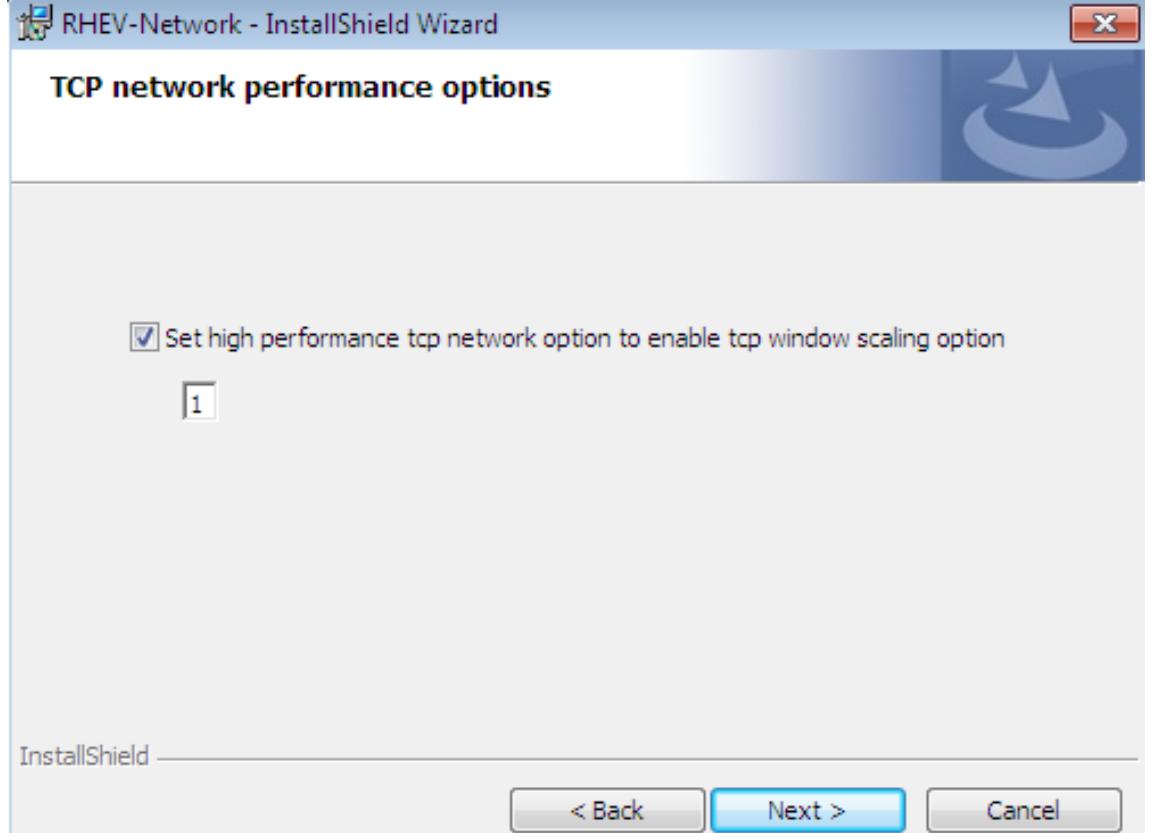
b. Performance setting

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](#). 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 supported by some networking hardware or operating systems.

TCP timestamps are also defined by [IETF RFC 1323](#). 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.



Set high performance tcp network option to enable tcp window scaling option

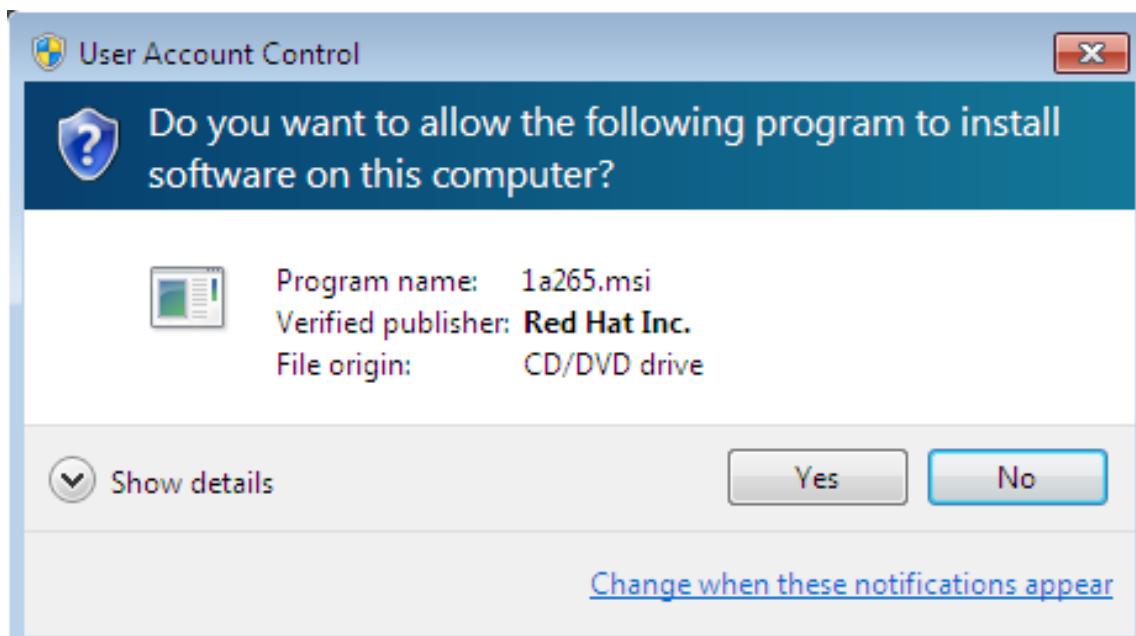
InstallShield

< Back Next > Cancel

Press **Next** to continue.

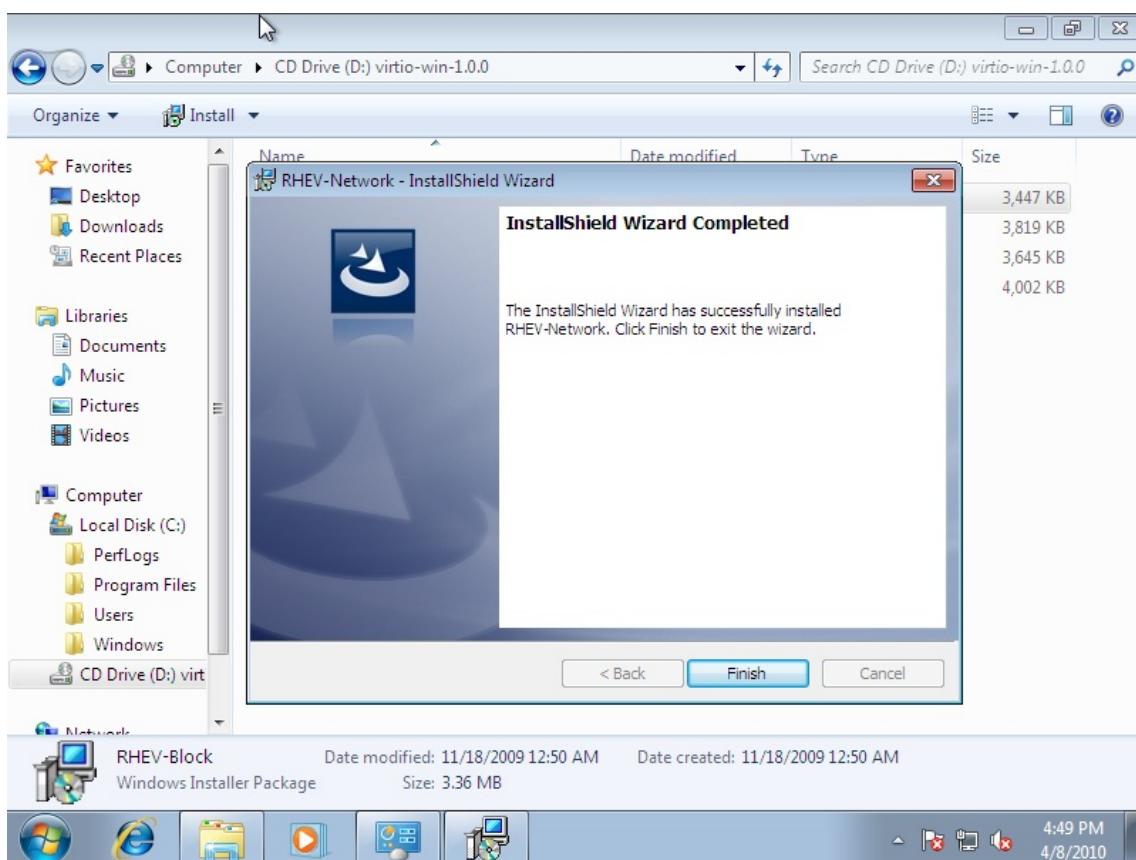
c. Confirm the exception

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 13.3, “Using KVM para-virtualized drivers for existing devices”](#)) or install a new device which uses the para-virtualized drivers ([Section 13.4, “Using KVM para-virtualized drivers for new devices”](#)).

13.2. 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 **viosstor.vfd** as a floppy

»

Windows Server 2003

When windows prompts to press F6 for third party drivers, do so and follow the onscreen instructions.

»

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.

13.3. 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 13.4, “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 guest not using the para-virtualized drivers.

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

13.4. 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.



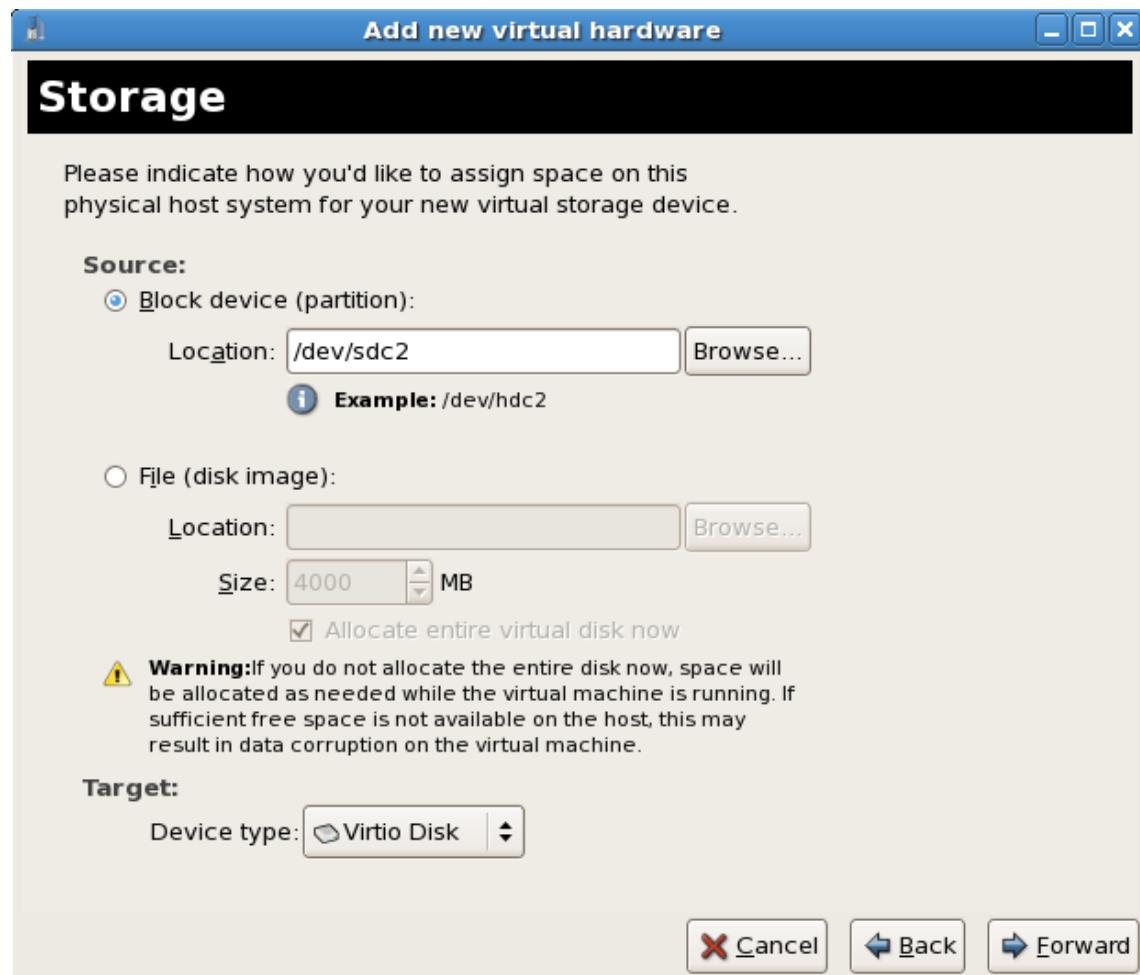
Install the drivers first

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 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.

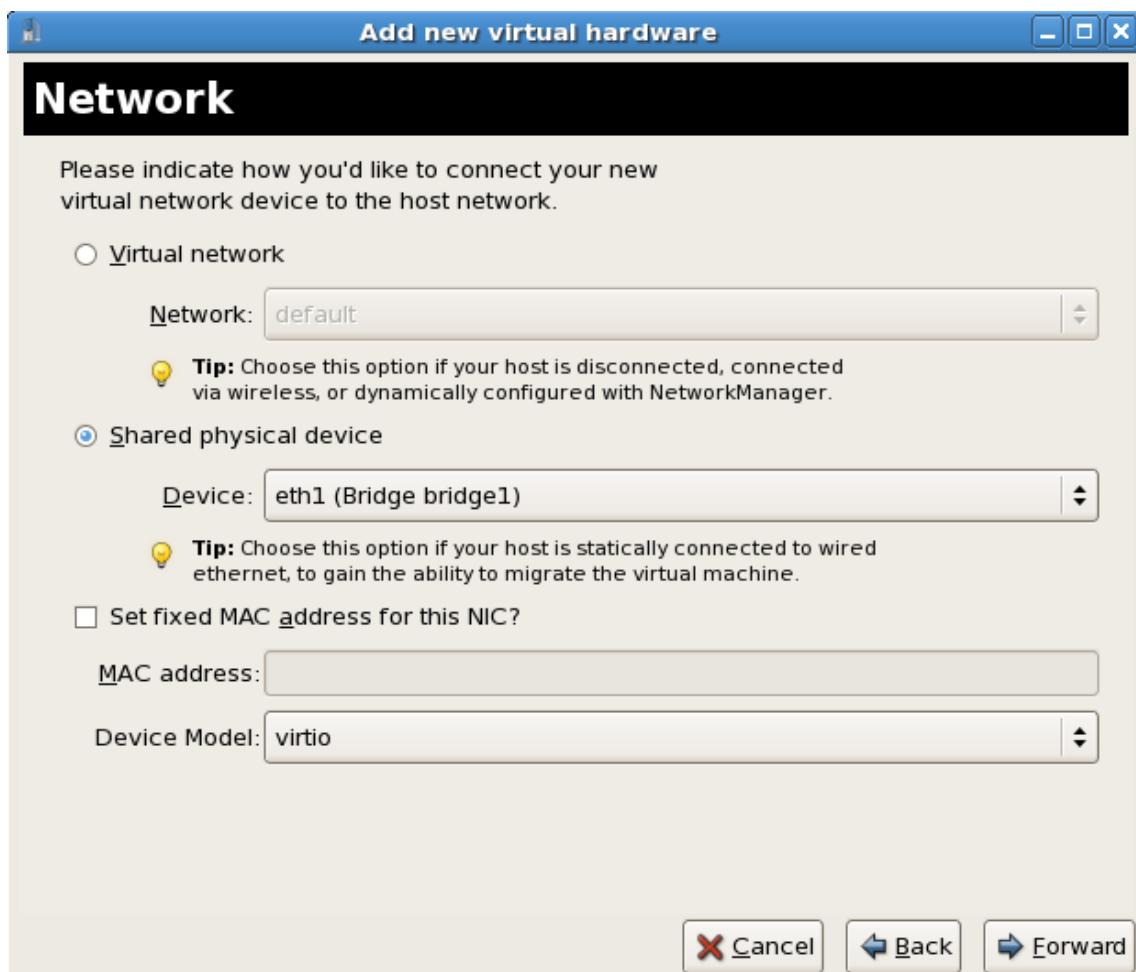
a. New disk devices

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



b. 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.

Chapter 14. Installing Red Hat Enterprise Linux 6 as a Xen guest on Red Hat Enterprise Linux 5

This chapter describes the installation of Red Hat Enterprise Linux 6 as a Xen guest on a Red Hat Enterprise Linux 5 host, both as para-virtualized and fully virtualized.

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

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



Important

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

14.1.1. Using `virt-install`

This section covers creating a Xen para-virtualized Red Hat Enterprise Linux 6 guest on a Red Hat Enterprise Linux 5 host using the **`virt-install`** command. For instructions on **`virt-manager`**, see the procedure in [Section 14.1.2, “Using `virt-manager`”](#).

Guests can be created with the command line **`virt-install`** tool. This method installs Red Hat Enterprise Linux 6 from a remote server hosting the network installation tree. The installation instructions presented in this section are similar to installing from the minimal installation live CD-ROM.

Procedure 14.1. Creating a Xen para-virtualized Red Hat Enterprise Linux 6 guest with `virt-install`

1. Create the guest with the `virt-install` command

In this example, the name of the guest is *rhel6pv-64*, the disk image file is *rhel6pv-64.img* and a local mirror of the Red Hat Enterprise Linux 6 installation tree is http://example.com/installation_tree/RHEL6-x86/.

Replace these values with values for your system and network, and use the following commands to create the guest:

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



Note

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

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

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

2. Install Red Hat Enterprise Linux 6

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

See the *Red Hat Enterprise Linux 6 Installation Guide* for more information on installing Red Hat Enterprise Linux 6.

14.1.2. Using virt-manager

Procedure 14.2. Creating a Xen virtualized Red Hat Enterprise Linux 6 guest with virt-manager

1. Open virt-manager

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

2. Select the hypervisor

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

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

3. Start the new virtual machine wizard

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



Figure 14.1. The virtual machine creation wizard

Click **Forward** to continue.

4. Name the virtual machine

Provide a name for your virtualized guest. The following punctuation and whitespace characters are permitted: '_', '.' and '-' characters.



Figure 14.2. The virtual machine creation wizard

Click **Forward** to continue.

5. Select the virtualization method

Select the appropriate virtualization method. The following example uses **Paravirtualization**.



Figure 14.3. The virtual machine creation wizard

Click **Forward** to continue.

6. Select the installation method and type

Select the appropriate installation method. In this example, use the **Network install tree** method.

Set the **OS Type** and **OS Variant**. In this case, we set **OS Type** to **Linux** and **OS Variant** to **Red Hat Enterprise Linux 6**.



Figure 14.4. The virtual machine creation wizard

Click **Forward** to continue.

7. Locate installation media

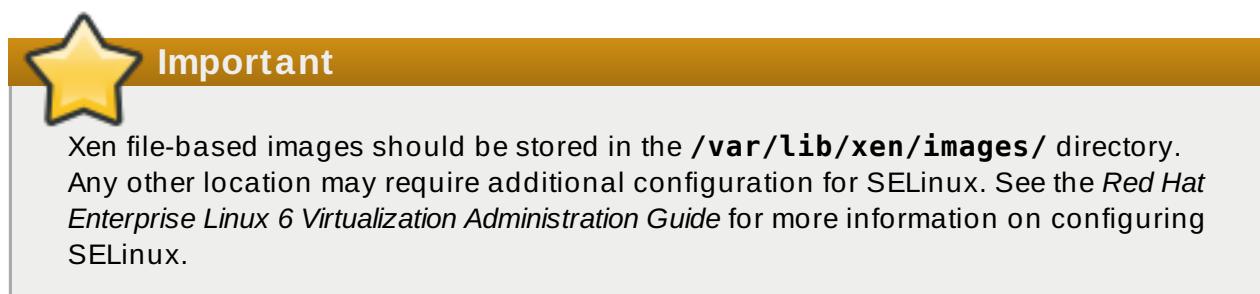
Enter the location of the installation tree.



Figure 14.5. The virtual machine creation wizard

Click **Forward** to continue.

8. Storage setup



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



Figure 14.6. The virtual machine creation wizard

Click **Forward** to continue.

Note

Live and offline migrations require guests to be installed on shared network storage. For information on setting up shared storage for guests, see the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* chapter on Storage Pools.

9. Network setup

Select either **Virtual network** or **Shared physical device**.

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

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

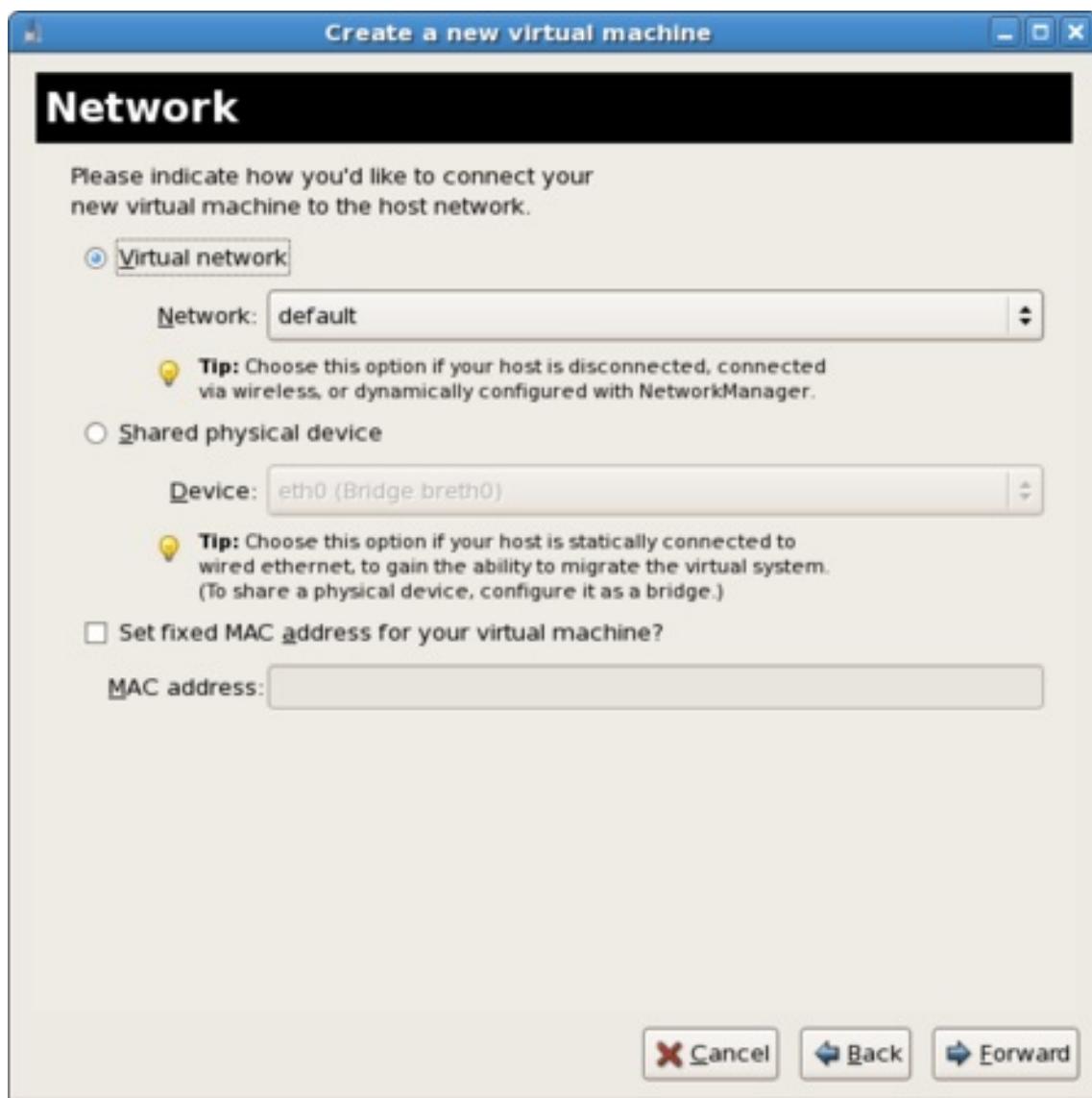


Figure 14.7. The virtual machine creation wizard

Click **Forward** to continue.

10. Memory and CPU allocation

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

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

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

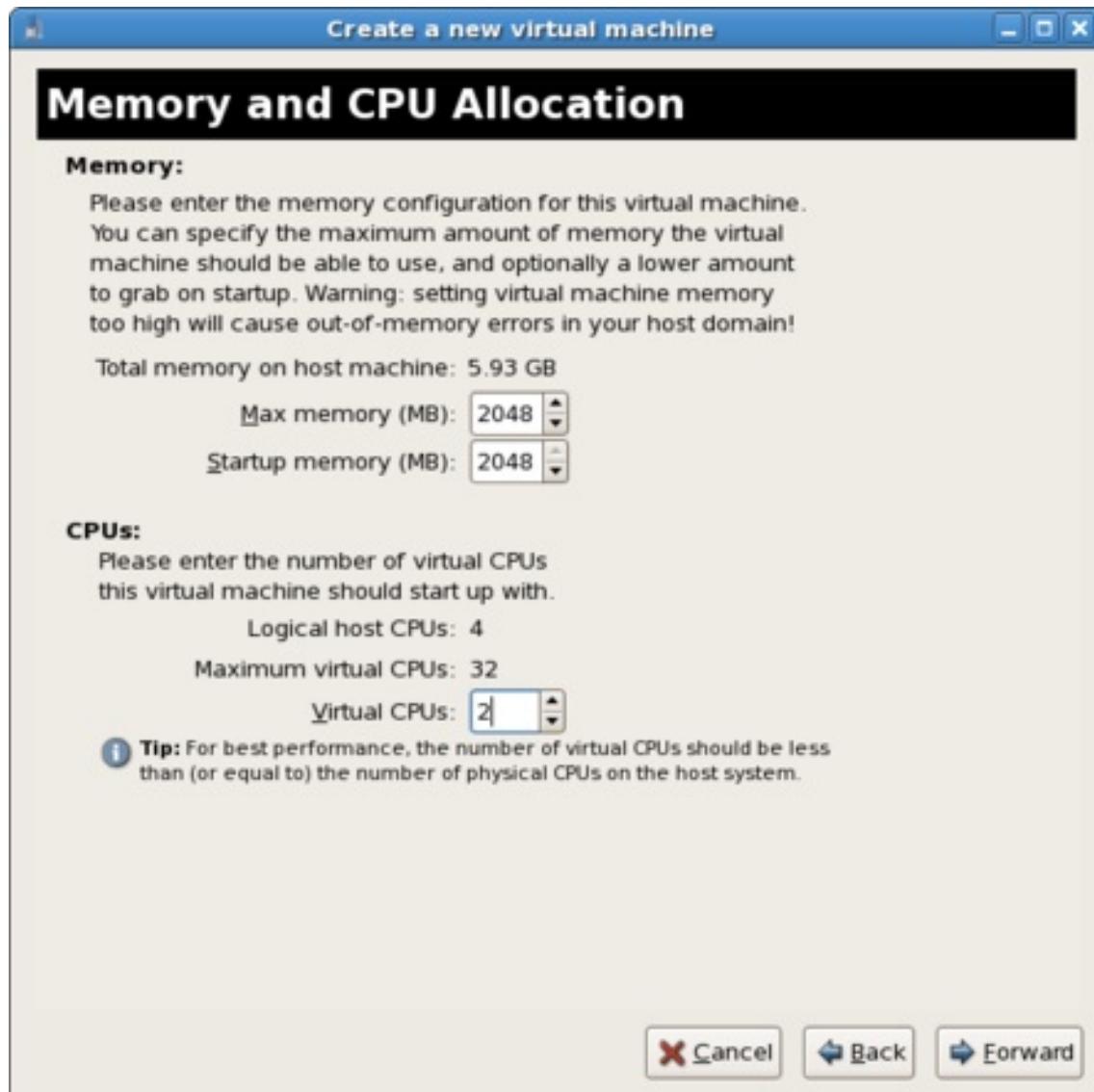


Figure 14.8. The virtual machine creation wizard

Click **Forward** to continue.

11. Verify and start guest installation

Verify the configuration.

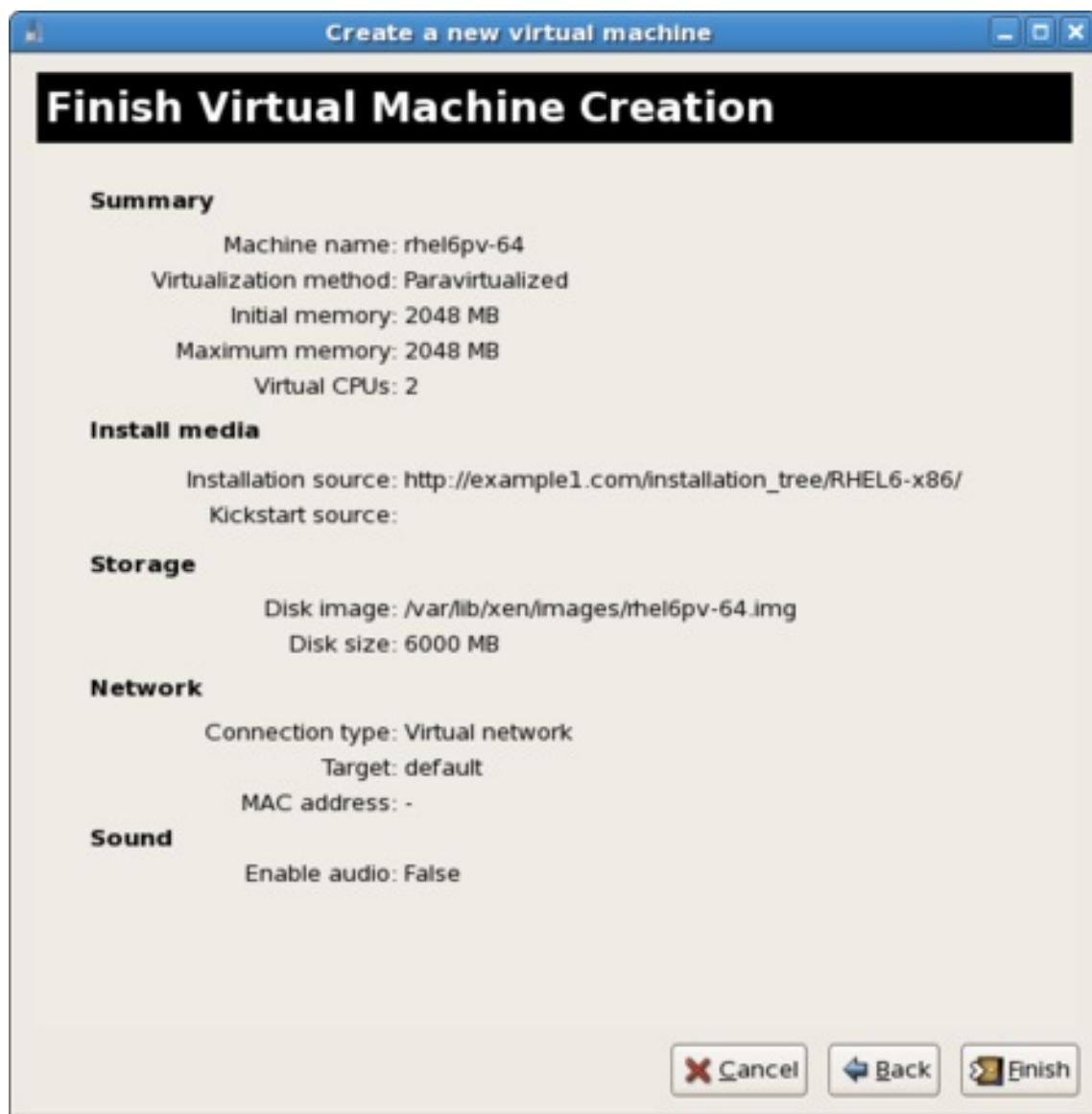


Figure 14.9. The virtual machine creation wizard

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

12. Installing Red Hat Enterprise Linux

Complete the Red Hat Enterprise Linux installation sequence. See the *Red Hat Enterprise Linux 6 Installation Guide* for detailed installation instructions.

14.2. Installing Red Hat Enterprise Linux 6 as a Xen fully virtualized guest on Red Hat Enterprise Linux 5

This section describes how to install Red Hat Enterprise Linux 6 as a Xen fully virtualized guest on Red Hat Enterprise Linux 5 using PV-on-HVM drivers. You must have at least Red Hat Enterprise Linux 5 with update 7 (5.7) to support this action.

Chapter 15. PCI passthrough

This chapter covers using PCI passthrough with Xen and KVM hypervisors.

KVM and Xen hypervisors support attaching PCI devices on the host system to 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 4 are not removable. This means there are up to 28 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 15.1. 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 Red Hat Enterprise Linux. The extensions must be enabled in the BIOS. Some system manufacturers disable these extensions by default.

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

2. 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 Red Hat Enterprise Linux Server (2.6.18-190.el5)
    root (hd0,0)
    kernel /vmlinuz-2.6.18-190.el5 ro root=/dev/VolGroup00/LogVol00
    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 15.2. Preparing an AMD system for PCI passthrough

► Enable AMD IOMMU extensions

The AMD IOMMU extensions are required for PCI passthrough with Red Hat Enterprise Linux. 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.



Important

Xen and KVM require different kernel arguments to enable PCI passthrough. The previous instructions are for KVM. For both AMD and Intel systems, PCI passthrough on Xen requires the **iommu=on** parameter to the hypervisor command line. Modify the **/boot/grub/grub.conf** file as follows to enable PCI passthrough:

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.18-192.el5)
root (hd0,0)
kernel /xen.gz-2.6.18-192.el5 iommu=on
module /vmlinuz-2.6.18-192.el5xen ro root=/dev/VolGroup00/LogVol00
module /initrd-2.6.18-190.el5xen.img
```

15.1. Adding a PCI device with virsh

These steps cover adding a PCI device to a fully virtualized guest under the Xen or KVM hypervisors using hardware-assisted PCI passthrough. See [Section 15.5, “PCI passthrough for para-virtualized Xen guests on Red Hat Enterprise Linux”](#) for details on adding a PCI device to a para-virtualized Xen guest.



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 8086 is a variable that in this case represents Intel equipment, and **** is a four digit hexadecimal code specific to each device):

```
pci_8086_****
```



Note

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.

- 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/D0 (ICH10 Family) USB2 EHCI Controller #2</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
  </capability>
</device>
```

- 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-detach pci_8086_3a6c
Device pci_8086_3a6c detached
```

- 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'
```

-

Run **virsh edit** (or **virsh attach device**) and add a device entry in the **<devices>** section to attach the PCI device to the guest. Only run this command on offline guests. Red Hat Enterprise Linux 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>
```

- 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
```

- Detach the device:

```
$ virsh nodedev-detach pci_8086_3a6c
```

- 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
```

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

```
# setsebool -P virt_use_sysfs 1
```

- 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.

15.2. 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 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 8086 is a variable that in this case represents Intel equipment, and **** is a four digit hexadecimal code specific to each device):

```
pci_8086_****
```



Note

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.

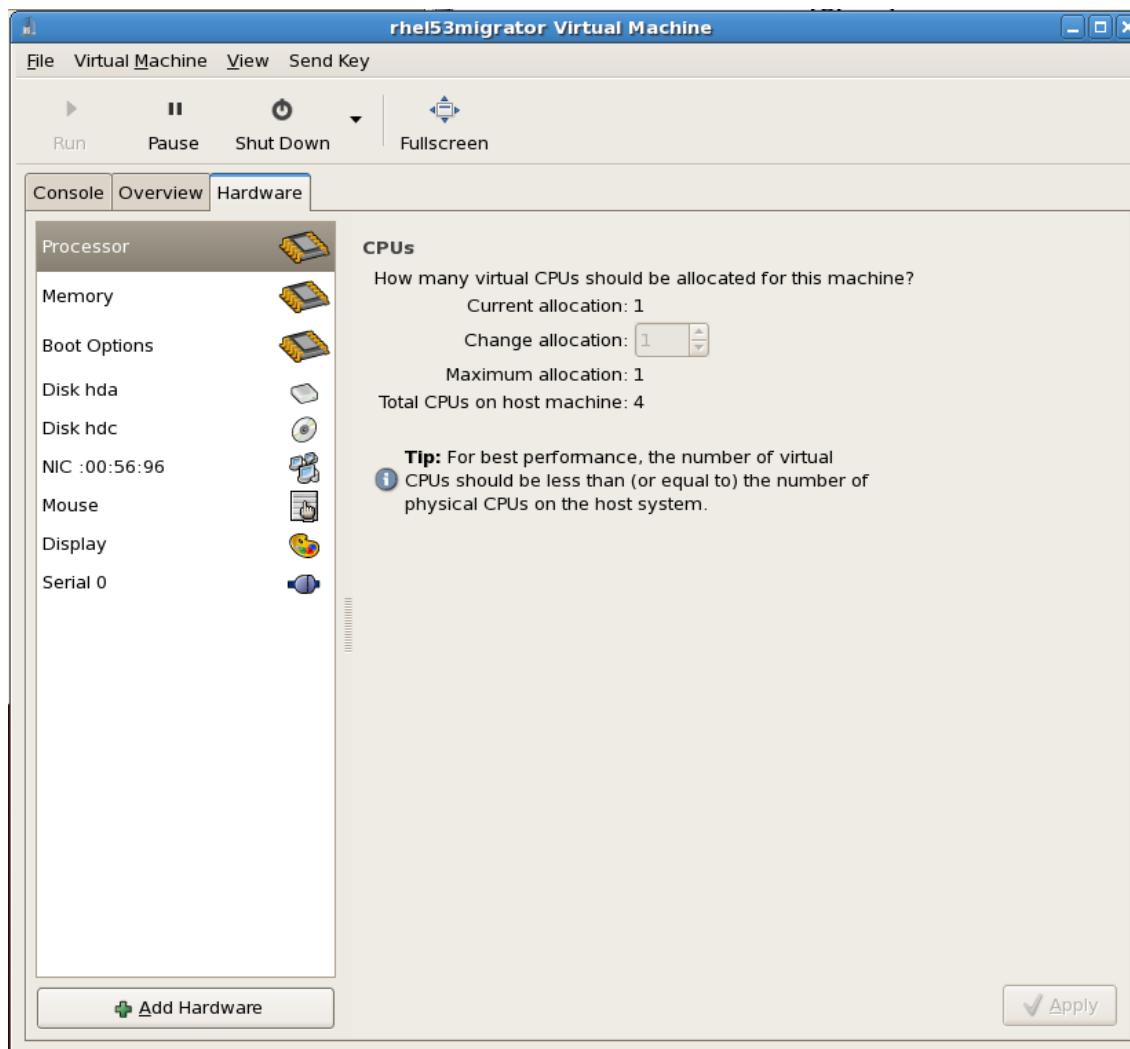
```
# virsh nodedev-detach pci_8086_3a6c  
Device pci_8086_3a6c detached
```

3. Power off the guest

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

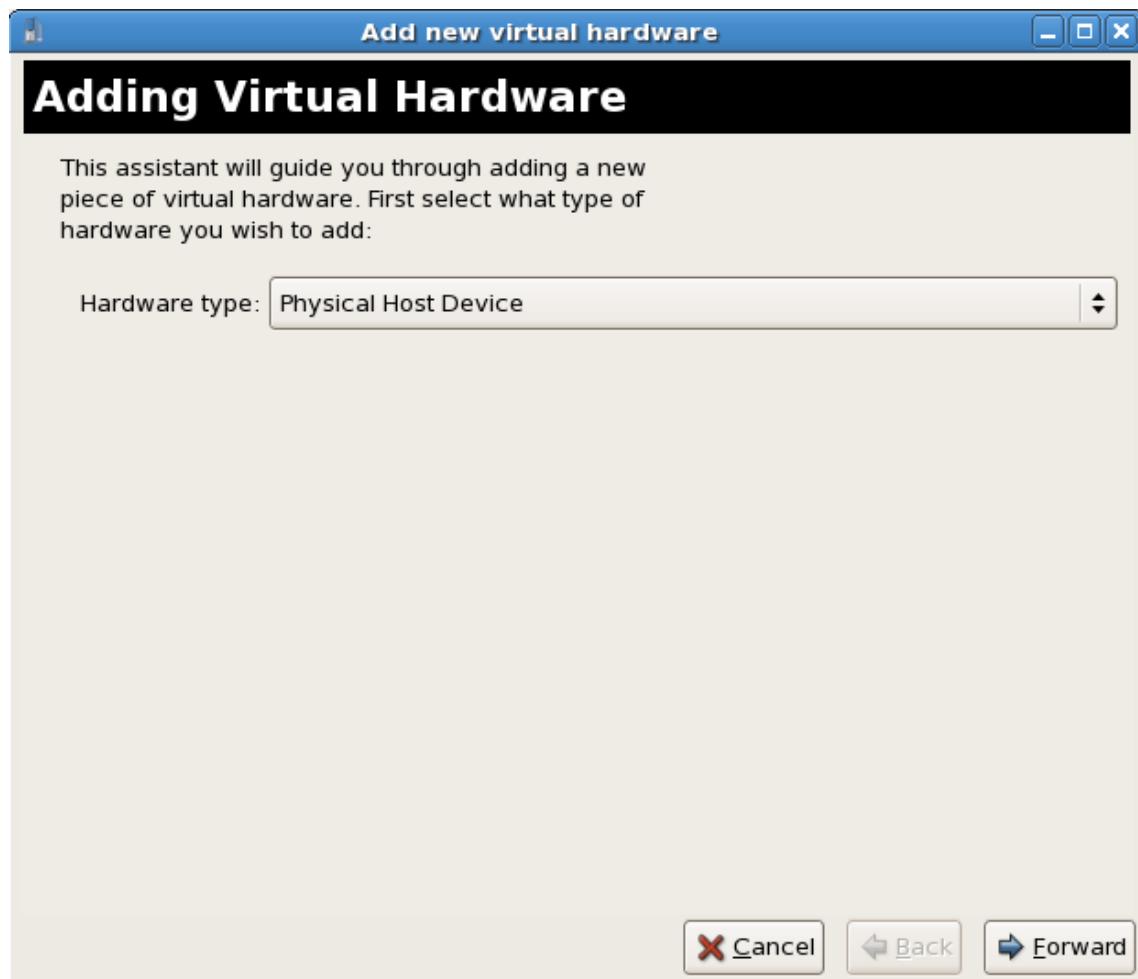
4. Open the hardware settings

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 that 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.

15.3. 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 `8086` is a variable that in this case represents Intel equipment, and `****` is a four digit hexadecimal code specific to each device):

```
pci_8086_****
```

**Note**

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 fedorall -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.

15.4. Removing a PCI passthrough device for host re-use

Perform the following steps to remove a PCI passthrough device so that the host regains full access to the device:

1. List all PCI devices

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 8086 is a variable that in this case represents Intel equipment, and **** is a four digit hexadecimal code specific to each device):

```
pci_8086_****
```

**Note**

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

2. Remove and re-attach the device

After removing the device either from the guest XML file or virt-manager, run the **virsh nodedev-reattach** command to return its use to the host, substituting your PCI device name that is designated for removal:

```
# virsh nodedev-reattach pci_8086_3a6c
```

15.5. PCI passthrough for para-virtualized Xen guests on Red Hat Enterprise Linux

PCI passthrough is used to allow a Xen guest exclusive access to a PCI device, rather than sharing with other guests or with dom0. PCI passthrough for para-virtualized Xen guests is supported on all Red Hat Enterprise Linux 5 systems, however PCI passthrough with *fully virtualized guests* is only supported on Red Hat Enterprise Linux 5.4 and newer.



Warning

PCI passthrough to para-virtualized guests is considered insecure and is not supported for Red Hat Enterprise Linux 6 guests.

Limitations of Xen PCI passthrough:

Any guest using PCI passthrough will no longer be available for save, restore, or migration capabilities, as it will be tied to a particular non-virtualized hardware configuration.

A guest which has access to a non-virtualized PCI device via PCI passthrough also has the potential to access the DMA address space of dom0, which is a potential security concern.

To link a PCI device to a guest the device must first be hidden from the host. If the host is using the device, the device cannot be assigned to the guest.

Procedure 15.3. Example: attaching a PCI device

- Given a network device which uses the bnx2 driver and has a PCI id of 0000:09:00.0, the following lines added to **/etc/modprobe.conf** hides the device from dom0. Either the **bnx2** module must be reloaded or the host must be restarted.

```
install bnx2 /sbin/modprobe pciback; /sbin/modprobe --first-time --ignore-
install bnx2
options pciback hide=(0000:09:00.0)
```

- Multiple PCI identifiers can be added to **/etc/modprobe.conf** to hide multiple devices.

```
options pciback hide=(0000:09:00.0)(0000:0a:04.1)
```

- Use one of the following methods to add the passed-through device to the guest's configuration file:

- » **virsh** ([Section 15.1, “Adding a PCI device with virsh”](#) - Step 5);
- » **virt-manager** ([Section 15.2, “Adding a PCI device with virt-manager”](#)); or
- » **virt-install** ([Section 15.3, “PCI passthrough with virt-install”](#))



Warning

Due to interrupt tracking, repeatedly hotplugging or hotunplugging an assigned device more than 512 times in a brief period of time can cause a kernel error. Please do not repeatedly hotplug/hotunplug an assigned device.



Note

When running Red Hat Enterprise Linux 5 as a KVM guest, the **acpiphp** kernel module must be loaded in the guest to support dynamic addition and removal of PCI devices. This module enables the guest to receive insertion and removal notifications from **qemu**. To manually load this module, run the following command in the guest:

```
# modprobe acpiphp
```

To enable this module to be loaded automatically on every guest boot, perform the following commands in the guest:

```
# echo 'modprobe acpiphp' > /etc/sysconfig/modules/acpiphp.modules
```

```
# chmod +x /etc/sysconfig/modules/acpiphp.modules
```

After reboot, the module should be loaded and can be confirmed with the **lsmod | grep acpiphp** command. More information on persistent module loading in Red Hat Enterprise Linux 5 can be found in the *Red Hat Enterprise Linux 5 Deployment Guide*.

Chapter 16. SR-IOV

16.1. Introduction

The PCI-SIG (PCI Special Interest Group) developed the Single Root I/O Virtualization (SR-IOV) specification. The PCI-SIG Single Root 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 guests.

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

SR-IOV uses two 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 guests.

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

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

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

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

Disadvantages of SR-IOV

Live migration is presently unsupported. 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.

16.2. Using SR-IOV

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

SR-IOV requires Intel VT-d support.



Important

Xen requires additional kernel arguments to use SR-IOV. Modify the **/boot/grub/grub.conf** file to enable SR-IOV. To enable SR-IOV with Xen for Intel systems append the **pci_pt_e820_access=on** parameter to the kernel.

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.18-192.el5xen)
root (hd0,0)
kernel /xen.gz-2.6.18-192.el5 iommu=1
module /vmlinuz-2.6.18-192.el5xen ro root=/dev/VolGroup00/LogVol00
pci_pt_e820_access=on
module /initrd-2.6.18-192.el5xen.img
```

Procedure 16.1. Attach an SR-IOV network device

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

Enable Intel VT-D in BIOS. See [Procedure 15.1, “Preparing an Intel system for PCI passthrough”](#) for more information on enabling Intel VT-d in BIOS and the kernel, or see your system manufacturer's documentation for specific instructions.

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

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

3. Start the SR-IOV kernel modules

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

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

4. Activate Virtual Functions

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

Remove the module to change the variable.

```
# modprobe -r igb
```

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

```
# modprobe igb max_vfs=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 8](#). Note the **bus**, **slot** and **function** parameters of the Virtual Function, these are required for adding the device.

8.

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' managed='yes'>
  <source>
    <address bus='0x03' slot='0x10' function='0x01' />
  </source>
</hostdev>
```

- e. Save the configuration.

9. 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.

16.3. 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.

Chapter 17. 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.

Presently, only Red Hat Enterprise Linux 5.4 and newer guests fully support the para-virtualized clock.

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.



Important

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.



Important

These instructions are for AMD revision F cpus only.

If the CPU lacks the **constant_tsc** bit, disable all power management features ([BZ#513138](#)). 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	notsc lpj=n
5.4 x86 with the para-virtualized clock	Additional parameters are not required
5.4 x86 without the para-virtualized clock	clocksource=acpi_pm lpj=n
5.3 AMD64/Intel 64	notsc
5.3 x86	clocksource=acpi_pm
4.8 AMD64/Intel 64	notsc
4.8 x86	clock=pmtmr
3.9 AMD64/Intel 64	Additional parameters are not required
3.9 x86	Additional parameters are not required



Warning

The ***divider*** kernel parameter was previously recommended for Red Hat Enterprise Linux 4 and 5 guests that did not have high responsiveness requirements, or exist on systems with high guest density. It is no longer recommended for use with guests running Red Hat Enterprise Linux 4, or Red Hat Enterprise Linux 5 versions prior to version 5.8.

divider can improve throughput on Red Hat Enterprise Linux 5 versions equal to or later than 5.8 by lowering the frequency of timer interrupts. For example, if **HZ=1000**, and ***divider*** is set to **10** (that is, **divider=10**), the number of timer interrupts per period changes from the default value (1000) to 100 (the default value, 1000, divided by the divider value, 10).

[BZ#698842](#) details a bug in the way that the ***divider*** parameter interacts with interrupt and tick recording. This bug is fixed as of Red Hat Enterprise Linux 5.8. However, the ***divider*** parameter can still cause kernel panic in guests using Red Hat Enterprise Linux 4, or Red Hat Enterprise Linux 5 versions prior to version 5.8.

This parameter was not implemented in Red Hat Enterprise Linux 3, so this bug does not affect Red Hat Enterprise Linux 3 guests.

Red Hat Enterprise Linux 6 does not have a fixed-frequency clock interrupt; it operates in *tickless mode* and uses the timer dynamically as required. The ***divider*** parameter is therefore not useful for Red Hat Enterprise Linux 6, and Red Hat Enterprise Linux 6 guests are not affected by this bug.

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

Windows uses 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, see [Available switch options for the Windows XP and the Windows Server 2003 Boot.ini files](#).

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

Windows supports both the Real-Time Clock (RTC) and the Time Stamp Counter (TSC). For some Windows guests, the RTC can be selected for use instead of the TSC for all of the guest's time sources. This can resolve some guest timing issues.



Note

The **boot.ini** file is no longer used in Windows Vista and newer. As shown in this procedure, Windows Vista, Windows Server 2008 and Windows 7 use the **Boot Configuration Data Editor (bcdedit.exe)** application to modify this boot parameter.



Important

This procedure is only recommended if the guest is having time keeping issues, and is not supported in Windows Server 2008 or Windows Server 2008 SP2 as they do not support the USEPLATFORMCLOCK parameter demonstrated here and already use the RTC by default. This procedure is however supported in Windows Server 2008 R2.

1. Open the Windows guest.
2. In the guest, open the **Accessories** menu of the **start** menu. Right click on the **Command Prompt** application, and select **Run as Administrator**.
3. Confirm any security exception, if prompted.
4. Set the boot manager to use the RTC (platform clock) for the primary clock source. The system UUID (**{default}** in the following example) should be changed if your system UUID is different than the default boot device.

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

Time keeping for Windows Vista, Windows Server 2008 R2 and Windows 7 guests should now be improved.

Part IV. Administration

Administering virtualized systems

These chapters contain information for administering host and guests using tools included in Red Hat Enterprise Linux.

Chapter 18. Server best practices

The following tasks and tips can assist you with securing and ensuring reliability of your Red Hat Enterprise Linux 5 server host (dom0).

- » 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.

Chapter 19. Security for virtualization

When deploying virtualization technologies on your corporate infrastructure, you must ensure that the host cannot be compromised. The host, in the Xen hypervisor, is a privileged domain that handles system management and manages all virtual machines. If the host is insecure, all other domains 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 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 Security-Enhanced Linux (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 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 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 Red Hat recommends all Red Hat Enterprise Linux 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 correctly labeled. SELinux requires that image files have the **virt_image_t** label applied to them. The **/var/lib/libvirt/images** directory has this label applied to it and its contents by default. This does not mean that images must be stored in this directory; images can be stored anywhere, provided they are labeled with **virt_image_t**.

Adding LVM based storage with SELinux in enforcing mode

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

Procedure 19.1. Creating and mounting a logical volume on a 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 a Xen folder.

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

Alternatively, set the correct SELinux type for a KVM folder.

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

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

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

6. Label the device node (for example, **/dev/volumegroup/NewVolumeName** with the correct label:

```
# semanage fcontext -a -t xen_image_t /dev/volumegroup/NewVolumeName
# restorecon /dev/volumegroup/NewVolumeName
```

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 xen_image_t -f -b /dev/sda2
# restorecon /dev/sda2
```

The Boolean parameter `xend_disable_t` can set the `xend` to unconfined mode after restarting the daemon. It is better to disable protection for a single daemon than the whole system. It is advisable that you should not re-label directories as `xen_image_t` that you will use elsewhere.

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
<code>allow_unconfined_qemu_transition</code>	Default: off. This boolean controls whether KVM guests can be transitioned to unconfined users.
<code>qemu_full_network</code>	Default: on. This boolean controls full network access to KVM guests.
<code>qemu_use_cifs</code>	Default: on. This boolean controls KVM's access to CIFS or Samba file systems.
<code>qemu_use_comm</code>	Default: off. This boolean controls whether KVM can access serial or parallel communications ports.
<code>qemu_use_nfs</code>	Default: on. This boolean controls KVM's access to NFS file systems.
<code>qemu_use_usb</code>	Default: on. This boolean allows KVM to access USB devices.

19.4. Virtualization firewall information

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

Note

Any network service on a 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.
- » Port 8002 is used by Xen for live migration.
- » 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 required for virtual bridge devices. Note that installing libvirt enables this variable so it will be enabled when the virtualization packages are installed unless it was manually disabled.



Note

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

Chapter 20. Managing guests with xend

The **xend** node control daemon performs certain system management functions that relate to virtual machines. This daemon controls the virtualized resources, and **xend** must be running to interact with virtual machines. Before you start **xend**, you must specify the operating parameters by editing the **xend** configuration file `/etc/xen/xend-config.sxp`. Here are the parameters you can enable or disable in the **xend-config.sxp** configuration file:

Table 20.1. xend configuration parameters

Item	Description
(console-limit)	Determines the console server's memory buffer limit and assigns that limit on a per domain basis.
(min-mem)	Determines the minimum number of megabytes that is reserved for domain0 (if you enter 0, the value does not change).
(dom0-cpus)	Determines the number of CPUs in use by domain0 (at least 1 CPU is assigned by default).
(enable-dump)	If this is enabled, when a crash occurs Xen creates a dump file (the default is 0).
(external-migration-tool)	Determines the script or application that handles external device migration. The scripts must reside in the <code>/etc/xen/scripts/external-device-migrate</code> directory.
(logfile)	Determines the location of the log file (default is <code>/var/log/xend.log</code>).
(loglevel)	Filters out the log mode values: DEBUG, INFO, WARNING, ERROR, or CRITICAL (default is DEBUG).
(network-script)	Determines the script that enables the networking environment. The scripts must reside in the <code>/etc/xen/scripts/</code> directory.
(xend-http-server)	Enables the http stream packet management server (the default is no).
(xend-unix-server)	Enables the UNIX domain socket server. The socket server is a communications endpoint that handles low level network connections and accepts or rejects incoming connections. The default value is set to yes.
(xend-relocation-server)	Enables the relocation server for cross-machine migrations (the default is no).
(xend-unix-path)	Determines the location where the xend-unix-server command outputs data (default is <code>/var/lib/xend/xend-socket</code>)
(xend-port)	Determines the port that the http management server uses (the default is 8000).
(xend-relocation-port)	Determines the port that the relocation server uses (the default is 8002).

Item	Description
(xend-relocation-address)	Determines the host addresses allowed for migration. The default value is the value of xend-address .
(xend-address)	Determines the address that the domain socket server binds to. The default value allows all connections.

After setting these operating parameters, you should verify that xend is running and if not, initialize the daemon. At the command prompt, you can start the **xend** daemon by entering the following:

```
service xend start
```

You can use **xend** to stop the daemon:

```
service xend stop
```

This stops the daemon from running.

You can use **xend** to restart the daemon:

```
service xend restart
```

The daemon starts once again.

You check the status of the **xend** daemon.

```
service xend status
```

The output displays the daemon's status.

Note

Use the **chkconfig** command to add the **xend** to the **initscript**.

```
chkconfig --level 345 xend
```

The **xend** will now start in runlevels 3, 4 and 5.

Chapter 21. Xen live migration

The Xen hypervisor supports Virtualization Migration for para-virtualized guests and fully virtualized guests. Migration is only supported on Red Hat Enterprise Linux 5.1 and newer systems. Migration can be performed offline or live.

- » Offline migration suspends the guest on the original host, transfers it to the destination host and then resumes it once the guest is fully transferred. Offline migration uses the **virsh migrate** command.
- ```
virsh migrate GuestName libvirtURI
```
- » 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 transferred. The Xen hypervisor estimates the time remaining and attempts to transfer the maximum amount of page files from the source to the destination until Xen predicts the amount of remaining pages can be transferred during a very brief time while the 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.

Live migration uses the **--live** option for the **virsh migrate** command.

```
virsh migrate--live GuestName libvirtURI
```



### Important

Migration is presently unsupported on the Itanium® architecture.

To enable migration with Xen a few changes must be made to the **/etc/xen/xend-config.sxp** configuration file. By default, migration is disabled as migration can be a potential security hazard if incorrectly configured. Opening the migration port can allow an unauthorized host to initiate a migration or connect to the migration ports. Authentication and authorization are not configured for migration requests and the only control mechanism is based on hostnames and IP addresses. Special care should be taken to ensure the migration port is not accessible to unauthorized hosts.



### Important

IP address and hostname filters only offer minimal security. Both of these attributes can be forged if the attacker knows the address or hostname of the migration client. The best method for securing migration is to isolate the network from external and unauthorized internal connections.

### Enabling migration

Modify the following entries in **/etc/xen/xend-config.sxp** to enable migration. Modify the values, when necessary, and remove the comments (the **#** symbol) preceding the following parameters:

**(xend-relocation-server yes)**

The default value, which disables migration, is **no**. Change the value of **xend-relocation-server** to **yes** to enable migration.

**(xend-relocation-port 8002)**

The parameter, **(xend-relocation-port)**, specifies the port **xend** should use for the relocation interface, if **xend-relocation-server** is set to **yes**.

The default value of this variable should work for most installations. If you change the value make sure you are using an unused port on the relocation server.

The port set by the **xend-relocation-port** parameter must be open on both systems.

**(xend-relocation-address '')**

**(xend-relocation-address)** is the address the **xend** listens for migration commands on the **relocation-socket** connection if **xend-relocation-server** is set.

The default is to listen on all active interfaces. The **(xend-relocation-address)** parameter restricts the migration server to only listen to a specific interface. The default value in **/etc/xen/xend-config.sxp** is an empty string('''). This value should be replaced with a single, valid IP address. For example:

```
(xend-relocation-address '10.0.0.1')
```

**(xend-relocation-hosts-allow '')**

The **(xend-relocation-hosts-allow 'hosts')** parameter controls which hostnames can communicate on the relocation port.

Unless you are using SSH or TLS, the guest's virtual memory is transferred in raw form without encryption of the communication. Modify the **xend-relocation-hosts-allow** option to restrict access to the migration server.

If the value is empty, as denoted in the example above by an empty string surrounded by single quotes, then all connections are allowed. This assumes the connection arrives on a port and interface which the relocation server listens on, see also **xend-relocation-port** and **xend-relocation-address**.

Otherwise, the **(xend-relocation-hosts-allow)** parameter should be a sequence of regular expressions separated by spaces. Any host with a fully-qualified domain name or an IP address which matches one of these regular expressions will be accepted.

An example of a **(xend-relocation-hosts-allow)** attribute:

```
(xend-relocation-hosts-allow '^localhost$ ^localhost\\.localdomain$')
```

After you have configured the parameters in your configuration file, restart the Xen service.

```
service xend restart
```

## 21.1. A live migration example

Below is an example of how to setup a simple environment for live migration. This configuration is using **NFS** for the shared storage. **NFS** is suitable for demonstration environments but for a production environment a more robust shared storage configuration using Fibre Channel or iSCSI and **GFS** is recommended.

The configuration below consists of two servers (**et-virt07** and **et-virt08**), both of them are using **eth1** as their default network interface hence they are using **xenbr1** as their Xen networking bridge. We are using a locally attached SCSI disk (**/dev/sdb**) on **et-virt07** for shared storage using **NFS**.

## Setup for live migration

Create and mount the directory used for the migration:

```
mkdir /var/lib/libvirt/images
mount /dev/sdb /var/lib/libvirt/images
```



### Important

Ensure the directory is exported with the correct options. If you are exporting the default directory **/var/lib/libvirt/images/** make sure you *only* export **/var/lib/libvirt/images/** and *not* **/var/lib/xen/** as this directory is used by the **xend** daemon and other tools. Sharing **/var/lib/xen/** will cause unpredictable behavior.

```
cat /etc/exports
/var/lib/libvirt/images *(rw,async,no_root_squash)
```

Verify it is exported via **NFS**:

```
showmount -e et-virt07
Export list for et-virt07:
/var/lib/libvirt/images *
```

## Install the guest

The install command in the example used for installing the guest:

```
virt-install -p -f /var/lib/libvirt/images/testvm1.dsk -s 5 -n\
testvm1 --vnc -r 1024 -l http://example.com/RHEL5-tree\
Server/x86-64/os/ -b xenbr1
```

For step by step installation instructions, see [Chapter 8, Guest operating system installation procedures](#).

## Verify environment for migration

Make sure the virtualized network bridges are configured correctly and have the same name on both hosts:

```
[et-virt08 ~]# brctl show
bridge name bridge id STP enabled interfaces
xenbr1 8000.ffffffffffff no peth1
vif0.1
```

```
[et-virt07 ~]# brctl show
bridge name bridge id
xenbr1 8000.firebaseio
vif0.1 STP enabled
 no
 interfaces
 peth1
```

Verify the relocation parameters are configured on both hosts:

```
[et-virt07 ~]# grep xend-relocation /etc/xen/xend-config.sxp |grep -v '#'
(xend-relocation-server yes)
(xend-relocation-port 8002)
(xend-relocation-address '')
(xend-relocation-hosts-allow '')
```

```
[et-virt08 ~]# grep xend-relocation /etc/xen/xend-config.sxp |grep -v '#'
(xend-relocation-server yes)
(xend-relocation-port 8002)
(xend-relocation-address '')
(xend-relocation-hosts-allow '')
```

Make sure the relocation server has started and is listening on the dedicated port for Xen migrations (8002):

```
[et-virt07 ~]# lsof -i :8002
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
python 3445 root 14u IPv4 10223 TCP *:teradataordbms (LISTEN)
```

```
[et-virt08 ~]# lsof -i :8002
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
python 3252 root 14u IPv4 10901 TCP *:teradataordbms (LISTEN)
```

That the default **/var/lib/libvirt/images** directory is available and mounted with networked storage on both hosts. Shared, networked storage is required for migrations.

```
[et-virt08 ~]# df /var/lib/libvirt/images
Filesystem 1K-blocks Used Available Use% Mounted on
et-virt07:/var/lib/libvirt/images 70562400 2379712 64598336 4%
/var/lib/libvirt/images
```

```
[et-virt08 ~]# file /var/lib/libvirt/images/testvm1.dsk
/var/lib/libvirt/images/testvm1.dsk: x86 boot sector; partition 1: ID=0x83,
active, starthead 1, startsector 63, 208782 sectors; partition 2: ID=0x8e,
starthead 0, startsector 208845, 10265535 sectors, code offset 0x48
```

```
[et-virt08 ~]# touch /var/lib/libvirt/images/foo
[et-virt08 ~]# rm -f /var/lib/libvirt/images/foo
```

## Verify saving and restoring the guest

Start the virtual machine (if the virtual machine is not on):

```
[et-virt07 ~]# virsh list
 Id Name State

 Domain-0 running
```

```
[et-virt07 ~]# virsh start testvm1
Domain testvm1 started
```

Verify the virtual machine is running:

```
[et-virt07 ~]# virsh list
 Id Name State

Domain-0 running
testvm1 blocked
```

Save the virtual machine on the local host:

```
[et-virt07 images]# time virsh save testvm1 testvm1.sav
real 0m15.744s
user 0m0.188s
sys 0m0.044s
```

```
[et-virt07 images]# ls -lrt testvm1.sav
-rwxr-xr-x 1 root root 1075657716 Jan 12 06:46 testvm1.sav
```

```
[et-virt07 images]# virsh list
 Id Name State

Domain-0 running
testvm1 blocked
```

Restore the virtual machine on the local host:

```
[et-virt07 images]# virsh restore testvm1.sav
```

```
[et-virt07 images]# virsh list
 Id Name State

Domain-0 running
testvm1 blocked
```

Start the live migration of **domain-id** from **et-virt08** to **et-virt07**. The hostname you are migrating to and <domain-id> must be replaced with valid values. This example uses the **et-virt08** host which must have SSH access to **et-virt07**

```
[et-virt08 ~]# xm migrate --live testvm1 et-virt07
```

Verify the virtual machine is no longer present on **et-virt08**

```
[et-virt08 ~]# virsh list
 Id Name State

Domain-0 running
testvm1 blocked
```

Verify the virtual machine has been migrated to **et-virt07**:

```
[et-virt07 ~]# virsh list
 Id Name State

Domain-0 running
testvm1 running
```

## Testing the progress and initiating the live migration

Create the following script inside the virtual machine to log date and hostname during the migration. This script performs I/O tasks on the guest's file system.

```
#!/bin/bash

while true
do
touch /var/tmp/$$.log
echo `hostname` >> /var/tmp/$$.log
echo `date` >> /var/tmp/$$.log
cat /var/tmp/$$.log
df /var/tmp
ls -l /var/tmp/$$.log
sleep 3
done
```

Remember, that script is only for testing purposes and unnecessary for a live migration in a production environment.

Verify the virtual machine is running on **et-virt08** before we try to migrate it to **et-virt07**:

| Id       | Name           | State   |
|----------|----------------|---------|
| Domain-0 |                | running |
|          | <i>testvm1</i> | blocked |

Initiate a live migration to **et-virt07**. You can add the **time** command to see how long the migration takes:

```
[et-virt08 ~]# xm migrate --live testvm1 et-virt07
```

run the script inside the guest:

```
./doit
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:27 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 62 Jan 12 02:26 /var/tmp/2279.log
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:27 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:30 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 124 Jan 12 02:26 /var/tmp/2279.log
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:27 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:30 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:33 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 186 Jan 12 02:26 /var/tmp/2279.log
Fri Jan 12 02:26:45 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:48 EST 2007
```

```

dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:51 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:54:57 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:55:00 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:55:03 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 744 Jan 12 06:55 /var/tmp/2279.log
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:26:27 EST 2007

```

Verify the virtual machine has been shut down on **et-virt08**:

```
[et-virt08 ~]# virsh list
 Id Name State

 Domain-0 running
```

Verify the virtual machine has started up on **et-virt07**:

```
[et-virt07 images]# virsh list
 Id Name State

 Domain-0 running
 testvm1 blocked
```

Run through another cycle migrating from **et-virt07** to **et-virt08**. Initiate a migration from **et-virt07** to **et-virt08**:

```
[et-virt07 images]# xm migrate --live testvm1 et-virt08
```

Verify the virtual machine has been shut down:

```
[et-virt07 images]# virsh list
 Id Name State

 Domain-0 running
```

Before initiating the migration start the simple script in the guest and note the change in time when migrating the guest:

```

./doit
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:53 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 62 Jan 12 06:57 /var/tmp/2418.log
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:53 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:56 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 124 Jan 12 06:57 /var/tmp/2418.log
dhcp78-218.lab.boston.redhat.com

```

```

Fri Jan 12 06:57:53 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:56 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:58:00 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 186 Jan 12 06:57 /var/tmp/2418.log
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:53 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:56 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:58:00 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:30:00 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 248 Jan 12 02:30 /var/tmp/2418.log
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:53 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:56 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:58:00 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:30:00 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:30:03 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 310 Jan 12 02:30 /var/tmp/2418.log
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:53 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:57:56 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 06:58:00 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:30:00 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:30:03 EST 2007
dhcp78-218.lab.boston.redhat.com
Fri Jan 12 02:30:06 EST 2007
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
2983664 2043120 786536 73% /
-rw-r--r-- 1 root root 372 Jan 12 02:30 /var/tmp/2418.log

```

After the migration command completes on **et-virt07** verify on **et-virt08** that the virtual machine has started:

```
[et-virt08 ~]# virsh list
 Id Name State

 Domain-0 running
 testvm1 blocked
```

and run another cycle:

```
[et-virt08 ~]# time virsh migrate --live testvm1 et-virt07
real 0m10.378s
user 0m0.068s
sys 0m0.052s
```

At this point you have successfully performed an offline and a live migration test.

## 21.2. Configuring guest live migration

This section covers offline migration of Xen guests to other servers running Red Hat Enterprise Linux. Further, migration is performed in an offline method (using the **xm migrate** command). Live migration can be done from the same command. However there are some additional modifications that you must do to the **xend-config** configuration file. This example identifies the entries that you must modify to ensure a successful migration:

### **(xend-relocation-server yes)**

The default for this parameter is 'no', which keeps the relocation/migration server deactivated (unless on a trusted network) and the domain virtual memory is exchanged in raw form without encryption.

### **(xend-relocation-port 8002)**

This parameter sets the port that **xend** uses for migration. Use this value unless your network environment requires a custom value. Remove the comment symbol to enable it.

### **(xend-relocation-address )**

This parameter is the address that listens for relocation socket connections, after you enable the **xend-relocation-server**. The Xen hypervisor only listens for migration network traffic on the specified interface.

### **(xend-relocation-hosts-allow )**

This parameter controls the host that communicates with the relocation port. If the value is empty, then all incoming connections are allowed. You must change this to a space-separated sequences of regular expressions, for example:

```
(xend-relocation-hosts-allow- '^localhost\\.localdomain$')>
```

Accepted values included fully-qualified domain names, IP addresses or space separated regular expressions.

After configuring, reboot the host to load new settings.

## Chapter 22. KVM live migration

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

Migration is the process of moving a 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 guest's 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 guest's 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 on 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 transferred. 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 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 on network bandwidth and latency. If the network is in heavy use or a low bandwidth the migration will take much longer.

### 22.1. Live migration requirements

Migrating guests requires the following:

#### Migration requirements

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

- iSCSI
  - NFS
  - GFS2
- » Two or more Red Hat Enterprise Linux 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, see [Part V, “Virtualization Storage Topics”](#).

Alternatively, use the NFS example in [Section 22.2, “Share storage example: NFS for a simple migration”](#).

## 22.2. 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 guests.

For advanced and more robust shared storage instructions, see [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
```



#### Warning

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.

## 22.3. Live KVM migration with virsh

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

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

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

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

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

#### 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.

### Note

libvirt supports a variety of networking methods including TLS/SSL, unix sockets, SSH, and unencrypted TCP. See [Chapter 23, Remote management of guests](#) for more information on using other methods.

## 22.4. Migrating with virt-manager

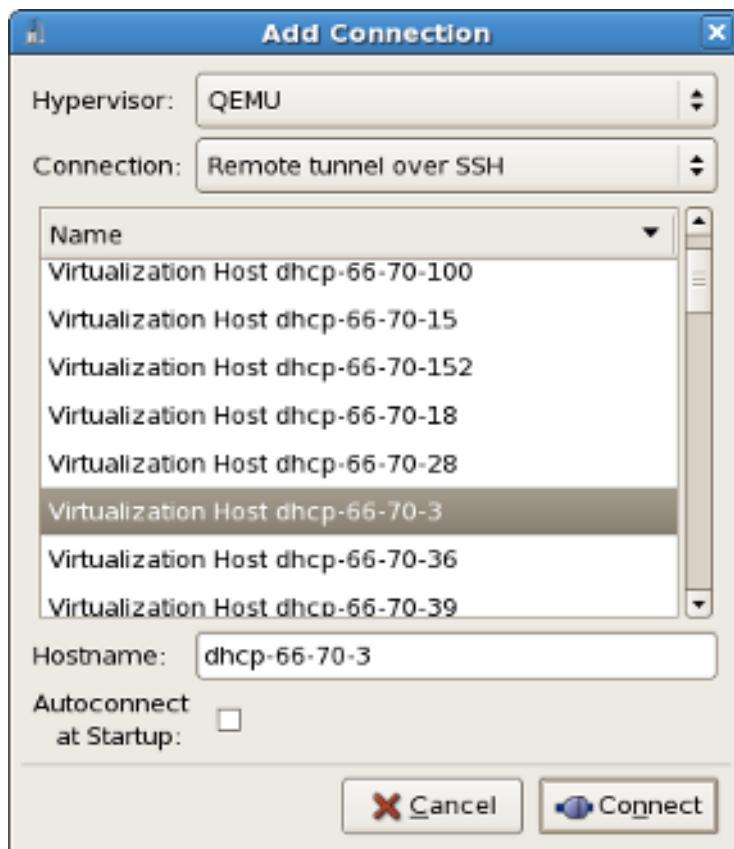
This section covers migrating KVM based guests with **virt-manager**.

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

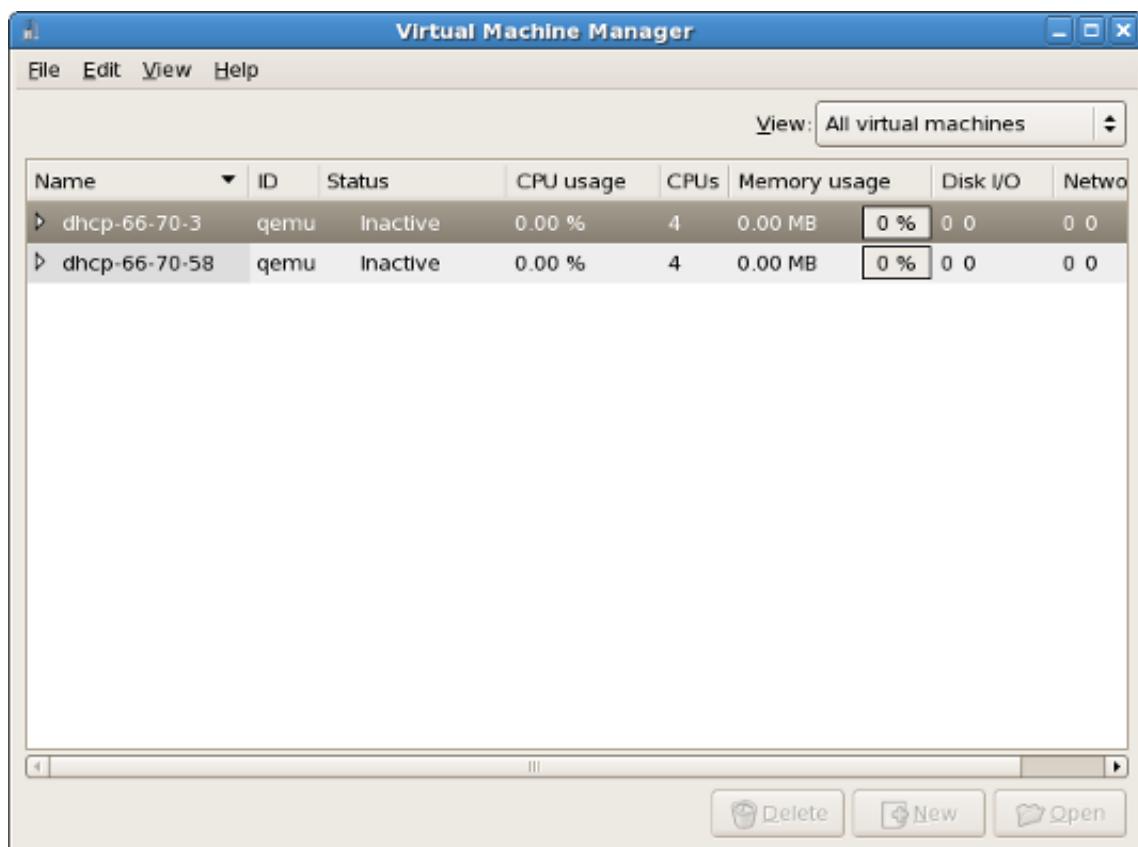
Enter the following details:

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

Click **Connect**.



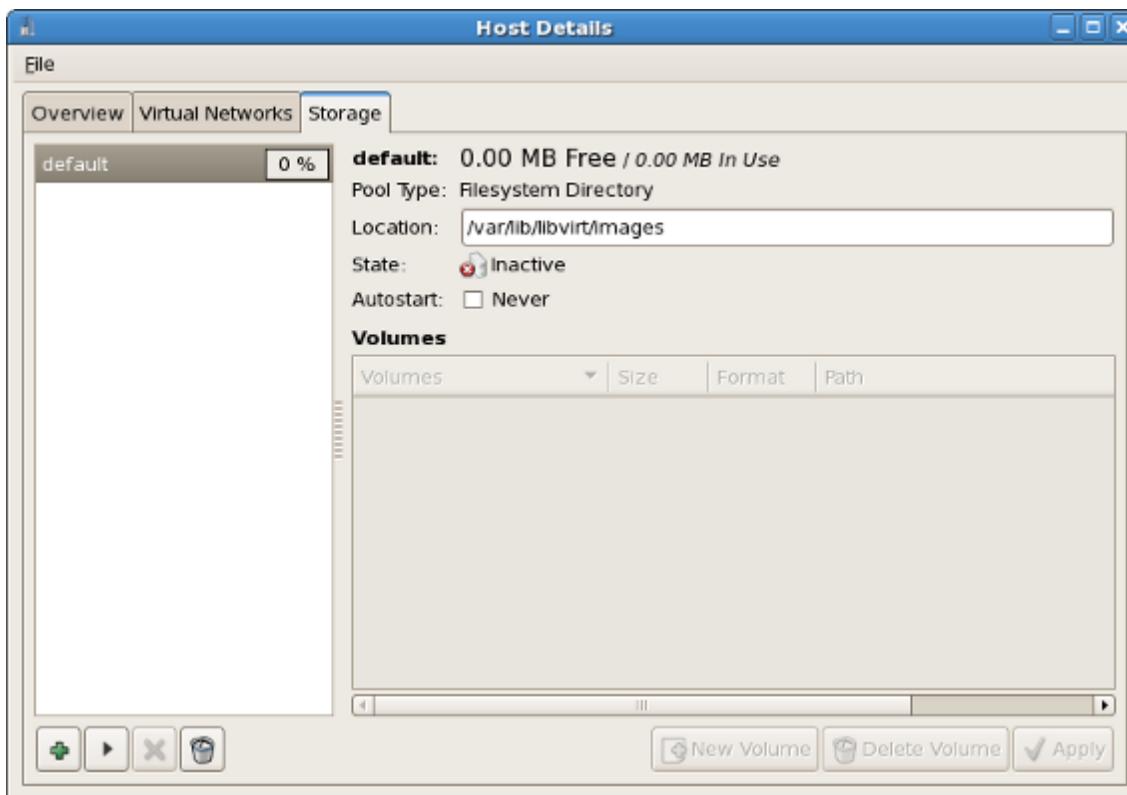
The Virtual Machine Manager displays a list of connected hosts.



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.



- 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**.

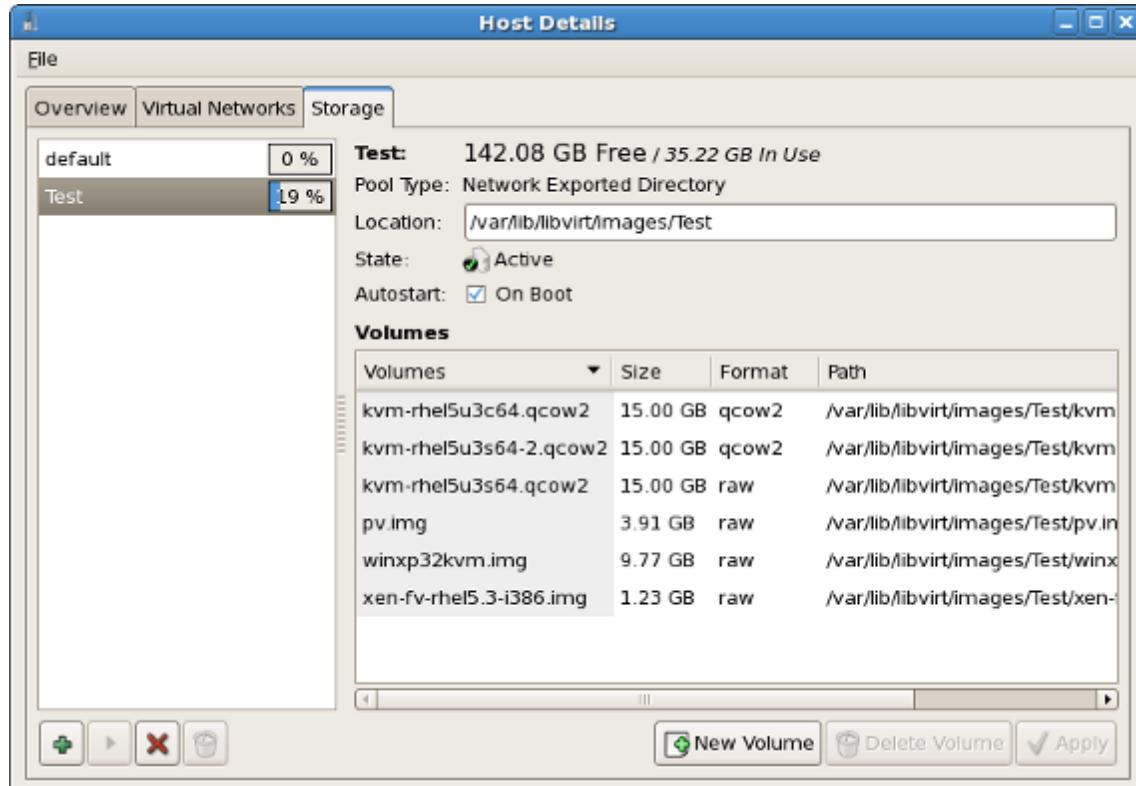
- Enter the following details:

- » **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.

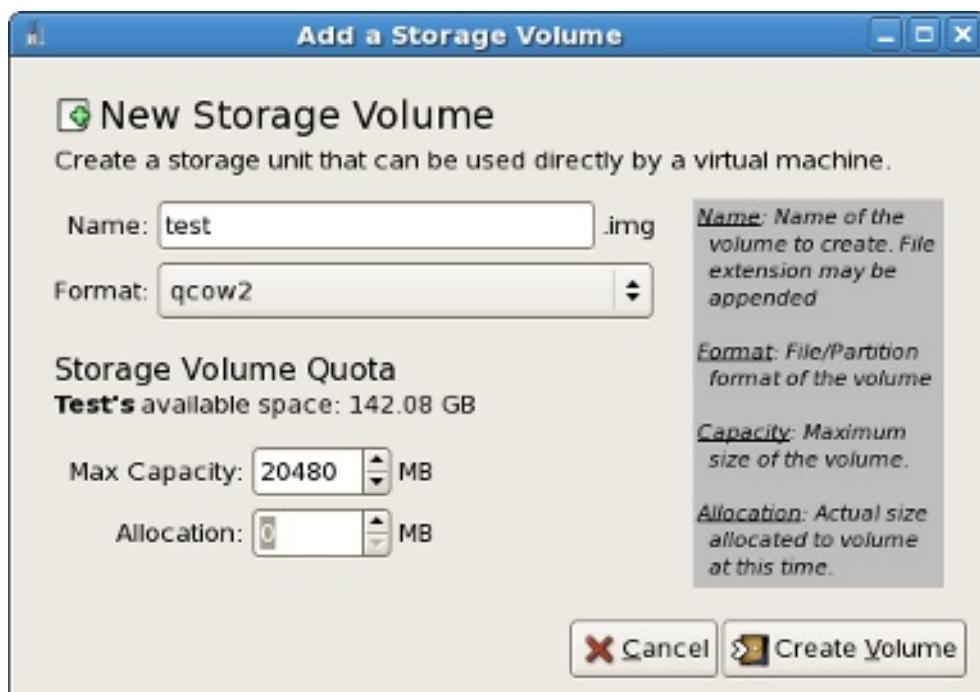


Click **Finish**.

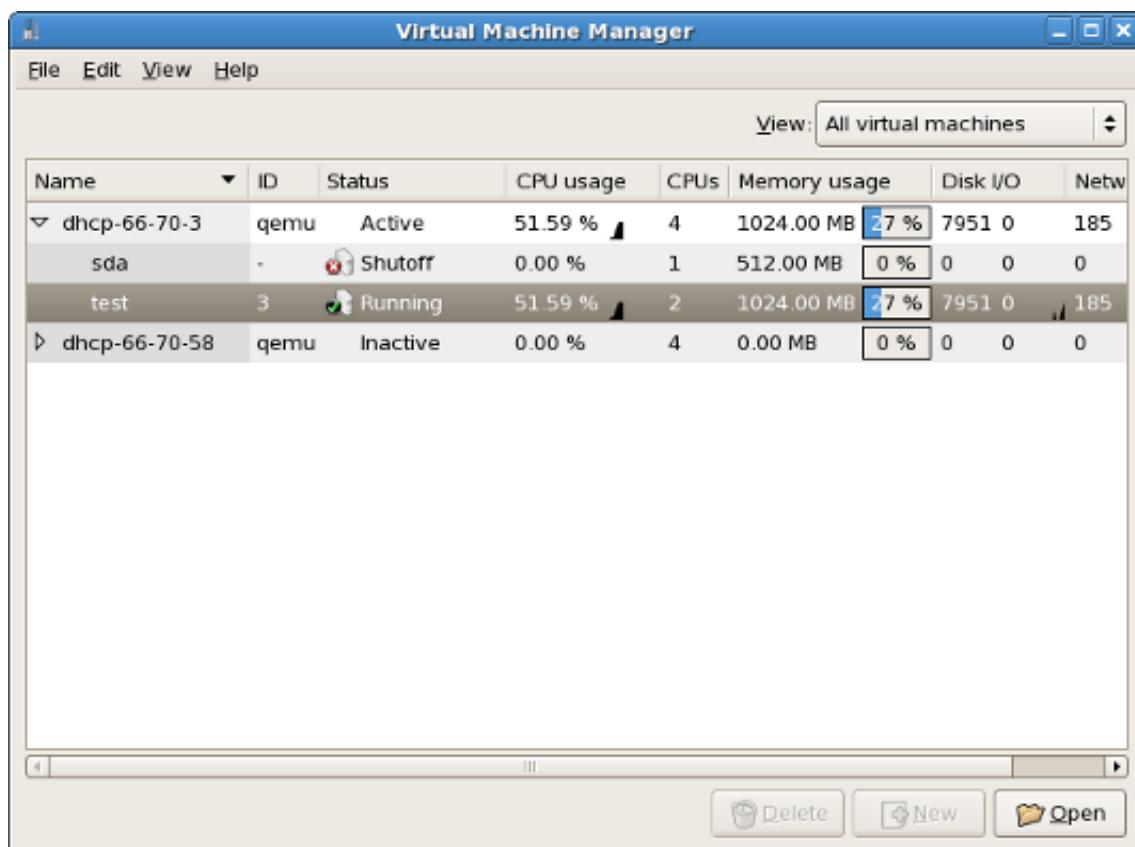
5. Create a new volume in the shared storage pool, click **New Volume**.



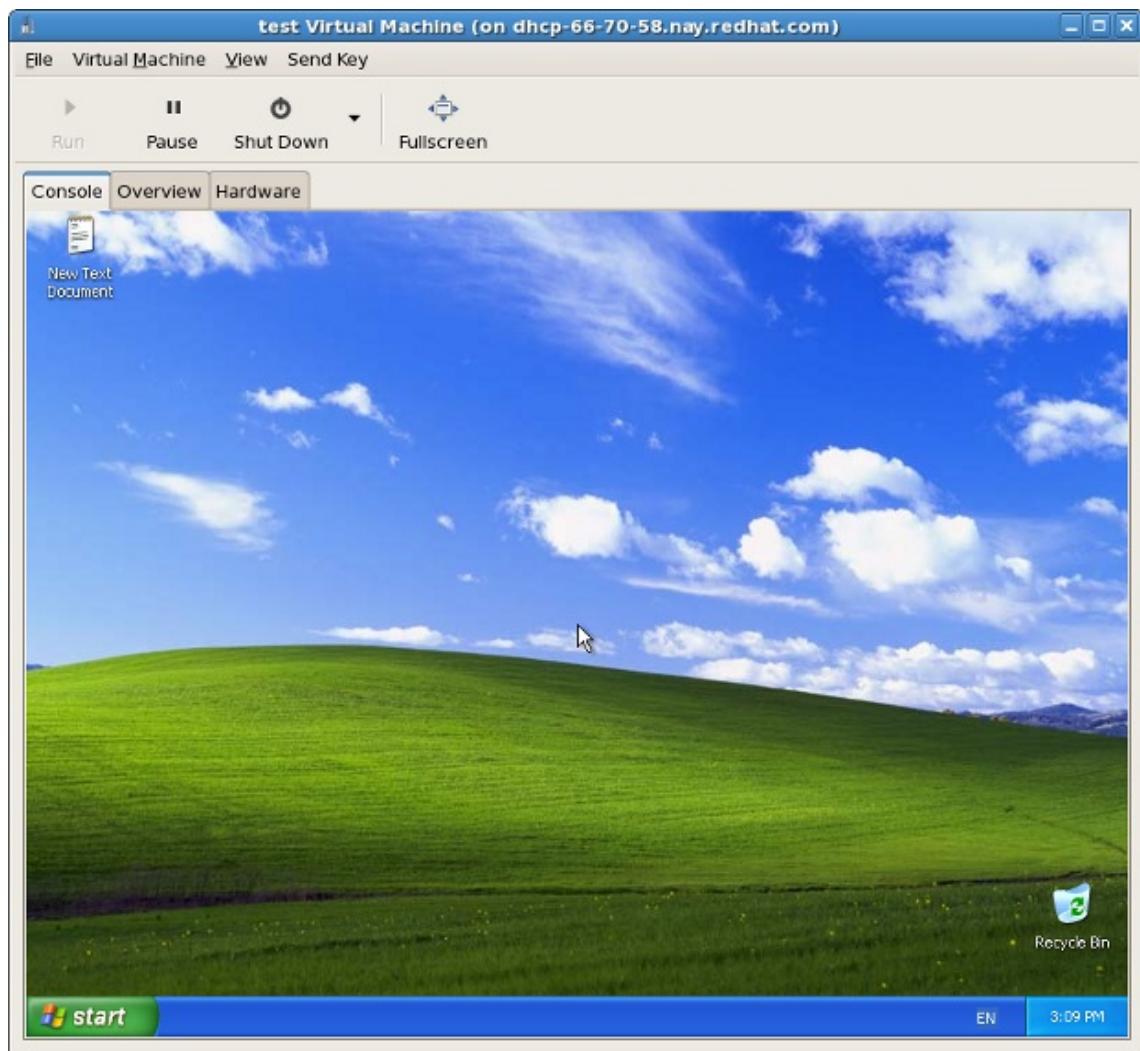
6. Enter the details, then click **Create Volume**.



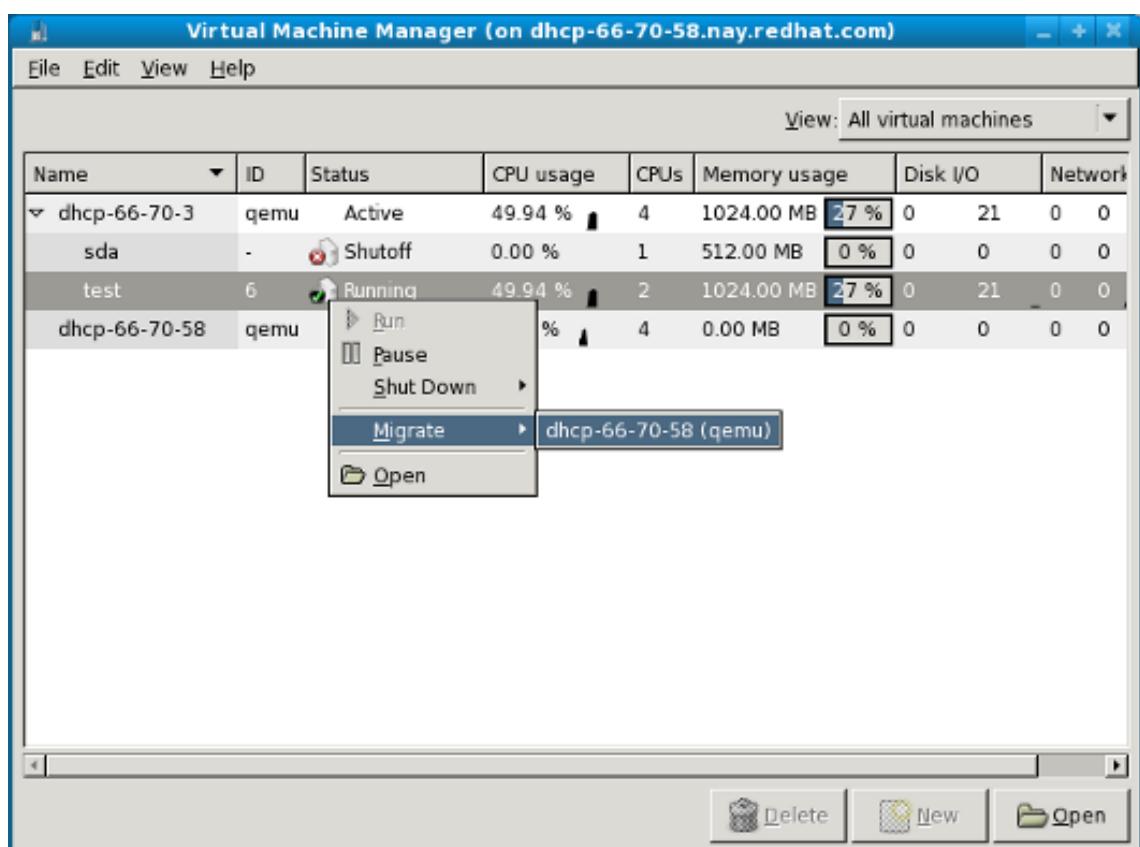
7. Create a virtual machine with the new volume, then run the virtual machine.



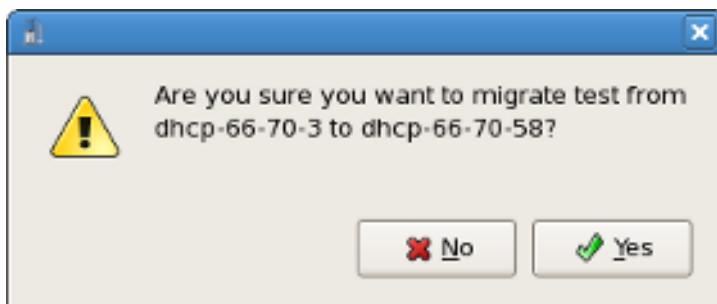
The Virtual Machine window appears.



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.

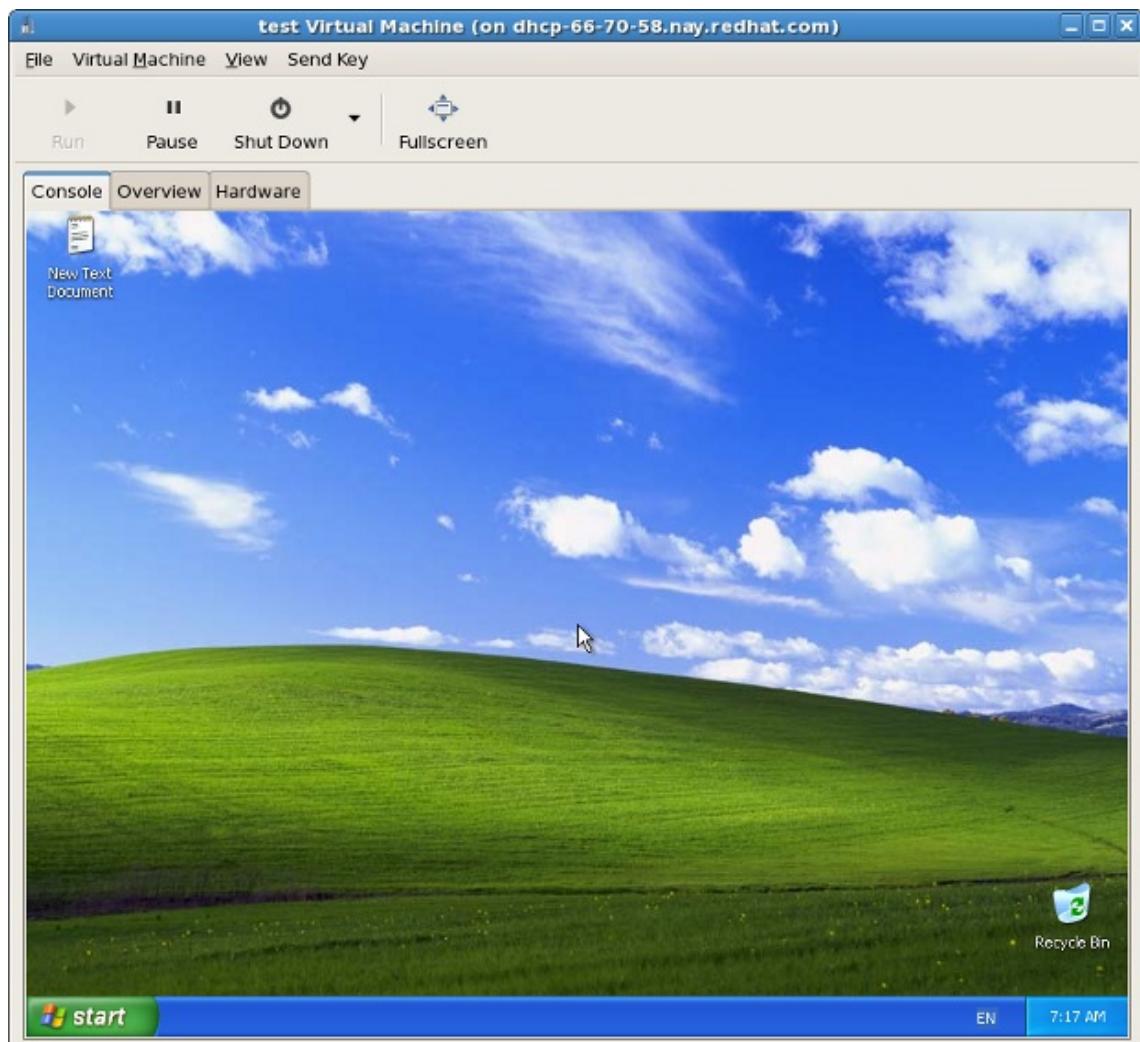


The Virtual Machine Manager displays the virtual machine in its new location.

A screenshot of the Virtual Machine Manager application window. The window title is "Virtual Machine Manager". The menu bar includes "File", "Edit", "View", and "Help". A toolbar at the bottom has three buttons: "Delete" (trash can), "New" (plus sign), and "Open" (document icon). The main area is a table view titled "View: All virtual machines". The table has columns: Name, ID, Status, CPU usage, CPUs, Memory usage, Disk I/O, and Net. Data rows are as follows:

| Name          | ID | Status | CPU usage | CPUs   | Memory usage | Disk I/O   | Net  |   |   |   |
|---------------|----|--------|-----------|--------|--------------|------------|------|---|---|---|
| dhcp-66-70-3  | -  | qemu   | Active    | 0.00 % | 4            | 0.00 MB    | 0 %  | 0 | 0 | 0 |
| sda           | -  | -      | Shutoff   | 0.00 % | 1            | 512.00 MB  | 0 %  | 0 | 0 | 0 |
| test          | -  | -      | Shutoff   | 0.00 % | 2            | 1024.00 MB | 0 %  | 0 | 0 | 0 |
| dhcp-66-70-58 | -  | qemu   | Active    | 0.00 % | 4            | 1024.00 MB | 26 % | 0 | 0 | 8 |
| test          | 2  | -      | Running   | 0.00 % | 2            | 1024.00 MB | 26 % | 0 | 0 | 8 |

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



# Chapter 23. Remote management of guests

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

## 23.1. 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.



#### Important

SSH keys are user dependent. 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.

## 23.2. 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 (see [Section 23.1, “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, see the **libvirt** website, <http://libvirt.org/remote.html>.

#### Xen VNC Server

The Xen VNC server can have TLS enabled by editing the configuration file, **/etc/xen/xend-config.sxp**. Remove the commenting on the (**vnc-tls 1**) configuration parameter in the configuration file.

The **/etc/xen/vnc** directory needs the following 3 files:

- » **ca-cert.pem** - The CA certificate
- » **server-cert.pem** - The Server certificate signed by the CA
- » **server-key.pem** - The server private key

This provides encryption of the data channel. It might be appropriate to require that clients present their own x509 certificate as a form of authentication. To enable this remove the commenting on the (**vnc-x509-verify 1**) parameter.

### **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.
- » **xen://hostname.guestname/** for Xen.

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.

### 23.3. Transport modes

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

#### Transport Layer Security (TLS)

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

#### UNIX sockets

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

#### SSH

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

#### ext

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

#### tcp

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

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

#### Remote URIs

A Uniform Resource Identifier (URI) is used by **virsh** and **libvirt** to connect to a remote host. URLs 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 Xen hypervisor on the host named **towada**, using SSH transport and the SSH username **ccurran**.

```
xen+ssh://ccurran@towada/
```

- » Connect to a remote Xen hypervisor on the host named **towada** using TLS.

```
xen://towada/
```

- » Connect to a remote Xen hypervisor on host **towada** using TLS. The **no\_verify=1** tells libvirt not to verify the server's certificate.

```
xen://towada/?no_verify=1
```

- » Connect to a remote KVM hypervisor on host **towada** using SSH.

```
qemu+ssh://towada/system
```

## 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 23.1, “Extra URI parameters”](#) covers the recognized parameters. All other parameters are ignored. Note that parameter values must be URI-escaped (that is, a question mark (?) is appended before the parameter and special characters are converted into the URI format).

**Table 23.1. Extra URI parameters**

| Name | Transport mode | Description                                                                                                                                                                                                                                                           | Example usage       |
|------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| name | all modes      | The name passed to the remote virConnectOpen function. The name is normally formed by removing transport, hostname, port number, username and extra parameters from the remote URI, but in certain very complex cases it may be better to supply the name explicitly. | name=qemu:///system |

| Name      | Transport mode | Description                                                                                                                                                                                                   | Example usage                                |
|-----------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| command   | ssh and ext    | The external command. For ext transport this is required. For ssh the default is ssh. The PATH is searched for the command.                                                                                   | command=/opt/openssh/bin/ssh                 |
| socket    | unix and ssh   | The path to the UNIX domain socket, which overrides the default. For ssh transport, this is passed to the remote netcat command (see netcat).                                                                 | socket=/opt/libvirt/run/libvirt/libvirt-sock |
| netcat    | ssh            | The <b>netcat</b> command can be used to connect to remote systems. The default netcat parameter uses the <b>nc</b> command. For SSH transport, libvirt constructs an SSH command using the form below:       | netcat=/opt/netcat/bin/nc                    |
|           |                | <i>command</i> -p <i>port</i> [-l <i>username</i> ] <i>hostname</i> <b>netcat</b> -U <i>socket</i>                                                                                                            |                                              |
|           |                | The <i>port</i> , <i>username</i> and <i>hostname</i> parameters can be specified as part of the remote URI. The <b>command</b> , <b>netcat</b> and <b>socket</b> come from other extra parameters.           |                                              |
| no_verify | tls            | If set to a non-zero value, this disables client checks of the server's certificate. Note that to disable server checks of the client's certificate or IP address you must change the libvirtd configuration. | no_verify=1                                  |

| Name   | Transport mode | Description                                                                                                                                                                                                                                                               | Example usage |
|--------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| no_tty | ssh            | If set to a non-zero value, this stops ssh from asking for a password if it cannot log in to the remote machine automatically (for using ssh-agent or similar). Use this when you do not have access to a terminal - for example in graphical programs which use libvirt. | no_tty=1      |

## Part V. Virtualization Storage Topics

### Introduction to storage administration for virtualization

This part covers using shared, networked storage with virtualization on Red Hat Enterprise Linux.

The following methods are supported for virtualization:

- » Fibre Channel
- » iSCSI
- » NFS
- » GFS2

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

# Chapter 24. Using shared storage with virtual disk images

This chapter covers the use of shared and network storage devices for virtual disks.

## 24.1. Using iSCSI for storing virtual disk images

This section demonstrates how to set up an iSCSI target on Red Hat Enterprise Linux and how to configure iSCSI on a libvirt KVM host using virsh, and finally how to provision a guest on iSCSI using `virt-install`.



### Important

Setting up a Red Hat Enterprise Linux server as an iSCSI target is **not** recommended. The example used in this section should not be used in production, and is provided as an example which should only be referred to for basic shared storage testing and educational purposes.

### 24.1.1. How to set up an iSCSI target on Red Hat Enterprise Linux

#### 1. Install and enable the iSCSI target service

Install and enable the iSCSI target service with the following commands:

```
yum install scsi-target-utils
chkconfig tgtd on
service tgtd start
```



### Important

The `scsi-target-utils` package required for this example is provided only in the Cluster Storage add-on for Red Hat Enterprise Linux 5, and may not be available for your system. Contact your support agent to activate this add-on if you are currently unable to install this package.

#### 2. Allocate storage for the LUNs

The iSCSI target service is not dependent on a particular type of exported LUN. The LUNs can be plain files, LVM volumes, or block devices. There is however a performance overhead if using the LVM and/or file system layers as compared to block devices. The guest providing the iSCSI service in this example has no spare block or LVM devices, so raw files will be used.

The following commands demonstrate the creation of two LUNs; one is 10GB (sparse file), the other is 500MB (fully-allocated). Be aware that your environment will be different and this is provided only as an example:

```
mkdir -p /var/lib/tgtd/kvmguest
dd if=/dev/zero of=/var/lib/tgtd/kvmguest/rhelx86_64.img bs=1M seek=10240
count=0
dd if=/dev/zero of=/var/lib/tgtd/kvmguest/shareddata.img bs=1M count=512
restorecon -R /var/lib/tgtd
```

### 3. Export the iSCSI target and LUNs

For Red Hat Enterprise Linux 5, several **tgtadm** commands are required to create a target and associate the storage volumes created earlier. First, the following command adds a target using an iSCSI Qualified Name (IQN):

```
tgtadm --lld iscsi --op new --mode target --tid 1 --targetname \
iqn.2004-04.rhel:rhel5:iscsi.kvmguest
```

Now the storage volumes must be associated with LUNs in the iSCSI target with these two commands:

```
tgtadm --lld iscsi --op new --mode logicalunit --tid 1 --lun 1 \
--backing-store /var/lib/tgtd/kvmguest/rhelx86_64.img

tgtadm --lld iscsi --op new --mode logicalunit --tid 1 --lun 2 \
--backing-store /var/lib/tgtd/kvmguest/shareddata.img
```



#### Important

Add the previous 3 **tgtadm** commands to the end of the **/etc/rc.local** file to ensure normal operation upon restarting of the system.

To confirm the successful operation of the previous commands, query the iSCSI target setup:

```
tgtadm --lld iscsi --op show --mode target
Target 1: iqn.2004-04.rhel:rhel5:iscsi.kvmguest
 System information:
 Driver: iscsi
 State: ready
 I_T nexus information:
 LUN information:
 LUN: 0
 Type: controller
 SCSI ID: IET 00010000
 SCSI SN: beaf10
 Size: 0 MB, Block size: 1
 Online: Yes
 Removable media: No
 Readonly: No
 Backing store type: null
 Backing store path: None
 Backing store flags:
 LUN: 1
 Type: disk
 SCSI ID: IET 00010001
 SCSI SN: beaf11
 Size: 10737 MB, Block size: 1024
 Online: Yes
 Removable media: No
 Readonly: No
 Backing store type: rdwr
 Backing store path: /var/lib/tgtd/kvmguest/rhelx86_64.img
 Backing store flags:
 LUN: 2
 Type: disk
 SCSI ID: IET 00010002
 SCSI SN: beaf12
 Size: 537 MB, Block size: 512
 Online: Yes
```

```

Removable media: No
 Readonly: No
 Backing store type: rdwr
 Backing store path: /var/lib/tgtd/kvmguest/shareddata.img
 Backing store flags:
 Account information:
 ACL information:

```

#### 4. Allow client access to the target

Finally, this example command allows access to all clients without any authentication:

```
tgtadm --lld iscsi --op bind --mode target --tid 1 --initiator-address ALL
```

#### Note

The two most common problems encountered when configuring the iSCSI target are related to SELinux and iptables. If adding plain files as LUNs in an iSCSI target, ensure the files are labeled **system\_u:object\_r:tgtd\_var\_lib\_t:s0**. TCP port 3260 must be open in your iptables configuration.

### 24.1.2. How to configure iSCSI on a libvirt KVM host and provision a guest using virt-install

The previous section described how to set up an iSCSI target. This section demonstrates how to configure iSCSI on a libvirt KVM instance using virsh, and then how to provision a guest using **virt-install**.

#### 1. Defining the storage pool

All libvirt storage is managed through *storage pools*, of which there are many possible types: SCSI, NFS, ext4, plain directory and iSCSI. All libvirt objects are configured via XML description files, and storage pools are no different in this regard. For an iSCSI storage pool there are three important pieces of information to provide:

- ✖ The *target path* - this determines how libvirt will expose device paths for the pool. Paths like **/dev/sda** and **/dev/sdb** are not an ideal choice as they can change between reboots, and can change across machines within a cluster; in other words, the names are assigned on a first come, first served basis by the kernel. It is strongly recommended to use the **/dev/disk/by-path** format. This results in a consistent naming scheme across all machines.
- ✖ The *host name* - this is the fully-qualified DNS name of the iSCSI server.
- ✖ The *source path* - this is the iSCSI qualified name (IQN) seen in the previous setup procedure (`iqn.2004-04.rhel5:iscsi.kvmguest`).

Although your environment will likely be different, the following text is what an iSCSI configuration will look like:

```

<pool type='iscsi'>
 <name>kvmguest</name>
 <source>
 <host name='myiscsi.example.com' />
 <device path='iqn.2004-04.rhel5:iscsi.kvmguest' />
 </source>
</pool>

```

```

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

```

Save this XML code to a file named **iscsirhel5.xml** and load it into libvirt using the **pool-define** command:

```

virsh pool-define iscsirhel5.xml
Pool kvmguest defined from iscsirhel5.xml

virsh pool-list --all
Name State Autostart

default active yes
kvmguest inactive no

```

## 2. Starting the storage pool

The configuration is saved, but the iSCSI target has not yet been logged into, so no LUNs will be visible on the host at this point. Use the **pool-start** command to make the LUNs visible with the **vol-list** command.

```

virsh pool-start kvmguest
Pool kvmguest started

virsh vol-list kvmguest
Name Path

10.0.0.1 /dev/disk/by-path/ip-192.168.122.2:3260-iscsi-iqn.2004-
04.rhel5:iscsi.kvmguest-lun-1
10.0.0.2 /dev/disk/by-path/ip-192.168.122.2:3260-iscsi-iqn.2004-
04.rhel5:iscsi.kvmguest-lun-2

```

## 3. Querying LUN information

Further information about each LUN can be obtained using the **vol-info** and **vol-dumpxml** commands:

```

virsh vol-info /dev/disk/by-path/ip-192.168.122.2:3260-iscsi-iqn.2004-
04.rhel5:iscsi.kvmguest-lun-1

Name: 10.0.0.1
Type: block
Capacity: 10.00 GB
Allocation: 10.00 GB

virsh vol-dumpxml /dev/disk/by-path/ip-192.168.122.2:3260-iscsi-iqn.2004-
04.rhel5:iscsi.kvmguest-lun-1
<volume>
 <name>10.0.0.1</name>
 <key>/dev/disk/by-path/ip-192.168.122.2:3260-iscsi-iqn.2004-
04.rhel5:iscsi.kvmguest-lun-1</key>
 <source>
 </source>
 <capacity>10737418240</capacity>
 <allocation>10737418240</allocation>
 <target>
 <path>/dev/disk/by-path/ip-192.168.122.2:3260-iscsi-iqn.2004-

```

```

04.rhel:rhel5:iscsi.kvmguest-lun-2</path>
 <format type='unknown' />
 <permissions>
 <mode>0660</mode>
 <owner>0</owner>
 <group>6</group>
 <label>system_u:object_r:fixed_disk_device_t:s0</label>
 </permissions>
</target>
</volume>

```

#### 4. Activating the storage at boot time

Once correctly configured, the pool can be set to start automatically upon booting of the host:

```

virsh pool-autostart kvmguest
Pool kvmguest marked as autostarted

```

#### 5. Provisioning a guest on iSCSI

The **virt-install** command can be used to install new guests from the command line. The **--disk** argument can take the name of a storage pool, followed by the name of any contained volumes. Continuing this example, the following command will begin the installation of a guest with two disks; the first disk is the root file system, the second disk can be shared between multiple guests for common data:

```

virt-install --accelerate --name rhelx86_64 --ram 800 --vnc --disk \
vol=kvmguest/10.0.0.1 --disk vol=kvmguest/10.0.0.2,perms=sh --pxe

```

Once this **rhelx86\_64** guest is installed, the following command and output shows the XML that **virt-install** used to associate the guest with the iSCSI LUNs:

```

virsh dumpxml rhelx86_64

<domain type='kvm' id='4'>
 <name>rhelx86_64</name>
 <uuid>ad8961e9-156f-746f-5a4e-f220bfaf08d</uuid>
 <memory>819200</memory>
 <currentMemory>819200</currentMemory>
 <vcpu>1</vcpu>
 <os>
 <type arch='x86_64' machine='rhel'>hvm</type>
 <boot dev='network' />
 </os>
 <features>
 <acpi/>
 <apic/>
 <pae/>
 </features>
 <clock offset='utc' />
 <on_poweroff>destroy</on_poweroff>
 <on_reboot>destroy</on_reboot>
 <on_crash>destroy</on_crash>
 <devices>
 <emulator>/usr/libexec/qemu-kvm</emulator>
 <disk type='block' device='disk'>
 <driver name='qemu' type='raw' />
 <source dev='/dev/disk/by-path/ip-192.168.122.170:3260-iscsi-iqn.2004-
04.rhel:rhel5:iscsi.kvmguest-lun-1' />
 <target dev='hda' bus='ide' />
 <alias name='ide0-0-0' />
 </disk>
 </devices>

```

```

<address type='drive' controller='0' bus='0' unit='0' />
</disk>
<disk type='block' device='disk'>
 <driver name='qemu' type='raw' />
 <source dev='/dev/disk/by-path/ip-192.168.122.170:3260-iscsi-iqn.2004-
04.rhel:rhel5:iscsi.kvmguest-lun-2' />
 <target dev='hdb' bus='ide' />
 <shareable/>
 <alias name='ide0-0-1' />
 <address type='drive' controller='0' bus='0' unit='1' />
</disk>
<controller type='ide' index='0'>
 <alias name='ide0' />
 <address type='pci' domain='0x0000' bus='0x00' slot='0x01'
function='0x1' />
</controller>
<interface type='network'>
 <mac address='52:54:00:0a:ca:84' />
 <source network='default' />
 <target dev='vnet1' />
 <alias name='net0' />
 <address type='pci' domain='0x0000' bus='0x00' slot='0x04'
function='0x0' />
</interface>
<serial type='pty'>
 <source path='/dev/pts/28' />
 <target port='0' />
 <alias name='serial0' />
</serial>
<console type='pty' tty='/dev/pts/28'>
 <source path='/dev/pts/28' />
 <target port='0' />
 <alias name='serial0' />
</console>
<input type='mouse' bus='ps2' />
<graphics type='vnc' port='5901' autoport='yes' keymap='en-us' />
<video>
 <model type='cirrus' vram='9216' heads='1' />
 <alias name='video0' />
 <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0' />
</video>
</devices>
</domain>
```

There are two important items of note in this output:

- ⌘ The guest uses the large **/dev/disk/by-path** paths to reference the LUNs. As described earlier, this is so that file names and paths will remain constant.
- ⌘ The second disk has the **<shareable/>** flag set. Critical for the disk to be safely shared between guests, this ensures that the SELinux labelling will be appropriate for multiple guests to access the disk and that all I/O caching is disabled on the host.

For migration of guests between hosts to succeed, some form of shared storage is required. Although NFS is often used for this purpose, the lack of SELinux labelling for NFS means there is limited sVirt protection between guests. This lack of sVirt support could allow one guest to use another guest's disks, which is usually undesirable.

Using iSCSI provides full sVirt isolation between guests to the same degree of non-shared storage.

## 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 Red Hat Enterprise Linux. These chapters are designed for users requiring information on advanced functionality and other features.

## Chapter 25. Virtualization tools

The following is a list of tools for virtualization administration, debugging and networking tools that are useful for systems running Xen.

### System Administration Tools

- » **vmstat**

- » **iostat**

- » **lsof**

```
lsof -i :5900
xen-vncfb 10635 root 5u IPv4 218738 TCP grumble.boston.redhat.com:5900
(LISTEN)
```

- » **qemu-img**

### Advanced Debugging Tools

- » **systemTap**

- » **crash**

- » **xen-gdbserver**

- » **sysrq**

- » **sysrq t**

- » **sysrq w**

- » **sysrq c**

### Networking

#### **brctl**

- » 

```
brctl show
bridge name bridge id STP enabled interfaces
xenbr0 8000.ffffffffffff no vif13.0
 pdummy0
 vif0.0
```

- » 

```
brctl showmacs xenbr0
port no mac addr is local? aging timer
 1 fe:ff:ff:ff:ff:ff yes 0.00
```

- » 

```
brctl showstp xenbr0
xenbr0
bridge id 8000.ffffffffffff
designated root 8000.ffffffffffff
root port 0
max age 20.00
 20.00
hello time 2.00
forward delay 0.00
aging time 300.01
hello timer 1.43
 path cost 0
 bridge max age
 bridge hello time 2.00
 bridge forward delay 0.00
 tcn timer 0.00
```

topology change timer 0.00	gc timer	0.02
flags		
vif13.0 (3)		
port id 8003	state	
forwarding		
designated root 8000.feffffffffff	path cost	100
designated bridge 8000.feffffffffff	message age timer	0.00
designated port 8003	forward delay timer	0.00
designated cost 0	hold timer	0.43
flags		
pdummy0 (2)		
port id 8002	state	
forwarding		
designated root 8000.feffffffffff	path cost	100
designated bridge 8000.feffffffffff	message age timer	0.00
designated port 8002	forward delay timer	0.00
designated cost 0	hold timer	0.43
flags		
vif0.0 (1)		
port id 8001	state	
forwarding		
designated root 8000.feffffffffff	path cost	100
designated bridge 8000.feffffffffff	message age timer	0.00
designated port 8001	forward delay timer	0.00
designated cost 0	hold timer	0.43
flags		

» **ifconfig**

» **tcpdump**

## KVM tools

- » **ps**
- » **pstree**
- » **top**
- » **kvmtrace**
- » **kvm\_stat**

## Xen tools

- » **xentop**
- » **xm dmesg**
- » **xm log**

## Chapter 26. 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.

**Table 26.1. Guest management commands**

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

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

**Table 26.2. Resource management options**

Command	Description
<b>setmem</b>	Sets the allocated memory for a guest.
<b>setmaxmem</b>	Sets maximum memory limit for the hypervisor.
<b>setvcpus</b>	Changes number of virtual CPUs assigned to a guest. Note that this feature is unsupported in Red Hat Enterprise Linux 5.
<b>vcpuinfo</b>	Displays virtual CPU information about a guest.

Command	Description
<b>vcpuin</b>	Controls the virtual CPU affinity of a guest.
<b>domblkstat</b>	Displays block device statistics for a running guest.
<b>domifstat</b>	Displays network interface statistics for a running guest.
<b>attach-device</b>	Attach a device to a guest, using a device definition in an XML file.
<b>attach-disk</b>	Attaches a new disk device to a guest.
<b>attach-interface</b>	Attaches a new network interface to a guest.
<b>detach-device</b>	Detach a device from a guest, takes the same kind of XML descriptions as command <b>attach-device</b> .
<b>detach-disk</b>	Detach a disk device from a guest.
<b>detach-interface</b>	Detach a network interface from a guest.
<b>domxml-from-native</b>	Convert from native guest configuration format to domain XML format. See the <b>virsh</b> man page for more details.
<b>domxml-to-native</b>	Convert from domain XML format to native guest configuration format. See the <b>virsh</b> man page for more details.

These are miscellaneous **virsh** options:

**Table 26.3. Miscellaneous options**

Command	Description
<b>version</b>	Displays the version of <b>virsh</b>
<b>nodeinfo</b>	Outputs information about the hypervisor

### 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 {domain-id, domain-name or domain-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 (see [Editing a guest's configuration file](#). You can edit this XML configuration file to configure additional devices or to deploy additional guests. See [Section 34.1, “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='xen' 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='xenbr0' />
 <mac address='00:16:3e:49:1d:11' />
 <script path='vif-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 (see [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 (see [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

Shutdown 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([Shutdown 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 the 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.



### Important

This feature is unsupported and known not to work in Red Hat Enterprise Linux 5.

## 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:

- » the same version of Red Hat Enterprise Linux,
- » the same hypervisor (KVM or Xen), 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:

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.

# Chapter 27. 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 guests,
- » links to the textual and graphical consoles, and
- » live and offline migrations.

## 27.1. 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 **Local Xen host** option or QEMU (for KVM).

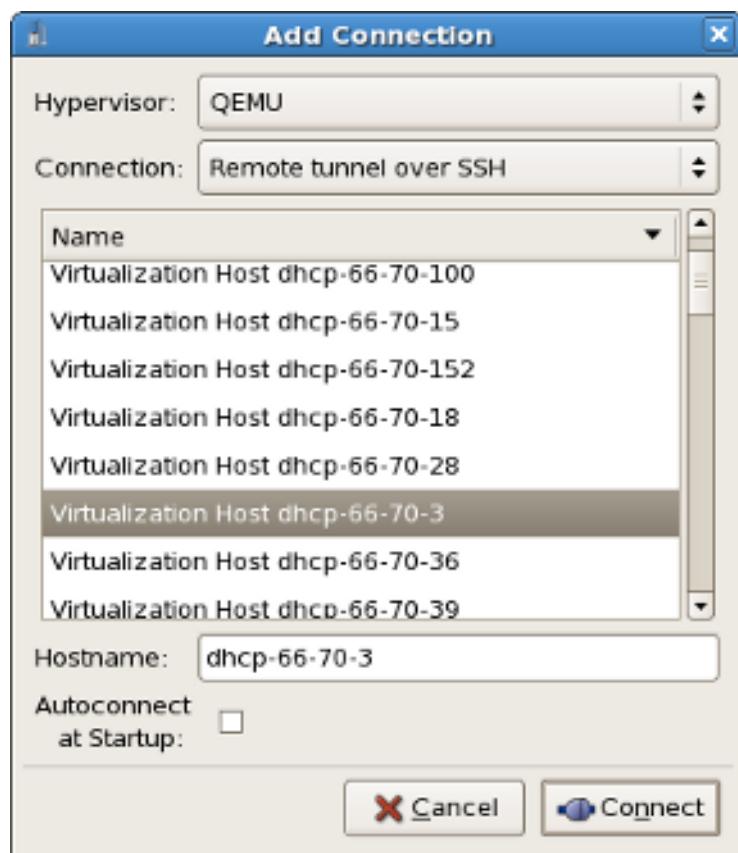


Figure 27.1. Virtual Machine Manager connection window

## 27.2. The Virtual Machine Manager main window

This main window displays all the running guests and resources used by guests. Select a guest by double clicking the guest's name.

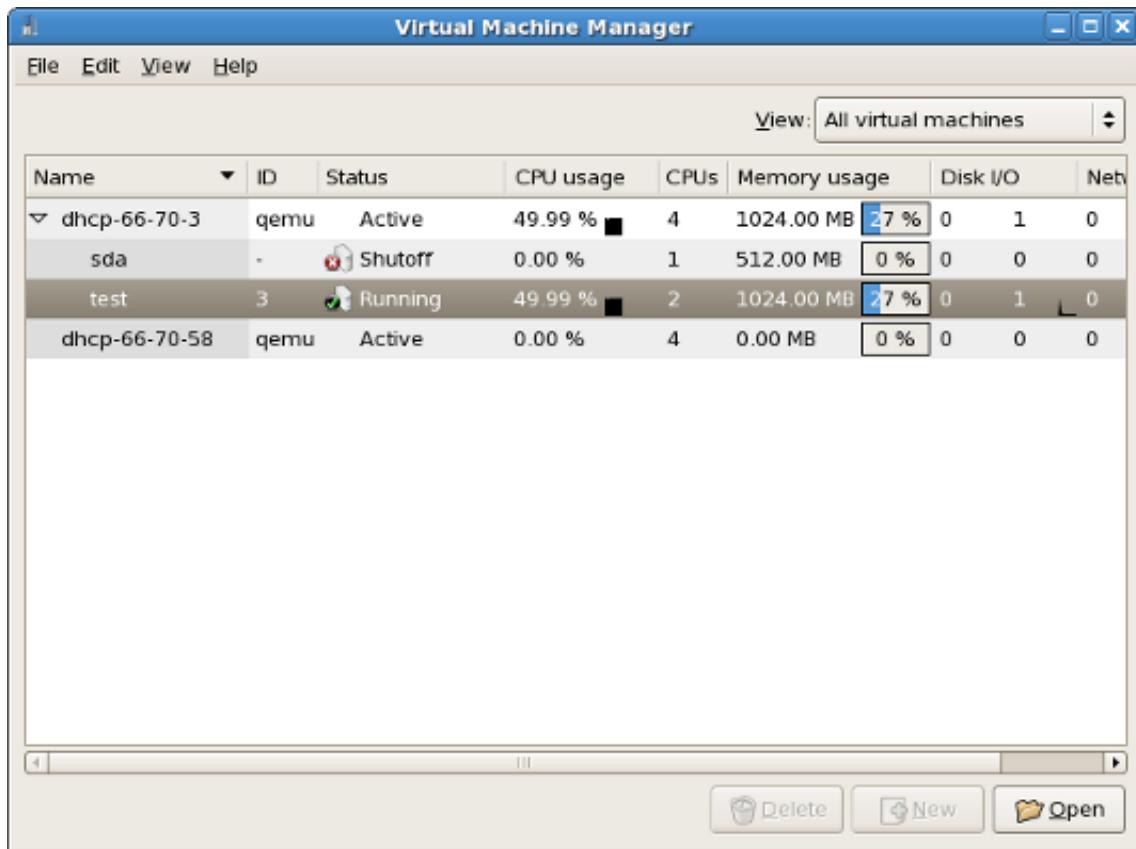


Figure 27.2. Virtual Machine Manager main window

## 27.3. 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.

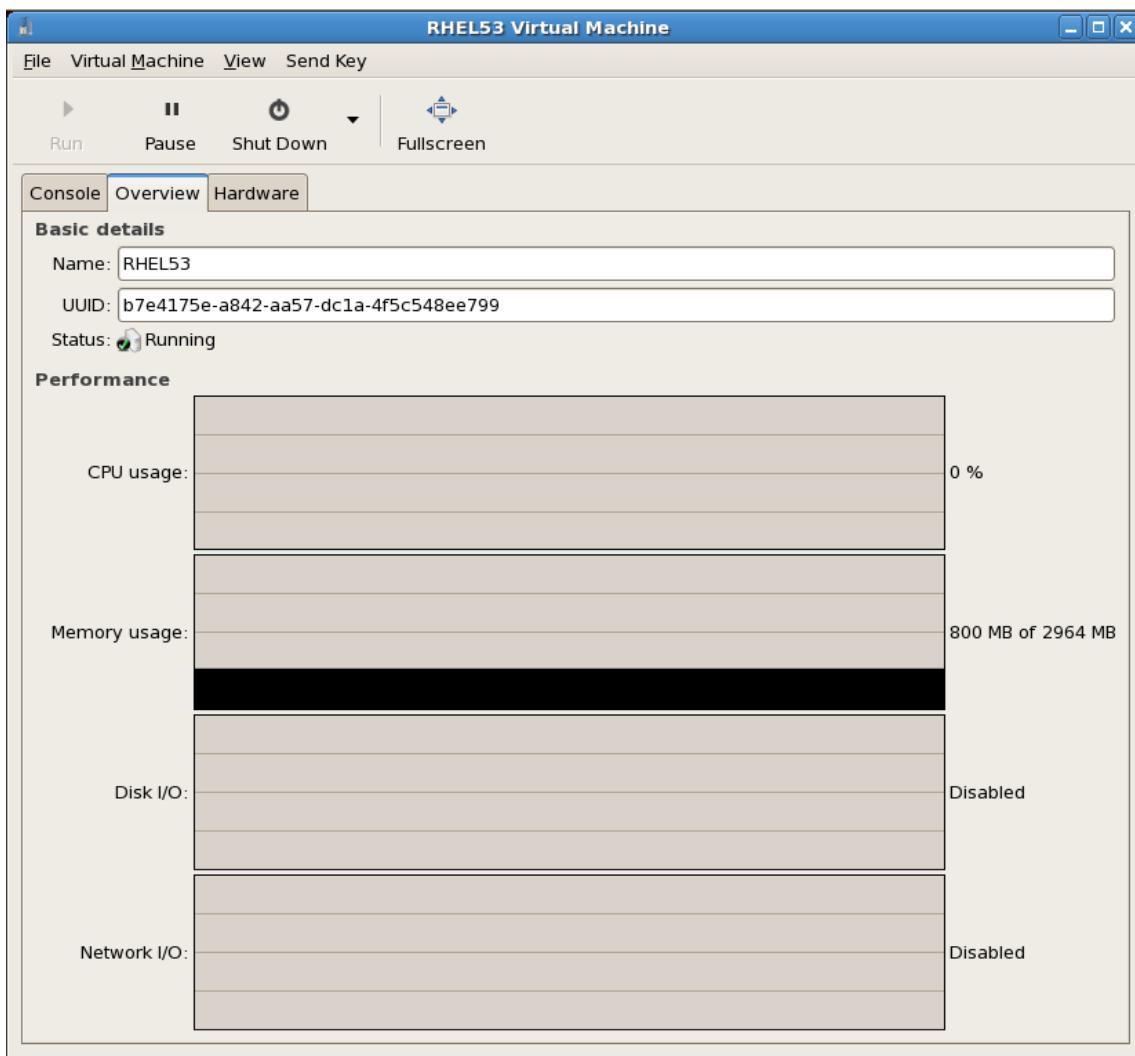
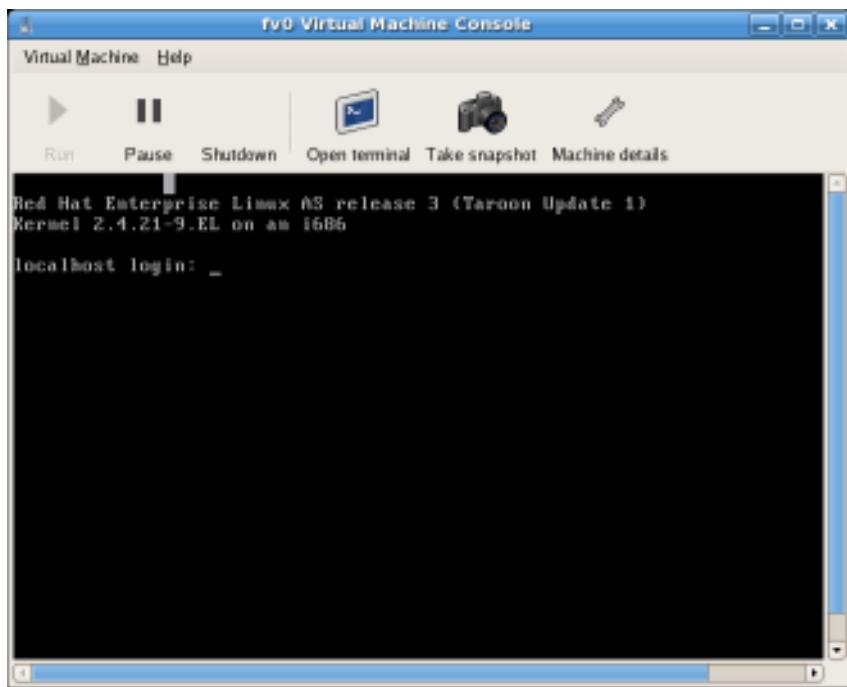


Figure 27.3. The Overview tab

## 27.4. 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 27.4. Graphical console window**

### Note

VNC is considered insecure by many security experts, however, several changes have been made to enable the secure usage of VNC for virtualization on Red Hat enterprise Linux. The guest machines only listen to the local host (**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 23, Remote management of guests](#). TLS can provide enterprise level security for managing guest and host systems.

Your local desktop can intercept key combinations (for example, Ctrl+Alt+F11) to prevent them from being sent to the guest machine. You can use the **virt-manager** sticky key capability to send these sequences. To use this capability, 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. You can then send Ctrl-Alt-F11 to the guest by entering the key sequence 'Ctrl Ctrl Ctrl Alt+F1'.

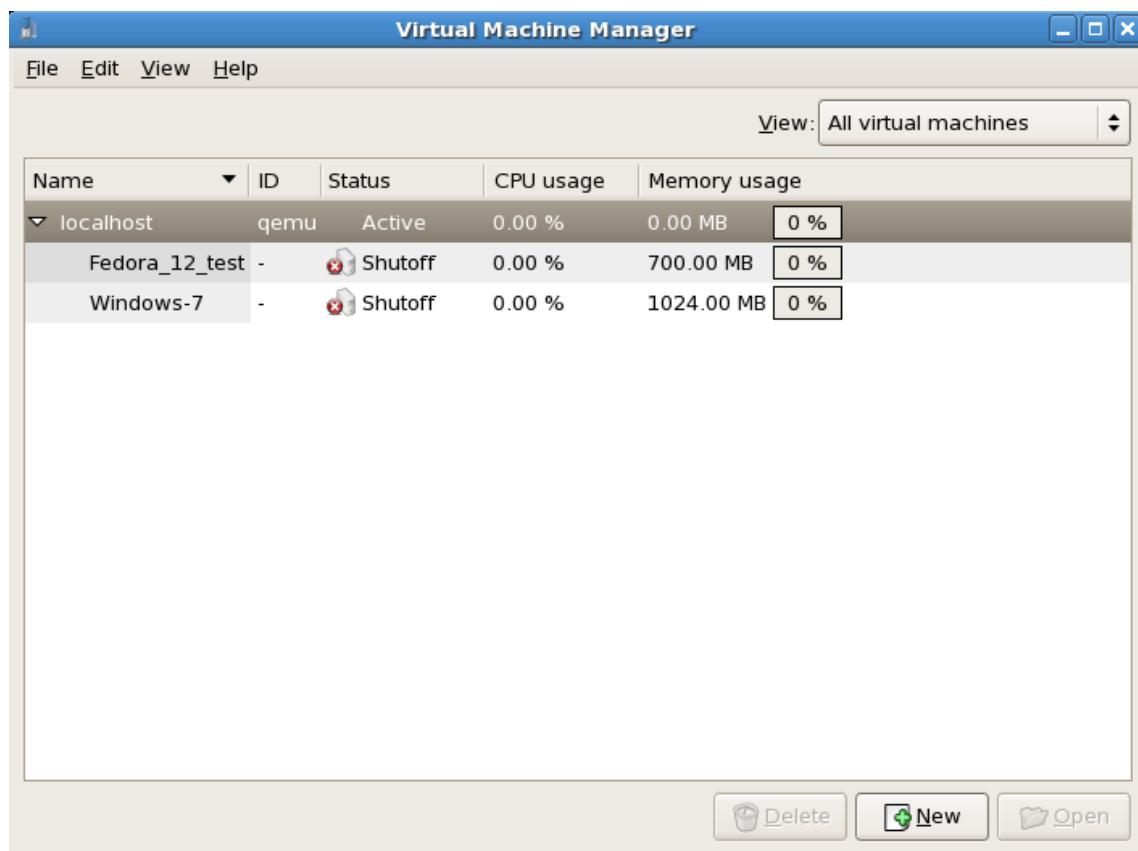
### Note

Due to a limitation of **virt-manager**, it is not possible to use this *sticky key* feature to send a **Sysrq** key combination to a guest.

## 27.5. 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 27.5. 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
```

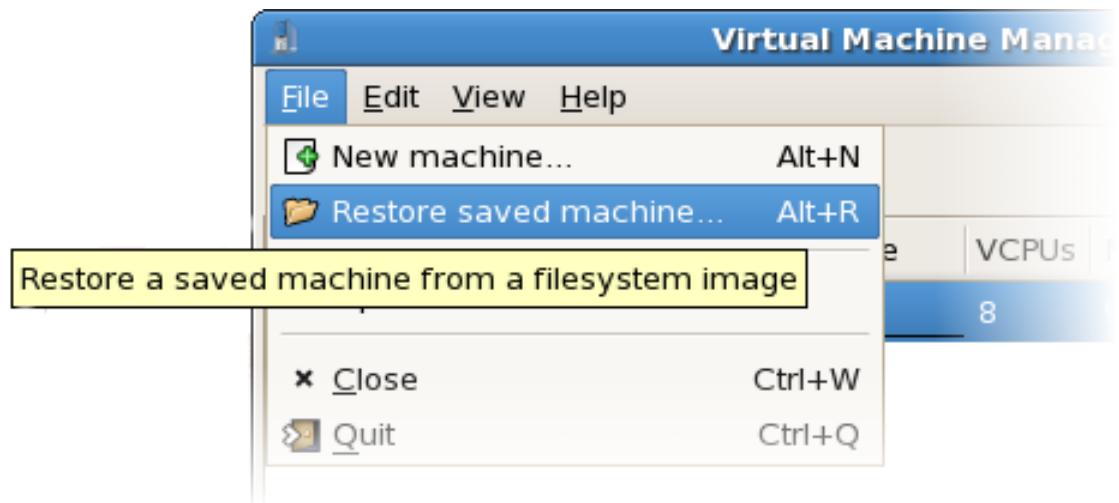
Using **ssh** to manage virtual machines and hosts is discussed further in [Section 23.1, “Remote management with SSH”](#).

## 27.6. 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:

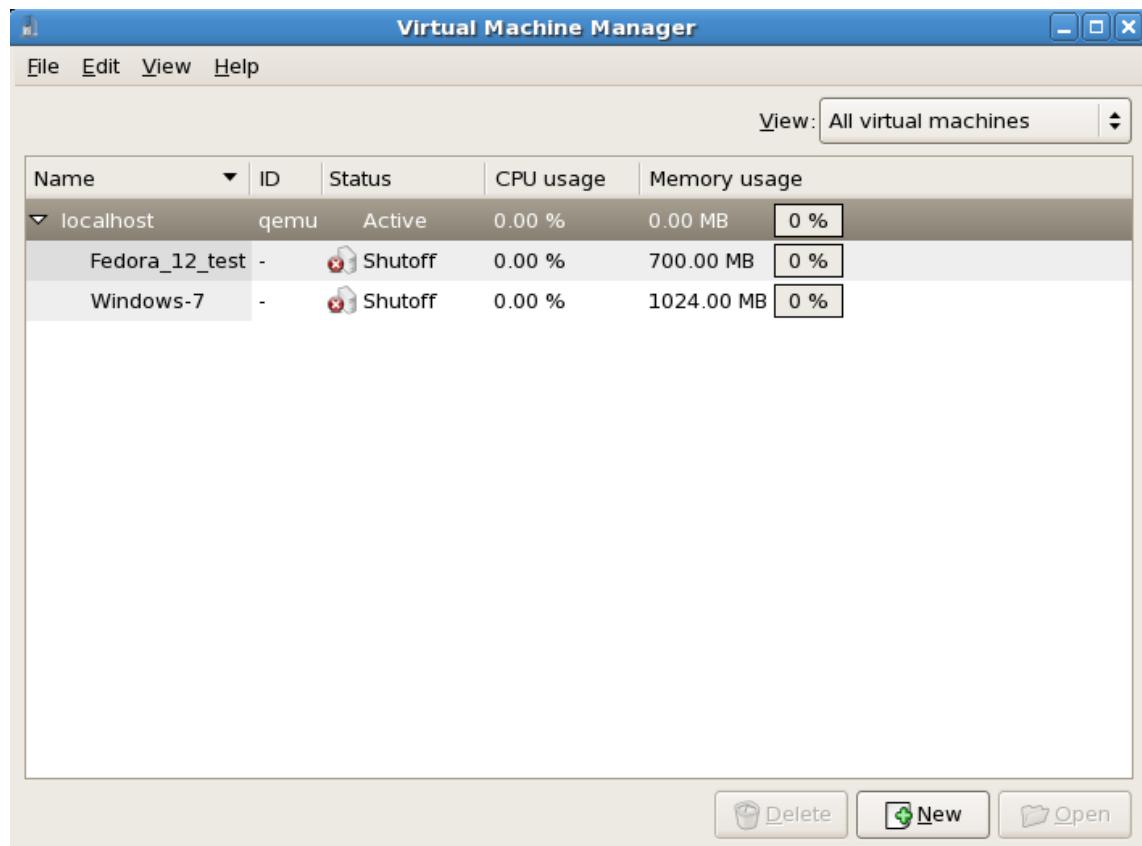
1. From the **File** menu, select **Restore a saved machine**.



**Figure 27.6. 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.



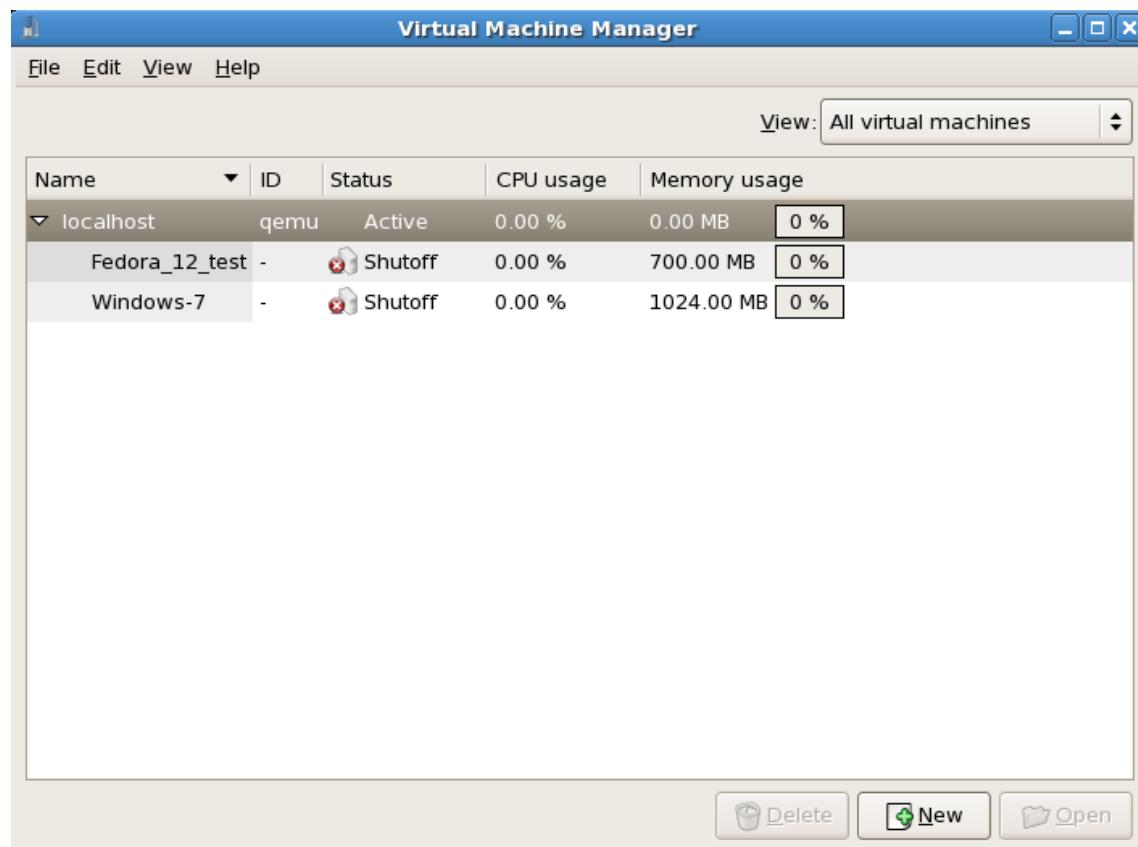
**Figure 27.7. A restored virtual machine manager session**

## 27.7. Displaying guest details

You can use the Virtual Machine Monitor to view activity data information for any virtual machines on your system.

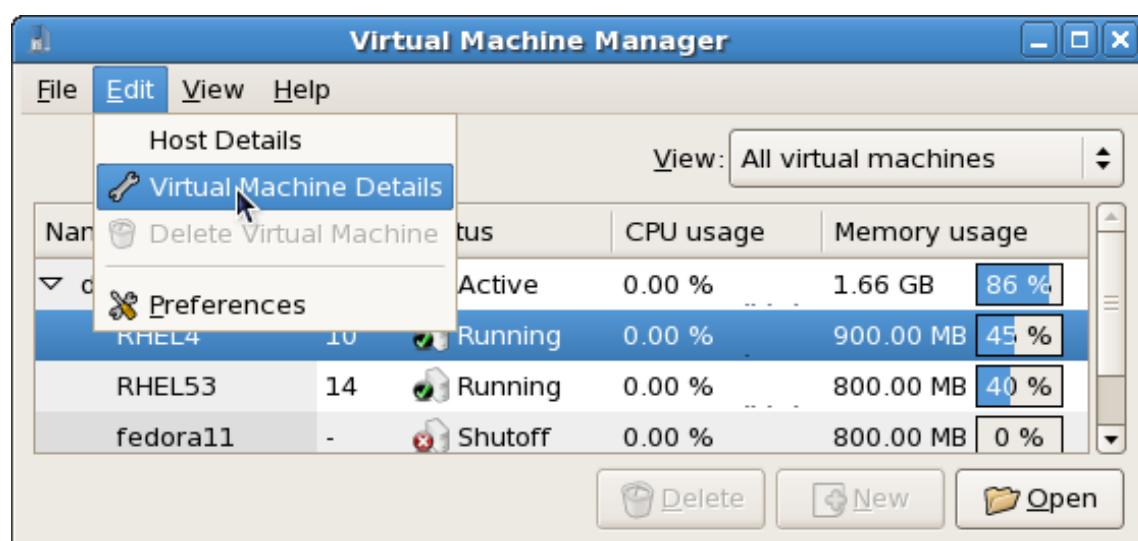
To view a virtual system's details:

1. In the Virtual Machine Manager main window, highlight the virtual machine that you want to view.



**Figure 27.8. Selecting a virtual machine to display**

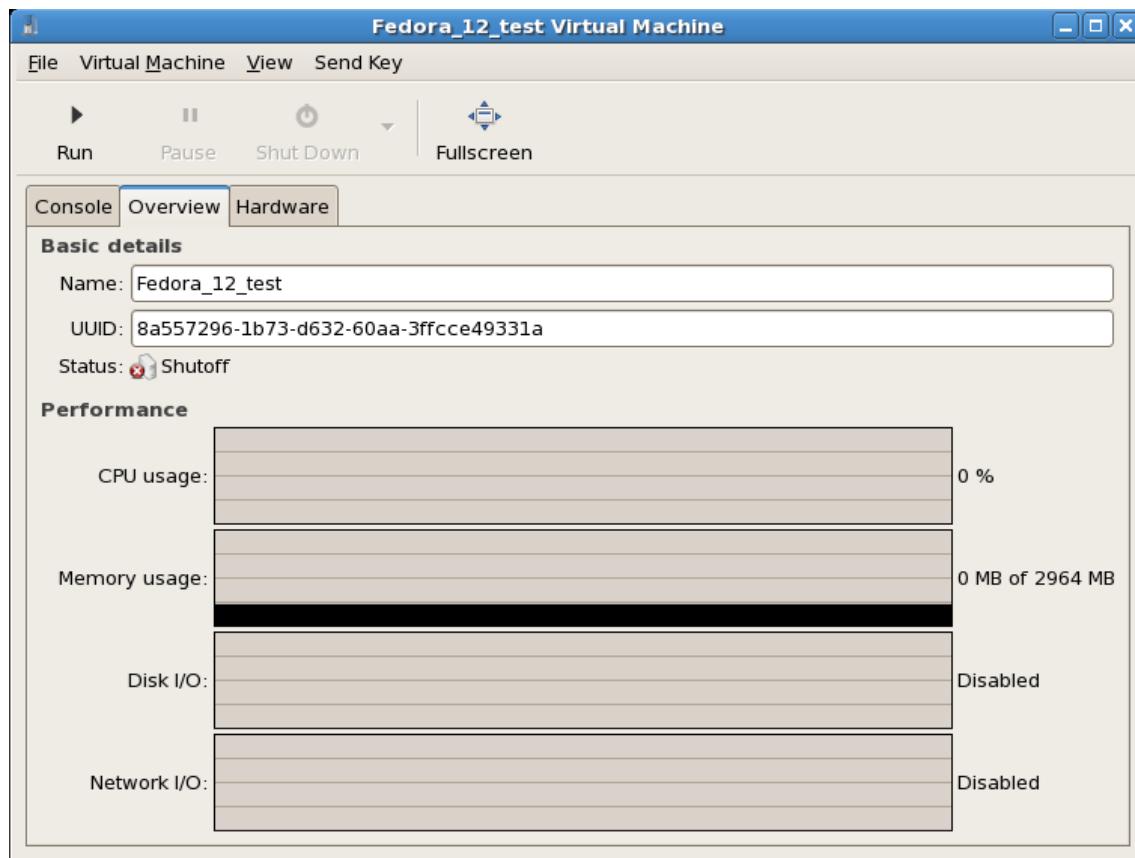
2. From the Virtual Machine Manager **Edit** menu, select **Machine Details** (or click the **Details** button on the bottom of the Virtual Machine Manager main window).



**Figure 27.9. Displaying the overview window**

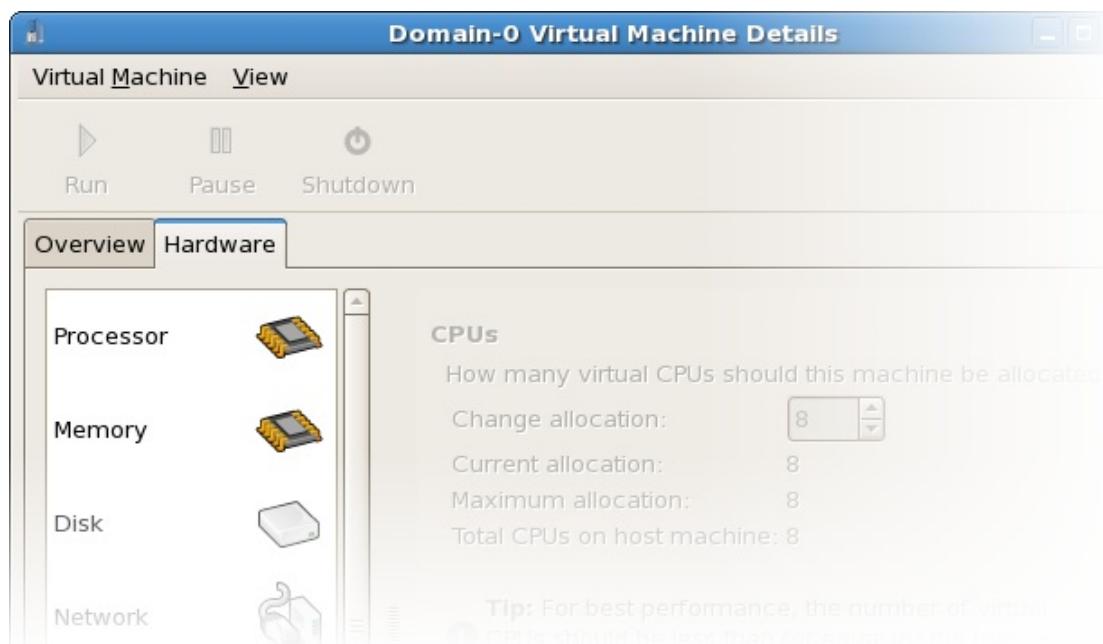
On the **Virtual Machine** window, click the **Overview** tab.

The **Overview** tab summarizes CPU and memory usage for the guest you specified.



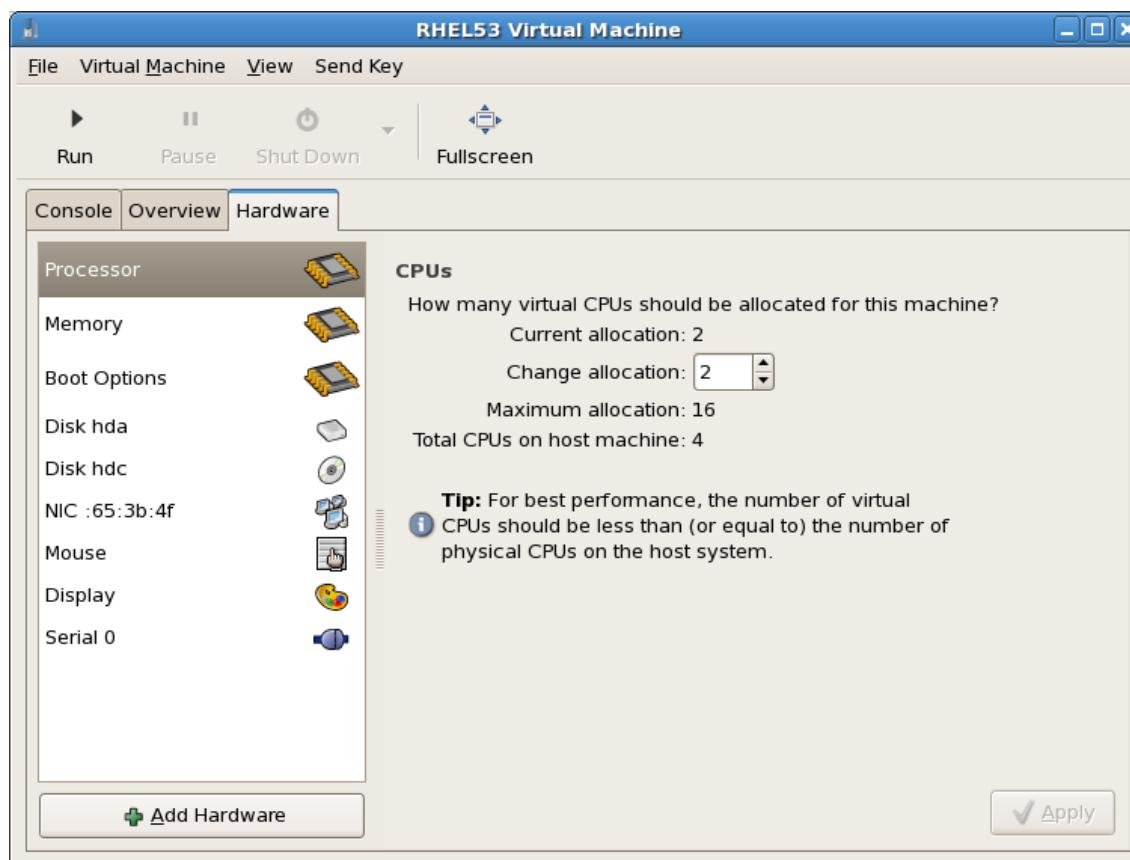
**Figure 27.10. Displaying guest details overview**

3. On the **Virtual Machine** window, click the **Hardware** tab.



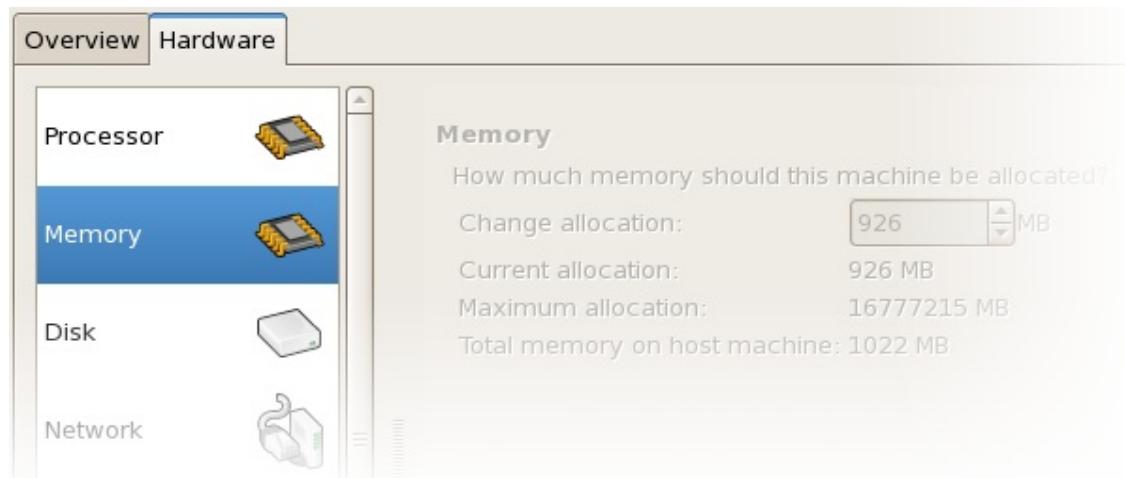
**Figure 27.11. Displaying guest hardware details**

- On the **Hardware** tab, click on **Processor** to view or change the current processor allocation.



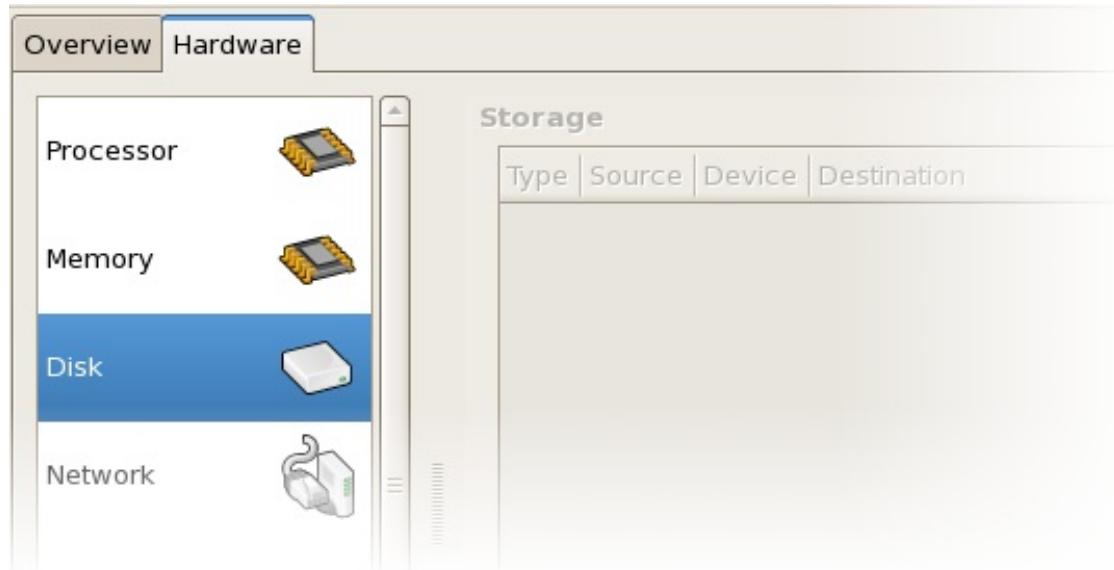
**Figure 27.12. Processor allocation panel**

- On the **Hardware** tab, click on **Memory** to view or change the current RAM memory allocation.



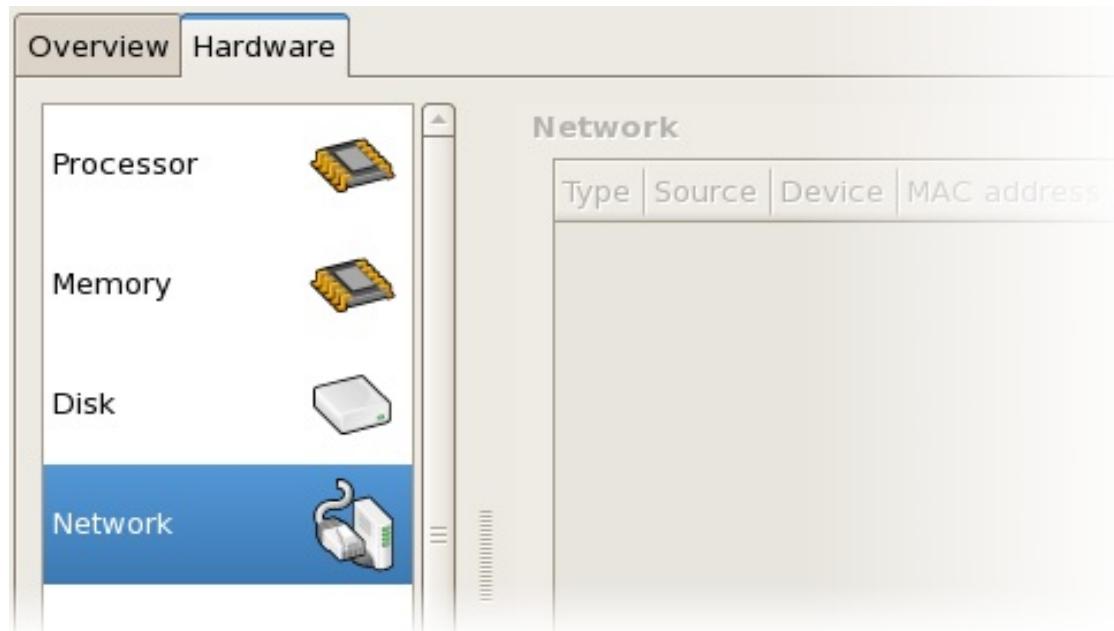
**Figure 27.13. Displaying memory allocation**

- On the **Hardware** tab, click on **Disk** to view or change the current hard disk configuration.



**Figure 27.14. Displaying disk configuration**

7. On the **Hardware** tab, click on **NIC** to view or change the current network configuration.



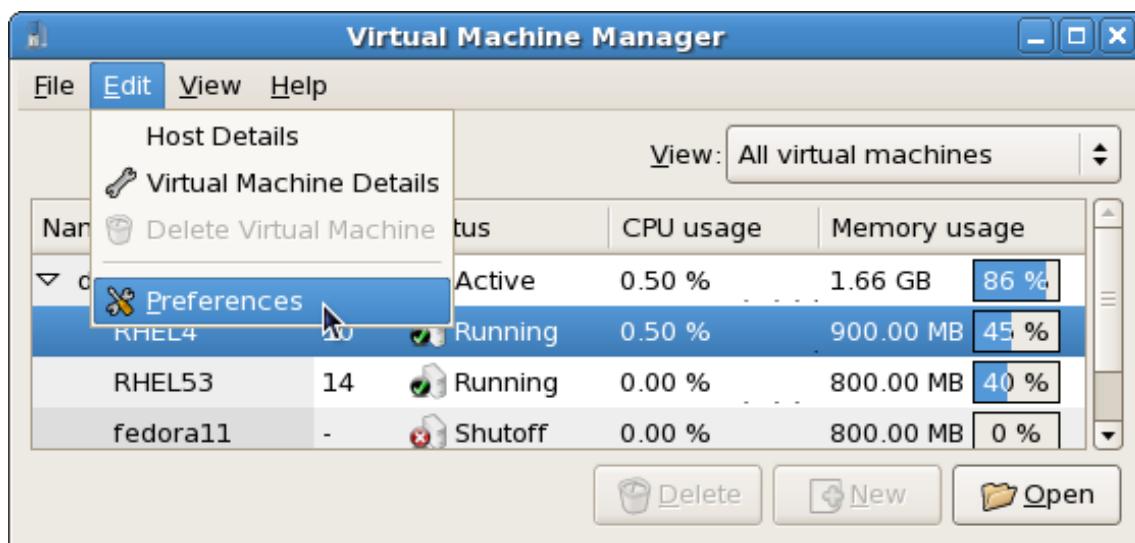
**Figure 27.15. Displaying network configuration**

## 27.8. Status monitoring

Status status monitoring preferences can be modified with **virt-manager**'s preferences window.

To configure status monitoring:

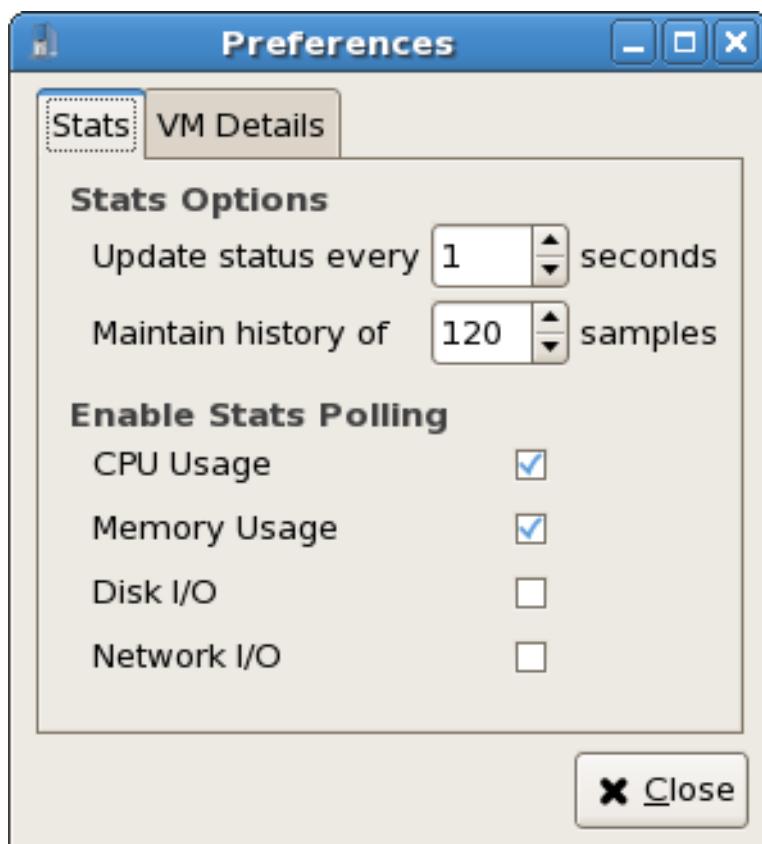
1. From the **Edit** menu, select **Preferences**.



**Figure 27.16. Modifying guest preferences**

The **Preferences** window appears.

- From the **Stats** tab specify the time in seconds or stats polling options.

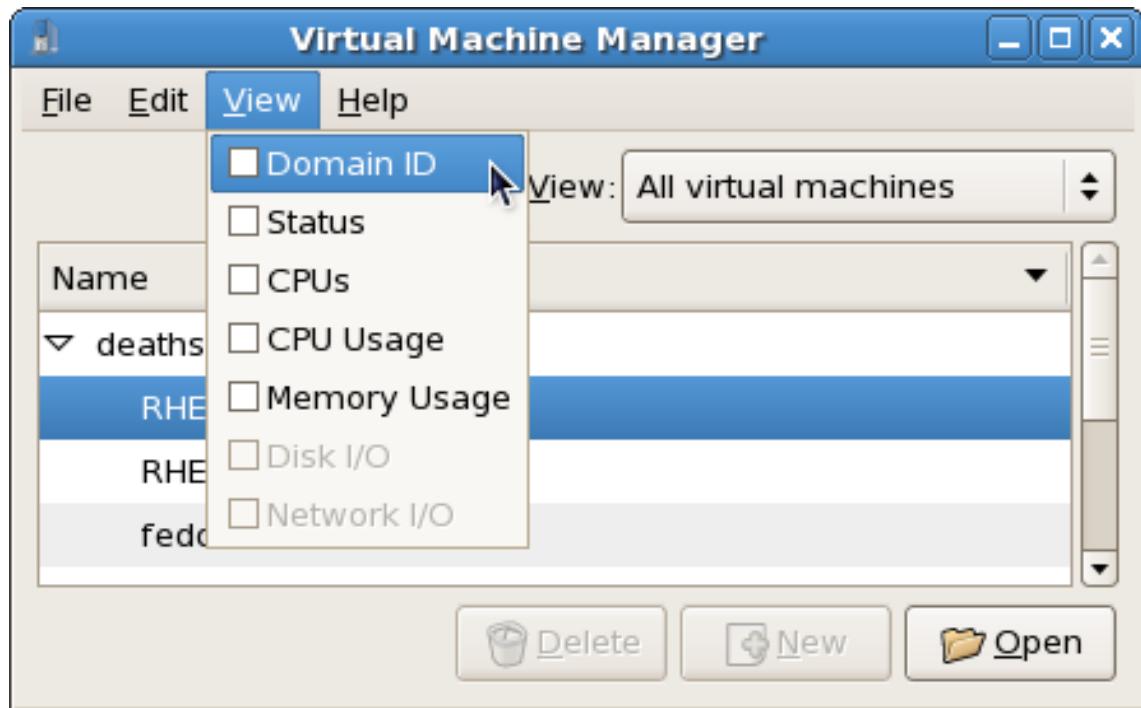


**Figure 27.17. Configuring status monitoring**

## 27.9. Displaying guest identifiers

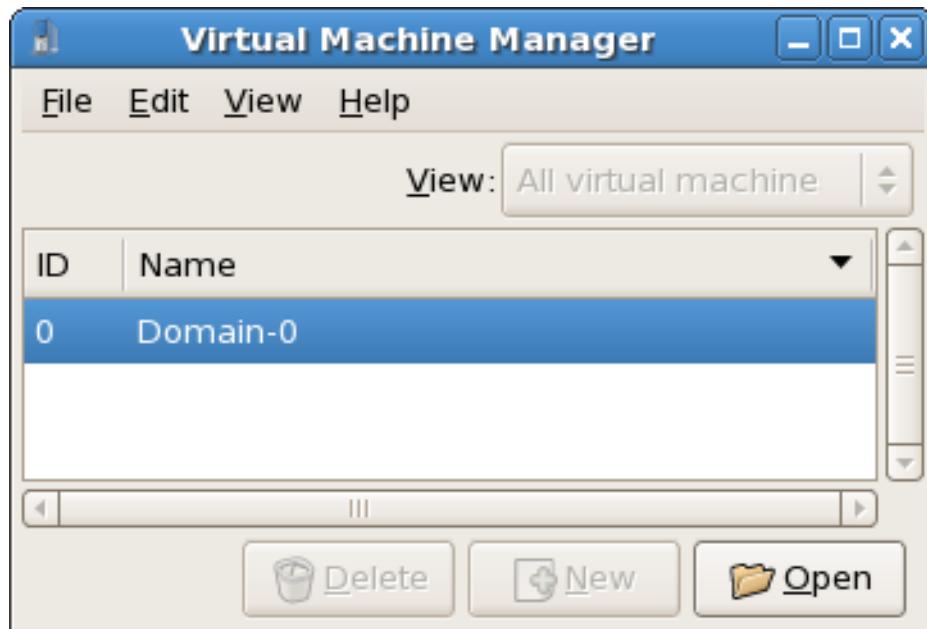
To view the guest IDs for all virtual machines on your system:

- From the **View** menu, select the **Domain ID** check box.



**Figure 27.18. Viewing guest IDs**

2. The Virtual Machine Manager lists the Domain IDs for all domains on your system.



**Figure 27.19. Displaying domain IDs**

## 27.10. 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.

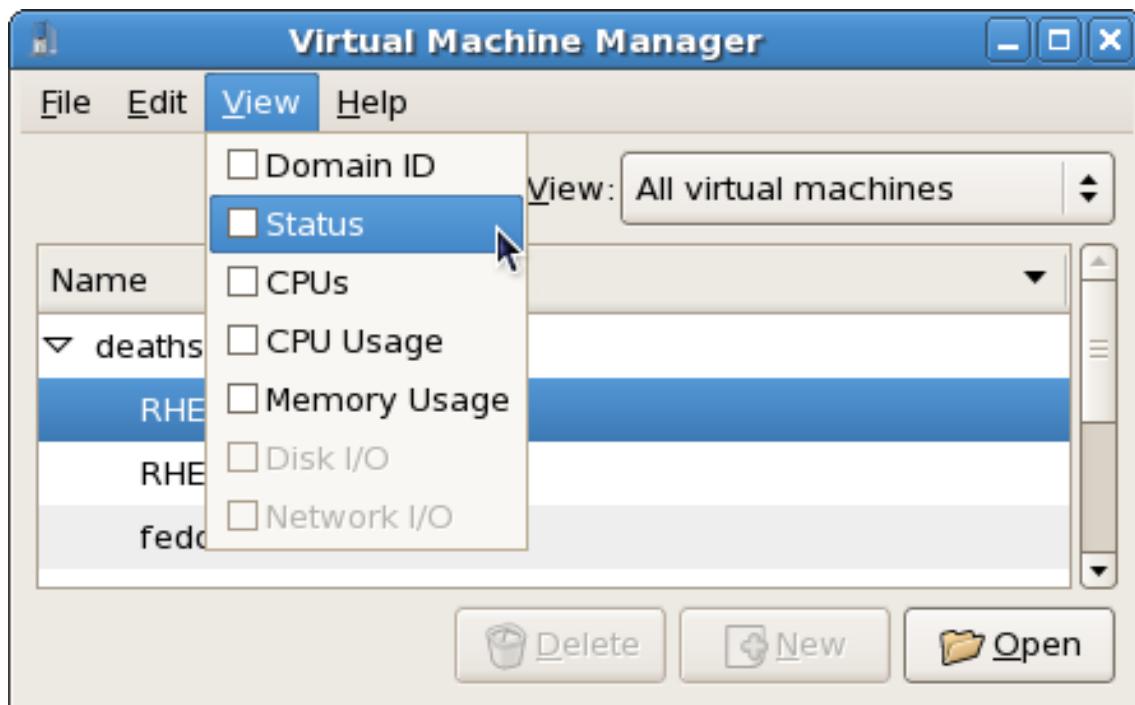


Figure 27.20. Selecting a virtual machine's status

2. The Virtual Machine Manager lists the status of all virtual machines on your system.

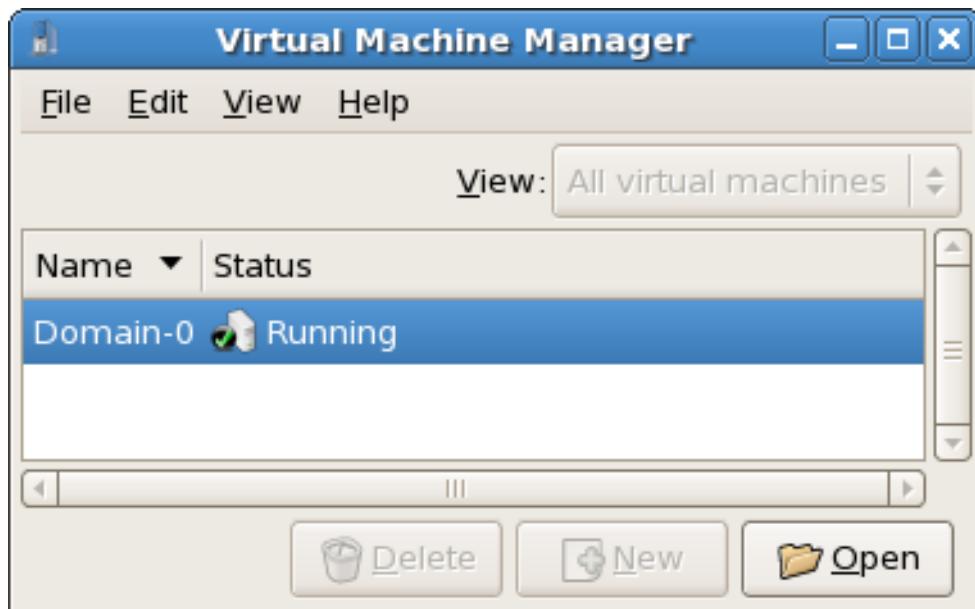
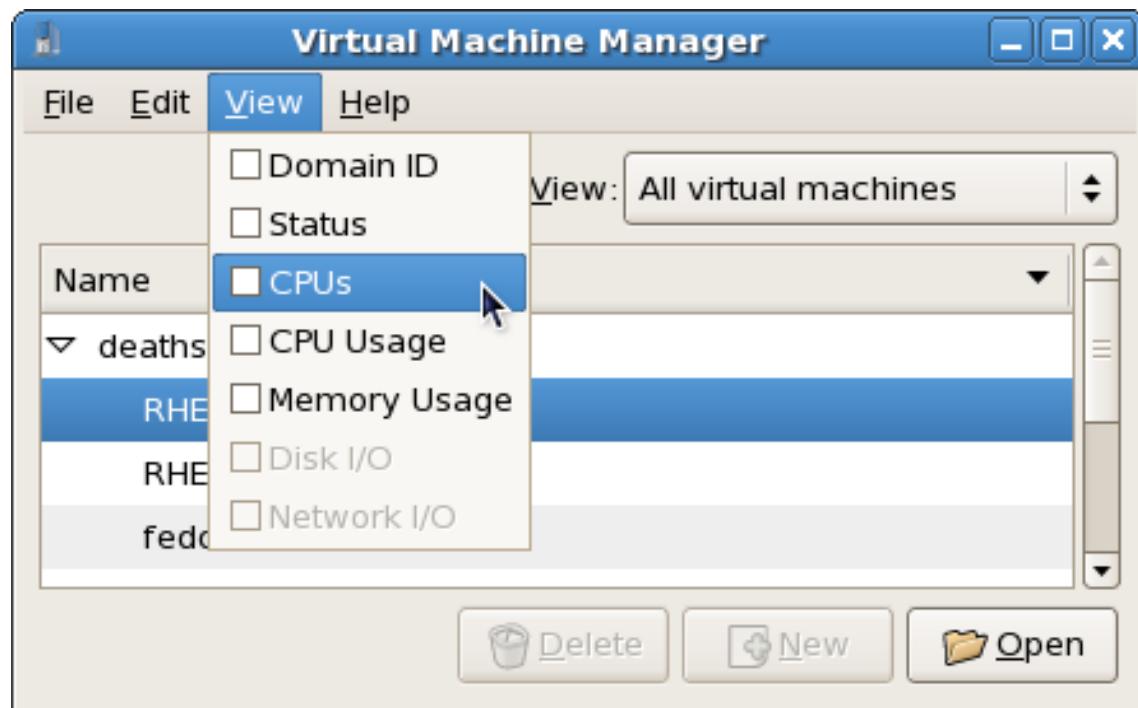


Figure 27.21. Displaying a virtual machine's status

## 27.11. Displaying virtual CPUs

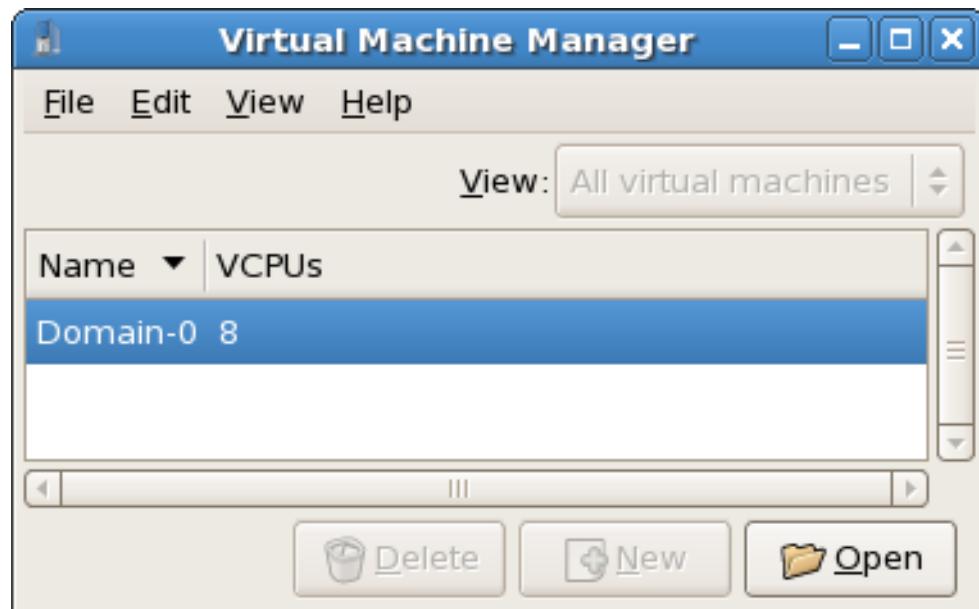
To view the amount of virtual CPUs for all virtual machines on your system:

1. From the **View** menu, select the **Virtual CPUs** check box.



**Figure 27.22. Selecting the virtual CPUs option**

2. The Virtual Machine Manager lists the Virtual CPUs for all virtual machines on your system.

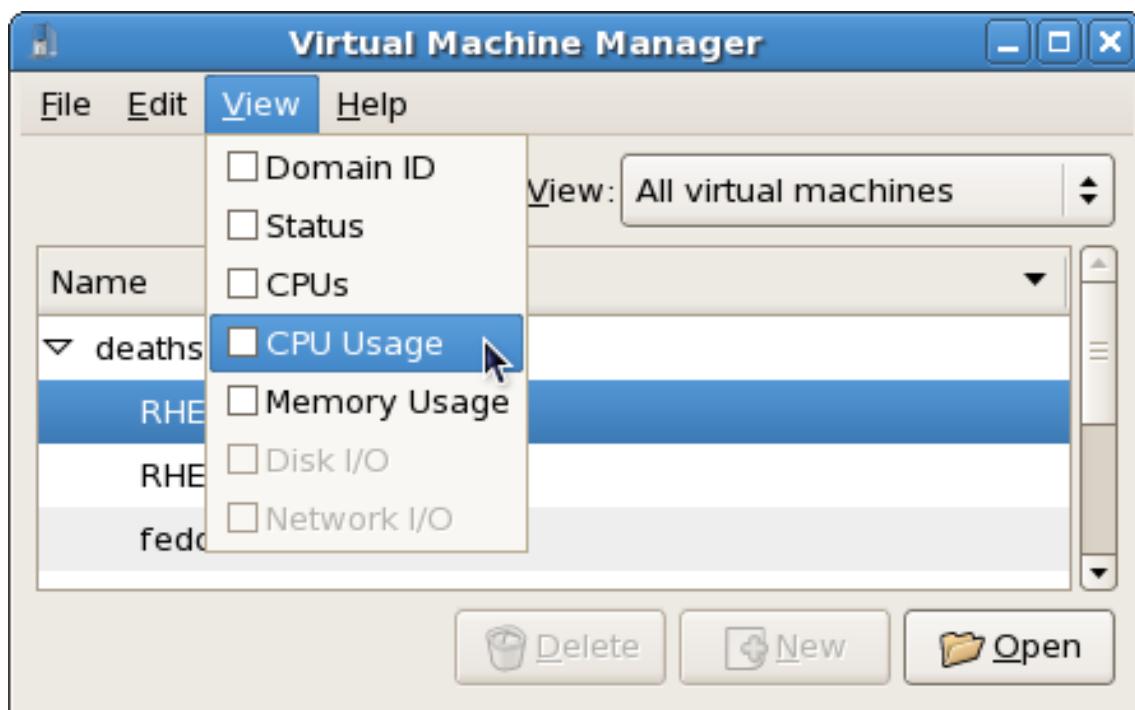


**Figure 27.23. Displaying Virtual CPUs**

## 27.12. Displaying CPU usage

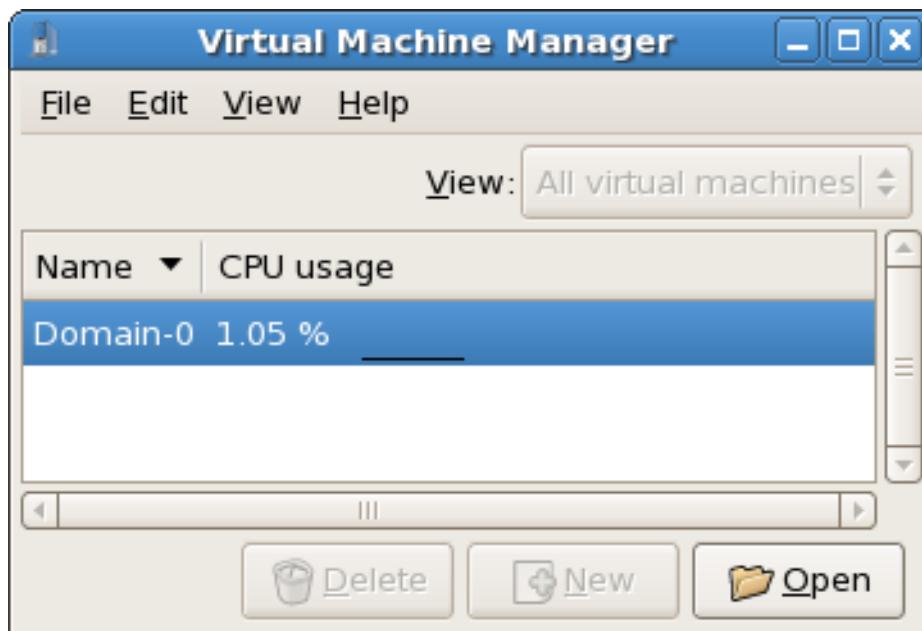
To view the CPU usage for all virtual machines on your system:

1. From the **View** menu, select the **CPU Usage** check box.



**Figure 27.24. Selecting CPU usage**

2. The Virtual Machine Manager lists the percentage of CPU in use for all virtual machines on your system.

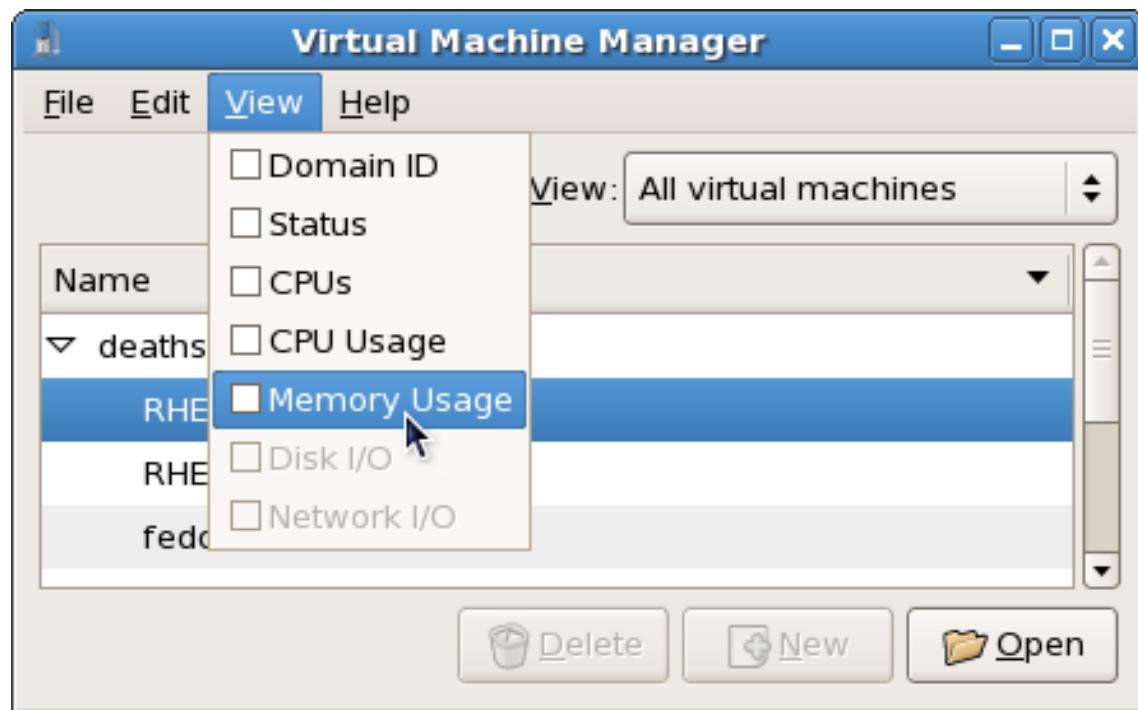


**Figure 27.25. Displaying CPU usage**

## 27.13. Displaying memory usage

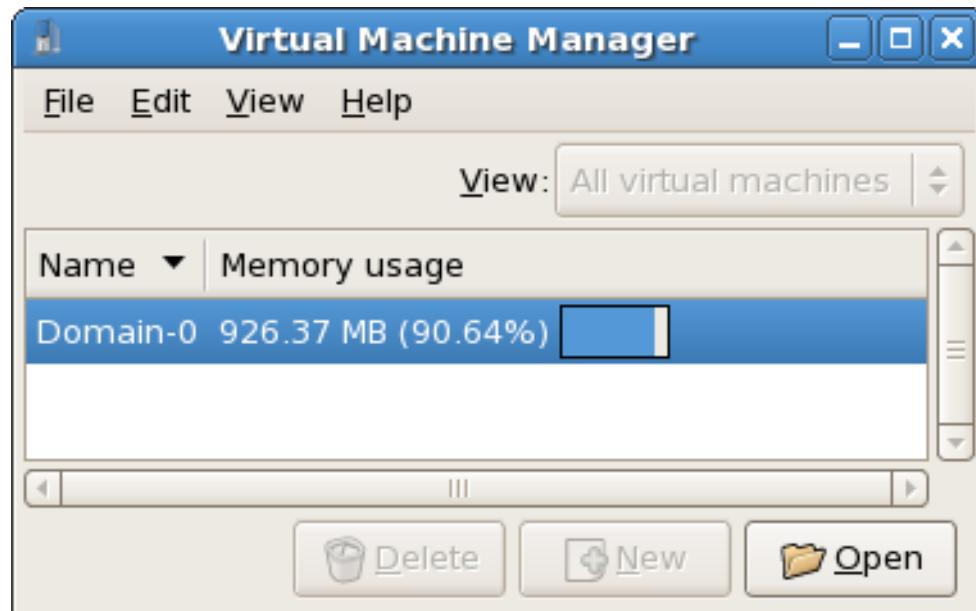
To view the memory usage for all virtual machines on your system:

1. From the **View** menu, select the **Memory Usage** check box.



**Figure 27.26. Selecting Memory Usage**

2. The Virtual Machine Manager lists the percentage of memory in use (in megabytes) for all virtual machines on your system.

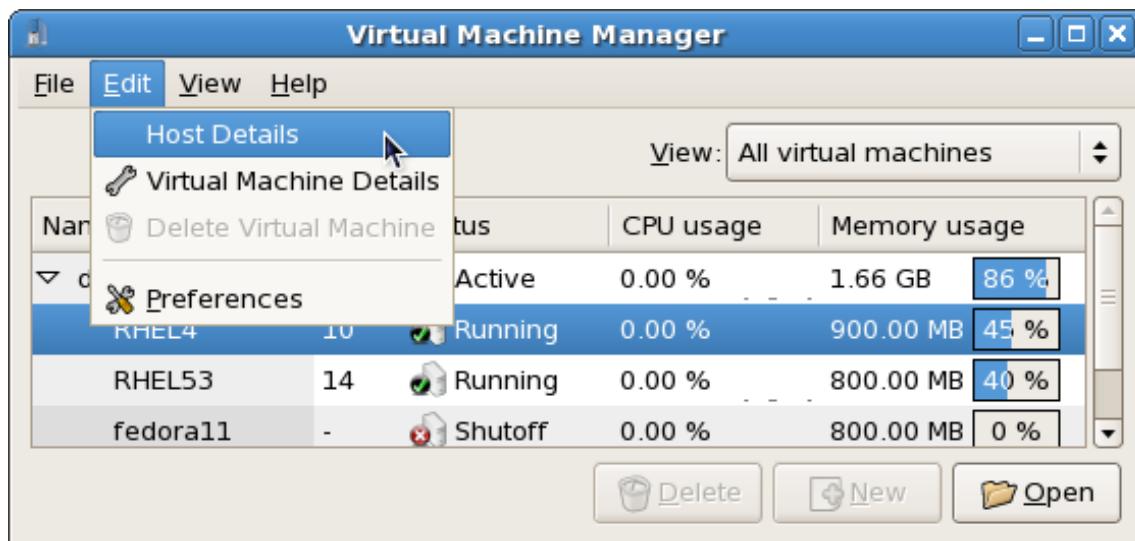


**Figure 27.27. Displaying memory usage**

## 27.14. Managing a virtual network

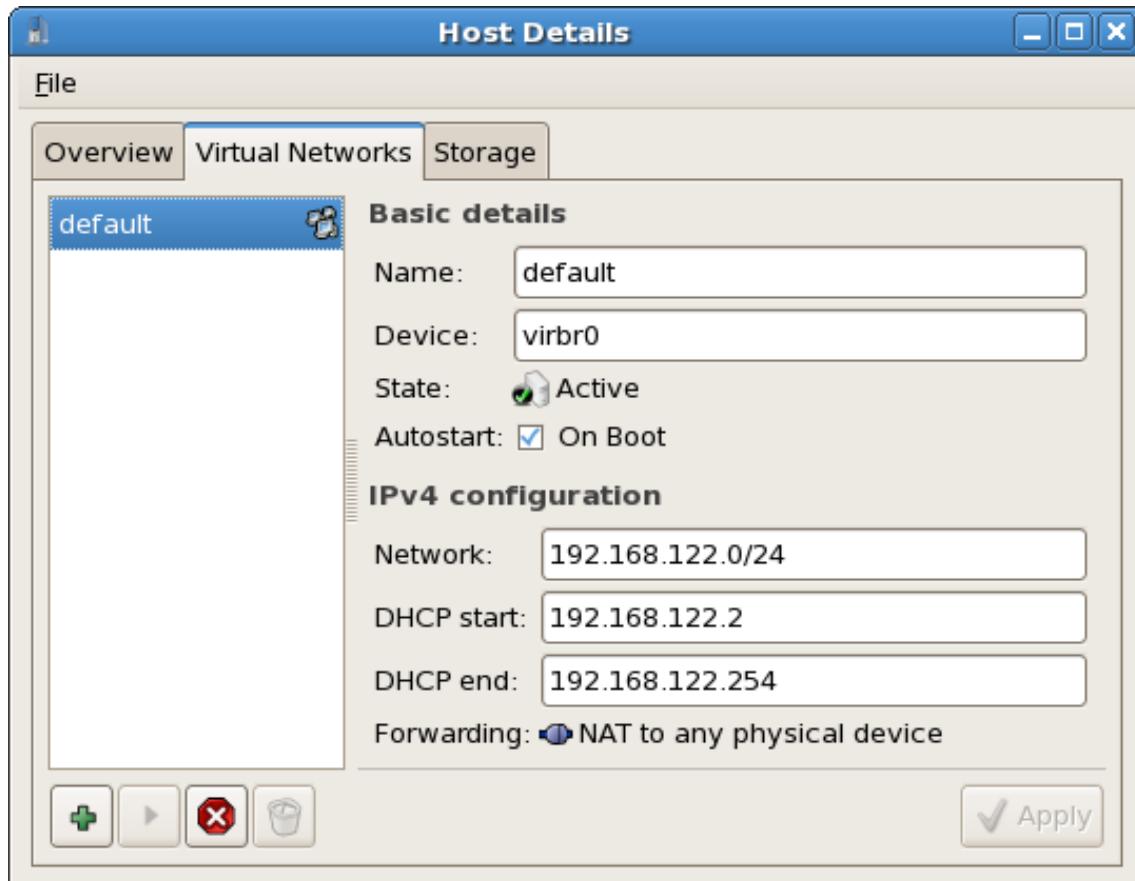
To configure a virtual network on your system:

1. From the **Edit** menu, select **Host Details**.



**Figure 27.28. Selecting a host's details**

2. This will open the **Host Details** menu. Click the **Virtual Networks** tab.



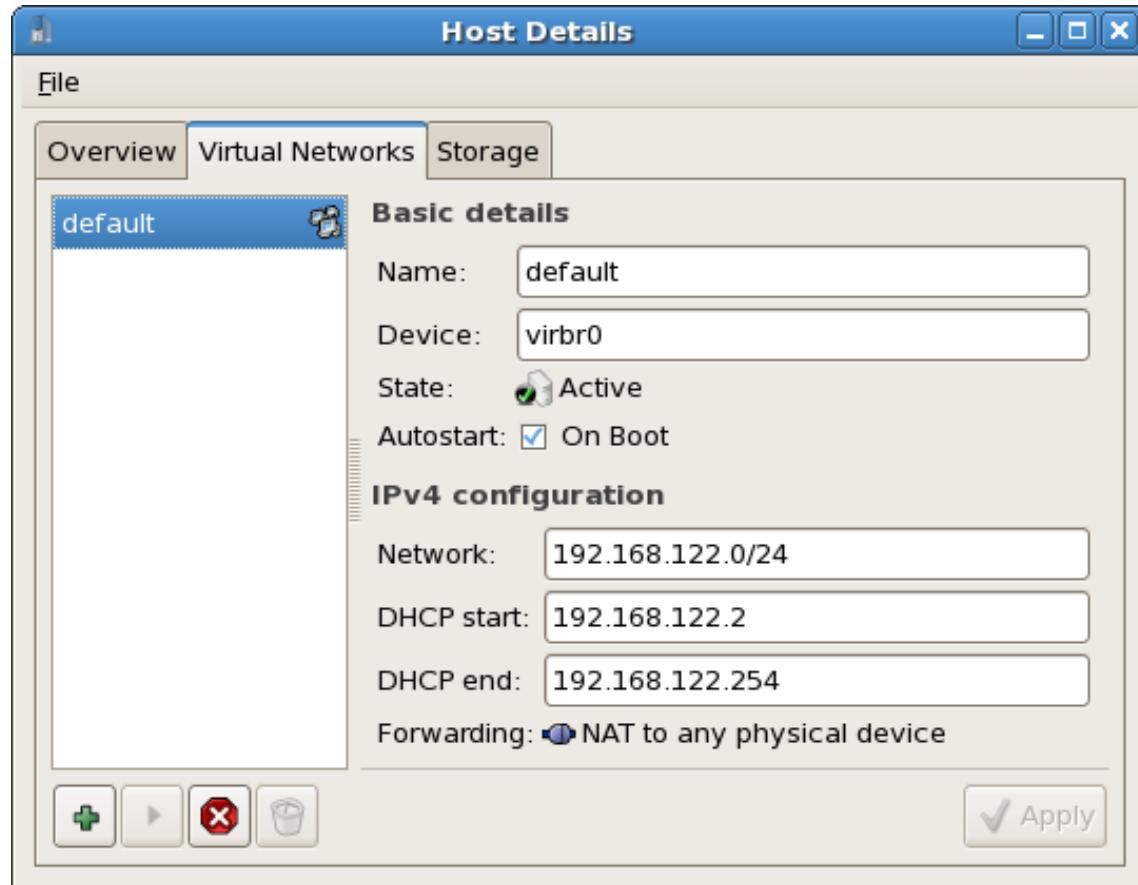
**Figure 27.29. Virtual network configuration**

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.

## 27.15. Creating a virtual network

To create a virtual network on your system:

1. Open the **Host Details** menu (see [Section 27.14, “Managing a virtual network”](#)) and click the **Add** button.



**Figure 27.30. Virtual network configuration**

This will open the **Create a new virtual network** menu. Click **Forward** to continue.

## Creating a new virtual network

This assistant will guide you through creating a new virtual network. You will be asked for some information about the virtual network you'd like to create, such as:

- A **name** for your new virtual network
- The IPv4 **address** and **netmask** to assign
- The **address range** from which the **DHCP** server will allocate addresses for virtual machines
- Whether to **forward** traffic to the physical network

 Cancel  Back  Forward

**Figure 27.31. Creating a new virtual network**

2. Enter an appropriate name for your virtual network and click **Forward**.

## Naming your virtual network

Please choose a name for your virtual network:

Network Name:

 **Example:** network1

 [Cancel](#)    [Back](#)    [Forward](#)

**Figure 27.32. Naming your virtual network**

3. Enter an IPv4 address space for your virtual network and click **Forward**.

## Choosing an IPv4 address space

You will need to choose an IPv4 address space for the virtual network:

Network:

 **Hint:** The network should be chosen from one of the IPv4 private address ranges. eg 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16

Netmask: 255.255.255.0

Broadcast: 192.168.100.255

Gateway: 192.168.100.1

Size: 256 addresses

Type: Private

 Cancel

 Back

 Forward

**Figure 27.33. 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.

## Selecting the DHCP range

Please choose the range of addresses the DHCP server can use to allocate to guests attached to the virtual network

Start: 192.168.100.128

End: 192.168.100.254

 **Tip:** Unless you wish to reserve some addresses to allow static network configuration in virtual machines, these parameters can be left with their default values.

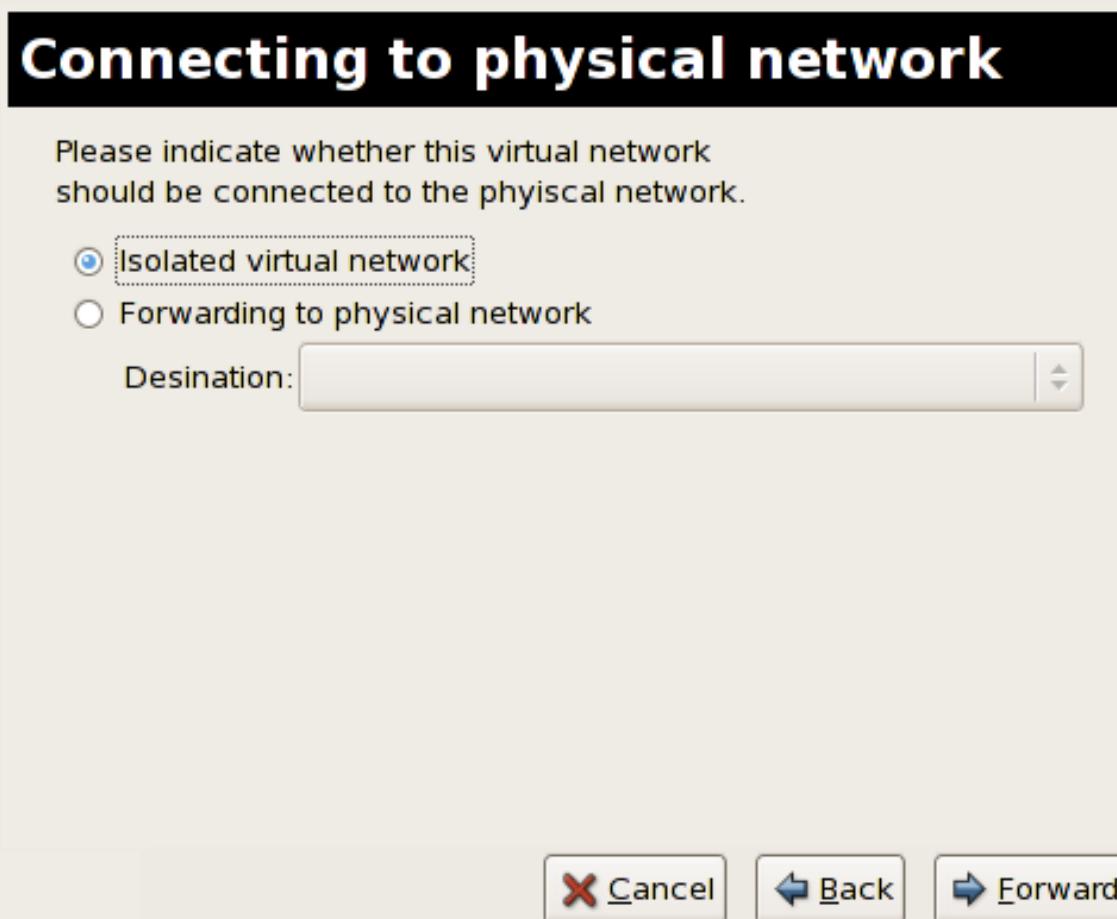
 Cancel

 Back

 Forward

**Figure 27.34. Selecting the DHCP range**

5. Select how the virtual network should connect to the physical network.



**Figure 27.35. 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**.

# Ready to create network

## Summary

Network name: network1

## IPv4 network

Network: 192.168.100.0/24

Gateway: 192.168.100.1

Netmask: 255.255.255.0

## DHCP

Start address: 192.168.100.128

End address: 192.168.100.254

## Forwarding

Connectivity: Isolated virtual network

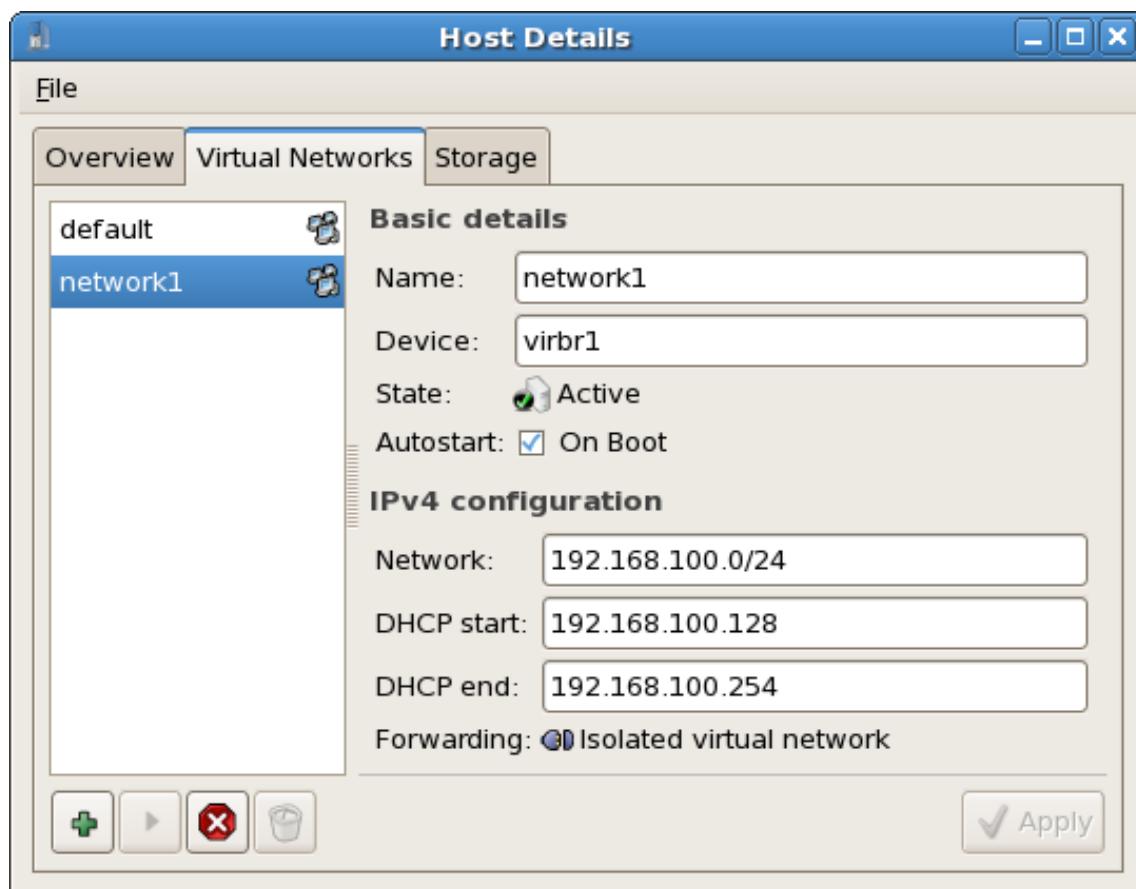
 Cancel

 Back

 Finish

**Figure 27.36. Ready to create network**

7. The new virtual network is now available in the **Virtual Network** tab of the **Host Details** menu.



**Figure 27.37. New virtual network is now available**

## Chapter 28. The xm command quick reference

The **xm** command can manage the Xen hypervisor. Most operations can be performed with the libvirt tools, **virt-manager** application or the **virsh** command. The **xm** command does not have the error checking capacity of the libvirt tools and should not be used for tasks the libvirt tools support.

There are a few operations which currently can not be performed using **virt-manager**. Some options for other Xen implementations of the **xm** command do not work in Red Hat Enterprise Linux 5. The list below provides an overview of command options available and unavailable.



### Warning

It is advised to use **virsh** or **virt-manager** instead of **xm**. The **xm** command does not handle error checking or configuration file errors very well and mistakes can lead to system instability or errors in virtual machines. Editing Xen configuration files manually is dangerous and should be avoided. Use this chapter at your own risk.

### Basic management options

The following are basic and commonly used **xm** commands:

- » **xm help [--long]**: view available options and help text.
- » use the **xm list** command to list active domains:

```
$ xm list
Name ID Mem(MiB) VCPUs State Time(s)
Domain-0 0 520 2 r----- 1275.5
r5b2-mySQL01 13 500 1 -b---- 16.1
```

- » **xm create [-c] DomainName/ID**: start a virtual machine. If the -c option is used, the start up process will attach to the guest's console.
- » **xm console DomainName/ID**: attach to a virtual machine's console.
- » **xm destroy DomainName/ID**: terminates a virtual machine , similar to a power off.
- » **xm reboot DomainName/ID**: reboot a virtual machine, runs through the normal system shut down and start up process.
- » **xm shutdown DomainName/ID**: shut down a virtual machine, runs a normal system shut down procedure.
- » **xm pause**
- » **xm unpause**
- » **xm save**
- » **xm restore [-p] DomainName/ID**: resume the state, and the execution of the domain. If the -p option is used, the domain will not be unpause after restoring it.
- » **xm migrate**

### Resource management options

Use the following **xm** commands to manage resources:

- » **xm mem-set**
- » use the **xm vcpu-list** to list virtualized CPU affinities:

```
$ xm vcpu-list
Name ID VCPUs CPU State Time(s) CPU Affinity
Domain-0 0 0 0 r-- 708.9 any cpu
Domain-0 0 1 1 -b- 572.1 any cpu
r5b2-mySQL01 13 0 1 -b- 16.1 any cpu
```

- » **xm vcpu-pin**

- » **xm vcpu-set**

- » use the **xm sched-credit** command to display scheduler parameters for a given domain:

```
$ xm sched-credit -d 0
{'cap': 0, 'weight': 256}
$ xm sched-credit -d 13
{'cap': 25, 'weight': 256}
```

## Monitoring and troubleshooting options

Use the following **xm** commands for monitoring and troubleshooting:

- » **xm top**
- » **xm dmesg**
- » **xm info**
- » **xm log**
- » use the **xm uptime** to display the uptime of guests and hosts:

```
$ xm uptime
Name ID Uptime
Domain-0 0 3:42:18
r5b2-mySQL01 13 0:06:27
```

- » **xm sysrq**
- » **xm dump-core**
- » **xm rename**
- » **xm domid**
- » **xm domname**

## Currently unsupported options

The **xm vnet-list** is currently unsupported.

## Chapter 29. Configuring the Xen kernel boot parameters

The GNU Grand Unified Boot Loader (GRUB) is a program for booting various installed operating systems or kernels. GRUB also allows the user to pass arguments to the kernel. The GRUB configuration file (located in **/boot/grub/grub.conf**) creates the list of operating systems the GRUB boot menu interface. When you install the **kernel-xen** RPM, a script adds the **kernel-xen** entry to the GRUB configuration file which boots **kernel-xen** by default. Edit the **grub.conf** file to modify the default kernel or to add additional kernel parameters.

```
title Red Hat Enterprise Linux Server (2.6.18-3.el5xen)
root (hd0,0)
kernel /xen.gz.-2.6.18-3.el5
module /vmlinuz-2.6..18-3.el5xen ro root=/dev/VolGroup00/LogVol00 rhgb quiet
module /initrd-2.6.18-3. el5xenxen.img
```

If you set your Linux grub entries to reflect this example, the boot loader loads the hypervisor, **initrd** image, and Linux kernel. Since the kernel entry is on top of the other entries, the kernel loads into memory first. The boot loader sends, and receives, command line arguments to and from the hypervisor and Linux kernel. This example entry shows how you would restrict the Dom0 linux kernel memory to 800 MB.

```
title Red Hat Enterprise Linux Server (2.6.18-3.el5xen)
root (hd0,0)
kernel /xen.gz.-2.6.18-3.el5 dom0_mem=800M
module /vmlinuz-2.6..18-3.el5xen ro root=/dev/VolGroup00/LogVol00 rhgb quiet
module /initrd-2.6.18-3. el5xenxen.img
```

You can use these GRUB parameters to configure the Virtualization hypervisor:

**mem**

This limits the amount of memory that is available to the hypervisor kernel.

**com1=115200, 8n1**

This enables the first serial port in the system to act as serial console (com2 is assigned for the next port, and so on).

**dom0\_mem**

This limits the memory available for the hypervisor.

**dom0\_max\_vcpus**

This limits the amount of CPUs visible to the Xen domain0.

**acpi**

This switches the ACPI hypervisor to the hypervisor and domain0. The ACPI parameter options include:

```
/* **** Linux config options: propagated to domain0 ****/
/* "acpi=off": Disables both ACPI table parsing and interpreter. */
/* "acpi=force": Overrides the disable blacklist. */
/* "acpi=strict": Disables out-of-spec workarounds. */
```

```
/* "acpi=ht": Limits ACPI from boot-time to enable HT. */
/* "acpi=noirq": Disables ACPI interrupt routing. */
```

```
noacpi
```

This disables ACPI for interrupt delivery.

## Chapter 30. Configuring ELILO

ELILO is the boot loader used on EFI-based systems, notably Itanium®. Similar to the GRUB, the boot loader on x86 and x86-64 systems, ELILO allows the user to select which installed kernel to load during the system boot sequence. ELILO also allows the user to pass arguments to the kernel. The ELILO configuration file, which is located in the EFI boot partition and symbolically linked to `/etc/elilo.conf`, contains a list of global options and image stanzas. When you install the `kernel-xen` RPM, a post install script adds the appropriate image stanza to the `elilo.conf`.



### Important

This section on ELILO is for systems running the Xen kernel on the Intel Itanium architecture.

The ELILO configuration file has two sections:

- » Global options that affect the behavior of ELILO and all the entries. Typically there is no need to change these from the default values.
- » Image stanzas that define a boot selection along with associated options.

Here is a sample image stanza in `elilo.conf`:

```
image=vmlinuz-2.6.18-92.el5xen
 vmm=xen.gz-2.6.18-92.el5
 label=linux
 initrd=initrd-2.6.18-92.el5xen.img
 read-only
 root=/dev/VolGroup00/rhel5_2
 append="-- rhgb quiet"
```

The **`image`** parameter indicates the following lines apply to a single boot selection. This stanza defines a hypervisor (**`vmm`**), **`initrd`**, and command line arguments (**`read-only`**, **`root`** and **`append`**) to the hypervisor and kernel. When ELILO is loaded during the boot sequence, the image is labeled **`linux`**.

ELILO translates **`read-only`** to the kernel command line option **`ro`** which causes the root file system to be mounted read-only until the **`initscripts`** mount the root drive as read-write. ELILO copies the "**`root`**" line to the kernel command line. These are merged with the "**`append`**" line to build a complete command line:

```
-- root=/dev/VolGroup00/rhel5_2 ro rhgb quiet"
```

The **--** symbols delimit hypervisor and kernel arguments. The hypervisor arguments come first, then the **--** delimiter, followed by the kernel arguments. The hypervisor does not usually have any arguments.

### Note

ELILO passes the entire command line to the hypervisor. The hypervisor divides the content and passes the kernel options to the kernel.

To customize the hypervisor, insert parameters before the `--`. An example of the hypervisor memory (`mem`) parameter and the `quiet` parameter for the kernel:

```
append="dom0_mem=2G -- quiet"
```

## ELILO hypervisor parameters

Parameter	Description
<code>mem=</code>	The <code>mem</code> parameter defines the hypervisor maximum RAM usage. Any additional RAM in the system is ignored. The parameter may be specified with a B, K, M or G suffix; representing bytes, kilobytes, megabytes and gigabytes respectively. If no suffix is specified the default unit is kilobytes.
<code>dom0_mem=</code>	<code>dom0_mem=</code> sets the amount of RAM to allocate to dom0. The same suffixes are respected as for the mem parameter above. The default in Red Hat Enterprise Linux 5.2 on Itanium® is 4G.
<code>dom0_max_vcpus=</code>	<code>dom0_max_vcpus=</code> sets the number of CPUs to allocate to the hypervisor. The default in Red Hat Enterprise Linux 5.2 on Itanium® is 4.
<code>com1=&lt;baud&gt;,DPS,&lt;io_base&gt;,&lt;irq&gt;</code>	<code>com1=</code> sets the parameters for the first serial line. For example, <code>com1=9600,8n1,0x408,5</code> . The <code>io_base</code> and <code>irq</code> options can be omitted to leave them as the standard defaults. The <code>baud</code> parameter can be set as <code>auto</code> to indicate the boot loader setting should be preserved. The <code>com1</code> parameter can be omitted if serial parameters are set as global options in ELILO or in the EFI configuration.
<code>com2=&lt;baud&gt;,DPS,&lt;io_base&gt;,&lt;irq&gt;</code>	Set the parameters for the second serial line. Refer the description of the <code>com1</code> parameter above.
<code>console=&lt;specifier_list&gt;</code>	The <code>console</code> is a comma delimited preference list for the console options. Options include vga, com1 and com2. This setting should be omitted because the hypervisor attempts to inherit EFI console settings.

A modified example of the configuration above, showing syntax for appending memory and cpu allocation parameters to the hypervisor:

```
image=vmlinuz-2.6.18-92.el5xen
vmm=xen.gz-2.6.18-92.el5
label=linux
initrd=initrd-2.6.18-92.el5xen.img
read-only
root=/dev/VolGroup00/rhel5_2
append="dom0_mem=2G dom0_max_vcpus=2 --"
```

Additionally this example removes the kernel parameters "***rhgb quiet***" so that kernel and ***initscript*** output are generated on the console. Note the double-dash remains so that the append line is correctly interpreted as hypervisor arguments.

## Chapter 31. libvirt configuration reference

This chapter provides references for various parameters of libvirt XML configuration files

**Table 31.1. libvirt configuration files**

Item	Description
<b>pae</b>	Specifies the physical address extension configuration data.
<b>apic</b>	Specifies the advanced programmable interrupt controller configuration data.
<b>memory</b>	Specifies the memory size in megabytes.
<b>vcpus</b>	Specifies the numbers of virtual CPUs.
<b>console</b>	Specifies the port numbers to export the domain consoles to.
<b>nic</b>	Specifies the number of virtual network interfaces.
<b>vif</b>	Lists the randomly-assigned MAC addresses and bridges assigned to use for the domain's network addresses.
<b>disk</b>	Lists the block devices to export to the domain and exports physical devices to domain with read only access.
<b>dhcp</b>	Enables networking using DHCP.
<b>netmask</b>	Specifies the configured IP netmasks.
<b>gateway</b>	Specifies the configured IP gateways.
<b>acpi</b>	Specifies the advanced configuration power interface configuration data.

## Chapter 32. Xen configuration files

Red Hat Enterprise Linux uses **libvirt** configuration files for most tasks. Some users may need Xen configuration files which contain the following standard variables. Configuration items within these files must be enclosed in single quotes('). These configuration files reside in the **/etc/xen** directory.

The table below, [Table 32.1, “Xen configuration file reference”](#), is formatted output from **xm create --help\_config**.

**Table 32.1. Xen configuration file reference**

Parameter	Description
<b>vncpasswd=NAME</b>	Password for VNC console on HVM domain.
<b>vncviewer=no   yes</b>	Spawn a vncviewer listening for a vnc server in the domain. The address of the vncviewer is passed to the domain on the kernel command line using <b>VNC_SERVER=&lt;host&gt;:&lt;port&gt;</b> . The port used by vnc is 5500 + DISPLAY. A display value with a free port is chosen if possible. Only valid when vnc=1.
<b>vncconsole=no   yes</b>	Spawn a vncviewer process for the domain's graphical console. Only valid when vnc=1.
<b>name=NAME</b>	Domain name. Must be unique.
<b>bootloader=FILE</b>	Path to boot loader.
<b>bootargs=NAME</b>	Arguments to pass to boot loader.
<b>bootentry=NAME</b>	DEPRECATED. Entry to boot via boot loader. Use <b>bootargs</b> .
<b>kernel=FILE</b>	Path to kernel image.
<b>ramdisk=FILE</b>	Path to ramdisk.
<b>features=FEATURES</b>	Features to enable in guest kernel
<b>builder=FUNCTION</b>	Function to use to build the domain.
<b>memory=MEMORY</b>	Domain memory in MB.
<b>maxmem=MEMORY</b>	Maximum domain memory in MB.
<b>shadow_memory=MEMORY</b>	Domain shadow memory in MB.
<b>cpu=CPU</b>	CPU which hosts VCPU0.
<b>cpus=CPUS</b>	CPUS to run the domain on.
<b>pae=PAE</b>	Disable or enable PAE of HVM domain.
<b>acpi=ACPI</b>	Disable or enable ACPI of HVM domain.
<b>apic=APIC</b>	Disable or enable APIC of HVM domain.
<b>vcpus=VCPU斯</b>	The number of Virtual CPUS in domain.
<b>cpu_weight=WEIGHT</b>	Set the new domain's cpu weight. <b>WEIGHT</b> is a float that controls the domain's share of the cpu.
<b>restart=onreboot   always   never</b>	Deprecated. Use <b>on_poweroff</b> , <b>on_reboot</b> , and <b>on_crash</b> instead. Whether the domain should be restarted on exit. - <b>onreboot</b> : restart on exit with shutdown code reboot - <b>always</b> : always restart on exit, ignore exit code - <b>never</b> : never restart on exit, ignore exit code

Parameter	Description
<b>on_poweroff=</b> destroy   restart   preserve   destroy	Behavior when a domain exits with reason ' <b>poweroff</b> '. - destroy: the domain is cleaned up as normal; - restart: a new domain is started in place of the old one; - preserve: no clean-up is done until the domain is manually destroyed (using <b>xm destroy</b> , for example); - rename-restart: the old domain is not cleaned up, but is renamed and a new domain started in its place.
<b>on_reboot=</b> destroy   restart   preserve   destroy	Behavior when a domain exits with reason 'reboot'. - destroy: the domain is cleaned up as normal; - restart: a new domain is started in place of the old one; - preserve: no clean-up is done until the domain is manually destroyed (using <b>xm destroy</b> , for example); - rename-restart: the old domain is not cleaned up, but is renamed and a new domain started in its place.
<b>on_crash=</b> destroy   restart   preserve   destroy	Behavior when a domain exits with reason 'crash'. - destroy: the domain is cleaned up as normal; - restart: a new domain is started in place of the old one; - preserve: no clean-up is done until the domain is manually destroyed (using <b>xm destroy</b> , for example); - rename-restart: the old domain is not cleaned up, but is renamed and a new domain started in its place.
<b>blkif=no   yes</b>	Make the domain a block device backend.
<b>netif=no   yes</b>	Make the domain a network interface backend.
<b>tpmif=no   yes</b>	Make the domain a TPM interface backend.
<b>disk=</b> phy: <i>DEV</i> , <i>VDEV</i> , <i>MODE</i> [, <i>DOM</i> ]	Add a disk device to a domain. The physical device is <i>DEV</i> , which is exported to the domain as <i>VDEV</i> . The disk is read-only if <i>MODE</i> is <b>r</b> , read-write if <i>MODE</i> is <b>w</b> . If <i>DOM</i> is specified it defines the backend driver domain to use for the disk. The option may be repeated to add more than one disk.
<b>pci=</b> <i>BUS:DEV.FUNC</i>	Add a PCI device to a domain, using given parameters (in hex). For example <b>pci=c0:02.1a</b> . The option may be repeated to add more than one pci device.
<b>ioports=</b> <i>FROM</i> [- <i>TO</i> ]	Add a legacy I/O range to a domain, using given params (in hex). For example <b>ioports=02f8-02ff</b> . The option may be repeated to add more than one i/o range.
<b>irq=</b> <i>IRQ</i>	Add an IRQ (interrupt line) to a domain. For example <b>irq=7</b> . This option may be repeated to add more than one IRQ.
<b>usbport=</b> <i>PATH</i>	Add a physical USB port to a domain, as specified by the path to that port. This option may be repeated to add more than one port.

Parameter	Description
<b>vfb=</b> <i>type={vnc,sdl}</i> , <b>vncunused=</b> <i>1</i> , <b>vncdisplay=</b> <i>N</i> ,  <b>vnclisten=</b> <i>ADDR</i> , <b>display=</b> <i>DISPLAY</i> , <b>xauthority=</b> <i>XAUTHORITY</i> , <b>vncpasswd=</b> <i>PASSWORD</i> ,  <b>keymap=</b> <i>KEYMAP</i>	Make the domain a framebuffer backend. The backend type should be either <b>sdl</b> or <b>vnc</b> . For <b>type=vnc</b> , connect an external vncviewer. The server will listen on <i>ADDR</i> (default 127.0.0.1) on port <i>N</i> +5900. <i>N</i> defaults to the domain id. If <b>vncunused=1</b> , the server will try to find an arbitrary unused port above 5900. For <b>type=sdl</b> , a viewer will be started automatically using the given <i>DISPLAY</i> and <i>XAUTHORITY</i> , which default to the current user's ones.
<b>vif=</b> <i>type=TYPE</i> , <b>mac=</b> <i>MAC</i> , <b>bridge=</b> <i>BRIDGE</i> , <b>ip=</b> <i>IPADDR</i> ,  <b>script=</b> <i>SCRIPT</i> , <b>backend=</b> <i>DOM</i> , <b>vifname=</b> <i>NAME</i>	Add a network interface with the given <i>MAC</i> address and bridge. The <b>vif</b> is configured by calling the given configuration script. If <i>type</i> is not specified, default is netfront not ioemu device. If <i>mac</i> is not specified a random <i>MAC</i> address is used. If not specified then the network backend chooses its own <i>MAC</i> address. If <i>bridge</i> is not specified the first bridge found is used. If <i>script</i> is not specified the default script is used. If <i>backend</i> is not specified the default backend driver domain is used. If <i>vifname</i> is not specified the backend virtual interface will have name <i>vifD.N</i> where <i>D</i> is the domain id and <i>N</i> is the interface id. This option may be repeated to add more than one <i>vif</i> . Specifying <i>vifs</i> will increase the number of interfaces as needed.
<b>vtpm=</b> <i>instance=INSTANCE</i> , <i>backend=DOM</i>	Add a TPM interface. On the backend side use the given instance as virtual TPM instance. The given number is merely the preferred instance number. The hotplug script will determine which instance number will actually be assigned to the domain. The association between virtual machine and the TPM instance number can be found in <b>/etc/xen/vtpm.db</b> . Use the backend in the given domain.
<b>access_control=</b> <i>policy=POLICY</i> , <i>label=LABEL</i>	Add a security label and the security policy reference that defines it. The local ssid reference is calculated when starting or resuming the domain. At this time, the policy is checked against the active policy as well. This way, migrating through the save or restore functions are covered and local labels are automatically created correctly on the system where a domain is started or resumed.
<b>nics=</b> <i>NUM</i>	DEPRECATED. Use empty <i>vif</i> entries instead. Set the number of network interfaces. Use the <i>vif</i> option to define interface parameters, otherwise defaults are used. Specifying <i>vifs</i> will increase the number of interfaces as needed.
<b>root=</b> <i>DEVICE</i>	Set the <b>root=</b> parameter on the kernel command line. Use a device, e.g. <b>/dev/sda1</b> , or <b>/dev/nfs</b> for NFS root.

Parameter	Description
<b>extra=ARGS</b>	Set extra arguments to append to the kernel command line.
<b>ip=IPADDR</b>	Set the kernel IP interface address.
<b>gateway=IPADDR</b>	Set the kernel IP gateway.
<b>netmask=MASK</b>	Set the kernel IP netmask.
<b>hostname=NAME</b>	Set the kernel IP hostname.
<b>interface=INTF</b>	Set the kernel IP interface name.
<b>dhcp=off dhcp</b>	Set the kernel dhcp option.
<b>nfs_server=IPADDR</b>	Set the address of the NFS server for NFS root.
<b>nfs_root=PATH</b>	Set the path of the root NFS directory.
<b>device_model=FILE</b>	Path to device model program.
<b>fda=FILE</b>	Path to fda
<b>fdb=FILE</b>	Path to fdb
<b>serial=FILE</b>	Path to serial or pty or vc
<b>localtime=no   yes</b>	Is RTC set to localtime
<b>keymap=FILE</b>	Set keyboard layout used
<b>usb=no   yes</b>	Emulate USB devices
<b>usbdevice=NAME</b>	Name of a <b>USB</b> device to add
<b>stdvga=no   yes</b>	Use <b>std vga</b> or Cirrus Logic graphics
<b>isa=no   yes</b>	Simulate an <b>ISA</b> only system
<b>boot=a b c d</b>	Default boot device
<b>nographic=no   yes</b>	Should device models use graphics?
<b>soundhw=audiodev</b>	Should device models enable audio device?
<b>vnc</b>	Should the device model use VNC?
<b>vncdisplay</b>	VNC display to use
<b>vnclisten</b>	Address for VNC server to listen on.
<b>vncunused</b>	Try to find an unused port for the VNC server. Only valid when vnc=1.
<b>sdl</b>	Should the device model use SDL?
<b>display=DISPLAY</b>	X11 display to use
<b>xauthority=XAUTHORITY</b>	X11 Authority to use
<b>uuid</b>	xenstore UUID (universally unique identifier) to use. One will be randomly generated if this option is not set, just like MAC addresses for virtual network interfaces. This must be a unique value across the entire cluster.

[Table 32.3, “Configuration parameter default values”](#) lists all configuration parameters available, the Python parser function which sets the value and default values. The setter function gives an idea of what the parser does with the values you specify. It reads these as Python values, then feeds them to a setter function to store them. If the value is not valid Python, you get an obscure error message. If the setter rejects your value, you should get a reasonable error message, except it appears to get lost somehow, along with your bogus setting. If the setter accepts, but the value is incorrect the application may fail.

**Table 32.2. Python functions which set parameter values**

Parser function	Valid arguments
<code>set_bool</code>	Accepted values: <ul style="list-style-type: none"> <li>» yes</li> <li>» y</li> <li>» no</li> <li>» Yes</li> </ul>
<code>set_float</code>	Accepts a floating point number with Python's float(). For example: <ul style="list-style-type: none"> <li>» 3.14</li> <li>» 10.</li> <li>» .001</li> <li>» 1e100</li> <li>» 3.14e-10</li> </ul>
<code>set_int</code>	Accepts an integer with Python's int().
<code>set_value</code>	accepts any Python value.
<code>append_value</code>	accepts any Python value, and appends it to the previous value which is stored in an array.

Table 32.3. Configuration parameter default values

Parameter	Parser function	Default value
<code>name</code>	<code>setter</code>	<i>default value</i>
<code>vncpasswd</code>	<code>set_value</code>	None
<code>vncviewer</code>	<code>set_bool</code>	None
<code>vncconsole</code>	<code>set_bool</code>	None
<code>name</code>	<code>set_value</code>	None
<code>bootloader</code>	<code>set_value</code>	None
<code>bootargs</code>	<code>set_value</code>	None
<code>bootentry</code>	<code>set_value</code>	None
<code>kernel</code>	<code>set_value</code>	None
<code>ramdisk</code>	<code>set_value</code>	"
<code>features</code>	<code>set_value</code>	"
<code>builder</code>	<code>set_value</code>	'linux'
<code>memory</code>	<code>set_int</code>	128
<code>maxmem</code>	<code>set_int</code>	None
<code>shadow_memory</code>	<code>set_int</code>	0
<code>cpu</code>	<code>set_int</code>	None
<code>cpus</code>	<code>set_value</code>	None
<code>pae</code>	<code>set_int</code>	0
<code>acpi</code>	<code>set_int</code>	0
<code>apic</code>	<code>set_int</code>	0
<code>vcpus</code>	<code>set_int</code>	1
<code>cpu_weight</code>	<code>set_float</code>	None
<code>restart</code>	<code>set_value</code>	None
<code>on_poweroff</code>	<code>set_value</code>	None
<code>on_reboot</code>	<code>set_value</code>	None

Parameter	Parser function	Default value
<code>on_crash</code>	<code>set_value</code>	<code>None</code>
<code>blkif</code>	<code>set_bool</code>	<code>0</code>
<code>netif</code>	<code>set_bool</code>	<code>0</code>
<code>tpmif</code>	<code>append_value</code>	<code>0</code>
<code>disk</code>	<code>append_value</code>	<code>[]</code>
<code>pci</code>	<code>append_value</code>	<code>[]</code>
<code>ioports</code>	<code>append_value</code>	<code>[]</code>
<code>irq</code>	<code>append_value</code>	<code>[]</code>
<code>usbport</code>	<code>append_value</code>	<code>[]</code>
<code>vfb</code>	<code>append_value</code>	<code>[]</code>
<code>vif</code>	<code>append_value</code>	<code>[]</code>
<code>vtpm</code>	<code>append_value</code>	<code>[]</code>
<code>access_control</code>	<code>append_value</code>	<code>[]</code>
<code>nics</code>	<code>set_int</code>	<code>-1</code>
<code>root</code>	<code>set_value</code>	<code>"</code>
<code>extra</code>	<code>set_value</code>	<code>"</code>
<code>ip</code>	<code>set_value</code>	<code>"</code>
<code>gateway</code>	<code>set_value</code>	<code>"</code>
<code>netmask</code>	<code>set_value</code>	<code>"</code>
<code>hostname</code>	<code>set_value</code>	<code>"</code>
<code>interface</code>	<code>set_value</code>	<code>"eth0"</code>
<code>dhcp</code>	<code>set_value</code>	<code>'off'</code>
<code>nfs_server</code>	<code>set_value</code>	<code>None</code>
<code>nfs_root</code>	<code>set_value</code>	<code>None</code>
<code>device_model</code>	<code>set_value</code>	<code>"</code>
<code>fda</code>	<code>set_value</code>	<code>"</code>
<code>fdb</code>	<code>set_value</code>	<code>"</code>
<code>serial</code>	<code>set_value</code>	<code>"</code>
<code>localtime</code>	<code>set_bool</code>	<code>0</code>
<code>keymap</code>	<code>set_value</code>	<code>"</code>
<code>usb</code>	<code>set_bool</code>	<code>0</code>
<code>usbdevice</code>	<code>set_value</code>	<code>"</code>
<code>stdvga</code>	<code>set_bool</code>	<code>0</code>
<code>isa</code>	<code>set_bool</code>	<code>0</code>
<code>boot</code>	<code>set_value</code>	<code>'c'</code>
<code>nographic</code>	<code>set_bool</code>	<code>0</code>
<code>soundhw</code>	<code>set_value</code>	<code>"</code>
<code>vnc</code>	<code>set_value</code>	<code>None</code>
<code>vncdisplay</code>	<code>set_value</code>	<code>None</code>
<code>vnclisten</code>	<code>set_value</code>	<code>None</code>
<code>vncunused</code>	<code>set_bool</code>	<code>1</code>
<code>sdl</code>	<code>set_value</code>	<code>None</code>
<code>display</code>	<code>set_value</code>	<code>None</code>
<code>xauthority</code>	<code>set_value</code>	<code>None</code>
<code>uuid</code>	<code>set_value</code>	<code>None</code>

## Part VII. Tips and Tricks

### Tips and Tricks to Enhance Productivity

These chapters contain useful hints and tips to improve virtualization performance, scale and stability.

## Chapter 33. Tips and tricks

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

### 33.1. Automatically starting guests

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

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

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

The guest now automatically starts with the host.

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

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

The guest no longer automatically starts with the host.

### 33.2. Changing between the KVM and Xen hypervisors

This section covers changing between the KVM and Xen hypervisors.

Red Hat only supports one active hypervisor at a time.



#### Important

Presently, there is no application for switching Xen-based guests to KVM or KVM-based guests to Xen.



#### Warning

This procedure is only available for the Intel 64 or AMD64 version of Red Hat Enterprise Linux 5.4 or newer. No other configurations or Red Hat Enterprise Linux versions are supported. KVM is not available in versions earlier than Red Hat Enterprise Linux 5.4.

#### 33.2.1. Xen to KVM

The following procedure covers changing from the Xen hypervisor to the KVM hypervisor. This procedure assumes the *kernel-xen* package is installed and enabled.

1.

##### **Install the KVM package**

Install the *kvm* package if you have not already done so.

```
yum install kvm
```

2.

### Verify which kernel is in use

The *kernel-xen* package may be installed. Use the **uname** command to determine which kernel is running:

```
$ uname -r
2.6.18-159.el5xen
```

The present kernel, "**2.6.18-159.el5xen**", is running on the system. If the default kernel, "**2.6.18-159.el5**", is running you can skip the substep.

a.

### Changing the Xen kernel to the default kernel

The **grub.conf** file determines which kernel is booted. To change the default kernel edit the **/boot/grub/grub.conf** file as shown below.

```
default=1
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title 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
 initrd /initrd-2.6.18-159.el5.img
title Red Hat Enterprise Linux Server (2.6.18-159.el5xen)
 root (hd0,0)
 kernel /xen.gz-2.6.18-159.el5
 module /vmlinuz-2.6.18-159.el5xen ro
 root=/dev/VolGroup00/LogVol00 rhgb quiet
 module /initrd-2.6.18-159.el5xen.img
```

Notice the **default=1** parameter. This is instructing the GRUB boot loader to boot the second entry, the Xen kernel. Change the default to **0** (or the number for the default kernel):

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title 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
 initrd /initrd-2.6.18-159.el5.img
title Red Hat Enterprise Linux Server (2.6.18-159.el5xen)
 root (hd0,0)
 kernel /xen.gz-2.6.18-159.el5
 module /vmlinuz-2.6.18-159.el5xen ro
 root=/dev/VolGroup00/LogVol00 rhgb quiet
 module /initrd-2.6.18-159.el5xen.img
```

3.

## Reboot to load the new kernel

Reboot the system. The computer will restart with the default kernel. The KVM module should be automatically loaded with the kernel. Verify KVM is running:

```
$ lsmod | grep kvm
kvm_intel 85992 1
kvm 222368 2 ksm,kvm_intel
```

The **kvm** module and either the **kvm\_intel** module or the **kvm\_amd** module are present if everything worked.

### 33.2.2. KVM to Xen

The following procedure covers changing from the KVM hypervisor to the Xen hypervisor. This procedure assumes the *kvm* package is installed and enabled.

1.

#### Install the Xen packages

Install the *kernel-xen* and *xen* package if you have not already done so.

```
yum install kernel-xen xen
```

The *kernel-xen* package may be installed but disabled.

2.

#### Verify which kernel is in use

Use the **uname** command to determine which kernel is running.

```
$ uname -r
2.6.18-159.el5
```

The present kernel, "**2.6.18-159.el5**", is running on the system. This is the default kernel. If the kernel has **xen** on the end (for example, **2.6.18-159.el5xen**) then the Xen kernel is running and you can skip the substep.

a.

#### Changing the default kernel to the Xen kernel

The **grub.conf** file determines which kernel is booted. To change the default kernel edit the **/boot/grub/grub.conf** file as shown below.

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title 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
 initrd /initrd-2.6.18-159.el5.img
title Red Hat Enterprise Linux Server (2.6.18-159.el5xen)
 root (hd0,0)
```

```

kernel /xen.gz-2.6.18-159.el5
module /vmlinuz-2.6.18-159.el5xen ro
root=/dev/VolGroup00/LogVol00 rhgb quiet
module /initrd-2.6.18-159.el5xen.img

```

Notice the **default=0** parameter. This is instructing the GRUB boot loader to boot the first entry, the default kernel. Change the default to **1** (or the number for the Xen kernel):

```

default=1
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title 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
 initrd /initrd-2.6.18-159.el5.img
title Red Hat Enterprise Linux Server (2.6.18-159.el5xen)
 root (hd0,0)
 kernel /xen.gz-2.6.18-159.el5
 module /vmlinuz-2.6.18-159.el5xen ro
 root=/dev/VolGroup00/LogVol00 rhgb quiet
 module /initrd-2.6.18-159.el5xen.img

```

3.

### Reboot to load the new kernel

Reboot the system. The computer will restart with the Xen kernel. Verify with the **uname** command:

```
$ uname -r
2.6.18-159.el5xen
```

If the output has **xen** on the end the Xen kernel is running.

## 33.3. Using qemu-img

The **qemu-img** command line tool is used for formatting various file systems used by Xen and KVM. **qemu-img** should be used for formatting 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 format "**qcow**" 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 support 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.

### 33.4. Overcommitting Resources

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



#### Important

Memory overcommitting is not supported for the Xen hypervisor, however CPU overcommitting is supported.

#### 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 guests than what is physically available.

When using KVM, virtual machines operate as Linux processes. Guests on the KVM hypervisor do not have blocks of physical RAM assigned to them, instead they function as processes. Each process in a Linux system is allocated memory when it requests more memory. In a similar way, KVM allocates memory for guests when the guest 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 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 guests use their total RAM. Without sufficient memory and swap space for the virtual machine processes, the system can run completely out of memory, leading to the failure of one or more virtual machine processes.



#### 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:

$$(0.5 * \text{RAM}) + (\text{overcommit ratio} * \text{RAM}) = \text{Recommended swap size}$$

Red Hat [Knowledgebase](#) has an article on safely and efficiently determining the size of the swap partition.

Overcommitting guests by swapping out temporarily unused guest memory can be very slow, due to the IO latency introduced by disk seek times. However, Red Hat Enterprise Linux virtualization with KVM can often avoid this disk IO penalty by merging multiple pages with identical content into the same physical pages. This is done by the KSM (Kernel Samepage Merging) kernel process, which scans memory to find identical pages. The KSM kernel process uses CPU time to avoid disk IO. This tradeoff is often beneficial in workloads with many smaller, similar guests.

It is possible to run with an overcommit ratio of ten times the number of 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 guests allow. Use caution when overcommitting VCPUs as loads near 100% may cause dropped requests or unusable response times.

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

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

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



#### Important

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.

### 33.5. Modifying `/etc/grub.conf`

This section describes how to safely and correctly change your `/etc/grub.conf` file to use the virtualization kernel. You must use the `xen` kernel to use the Xen hypervisor. Copy your existing `xen` kernel entry make sure you copy all of the important lines or your system will panic upon boot (`initrd` will have a length of '`0`'). If you require `xen` hypervisor specific values you must append them to the `xen` line of your grub entry.

The output below is an example of a `grub.conf` entry from a system running the `kernel-xen` package. The `grub.conf` on your system may vary. The important part in the example below is the section from the `title` line to the next new line.

```
#boot=/dev/sda
default=0
timeout=15
#splashimage=(hd0,0)/grub/splash.xpm.gz hiddenmenu
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
```

```
terminal --timeout=10 serial console

title Red Hat Enterprise Linux Server (2.6.17-1.2519.4.21.el5xen)
root (hd0,0)
kernel /xen.gz-2.6.17-1.2519.4.21.el5 com1=115200,8n1
module /vmlinuz-2.6.17-1.2519.4.21.el5xen ro root=/dev/VolGroup00/LogVol00
module /initrd-2.6.17-1.2519.4.21.el5xen.img
```



### Note

Your **grub.conf** could look very different if it has been manually edited before or copied from an example. Read [Chapter 29, Configuring the Xen kernel boot parameters](#) for more information on using virtualization and grub.

To set the amount of memory assigned to your host system at boot time to 256MB you need to append **dom0\_mem=256M** to the **xen** line in your **grub.conf**. A modified version of the grub configuration file in the previous example:

```
#boot=/dev/sda
default=0
timeout=15
#splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console

title Red Hat Enterprise Linux Server (2.6.17-1.2519.4.21.el5xen)
root (hd0,0)
kernel /xen.gz-2.6.17-1.2519.4.21.el5 com1=115200,8n1 dom0_mem=256MB
module /vmlinuz-2.6.17-1.2519.4.21.el5xen ro
root=/dev/VolGroup00/LogVol00
module /initrd-2.6.17-1.2519.4.21.el5xen.img
```

## 33.6. 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.



### Note

If hardware virtualization extensions are not present you can use Xen para-virtualization with the Red Hat *kernel-xen* package.

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:" content may appear multiple times 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, see [Procedure 36.1, “Enabling virtualization extensions in BIOS”](#).

### 3. For users of the KVM hypervisor

If the **kvm** package is installed. 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

#### Note

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

## 33.7. 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, available from the [EPEL](#) 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 33.1. 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
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
```

- a. 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 available 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. Disconnect the image file from the partition mappings.

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

The guest can now be started again.

## 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 available 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. Disconnect the image file from the partition mappings.

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

The guest can now be restarted.

### 33.8. Setting KVM processor affinities

This section covers setting processor and processing core affinities with `libvirt` for 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>
 </host>
</capabilities>
```

```

 </cells>
</topology>
<secmodel>
 <model>selinux</model>
 <doi>0</doi>
</secmodel>
</host>

[Additional XML removed]

</capabilities>
```

The output shows two NUMA nodes (also known 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, see 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>
```

```
</cpus>
</cell>
</cells>
</topology>
```

2. Observe that the node 1, <b><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 relevant NUMA cell.

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

4. Save the configuration file and restart the guest.

The guest has been locked to CPUs 4 to 7.

### Automatically locking guests to CPUs with virt-install

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

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

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

### Tuning CPU affinity on running guests

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

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

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

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

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

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

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

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

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

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

### 33.9. Generating a new unique MAC address

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

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

#!/usr/bin/python
macgen.py script to generate a MAC address for guests on Xen
#
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.randomUUIDToString(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.randomUUIDToString(virtinst.util.randomUUID())
print "New MAC:"
import virtinst.util ; print virtinst.util.randomMAC()
print ""

```

## 33.10. Limit network bandwidth for a Xen guest

In some environments it may be required to limit the network bandwidth available to certain guests. This can be used to implement basic Quality of Service on a host running multiple virtual machines. By default, the guest can use any bandwidth setting available which your physical network card supports. The physical network card must be mapped to one of virtual machine's virtual network interfaces. In Xen the “**rate**” parameter part of the **VIF** entries can throttle guests.

This list covers the variables

### **rate**

The **rate=** option can be added to the **VIF=** entry in a virtual machine configuration file to limit a virtual machine's network bandwidth or specify a specific time interval for a time window.

### **time window**

The time window is optional to the **rate=** option:

The default time window is 50ms.

A smaller time window will provide less burst transmission, however, the replenishment rate and latency will increase.

The default 50ms time window is a good balance between latency and throughput and in most cases will not require changing.

Examples of **rate** parameter values and uses.

**rate=10Mb/s**

Limit the outgoing network traffic from the guest to 10MB/s.

**rate=250KB/s**

Limit the outgoing network traffic from the guest to 250KB/s.

**rate=10MB/s@50ms**

Limit bandwidth to 10MB/s and provide the guest with a 50KB chunk every 50ms.

In the virtual machine configuration a sample **VIF** entry would look like the following:

```
vif = ['rate=10MB/s , mac=00:16:3e:7a:55:1c, bridge=xenbr1']
```

This **rate** entry would limit the virtual machine's interface to 10MB/s for outgoing traffic

## 33.11. Configuring Xen processor affinities

Xen can allocate virtual CPUs to associate with one or more host CPUs. This allocates real processing resources to guests. This approach allows Red Hat Enterprise Linux optimize processor resources when employing dual-core, hyper-threading, or other CPU concurrency technologies. The Xen credit scheduler automatically balances virtual CPUs between physical ones, to maximize system use. Red Hat Enterprise Linux allows the credit scheduler to move CPUs around as necessary, as long as the virtual CPU is pinned to a physical CPU.

If you are running I/O intensive tasks, it is recommended to dedicate either a hyperthread or an entire processor core to run domain0.

Note that this is unnecessary for KVM as KVM uses the default Linux kernel scheduler.

CPU affinities can be set with **virsh** or **virt-manager**:

To set CPU affinities using **virsh** see [Configuring virtual CPU affinity](#) for more information.

To configure and view CPU information with **virt-manager** see [Section 27.11, “Displaying virtual CPUs”](#) for more information.

## 33.12. Modifying the Xen hypervisor

Managing host systems often involves changing the boot configuration file **/boot/grub/grub.conf**. Managing several or more hosts configuration files quickly becomes difficult. System administrators often prefer to use the 'cut and paste' method for editing multiple **grub.conf** files. If you do this, ensure you include all five lines in the Virtualization entry (or this will create system errors). Hypervisor specific values are all found on the '**xen**' line. This example represents a correct **grub.conf** virtualization entry:

```
boot=/dev/sda/
default=0
timeout=15
#splashimage=(hd0, 0)/grub/splash.xpm.gz

hiddenmenu
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console
title Red Hat Enterprise Linux Server (2.6.17-1.2519.4.21. el5xen)
```

```
root (hd0, 0)
kernel /xen.gz-2.6.17-1.2519.4.21.el5 com1=115200,8n1
 module /vmlinuz-2.6.17-1.2519.4.21.el5 xen ro root=/dev/VolGroup00/LogVol00
 module /initrd-2.6.17-1.2519.4.21.el5 xen.img
```

For example, to change the memory entry on your hypervisor (dom0) to 256MB at boot time, edit the 'xen' line and append it with this entry: '**dom0\_mem=256M**'. This example is the **grub.conf** with the hypervisor's memory entry modified.

```
boot=/dev/sda
default=0
timeout=15
#splashimage=(hd0,0)/grubs/splash.xpm.gz
hiddenmenu
serial --unit=0 --speed =115200 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console
title Red Hat Enterprise Linux Server (2.6.17-1.2519.4.21. el5xen)
root (hd0,0)
kernel /xen.gz-2.6.17-1.2519.4.21.el5 com1=115200,8n1 dom0_mem=256MB
 module /vmlinuz-2.6.17-1.2519.4.21.el5 xen ro root=/dev/VolGroup00/LogVol00
 module /initrd-2.6.17-1.2519.4.21.el5 xen.img
```

### 33.13. Very Secure vsftpd

**vsftpd** can provide access to installation trees for para-virtualized guests (for example, the Red Hat Enterprise Linux 5 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 **vi pw** 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:/xen/pub:/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]
```

### 33.14. 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
3600a0b80001327510000015427b625e
```

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="3600a0b80001327510000015427b625e", 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/mydevicename**. 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 3600a0b80001327510000015427b625e
 alias oramp1
}
multipath {
 wwid 3600a0b80001327510000015427b6
 alias oramp2
```

```

}
multipath {
 wwid 3600a0b80001327510000015427b625e
 alias oramp3
}
multipath {
 wwid 3600a0b80001327510000015427b625e
 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.

### 33.15. 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
```

### 33.16. Cleaning up old Xen configuration files

Over time you will see a number of files accumulate in **/var/lib/xen**, the usually named **vmlinuz.\*\*\*\*\*** and **initrd.\*\*\*\*\***. These files are the initrd and vmlinuz files from virtual machines which either failed to boot or failed for some other reason. These files are temporary files extracted from virtual machine's boot disk during the start up sequence. These files should be automatically removed after the virtual machine is shut down cleanly. Then you can safely delete old and stale copies from this directory.

### 33.17. 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

```

## 33.18. Cloning guest configuration files

You can copy an existing configuration file to create an all new guest. You must modify the name parameter of the guests' configuration file. The new, unique name then appears in the hypervisor and is viewable by the management utilities. You must generate an all new UUID as well by using the **uuidgen** command. Then for the **vif** entries you must define a unique MAC address for each guest (if you are copying a guest configuration from an existing guest, you can create a script to handle it). For the xen bridge information, if you move an existing guest configuration file to a new host, you must update the **xenbr** entry to match your local networking configuration. For the Device entries, you must modify the entries in the '**disk=**' section to point to the correct guest image.

You must also modify these system configuration settings on your guest. You must modify the HOSTNAME entry of the **/etc/sysconfig/network** file to match the new guest's hostname.

You must modify the **HWADDR** address of the **/etc/sysconfig/network-scripts/ifcfg-eth0** file to match the output from **ifconfig eth0** file and if you use static IP addresses, you must modify the **IPADDR** entry.

## 33.19. Duplicating an existing guest and its configuration file

This section outlines copying an existing configuration file to create a new guest. There are key parameters in your guest's configuration file you must be aware of, and modify, to successfully duplicate a guest.

### **name**

The name of your guest as it is known to the hypervisor and displayed in the management utilities. This entry should be unique on your system.

### **uuid**

A unique handle for the guest, a new UUID can be regenerated using the **uuidgen** command. A sample UUID output:

```

$ uuidgen
a984a14f-4191-4d14-868e-329906b211e5

```

### **vif**

- The MAC address must define a unique MAC address for each guest. This is automatically done if the standard tools are used. If you are copying a guest configuration from an existing guest you can use the script [Section 33.9, “Generating a new unique MAC address”](#).
- If you are moving or duplicating an existing guest configuration file to a new host you have to make sure you adjust the **xenbr** entry to correspond with your local networking configuration (you can obtain the bridge information using the command **brctl show** command).
- Device entries, make sure you adjust the entries in the **disk=** section to point to the correct guest image.

Now, adjust the system configuration settings on your guest:

**/etc/sysconfig/network**

Modify the **HOSTNAME** entry to the guest's new **hostname**.

**/etc/sysconfig/network-scripts/ifcfg-eth0**

- ✖ Modify the **HWADDR** address to the output from **ifconfig eth0**
- ✖ Modify the **IPADDR** entry if a static IP address is used.

## Chapter 34. 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 33, \*Tips and tricks\*](#) is recommended reading for programmers thinking of writing new applications which use **libvirt**.

### 34.1. 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 VIII. Troubleshooting

### Introduction to Troubleshooting and Problem Solving

The following chapters provide information to assist you in troubleshooting issues you may encounter using virtualization.



#### Note

Your particular problem may not appear in this book due to ongoing development which creates and fixes bugs. For the most up to date list of known bugs, issues and bug fixes read the Red Hat Enterprise Linux *Release Notes* for your version and hardware architecture. The *Release Notes* can be found in the documentation section of the Red Hat website, <https://access.redhat.com/site/documentation/>.



#### Note

Contact Red Hat Global Support Services (<https://www.redhat.com/support>). Staff can assist you in resolving your issues.

# Chapter 35. Troubleshooting Xen

This chapter covers essential concepts to assist you in troubleshooting problems in Xen. Troubleshooting topics covered in this chapter include:

- » troubleshooting tools for Linux and virtualization.
- » troubleshooting techniques for identifying problems.
- » The location of log files and explanations of the information in logs.

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 Red Hat Enterprise Linux to develop your troubleshooting skills.

If you cannot find the answer in this document there may be an answer online from the virtualization community. See [Section A.1, “Online resources”](#) for a list of Linux virtualization websites.

## 35.1. Debugging and troubleshooting Xen

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:

### Useful commands and applications for troubleshooting

#### **xentop**

**xentop** displays real-time information about a host system and the guest domains.

#### **xm**

Using the **dmesg** and **log**

- » **vmstat**
- » **iostat**
- » **lsof**

The **iostat**, **mpstat** and **sar** commands are all provided by the **sysstat** package.

You can employ these Advanced Debugging Tools and logs to assist with troubleshooting:

- » **XenOprofile**
- » **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

xenbr0 8000.ffffffff no vif13.0
xenbr1 8000.fffffeff yes pddummy0
xenbr2 8000.fffffffef no vif0.0

brctl showmacs xenbr0

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 xenbr0
xenbr0
bridge-id 8000.ffffffffffff
designated-root 8000.ffffffffffff
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
tcn-timer 0.00
hello-timer 1.43
gc-timer 0.02
topology-change-timer 0.00
```

Listed below are some other useful commands for troubleshooting virtualization on Red Hat Enterprise Linux 5. All utilities mentioned can be found in the **Server** repositories Red Hat Enterprise Linux 5.

- » **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 **vncviewer** 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.

## 35.2. Log files overview

When deploying Red Hat Enterprise Linux 5 with Virtualization into your network infrastructure, the host's Virtualization software uses many specific directories for important configuration, log files, and other utilities. All the Xen logs files are standard ASCII files, and accessible with a text editor:

- » The Xen configuration directory is **/etc/xen/**. This directory contains the **xend** daemon and other virtual machine configuration files. The networking script files are found in the **scripts** directory.
- » All Xen log files are stored in the **/var/log/xen** directory.
- » The default directory for all file-based images is the **/var/lib/libvirt/images** directory.
- » Xen kernel information is stored in the **/proc/xen/** directory.

### 35.3. Log file descriptions

Xen features the **xend** daemon and **qemu-dm** process, two utilities that write the multiple log files to the **/var/log/xen/** directory:

- » **xend.log** is the log file that contains all the data collected by the **xend** daemon, whether it is a normal system event, or an operator initiated action. All virtual machine operations (such as create, shutdown, destroy and so on) appear in this log. The **xend.log** is usually the first place to look when you track down event or performance problems. It contains detailed entries and conditions of the error messages.
- » **xend-debug.log** is the log file that contains records of event errors from **xend** and the Virtualization subsystems (such as framebuffer, Python scripts, and so on).
- » **xen-hotplug-log** is the log file that contains data from hotplug events. If a device or a network script does not come online, the event appears here.
- » **qemu-dm.[PID].log** is the log file created by the **qemu-dm** process for each fully virtualized guest. When using this log file, you must retrieve the given **qemu-dm** process PID, by using the **ps** command to examine process arguments to isolate the **qemu-dm** process on the virtual machine. Note that you must replace the [PID] symbol with the actual PID **qemu-dm** 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.

### 35.4. Important directory locations

There are other utilities and log files you should be aware of for tracking errors and troubleshooting problems with Xen:

- » Guest images reside in the **/var/lib/libvirt/images** directory by default.
- » When you restart the **xend** daemon, it updates the **xend-database** that resides in the **/var/lib/xen/xend-db** directory.
- » Virtual machine dumps (that you perform with the **xm dump-core** command) resides in the **/var/lib/xen/dumps** directory.

- » The **/etc/xen** directory contains the configuration files that you use to manage system resources. The **xend** daemon configuration file is **/etc/xen/xend-config.sxp**. This file can be edited to implement system-wide changes and configure the networking. However, manually editing files in the **/etc/xen/** folder is not advised.
- » The **proc** folders are another resource that allows you to gather system information. These proc entries reside in the **/proc/xen** directory:

**/proc/xen/capabilities**

**/proc/xen/balloon**

**/proc/xen/xenbus/**

## 35.5. Troubleshooting with the logs

When encountering installation issues with Xen, check the host system's two logs to assist with troubleshooting. The **xend.log** file contains the same basic information as when you run the **xm log** command. This log is found in the **/var/log/** directory. Here is an example log entry for when you create a domain running a kernel:

```
[2006-12-27 02:23:02 xend] ERROR (SrvBase: 163) op=create: Error creating domain: (0, 'Error')
Traceback (most recent call list)
File "/usr/lib/python2.4/site-packages/xen/xend/server/SrvBase.py" line 107
in_perform val = op_method (op,req)
File
"/usr/lib/python2.4/site-packages/xen/xend/server/SrvDomainDir.py line 71 in
op_create
raise XendError ("Error creating domain: " + str(ex))
XendError: Error creating domain: (0, 'Error')
```

The other log file, **xend-debug.log**, is very useful to system administrators since it contains even more detailed information than **xend.log**. Here is the same error data for the same kernel domain creation problem:

```
ERROR: Will only load images built for Xen v3.0
ERROR: Actually saw: GUEST_OS=netbsd, GUEST_VER=2.0, XEN_VER=2.0; LOADER=generic,
BSD_SYMTAB'
ERROR: Error constructing guest OS
```

When calling customer support, always include a copy of both these log files when contacting the technical support staff.

## 35.6. Troubleshooting with the serial console

The serial console is helpful in troubleshooting difficult problems. If the Virtualization kernel crashes and the hypervisor generates an error, there is no way to track the error on a local host. However, the serial console allows you to capture it on a remote host. You must configure the host to output data to the serial console. Then you must configure the remote host to capture the data. To do this, you must modify these options in the **grub.conf** file to enable a 38400-bps serial console on **com1 /dev/ttys0**:

```
title Red Hat Enterprise Linux (2.6.18-8.2080_xen0)
root (hd0,2)
kernel /xen.gz-2.6.18-8.el5 com1=38400,8n1
```

```
module /vmlinuz-2.6.18-8.el5xen ro root=LABEL=/rhgb quiet console=xvc console=tty
xencons=xvc
module /initrd-2.6.18-8.el5xen.img
```

The **sync\_console** can help determine a problem that causes hangs with asynchronous hypervisor console output, and the "**pnpacpi=off**" works around a problem that breaks input on the serial console. The parameters "**console=ttyS0**" and "**console=tty**" means that kernel errors get logged with on both the normal VGA console and on the serial console. Then you can install and set up **ttywatch** to capture the data on a remote host connected by a standard null-modem cable. For example, on the remote host you could type:

### Note

To access the hypervisor via a serial console on the Itanium® architecture you must enable the console in ELILO. For more information on configuring ELILO, see [Chapter 30, Configuring ELILO](#).

```
ttywatch --name myhost --port /dev/ttyS0
```

This pipes the output from **/dev/ttyS0** into the file **/var/log/ttywatch/myhost.log**.

## 35.7. Para-virtualized guest console access

Para-virtualized guest operating systems automatically has a virtual text console configured to transmit data to the host operating system. Connect to a guest's virtual console with the following command:

```
virsh console [guest name, ID or UUID]
```

You can also use **virt-manager** to display the virtual text console. In the guest console window, select **Serial Console** from the **View** menu.

## 35.8. Fully virtualized guest console access

Fully virtualized guest operating systems automatically has a text console configured for use, but the difference is the kernel guest is not configured. To enable the guest virtual serial console to work with the fully virtualized guest, you must modify the guest's **grub.conf** file, and include the '**console=ttyS0 console=tty0**' parameter. This ensures that the kernel messages are sent to the virtual serial console (and the normal graphical console). To use the guest's serial console, you must edit the libvirt configuration file. 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.

## 35.9. Common Xen problems

When attempting to start the **xend** service, nothing happens. Type **virsh list** and receive the following:

Error: Error connecting to xend: Connection refused. Is xend running?

Try to run **xend start** manually and receive more errors:

```
Error: Could not obtain handle on privileged command interfaces (2 = No such file or
directory)
Traceback (most recent call last):
File "/usr/sbin/xend/", line 33 in ?
from xen.xend.server. import SrvDaemon
File "/usr/lib/python2.4/site-packages/xen/xend/server/SrvDaemon.py" , line 26 in ?
from xen.xend import XendDomain
File "/usr//lib/python2.4/site-packages/xen/xend/XendDomain.py" , line 33, in ?
from xen.xend import XendDomainInfo
File "/usr/lib/python2.4/site-packages/xen/xend/image.py" , line37, in ?
import images
File "/usr/lib/python2.4/site-packages/xen/xend/image.py" , line30, in ?
xc = xen.lowlevel.xc.xc ()
RuntimeError: (2, 'No such file or directory')
```

What has most likely happened here is that you rebooted your host into a kernel that is not a **kernel-xen** kernel. To correct this, you must select the **kernel-xen** kernel at boot time (or set the **kernel-xen** kernel to the default in the **grub.conf** file).

## 35.10. Guest creation errors

When you attempt to create a guest, you receive an "**Invalid argument**" error message. This usually means that the kernel image you are trying to boot is incompatible with the hypervisor. An example of this would be if you were attempting to run a non-PAE FC5 kernel on a PAE only FC6 hypervisor.

You do a yum update and receive a new kernel, the **grub.conf** default kernel switches right back to a bare-metal kernel instead of the Virtualization kernel.

To correct this problem you must modify the default kernel RPM that resides in the **/etc/sysconfig/kernel/** directory. You must ensure that **kernel-xen** parameter is set as the default option in your **grub.conf** file.

## 35.11. Troubleshooting with serial consoles

Linux kernels can output information to serial ports. This is useful for debugging kernel panics and hardware issues with video devices or headless servers. The subsections in this section cover setting up serial console output for machines running Red Hat Enterprise Linux virtualization kernels and their guests.

### 35.11.1. Serial console output for Xen

By default, the serial console for the Xen hypervisor is disabled and no data is output from serial ports.

To receive kernel information on a serial port modify the **/boot/grub/grub.conf** file by setting the appropriate serial device parameters.

If your serial console is on **com1**, modify **/boot/grub/grub.conf** by inserting the lines **com1=115200,8n1**, **console=tty0** and **console=ttyS0,115200** where shown.

```

title Red Hat Enterprise Linux 5 i386 Xen (2.6.18-92.el5xen)
root (hd0, 8)
kernel /boot/xen.gz-2.6.18-92.el5 com1=115200,8n1
module /boot/vmlinuz-2.6.18-92.el5xen ro root=LABEL=VG_i386 console=tty0
console=ttyS0,115200
module /boot/initrd-2.6.18-92.el5xen.img

```

If your serial console is on **com2**, modify **/boot/grub/grub.conf** by inserting the lines **com2=115200,8n1** **console=com2L**, **console=tty0** and **console=ttyS0,115200** where shown.

```

title Red Hat Enterprise Linux 5 i386 Xen (2.6.18-92.el5xen)
root (hd0, 8)
kernel /boot/xen.gz-2.6.18-92.el5 com2=115200,8n1 console=com2L
module /boot/vmlinuz-2.6.18-92.el5xen ro root=LABEL=VG_i386 console=tty0
console=ttyS0,115200
module /boot/initrd-2.6.18-92.el5xen.img

```

Save the changes and reboot the host. The hypervisor outputs serial data on the serial (**com1**, **com2** and so on) you selected in the previous step.

Note the example using the **com2** port, the parameter **console=ttyS0** on the **vmlinuz** line us used. The behavior of every port being used as **console=ttyS0** is not standard Linux behavior and is specific to the Xen environment.

### 35.11.2. Xen serial console output from para-virtualized guests

This section describes how to configure a virtualized serial console for Red Hat Enterprise Linux para-virtualized guests.

Serial console output from para-virtualized guests can be received using the "**virsh console**" or in the "**Serial Console**" window of **virt-manager**. Set up the virtual serial console using this procedure:

1. Log in to your para-virtualized guest.
2. Edit **/boot/grub/grub.conf** as follows:

```

Red Hat Enterprise Linux 5 i386 Xen (2.6.18-92.el5xen)
root (hd0, 0) kernel /boot/vmlinuz-2.6.18-92.el5xen ro root=LABEL=VG_i386
console=xvc0
initrd /boot/initrd-2.6.18-92.el5xen.img

```

3. Reboot the para-virtualized guest.

You should now get kernel messages on the virt-manager "Serial Console" and "virsh console".

#### Logging the para-virtualized domain serial console output

The Xen daemon(**xend**) can be configured to log the output from serial consoles of para-virtualized guests.

To configure **xend** edit **/etc/sysconfig/xend**. Change the entry:

```

Log all guest console output (cf xm console)
#XENCONSOLED_LOG_GUESTS=no

```

to:

```
Log all guest console output (cf xm console)
XENCONSOLED_LOG_GUESTS=yes
```

Reboot the host to activate logging the guest serial console output.

Logs from the guest serial consoles are stored in the **/var/log/xen/console** file.

### 35.11.3. Serial console output from fully 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:

- » logging output with xend is unavailable.
- » 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 Red Hat Enterprise Linux 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.

View the serial console messages using the "**virsh console**" command.



#### Note

Serial console messages from fully virtualized domains are not logged in **/var/log/xen/console** as they are for para-virtualized guests.

## 35.12. Xen configuration files

When you create guests with the **virt-manager** or **virt-install** tools on Red Hat Enterprise Linux 5, the guests configuration files are created automatically in the **/etc/xen** directory.



#### Warning

Red Hat advises users not to manually edit Xen configuration files. Xen configuration files have limited error checking and many unsupported variables. Editing Xen configuration files may damage your guests, make guests unbootable or cause data loss.

The example below is a typical a para-virtualized guest configuration file:

```
name = "rhel5vm01"
memory = "2048"
disk = ['tap:aio:/var/lib/libvirt/images/rhel5vm01.dsk,xvda,w',]
vif = ["type=ieomu, mac=00:16:3e:09:f0:12 bridge=xenbr0",
 "type=ieomu, mac=00:16:3e:09:f0:13"]
vnc = 1
vncunused = 1
uuid = "302bd9ce-4f60-fc67-9e40-7a77d9b4e1ed"
bootloader = "/usr/bin/pygrub"
vcpus=2
on_reboot = "restart"
on_crash = "restart"
```

Note that the **serial="pty"** is the default for the configuration file. This configuration file example is for a fully-virtualized guest:

```
name = "rhel5u5-86_64"
builder = "hvm"
memory = 500
disk = ['/var/lib/libvirt/images/rhel5u5-x86_64.dsk.hda,w']
vif = ['type=ioemu, mac=00:16:3e:09:f0:12, bridge=xenbr0', 'type=ieomu,
mac=00:16:3e:09:f0:13, bridge=xenbr1']
uuid = "b10372f9-91d7-ao5f-12ff-372100c99af5"
device_model = "/usr/lib64/xen/bin/qemu-dm"
kernel = "/usr/lib/xen/boot/hvmloader/"
vnc = 1
vncunused = 1
apic = 1
acpi = 1
pae = 1
vcpus =1
serial ="pty" # enable serial console
on_boot = 'restart'
```

### Note

Editing Xen configuration files is unsupported. Use **virsh dumpxml** and **virsh create** (or **virsh edit**) to edit the **libvirt** configuration files (xml based) which have error checking and safety checks.

## 35.13. Interpreting Xen error messages

You receive the following error:

```
failed domain creation due to memory shortage, unable to balloon domain0
```

A domain can fail if there is not enough RAM available. Domain0 does not balloon down enough to provide space for the newly created guest. You can check the **xend.log** file for this error:

```
[2006-12-21] 20:33:31 xend 3198] DEBUG (balloon:133) Balloon: 558432 Kib free; 0 to
scrub; need 1048576; retries: 20
[2006-12-21] 20:33:31 xend. XendDomainInfo 3198] ERROR (XendDomainInfo: 202
Domain construction failed
```

You can check the amount of memory in use by domain0 by using the `xm list domain0` command. If dom0 is not ballooned down, you can use the command `virsh setmem dom0 NewMemSize` to check memory.

You receive the following error:

```
wrong kernel image: non-PAE kernel on a PAE
```

This message indicates that you are trying to run an unsupported guest kernel image on your hypervisor. This happens when you try to boot a non-PAE, para-virtualized guest kernel on a Red Hat Enterprise Linux 5 host. The Red Hat `kernel-xen` package only supports guest kernels with PAE and 64 bit architectures.

Type this command:

```
xm create -c va-base

Using config file "va-base"
Error: (22, 'invalid argument')
[2006-12-14 14:55:46 xend.XendDomainInfo 3874] ERRORS
(XendDomainInfo:202) Domain construction failed

Traceback (most recent call last)
File "/usr/lib/python2.4/site-packages/xen/xend/XendDomainInfo.py", line 195 in
create vm.initDomain()
File "/usr/lib/python2.4/site-packages/xen/xend/XendDomainInfo.py", line 1363 in
initDomain raise VmError(str(exn))
VmError: (22, 'Invalid argument')
[2006-12-14 14:55:46 xend.XendDomainInfo 3874] DEBUG (XenDomainInfo: 1449)
XendDomainInfo.destroy: domain=1
[2006-12-14 14:55:46 xend.XendDomainInfo 3874] DEBUG (XenDomainInfo: 1457)
XendDomainInfo.destroy:Domain(1)
```

If you need to run a 32 bit non-PAE kernel you will need to run your guest as a fully virtualized virtual machine. For para-virtualized guests, if you need to run a 32 bit PAE guest, then you must have a 32 bit PAE hypervisor. For para-virtualized guests, to run a 64 bit PAE guest, then you must have a 64 bit PAE hypervisor. For full virtualization guests you must run a 64 bit guest with a 64 bit hypervisor. The 32 bit PAE hypervisor that comes with Red Hat Enterprise Linux 5 i686 only supports running 32 bit PAE para virtualized and 32 bit fully virtualized guest OSes. The 64 bit hypervisor only supports 64 bit para-virtualized guests.

This happens when you move the full virtualized HVM guest onto a Red Hat Enterprise Linux 5 system. Your guest may fail to boot and you will see an error in the console screen. Check the PAE entry in your configuration file and ensure that `pae=1`. You should use a 32 bit distribution.

You receive the following error:

```
Unable to open a connection to the Xen hypervisor or daemon
```

This happens when the virt-manager application fails to launch. This error occurs when there is no localhost entry in the `/etc/hosts` configuration file. Check the file and verify if the localhost entry is enabled. Here is an example of an incorrect localhost entry:

```
Do not remove the following line, or various programs
that require network functionality will fail.
localhost.localdomain localhost
```

Here is an example of a correct localhost entry:

```
Do not remove the following line, or various programs
that require network functionality will fail.
127.0.0.1 localhost.localdomain localhost
localhost.localdomain. localhost
```

You receive the following error (in the **xen-xend.logfile** ):

```
Bridge xenbr1 does not exist!
```

This happens when the guest's bridge is incorrectly configured and this forces the Xen hotplug scripts to timeout. If you move configuration files between hosts, you must ensure that you update the guest configuration files to reflect network topology and configuration modifications. When you attempt to start a guest that has an incorrect or non-existent Xen bridge configuration, you will receive the following errors:

```
xm create mySQL01

Using config file " mySQL01"
Going to boot Red Hat Enterprise Linux Server (2.6.18.-1.2747 .el5xen)
kernel: /vmlinuz-2.6.18-12747.el5xen
initrd: /initrd-2.6.18-1.2747.el5xen.img
Error: Device 0 (vif) could not be connected. Hotplug scripts not working.
```

In addition, the **xend.log** displays the following errors:

```
[2006-11-14 15:07:08 xend 3875] DEBUG (DevController:143) Waiting for devices vif
[2006-11-14 15:07:08 xend 3875] DEBUG (DevController:149) Waiting for 0
[2006-11-14 15:07:08 xend 3875] DEBUG (DevController:464) hotplugStatusCallback
/local/domain/0/backend/vif/2/0/hotplug-status

[2006-11-14 15:08:09 xend.XendDomainInfo 3875] DEBUG (XendDomainInfo:1449)
XendDomainInfo.destroy: domid=2
[2006-11-14 15:08:09 xend.XendDomainInfo 3875] DEBUG (XendDomainInfo:1457)
XendDomainInfo.destroyDomain(2)
[2006-11-14 15:07:08 xend 3875] DEBUG (DevController:464) hotplugStatusCallback
/local/domain/0/backend/vif/2/0/hotplug-status
```

To resolve this problem, open the guest's configuration file found in the **/etc/xen** directory. For example, editing the guest **mySQL01**

```
vim /etc/xen/mySQL01
```

Locate the **vif** entry. Assuming you are using **xenbr0** as the default bridge, the proper entry should resemble the following:

```
vif = ['mac=00:16:3e:49:1d:11, bridge=xenbr0' ,]
```

You receive these python deprecation errors:

```
xm shutdown win2k3xen12
xm create win2k3xen12

Using config file "win2k3xen12".

/usr/lib64/python2.4/site-packages/xenxm/opts.py:520: Deprecation Warning:
Non ASCII character '\xc0' in file win2k3xen12 on line 1, but no encoding
```

```
declared; see http://www.python.org/peps/pep-0263.html for details
```

```
execfile (defconfig, globs, locs,)
Error: invalid syntax 9win2k3xen12, line1
```

Python generates these messages when an invalid (or incorrect) configuration file. To resolve this problem, you must modify the incorrect configuration file, or you can generate a new one.

## 35.14. The layout of the log directories

The basic directory structure in a Red Hat Enterprise Linux 5 Virtualization environment is as follows:

### /etc/xen/ directory contains

- » configuration files used by the **xend** daemon.
- » the **scripts** directory which contains the scripts for Virtualization networking.

### /var/log/xen/

- » directory holding all Xen related log files.

### /var/lib/libvirt/images/

- » The default directory for virtual machine image files.
- » 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.

### /proc/xen/

- » The xen related information in the /proc file system.

## Chapter 36. Troubleshooting

This chapter covers common problems and solutions with Red Hat Enterprise Linux virtualization.

### 36.1. Identifying available storage and partitions

Verify the block driver is loaded and the devices and partitions are available to the guest. This can be done by executing "**cat /proc/partitions**" as seen below.

```
cat /proc/partitions
major minor #blocks name
 202 16 104857600 xvdb
 3 0 8175688 hda
```

### 36.2. After rebooting Xen-based guests the console freezes

Occasionally, a Xen guest's console freezes when the guest boots. The console still displays messages but the user cannot log in.

To fix this issue add the following line to the **/etc/inittab** file:

```
1:12345:respawn:/sbin/mingetty xvc0
```

After saving the file, reboot. The console session should now be interactive as expected.

### 36.3. Virtualized Ethernet devices are not found by networking tools

The networking tools cannot identify the **Xen Virtual Ethernet** networking card inside the guest operation system. Verify this by executing the following (for Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5):

```
cat /etc/modprobe.conf
```

Or (for Red Hat Enterprise Linux 3):

```
cat /etc/modules.conf
```

The output should contain the line and a similar line for each additional interface.

```
alias eth0 xen-vnif
```

To fix this problem you will need to add the aliasing lines (for example, **alias eth0 xen-vnif**) for every para-virtualized interface for the guest.

### 36.4. 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.

## 36.5. Failed domain creation caused by a memory shortage

This may cause a domain to fail to start. The reason for this is there is not enough memory available or **dom0** has not ballooned down enough to provide space for a recently created or started guest. In your **/var/log/xen/xend.log**, an example error message indicating this has occurred:

```
[2006-11-21 20:33:31 xend 3198] DEBUG (balloon:133) Balloon: 558432 KiB free;
0 to scrub; need 1048576; retries: 20.
[2006-11-21 20:33:52 xend.XendDomainInfo 3198] ERROR (XendDomainInfo:202) Domain
construction failed
```

You can verify the amount of memory currently used by **dom0** with the command “**xm list Domain-0**”. If **dom0** is not ballooned down you can use the command “**xm mem-set Domain-0 NewMemSize**” where **NewMemSize** should be a smaller value.

## 36.6. Wrong kernel image error

Para-virtualized guests cannot use the kernel-xen kernel. Use only the standard kernel for para-virtualized guests.

Attempting to boot any kernel other than the Xen kernel as a para-virtualized guest results in the following error message:

```
xm create testVM
Using config file "./testVM".
Going to boot Red Hat Enterprise Linux Server (2.6.18-1.2839.el5)
kernel: /vmlinuz-2.6.18-1.2839.el5
initrd: /initrd-2.6.18-1.2839.el5.img
Error: (22, 'Invalid argument')
```

In the above error you can see that the kernel line shows that the system is trying to boot a non kernel-xen kernel. The correct entry in the example is "**kernel: /vmlinuz-2.6.18-1.2839.el5xen**".

The solution is to verify you have indeed installed a kernel-xen in your guest and it is the default kernel to boot in your **/etc/grub.conf** configuration file.

If you have **kernel-xen** installed in your guest you can start your guest:

```
xm create -c GuestName
```

Where **GuestName** is the name of the guest. The previous command will present you with the **GRUB** boot loader screen and allow you to select the kernel to boot. You will have to choose the kernel-xen kernel to boot. Once the guest has completed the boot process you can log into the guest and edit **/etc/grub.conf** to change the default boot kernel to your kernel-xen. Change the line

"**default=X**" (where X is a number starting at '**0**') to correspond to the entry with your kernel-xen line. The numbering starts at '**0**' so if your kernel-xen entry is the second entry you would enter '**1**' as the default, for example "**default=1**".

## 36.7. Wrong kernel image error - non-PAE kernel on a PAE platform

If you try to boot a non-PAE kernel, para-virtualized guest the error message below will display. It indicates you are trying to run a guest kernel on your Hypervisor which at this time is not supported. The Xen hypervisor presently only supports PAE and 64 bit para-virtualized guest kernels.

```
xm create -c va-base
Using config file "va-base".
Error: (22, 'Invalid argument')
[2006-12-14 14:55:46 xend.XendDomainInfo 3874] ERROR (XendDomainInfo:202) Domain
construction failed
Traceback (most recent call last):
File "/usr/lib/python2.4/site-packages/xen/xend/XendDomainInfo.py",
line 195, in create vm.initDomain()
File "/usr/lib/python2.4/site-packages/xen/xend/XendDomainInfo.py",
line 1363, in initDomain raise VmError(str(exn))
VmError: (22, 'Invalid argument')
[2006-12-14 14:55:46 xend.XendDomainInfo 3874] DEBUG (XendDomainInfo:1449)
XendDomainInfo.destroy: domid=1
[2006-12-14 14:55:46 xend.XendDomainInfo 3874] DEBUG (XendDomainInfo:1457)
XendDomainInfo.destroyDomain(1)
```

If you need to run a 32 bit or non-PAE kernel you will need to run your guest as a fully-virtualized virtual machine. The rules for hypervisor compatibility are:

- » para-virtualized guests must match the architecture type of your hypervisor. To run a 32 bit PAE guest you must have a 32 bit PAE hypervisor.
- » to run a 64 bit para-virtualized guest your Hypervisor must be a 64 bit version too.
- » fully virtualized guests your hypervisor must be 32 bit or 64 bit for 32 bit guests. You can run a 32 bit (PAE and non-PAE) guest on a 32 bit or 64 bit hypervisor.
- » to run a 64 bit fully virtualized guest your hypervisor must be 64 bit too.

## 36.8. Fully-virtualized 64 bit guest fails to boot

If you have moved the configuration file to a Red Hat Enterprise Linux 5 causing your fully-virtualized guest to fail to boot and present the error, **Your CPU does not support long mode. Use a 32 bit distribution.** This problem is caused by a missing or incorrect **pae** setting. Ensure you have an entry "**pae=1**" in your guest's configuration file.

## 36.9. A missing localhost entry causes virt-manager to fail

The **virt-manager** application may fail to launch and display an error such as "**Unable to open a connection to the Xen hypervisor/daemon**". This is usually caused by a missing **localhost** entry in the **/etc/hosts** file. Verify that you indeed have a **localhost** entry and if it is missing from **/etc/hosts** and insert a new entry for **localhost** if it is not present. An incorrect **/etc/hosts** may resemble the following:

```
Do not remove the following line, or various programs
that require network functionality will fail.
localhost.localdomain localhost
```

The correct entry should look similar to the following:

```
Do not remove the following line, or various programs
that require network functionality will fail.
127.0.0.1 localhost.localdomain localhost
localhost.localdomain localhost
```

## 36.10. Microcode error during guest boot

During the boot phase of your virtual machine you may see an error message similar to:

```
Applying Intel CPU microcode update: FATAL: Module microcode not found.
ERROR: Module microcode does not exist in /proc/modules
```

As the virtual machine is running on virtual CPUs there is no point updating the microcode. Disabling the microcode update for your virtual machines will stop this error:

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

## 36.11. Python depreciation warning messages when starting a virtual machine

Sometimes Python will generate a message like the one below, these are often caused by either an invalid or incorrect configuration file. A configuration file containing non-ascii characters will cause these errors. The solution is to correct the configuration file or generate a new one.

Another cause is an incorrect configuration file in your current working directory. “**xm create**” will look in the current directory for a configuration file and then in **/etc/xen**

```
xm shutdown win2k3xen12
xm create win2k3xen12
Using config file "win2k3xen12".
/usr/lib64/python2.4/site-packages/xen/xm/opts.py:520: DeprecationWarning:
Non-ASCII character '\xc0' in file win2k3xen12 on line 1, but no encoding
declared; see http://www.python.org/peps/pep-0263.html for details
execfile(defconfig, globs, locs)
Error: invalid syntax (win2k3xen12, line 1)
```

## 36.12. 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. See [Section 36.12, “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 36.1. 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**

Many of the steps below may vary depending on your motherboard, processor type, chipset and OEM. See 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.

### **36.13. 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 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](#))

# Chapter 37. Troubleshooting the Xen para-virtualized drivers

This chapter deals with issues you may encounter with Xen hosts and fully virtualized Red Hat Enterprise Linux guests using the para-virtualized drivers.

## 37.1. Red Hat Enterprise Linux 5 Virtualization log file and directories

### Red Hat Enterprise Linux 5 Virtualization related log file

In Red Hat Enterprise Linux 5, the log file written by the **xend** daemon and the **qemu-dm** process are all kept in the following directories:

**/var/log/xen/**

directory holding all log file generated by the **xend** daemon and qemu-dm process.

**xend.log**

- » This logfile is used by xend to log any events generate by either normal system events or operator initiated events.
- » virtual machine operations such as create, shutdown, destroy etc are all logged in this logfile.
- » Usually this logfile will be the first place to look at in the event of a problem. In many cases you will be able to identify the root cause by scanning the logfile and review the entries logged just prior to the actual error message.

**xend-debug.log**

- » Records error events from xend and its subsystems (from the framebuffer and Python scripts)

**xen-hotplug.log**

- » Logs events from hotplug events.
- » Event notifications from devices not coming online or network bridges not online are logged in this file.

**qemu-dm.*PID*.log**

- » This file is created by the **qemu-dm** process which is started for each fully virtualized guest.
- » The **PID** is replaced with the **PID** of the process of the related qemu-dm process
- » You can retrieve the **PID** for a given **qemu-dm** process using the **ps** command and in looking at the process arguments you can identify the virtual machine the **qemu-dm** process belongs to.

If you are troubleshooting a problem with the **virt-manager** application you can also review the logfile generated by it. The logfile for **virt-manager** will be in a directory called **.virt-manager** in the user's **home** directory whom ran **virt-manager**. This directory will usually be **~/.virt-manager/virt-manager**.



## Note

The logfile is overwritten every time you start **virt-manager**. If you are troubleshooting a problem with **virt-manager** make sure you save the logfile before you restart **virt-manager** after an error has occurred.

### Red Hat Enterprise Linux 5 Virtualization related directories

There are a few other directories and files which may be of interest when troubleshooting a Red Hat Enterprise Linux 5 Virtualization environment:

#### **/var/lib/libvirt/images/**

the standard directory for file-based guest images.

#### **/var/lib/xen/xend-db/**

directory that hold the xend database which is generated every time the daemon is restarted.

#### **/etc/xen/**

Stores a number of configuration files for the Xen hypervisor.

- » **/etc/xen/xend-config.sxp** is the main configuration for the xend daemon. The **xend-config.sxp** file enables or disables migration and other features not configured by **libvirt**. Use the **libvirt** tools for all other features.

#### **/var/lib/xen/dump/**

Holds dumps generate by virtual machines or when using the **xm dump-core** command.

#### **/proc/xen/**

Stores **xen-kernel** information in the following files:

- » **/proc/xen/capabilities**
- » **/proc/xen/privcmd**
- » **/proc/xen/balloon**
- » **/proc/xen/xenbus**
- » **/proc/xen/xsd\_port**
- » **/proc/xen/xsd\_kva**

## 37.2. Para-virtualized guest fail to load on a Red Hat Enterprise Linux 3 guest operating system

Red Hat Enterprise Linux 3 is uses processor architecture specific kernel RPMs and because of this the para-virtualized drivers may fail to load if the para-virtualized driver RPM does not match the installed kernel architecture.

When the para-virtualized driver modules are inserted, a long list of unresolved modules will be

displayed. A shortened excerpt of the error can be seen below.

```
insmod xen-platform-pci.o
Warning: kernel-module version mismatch
xen-platform-pci.o was compiled for kernel version 2.4.21-52.EL
while this kernel is version 2.4.21-50.EL
xen-platform-pci.o: unresolved symbol __ioremap_R9eac042a
xen-platform-pci.o: unresolved symbol flush_signals_R50973be2
xen-platform-pci.o: unresolved symbol pci_read_config_byte_R0e425a9e
xen-platform-pci.o: unresolved symbol __get_free_pages_R9016dd82
[...]
```

The solution is to use the correct RPM package for your hardware architecture for the para-virtualized drivers.

### 37.3. A warning message is displayed while installing the para-virtualized drivers on Red Hat Enterprise Linux 3

Installing the para-virtualized drivers on a Red Hat Enterprise Linux 3 kernel prior to 2.4.21-52 may result in a warning message being displayed stating the modules have been compiled with a newer version than the running kernel.

This message, as seen below, can be safely ignored.

```
Warning: kernel-module version mismatch
xen-platform-pci.o was compiled for kernel version 2.4.21-52.EL
while this kernel is version 2.4.21-50.EL
Warning: loading xen-platform-pci.o will taint the kernel: forced load
See http://www.tux.org/lkml/#export-tainted for information about tainted modules
Module xen-platform-pci loaded, with warnings
```

The important part of the message above is the last line which should state the module has been loaded with warnings.

### 37.4. Manually loading the para-virtualized drivers

If for some reason the para-virtualized drivers failed to load automatically during the boot process you can attempt to load them manually.

This will allow you to reconfigure network or storage entities or identify why they failed to load in the first place. The steps below should load the para-virtualized driver modules.

First, locate the para-virtualized driver modules on your system.

```
cd /lib/modules/`uname -r`/
find . -name 'xen_*.ko' -print
```

Take note of the location and load the modules manually. Substitute {LocationofPV-drivers} with the correct location you noted from the output of the commands above.

```
insmod \
/lib/modules/'uname -r'/{LocationofPV-drivers}/xen_platform_pci.ko
insmod /lib/modules/'uname -r'/{LocationofPV-drivers}/xen_balloon.ko
insmod /lib/modules/'uname -r'/{LocationofPV-drivers}/xen_vnif.ko
insmod /lib/modules/'uname -r'/{LocationofPV-drivers}/xen_vbd.ko
```

## 37.5. Verifying the para-virtualized drivers have successfully loaded

One of the first tasks you will want to do is to verify that the drivers have actually been loaded into your system.

After the para-virtualized drivers have been installed and the guest has been rebooted you can verify that the drivers have loaded. First you should confirm the drivers have logged their loading into **/var/log/messages**

```
grep -E "vif|vbd|xen" /var/log/messages
xen_mem: Initialising balloon driver
vif vif-0: 2 parsing device/vif/0/mac
vbd vbd-768: 19 xlvbda at /local/domain/0/backend/vbd/21/76
vbd vbd-768: 19 xlvbda at /local/domain/0/backend/vbd/21/76
xen-vbd: registered block device major 202
```

You can also use the **lsmod** command to list the loaded para-virtualized drivers. It should output a list containing the **xen\_vnif**, **xen\_vbd**, **xen\_platform\_pci** and **xen\_balloon** modules.

```
lsmod | grep xen
xen_vbd 19168 1
xen_vnif 28416 0
xen_balloon 15256 1 xen_vnif
xen_platform_pci 98520 3 xen_vbd,xen_vnif,xen_balloon,[permanent]
```

## 37.6. The system has limited throughput with para-virtualized drivers

If network throughput is still limited even after installing the para-virtualized drivers and you have confirmed they are loaded correctly (see [Section 37.5, “Verifying the para-virtualized drivers have successfully loaded”](#)). To fix this problem, remove the '**type=ioemu**' part of '**vif=**' line in your guest's configuration file.

## Additional resources

To learn more about virtualization and Red Hat Enterprise Linux, see the following resources.

### A.1. Online resources

- » <http://www.cl.cam.ac.uk/research/srg/netos/xen/> The project website of the Xen™ para-virtualization machine manager from which the Red Hat *kernel-xen* package is derived. The site maintains the upstream xen project binaries and source code and also contains information, architecture overviews, documentation, and related links regarding xen and its associated technologies.
- » The Xen Community website  
<http://www.xen.org/>
- » <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>
- » Red Hat Documentation  
<http://www.redhat.com/docs/>
- » Virt Tools  
<http://virt-tools.org/>
- » Virtualization technologies overview  
<http://virt.kernelnewbies.org>
- » Red Hat Emerging Technologies group  
<http://et.redhat.com>

### A.2. Installed documentation

- » **/usr/share/doc/xen-<version-number>/** is the directory which contains information about the Xen para-virtualization hypervisor and associated management tools, including various example configurations, hardware-specific information, and the current Xen upstream user 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.

## Colophon

This manual was written in the DocBook XML v4.3 format.

This book is based on the work of Jan Mark Holzer, Chris Curran and Scott Radvan.

Other writing credits go to:

- » Don Dutille 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.

The Red Hat Localization Team consists of the following people:

- » Simplified Chinese

- Leah Wei Liu

- » Traditional Chinese

- Chester Cheng

- Terry Chuang

- » Japanese

- Kiyoto Hashida

- » Korean

- Eun-ju Kim

- » French

- Sam Friedmann

- » German

- Hedda Peters

- » Italian

- Francesco Valente

- » Brazilian Portuguese

- Glaucia de Freitas

- Leticia de Lima

- » Spanish
  - Angela Garcia
  - Gladys Guerrero
- » Russian
  - Yuliya Poyarkova