



Red Hat Enterprise Linux 6 Virtualization Administration Guide

Managing your virtual environment

Laura Novich

Red Hat Enterprise Linux 6 Virtualization Administration Guide

Managing your virtual environment

Laura Novich
Red Hat Engineering Content Services
Inovich@redhat.com

Legal Notice

Copyright © 2013 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 Virtualization Administration Guide covers administration of host physical machines, networking, storage, device and guest virtual machine management, and troubleshooting.

Table of Contents

Preface	10
1. Document Conventions		10
1.1. Typographic Conventions		10
1.2. Pull-quote Conventions		11
1.3. Notes and Warnings		12
2. Getting Help and Giving Feedback		12
2.1. Do You Need Help?		12
2.2. We Need Feedback!		13
Chapter 1. Introduction	14
1.1. Virtualization Documentation Suite		14
Chapter 2. Server best practices	16
Chapter 3. Security for virtualization	17
3.1. Storage security issues		17
3.2. SELinux and virtualization		17
3.3. SELinux		18
3.4. Virtualization firewall information		19
Chapter 4. sVirt	20
4.1. Security and Virtualization		21
4.2. sVirt labeling		21
Chapter 5. KVM live migration	23
5.1. Live migration requirements		23
5.2. Live migration and Red Hat Enterprise Linux version compatibility		24
5.3. Shared storage example: NFS for a simple migration		25
5.4. Live KVM migration with virsh		26
5.4.1. Additional tips for migration with virsh		28
5.4.2. Additional options for the virsh migrate command		30
5.5. Migrating with virt-manager		30
Chapter 6. Remote management of guests	37
6.1. Remote management with SSH		37
6.2. Remote management over TLS and SSL		39
6.3. Transport modes		41
Chapter 7. Overcommitting with KVM	46
7.1. Introduction		46
7.2. Overcommitting virtualized CPUs		47
Chapter 8. KSM	49
Chapter 9. Advanced guest virtual machine administration	53
9.1. Control Groups (cgroups)		53
9.2. Hugepage support		53
9.3. Running Red Hat Enterprise Linux as a guest virtual machine on a Hyper-V hypervisor		53
Chapter 10. Miscellaneous administration tasks	55
10.1. Automatically starting guest virtual machines		55
10.2. Guest virtual machine memory allocation		55
10.3. Assigning USB devices to guest virtual machines		56
10.3.1. Redirecting USB devices using libvirt		56

10.4. Using qemu-img	56
10.5. Verifying virtualization extensions	60
10.6. Setting KVM processor affinities	61
10.7. Generating a new unique MAC address	65
10.8. Improving guest virtual machine response time	66
10.9. Disable SMART disk monitoring for guest virtual machines	67
10.10. Configuring a VNC Server	67
10.11. Gracefully shutting down guest virtual machines using virsh shutdown	67
10.12. Manipulating the libvirt-guests configuration settings	68
10.13. Virtual machine timer management with libvirt	70
10.14. Using PMU to monitor guest virtual machine performance	73
10.15. Guest virtual machine power management	74
10.16. QEMU guest virtual machine agent protocol	74
10.16.1. guest-sync	74
10.16.2. guest-sync-delimited	75
10.16.3. Creating a guest virtual machine disk backup	75
10.17. Setting a limit on device redirection	77
10.18. Dynamically changing a host physical machine or a network bridge that is attached to a virtual NIC	78
Chapter 11. Storage concepts	79
11.1. Storage pools	79
11.2. Volumes	79
Chapter 12. Storage pools	82
12.1. Creating storage pools	82
12.1.1. Disk-based storage pools	82
12.1.1.1. Creating a disk based storage pool using virsh	82
12.1.1.2. Deleting a storage pool using virsh	85
12.1.2. Partition-based storage pools	85
12.1.2.1. Creating a partition-based storage pool using virt-manager	85
12.1.2.2. Deleting a storage pool using virt-manager	88
12.1.2.3. Creating a partition-based storage pool using virsh	89
12.1.2.4. Deleting a storage pool using virsh	90
12.1.3. Directory-based storage pools	91
12.1.3.1. Creating a directory-based storage pool with virt-manager	91
12.1.3.2. Deleting a storage pool using virt-manager	94
12.1.3.3. Creating a directory-based storage pool with virsh	94
12.1.3.4. Deleting a storage pool using virsh	96
12.1.4. LVM-based storage pools	96
12.1.4.1. Creating an LVM-based storage pool with virt-manager	97
12.1.4.2. Deleting a storage pool using virt-manager	101
12.1.4.3. Creating an LVM-based storage pool with virsh	102
12.1.4.4. Deleting a storage pool using virsh	103
12.1.5. iSCSI-based storage pools	104
12.1.5.1. Configuring a software iSCSI target	104
12.1.5.2. Adding an iSCSI target to virt-manager	107
12.1.5.3. Deleting a storage pool using virt-manager	109
12.1.5.4. Creating an iSCSI-based storage pool with virsh	110
12.1.5.5. Deleting a storage pool using virsh	112
12.1.6. NFS-based storage pools	112
12.1.6.1. Creating a NFS-based storage pool with virt-manager	112
12.1.6.2. Deleting a storage pool using virt-manager	114
Chapter 13. Volumes	116
13.1. Creating volumes	116

13.2. Cloning volumes	116
13.3. Adding storage devices to guests	117
13.3.1. Adding file based storage to a guest	117
13.3.2. Adding hard drives and other block devices to a guest	120
13.3.3. Managing storage controllers in a guest virtual machine	121
13.4. Deleting and removing volumes	122
Chapter 14. Managing guest virtual machines with virsh	123
14.1. virsh command quick reference	123
14.2. Attaching and updating a device with virsh	127
14.3. Connecting to the hypervisor	127
14.4. Creating a virtual machine XML dump (configuration file)	127
14.4.1. Adding multifunction PCI devices to KVM guest virtual machines	129
14.5. Suspending, resuming, saving and restoring a guest virtual machine	130
14.6. Shutting down, rebooting and force-shutdown of a guest virtual machine	130
14.7. Retrieving guest virtual machine information	131
14.8. Retrieving node information	132
14.9. Storage pool information	132
14.10. Displaying per-guest virtual machine information	133
14.11. Managing virtual networks	135
14.12. Migrating guest virtual machines with virsh	136
14.13. Managing snapshots	136
14.13.1. virsh snapshot create	136
14.13.2. snapshot-create-as-domain	137
14.13.3. snapshot-current-domain	138
14.13.4. snapshot-edit-domain	138
14.13.5. snapshot-info-domain	138
14.13.6. snapshot-list-domain	138
14.13.7. snapshot-dumpxml domain snapshot	139
14.13.8. snapshot-parent domain	139
14.13.9. snapshot-revert domain	139
14.13.10. snapshot-delete domain	140
14.14. Guest virtual machine CPU model configuration	140
14.14.1. Introduction	140
14.14.2. Learning about the host physical machine CPU model	140
14.14.3. Determining a compatible CPU model to suit a pool of host physical machines	141
14.15. Configuring the guest virtual machine CPU model	143
14.16. Managing resources for guest virtual machines	144
14.17. Setting schedule parameters	144
14.18. Disk I/O throttling	145
14.19. Display or set block I/O parameters	146
14.20. Setting network interface bandwidth parameters	146
14.21. Configuring memory Tuning	146
14.22. Converting QEMU arguments to domain XML	147
Chapter 15. Managing guests with the Virtual Machine Manager (virt-manager)	149
15.1. Starting virt-manager	149
15.2. The Virtual Machine Manager main window	149
15.3. The virtual hardware details window	150
15.4. Virtual Machine graphical console	152
15.5. Adding a remote connection	154
15.6. Displaying guest details	155
15.7. Performance monitoring	161
15.8. Displaying CPU usage for guests	162
15.9. Displaying CPU usage for hosts	162
15.10. Displaying Disk I/O	163

15.11. Displaying Network I/O	165
Chapter 16. Guest virtual machine disk access with offline tools	168
16.1. Introduction	168
16.2. Terminology	168
16.3. Installation	168
16.4. The guestfish shell	169
16.4.1. Viewing file systems with guestfish	170
16.4.1.1. Manual listing and viewing	170
16.4.1.2. Via guestfish inspection	171
16.4.1.3. Accessing a guest virtual machine by name	171
16.4.2. Modifying files with guestfish	171
16.4.3. Other actions with guestfish	172
16.4.4. Shell scripting with guestfish	172
16.4.5. Augeas and libguestfs scripting	172
16.5. Other commands	173
16.6. virt-rescue: The rescue shell	174
16.6.1. Introduction	174
16.6.2. Running virt-rescue	174
16.7. virt-df: Monitoring disk usage	175
16.7.1. Introduction	175
16.7.2. Running virt-df	175
16.8. virt-resize: resizing guest virtual machines offline	176
16.8.1. Introduction	176
16.8.2. Expanding a disk image	176
16.9. virt-inspector: inspecting guest virtual machines	177
16.9.1. Introduction	178
16.9.2. Installation	178
16.9.3. Running virt-inspector	178
16.10. virt-win-reg: Reading and editing the Windows Registry	180
16.10.1. Introduction	180
16.10.2. Installation	180
16.10.3. Using virt-win-reg	180
16.11. Using the API from Programming Languages	180
16.11.1. Interaction with the API via a C program	181
16.12. Troubleshooting	185
16.13. Where to find further documentation	185
Chapter 17. Using simple tools for guest virtual machine management	186
17.1. Using virt-viewer	186
17.2. remote-viewer	187
Chapter 18. Virtual Networking	189
18.1. Virtual network switches	189
18.2. Network Address Translation	190
18.3. Networking protocols	191
18.3.1. DNS and DHCP	191
18.3.2. Routed mode	191
18.3.3. Isolated mode	192
18.4. The default configuration	193
18.5. Examples of common scenarios	193
18.5.1. Routed mode	193
18.5.2. NAT mode	195
18.5.3. Isolated mode	195
18.6. Managing a virtual network	196
18.7. Creating a virtual network	196

18.8. Attaching a virtual network to a guest	200
18.9. Directly attaching to physical interface	203
18.10. Applying network filtering	204
18.10.1. Introduction	205
18.10.2. Filtering chains	206
18.10.3. Filtering chain priorities	207
18.10.4. Usage of variables in filters	208
18.10.5. Automatic IP address detection and DHCP snooping	210
18.10.5.1. Introduction	210
18.10.5.2. DHCP snooping	211
18.10.6. Reserved Variables	211
18.10.7. Element and attribute overview	212
18.10.8. References to other filters	212
18.10.9. Filter rules	212
18.10.10. Supported protocols	213
18.10.10.1. MAC (Ethernet)	214
18.10.10.2. VLAN (802.1Q)	214
18.10.10.3. STP (Spanning Tree Protocol)	215
18.10.10.4. ARP/RARP	216
18.10.10.5. IPv4	216
18.10.10.6. IPv6	217
18.10.10.7. TCP/UDP/SCTP	218
18.10.10.8. ICMP	219
18.10.10.9. IGMP, ESP, AH, UDPLITE, 'ALL'	220
18.10.10.10. TCP/UDP/SCTP over IPV6	221
18.10.10.11. ICMPv6	222
18.10.10.12. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv6	223
18.10.11. Advanced Filter Configuration Topics	224
18.10.11.1. Connection tracking	224
18.10.11.2. Limiting Number of Connections	225
18.10.11.3. Command line tools	226
18.10.11.4. Pre-existing network filters	226
18.10.11.5. Writing your own filters	227
18.10.11.6. Sample custom filter	229
18.10.12. Limitations	233
Chapter 19. qemu-kvm Whitelist	235
19.1. Introduction	235
Whitelist format	235
19.2. Basic options	235
Emulated machine	235
Processor type	235
Processor Topology	235
NUMA system	236
Memory size	236
Keyboard layout	236
Guest name	236
Guest UUID	236
19.3. Disk options	236
Generic drive	236
Boot option	237
Snapshot mode	237
19.4. Display options	237
Disable graphics	237
VGA card emulation	237
VNC display	237

Spice desktop	237
19.5. Network options	238
TAP network	238
19.6. Device options	238
General device	238
Global device setting	243
Character device	243
Enable USB	243
19.7. Linux/Multiboot boot	244
Kernel file	244
Ram disk	244
Command line parameter	244
19.8. Expert options	244
KVM virtualization	244
Disable kernel mode PIT reinjection	244
No shutdown	244
No reboot	244
Serial port, monitor, QMP	244
Monitor redirect	244
Manual CPU start	245
RTC	245
Watchdog	245
Watchdog reaction	245
Guest memory backing	245
SMBIOS entry	245
19.9. Help and information options	245
Help	245
Version	245
Audio help	245
19.10. Miscellaneous options	245
Migration	245
No default configuration	245
Device configuration file	245
Loaded saved state	245
Chapter 20. Manipulating the domain xml	247
20.1. General information and metadata	247
20.2. Operating system booting	248
20.2.1. BIOS bootloader	248
20.2.2. Host physical machine bootloader	251
20.2.3. Direct kernel boot	251
20.2.4. Container boot	252
20.3. SMBIOS system information	252
20.4. CPU allocation	253
20.5. CPU tuning	254
20.6. Memory backing	256
20.7. Memory tuning	256
20.8. NUMA node tuning	257
20.9. Block I/O tuning	258
20.10. Resource partitioning	259
20.11. CPU model and topology	259
20.11.1. Guest virtual machine NUMA topology	263
20.12. Events configuration	263
20.13. Power Management	266
20.14. Hypervisor features	266
20.15. Time keeping	267

20.16. Devices	270
20.16.1. Hard drives, floppy disks, CDROMs	270
20.16.1.1. Disk element	272
20.16.1.2. Source element	272
20.16.1.3. Mirror element	272
20.16.1.4. Target element	272
20.16.1.5. iotune	273
20.16.1.6. driver	273
20.16.1.7. Additional Device Elements	274
20.16.2. Filesystems	275
20.16.3. Device addresses	276
20.16.4. Controllers	277
20.16.5. Device leases	278
20.16.6. Host physical machine device assignment	279
20.16.6.1. USB / PCI devices	279
20.16.6.2. Block / character devices	281
20.16.7. Redirected devices	282
20.16.8. Smartcard devices	283
20.16.9. Network interfaces	285
20.16.9.1. Virtual networks	286
20.16.9.2. Bridge to LAN	287
20.16.9.3. Setting a port masquerading range	288
20.16.9.4. Userspace SLIRP stack	288
20.16.9.5. Generic Ethernet connection	289
20.16.9.6. Direct attachment to physical interfaces	289
20.16.9.7. PCI passthrough	292
20.16.9.8. Multicast tunnel	293
20.16.9.9. TCP tunnel	293
20.16.9.10. Setting NIC driver-specific options	294
20.16.9.11. Overriding the target element	295
20.16.9.12. Specifying boot order	296
20.16.9.13. Interface ROM BIOS configuration	296
20.16.9.14. Quality of service	297
20.16.9.15. Setting VLAN tag (on supported network types only)	297
20.16.9.16. Modifying virtual link state	298
20.16.10. Input devices	298
20.16.11. Hub devices	299
20.16.12. Graphical framebuffers	299
20.16.13. Video devices	304
20.16.14. Consoles, serial, parallel, and channel devices	304
20.16.15. Guest virtual machine interfaces	305
20.16.16. Channel	307
20.16.17. Host physical machine interface	308
20.17. Sound devices	313
20.18. Watchdog device	313
20.19. Memory balloon device	314
20.20. Random number generator device	315
20.21. TPM devices	316
20.22. Security label	316
20.23. Example domain XML configuration	317
Chapter 21. Troubleshooting	320
21.1. Debugging and troubleshooting tools	320
21.2. Creating virsh dump files	321
21.3. kvm_stat	322
21.4. Troubleshooting with serial consoles	325

21.5. Virtualization log files	326
21.6. Loop device errors	326
21.7. Live Migration Errors	326
21.8. Enabling Intel VT-x and AMD-V virtualization hardware extensions in BIOS	327
21.9. KVM networking performance	327
21.10. Workaround for creating external snapshots with libvirt	329
21.11. Missing characters on guest console with Japanese keyboard	329
21.12. Known Windows XP guest issues	329
The Virtual Host Metrics Daemon (vhostmd)	331
Additional resources	332
B.1. Online resources	332
B.2. Installed documentation	332
Revision History	333
Index	346
F	346
H	346
V	347

Preface

1. Document Conventions

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

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

1.1. Typographic Conventions

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

Mono-spaced Bold

Used to highlight system input, including shell commands, file names and paths. Also used to highlight 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.

1.2. Pull-quote Conventions

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

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

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

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

```

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;
}

```

1.3. Notes and Warnings

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



Note

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



Important

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



Warning

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

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

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

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

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

2.2. We Need Feedback!

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

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

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

Chapter 1. Introduction

1.1. Virtualization Documentation Suite

Red Hat offers a wealth of documentation solutions across its various virtualization products. Coverage of Red Hat Enterprise Linux and its inbuilt virtualization products includes:

- ▶ *Red Hat Enterprise Linux — Virtualization Getting Started Guide*: This guide provides an introduction to virtualization concepts, advantages, and tools, and an overview of Red Hat virtualization documentation and products.
- ▶ *Red Hat Enterprise Linux — Virtualization Host Configuration and Guest Installation Guide*: This guide covers the installation of virtualization software and configuration of guest virtual machines on a host physical machine.
- ▶ *Red Hat Enterprise Linux — Virtualization Administration Guide*: This guide covers administration of host physical machines, networking, storage, and device and guest virtual machine management using either virt-manager or virsh as primary configuration tools. This guide also includes a libvirt and QEMU reference, as well as troubleshooting information.
- ▶ *Red Hat Enterprise Linux — Virtualization Security Guide*: This guide provides an overview of virtualization security technologies provided by Red Hat. Also included are recommendations for securing host physical machines, guest virtual machines, and shared infrastructure and resources in virtualized environments.
- ▶ *Red Hat Enterprise Linux — Virtualization Tuning and Optimization Guide*: This guide provides tips, tricks and suggestions for making full use of virtualization performance features and options for your systems and guest virtual machines.
- ▶ *Red Hat Enterprise Linux — Hypervisor Deployment Guide* describes how to deploy and install the Red Hat Enterprise Virtualization Hypervisor. Read this guide if you need advanced information about installing and deploying Hypervisors. The basic installation of Hypervisor host physical machines is also described in the *Red Hat Enterprise Virtualization Installation Guide*.
- ▶ *Red Hat Enterprise Linux — V2V Guide* describes importing virtual machines from KVM, Xen and VMware ESX/ESX(i) hypervisors to Red Hat Enterprise Virtualization and KVM managed by libvirt.

The Red Hat Enterprise Virtualization documentation suite provides information on installation, development of applications, configuration and usage of the Red Hat Enterprise Virtualization platform and its related products.

- ▶ *Red Hat Enterprise Virtualization — Administration Guide* describes how to set up, configure and manage Red Hat Enterprise Virtualization. It assumes that you have successfully installed the Red Hat Enterprise Virtualization Manager and host physical machines.
- ▶ *Red Hat Enterprise Virtualization — Command Line Shell Guide* contains information for installing and using the Red Hat Enterprise Virtualization Manager command line shell.
- ▶ *Red Hat Enterprise Virtualization — Developer Guide* explains how to use the REST API. It covers the fundamentals of the REST architectural concepts in the context of a virtualization environment and provides examples of the API in operation. It also documents the installation and use of the Python Software Development Kit.
- ▶ *Red Hat Enterprise Virtualization — Evaluation Guide* enables prospective customers to evaluate the features of Red Hat Enterprise Virtualization. Use this guide if you have an evaluation license.
- ▶ *Red Hat Enterprise Virtualization — Installation Guide* describes the installation prerequisites and procedures. Read this if you need to install Red Hat Enterprise Virtualization. The installation of host physical machines, Manager and storage are covered in this guide. You will need to refer to the *Red Hat Enterprise Virtualization Administration Guide* to configure the system before you can start using the platform.
- ▶ *Red Hat Enterprise Virtualization — Manager Release Notes* contain release specific information for Red Hat Enterprise Virtualization Managers.
- ▶ *Red Hat Enterprise Virtualization — Power User Portal Guide* describes how power users can create and manage virtual machines from the Red Hat Enterprise Virtualization User Portal.

- ▶ *Red Hat Enterprise Virtualization — Quick Start Guide* provides quick and simple instructions for first time users to set up a basic Red Hat Enterprise Virtualization environment.
- ▶ *Red Hat Enterprise Virtualization — Technical Notes* describe the changes made between the current release and the previous one.
- ▶ *Red Hat Enterprise Virtualization — Technical Reference Guide* describes the technical architecture of Red Hat Enterprise Virtualization and its interactions with existing infrastructure.
- ▶ *Red Hat Enterprise Virtualization — User Portal Guide* describes how users of the Red Hat Enterprise Virtualization system can access and use virtual desktops from the User Portal.



Note

All of the guides for these products are available at the Red Hat Customer Portal:

<https://access.redhat.com/site/documentation/>.

Chapter 2. Server best practices

The following tasks and tips can assist you with increasing the performance of your Red Hat Enterprise Linux host. Additional tips can be found in the *Red Hat Enterprise Linux Virtualization Tuning and Optimization Guide*

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

```
# setenforce 1
```

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

Chapter 3. Security for virtualization

When deploying virtualization technologies, you must ensure that the host physical machine and its operating system cannot be compromised. In this case *the host* is a Red Hat Enterprise Linux system that manages the system, devices, memory and networks as well as all guest virtual machines. If the host physical machine is insecure, all guest virtual machines 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*. This plan needs to contain the following:

- ▶ Operating specifications
- ▶ Specifies which services are needed on your guest virtual machines
- ▶ Specifies the host physical 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 host physical machines. The fewer processes and services running on the host physical machine, the higher the level of security and performance.
- ▶ Enable SELinux on the hypervisor. Read [Section 3.2, “SELinux and virtualization”](#) for more information on using SELinux and virtualization. Additional security tips are located in the *Red Hat Enterprise Linux Virtualization Security Guide*
- ▶ Use a firewall to restrict traffic to the host physical machine. You can setup a firewall with default-reject rules that will help secure the host physical machine from attacks. It is also important to limit network-facing services.
- ▶ Do not allow normal users to access the host operating system. If the host operating system is privileged, granting access to unprivileged accounts may compromise the level of security.

3.1. Storage security issues

Keeping in mind that any user with administrative security privileges for guest virtual machines can potentially change the partitions in the host physical machine, it is imperative that only actual system administrators are granted this level of security. In addition, the following should be considered:

- ▶ The host physical machine should not use disk labels to identify file systems in the **`fstab`** file, the **`initrd`** file which are accessed by the command line. If less privileged users, especially users of guest virtual machines have write access to whole partitions or LVM volumes, then they can be accidentally deleted and this mistake will impact all other guest virtual machines that are using the same storage.
- ▶ Users of guest virtual machines should not be given write access to entire disks or block devices (for example, **`/dev/sdb`**). To avoid this, use partitions such as **`/dev/sdb1`** or LVM volumes.

3.2. SELinux and virtualization

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

Procedure 3.1. Creating and mounting a logical volume on a guest virtual machine with SELinux enabled

1. Create a logical volume. This example creates a 5 gigabyte logical volume named **`NewVolumeName`** on the volume group named **`volumeGroup`**. This example also assumes that there is enough disk space. You may have to create additional storage on a network device and give the guest access to it. Refer to [Chapter 13, Volumes](#) for more information.

```
# 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 SELinux type for the folder you just created.

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

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

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

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

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

Note

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

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

Verify the file has been relabeled using the following command:

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

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

3.3. SELinux

This section contains topics to consider when using SELinux with your virtualization deployment. When you deploy system changes or add devices, you must update your SELinux policy accordingly. To configure an LVM volume for a guest virtual machine, you must modify the SELinux context for the respective underlying block device and volume group. Make sure that you have installed the **policycoreutils-python** package (**yum install policycoreutils-python**) before running the command.

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

KVM and SELinux

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

KVM SELinux Booleans

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

3.4. Virtualization firewall information

Various ports are used for communication between guest virtual machines and cooresponding management utilities.

Note

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

- ▶ ICMP requests must be accepted. ICMP packets are used for network testing. You cannot ping guest virtual machines if the 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 physical machine.
- ▶ Ports 5634 to 6166 are used for guest virtual machine console access with the SPICE protocol.
- ▶ Ports 49152 to 49216 are used for migrations with KVM. Migration may use any port in this range depending on the number of concurrent migrations occurring.
- ▶ Enabling IP forwarding (**net.ipv4.ip_forward = 1**) is also required for shared bridges and the default bridge. Note that installing libvirt enables this variable so it will be enabled when the virtualization packages are installed unless it was manually disabled.

Note

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

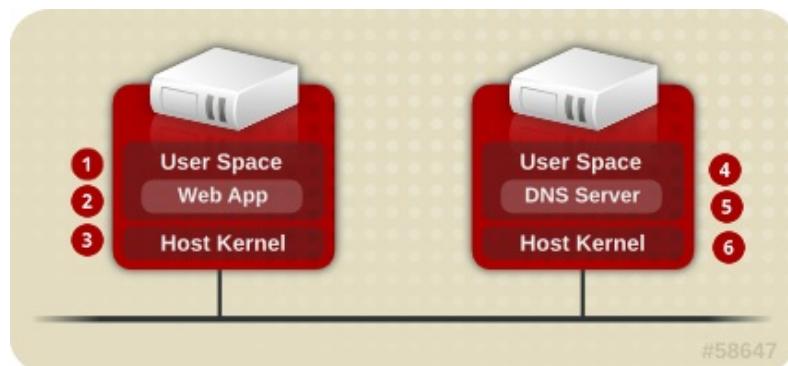
Chapter 4. sVirt

sVirt is a technology included in Red Hat Enterprise Linux 6 that integrates SELinux and virtualization. sVirt applies Mandatory Access Control (MAC) to improve security when using guest virtual machines. This integrated technology improves security and hardens the system against bugs in the hypervisor. It is particularly helpful in preventing attacks on the host physical machine or on another guest virtual machine.

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

Non-virtualized environments

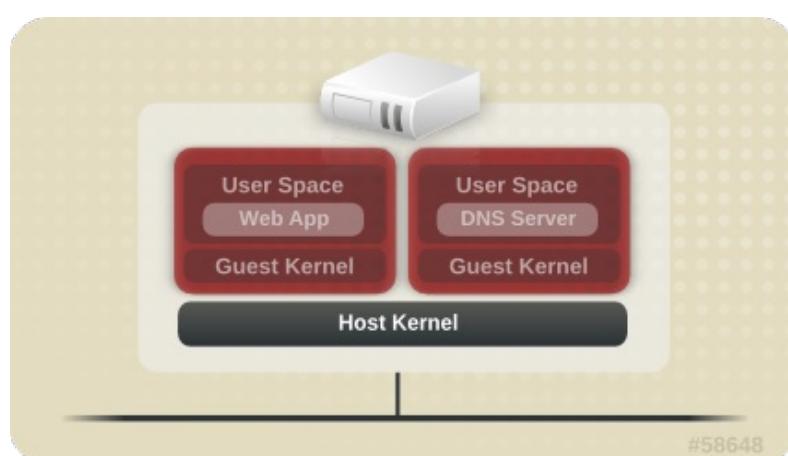
In a non-virtualized environment, host physical machines are separated from each other physically and each host physical machine has a self-contained environment, consisting of services such as a web server, or a DNS server. These services communicate directly to their own user space, host physical machine's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



- ① User Space - memory area where all user mode applications and some drivers execute.
- ④
- ② Web App (web application server) - delivers web content that can be accessed through a browser.
- ③ Host Kernel - is strictly reserved for running the host physical machine's privileged kernel, kernel extensions, and most device drivers.
- ⑥
- ⑤ DNS Server - stores DNS records allowing users to access web pages using logical names instead of IP addresses.

Virtualized environments

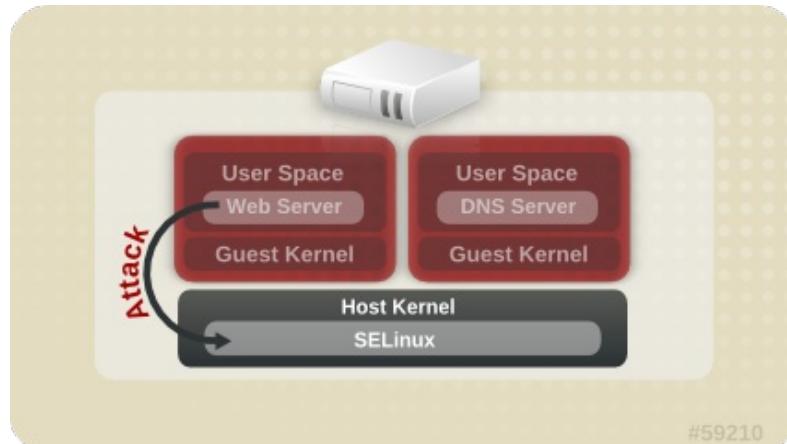
In a virtualized environment, several virtual operating systems can run on a single kernel residing on a host physical machine. The following image represents a virtualized environment:



4.1. Security and Virtualization

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

sVirt is an effort to isolate guest virtual machines and limit their ability to launch further attacks if exploited. This is demonstrated in the following image, where an attack can not break out of the guest virtual machine and invade other guest virtual machines:



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

4.2. sVirt labeling

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

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

```
# ps -eZ | grep qemu
system_u:system_r:svirt_t:s0:c87,c520 27950 ? 00:00:17 qemu-kvm
```

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

```
# ls -lZ /var/lib/libvirt/images/*
system_u:object_r:svirt_image_t:s0:c87,c520 image1
```

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

Table 4.1. sVirt context labels

SELinux Context	Type / Description
system_u:system_r:svirt_t:MCS1	Guest virtual machine processes. MCS1 is a random MCS field. Approximately 500,000 labels are supported.
system_u:object_r:svirt_image_t:MCS1	Guest virtual machine images. Only <i>svirt_t</i> processes with the same MCS fields can read/write these images.
system_u:object_r:svirt_image_t:s0	Guest virtual machine shared read/write content. All <i>svirt_t</i> processes can write to the <i>svirt_image_t:s0</i> files.

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

Chapter 5. KVM live migration

This chapter covers migrating guest virtual machines running on one host physical machine to another. In both instances, the host physical machines are running the KVM hypervisor.

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

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

Migration works by sending the state of the guest virtual machine's memory and any virtualized devices to a destination host physical machine. It is recommended to use shared, networked storage to store the guest virtual machine's images to be migrated. It is also recommended to use libvirt-managed storage pools for shared storage when migrating virtual machines.

Migrations can be performed live or not.

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

A migration that is not performed live, suspends the guest virtual machine, then moves an image of the guest virtual machine's memory to the destination host physical machine. The guest virtual machine is then resumed on the destination host physical machine and the memory the guest virtual machine used on the source host physical machine is freed. The time it takes to complete such a migration depends on network bandwidth and latency. If the network is experiencing heavy use or low bandwidth, the migration will take much longer.

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

5.1. Live migration requirements

Migrating guest virtual machines requires the following:

Migration requirements

- ▶ A guest virtual machine installed on shared storage using one of the following protocols:
 - Fibre Channel-based LUNs
 - iSCSI
 - FCoE
 - NFS
 - GFS2
 - SCSI RDMA protocols (SCSI RCP): the block export protocol used in Infiniband and 10GbE iWARP

adapters

- ▶ The migration platforms and versions should be checked against table [Table 5.1, “Live Migration Compatibility”](#).
- ▶ Both systems must have the appropriate TCP/IP ports open. In cases where a firewall is used refer to [Section 3.4, “Virtualization firewall information”](#) for detailed port information.
- ▶ A separate system exporting the shared storage medium. Storage should not reside on either of the two host physical machines being used for migration.
- ▶ Shared storage must mount at the same location on source and destination systems. The mounted directory names must be identical. Although it is possible to keep the images using different paths, it is not recommended. Note that, if you are intending to use virt-manager to perform the migration, the path names must be identical. If however you intend to use virsh to perform the migration, different network configurations and mount directories can be used with the help of --xml option or pre-hooks when doing migrations. Even with out shared storage, migration can still succeed with the command **-copy-storage-all** (deprecated). For more information on **prehooks**, refer to [libvirt.org](#), and for more information on the XML option, refer to [Chapter 20, Manipulating the domain xml](#).
- ▶ When migration is attempted on an existing guest virtual machine in a public bridge+tap network, the source and destination host physical machines must be located in the same network. Otherwise, the guest virtual machine network will not operate after migration.

Make sure that the **libvирtd** service is enabled (# **chkconfig libvирtd on**) and running (# **service libvирtd start**). It is also important to note that the ability to migrate effectively is dependent on the parameter settings in the **/etc/libvirt/libvирtd.conf** configuration file.

Procedure 5.1. Configuring libvирtd.conf

1. Opening the **libvирtd.conf** requires running the command as root:

```
# vim /etc/libvirt/libvирtd.conf
```

2. Change the parameters as needed and save the file.
3. Restart the **libvирtd** service:

```
# service libvирtd restart
```

5.2. Live migration and Red Hat Enterprise Linux version compatibility

Live Migration is supported as shown in table [Table 5.1, “Live Migration Compatibility”](#):

Table 5.1. Live Migration Compatibility

Migration Method	Release Type	Example	Live Migration Support	Notes
Forward	Major release	5.x → 6.y	Not supported	
Forward	Minor release	5.x → 5.y ($y > x$, $x \geq 4$)	Fully supported	Any issues should be reported
Forward	Minor release	6.x → 6.y ($y > x$, $x \geq 0$)	Fully supported	Any issues should be reported
Backward	Major release	6.x → 5.y	Not supported	
Backward	Minor release	5.x → 5.y ($x > y, y \geq 4$)	Supported	Refer to Troubleshooting problems with migration for known issues
Backward	Minor release	6.x → 6.y ($x > y$, $y \geq 0$)	Supported	Refer to Troubleshooting problems with migration for known issues

Troubleshooting problems with migration

- ▶ **Issues with SPICE** — It has been found that SPICE has an incompatible change when migrating from 6.0 → 6.1. In such cases, the client may disconnect and then reconnect, causing a temporary loss of audio and video. This is only temporary and all services will resume.
- ▶ **Issues with USB** — Red Hat Enterprise Linux 6.2 added USB functionality which included migration support, but not without certain caveats which reset USB devices and caused any application running over the device to abort. This problem was fixed in Red Hat Enterprise Linux 6.4, and should not occur in future versions. To prevent this from happening in a version prior to 6.4, abstain from migrating while USB devices are in use.
- ▶ **Issues with the migration protocol** — If backward migration ends with "unknown section error", repeating the migration process can repair the issue as it may be a transient error. If not, please report the problem.

Configuring network storage

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

Alternatively, use the NFS example in [Section 5.3, “Shared storage example: NFS for a simple migration”](#).

5.3. Shared storage example: NFS for a simple migration



Important

This example uses NFS to share guest virtual machine images with other KVM host physical machines. Although not practical for large installations, it is presented to demonstrate migration techniques only. Do not use this example for migrating or running more than a few guest virtual machines. In addition, it is required that the ***synch*** parameter is enabled. This is required for proper export of the NFS storage.

iSCSI storage is a better choice for large deployments. Refer to [Section 12.1.5, “iSCSI-based storage pools”](#) for configuration details.

Also note, that the instructions provided herein are not meant to replace the detailed instructions found in *Red Hat Linux Storage Administration Guide*. Refer to this guide for information on configuring NFS,

opening IP tables, and configuring the firewall.

Make sure that NFS filelocking is not used as it is not supported in KVM.

1. Export your libvirt image directory

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

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

Change the hostname parameter as required for your environment.

2. Start NFS

- Install the NFS packages if they are not yet installed:

```
# yum install nfs
```

- Make sure that the ports for NFS in `iptables` (2049, for example) are opened and add NFS to the `/etc/hosts.allow` file.
- Start the NFS service:

```
# service nfs start
```

3. Mount the shared storage on the destination

On the migration destination system, mount the `/var/lib/libvirt/images` directory:

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



Warning

Whichever directory is chosen for the guest virtual machine must be exactly the same as that on the host physical machine. This applies to all types of shared storage. The directory must be the same or the migration with virt-manager will fail.

5.4. Live KVM migration with virsh

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

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

Note that the `--live` option may be eliminated when live migration is not desired. Additional options are listed in [Section 5.4.2, “Additional options for the virsh migrate command”](#).

The `GuestName` parameter represents the name of the guest virtual machine which you want to migrate.

The `DestinationURL` parameter is the connection URL of the destination host physical machine. The destination system must run the same version of Red Hat Enterprise Linux, be using the same hypervisor and have `libvirt` running.



Note

The ***DestinationURL*** parameter for normal migration and peer2peer migration has different semantics:

- ▶ normal migration: the ***DestinationURL*** is the URL of the target host physical machine as seen from the source guest virtual machine.
- ▶ peer2peer migration: ***DestinationURL*** is the URL of the target host physical machine as seen from the source host physical machine.

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



Important

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

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

Example: live migration with virsh

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

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

1. Verify the guest virtual machine is running

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

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

2. Migrate the guest virtual machine

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

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

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

3. Wait

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

4. Verify the guest virtual machine has arrived at the destination host

From the destination system, **host2.example.com**, verify **guest1-rhel6-64** is running:

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

The live migration is now complete.

Note

libvirt supports a variety of networking methods including TLS/SSL, UNIX sockets, SSH, and unencrypted TCP. Refer to [Chapter 6, Remote management of guests](#) for more information on using other methods.

Note

Non-running guest virtual machines cannot be migrated with the **virsh migrate** command. To migrate a non-running guest virtual machine, the following script should be used:

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

5.4.1. Additional tips for migration with virsh

It is possible to perform multiple, concurrent live migrations where each migration runs in a separate command shell. However, this should be done with caution and should involve careful calculations as each migration instance uses one MAX_CLIENT from each side (source and target). As the default setting is 20, there is enough to run 10 instances without changing the settings. Should you need to change the settings, refer to the procedure [Procedure 5.1, “Configuring libvirtd.conf”](#).

1. Open the libvirtd.conf file as described in [Procedure 5.1, “Configuring libvirtd.conf”](#).
2. Look for the Processing controls section.

```
#####
#
# Processing controls
#
# The maximum number of concurrent client connections to allow
# over all sockets combined.
#max_clients = 20

# The minimum limit sets the number of workers to start up
# initially. If the number of active clients exceeds this,
# then more threads are spawned, upto max_workers limit.
# Typically you'd want max_workers to equal maximum number
# of clients allowed
#min_workers = 5
#max_workers = 20

# The number of priority workers. If all workers from above
# pool will stuck, some calls marked as high priority
# (notably domainDestroy) can be executed in this pool.
#prio_workers = 5

# Total global limit on concurrent RPC calls. Should be
# at least as large as max_workers. Beyond this, RPC requests
# will be read into memory and queued. This directly impact
# memory usage, currently each request requires 256 KB of
# memory. So by default upto 5 MB of memory is used
#
# XXX this isn't actually enforced yet, only the per-client
# limit is used so far
#max_requests = 20

# Limit on concurrent requests from a single client
# connection. To avoid one client monopolizing the server
# this should be a small fraction of the global max_requests
# and max_workers parameter
#max_client_requests = 5

#####
```

3. Change the ***max_clients*** and ***max_workers*** parameters settings. It is recommended that the number be the same in both parameters. The ***max_clients*** will use 2 clients per migration (one per side) and ***max_workers*** will use 1 worker on the source and 0 workers on the destination during the perform phase and 1 worker on the destination during the finish phase.



Important

The ***max_clients*** and ***max_workers*** parameters settings are effected by all guest virtual machine connections to the libvirtd service. This means that any user that is using the same guest virtual machine and is performing a migration at the same time will also be beholden to the limits set in the the ***max_clients*** and ***max_workers*** parameters settings. This is why the maximum value needs to be considered carefully before performing a concurrent live migration.

4. Save the file and restart the service.

**Note**

There may be cases where a migration connection drops because there are too many ssh sessions that have been started, but not yet authenticated. By default, *sshd* allows only 10 sessions to be in a "pre-authenticated state" at any time. This setting is controlled by the **MaxStartups** parameter in the *sshd* configuration file (located here:

`/etc/ssh/sshd_config`), which may require some adjustment. Adjusting this parameter should be done with caution as the limitation is put in place to prevent DoS attacks (and over-use of resources in general). Setting this value too high will negate its purpose. To change this parameter, edit the file `/etc/ssh/sshd_config`, remove the # from the beginning of the **MaxStartups** line, and change the **10** (default value) to a higher number. Remember to save the file and restart the *sshd* service. For more information, refer to the `sshd_config` MAN page.

5.4.2. Additional options for the `virsh migrate` command

In addition to `--live`, `virsh migrate` accepts the following options:

- ▶ `--direct` - used for direct migration
- ▶ `--p2p` - used for peer-2-peer migration
- ▶ `--tunneled` - used for tunneled migration
- ▶ `--persistent` - leaves the domain persistent on destination host physical machine
- ▶ `--undefinesource` - undefines the domain on the source host physical machine
- ▶ `--suspend` - leaves the domain paused on the destination host physical machine
- ▶ `--copy-storage-all` (deprecated) - indicates migration with non-shared storage with full disk copy.
- ▶ `--copy-storage-inc` (deprecated) - indicates migration with non-shared storage with incremental copy (same base image shared between source and destination). In both cases the disk images have to exist on the destination host physical machine, the `--copy-storage-*`. (deprecated) options only tell libvirt to transfer data from the images on source host physical machine to the images found at the same place on the destination host physical machine
- ▶ `--change-protection` - enforces that no incompatible configuration changes will be made to the domain while the migration is underway; this flag is implicitly enabled when supported by the hypervisor, but can be explicitly used to reject the migration if the hypervisor lacks change protection support.
- ▶ `--unsafe` - forces the migration to occur, ignoring all safety procedures.
- ▶ `--verbose` - displays the progress of migration as it is occurring
- ▶ `--migrateuri` - the migration URI which is usually omitted.
- ▶ `--timeout seconds` - forces a guest virtual machine to suspend when the live migration counter exceeds N seconds. It can only be used with a live migration. Once the timeout is initiated, the migration continues on the suspended guest virtual machine.
- ▶ `dname` - is used for renaming the domain to new name during migration, which also usually can be omitted
- ▶ `--xml` - the filename indicated can be used to supply an alternative XML file for use on the destination to supply a larger set of changes to any host-specific portions of the domain XML, such as accounting for naming differences between source and destination in accessing underlying storage. This option is usually omitted.

Refer to the `virsh` man page for more information.

5.5. Migrating with `virt-manager`

This section covers migrating a KVM guest virtual machine with `virt-manager` from one host physical

machine to another.

1. Open virt-manager

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

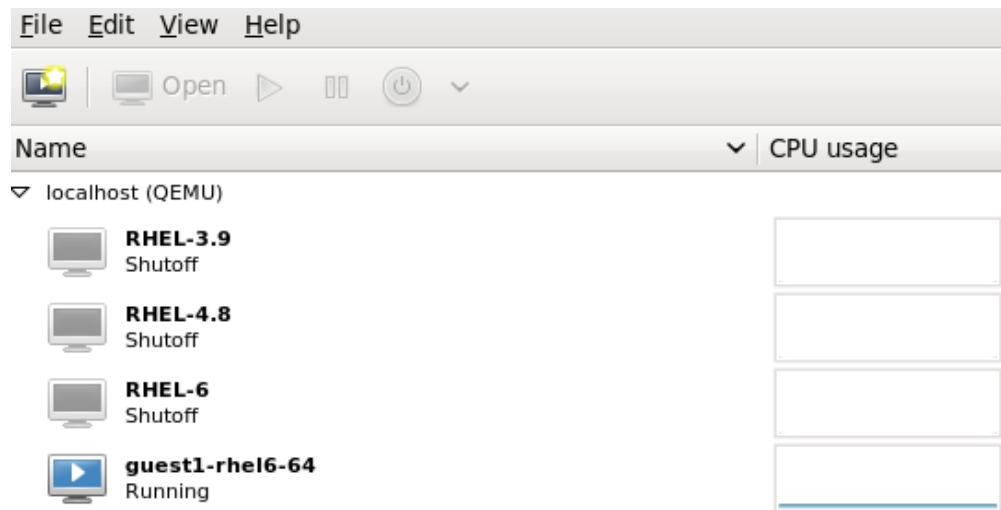


Figure 5.1. Virt-Manager main menu

2. Connect to the target host physical machine

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

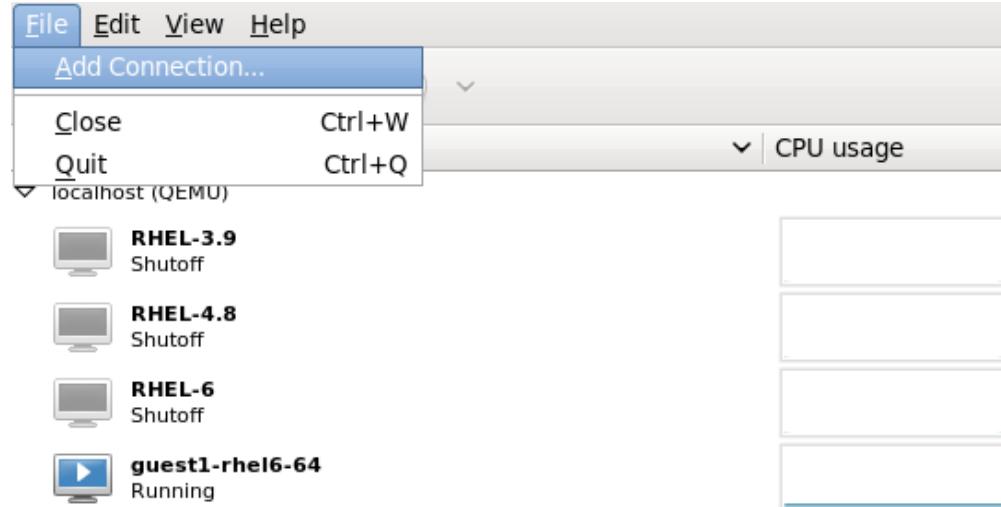
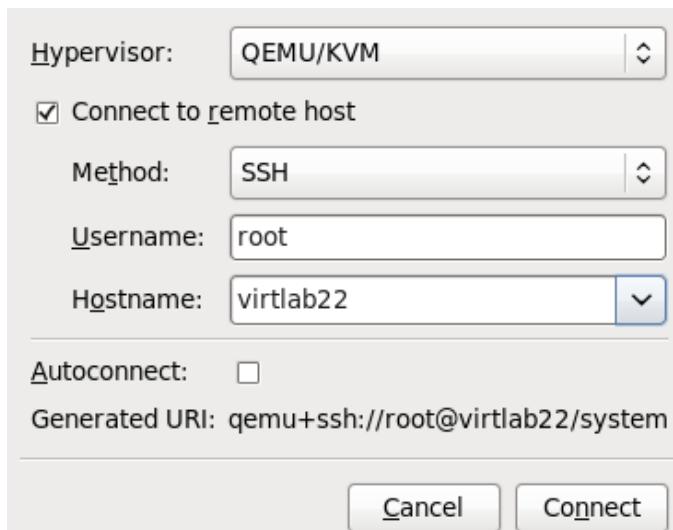


Figure 5.2. Open Add Connection window

3. Add connection

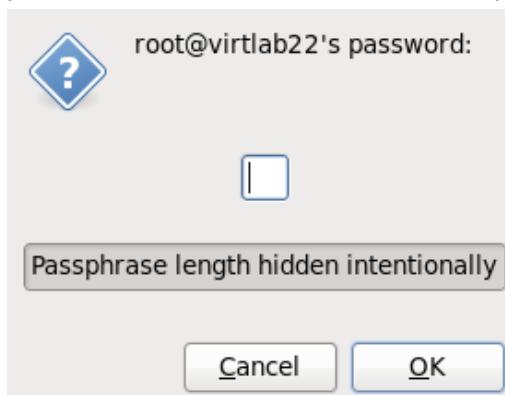
The **Add Connection** window appears.

**Figure 5.3. Adding a connection to the target host physical machine**

Enter the following details:

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

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

**Figure 5.4. Enter password**

4. Migrate guest virtual machines

Open the list of guests inside the source host physical machine (click the small triangle on the left of the host name) and right click on the guest that is to be migrated (**guest1-rhel6-64** in this example) and click **Migrate**.

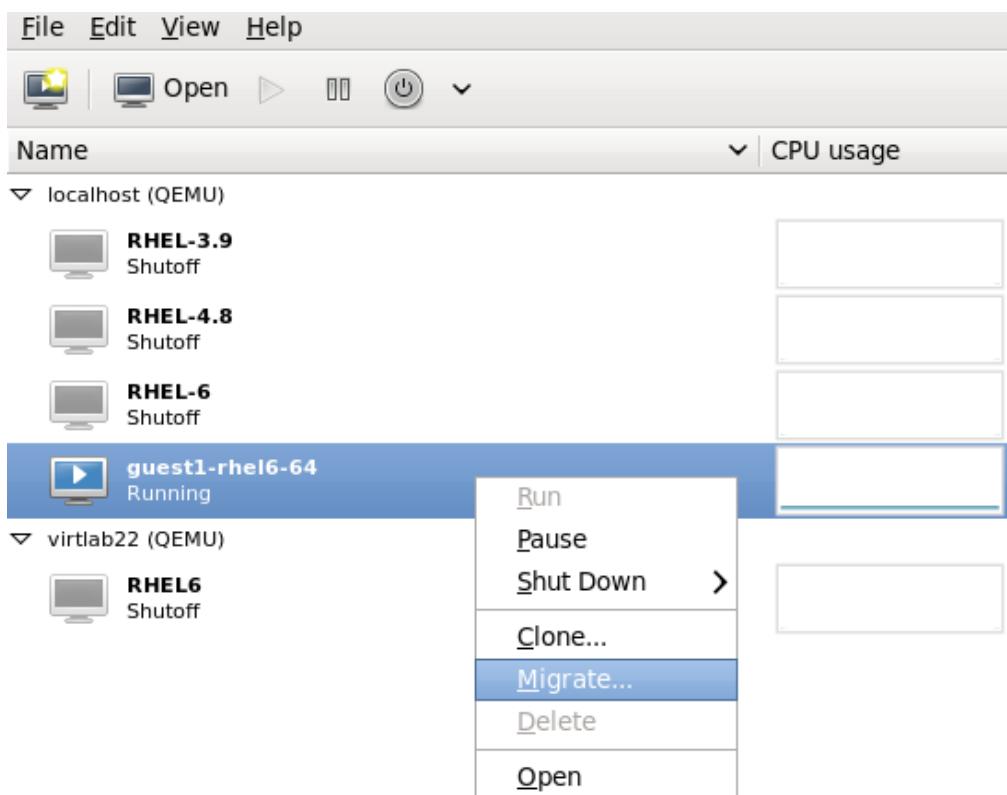


Figure 5.5. Choosing the guest to be migrated

In the **New Host** field, use the drop-down list to select the host physical machine you wish to migrate the guest virtual machine to and click **Migrate**.

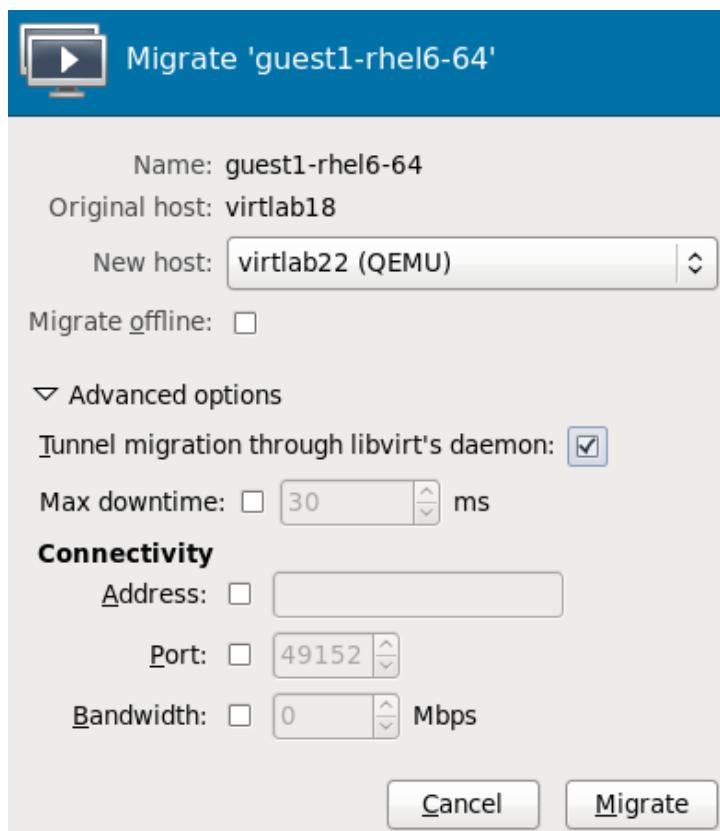


Figure 5.6. Choosing the destination host physical machine and starting the migration process

A progress window will appear.

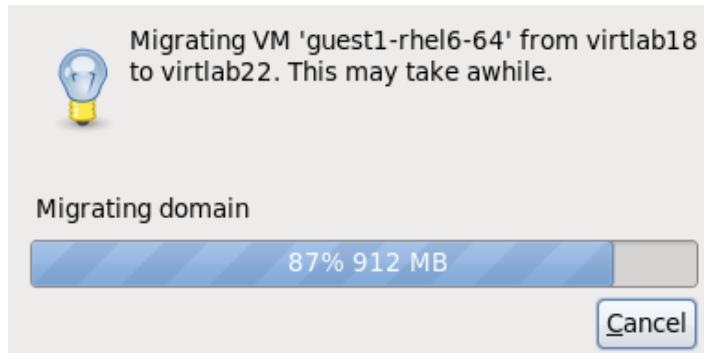


Figure 5.7. Progress window

virt-manager now displays the newly migrated guest virtual machine running in the destination host. The guest virtual machine that was running in the source host physical machine is now listed in the Shutoff state.

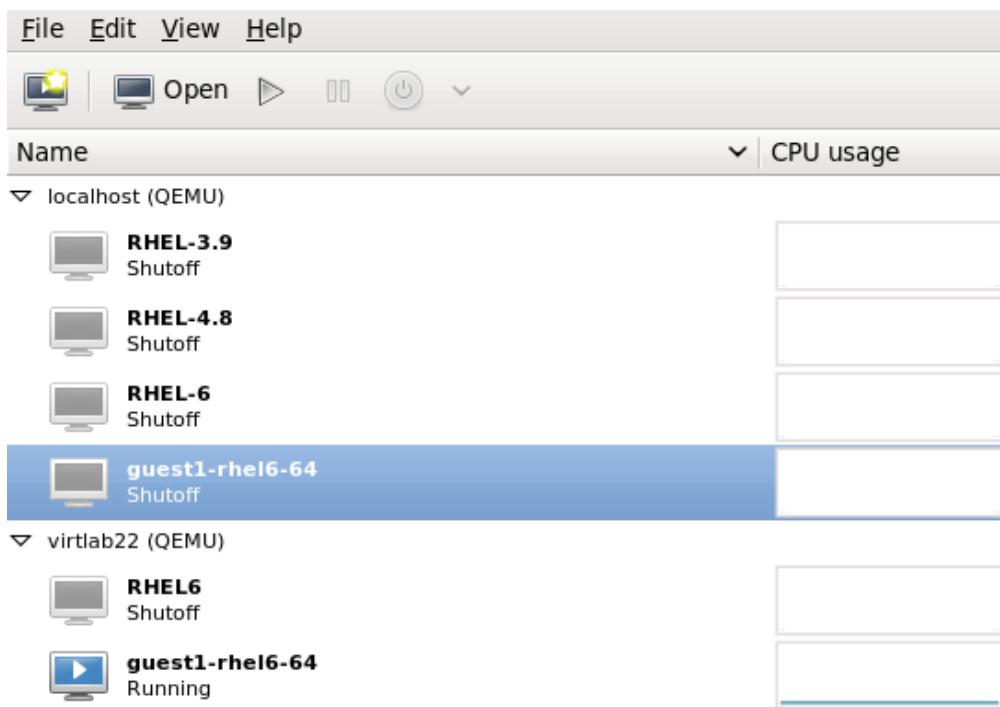


Figure 5.8. Migrated guest virtual machine running in the destination host physical machine

5. Optional - View the storage details for the host physical machine

In the **Edit** menu, click **Connection Details**, the Connection Details window appears.

Click the **Storage** tab. The iSCSI target details for the destination host physical machine is shown. Note that the migrated guest virtual machine is listed as using the storage

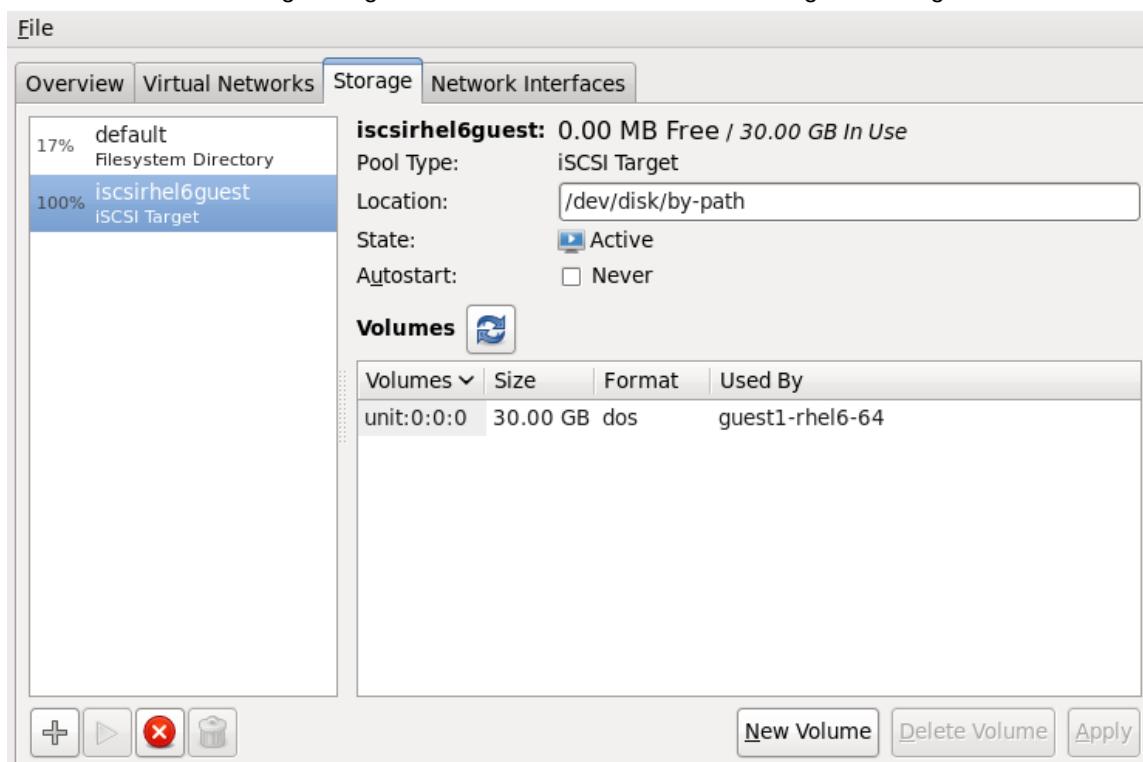


Figure 5.9. Storage details

This host was defined by the following XML configuration:

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

Figure 5.10. XML configuration for the destination host physical machine

Chapter 6. Remote management of guests

This section explains how to remotely manage your guests using **ssh** or TLS and SSL. More information on SSH can be found in the *Red Hat Enterprise Linux Deployment Guide*

6.1. Remote management with SSH

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

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

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



Note

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

The following packages are required for ssh access:

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

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

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



Important

SSH keys are user dependent and may only be used by their owners. A key's owner is the one who generated it. Keys may not be shared.

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

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

1. Optional: Changing user

Change user, if required. This example uses the local root user for remotely managing the other hosts and the local host.

```
$ su -
```

2. Generating the **SSH** key pair

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

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

3. Copying the keys to the remote hosts

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

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

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

Repeat for other systems, as required.

4. Optional: Add the passphrase to the ssh-agent

The instructions below describe how to add a passphrase to an existing ssh-agent. It will fail to run if the ssh-agent is not running. To avoid errors or conflicts make sure that your SSH parameters are set correctly. Refer to the *Red Hat Enterprise Linux Deployment Guide* for more information.

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 is added to the remote system.

The libvirt daemon (**libvirtd**)

The **libvirt** daemon provides 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.

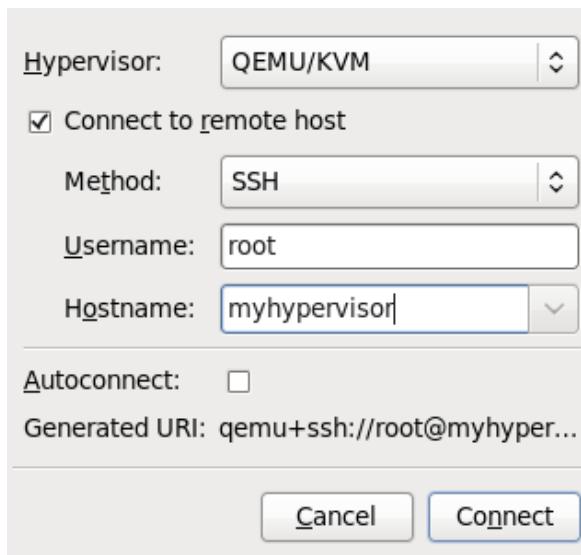


Figure 6.1. Add connection menu

3. Use the drop down menu to select hypervisor type, and click the **Connect to remote host** check box to open the Connection **Method** (in this case Remote tunnel over SSH), and enter the desired **User name** and **Hostname**, then click **Connect**.

6.2. Remote management over TLS and SSL

You can manage virtual machines using TLS and SSL. TLS and SSL provides greater scalability but is more complicated than ssh (refer to [Section 6.1, “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. The procedures that follow provide instructions on creating and deploying authentication certificates for TLS and SSL management.

Procedure 6.1. Creating a certificate authority (CA) key for TLS management

1. Before you begin, confirm that **certtool** is installed. If not:

```
# yum install certtool
```

2. Generate a private key, using the following command:

```
# certtool --generate-privkey > cakey.pem
```

3. Once the key generates, the next step is to create a signature file so the key can be self-signed. To do this, create a file with signature details and name it **ca.info**. This file should contain the following:

```
# vim ca.info
```

```
cn = Name of your organization
ca
cert_signing_key
```

4. Generate the self-signed key with the following command:

```
# certtool --generate-self-signed --load-privkey cakey.pem --template
ca.info --outfile cacert.pem
```

Once the file generates, the `ca.info` file may be deleted using the `rm` command. The file that results from the generation process is named **cacert.pem**. This file is the public key (certificate). The loaded file **cakey.pem** is the private key. This file should not be kept in a shared space. Keep this key private.

5. Install the **cacert.pem** Certificate Authority Certificate file on all clients and servers in the `/etc/pki/CA/cacert.pem` directory to let them know that the certificate issued by your CA can be trusted. To view the contents of this file, run:

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

This is all that is required to set up your CA. Keep the CA's private key safe as you will need it in order to issue certificates for your clients and servers.

Procedure 6.2. Issuing a server certificate

This procedure demonstrates how to issue a certificate with the X.509 CommonName (CN) field set to the hostname of the server. The CN must match the hostname which clients will be using to connect to the server. In this example, clients will be connecting to the server using the URI: `qemu://mycommonname/system`, so the CN field should be identical, ie `mycommonname`.

1. Create a private key for the server.

```
# certtool --generate-privkey > serverkey.pem
```

2. Generate a signature for the CA's private key by first creating a template file called **server.info**. Make sure that the CN is set to be the same as the server's hostname:

```
organization = Name of your organization
cn = mycommonname
tls_www_server
encryption_key
signing_key
```

3. Create the certificate with the following command:

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

4. This results in two files being generated:

- ▶ `serverkey.pem` - The server's private key
- ▶ `servercert.pem` - The server's public key

Make sure to keep the location of the private key secret. To view the contents of the file, perform the following command:

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

When opening this file the **CN=** parameter should be the same as the CN that you set earlier. For example, `mycommonname`.

5. Install the two files in the following locations:

- ▶ **serverkey.pem** - the server's private key. Place this file in the following location:
`/etc/pki/libvirt/private/serverkey.pem`
- ▶ **servercert.pem** - the server's certificate. Install it in the following location on the server:
`/etc/pki/libvirt/servercert.pem`

Procedure 6.3. Issuing a client certificate

1. For every client (ie. any program linked with libvirt, such as virt-manager), you need to issue a

certificate with the X.509 Distinguished Name (DN) set to a suitable name. This needs to be decided on a corporate level.

For example purposes the following information will be used:

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

This process is quite similar to [Procedure 6.2, “Issuing a server certificate”](#), with the following exceptions noted.

2. Make a private key with the following command:

```
# certtool --generate-privkey > clientkey.pem
```

3. Generate a signature for the CA's private key by first creating a template file called **client.info**. The file should contain the following (fields should be customized to reflect your region/location):

```
country = USA
state = North Carolina
locality = Raleigh
organization = Red Hat
cn = client1
tls_www_client
encryption_key
signing_key
```

4. Sign the certificate with the following command:

```
# certtool --generate-certificate --load-privkey clientkey.pem --load-ca-certificate cacert.pem \ --load-ca-privkey cakey.pem --template client.info --outfile clientcert.pem
```

5. Install the certificates on the client machine:

```
# cp clientkey.pem /etc/pki/libvirt/private/clientkey.pem
# cp clientcert.pem /etc/pki/libvirt/clientcert.pem
```

6.3. 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 (**libvиртd**) must be running on the remote machine. Port 22 must be open for SSH access. You should use some sort of SSH key management (for example, the **ssh-agent** utility) or you will be prompted for a password.

ext

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

TCP

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

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

Remote URIs

A Uniform Resource Identifier (URI) is used by **virsh** and *libvirt* to connect to a remote host. URIs can also be used with the **--connect** parameter for the **virsh** command to execute single commands or migrations on remote hosts. Remote URIs are formed by taking ordinary local URIs and adding a hostname and/or transport name. As a special case, using a URI scheme of 'remote', will tell the remote libvirtd server to probe for the optimal hypervisor driver. This is equivalent to passing a NULL URI for a local connection

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

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

Note that if the hypervisor(driver) is QEMU, the path is mandatory. If it is XEN, it is optional.

The following are examples of valid remote URIs:

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

The transport method or the hostname must be provided to target an external location. For more information refer to http://libvirt.org/guide/html/Application_Development_Guide-Architecture-Remote_URLs.html.

Examples of remote management parameters

- ▶ Connect to a remote KVM host named **host2**, using SSH transport and the SSH username **virtuser**. The connect command for each is **connect [<name>] [--readonly]**, where **<name>** is a valid URI as explained here. For more information about the **virsh connect** command refer to [Section 14.3, “Connecting to the hypervisor”](#)

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

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

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

Testing examples

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

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

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

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

Extra URI parameters

Extra parameters can be appended to remote URIs. The table below [Table 6.1. “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 6.1. Extra URI parameters

Name	Transport mode	Description	Example usage
name	all modes	The name passed to the remote <code>virConnectOpen</code> 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.	<code>name=qemu:///system</code>
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.	<code>command=/opt/openssl/bin/ssh</code>
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).	<code>socket=/opt/libvirt/run/libvirt/libvirt-sock</code>
netcat	ssh	<p>The netcat command can be used to connect to remote systems. The default netcat parameter uses the nc command. For SSH transport, libvirt constructs an SSH command using the form below:</p> <pre>command -p port [-l username] hostname</pre> <p>netcat -U socket</p> <p>The port, username and hostname parameters can be specified as part of the remote URI. The command, netcat and socket come from other extra parameters.</p>	<code>netcat=/opt/netcat/bin/nc</code>
no_verify	tls	If set to a non-zero value, this disables client checks of the	<code>no_verify=1</code>

		server's certificate. Note that to disable server checks of the client's certificate or IP address you must change the libvirtd configuration.
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 . Use this when you do not have access to a terminal .

Chapter 7. Overcommitting with KVM

7.1. Introduction

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

As most processes do not access 100% of their allocated memory all the time, KVM can use this behavior to its advantage and allocate more memory for guest virtual machines than the host physical machine actually has available, in a process called overcommitting of resources.



Important

Overcommitting is not an ideal solution for all memory issues as the recommended method to deal with memory shortage is to allocate less memory per guest so that the sum of all guests memory (+4G for the host O/S) is lower than the host physical machine's physical memory. If the guest virtual machines need more memory, then increase the guest virtual machines' swap space allocation. If however, should you decide to overcommit, do so with caution.

Guest virtual machines running on a KVM hypervisor do not have dedicated blocks of physical RAM assigned to them. Instead, each guest virtual machine functions as a Linux process where the host physical machine's Linux kernel allocates memory only when requested. In addition the host physical machine's memory manager can move the guest virtual machine's memory between its own physical memory and swap space. This is why overcommitting requires allotting sufficient swap space on the host physical machine to accommodate all guest virtual machines as well as enough memory for the host physical machine's processes. As a basic rule, the host physical machine's operating system requires a maximum of 4GB of memory along with a minimum of 4GB of swap space. Refer to [Example 7.1, “Memory overcommit example”](#) for more information.

Red Hat [Knowledgebase](#) has an article on safely and efficiently determining the size of the swap partition.



Note

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

Example 7.1. Memory overcommit example

This example demonstrates how to calculate swap space for overcommitting. Although it may appear to be simple in nature, the ramifications of overcommitting should not be ignored. Refer to [Important](#) before proceeding.

ExampleServer1 has 32GB of physical RAM. The system is being configured to run 50 guest virtual machines, each requiring 1GB of virtualized memory. As mentioned above, the host physical machine's system itself needs a maximum of 4GB (apart from the guest virtual machines) as well as an additional 4GB as a swap space minimum.

The swap space is calculated as follows:

- ▶ Calculate the amount of memory needed for the sum of all the guest virtual machines - In this example: (50 guest virtual machines * 1GB of memory per guest virtual machine) = 50GB
- ▶ Add the guest virtual machine's memory amount to the amount needed for the host physical machine's OS and for the host physical machine's minimum swap space - In this example: 50GB guest virtual machine memory + 4GB host physical machine's OS + 4GB minimal swap = 58GB
- ▶ Subtract this amount from the amount of physical RAM there is on the system - In this example 58GB - 32GB = 26GB
- ▶ The answer is the amount of swap space that needs to be allocated. In this example 26GB

Note

Overcommitting does not work with all guest virtual machines, but has been found to work in a desktop virtualization setup with minimal intensive usage or running several identical guest virtual machines with KSM. It should be noted that configuring swap and memory overcommit is not a simple plug-in and configure formula, as each environment and setup is different. Proceed with caution before changing these settings and make sure you completely understand your environment and setup before making any changes.

For more information on KSM and overcommitting, refer to [Chapter 8, KSM](#).

7.2. Overcommitting virtualized CPUs

The KVM hypervisor supports overcommitting virtualized CPUs. Virtualized CPUs can be overcommitted as far as load limits of guest virtual machines 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 virtual machine only has a single VCPU. The Linux scheduler is very efficient with this type of load. KVM should safely support guest virtual machines with loads under 100% at a ratio of five VCPUs. Overcommitting single VCPU guest virtual machines is not an issue.

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

Assigning guest virtual machines VCPUs up to the number of physical cores is appropriate and works as expected. For example, running guest virtual machines with four VCPUs on a quad core host. Guest virtual machines 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.

Chapter 8. KSM

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

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

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

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

Note

Starting in Red Hat Enterprise Linux 6.5, KSM is NUMA aware. This allows it to take NUMA locality into account while coalescing pages, thus preventing performance drops related to pages being moved to a remote node. Red Hat recommends avoiding cross-node memory merging when KSM is in use. If KSM is in use, change the **/sys/kernel/mm/ksm/merge_nodes** tunable to **0** to avoid merging pages across NUMA nodes. Kernel memory accounting statistics can eventually contradict each other after large amounts of cross-node merging. As such, numad can become confused after the KSM daemon merges large amounts of memory. If your system has a large amount of free memory, you may achieve higher performance by turning off and disabling the KSM daemon. Refer to the *Red Hat Enterprise Linux Performance Tuning Guide* for more information on NUMA.

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

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

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

The KSM service

The **ksm** service is included in the *qemu-kvm* package. KSM is off by default on Red Hat Enterprise Linux 6. When using Red Hat Enterprise Linux 6 as a KVM host physical machine, however, it is likely turned on by the **ksm/ksmtuned** services.

When the **ksm** service is not started, KSM shares only 2000 pages. This default is low and provides limited memory saving benefits.

When the **ksm** service is started, KSM will share up to half of the host physical machine system's main

memory. Start the **ksm** service to enable KSM to share more memory.

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

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

```
# chkconfig ksm on
```

The KSM tuning service

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

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

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

Before changing the parameters in the file, there are a few terms that need to be clarified:

- ▶ **npages** - How many pages ksm will scan before **ksmd** goes to sleep. It will be set at **/sys/kernel/mm/ksm/pages_to_scan**.
- ▶ **thres`** - Activation threshold, in kbytes. A KSM cycle is triggered when the `thres` value added to the sum of all `qemu-kvm` processes RSZ exceeds total system memory. This parameter is the equivalent in kbytes of the percentage defined in parameter `KSM_THRES_COEF`.

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

```
# Configuration file for ksmtuned.

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

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

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

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

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

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

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

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

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

KSM variables and monitoring

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

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

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

full_scans

Full scans run.

pages_shared

Total pages shared.

pages_sharing

Pages presently shared.

pages_to_scan

Pages not scanned.

pages_unshared

Pages no longer shared.

pages_volatile

Number of volatile pages.

run

Whether the KSM process is running.

sleep_millisecs

Sleep milliseconds.

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

Deactivating KSM

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

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

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

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

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

**Important**

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

Chapter 9. Advanced guest virtual machine administration

This chapter covers advanced administration tools for fine tuning and controlling system resources as they are made available to guest virtual machines.

9.1. Control Groups (cgroups)

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

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

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

9.2. Hugepage support

Introduction

x86 CPUs usually address memory in 4kB pages, but they are capable of using larger pages known as *hugepages*. KVM guests can be deployed with **hugepage** memory support in order to improve performance by increasing CPU cache hits against the *Transaction Lookaside Buffer (TLB)*. **hugepages** can significantly increase performance, particularly for large memory and memory-intensive workloads. Red Hat Enterprise Linux 6 is able to more effectively manage large amounts of memory by increasing the page size through the use of hugepages.

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

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

Using Transparent Hugepage Support

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

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

9.3. Running Red Hat Enterprise Linux as a guest virtual machine on a Hyper-V hypervisor

It is possible to run a Red Hat Enterprise Linux guest virtual machine on a Microsoft Windows host physical machine running the Microsoft Windows Hyper-V hypervisor. In particular, the following enhancements have been made to allow for easier deployment and management of Red Hat Enterprise Linux guest virtual machines:

- ▶ Upgraded VMBUS protocols - VMBUS protocols have been upgraded to Windows 8 level. As part of this work, now VMBUS interrupts can be processed on all available virtual CPUs in the guest. Furthermore, the signaling protocol between the Red Hat Enterprise Linux guest virtual machine and

the Windows host physical machine has been optimized.

- ▶ Synthetic frame buffer driver - Provides enhanced graphics performance and superior resolution for Red Hat Enterprise Linux desktop users.
- ▶ Live Virtual Machine Backup support - Provisions uninterrupted backup support for live Red Hat Enterprise Linux guest virtual machines.
- ▶ Dynamic expansion of fixed size Linux VHDs - Allows expansion of live mounted fixed sized Red Hat Enterprise Linux VHDs.
- ▶ Red Hat Enterprise Linux Dynamic memory support - Provides full hot-add and ballooning support thereby improving Windows guest virtual machine density when used on Red Hat Enterprise Linux host physical machines.

For more information, refer to the following article: "[Enabling Linux Support on Windows Server 2012 R2 Hyper-V](#)".

Chapter 10. Miscellaneous administration tasks

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

10.1. Automatically starting guest virtual machines

This section covers how to make guest virtual machines start automatically during the host physical machine system's boot phase.

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

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

The guest virtual machine now automatically starts with the host physical machine.

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

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

The guest virtual machine no longer automatically starts with the host physical machine.

10.2. Guest virtual machine memory allocation

The following procedure shows how to allocate memory for a guest virtual machine. This allocation and assignment works only at boot time and any changes to any of the memory values will not take effect until the next reboot. The maximum memory that can be allocated per guest is 4 TiB, providing that this memory allocation isn't more than what the host physical machine resources can provide.

Valid memory units include:

- ▶ **b** or **bytes** for bytes
- ▶ **KB** for kilobytes (10^3 or blocks of 1,000 bytes)
- ▶ **k** or **KiB** for kibibytes (2^{10} or blocks of 1024 bytes)
- ▶ **MB** for megabytes (10^6 or blocks of 1,000,000 bytes)
- ▶ **M** or **MiB** for mebibytes (2^{20} or blocks of 1,048,576 bytes)
- ▶ **GB** for gigabytes (10^9 or blocks of 1,000,000,000 bytes)
- ▶ **G** or **GiB** for gibibytes (2^{30} or blocks of 1,073,741,824 bytes)
- ▶ **TB** for terabytes (10^{12} or blocks of 1,000,000,000,000 bytes)
- ▶ **T** or **TiB** for tebibytes (2^{40} or blocks of 1,099,511,627,776 bytes)

Note that all values will be rounded up to the nearest kibibyte by libvirt, and may be further rounded to the granularity supported by the hypervisor. Some hypervisors also enforce a minimum, such as 4000KiB (or 4000×2^{10} or 4,096,000 bytes). The units for this value are determined by the optional attribute **memory unit**, which defaults to the kibibytes (KiB) as a unit of measure where the value given is multiplied by 2^{10} or blocks of 1024 bytes.

In the cases where the guest virtual machine crashes the optional attribute **dumpCore** can be used to control whether the guest virtual machine's memory should be included in the generated coredump (**dumpCore='on'**) or not included (**dumpCore='off'**). Note that the default setting is **on** so if the parameter is not set to **off**, the guest virtual machine memory will be included in the coredump file.

The ***currentMemory*** attribute determines the actual memory allocation for a guest virtual machine. This value can be less than the maximum allocation, to allow for ballooning up the guest virtual machines memory on the fly. If this is omitted, it defaults to the same value as the memory element. The unit attribute behaves the same as for memory.

In all cases for this section, the domain XML needs to be altered as follows:

```
<domain>
  <memory unit='KiB' dumpCore='off'>524288</memory>
  <!-- changes the memory unit to KiB and does not allow the guest virtual
  machine's memory to be included in the generated coredump file -->
  <currentMemory unit='KiB'>524288</currentMemory>
  <!-- makes the current memory unit 524288 KiB -->
  ...
</domain>
```

10.3. Assigning USB devices to guest virtual machines

USB devices such as web cameras, card readers, disk drives, etc. need to be attached to a host physical machine in order to use the services they provide. Using **usb-redirect** the USB device can be physically attached to a host physical machine and it can be used by the guest virtual machine. This process can be automated to work as soon as the device is attached, or it can be set manually.

Note that this feature requires that the guest virtual machine is running the SPICE display protocol. All other configurations are done on the host physical machine. When the guest virtual machine is running, it will see an additional emulated USB device connected to the emulated USB controller. USB redirect cannot be performed on stopped or paused guest virtual machines. In addition the running guest virtual machine must have active window focus.

USB devices can be redirected via the command line or **virt manager**.

10.3.1. Redirecting USB devices using libvirt

information to go here

10.4. Using qemu-img

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

Check

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

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



Note

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

Commit

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

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

Convert

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

Command format:

```
# qemu-img convert [-c] [-p] [-f fmt] [-t cache] [-O output_fmt] [-o options] [-S sparse_size] filename output_filename
```

The **-p** parameter shows the progress of the command (optional and not for every command) and **-S** flag allows for the creation of a *sparse file*, which is included within the disk image. Sparse files in all purposes function like a standard file, except that the physical blocks that only contain zeros (i.e., nothing). When the Operating System sees this file, it treats it as it exists and takes up actual disk space, even though in reality it doesn't take any. This is particularly helpful when creating a disk for a guest virtual machine as this gives the appearance that the disk has taken much more disk space than it has. For example, if you set **-S** to 50Gb on a disk image that is 10Gb, then your 10Gb of disk space will appear to be 60Gb in size even though only 10Gb is actually being used.

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

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

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

Create

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

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

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

Info

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

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

This command is often used to discover the size reserved on disk which can be different from the displayed size. If snapshots are stored in the disk image, they are displayed also. This command will show for example, how much space is being taken by a qcow2 image on a block device. This is done by running the **qemu-img**. You can check that the image in use is the one that matches the output of the **qemu-img info** command with the **qemu-img check** command. Refer to [Section 10.4, “Using qemu-img”](#).

```
# qemu-img info /dev/vg-90.100-sluo/lv-90-100-sluo
image: /dev/vg-90.100-sluo/lv-90-100-sluo
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 0
cluster_size: 65536
```

Rebase

Changes the backing file of an image.

```
# qemu-img rebase [-f fmt] [-t cache] [-p] [-u] -b Backing_file [-F Backing_fmt]
filename
```

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



Note

Only the **qcow2** format supports changing the backing file (rebase).

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

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

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

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

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

Resize

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

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

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

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

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



Warning

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

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

Snapshot

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

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

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

Supported formats

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

raw

Raw disk image format (default). This can be the fastest file-based format. If your file system supports holes (for example in ext2 or ext3 on Linux or NTFS on Windows), then only the written sectors will reserve space. Use **qemu-img info** to obtain the real size used by the image or **ls -ls** on Unix/Linux. Although Raw images give optimal performance, only very basic features are available with a Raw image (no snapshots etc.).

qcow2

QEMU image format, the most versatile format with the best feature set. Use it to have optional AES encryption, zlib-based compression, support of multiple VM snapshots, and smaller images, which are useful on file systems that do not support holes (non-NTFS file systems on Windows). Note that this expansive feature set comes at the cost of performance.

Although only the formats above can be used to run on a guest virtual machine or host physical machine, **qemu-img** also recognizes and supports the following formats in order to convert from them into either **raw** or **qcow2** format. The format of an image is usually detected automatically. In addition to converting these formats into **raw** or **qcow2**, they can be converted back from **raw** or **qcow2** to the original format.

bochs

Bochs disk image format.

cloop

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

cow

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

dmg

Mac disk image format.

nbd

Network block device.

parallels

Parallels virtualization disk image format.

qcow

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

vdi

Oracle VM VirtualBox hard disk image format.

vmddk

VMware 3 and 4 compatible image format.

vpc

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

vvfat

Virtual VFAT disk image format.

10.5. Verifying virtualization extensions

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

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

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

- Analyze the output.

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

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

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

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

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

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

The virtualization extensions may be disabled in the BIOS. If the extensions do not appear or full virtualization does not work refer to [Procedure 21.1, “Enabling virtualization extensions in BIOS”](#).

3. Ensure KVM subsystem is loaded

As an additional check, verify that the **kvm** modules are loaded in the kernel:

```
# lsmod | grep kvm
```

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

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.

10.6. Setting KVM processor affinities

Note

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

This section covers setting processor and processing core affinities with **libvirt** and KVM guest virtual machines.

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

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

Identifying CPU and NUMA topology

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

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

This output shows that the system has eight CPU cores and two sockets. Each CPU socket has four

cores. This splitting of CPU cores across multiple sockets suggests that the system has Non-Uniform Memory Access (NUMA) architecture.

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

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

[ Additional XML removed ]

</capabilities>
```

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

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

Decide which NUMA node can run the guest virtual machine

Locking a guest virtual machine to a particular NUMA node offers no benefit if that node does not have sufficient free memory for that guest virtual machine. libvirt stores information on the free memory

available on each node. Use the **virsh freecell --all** command to display the free memory on all NUMA nodes.

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

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

Lock a guest virtual machine to a NUMA node or physical CPU set

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

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

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

2. Observe that the node 1, **<cell id='1'>**, uses physical CPUs 4 to 7.
3. The guest virtual machine can be locked to a set of CPUs by appending the **cpuset** attribute to the configuration file.
 - a. While the guest virtual machine is offline, open the configuration file with **virsh edit**.
 - b. Locate the guest virtual machine's virtual CPU count, defined in the **vcpus** element.

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

The guest virtual machine 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 virtual machine.

The guest virtual machine has been locked to CPUs 4 to 7.

Automatically locking guest virtual machines to CPUs with virt-install

The **virt-install** provisioning tool provides a simple way to automatically apply a 'best fit' NUMA policy when guest virtual machines 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 guest virtual machines.

Tuning CPU affinity on running guest virtual machines

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

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

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

```
# 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
```

10.7. Generating a new unique MAC address

In some cases you will need to generate a new and unique MAC address for a guest virtual machine. 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 guest virtual machines. Save the script on your guest virtual machine 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 guest virtual machines
#
import random
#
def randomMAC():
    mac = [ 0x00, 0x16, 0x3e,
            random.randint(0x00, 0x7f),
            random.randint(0x00, 0xff),
            random.randint(0x00, 0xff) ]
    return ':' .join(map(lambda x: "%02x" % x, mac))
#
print randomMAC()
```

Another method to generate a new MAC for your guest virtual machine

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

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

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

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

10.8. Improving guest virtual machine response time

Guest virtual machines can sometimes be slow to respond with certain workloads and usage patterns. Examples of situations which may cause slow or unresponsive guest virtual machines:

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

KVM guest virtual machines function as Linux processes. Linux processes are not permanently kept in main memory (physical RAM) and will be placed into swap space (virtual memory) especially if they are not being used. If a guest virtual machine is inactive for long periods of time, the host physical machine kernel may move the guest virtual machine into swap. As swap is slower than physical memory it may appear that the guest is not responding. This changes once the guest is loaded into the main memory. Note that the process of loading a guest virtual machine from swap to main memory may take several seconds per gigabyte of RAM assigned to the guest virtual machine, depending on the type of storage used for swap and the performance of the components.

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

Using unsafe overcommit levels or overcommitting with swap turned off guest virtual machine processes or other critical processes is not recommended. Always ensure the host physical machine has sufficient swap space when overcommitting memory.

For more information on overcommitting with KVM, refer to [Section 7.1, “Introduction”](#).



Warning

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

If swap is turned off, do not overcommit guest virtual machines. Overcommitting guest virtual machines without any swap can cause guest virtual machines or the host physical machine system to crash.

Turning off swap

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

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

```
# swapoff -a
```

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

physical machine system.

Using SSDs for swap

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

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

10.9. Disable SMART disk monitoring for guest virtual machines

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

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

10.10. 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.

Use the following step set up a dedicated VNC server session:

If needed, Create and then 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. For more information on vnc server files refer to the *Red Hat Enterprise Linux Installation Guide*.

10.11. Gracefully shutting down guest virtual machines using virsh shutdown

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

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

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



Note

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

Procedure 10.1. Workaround for Red Hat Enterprise Linux 6

1. Install the acpid package

The **acpid** service listen and processes ACPI requests.

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

```
# yum install acpid
```

2. Enable the acpid service

Set the **acpid** service to start during the guest virtual machine boot sequence and start the service:

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

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

In addition to the method described above, a guest can be automatically shutdown, by stopping the *libvirt-guest* service. Refer to [Section 10.12, “Manipulating the libvirt-guests configuration settings”](#) for more information on this method.

10.12. Manipulating the libvirt-guests configuration settings

The *libvirt-guests* service has parameter settings that can be configured to assure that the guest is shutdown properly. It is a package that is a part of the libvirt installation and is installed by default. This service automatically saves guests to the disk when the host shuts down, and restores them to their pre-shutdown state when the host reboots. By default, this setting is set to suspend the guest. If you want the guest to be shutoff, you will need to change one of the parameters of the *libvirt-guests* configuration file.

Procedure 10.2. Changing the libvirt-guests service parameters to allow for the graceful shutdown of guests

The procedure described here allows for the graceful shutdown of guest virtual machines when the host physical machine is stuck, powered off, or needs to be restarted.

1. Open the configuration file

The configuration file is located in **/etc/sysconfig/libvirt-guests**. Edit the file, remove the comment mark (#) and change the **ON_SHUTDOWN=suspend** to **ON_SHUTDOWN=shutdown**.

Remember to save the change.

```
$ vi /etc/sysconfig/libvirt-guests

# URIs to check for running guests
# example: URIS='default xen:/// vbox+tcp://host/system lxc:///'
#URIS=default ①

# action taken on host boot
# - start all guests which were running on shutdown are started on boot
# regardless on their autostart settings
# - ignore libvirt-guests init script won't start any guest on boot,
however,
#           guests marked as autostart will still be automatically started by
#           libvirtd
#ON_BOOT=start ②

# Number of seconds to wait between each guest start. Set to 0 to allow
# parallel startup.
#START_DELAY=0 ③

# action taken on host shutdown
# - suspend all running guests are suspended using virsh managedsave
# - shutdown all running guests are asked to shutdown. Please be careful with
#   this settings since there is no way to distinguish between a
#   guest which is stuck or ignores shutdown requests and a guest
#   which just needs a long time to shutdown. When setting
#   ON_SHUTDOWN=shutdown, you must also set SHUTDOWN_TIMEOUT to a
#   value suitable for your guests.
ON_SHUTDOWN=shutdown ④

# If set to non-zero, shutdown will suspend guests concurrently. Number of
# guests on shutdown at any time will not exceed number set in this variable.
#PARALLEL_SHUTDOWN=0 ⑤

# Number of seconds we're willing to wait for a guest to shut down. If
parallel
# shutdown is enabled, this timeout applies as a timeout for shutting down all
# guests on a single URI defined in the variable URIS. If this is 0, then
there
# is no time out (use with caution, as guests might not respond to a shutdown
# request). The default value is 300 seconds (5 minutes).
#SHUTDOWN_TIMEOUT=300 ⑥

# If non-zero, try to bypass the file system cache when saving and
# restoring guests, even though this may give slower operation for
# some file systems.
#BYPASS_CACHE=0 ⑦
```

- ① **URIS** - checks the specified connections for a running guest. The **Default** setting functions in the same manner as *virsh* does when no explicit URI is set. In addition, one can explicitly set the URI from **/etc/libvirt/libvirt.conf**. It should be noted that when using the libvirt configuration file default setting, no probing will be used.
- ② **ON_BOOT** - specifies the action to be done to / on the guests when the host boots. The **start** option starts all guests that were running prior to shutdown regardless on their autostart settings. The **ignore** option will not start the formally running guest on boot, however, any guest marked as autostart will still be automatically started by *libvirtd*.
- ③ The **START_DELAY** - sets a delay interval inbetween starting up the guests. This time period is set in seconds. Use the 0 time setting to make sure there is no delay and that all guests are started simultaneously.
- ④ **ON_SHUTDOWN** - specifies the action taken when a host shuts down. Options that can be set include: **suspend** which suspends all running guests using **virsh managedsave** and **shutdown** which shuts down all running guests. It is best to be careful with using the

shutdown option as there is no way to distinguish between a guest which is stuck or ignores shutdown requests and a guest that just needs a longer time to shutdown. When setting the **ON_SHUTDOWN=shutdown**, you must also set **SHUTDOWN_TIMEOUT** to a value suitable for the guests.

- 5. **PARALLEL_SHUTDOWN** Dictates that the number of guests on shutdown at any time will not exceed number set in this variable and the guests will be suspended concurrently. If set to **0**, then guests are not shutdown concurrently.
- 6. Number of seconds to wait for a guest to shut down. If **SHUTDOWN_TIMEOUT** is enabled, this timeout applies as a timeout for shutting down all guests on a single URI defined in the variable URIS. If **SHUTDOWN_TIMEOUT** is set to **0**, then there is no time out (use with caution, as guests might not respond to a shutdown request). The default value is 300 seconds (5 minutes).
- 7. **BYPASS_CACHE** can have 2 values, 0 to disable and 1 to enable. If enabled it will by-pass the file system cache when guests are restored. Note that setting this may effect performance and may cause slower operation for some file systems.

2. Start libvirt-guests service

If you have not started the service, start the *libvirt-guests* service. Do not restart the service as this will cause all running domains to shutdown.

10.13. Virtual machine timer management with libvirt

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

`<clock>`

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

» **offset**

Determines how the guest virtual machine clock is offset from the host physical machine clock. The offset attribute has the following possible values:

Table 10.1. Offset attribute values

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



Note

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

▶ **timezone**

The timezone to which the guest virtual machine clock is to be synchronized.

▶ **adjustment**

The delta for guest virtual machine clock synchronization. In seconds, relative to UTC.

Example 10.1. Always synchronize to UTC

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

Example 10.2. Always synchronize to the host physical machine timezone

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

Example 10.3. Synchronize to an arbitrary timezone

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

Example 10.4. Synchronize to UTC + arbitrary offset

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

<timer>

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

▶ **name**

The name of the time source to use.

Table 10.2. name attribute values

Value	Description
platform	The master virtual time source which may be used to drive the policy of other time sources.
pit	Programmable Interval Timer - a timer with periodic interrupts.
rtc	Real Time Clock - a continuously running timer with periodic interrupts.
hpet	High Precision Event Timer - multiple timers with periodic interrupts.
tsc	Time Stamp Counter - counts the number of ticks since reset, no interrupts.
kvmclock	KVM clock - recommended clock source for KVM guest virtual machines. KVM pvclock, or kvm-clock lets guest virtual machines read the host physical machine's wall clock time.

▶ **track**

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

Table 10.3. track attribute values

Value	Description
boot	Corresponds to old <i>host physical machine</i> option, this is an unsupported tracking option.
guest	RTC always tracks guest virtual machine time.
wall	RTC always tracks host time.

▶ **tickpolicy**

The policy used to pass ticks on to the guest virtual machine.

Table 10.4. tickpolicy attribute values

Value	Description
delay	Continue to deliver at normal rate (i.e. ticks are delayed).
catchup	Deliver at a higher rate to catch up.
merge	Ticks merged into one single tick.
discard	All missed ticks are discarded.

▶ **frequency**

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

▶ **mode**

Determines how the time source is exposed to the guest virtual machine. This attribute is only relevant for a name value of *tsc*. All other timers are always emulated. Command is as follows:

```
<timer name='tsc' frequency='NNN' mode='auto|native|emulate|smptsafe'/>
```

Mode definitions are given in the table.

Table 10.5. mode attribute values

Value	Description
auto	Native if TSC is unstable, otherwise allow native TSC access.
native	Always allow native TSC access.
emulate	Always emulate TSC.
smpsafe	Always emulate TSC and interlock SMP

▶ **present**

Used to override the default set of timers visible to the guest virtual machine. For example, to enable or disable the HPET.

Table 10.6. present attribute values

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

Example 10.5. Clock synchronizing to local time with RTC and PIT timers, and the HPET timer disabled

```
<clock offset="localtime">
  <timer name="rtc" tickpolicy="catchup" track="guest virtual machine" />
  <timer name="pit" tickpolicy="delay" />
  <timer name="hpet" present="no" />
</clock>
```

10.14. Using PMU to monitor guest virtual machine performance

In Red Hat Enterprise Linux 6.4, vPMU (virtual PMU) was introduced as technical-preview. vPMU is based on Intel's PMU (Performance Monitoring Units) and may only be used on Intel machines. PMU allows the tracking of statistics which indicate how a guest virtual machine is functioning.

Using performance monitoring, allows developers to use the CPU's PMU counter while using the performance tool for profiling. The virtual performance monitoring unit feature allows virtual machine users to identify sources of possible performance problems in their guest virtual machines, thereby improving the ability to profile a KVM guest virtual machine.

To enable the feature, the **-cpu host** flag must be set.

This feature is only supported with guest virtual machines running Red Hat Enterprise Linux 6 and is disabled by default. This feature only works using the Linux perf tool. Make sure the *perf* package is installed using the command:

```
# yum install perf.
```

See the man page on **perf** for more information on the perf commands.

10.15. Guest virtual machine power management

It is possible to forcibly enable or disable BIOS advertisements to the guest virtual machine's operating system by changing the following parameters in the Domain XML for Libvirt:

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

The element ***pm*** enables ('yes') or disables ('no') BIOS support for S3 (suspend-to-disk) and S4 (suspend-to-mem) ACPI sleep states. If nothing is specified, then the hypervisor will be left with its default value.

10.16. QEMU guest virtual machine agent protocol

The QEMU guest virtual machine agent protocol (QEMU GA), uses the same protocol as QMP. The QEMU GA package, *qemu-guest-agent*, is fully supported in Red Hat Enterprise Linux 6.5. There are some issues regarding its isa-serial/virtio-serial transport, and the following caveats have been noted:

- ▶ There is no way for *qemu-guest-agent* to detect whether or not a client has connected to the channel.
- ▶ There is no way for a client to detect whether or not *qemu-guest-agent* has disconnected or reconnected to the backend.
- ▶ If the virtio-serial device resets and *qemu-guest-agent* has not connected to the channel as a result, (generally caused by a reboot or hotplug), data from the client will be dropped.
- ▶ If *qemu-guest-agent* has connected to the channel following a virtio-serial device reset, data from the client will be queued (and eventually throttled if available buffers are exhausted), regardless of whether or not *qemu-guest-agent* is still running/connected.

QEMU GA uses the guest virtual machine-sync or guest virtual machine-sync-delimited command to address the problem of re-synchronizing the channel after re-connection or client-side timeouts. These are described below.

10.16.1. guest-sync

The guest-sync request/response exchange is simple. The client provides a unique numerical token, the agent sends it back in a response:

```
> { "execute": "guest-sync", "arguments": { "id": 123456 } }
< { "return": 123456 }
```

A successful exchange guarantees that the channel is now in sync and no unexpected data/responses will be sent. Note that for the reasons mentioned above there's no guarantee this request will be answered, so a client should implement a timeout and re-issue this periodically until a response is received for the most recent request.

This alone does not handle synchronization issues in all cases. For example, if *qemu-guest-agent*'s parser previously received a partial request from a previous client connection, subsequent attempts to issue the guest-sync request can be misconstrued as being part of the previous partial request.

Eventually *qemu-guest-agent* will hit its recursion or token size limit and flush its parser state, at which point it will begin processing the backlog of requests, but there's no guarantee this will occur before the channel is throttled due to exhausting all available buffers. Thus, there is a potential for a deadlock situation occurring for certain instances.

To avoid this, *qemu-guest-agent/QEMU*'s JSON parser has special handling for the 0xFF byte, which is an invalid UTF-8 character. Client requests should precede the guest-sync request with to ensure that

qemu-guest-agent flushes its parser state as soon as possible. As long as all clients abide by this, the deadlock state should be reliably avoidable.

For more information see the *qemu-guest-agent* wiki page on wiki.qemu.org.

10.16.2. guest-sync-delimited

If *qemu-guest-agent* attempts to communicate with a client, and the client receives a partial response from a previous *qemu-guest-agent* instance, the client might misconstrue responses to guest-sync as being part of this previous request. For client implementations that treat newlines as a delimiter for *qemu-guest-agent* responses, use **guest-sync-delimited**.

Even in some cases where there are JSON stream-based implementations that do not rely on newline delimiters, it may be considered invasive to implement a client's response/JSON handling, as it is the same deadlock scenario described previously. Using the **guest-sync-delimited** on the client, tells QEMU GA to place the same 0xFF character in front of the response, thereby preventing confusion.

```
> { "execute": "guest-sync-delimited", "arguments": { "id": 123456 } }
< { "return": 123456}
```

Actual hex values sent:

```
> 7b 27 65 78 65 63 75 74 65 27 3a 27 67 75 65 73 74 2d 73 79 6e 63 2d 64 65
  6c 69 6d 69 74 65 64 27 2c 27 61 72 67 75 6d 65 6e 74 73 27 3a 7b 27 69 64
  27 3a 31 32 33 34 35 36 7d 7d 0a
< ff 7b 22 72 65 74 75 72 6e 22 3a 20 31 32 33 34 35 36 7d 0a
```

As stated above, the request should also be preceded with a 0xFF to flush *qemu-guest-agent*'s parser state.

10.16.3. Creating a guest virtual machine disk backup

In Red Hat Enterprise Linux 6.4, the QEMU GA provided protection against the corruption of Linux guest virtual machines. Just before issuing the snapshot request, or creating a backup copy of the disk, the management stack (*libvirt*) sends a **guest-fsfreeze-freeze** QMP command to the QEMU GA via the virtio-serial port. This command causes the guest agent to freeze all of the guest virtual machine's filesystems, via the **FIFREEZE ioctl()** kernel function. This **ioctl()** function is implemented by the Linux kernel in the guest virtual machine. The function flushes the filesystem cache in the guest virtual machine's kernel, brings the filesystem into a consistent state, and denies all userspace threads write access to the filesystem.

Only after the QEMU GA reported success, *libvirt* would proceed with the snapshot. At its completion, *libvirt* sends the **guest-fsfreeze-thaw** QMP command to the QEMU GA over the virtio-serial port. This command tells the QEMU GA to issue a **FITHAW ioctl()**, which unblocks the userspace threads that were previously denied write access, and resumes normal processing. The caveat of this process is that it didn't ensure that application-level data is in a consistent state when the virtual disk snapshot is taken. This is evident in cases where the *fsck* utility doesn't find any problems on the filesystem restored from a snapshot, and yet applications are not able to resume processing from the point where the snapshot was taken and userspace processes may not have written their internal buffers to files on the disk.

Improvements in Red Hat Enterprise Linux 6.5 have been made to make sure that both file and application level synchronization (flushing) is done. Guest system administrators can write and install application-specific freezing and thawing hook scripts. Before freezing the filesystems, the QEMU GA invokes the main hook script (included in the QEMU GA package). The main hook script in turn calls individual application-specific scripts, prepared by the guest system administrators, that temporarily deactivates all guest virtual machine applications. All of these actions are done when the mode is changed to freeze.

Just before filesystems are frozen, the guest system administrator's scripts trigger the databases and

other file system applications to flush their working buffers to the virtual disk and to stop accepting further client connections. The applications should bring their data files into a consistent state where resumption of processing, with the reactivated (or a freshly started) instance of the application (after restoring the virtual disk from backup) would be possible. When all scripts are done inactivating their respective applications, and the main hook script returns, QEMU GA proceeds to freeze filesystems, and the management stack takes the snapshot. Once all this is done, and it is confirmed that the snapshot is taken, the file system will resume to serve write requests. This process is called thawing.

Thawing happens in reverse order. Instructed by *libvirt*, QEMU GA thaws the guest virtual machine's filesystems. It then invokes individual hook scripts (via the main hook script) to resume or restart applications that had been inactivated during the freeze process.

Tips for Windows guests

Windows guest virtual machines will use QEMU GA for windows, *qemu-guest-agent-win*. This agent allows for VSS (Volume Shadow Copy Service) support for Windows guest virtual machines running on Red Hat Enterprise Linux. More information can be found [here](#).

Tips when using SELinux

An application specific hook script might need various SELinux permissions in order to run correctly. As is done when the script needs to connect to a socket in order to talk to a database. In general, local SELinux policies should be developed and installed for such purposes. Accessing file system nodes should work out of the box, after issuing the **restorecon -FvvR** command listed in [Table 10.7, “QEMU guest agent package contents”](#) in the table row labeled **/usr/libexec/qemu-ga/fsfreeze-hook.d/**.

The *qemu-guest-agent* binary RPM includes the following files:

Table 10.7. QEMU guest agent package contents

File name	Description
/etc/rc.d/init.d/qemu-ga	Service control script (start/stop) for the QEMU GA.
/etc/sysconfig/qemu-ga	Configuration file for the QEMU guest agent, as it is read by the /etc/rc.d/init.d/qemu-ga control script. The settings are documented in the file with shell script comments.
/usr/bin/qemu-ga	QEMU GA binary file.
/usr/libexec/qemu-ga/	Root directory for hook scripts.
/usr/libexec/qemu-ga/fsfreeze-hook	Main hook script. No modifications are needed here.
/usr/libexec/qemu-ga/fsfreeze-hook.d/	Directory for individual, application-specific hook scripts. The guest system administrator should copy hook scripts manually into this directory, ensure proper file mode bits for them, and then run restorecon -FvvR on this directory.
/usr/share/qemu-kvm/qemu-ga/	Directory with sample scripts (for example purposes only). The scripts contained here are not executed.

The main hook script, **/usr/libexec/qemu-ga/fsfreeze-hook** logs its own messages, as well as the application-specific scripts' standard output and error messages, in the following log file:

`/var/log/qemu-ga/fsfreeze-hook.log`. For more information, refer to the qemu-guest-agent wiki page on wiki.qemu.org or libvirt.org.

10.17. Setting a limit on device redirection

To filter out certain devices from redirection, pass the filter property to `-device usb-redir`. The filter property takes a string consisting of filter rules, the format for a rule is:

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

Use the value `-1` to designate it to accept any value for a particular field. You may use multiple rules on the same command line using `|` as a separator. Note that if a device matches none of the passed in rules, redirecting it will not be allowed!

Example 10.6. An example of limiting redirection with a windows guest virtual machine

1. Prepare a Windows 7 guest virtual machine.
2. Add the following code excerpt to the guest virtual machine's domain xml file:

```
<redirdev bus='usb' type='spicevmc'>
    <alias name='redir0' />
    <address type='usb' bus='0' port='3' />
</redirdev>
<redirfilter>
    <usbdev class='0x08' vendor='0x1234' product='0xBEEF' version='2.0'
allow='yes' />
    <usbdev class='-1' vendor='-1' product='-1' version=' -1' allow='no' />
</redirfilter>
```

3. Start the guest virtual machine and confirm the setting changes by running the following:

```
#ps -ef | grep $guest_name
```

```
-device usb-redir,chardev=charredir0,id=redir0,/
filter=0x08:0x1234:0xBEEF:0x0200:1|-1:-1:-1:0,bus=usb.0,port=3
```

4. Plug a USB device into a host physical machine, and use `virt-viewer` to connect to the guest virtual machine.
5. Click **USB device selection** in the menu, which will produce the following message: "Some USB devices are blocked by host policy". Click **OK** to confirm and continue.
The filter takes effect.
6. To make sure that the filter captures properly check the USB device vendor and product, then make the following changes in the host physical machine's domain XML to allow for USB redirection.

```
<redirfilter>
    <usbdev class='0x08' vendor='0x0951' product='0x1625' version='2.0'
allow='yes' />
    <usbdev allow='no' />
</redirfilter>
```

7. Restart the guest virtual machine, then use `virt-viewer` to connect to the guest virtual machine.
The USB device will now redirect traffic to the guest virtual machine.

10.18. Dynamically changing a host physical machine or a network bridge that is attached to a virtual NIC

This section demonstrates how to move the vNIC of a guest virtual machine from one bridge to another while the guest virtual machine is running without compromising the guest virtual machine

1. Prepare guest virtual machine with a configuration similar to the following:

```
<interface type='bridge'>
    <mac address='52:54:00:4a:c9:5e' />
    <source bridge='virbr0' />
    <model type='virtio' />
</interface>
```

2. Prepare an XML file for interface update:

```
# cat br1.xml
```

```
<interface type='bridge'>
    <mac address='52:54:00:4a:c9:5e' />
    <source bridge='virbr1' />
    <model type='virtio' />
</interface>
```

3. Start the guest virtual machine, confirm the guest virtual machine's network functionality, and check that the guest virtual machine's vnetX is connected to the bridge you indicated.

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

4. Update the guest virtual machine's network with the new interface parameters with the following command:

```
# virsh update-device test1 br1.xml
Device updated successfully
```

5. On the guest virtual machine, run **service network restart**. The guest virtual machine gets a new IP address for virbr1. Check the guest virtual machine's vnet0 is connected to the new bridge(virbr1)

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

Chapter 11. Storage concepts

This chapter introduces the concepts used for describing and managing storage devices. Terms such as Storage pools and Volumes are explained in the sections that follow.

11.1. Storage pools

A *storage pool* is a file, directory, or storage device managed by libvirt for the purpose of providing storage to guest virtual machines. The storage pool can be local or it can be shared over a network. A storage pool is a quantity of storage set aside by an administrator, often a dedicated storage administrator, for use by guest virtual machines. Storage pools are divided into storage volumes either by the storage administrator or the system administrator, and the volumes are assigned to guest virtual machines as block devices. In short storage volumes are to partitions what storage pools are to disks. Although the storage pool is a virtual container it is limited by two factors: maximum size allowed to it by qemu-kvm and the size of the disk on the host physical machine. Storage pools may not exceed the size of the disk on the host physical machine. The maximum sizes are as follows:

- ▶ virtio-blk = 2^{63} bytes or 8 Exabytes(using raw files or disk)
- ▶ Ext4 = ~ 16 TB (using 4 KB block size)
- ▶ XFS = ~8 Exabytes
- ▶ qcow2 and host file systems keep their own metadata and scalability should be evaluated/tuned when trying very large image sizes. Using raw disks means fewer layers that could affect scalability or max size.

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

- ▶ **Local storage pools** - Local storage pools are directly attached to the host physical machine server. Local storage pools include: local directories, directly attached disks, physical partitions, and LVM volume groups. These storage volumes store guest virtual machine images or are attached to guest virtual machines as additional storage. As local storage pools are directly attached to the host physical machine server, they are useful for development, testing and small deployments that do not require migration or large numbers of guest virtual machines. Local storage pools are not suitable for many production environments as local storage pools do not support live migration.
- ▶ **Networked (shared) storage pools** - Networked storage pools include storage devices shared over a network using standard protocols. Networked storage is required when migrating virtual machines between host physical machines with virt-manager, but is optional when migrating with virsh. Networked storage pools are managed by libvirt. Supported protocols for networked storage pools include:
 - Fibre Channel-based LUNs
 - iSCSI
 - NFS
 - GFS2
 - SCSI RDMA protocols (SCSI RCP), the block export protocol used in InfiniBand and 10GbE iWARP adapters.



Note

Multi-path storage pools should not be created or used as they are not fully supported.

11.2. Volumes

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

underlying hardware.

Referencing volumes

To reference a specific volume, three approaches are possible:

The name of the volume and the storage pool

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

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

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

virsh #
```

The full path to the storage on the host physical machine system

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

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

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

The unique volume key

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

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

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

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

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

For example, a volume with the volume key of `Wlvnf7-a4a3-Tlje-1JDa-9eak-PZBv-LoZuUr`:

```
# virsh vol-info Wlvnf7-a4a3-Tlje-1JDa-9eak-PZBv-LoZuUr
Name:      firstimage
Type:      block
Capacity:   20.00 GB
Allocation: 20.00 GB
```

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

vol-name

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

```
# virsh vol-name /dev/guest_images/firstimage  
firstimage  
# virsh vol-name Wlvnf7-a4a3-Tlje-1JDa-9eak-PZBv-LoZuUr
```

vol-path

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

```
# virsh vol-path Wlvnf7-a4a3-Tlje-1JDa-9eak-PZBv-LoZuUr  
/dev/guest_images/firstimage  
# virsh vol-path --pool guest_images firstimage  
/dev/guest_images/firstimage
```

The vol-key command

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

```
# virsh vol-key /dev/guest_images/firstimage  
Wlvnf7-a4a3-Tlje-1JDa-9eak-PZBv-LoZuUr  
# virsh vol-key --pool guest_images firstimage  
Wlvnf7-a4a3-Tlje-1JDa-9eak-PZBv-LoZuUr
```

Chapter 12. Storage pools

This chapter includes instructions on creating storage pools of assorted types. A *storage pool* is a quantity of storage set aside by an administrator, often a dedicated storage administrator, for use by virtual machines. Storage pools are often divided into storage volumes either by the storage administrator or the system administrator, and the volumes are assigned to guest virtual machines as block devices.

Example 12.1. NFS storage pool

Suppose a storage administrator responsible for an NFS server creates a share to store guest virtual machines' data. The system administrator defines a pool on the host physical machine with the details of the share (`nfs.example.com:/path/to/share` should be mounted on `/vm_data`). When the pool is started, libvirt mounts the share on the specified directory, just as if the system administrator logged in and executed `mount nfs.example.com:/path/to/share /vmdatas`. If the pool is configured to autostart, libvirt ensures that the NFS share is mounted on the directory specified when libvirt is started.

Once the pool starts, the files that the NFS share, are reported as volumes, and the storage volumes' paths are then queried using the libvirt APIs. The volumes' paths can then be copied into the section of a guest virtual machine's XML definition file describing the source storage for the guest virtual machine's block devices. With NFS, applications using the libvirt APIs can create and delete volumes in the pool (files within the NFS share) up to the limit of the size of the pool (the maximum storage capacity of the share). Not all pool types support creating and deleting volumes. Stopping the pool negates the start operation, in this case, unmounts the NFS share. The data on the share is not modified by the destroy operation, despite the name. See `man virsh` for more details.

Note

Storage pools and volumes are not required for the proper operation of guest virtual machines. Pools and volumes provide a way for libvirt to ensure that a particular piece of storage will be available for a guest virtual machine, but some administrators will prefer to manage their own storage and guest virtual machines will operate properly without any pools or volumes defined. On systems that do not use pools, system administrators must ensure the availability of the guest virtual machines' storage using whatever tools they prefer, for example, adding the NFS share to the host physical machine's `fstab` so that the share is mounted at boot time.

12.1. Creating storage pools

12.1.1. Disk-based storage pools

This section covers creating disk based storage devices for guest virtual machines.



Warning

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

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

12.1.1.1. Creating a disk based storage pool using `virsh`

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



Warning

Dedicating a disk to a storage pool will reformat and erase all data presently stored on the disk device! It is strongly recommended to back up the storage device before commencing with the following procedure:

1. Create a GPT disk label on the disk

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

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

2. Create the storage pool configuration file

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

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

<name>guest_images_disk</name>

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

<device path='/dev/sdb'>

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

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

The file system **target** parameter with the **path** sub-parameter determines the location on the host physical machine file system to attach volumes created with this storage pool.

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

<format type='gpt'>

The **format** parameter specifies the partition table type. This example uses the **gpt** in the example below, to match the GPT disk label type created in the previous step.

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

Example 12.2. Disk based storage device storage pool

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

3. Attach the device

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

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

4. Start the storage pool

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

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

5. Turn on autostart

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

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

6. Verify the storage pool configuration

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

```
# virsh pool-info guest_images_disk
Name: guest_images_disk
UUID: 551a67c8-5f2a-012c-3844-df29b167431c
State: running
Capacity: 465.76 GB
Allocation: 0.00
Available: 465.76 GB
# ls -la /dev/sdb
brw-rw----. 1 root disk 8, 16 May 30 14:08 /dev/sdb
# virsh vol-list guest_images_disk
Name Path
-----
```

7. Optional: Remove the temporary configuration file

Remove the temporary storage pool XML configuration file if it is not needed.

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

A disk based storage pool is now available.

12.1.1.2. Deleting a storage pool using virsh

The following demonstrates how to delete a storage pool using virsh:

1. To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it.

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

2. Remove the storage pool's definition

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

12.1.2. Partition-based storage pools

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

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

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

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

Procedure 12.1. Creating a partition-based storage pool with virt-manager

1. Open the storage pool settings

- a. In the **virt-manager** graphical interface, select the host physical machine from the main window.

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

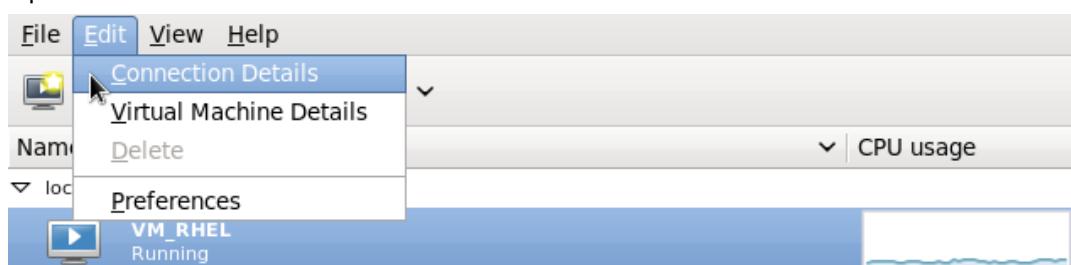
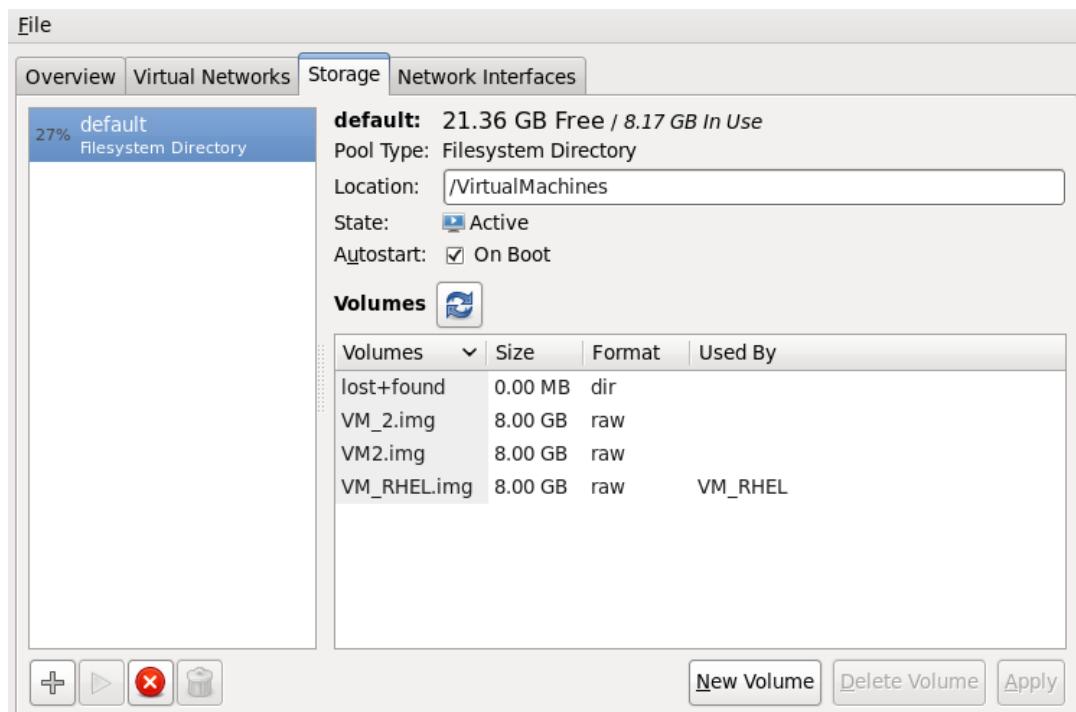


Figure 12.1. Connection Details

- b. Click on the **Storage** tab of the **Connection Details** window.

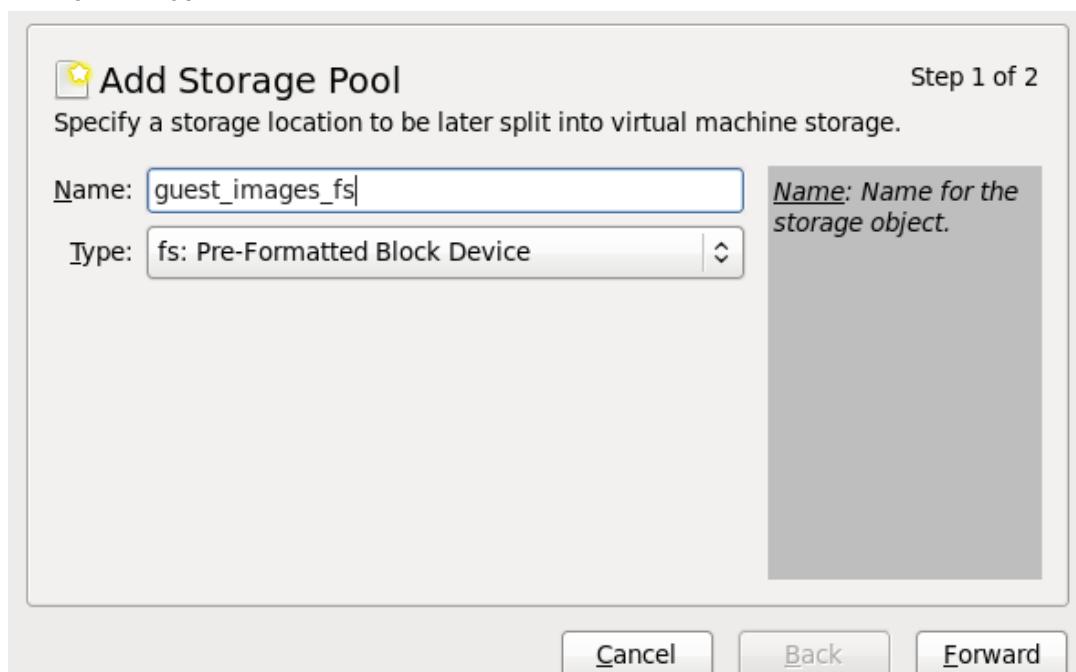
**Figure 12.2. Storage tab**

2. Create the new storage pool

a. Add a new pool (part 1)

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

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

**Figure 12.3. Storage pool name and type**

Press the **Forward** button to continue.

b. Add a new pool (part 2)

Change the **Target Path**, **Format**, and **Source Path** fields.

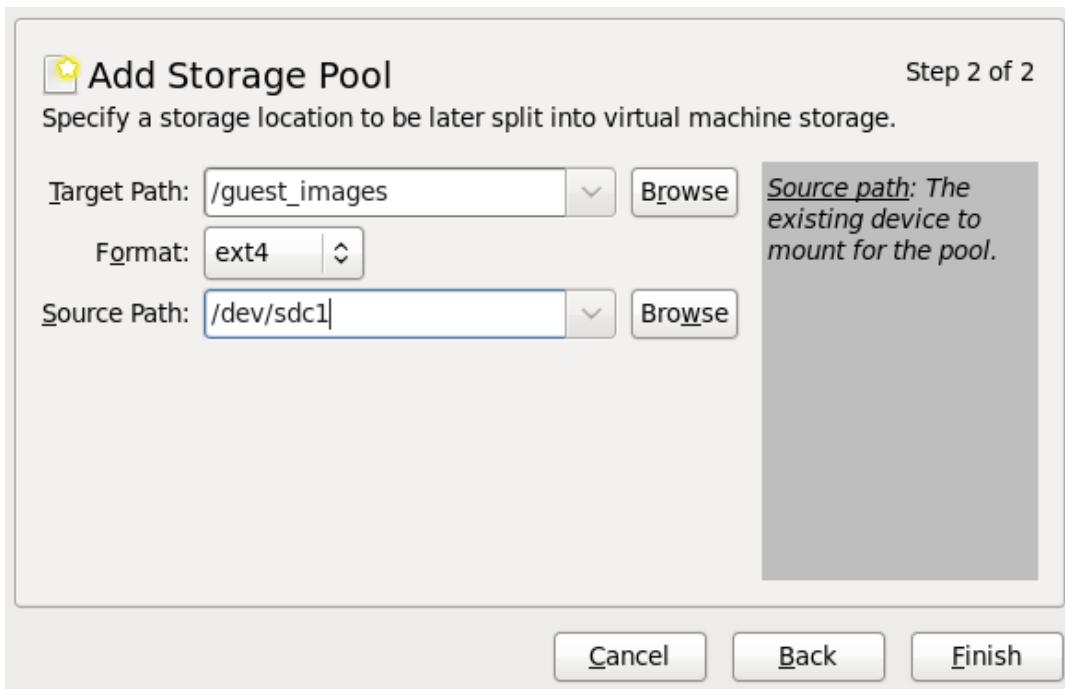


Figure 12.4. Storage pool path and format

Target Path

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

Format

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

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

Source Path

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

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

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

3. Verify the new storage pool

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

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

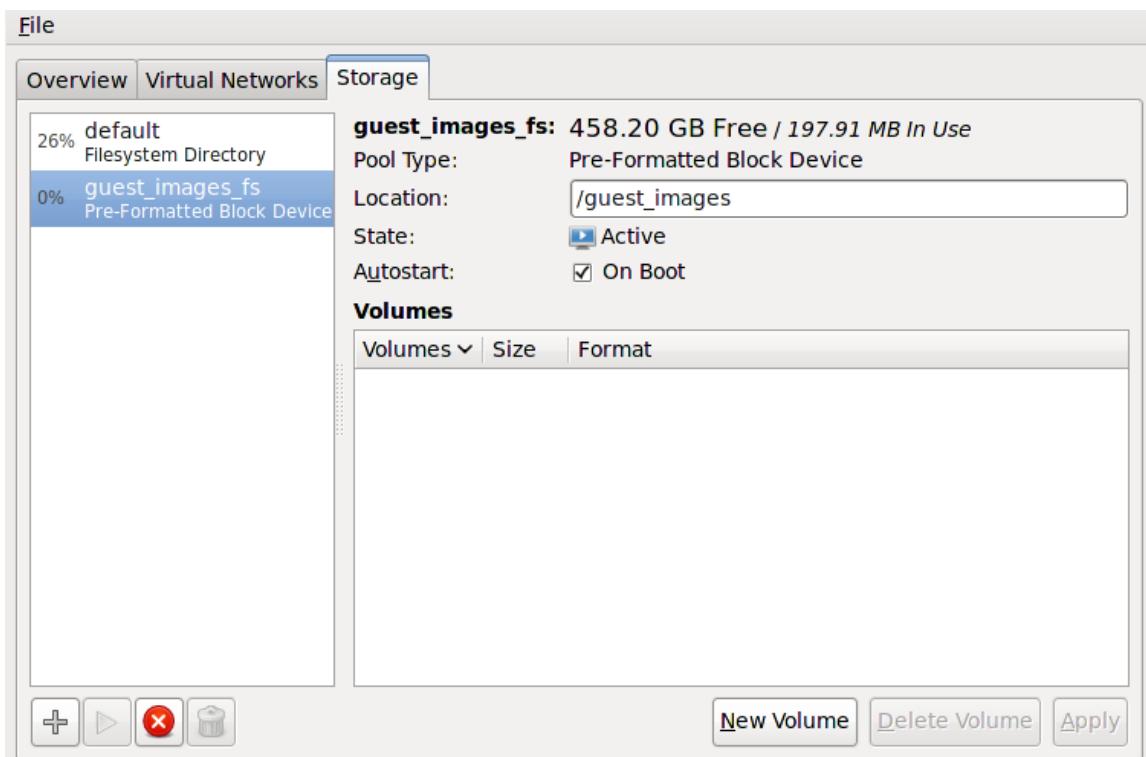


Figure 12.5. Storage list confirmation

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

12.1.2.2. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it. To do this, select the storage pool you want to stop and click the red X icon at the bottom of the Storage window.

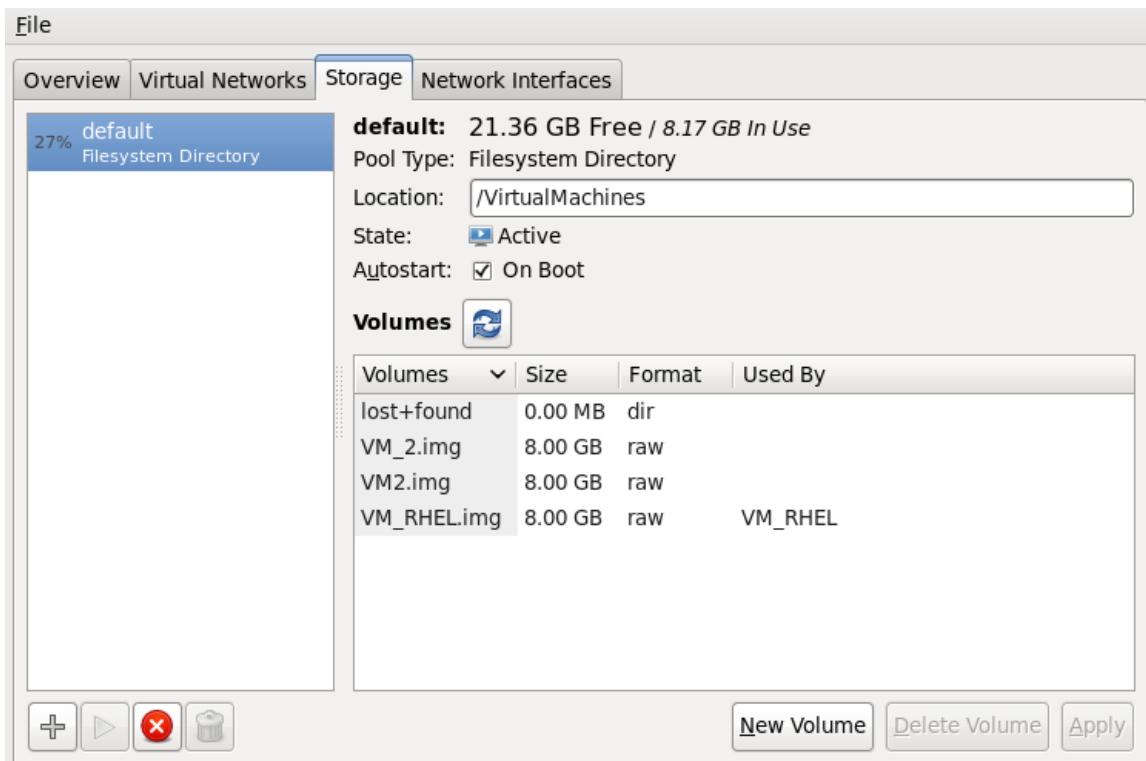


Figure 12.6. Stop Icon

2. Delete the storage pool by clicking the Trash can icon. This icon is only enabled if you stop the storage pool first.

12.1.2.3. Creating a partition-based storage pool using virsh

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



Warning

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

Procedure 12.2. Creating pre-formatted block device storage pools using virsh

1. Create the storage pool definition

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

Partition name

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

device

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

mountpoint

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

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

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

The new pool and mount points are now created.

2. Verify the new pool

List the present storage pools.

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

3. Create the mount point

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

```
# virsh pool-build guest_images_fs
Pool guest_images_fs built
# ls -la /guest_images
total 8
drwx----- 2 root root 4096 May 31 19:38 .
dr-xr-xr-x 25 root root 4096 May 31 19:38 ..
# virsh pool-list --all
Name State Autostart
-----
default active yes
guest_images_fs inactive no
```

4. Start the storage pool

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

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

5. Turn on autostart

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

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

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

6. Verify the storage pool

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

```
# virsh pool-info guest_images_fs
Name: guest_images_fs
UUID: c7466869-e82a-a66c-2187-dc9d6f0877d0
State: running
Persistent: yes
Autostart: yes
Capacity: 458.39 GB
Allocation: 197.91 MB
Available: 458.20 GB
# mount | grep /guest_images
/dev/sdc1 on /guest_images type ext4 (rw)
# ls -la /guest_images
total 24
drwxr-xr-x 3 root root 4096 May 31 19:47 .
dr-xr-xr-x 25 root root 4096 May 31 19:38 ..
drwx----- 2 root root 16384 May 31 14:18 lost+found
```

12.1.2.4. Deleting a storage pool using virsh

- To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it.

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

- Optionally, if you want to remove the directory where the storage pool resides use the following command:

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

- Remove the storage pool's definition

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

12.1.3. Directory-based storage pools

This section covers storing guest virtual machines in a directory on the host physical machine.

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

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

- Create the local directory

- Optional: Create a new directory for the storage pool

Create the directory on the host physical machine for the storage pool. This example uses a directory named */guest_virtual_machine_images*.

```
# mkdir /guest_images
```

- Set directory ownership

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

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

- Set directory permissions

Change the file permissions of the directory.

```
# chmod 700 /guest_images
```

- Verify the changes

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

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

- Configure SELinux file contexts

Configure the correct SELinux context for the new directory. Note that the name of the pool and the directory do not have to match. However, when you shutdown the guest virtual machine, libvirt has to set the context back to a default value. The context of the directory determines what this default value is. It is worth explicitly labeling the directory *virt_image_t*, so that when the guest virtual machine is shutdown, the images get labeled '*virt_image_t*' and are thus isolated from other processes running on the host physical machine.

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

3. Open the storage pool settings

- In the **virt-manager** graphical interface, select the host physical machine from the main window.

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

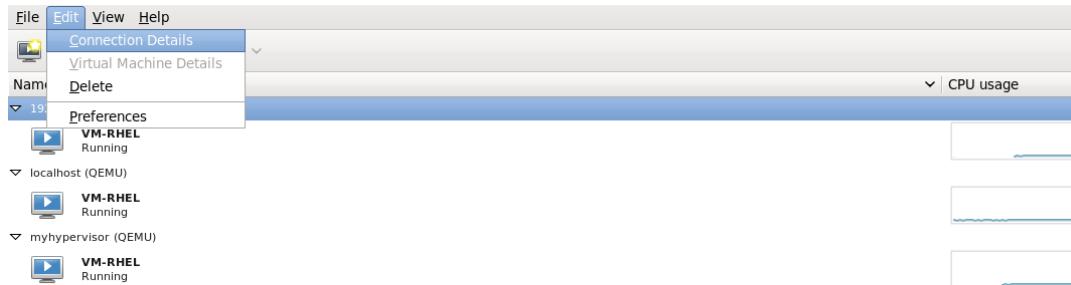


Figure 12.7. Connection details window

- Click on the **Storage** tab of the **Connection Details** window.

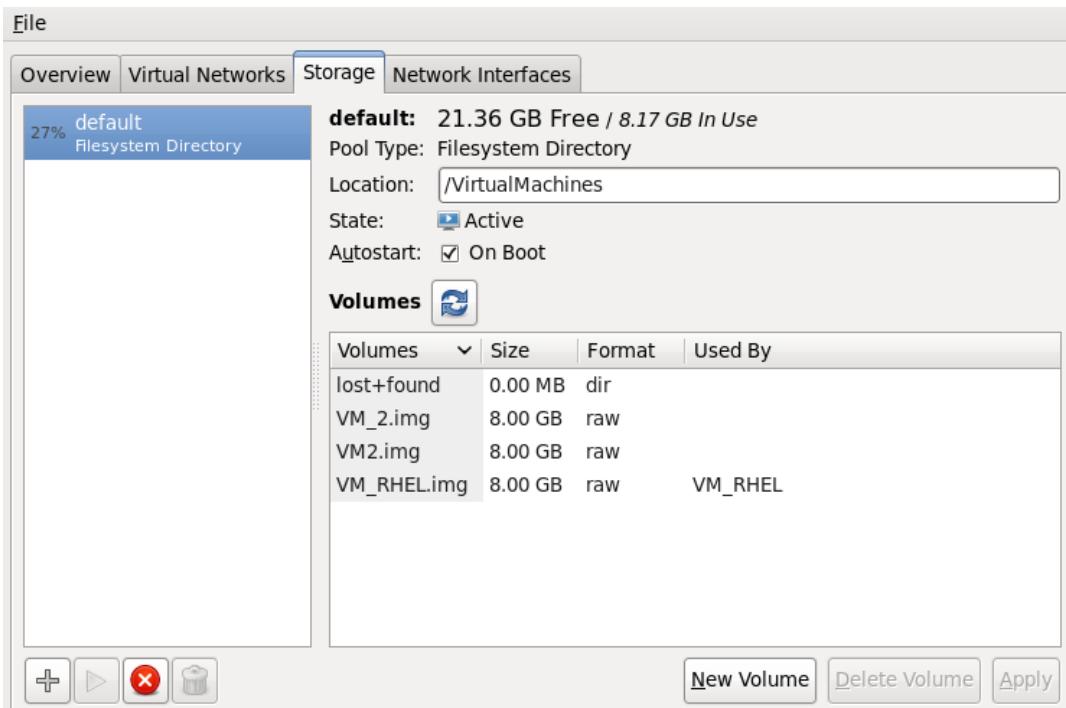


Figure 12.8. Storage tab

4. Create the new storage pool

- Add a new pool (part 1)

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

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

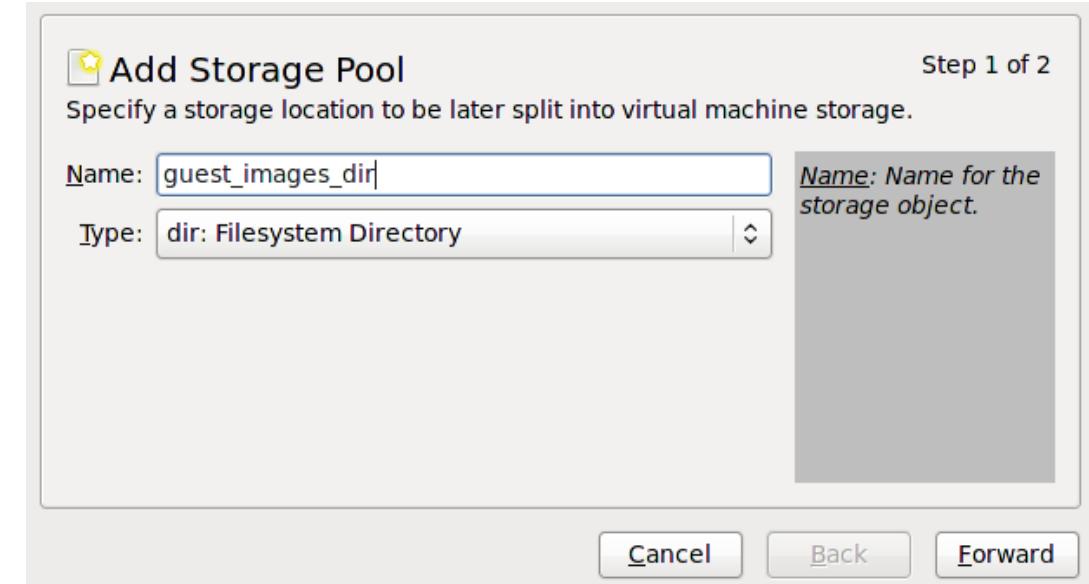


Figure 12.9. Name the storage pool

Press the **Forward** button to continue.

b. **Add a new pool (part 2)**

Change the **Target Path** field. For example, **/guest_images**.

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

5. **Verify the new storage pool**

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

Select the storage pool. In the **Autostart** field, confirm that the **On Boot** checkbox is checked. This will make sure the storage pool starts whenever the **libvirtd** service starts.

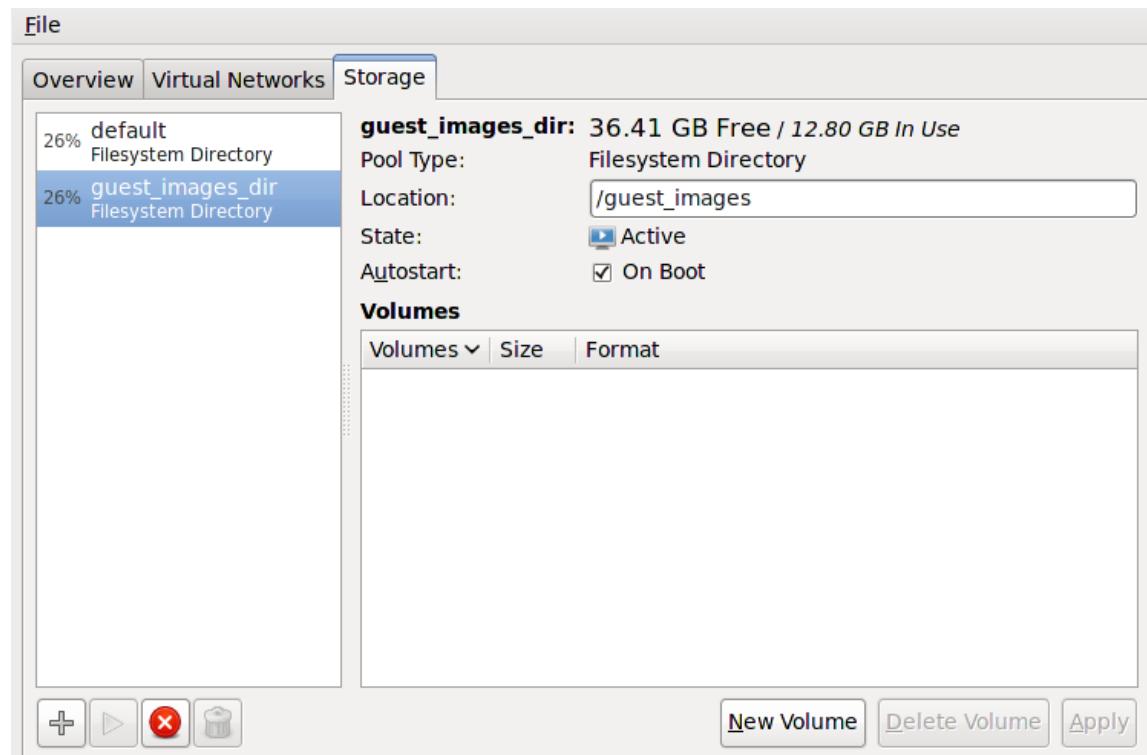


Figure 12.10. Verify the storage pool information

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

12.1.3.2. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it. To do this, select the storage pool you want to stop and click the red X icon at the bottom of the Storage window.

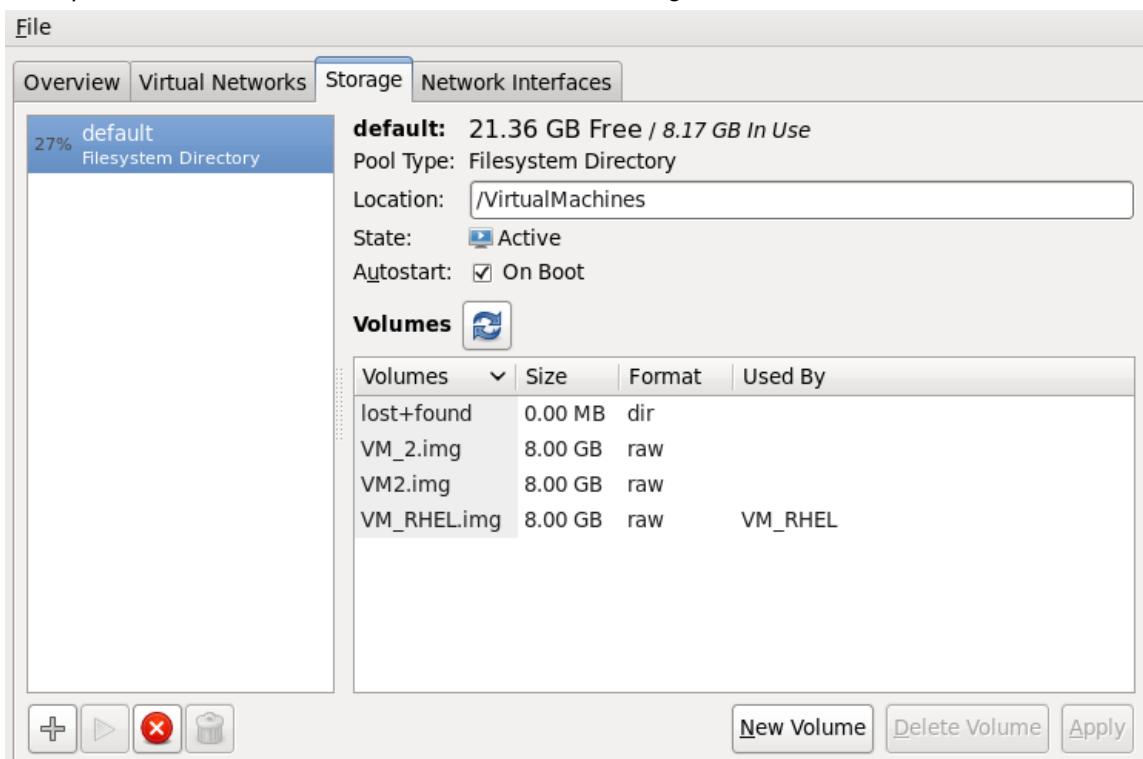


Figure 12.11. Stop icon

2. Delete the storage pool by clicking the Trash can icon. This icon is only enabled if you stop the storage pool first.

12.1.3.3. Creating a directory-based storage pool with virsh

1. Create the storage pool definition

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

- » The **name** of the storage pool.

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

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

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

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

2. Verify the storage pool is listed

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

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

3. Create the local directory

Use the **virsh pool-build** command to build the directory-based storage pool for the directory **guest_images** (for example), as shown:

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

4. Start the storage pool

Use the **virsh** command **pool-start** to enable a directory storage pool, thereby allowing volumes of the pool to be used as guest disk images.

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

5. Turn on autostart

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

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

6. Verify the storage pool configuration

Verify the storage pool was created correctly, the size is reported correctly, and the state is reported as **running**. If you want the pool to be accessible even if the guest virtual machine is not running, make sure that **Persistent** is reported as **yes**. If you want the pool to start automatically when the service starts, make sure that **Autostart** is reported as **yes**.

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

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

A directory-based storage pool is now available.

12.1.3.4. Deleting a storage pool using virsh

The following demonstrates how to delete a storage pool using virsh:

1. To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it.

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

2. Optionally, if you want to remove the directory where the storage pool resides use the following command:

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

3. Remove the storage pool's definition

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

12.1.4. LVM-based storage pools

This chapter covers using LVM volume groups as storage pools.

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



Note

Thin provisioning is currently not possible with LVM based storage pools.



Note

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



Warning

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

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

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

1. Optional: Create new partition for LVM volumes

These steps describe how to create a new partition and LVM volume group on a new hard disk drive.



Warning

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

a. Create a new partition

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

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

Press **n** for a new partition.

b. Press **p** for a primary partition.

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

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

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

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

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

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

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

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

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

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

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

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

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

- i. write changes to disk and quit.

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

j. **Create a new LVM volume group**

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

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

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

2. **Open the storage pool settings**

- a. In the **virt-manager** graphical interface, select the host from the main window.

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

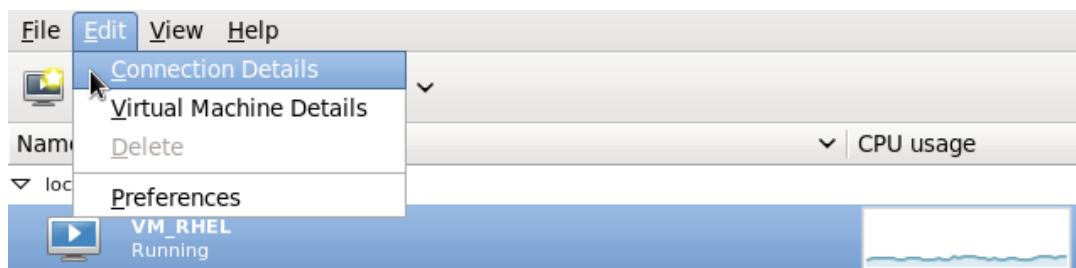


Figure 12.12. Connection details

- b. Click on the **Storage** tab.

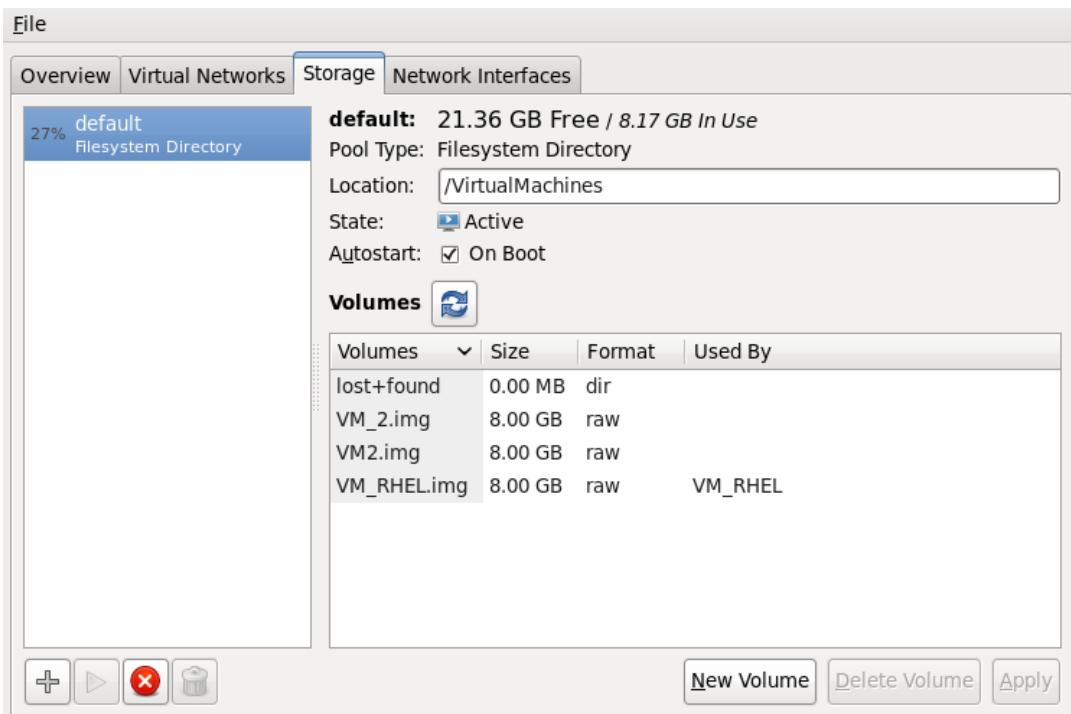


Figure 12.13. Storage tab

3. Create the new storage pool

a. Start the Wizard

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

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

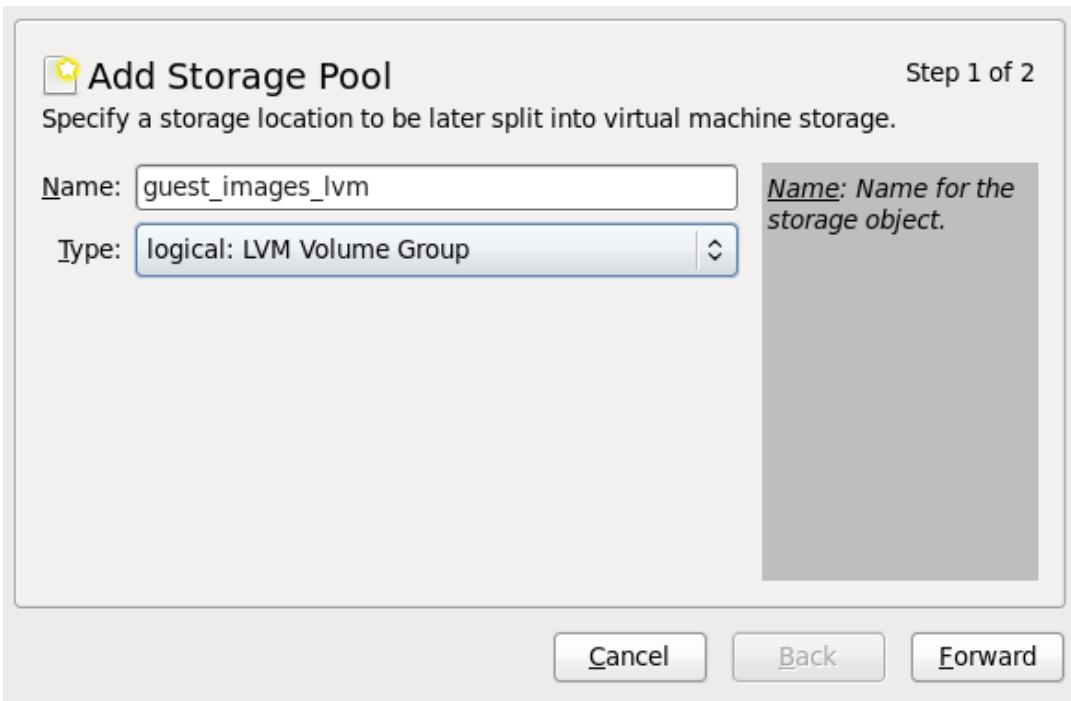


Figure 12.14. Add LVM storage pool

Press the **Forward** button to continue.

b. Add a new pool (part 2)

Change the **Target Path** field. This example uses `/guest_images`.

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

- » Use the **Target Path** field to either select an existing LVM volume group or as the name for a new volume group. The default format is `/dev/storage_pool_name`. This example uses a new volume group named `/dev/guest_images_lvm`.
- » The **Source Path** field is optional if an existing LVM volume group is used in the **Target Path**. For new LVM volume groups, input the location of a storage device in the **Source Path** field. This example uses a blank partition `/dev/sdc`.
- » The **Build Pool** checkbox instructs **virt-manager** to create a new LVM volume group. If you are using an existing volume group you should not select the **Build Pool** checkbox. This example is using a blank partition to create a new volume group so the **Build Pool** checkbox must be selected.

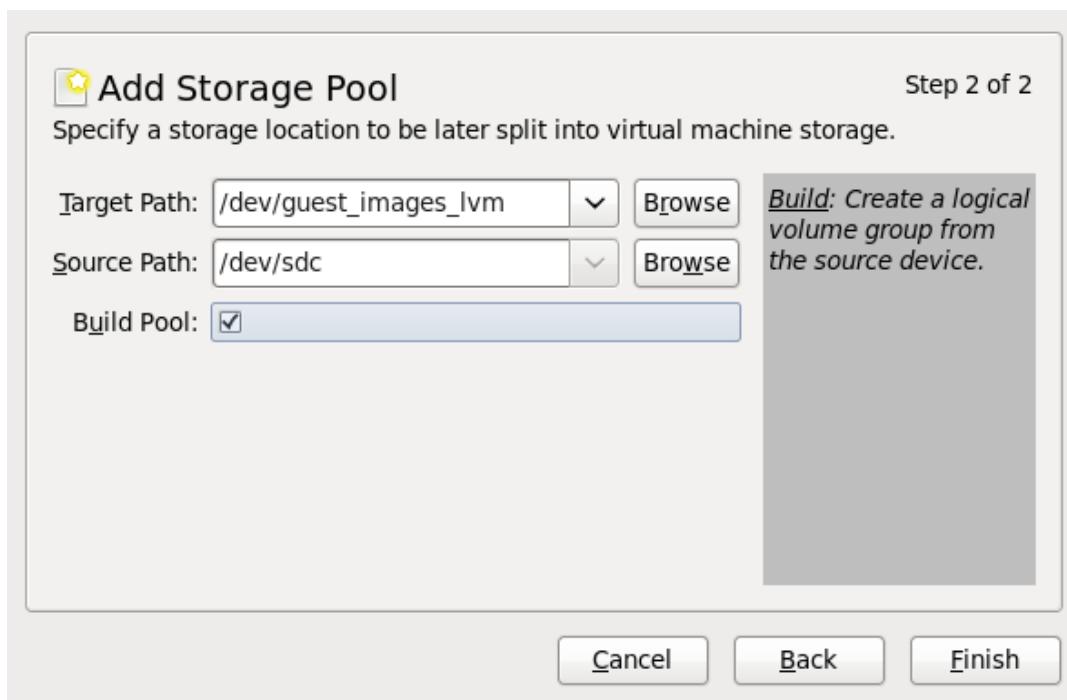


Figure 12.15. Add target and source

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

c. Confirm the device to be formatted

A warning message appears.



Figure 12.16. Warning message

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

4. Verify the new storage pool

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

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

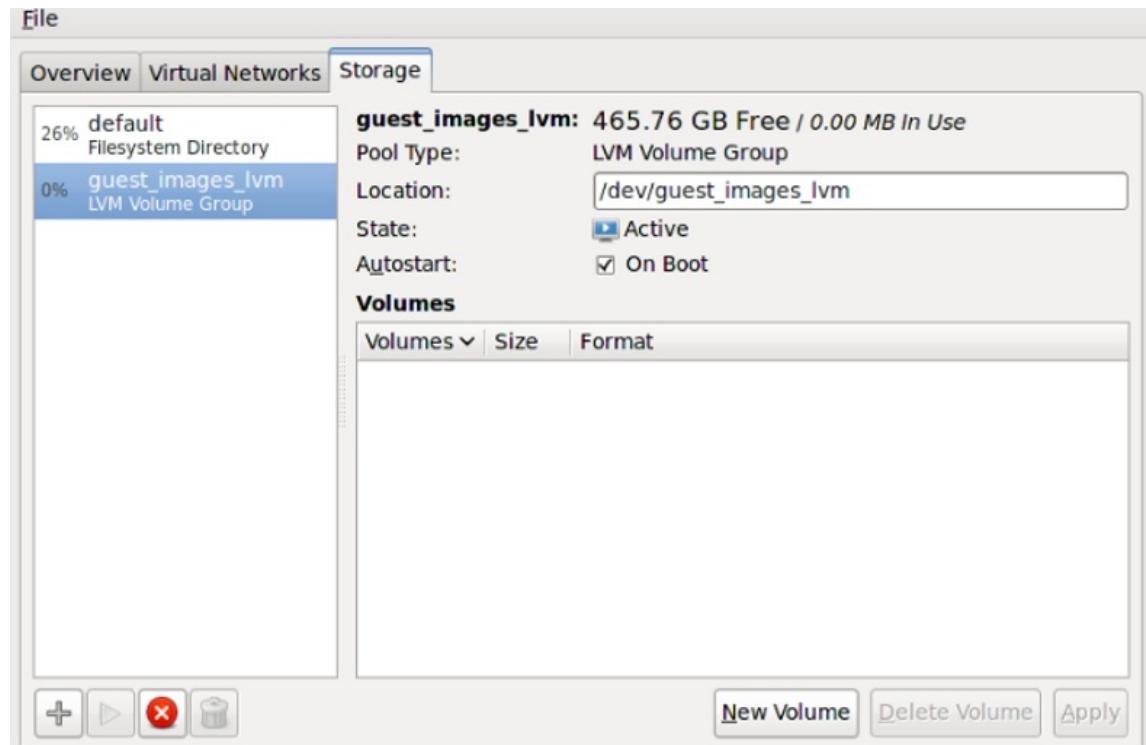


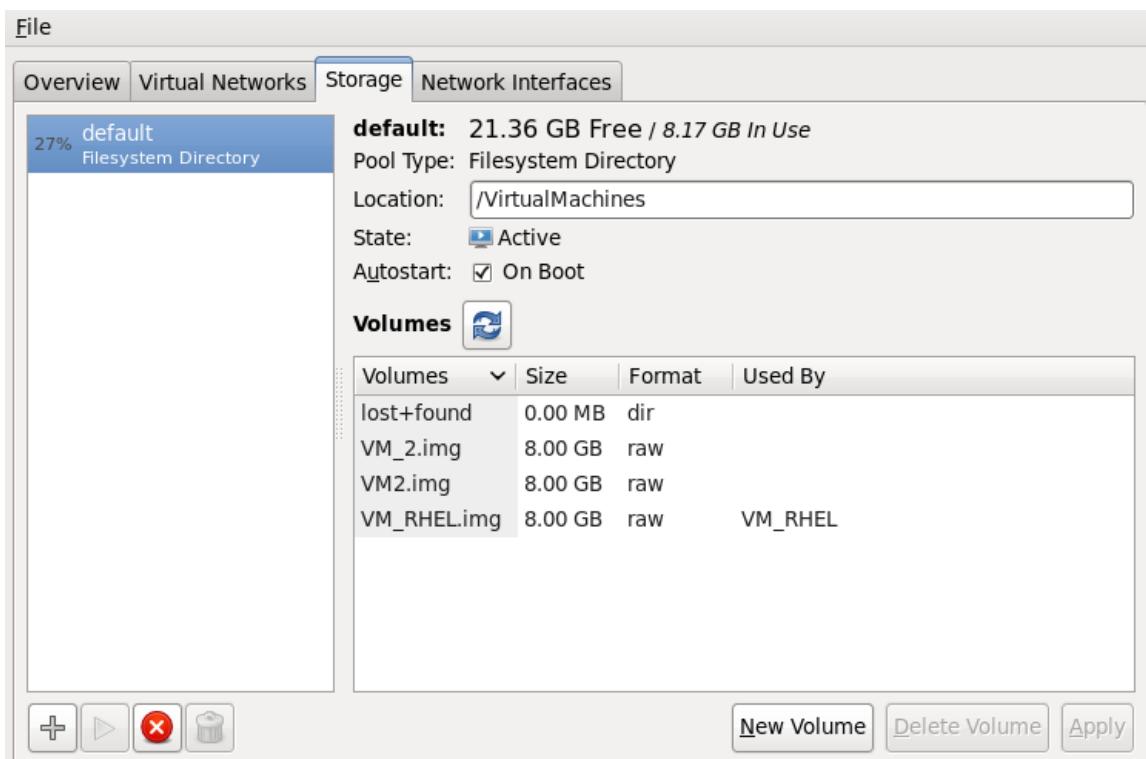
Figure 12.17. Confirm LVM storage pool details

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

12.1.4.2. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it. To do this, select the storage pool you want to stop and click the red X icon at the bottom of the Storage window.

**Figure 12.18. Stop Icon**

2. Delete the storage pool by clicking the Trash can icon. This icon is only enabled if you stop the storage pool first.

12.1.4.3. Creating an LVM-based storage pool with virsh

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

Procedure 12.3. Creating an LVM-based storage pool with virsh

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

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

2. Build the pool according to the specified name. If you are using an already existing volume group, skip this step.

```
# virsh pool-build guest_images_lvm
Pool guest_images_lvm built
```

3. Initialize the new pool.

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

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

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

5. Set the pool to start automatically.

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

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

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

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

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

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

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

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

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

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

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

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

12.1.4.4. Deleting a storage pool using virsh

The following demonstrates how to delete a storage pool using virsh:

- To avoid any issues with other guests using the same pool, it is best to stop the storage pool and release any resources in use by it.

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

2. Optionally, if you want to remove the directory where the storage pool resides use the following command:

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

3. Remove the storage pool's definition

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

12.1.5. iSCSI-based storage pools

This section covers using iSCSI-based devices to store guest virtual machines.

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

12.1.5.1. Configuring a software iSCSI target

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

Procedure 12.4. Creating an iSCSI target

1. **Install the required packages**

Install the *scsi-target-utils* package and all dependencies

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

2. **Start the tgtd service**

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

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

3. **Optional: Create LVM volumes**

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

- a. **Create the RAID array**

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

- b. **Create the LVM volume group**

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

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

- c. **Create a LVM logical volume**

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

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

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

4. **Optional: Create file-based images**

File-based storage is sufficient for testing but is not recommended for production environments or

any significant I/O activity. This optional procedure creates a file based imaged named ***virtimage2.img*** for an iSCSI target.

a. Create a new directory for the image

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

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

b. Create the image file

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

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

c. Configure SELinux file contexts

Configure the correct SELinux context for the new image and directory.

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

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

5. Create targets

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

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

Where:

- ▶ **yyyy-mm** represents the year and month the device was started (for example: **2010-05**);
- ▶ **reversed domain name** is the host physical machines domain name in reverse (for example **server1.example.com** in an IQN would be **com.example.server1**); and
- ▶ **optional identifier text** is any text string, without spaces, that assists the administrator in identifying devices or hardware.

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

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

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



Important

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

6. Restart the tgtd service

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

```
# service tgtd restart
```

7. iptables configuration

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

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

8. Verify the new targets

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

```
# tgt-admin --show
Target 1: iqn.2010-05.com.example.server1:iscsirhel6guest
System information:
Driver: iscsi
State: ready
I_T nexus information:
LUN information:
LUN: 0
Type: controller
SCSI ID: IET      00010000
SCSI SN: beaf10
Size: 0 MB
Online: Yes
Removable media: No
Backing store type: rdwr
Backing store path: None
LUN: 1
Type: disk
SCSI ID: IET      00010001
SCSI SN: beaf11
Size: 20000 MB
Online: Yes
Removable media: No
Backing store type: rdwr
Backing store path: /dev/virtstore/virtimage1
LUN: 2
Type: disk
SCSI ID: IET      00010002
SCSI SN: beaf12
Size: 10000 MB
Online: Yes
Removable media: No
Backing store type: rdwr
Backing store path: /var/lib/tgtd/virtualization/virtimage2.img
Account information:
ACL information:
ALL
```



Warning

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

9. Optional: Test discovery

Test whether the new iSCSI device is discoverable.

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

10. Optional: Test attaching the device

Attach the new device (`iqn.2010-05.com.example.server1:iscsirhel6guest`) to determine whether the device can be attached.

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

Logging in to [iface: default, target: iqn.2010-
05.com.example.server1:iscsirhel6guest, portal: 10.0.0.1,3260]
Login to [iface: default, target: iqn.2010-
05.com.example.server1:iscsirhel6guest, portal: 10.0.0.1,3260] successful.
```

Detach the device.

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

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

An iSCSI device is now ready to use for virtualization.

12.1.5.2. Adding an iSCSI target to virt-manager

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

Procedure 12.5. Adding an iSCSI device to virt-manager

1. Open the host physical machine's storage tab

Open the **Storage** tab in the **Host Details** window.

- a. Open **virt-manager**.
- b. Select a host physical machine from the main **virt-manager** window. Click **Edit** menu and select **Connection Details**.

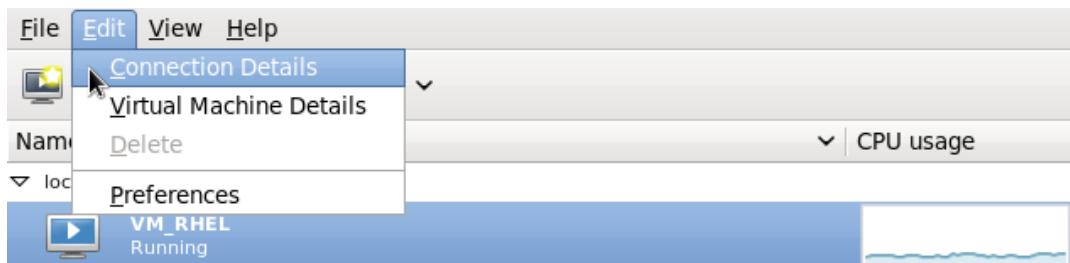
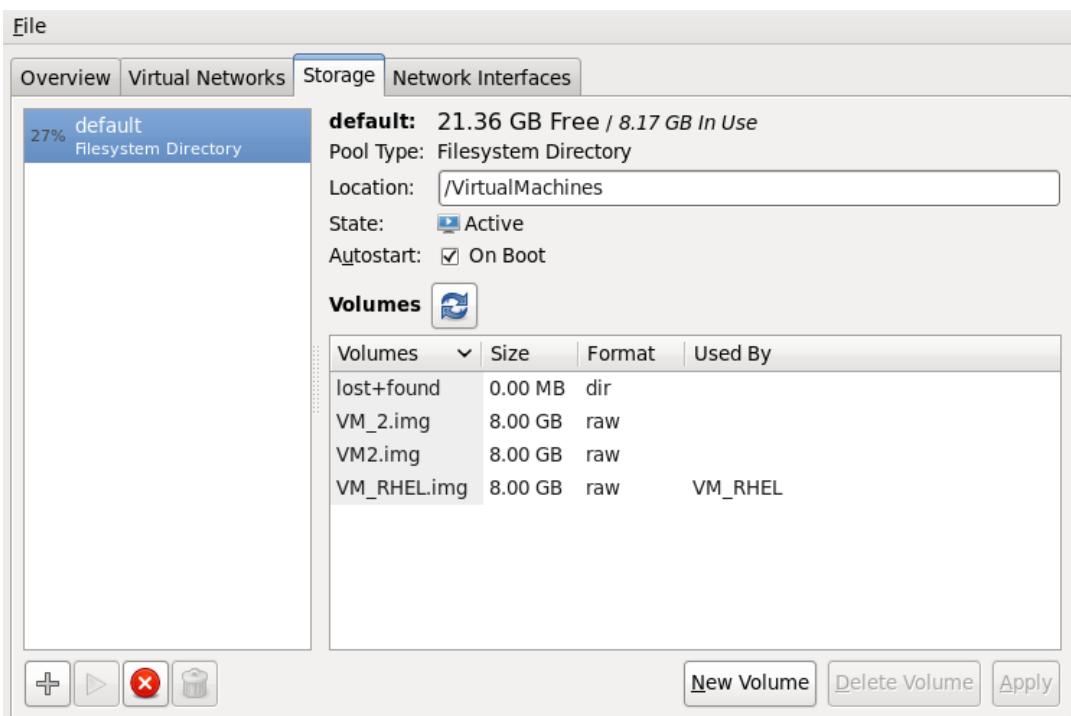


Figure 12.19. Connection details

- c. Click on the **Storage** tab.

**Figure 12.20. Storage menu****2. Add a new pool (part 1)**

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

Add Storage Pool Step 1 of 2

Specify a storage location to be later split into virtual machine storage.

Name:

Type: Type: Storage device type the pool will represent.

Cancel **Back** **Forward**

Figure 12.21. Add an iscsi storage pool name and type

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

3. Add a new pool (part 2)

You will need the information you used in [Section 12.1.5, “iSCSI-based storage pools”](#) and [Step 5](#) to complete the fields in this menu.

- Enter the iSCSI source and target. The **Format** option is not available as formatting is

handled by the guest virtual machines. It is not advised to edit the **Target Path**. The default target path value, **/dev/disk/by-path/**, adds the drive path to that directory. The target path should be the same on all host physical machines for migration.

- b. Enter the hostname or IP address of the iSCSI target. This example uses **host1.example.com**.
- c. In the **Source Path** field, enter the iSCSI target IQN. If you look at [Step 5](#) in [Section 12.1.5, "iSCSI-based storage pools"](#), this is the information you added in the **/etc/tgt/targets.conf** file. This example uses **iqn.2010-05.com.example.server1:iscsirhel6guest**.
- d. Check the **IQN** checkbox to enter the IQN for the initiator. This example uses **iqn.2010-05.com.example.host1:iscsirh6**.
- e. Click **Finish** to create the new storage pool.

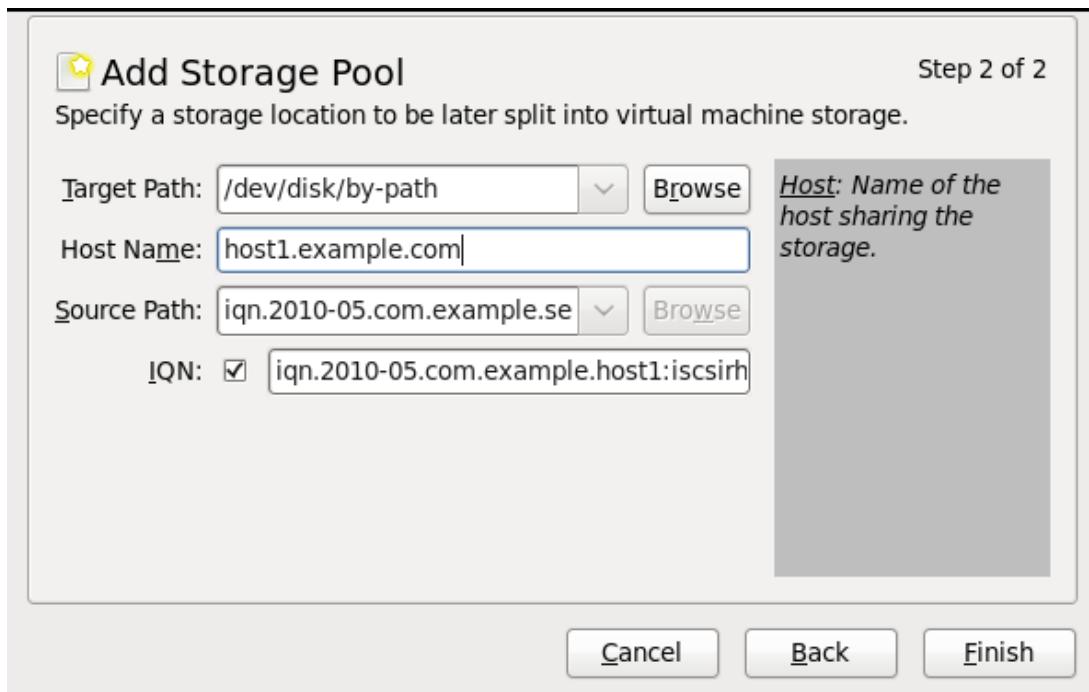
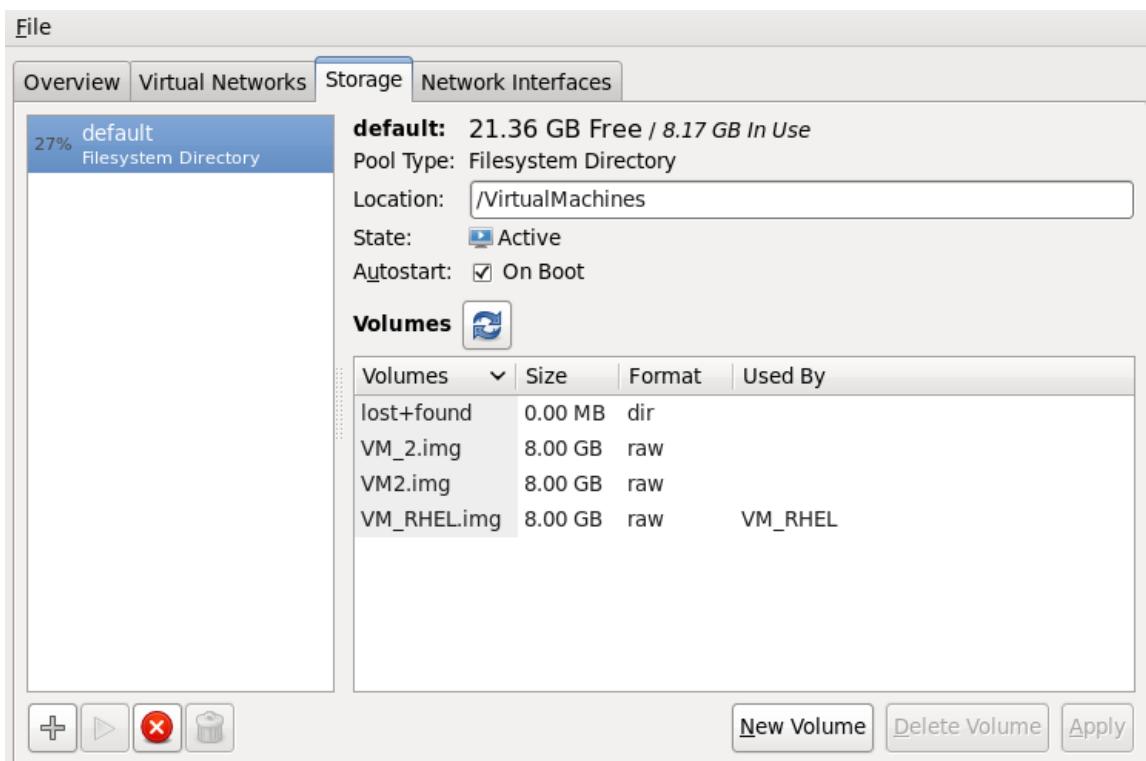


Figure 12.22. Create an iscsi storage pool

12.1.5.3. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it. To do this, select the storage pool you want to stop and click the red X icon at the bottom of the Storage window.

**Figure 12.23. Stop Icon**

2. Delete the storage pool by clicking the Trash can icon. This icon is only enabled if you stop the storage pool first.

12.1.5.4. Creating an iSCSI-based storage pool with virsh

1. Use pool-define-as to define the pool from the command line

Storage pool definitions can be created with the **virsh** command line tool. Creating storage pools with **virsh** is useful for systems administrators using scripts to create multiple storage pools.

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

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

The parameters are explained as follows:

type

defines this pool as a particular type, `iscsi` for example

name

must be unique and sets the name for the storage pool

source-host and source-path

the hostname and iSCSI IQN respectively

source-dev and source-name

these parameters are not required for iSCSI-based pools, use a `-` character to leave the field blank.

target

defines the location for mounting the iSCSI device on the host physical machine

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

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

2. Verify the storage pool is listed

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

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

3. Start the storage pool

Use the virsh command **pool-start** for this. **pool-start** enables a directory storage pool, allowing it to be used for volumes and guest virtual machines.

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

4. Turn on autostart

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

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

Verify that the **iscsirhel6guest** pool has autostart set:

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

5. Verify the storage pool configuration

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

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

An iSCSI-based storage pool is now available.

12.1.5.5. Deleting a storage pool using virsh

The following demonstrates how to delete a storage pool using virsh:

1. To avoid any issues with other guest virtual machines using the same pool, it is best to stop the storage pool and release any resources in use by it.

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

2. Remove the storage pool's definition

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

12.1.6. NFS-based storage pools

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

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

1. Open the host physical machine's storage tab

Open the **Storage** tab in the **Host Details** window.

- a. Open **virt-manager**.
- b. Select a host physical machine from the main **virt-manager** window. Click **Edit** menu and select **Connection Details**.

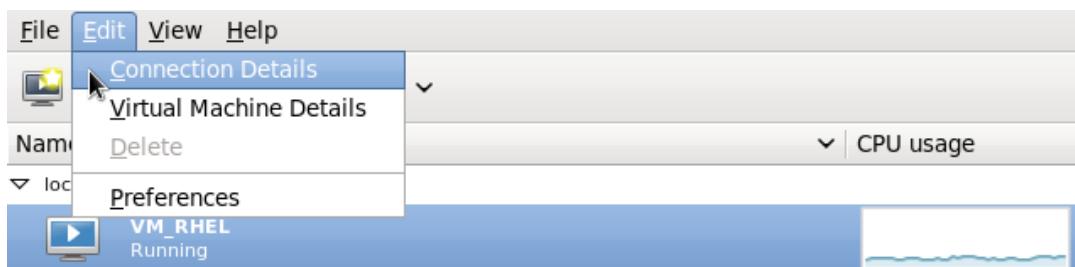


Figure 12.24. Connection details

- c. Click on the Storage tab.

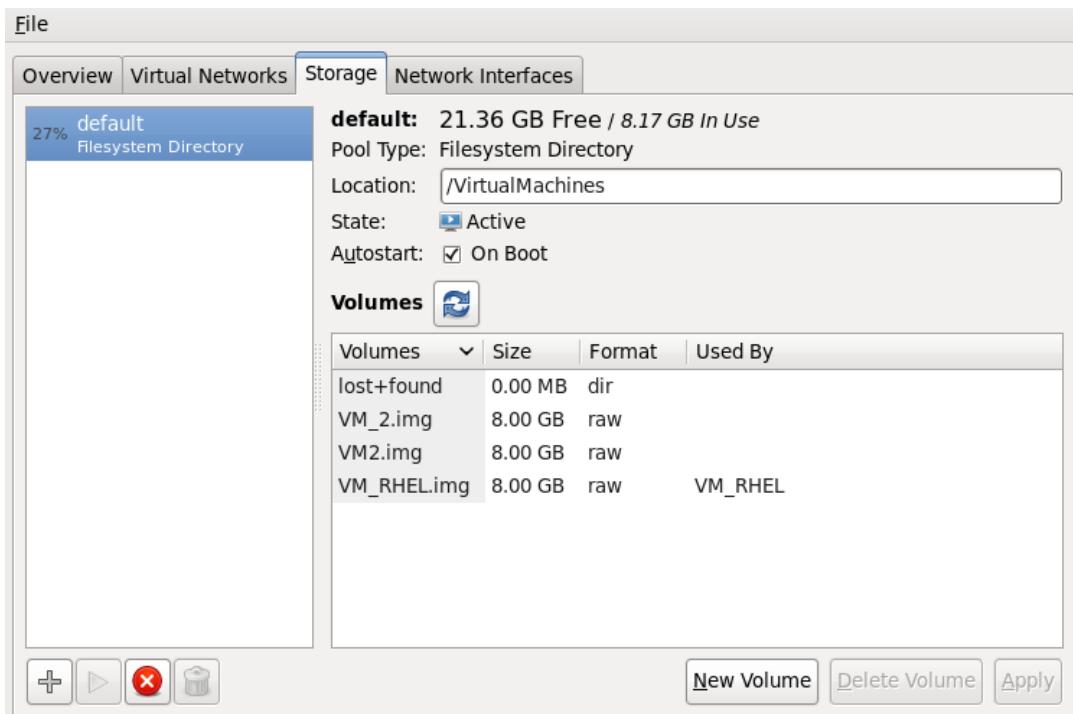


Figure 12.25. Storage tab

2. Create a new pool (part 1)

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

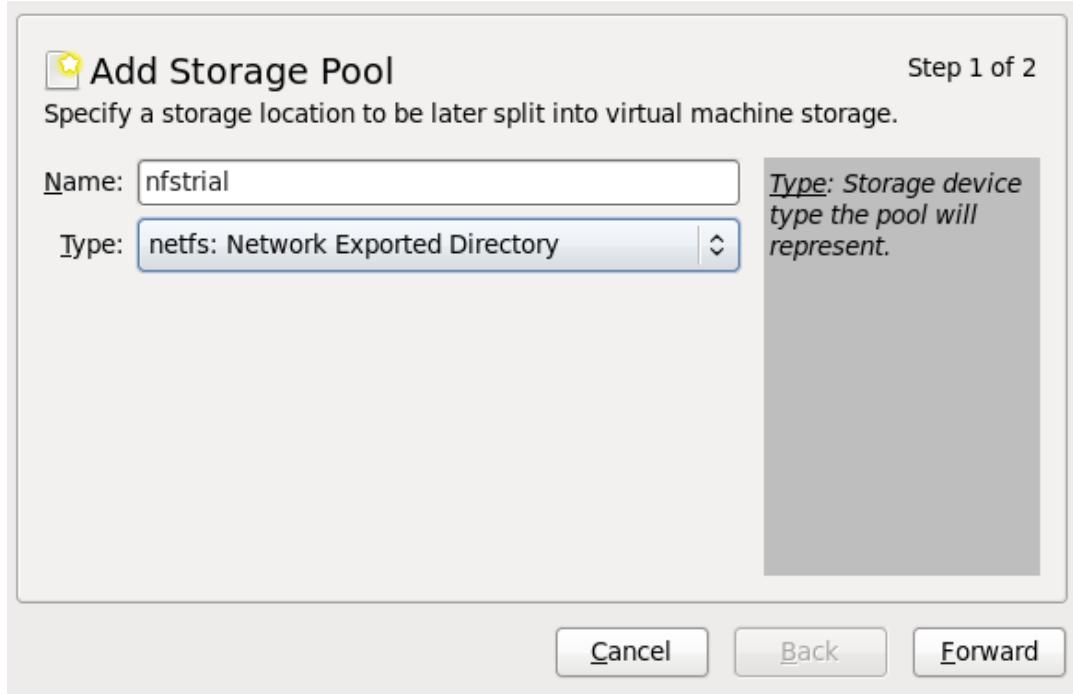


Figure 12.26. Add an NFS name and type

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

3. Create a new pool (part 2)

Enter the target path for the device, the hostname and the NFS share path. Set the **Format** option to **NFS** or **auto** (to detect the type). The target path must be identical on all host physical machines for migration.

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

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

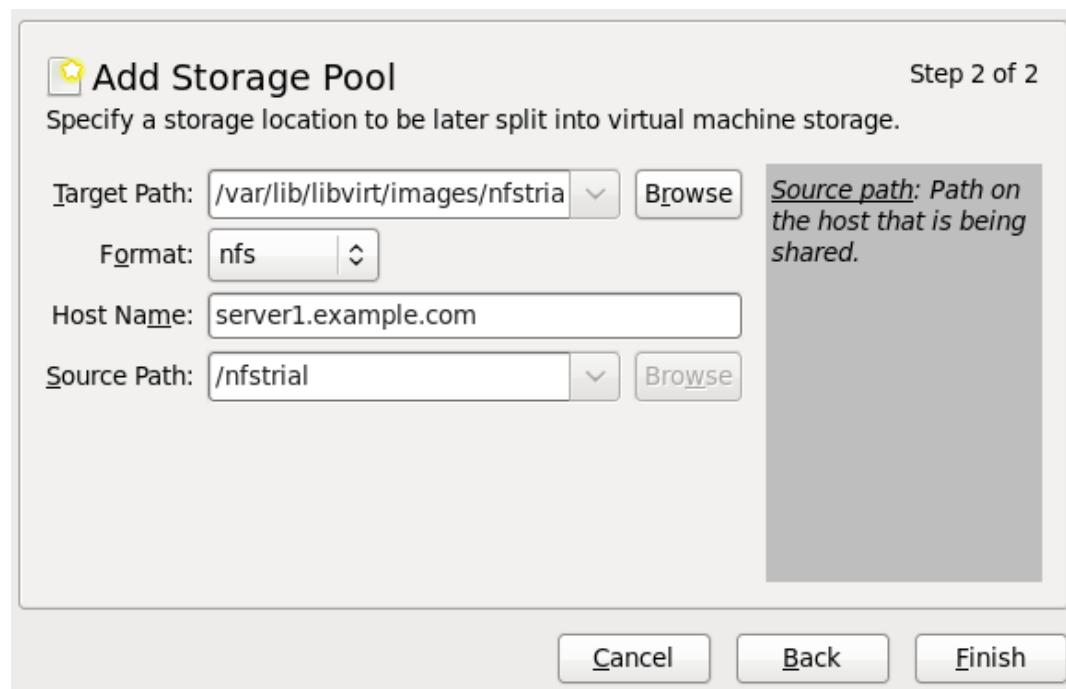


Figure 12.27. Create an NFS storage pool

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

12.1.6.2. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other guests using the same pool, it is best to stop the storage pool and release any resources in use by it. To do this, select the storage pool you want to stop and click the red X icon at the bottom of the Storage window.

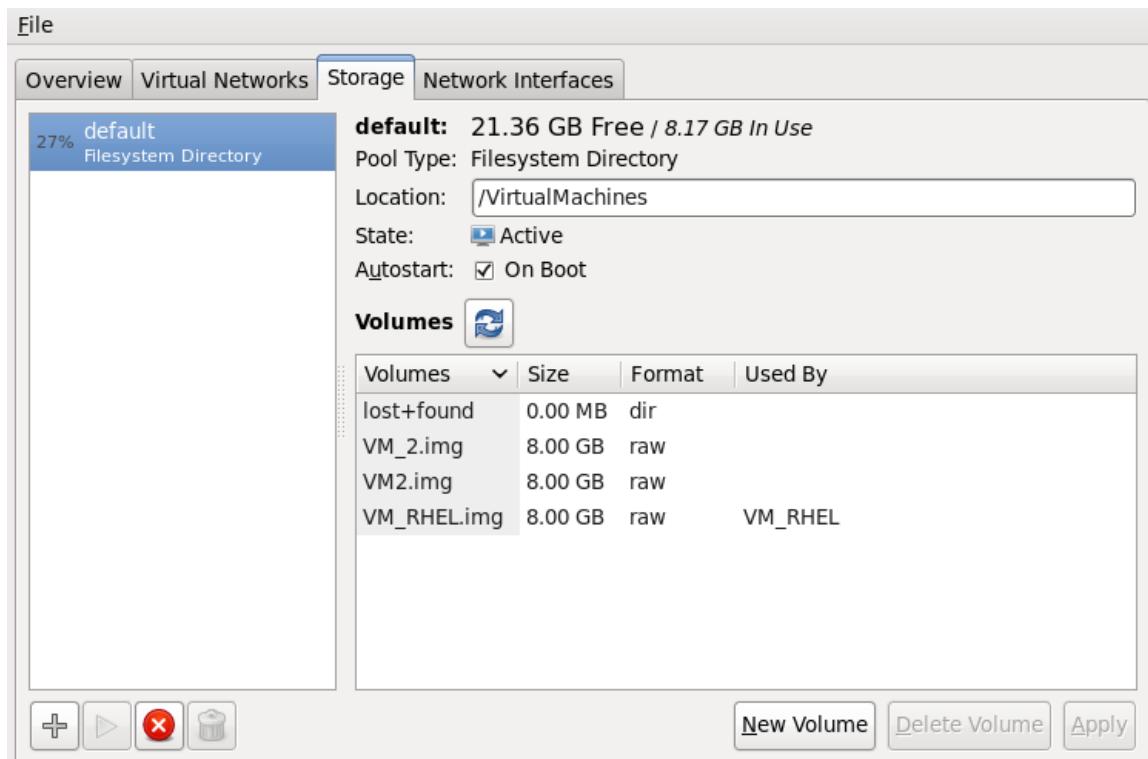


Figure 12.28. Stop Icon

2. Delete the storage pool by clicking the Trash can icon. This icon is only enabled if you stop the storage pool first.

Chapter 13. Volumes

13.1. Creating volumes

This section shows how to create disk volumes inside a block based storage pool. In the example below, the **virsh vol-create-as** command will create a storage volume with a specific size in GB within the **guest_images_disk** storage pool. As this command is repeated per volume needed, three volumes are created as shown in the example.

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

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

# virsh vol-create-as guest_images_disk volume3 8G
Vol volume3 created

# virsh vol-list guest_images_disk
Name Path
-----
volume1 /dev/sdb1
volume2 /dev/sdb2
volume3 /dev/sdb3

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

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

13.2. Cloning volumes

The new volume will be allocated from storage in the same storage pool as the volume being cloned. The **virsh vol-clone** must have the **--pool** argument which dictates the name of the storage pool that contains the volume to be cloned. The rest of the command names the volume to be cloned (volume3) and the name of the new volume that was cloned (clone1). The **virsh vol-list** command lists the volumes that are present in the storage pool (guest_images_disk).

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

# virsh vol-list guest_images_disk
Name          Path
-----
volume1       /dev/sdb1
volume2       /dev/sdb2
volume3       /dev/sdb3
clone1        /dev/sdb4

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

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

13.3. Adding storage devices to guests

This section covers adding storage devices to a guest. Additional storage can only be added as needed.

13.3.1. Adding file based storage to a guest

File-based storage is a collection of files that are stored on the host physical machines file system that act as virtualized hard drives for guests. To add file-based storage, perform the following steps:

Procedure 13.1. Adding file-based storage

1. Create a storage file or use an existing file (such as an **IMG** file). Note that both of the following commands create a 4GB file which can be used as additional storage for a guest:
 - ▶ Pre-allocated files are recommended for file-based storage images. Create a pre-allocated file using the following **dd** command as shown:

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

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

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

2. Create the additional storage by writing a **<disk>** element in a new file. In this example, this file will be known as **NewStorage.xml**.

A **<disk>** element describes the source of the disk, and a device name for the virtual block device. The device name should be unique across all devices in the guest, and identifies the bus on which the guest will find the virtual block device. The following example defines a virtio block device whose source is a file-based storage container named **FileName.img**:

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

Device names can also start with "hd" or "sd", identifying respectively an IDE and a SCSI disk. The configuration file can also contain an **<address>** sub-element that specifies the position on the bus for the new device. In the case of virtio block devices, this should be a PCI address. Omitting the **<address>** sub-element lets libvirt locate and assign the next available PCI slot.

3. Attach the CD-ROM as follows:

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

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

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

Note

This change will only apply after the guest has been destroyed and restarted. In addition, persistent devices can only be added to a persistent domain, that is a domain whose configuration has been saved with **virsh define** command.

If the guest is running, and you want the new device to be added temporarily until the guest is destroyed, omit the **--config** option:

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

Note

The **virsh** command allows for an **attach-disk** command that can set a limited number of parameters with a simpler syntax and without the need to create an XML file. The **attach-disk** command is used in a similar manner to the **attach-device** command mentioned previously, as shown:

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

Note that the **virsh attach-disk** command also accepts the **--config** option.

5. Start the guest machine (if it is currently not running):

```
# virsh start Guest1
```



Note

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

6. Partitioning the disk drive

The guest now has a hard disk device called **/dev/vdb**. If required, partition this disk drive and format the partitions. If you do not see the device that you added, then it indicates that there is an issue with the disk hotplug in your guest's operating system.

- Start **fdisk** for the new device:

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

- Type **n** for a new partition.
- The following appears:

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

Type **p** for a primary partition.

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

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

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

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

- 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):
```

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

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

- Select the partition you created in the previous steps. In this example, the partition number is **1** as there was only one partition created and fdisk automatically selected partition 1.

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

- Enter **83** for a Linux partition.

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

- Enter **w** to write changes and quit.

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

- Format the new partition with the **ext3** file system.

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

7. Create a mount directory, and mount the disk on the guest. In this example, the directory is located in ***myfiles***.

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

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

/dev/vdb1	/myfiles	ext3	defaults	0 0
-----------	-----------------	------	----------	-----

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

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

Procedure 13.2. Adding physical block devices to guests

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

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

2. Do one of the following:

- a. Create the additional storage by writing a **disk** element in a new file. In this example, this file will be known as **NewStorage.xml**. The following example is a configuration file section which contains an additional device-based storage container for the host physical machine partition **/dev/sr0**:

```
<disk type='block' device='disk'>
    <driver name='qemu' type='raw' cache='none' />
    <source dev='/dev/sr0' />
    <target dev='vdc' bus='virtio' />
</disk>
```

- b. Follow the instruction in the previous section to attach the device to the guest virtual machine. Alternatively, you can use the **virsh attach-disk** command, as shown:

```
# virsh attach-disk Guest1 /dev/sr0 vdc
```

Note that the following options are available:

- » The **virsh attach-disk** command also accepts the **--config**, **--type**, and **--mode** options, as shown:

```
# virsh attach-disk Guest1 /dev/sr0 vdc --config --type cdrom --
mode readonly
```

- » Additionally, **--type** also accepts **--type disk** in cases where the device is a hard drive.

3. The guest virtual machine now has a new hard disk device called **/dev/vdc** on Linux (or something similar, depending on what the guest virtual machine OS chooses) or **D: drive** (for example) on Windows. You can now initialize the disk from the guest virtual machine, following the standard procedures for the guest virtual machine's operating system. Refer to [Procedure 13.1, "Adding file-based storage"](#) and [Step 6](#) for an example.



Warning

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

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



Important

Guest virtual machines should not be given write access to whole disks or block devices (for example, **/dev/sdb**). Guest virtual machines with access to whole block devices may be able to modify volume labels, which can be used to compromise the host physical machine system. Use partitions (for example, **/dev/sdb1**) or LVM volumes to prevent this issue.

13.3.3. Managing storage controllers in a guest virtual machine

Starting from Red Hat Enterprise Linux 6.3, SCSI devices can be added to guest virtual machines. Unlike virtio disks, SCSI devices require the presence of a controller in the guest virtual machine. This section details the necessary steps to create a virtual SCSI controller (also known as "Host Bus Adapter", or HBA), and to add SCSI storage to the guest virtual machine.

Procedure 13.3. Creating a virtual SCSI controller

- Display the configuration of the guest virtual machine (**Guest1**) and look for a pre-existing SCSI controller:

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

If a device controller is present, the command will output one or more lines similar to the following:

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

- If the previous step did not show a device controller, create the description for one in a new file and add it to the virtual machine, using the following steps:

- Create the device controller by writing a **<controller>** element in a new file and save this file with an XML extension. **virtio-scsi-controller.xml**, for example.

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

- Associate the device controller you just created in **virtio-scsi-controller.xml** with your guest virtual machine (Guest1, for example):

```
# virsh attach-device --config Guest1 ~/virtio-scsi-controller.xml
```

In this example the **--config** option behaves the same as it does for disks. Refer to [Procedure 13.2, “Adding physical block devices to guests”](#) for more information.

- Add a new SCSI disk or CD-ROM. The new disk can be added using the methods in sections

[Section 13.3.1, “Adding file based storage to a guest”](#) and [Section 13.3.2, “Adding hard drives and other block devices to a guest”](#). In order to create a SCSI disk, specify a target device name that starts with **sd**.

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

Depending on the version of the driver in the guest virtual machine, the new disk may not be detected immediately by a running guest virtual machine. Follow the steps in the *Red Hat Enterprise Linux Storage Administration Guide*.

13.4. Deleting and removing volumes

This section shows how to delete a disk volume from a block based storage pool using the **virsh vol-delete** command. In this example, the volume is **volume 1** and the storage pool is **guest_images**.

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

Chapter 14. Managing guest virtual machines with virsh

virsh is a command line interface tool for managing guest virtual machines and the hypervisor. The **virsh** command-line tool is built on the **libvirt** management API and operates as an alternative to the **qemu-kvm** command and the graphical **virt-manager** application. The **virsh** command can be used in read-only mode by unprivileged users or, with root access, full administration functionality. The **virsh** command is ideal for scripting virtualization administration.

14.1. virsh command quick reference

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

Table 14.1. guest virtual machine management commands

Command	Description
help	Prints basic help information.
list	Lists all guest virtual machines.
dumpxml	Outputs the XML configuration file for the guest virtual machine.
create	Creates a guest virtual machine from an XML configuration file and starts the new guest virtual machine.
start	Starts an inactive guest virtual machine.
destroy	Forces a guest virtual machine to stop.
define	Creates a guest virtual machine from an XML configuration file without starting the new guest virtual machine.
domid	Displays the guest virtual machine's ID.
domuuid	Displays the guest virtual machine's UUID.
dominfo	Displays guest virtual machine information.
domname	Displays the guest virtual machine's name.
domstate	Displays the state of a guest virtual machine.
quit	Quits the interactive terminal.
reboot	Reboots a guest virtual machine.
restore	Restores a previously saved guest virtual machine stored in a file.
resume	Resumes a paused guest virtual machine.
save	Saves the present state of a guest virtual machine to a file.
shutdown	Gracefully shuts down a guest virtual machine.
suspend	Pauses a guest virtual machine.
undefine	Deletes all files associated with a guest virtual machine.
migrate	Migrates a guest virtual machine to another host physical machine.

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

Table 14.2. Resource management options

Command	Description
setmem	Sets the allocated memory for a guest virtual machine. Refer to the virsh manpage for more details.
setmaxmem	Sets maximum memory limit for the hypervisor. Refer to the virsh manpage for more details.
setvcpus	Changes number of virtual CPUs assigned to a guest virtual machine. Refer to the virsh manpage for more details.
vcpuinfo	Displays virtual CPU information about a guest virtual machine.
vcpupin	Controls the virtual CPU affinity of a guest virtual machine.
domblkstat	Displays block device statistics for a running guest virtual machine.
domifstat	Displays network interface statistics for a running guest virtual machine.
attach-device	Attach a device to a guest virtual machine, using a device definition in an XML file.
attach-disk	Attaches a new disk device to a guest virtual machine.
attach-interface	Attaches a new network interface to a guest virtual machine.
update-device	Detaches a disk image from a guest virtual machine's CD-ROM drive. See Section 14.2, “Attaching and updating a device with virsh” for more details.
detach-device	Detaches a device from a guest virtual machine, takes the same kind of XML descriptions as command attach-device .
detach-disk	Detaches a disk device from a guest virtual machine.
detach-interface	Detach a network interface from a guest virtual machine.

The **virsh** commands for managing and creating storage pools and volumes.

For more information on using storage pools with virsh, refer to <http://libvirt.org/formatstorage.html>

Table 14.3. Storage Pool options

Command	Description
find-storage-pool-sources	Returns the XML definition for all storage pools of a given type that could be found.
find-storage-pool-sources host port	Returns data on all storage pools of a given type that could be found as XML. If the host physical machine and port are provided, this command can be run remotely.
pool-autostart	Sets the storage pool to start at boot time.
pool-build	The pool-build command builds a defined pool. This command can format disks and create partitions.
pool-create	pool-create creates and starts a storage pool from the provided XML storage pool definition file.
pool-create-as name	Creates and starts a storage pool from the provided parameters. If the --print-xml parameter is specified, the command prints the XML definition for the storage pool without creating the storage pool.
pool-define	Creates a storage pool from an XML definition file but does not start the new storage pool.
pool-define-as name	Creates but does not start, a storage pool from the provided parameters. If the --print-xml parameter is specified, the command prints the XML definition for the storage pool without creating the storage pool.
pool-destroy	Permanently destroys a storage pool in libvirt . The raw data contained in the storage pool is not changed and can be recovered with the pool-create command.
pool-delete	Destroys the storage resources used by a storage pool. This operation cannot be recovered. The storage pool still exists after this command but all data is deleted.
pool-dumpxml	Prints the XML definition for a storage pool.
pool-edit	Opens the XML definition file for a storage pool in the users default text editor.
pool-info	Returns information about a storage pool.
pool-list	Lists storage pools known to libvirt. By default, pool-list lists pools in use by active guest virtual machines. The --inactive parameter lists inactive pools and the --all parameter lists all pools.
pool-undefine	Deletes the definition for an inactive storage pool.
pool-uuid	Returns the UUID of the named pool.
pool-name	Prints a storage pool's name when provided the UUID of a storage pool.
pool-refresh	Refreshes the list of volumes contained in a storage pool.
pool-start	Starts a storage pool that is defined but inactive.

Table 14.4. Volume options

Command	Description
vol-create	Create a volume from an XML file.
vol-create-from	Create a volume using another volume as input.
vol-create-as	Create a volume from a set of arguments.
vol-clone	Clone a volume.
vol-delete	Delete a volume.
vol-wipe	Wipe a volume.
vol-dumpxml	Show volume information in XML.
vol-info	Show storage volume information.
vol-list	List volumes.
vol-pool	Returns the storage pool for a given volume key or path.
vol-path	Returns the volume path for a given volume name or key.
vol-name	Returns the volume name for a given volume key or path.
vol-key	Returns the volume key for a given volume name or path.

Table 14.5. Secret options

Command	Description
secret-define	Define or modify a secret from an XML file.
secret-dumpxml	Show secret attributes in XML.
secret-set-value	Set a secret value.
secret-get-value	Output a secret value.
secret-undefine	Undefine a secret.
secret-list	List secrets.

Table 14.6. Network filter options

Command	Description
nwfilter-define	Define or update a network filter from an XML file.
nwfilter-undefine	Undefine a network filter.
nwfilter-dumpxml	Show network filter information in XML.
nwfilter-list	List network filters.
nwfilter-edit	Edit XML configuration for a network filter.

This table contains **virsh** command options for snapshots:

Table 14.7. Snapshot options

Command	Description
snapshot-create	Create a snapshot.
snapshot-current	Get the current snapshot.
snapshot-delete	Delete a domain snapshot.
snapshot-dumpxml	Dump XML for a domain snapshot.
snapshot-list	List snapshots for a domain.
snapshot-revert	Revert a domain to a snapshot.

This table contains miscellaneous **virsh** commands:

Table 14.8. Miscellaneous options

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

14.2. Attaching and updating a device with virsh

For information on this procedure refer to [Section 13.3.1, “Adding file based storage to a guest”](#).

14.3. Connecting to the hypervisor

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

```
# virsh connect {name}
```

Where **{name}** is the machine name (hostname) or URL (the output of the **virsh uri** command) of the hypervisor. To initiate a read-only connection, append the above command with **--readonly**.

14.4. Creating a virtual machine XML dump (configuration file)

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

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

This command outputs the guest virtual machine'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 virtual machine (refer to [Editing a guest virtual machine's configuration file](#)). You can edit this XML configuration file to configure additional devices or to deploy additional guest virtual machines.

An example of **virsh dumpxml** output:

```
# virsh dumpxml guest1-rhel6-64
<domain type='kvm'>
  <name>guest1-rhel6-64</name>
  <uuid>b8d7388a-bbf2-db3a-e962-b97ca6e514bd</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc' />
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='threads' />
      <source file='/home/guest-images/guest1-rhel6-64.img' />
      <target dev='vda' bus='virtio' />
      <shareable/<
        <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' />
      </shareable>
    </disk>
    <interface type='bridge'>
      <mac address='52:54:00:b9:35:a9' />
      <source bridge='br0' />
      <model type='virtio' />
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0' />
    </interface>
    <serial type='pty'>
      <target port='0' />
    </serial>
    <console type='pty'>
      <target type='serial' port='0' />
    </console>
    <input type='tablet' bus='usb' />
    <input type='mouse' bus='ps2' />
    <graphics type='vnc' port=''-1' autoport='yes' />
    <sound model='ich6'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0' />
    </sound>
    <video>
      <model type='cirrus' vram='9216' heads='1' />
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />
    </video>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0' />
    </memballoon>
  </devices>
</domain>
```

Note that the `<shareable/>` flag is set. This indicates the device is expected to be shared between domains (assuming the hypervisor and OS support this), which means that caching should be deactivated for that device.

Creating a guest virtual machine from a configuration file

Guest virtual machines can be created from XML configuration files. You can copy existing XML from previously created guest virtual machines or use the **dumpxml** option (refer to [Section 14.4, “Creating a virtual machine XML dump \(configuration file\)”\). To create a guest virtual machine with **virsh** from an XML file:](#)

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

Editing a guest virtual machine's configuration file

Instead of using the **dumpxml** option (refer to [Section 14.4, “Creating a virtual machine XML dump \(configuration file\)”\) guest virtual machines 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 virtual machine named **softwaretesting**:](#)

```
# virsh edit softwaretesting
```

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

14.4.1. Adding multifunction PCI devices to KVM guest virtual machines

This section will demonstrate how to add multi-function PCI devices to KVM guest virtual machines.

1. Run the **virsh edit [guestname]** command to edit the XML configuration file for the guest virtual machine.
2. In the address type tag, add a **multifunction='on'** entry for **function='0x0'**.

This enables the guest virtual machine to use the multifunction PCI devices.

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none' />
<source file='/var/lib/libvirt/images/rhel62-1.img' />
<target dev='vda' bus='virtio' />
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on' />
</disk>
```

For a PCI device with two functions, amend the XML configuration file to include a second device with the same slot number as the first device and a different function number, such as **function='0x1'**.

For Example:

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none' />
<source file='/var/lib/libvirt/images/rhel62-1.img' />
<target dev='vda' bus='virtio' />
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on' />
</disk>
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none' />
<source file='/var/lib/libvirt/images/rhel62-2.img' />
<target dev='vdb' bus='virtio' />
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1' />
</disk>
```

3. **lspci** output from the KVM guest virtual machine shows:

14.5. Suspending, resuming, saving and restoring a guest virtual machine

Suspending a guest virtual machine

Suspend a guest virtual machine with **virsh**:

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

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

Resuming a guest virtual machine

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

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

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

Save a guest virtual machine

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

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

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

Restore a guest virtual machine

Restore a guest virtual machine previously saved with the **virsh save** command ([Save a guest virtual machine](#)) using **virsh**:

```
# virsh restore filename
```

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

14.6. Shutting down, rebooting and force-shutdown of a guest virtual machine

Shut down a guest virtual machine

Shut down a guest virtual machine using the **virsh** command:

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

You can control the behavior of the rebooting guest virtual machine by modifying the **on_shutdown** parameter in the guest virtual machine's configuration file.

Rebooting a guest virtual machine

Reboot a guest virtual machine using **virsh** command:

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

You can control the behavior of the rebooting guest virtual machine by modifying the **on_reboot** element in the guest virtual machine's configuration file.

Forcing a guest virtual machine to stop

Force a guest virtual machine 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 virtual machine. Using **virsh destroy** can corrupt guest virtual machine file systems. Use the **destroy** option only when the guest virtual machine is unresponsive.

14.7. Retrieving guest virtual machine information

Getting the domain ID of a guest virtual machine

To get the domain ID of a guest virtual machine:

```
# virsh domid {domain-name or domain-uuid}
```

Getting the domain name of a guest virtual machine

To get the domain name of a guest virtual machine:

```
# virsh domname {domain-id or domain-uuid}
```

Getting the UUID of a guest virtual machine

To get the Universally Unique Identifier (UUID) for a guest virtual machine:

```
# virsh domuuid {domain-id or domain-name}
```

An example of **virsh domuuid** output:

```
# virsh domuuid r5b2-mySQL01
4a4c59a7-ee3f-c781-96e4-288f2862f011
```

Displaying guest virtual machine information

Using **virsh** with the guest virtual machine's domain ID, domain name or UUID you can display information on the specified guest virtual machine:

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

This is an example of **virsh dominfo** output:

```
# virsh dominfo vr-rhel6u1-x86_64-kvm
Id: 9
Name: vr-rhel6u1-x86_64-kvm
UUID: a03093a1-5da6-a2a2-3baf-a845db2f10b9
OS Type: hvm
State: running
CPU(s): 1
CPU time: 21.6s
Max memory: 2097152 kB
Used memory: 1025000 kB
Persistent: yes
Autostart: disable
Security model: selinux
Security DOI: 0
Security label: system_u:system_r:svirt_t:s0:c612,c921 (permissive)
```

14.8. Retrieving node information

Displaying node information

To display information about the node:

```
# 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
```

Returns basic information about the node, including the model number, number of CPUs, type of CPU, and size of the physical memory. The output corresponds to virNodeInfo structure. Specifically, the "CPU socket(s)" field indicates the number of CPU sockets per NUMA cell.

14.9. Storage pool information

Editing a storage pool definition

The **virsh pool-edit** command takes the name or UUID for a storage pool and opens the XML definition file for a storage pool in the users default text editor.

The **virsh pool-edit** command is equivalent to running the following commands:

```
# virsh pool-dumpxml pool > pool.xml
# vim pool.xml
# virsh pool-define pool.xml
```

**Note**

The default editor is defined by the **\$VISUAL** or **\$EDITOR** environment variables, and default is **vi**.

14.10. Displaying per-guest virtual machine information

Displaying the guest virtual machines

To display the guest virtual machine list and their current states with **virsh**:

```
# virsh list
```

Other options available include:

the **--inactive** option to list inactive guest virtual machines (that is, guest virtual machines that have been defined but are not currently active), and

the **--all** option lists all guest virtual machines. For example:

```
# virsh list --all
 Id Name           State
 -----
 0 Domain-0       running
 1 Domain202      paused
 2 Domain010      inactive
 3 Domain9600     crashed
```

There are seven states that can be visible using this command:

- ▶ Running - The **running** state refers to guest virtual machines which are currently active on a CPU.
- ▶ Idle - The **idle** state indicates that the domain is idle, and may not be running or able to run. This can be caused because the domain is waiting on IO (a traditional wait state) or has gone to sleep because there was nothing else for it to do.
- ▶ Paused - The **paused** state lists domains that are paused. This occurs if an administrator uses the **pause** button in **virt-manager** or **virsh suspend**. When a guest virtual machine is paused it consumes memory and other resources but it is ineligible for scheduling and CPU resources from the hypervisor.
- ▶ Shutdown - The **shutdown** state is for guest virtual machines in the process of shutting down. The guest virtual machine is sent a shutdown signal and should be in the process of stopping its operations gracefully. This may not work with all guest virtual machine operating systems; some operating systems do not respond to these signals.
- ▶ Shut off - The **shut off** state indicates that the domain is not running. This can be caused when a domain completely shuts down or has not been started.
- ▶ Crashed - The **crashed** state indicates that the domain has crashed and can only occur if the guest virtual machine has been configured not to restart on crash.
- ▶ Dying - Domains in the **dying** state are in process of dying, which is a state where the domain has not completely shut-down or crashed.

Displaying virtual CPU information

To display virtual CPU information from a guest virtual machine 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 virtual machine's ID number or name.

The **vcpu** parameter denotes the number of virtualized CPUs allocated to the guest virtual machine. 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.

Additional parameters such as **--config** effect the next boot, whereas **--live** effects the running domain and **--current** effects the current domain

Configuring virtual CPU count

To modify the number of CPUs assigned to a guest virtual machine with **virsh**:

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

This **count** value cannot exceed the number of CPUs that were assigned to the guest virtual machine when it was created.

Configuring memory allocation

To modify a guest virtual machine's memory allocation with **virsh**:

```
# virsh setmem {domain-id or domain-name} count
```

```
# virsh setmem vr-rhel6u1-x86_64-kvm --kilobytes 1025000
```

You must specify the **count** in kilobytes. The new count value cannot exceed the amount you specified when you created the guest virtual machine. Values lower than 64 MB are unlikely to work with most guest virtual machine operating systems. A higher maximum memory value does not affect active guest virtual machines. If the new value is lower than the available memory, it will shrink possibly causing the guest virtual machine to crash.

This command has the following options

- ▶ **--domain** <string> domain name, id or uuid
- ▶ **--size** <number> new memory size, as scaled integer (default KiB)
- ▶ **--config** takes affect next boot
- ▶ **--live** controls the memory of the running domain
- ▶ **--current** controls the memory on the current domain

Displaying guest virtual machine block device information

Use **virsh domblkstat** to display block device statistics for a running guest virtual machine.

```
# virsh domblkstat GuestName block-device
```

Displaying guest virtual machine network device information

Use **virsh domifstat** to display network interface statistics for a running guest virtual machine.

```
# virsh domifstat GuestName interface-device
```

14.11. Managing virtual networks

This section covers managing virtual networks with the **virsh** command. To list virtual networks:

```
# virsh net-list
```

This command generates output similar to:

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.

14.12. Migrating guest virtual machines with virsh

Information on migration using virsh is located in the section entitled Live KVM Migration with virsh Refer to [Section 5.4, “Live KVM migration with virsh”](#)

14.13. Managing snapshots

The sections that follow describe actions that can be done in order to manipulate domain snapshots. Snapshots take the disk, memory, and device state of a domain at a specified point-in-time, and save it for future use. Snapshots have many uses, from saving a "clean" copy of an OS image to saving a domain's state before what may be a potentially destructive operation. Snapshots are identified with a unique name. See [the libvirt website](#) for documentation of the XML format used to represent properties of snapshots.

14.13.1. virsh snapshot create

The **virsh snapshot create** command creates a snapshot for domain with the properties specified in the domain XML file (such as <name> and <description> elements, as well as <disks>). To create a snapshot run:

```
#snapshot-create <domain> <xmlfile> [--redefine [--current] [--no-metadata] [- -halt] [--disk-only] [--reuse-external] [--quiesce] [--atomic] [--live]
```

The domain name, id, or uid may be used as the domain requirement. The XML requirement is a string that must in the very least contain the <name>, <description> and <disks> elements.

The remaining optional arguments are as follows:

- ▶ **--disk-only** - causes the rest of the fields to be ignored, and automatically filled in by libvirt.
- ▶ If the XML file string is completely omitted, then libvirt will choose a value for all fields. The new snapshot will become current, as listed by **snapshot-current**. In addition the snapshot will only include the disk state rather than the usual system checkpoint with guest virtual machine state. Disk snapshots are faster than full system checkpoints, but reverting to a disk snapshot may require **fsck** or journal replays, since it is like the disk state at the point when the power cord is abruptly pulled. Note that mixing **--halt** and **--disk-only** loses any data that was not flushed to disk at the time.
- ▶ **--halt** - causes the domain to be left in an inactive state after the snapshot is created. Mixing **--halt** and **--disk-only** loses any data that was not flushed to disk at the time
- ▶ **--redefine** specifies that if all XML elements produced by **snapshot-dumpxml** are valid; it can be used to migrate snapshot hierarchy from one machine to another, to recreate hierarchy for the case of a transient domain that goes away and is later recreated with the same name and UUID, or to make slight alterations in the snapshot metadata (such as host-specific aspects of the domain XML embedded in the snapshot). When this flag is supplied, the **xmlfile** argument is mandatory, and the domain's current snapshot will not be altered unless the **--current** flag is also given.
- ▶ **--no-metadata** creates the snapshot, but any metadata is immediately discarded (that is, libvirt does not treat the snapshot as current, and cannot revert to the snapshot unless **--redefine** is later used to teach libvirt about the metadata again).
- ▶ **--reuse-external**, if used and snapshot XML requests an external snapshot with a destination of an existing file, then the destination must exist, and is reused; otherwise, a snapshot is refused to avoid losing contents of the existing files.
- ▶ **--quiesce** libvirt will try to freeze and unfreeze the domain's mounted file system(s), using the guest agent. However, if the domain doesn't have a guest agent, snapshot creation will fail. Currently, this requires **--disk-only** to be passed as well.
- ▶ **--atomic** causes libvirt to guarantee that the snapshot either succeeds, or fails with no changes. Note that not all hypervisors support this. If this flag is not specified, then some hypervisors may fail after partially performing the action, and **dumpxml** must be used to see whether any partial changes

occurred.

- ▶ **--live** causes libvirt to take the snapshot while the guest is running. This increases the size of the memory image of the external checkpoint. This is currently supported only for external checkpoints. Existence of snapshot metadata will prevent attempts to undefine a persistent domain. However, for transient domains, snapshot metadata is silently lost when the domain quits running (whether by command such as `destroy` or by internal guest action).

14.13.2. snapshot-create-as-domain

The `virsh snapshot-create-as-domain` command creates a snapshot for domain with the properties specified in the domain XML file (such as `<name>` and `<description>` elements). If these values are not included in the XML string, *libvirt* will choose a value. To create a snapshot run:

```
#snapshot-create-as domain {[--print-xml] | [--no-metadata] [--halt] [--reuse-external]} [name] [description] [--disk-only [--quiesce]] [--atomic] [[--live] [--memspec memspec] [--diskspec] diskspec]
```

The remaining optional arguments are as follows:

- ▶ **--print-xml** creates appropriate XML for `snapshot-create` as output, rather than actually creating a snapshot.
- ▶ **--halt** keeps the domain in an inactive state after the snapshot is created.
- ▶ **--disk-only** creates a snapshot that does not include the guest virtual machine state.
- ▶ **--memspec** can be used to control whether a checkpoint is internal or external. The flag is mandatory, followed by a `memspec` of the form `[file=]name[,snapshot=type]`, where type can be none, internal, or external. To include a literal comma in `file=name`, escape it with a second comma.
- ▶ **--diskspec** option can be used to control how **--disk-only** and external checkpoints create external files. This option can occur multiple times, according to the number of `<disk>` elements in the domain XML. Each `<diskspec>` is in the form `disk[,snapshot=type][,driver=type][,file=name]`. To include a literal comma in `disk` or in `file=name`, escape it with a second comma. A literal **--diskspec** must precede each `diskspec` unless all three of `<domain>`, `<name>`, and `<description>` are also present. For example, a `diskspec` of `vda,snapshot=external,file=/path/to,,new` results in the following XML:

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new' />
</disk>
```

- ▶ **--reuse-external** is specified, and the domain XML or `diskspec` option requests an external snapshot with a destination of an existing file, then the destination must exist, and is reused; otherwise, a snapshot is refused to avoid losing contents of the existing files.
- ▶ **--quiesce** is specified, *libvirt* will try to use guest agent to freeze and unfreeze domain's mounted file systems. However, if domain has no guest agent, snapshot creation will fail. Currently, this requires **--disk-only** to be passed as well.
- ▶ **--no-metadata** creates snapshot data but any metadata is immediately discarded (that is, *libvirt* does not treat the snapshot as current, and cannot revert to the snapshot unless `snapshot-create` is later used to teach *libvirt* about the metadata again). This flag is incompatible with **--print-xml**.
- ▶ **--atomic** will cause *libvirt* to guarantee that the snapshot either succeeds, or fails with no changes. It should be noted that not all hypervisors support this. If this flag is not specified, then some hypervisors may fail after partially performing the action, and `dumpxml` must be used to see whether any partial changes occurred.
- ▶ **--live** causes *libvirt* to take the snapshot while the guest virtual machine is running. This increases the size of the memory image of the external checkpoint. This is currently supported only for external checkpoints.

14.13.3. snapshot-current-domain

This command is used to query which snapshot is currently in use. To use, run:

```
# virsh snapshot-current domain {[--name] | [--security-info] | [snapshotname]}
```

If **snapshotname** is not used, snapshot XML for the domain's current snapshot (if there is one) will be displayed as output. If **--name** is specified, just the current snapshot name instead of the full XML will be sent as output. If **--security-info** is supplied security sensitive information will be included in the XML. Using **snapshotname**, generates a request to make the existing named snapshot become the current snapshot, without reverting it to the domain.

14.13.4. snapshot-edit-domain

This command is used to edit the snapshot that is currently in use. To use, run:

```
#virsh snapshot-edit domain [snapshotname] [--current] {[--rename] [--clone]}
```

If both **snapshotname** and **--current** are specified, it forces the edited snapshot to become the current snapshot. If **snapshotname** is omitted, then **--current** must be supplied, in order to edit the current snapshot.

This is equivalent to the following command sequence below, but it also includes some error checking:

```
# virsh snapshot-dumpxml dom name > snapshot.xml
# vi snapshot.xml [note - this can be any editor]
# virsh snapshot-create dom snapshot.xml --redefine [--current]
```

If **--rename** is specified, then the resulting edited file gets saved in a different file name. If **--clone** is specified, then changing the snapshot name will create a clone of the snapshot metadata. If neither is specified, then the edits will not change the snapshot name. Note that changing a snapshot name must be done with care, since the contents of some snapshots, such as internal snapshots within a single qcows2 file, are accessible only from the original snapshot filename.

14.13.5. snapshot-info-domain

snapshot-info-domain displays information about the snapshots. To use, run:

```
# snapshot-info domain {snapshot | --current}
```

Outputs basic information about a specified **snapshot**, or the current snapshot with **--current**.

14.13.6. snapshot-list-domain

List all of the available snapshots for the given domain, defaulting to show columns for the snapshot name, creation time, and domain state. To use, run:

```
#virsh snapshot-list domain {[--parent | --roots | --tree]} {[[--from] snapshot | --current] [--descendants]} [--metadata] [--no-metadata] [--leaves] [--no-leaves] [--inactive] [--active] [--disk-only] [--internal] [--external]
```

The remaining optional arguments are as follows:

- ▶ **--parent** adds a column to the output table giving the name of the parent of each snapshot. This option may not be used with **--roots** or **--tree**.
- ▶ **--roots** filters the list to show only the snapshots that have no parents. This option may not be used with **--parent** or **--tree**.

- ▶ **--tree** displays output in a tree format, listing just snapshot names. These three options are mutually exclusive. This option may not be used with **--roots** or **--parent**.
- ▶ **--from** filters the list to snapshots which are children of the given snapshot; or if **--current** is provided, will cause the list to start at the current snapshot. When used in isolation or with **--parent**, the list is limited to direct children unless **--descendants** is also present. When used with **--tree**, the use of **--descendants** is implied. This option is not compatible with **--roots**. Note that the starting point of **--from** or **--current** is not included in the list unless the **--tree** option is also present.
- ▶ **--leaves** is specified, the list will be filtered to just snapshots that have no children. Likewise, if **--no-leaves** is specified, the list will be filtered to just snapshots with children. (Note that omitting both options does no filtering, while providing both options will either produce the same list or error out depending on whether the server recognizes the flags) Filtering options are not compatible with **-tree**.
- ▶ **--metadata** is specified, the list will be filtered to just snapshots that involve libvirt metadata, and thus would prevent undefine of a persistent domain, or be lost on destroy of a transient domain. Likewise, if **--no-metadata** is specified, the list will be filtered to just snapshots that exist without the need for libvirt metadata.
- ▶ **--inactive** is specified, the list will be filtered to snapshots that were taken when the domain was shut off. If **--active** is specified, the list will be filtered to snapshots that were taken when the domain was running, and where the snapshot includes the memory state to revert to that running state. If **--disk-only** is specified, the list will be filtered to snapshots that were taken when the domain was running, but where the snapshot includes only disk state.
- ▶ **--internal** is specified, the list will be filtered to snapshots that use internal storage of existing disk images. If **--external** is specified, the list will be filtered to snapshots that use external files for disk images or memory state.

14.13.7. `snapshot-dumpxml` domain snapshot

`virsh snapshot-dumpxml domain snapshot` outputs the snapshot XML for the domain's snapshot named `snapshot`. To use, run:

```
# virsh snapshot-dumpxml domain snapshot [--security-info]
```

The **--security-info** option will also include security sensitive information. Use **snapshot-current** to easily access the XML of the current snapshot.

14.13.8. `snapshot-parent` domain

Outputs the name of the parent snapshot, if any, for the given snapshot, or for the current snapshot with **--current**. To use, run:

```
#virsh snapshot-parent domain {snapshot | --current}
```

14.13.9. `snapshot-revert` domain

Reverts the given domain to the snapshot specified by `snapshot`, or to the current snapshot with **--current**.



Warning

Be aware that this is a destructive action; any changes in the domain since the last snapshot was taken will be lost. Also note that the state of the domain after `snapshot-revert` is complete will be the state of the domain at the time the original snapshot was taken.

To revert the snapshot, run

```
# snapshot-revert domain {snapshot | --current} [{--running | --paused}] [--force]
```

Normally, reverting to a snapshot leaves the domain in the state it was at the time the snapshot was created, except that a disk snapshot with no guest virtual machine state leaves the domain in an inactive state. Passing either the **--running** or **--paused** flag will perform additional state changes (such as booting an inactive domain, or pausing a running domain). Since transient domains cannot be inactive, it is required to use one of these flags when reverting to a disk snapshot of a transient domain.

There are two cases where a **snapshot revert** involves extra risk, which requires the use of **--force** to proceed. One is the case of a snapshot that lacks full domain information for reverting configuration; since libvirt cannot prove that the current configuration matches what was in use at the time of the snapshot, supplying **--force** assures *libvirt* that the snapshot is compatible with the current configuration (and if it is not, the domain will likely fail to run). The other is the case of reverting from a running domain to an active state where a new hypervisor has to be created rather than reusing the existing hypervisor, because it implies drawbacks such as breaking any existing VNC or Spice connections; this condition happens with an active snapshot that uses a provably incompatible configuration, as well as with an inactive snapshot that is combined with the **--start** or **--pause** flag.

14.13.10. **snapshot-delete domain**

snapshot-delete domain deletes the snapshot for the specified domain. To do this, run:

```
# virsh snapshot-delete domain {snapshot | --current} [--metadata] [{--children | --children-only}]
```

This command Deletes the snapshot for the domain named **snapshot**, or the current snapshot with **--current**. If this snapshot has child snapshots, changes from this snapshot will be merged into the children. If the option **--children** is used, then it will delete this snapshot and any children of this snapshot. If **--children-only** is used, then it will delete any children of this snapshot, but leave this snapshot intact. These two flags are mutually exclusive.

The **--metadata** is used it will delete the snapshot's metadata maintained by *libvirt*, while leaving the snapshot contents intact for access by external tools; otherwise deleting a snapshot also removes its data contents from that point in time.

14.14. Guest virtual machine CPU model configuration

14.14.1. Introduction

Every hypervisor has its own policy for what a guest virtual machine will see for its CPUs by default. Whereas some hypervisors decide which CPU host physical machine features will be available for the guest virtual machine, QEMU/KVM presents the guest virtual machine with a generic model named **qemu32** or **qemu64**. These hypervisors perform more advanced filtering, classifying all physical CPUs into a handful of groups and have one baseline CPU model for each group that is presented to the guest virtual machine. Such behavior enables the safe migration of guest virtual machines between host physical machines, provided they all have physical CPUs that classify into the same group. *libvirt* does not typically enforce policy itself, rather it provides the mechanism on which the higher layers define their own desired policy. Understanding how to obtain CPU model information and define a suitable guest virtual machine CPU model is critical to ensure guest virtual machine migration is successful between host physical machines. Note that a hypervisor can only emulate features that it is aware of and features that were created after the hypervisor was released may not be emulated.

14.14.2. Learning about the host physical machine CPU model

The **virsh capabilities** command displays an XML document describing the capabilities of the hypervisor connection and host physical machine. The XML schema displayed has been extended to provide information about the host physical machine CPU model. One of the big challenges in describing

a CPU model is that every architecture has a different approach to exposing their capabilities. On x86, the capabilities of a modern CPU are exposed via the CPUID instruction. Essentially this comes down to a set of 32-bit integers with each bit given a specific meaning. Fortunately AMD and Intel agree on common semantics for these bits. Other hypervisors expose the notion of CPUID masks directly in their guest virtual machine configuration format. However, QEMU/KVM supports far more than just the x86 architecture, so CPUID is clearly not suitable as the canonical configuration format. QEMU ended up using a scheme which combines a CPU model name string, with a set of named flags. On x86, the CPU model maps to a baseline CPUID mask, and the flags can be used to then toggle bits in the mask on or off. libvirt decided to follow this lead and uses a combination of a model name and flags.

It is not practical to have a database listing all known CPU models, so libvirt has a small list of baseline CPU model names. It chooses the one that shares the greatest number of CPUID bits with the actual host physical machine CPU and then lists the remaining bits as named features. Notice that libvirt does not display which features the baseline CPU contains. This might seem like a flaw at first, but as will be explained in this section, it is not actually necessary to know this information.

14.14.3. Determining a compatible CPU model to suit a pool of host physical machines

Now that it is possible to find out what CPU capabilities a single host physical machine has, the next step is to determine what CPU capabilities are best to expose to the guest virtual machine. If it is known that the guest virtual machine will never need to be migrated to another host physical machine, the host physical machine CPU model can be passed straight through unmodified. A virtualized data center may have a set of configurations that can guarantee all servers will have 100% identical CPUs. Again the host physical machine CPU model can be passed straight through unmodified. The more common case, though, is where there is variation in CPUs between host physical machines. In this mixed CPU environment, the lowest common denominator CPU must be determined. This is not entirely straightforward, so libvirt provides an API for exactly this task. If libvirt is provided a list of XML documents, each describing a CPU model for a host physical machine, libvirt will internally convert these to CPUID masks, calculate their intersection, and convert the CPUID mask result back into an XML CPU description.

Here is an example of what libvirt reports as the capabilities on a basic workstation, when the **virsh capabilities** is executed:

```
<capabilities>
<host>
  <cpu>
    <arch>i686</arch>
    <model>pentium3</model>
    <topology sockets='1' cores='2' threads='1' />
    <feature name='lahf_lm' />
    <feature name='lm' />
    <feature name='xptr' />
    <feature name='cx16' />
    <feature name='ssse3' />
    <feature name='tm2' />
    <feature name='est' />
    <feature name='vmx' />
    <feature name='ds_cpl' />
    <feature name='monitor' />
    <feature name='pni' />
    <feature name='pbe' />
    <feature name='tm' />
    <feature name='ht' />
    <feature name='ss' />
    <feature name='sse2' />
    <feature name='acpi' />
    <feature name='ds' />
    <feature name='clflush' />
    <feature name='apic' />
  </cpu>
</host>
</capabilities>
```

Figure 14.1. Pulling host physical machine's CPU model information

Now compare that to any random server, with the same **virsh capabilities** command:

```
<capabilities>
<host>
  <cpu>
    <arch>x86_64</arch>
    <model>phenom</model>
    <topology sockets='2' cores='4' threads='1' />
    <feature name='osvw' />
    <feature name='3dnowprefetch' />
    <feature name='misalignsse' />
    <feature name='sse4a' />
    <feature name='abm' />
    <feature name='cr8legacy' />
    <feature name='extapic' />
    <feature name='cmp_legacy' />
    <feature name='lahf_lm' />
    <feature name='rdtscp' />
    <feature name='pdpe1gb' />
    <feature name='popcnt' />
    <feature name='cx16' />
    <feature name='ht' />
    <feature name='vme' />
  </cpu>
  ...snip...
```

Figure 14.2. Generate CPU description from a random server

To see if this CPU description is compatible with the previous workstation CPU description, use the **virsh cpu-compare** command.

The reduced content was stored in a file named **virsh-caps-workstation-cpu-only.xml** and the **virsh cpu-compare** command can be executed on this file:

```
# virsh cpu-compare virsh-caps-workstation-cpu-only.xml
Host physical machine CPU is a superset of CPU described in virsh-caps-
workstation-cpu-only.xml
```

As seen in this output, *libvirt* is correctly reporting that the CPUs are not strictly compatible. This is because there are several features in the server CPU that are missing in the client CPU. To be able to migrate between the client and the server, it will be necessary to open the XML file and comment out some features. To determine which features need to be removed, run the **virsh cpu-baseline** command, on the **both-cpus.xml** which contains the CPU information for both machines. Running `# virsh cpu-baseline both-cpus.xml`, results in:

```
<cpu match='exact'>
  <model>pentium3</model>
  <feature policy='require' name='lahf_lm'/>
  <feature policy='require' name='lm'/>
  <feature policy='require' name='cx16'/>
  <feature policy='require' name='monitor'/>
  <feature policy='require' name='pni'/>
  <feature policy='require' name='ht'/>
  <feature policy='require' name='sse2'/>
  <feature policy='require' name='clflush'/>
  <feature policy='require' name='apic'/>
</cpu>
```

Figure 14.3. Composite CPU baseline

This composite file shows which elements are in common. Everything that is not in common should be commented out.

14.15. Configuring the guest virtual machine CPU model

For simple defaults, the guest virtual machine CPU configuration accepts the same basic XML representation as the host physical machine capabilities XML exposes. In other words, the XML from the **cpu-baseline** virsh command can now be copied directly into the guest virtual machine XML at the top level under the `<domain>` element. In the previous XML snippet, there are a few extra attributes available when describing a CPU in the guest virtual machine XML. These can mostly be ignored, but for the curious here is a quick description of what they do. The top level `<cpu>` element has an attribute called `match` with possible values of:

- ▶ `match='minimum'` - the host physical machine CPU must have at least the CPU features described in the guest virtual machine XML. If the host physical machine has additional features beyond the guest virtual machine configuration, these will also be exposed to the guest virtual machine.
- ▶ `match='exact'` - the host physical machine CPU must have at least the CPU features described in the guest virtual machine XML. If the host physical machine has additional features beyond the guest virtual machine configuration, these will be masked out from the guest virtual machine.
- ▶ `match='strict'` - the host physical machine CPU must have exactly the same CPU features described in the guest virtual machine XML.

The next enhancement is that the <feature> elements can each have an extra 'policy' attribute with possible values of:

- ▶ policy='force' - expose the feature to the guest virtual machine even if the host physical machine does not have it. This is usually only useful in the case of software emulation.
- ▶ policy='require' - expose the feature to the guest virtual machine and fail if the host physical machine does not have it. This is the sensible default.
- ▶ policy='optional' - expose the feature to the guest virtual machine if it happens to support it.
- ▶ policy='disable' - if the host physical machine has this feature, then hide it from the guest virtual machine.
- ▶ policy='forbid' - if the host physical machine has this feature, then fail and refuse to start the guest virtual machine.

The 'forbid' policy is for a niche scenario where an incorrectly functioning application will try to use a feature even if it is not in the CPUID mask, and you wish to prevent accidentally running the guest virtual machine on a host physical machine with that feature. The 'optional' policy has special behavior with respect to migration. When the guest virtual machine is initially started the flag is optional, but when the guest virtual machine is live migrated, this policy turns into 'require', since you cannot have features disappearing across migration.

14.16. Managing resources for guest virtual machines

virsh allows the grouping and allocation of resources on a per guest virtual machine basis. This is managed by the *libvirt daemon* which creates *cgroups* and manages them on behalf of the guest virtual machine. The only thing that is left for the system administrator to do is to either query or set tunables against specified guest virtual machines. The following tunables may be used:

- ▶ **memory** - The memory controller allows for setting limits on RAM and swap usage and querying cumulative usage of all processes in the group
- ▶ **cpuset** - The CPU set controller binds processes within a group to a set of CPUs and controls migration between CPUs.
- ▶ **cpuacct** - The CPU accounting controller provides information about CPU usage for a group of processes.
- ▶ **cpu** - The CPU scheduler controller controls the prioritization of processes in the group. This is similar to granting **nice** level privileges.
- ▶ **devices** - The devices controller grants access control lists on character and block devices.
- ▶ **freezer** - The freezer controller pauses and resumes execution of processes in the group. This is similar to **SIGSTOP** for the whole group.
- ▶ **net_cls** - The network class controller manages network utilization by associating processes with a **tc** network class.

In creating a group hierarchy cgroup will leave mount point and directory setup entirely to the administrators' discretion and is more complex than just adding some mount points to **/etc/fstab**. It is necessary to setup the directory hierarchy and decide how processes get placed within it. This can be done with the following virsh commands:

- ▶ **schedinfo** - described in [Section 14.17, “Setting schedule parameters”](#)
- ▶ **blkdeviotune** - described in [Section 14.18, “Disk I/O throttling”](#)
- ▶ **blkiotune** - described in [Section 14.19, “Display or set block I/O parameters”](#)
- ▶ **domiftune** - described in [Section 14.20, “Setting network interface bandwidth parameters”](#)
- ▶ **memtune** - described in [Section 14.21, “Configuring memory Tuning”](#)

14.17. Setting schedule parameters

schedinfo allows scheduler parameters to be passed to guest virtual machines. The following command format should be used:

```
#virsh schedinfo DOMAIN [--set] [--weight] [--cap] [--current] [--config] [--live]
```

Each parameter is explained below:

- ▶ **domain** - this is the guest virtual machine domain
- ▶ **--set** - the string placed here is the controller or action that is to be called. Additional parameters or values if required should be added as well.
- ▶ **--current** - when used with **--set**, will use the specified **set** string as the current scheduler information. When used without will display the current scheduler information.
- ▶ **--config** - when used with **--set**, will use the specified **set** string on the next reboot. When used without will display the scheduler information that is saved in the configuration file.
- ▶ **--live** - when used with **--set**, will use the specified **set** string on a guest virtual machine that is currently running. When used without will display the configuration setting currently used by the running virtual machine

The scheduler can be set with any of the following parameters: **cpu_shares**, **vcpu_period** and **vcpu_quota**.

Example 14.1. schedinfo show

This example shows the shell guest virtual machine's schedule information

```
# virsh schedinfo shell
Scheduler      : posix
cpu_shares    : 1024
vcpu_period   : 100000
vcpu_quota    : -1
```

Example 14.2. schedinfo set

In this example, the **cpu_shares** is changed to 2046. This effects the current state and not the configuration file.

```
# virsh schedinfo --set cpu_shares=2046 shell
Scheduler      : posix
cpu_shares    : 2046
vcpu_period   : 100000
vcpu_quota    : -1
```

14.18. Disk I/O throttling

virsh blkdeviotune sets disk I/O throttling for a specified guest virtual machine. This can prevent a guest virtual machine from overutilizing shared resources and thus impacting the performance of other guest virtual machines. The following format should be used:

```
# virsh blkdeviotune <domain> <device> [[--config] [--live] | [--current]]
[[total-bytes-sec] | [read-bytes-sec] [write-bytes-sec]] [[total-iops-sec]
[read-iops-sec] [write-iops-sec]]
```

The only required parameter is the domain name of the guest virtual machine, the **--config**, **--live**, and **--current** functions the same as in [Section 14.17, “Setting schedule parameters”](#). If no limit is

specified, it will query current I/O limits setting. Otherwise, alter the limits with the following flags:

- ▶ **--total-bytes-sec** - specifies total throughput limit in bytes per second.
- ▶ **--read-bytes-sec** - specifies read throughput limit in bytes per second.
- ▶ **--write-bytes-sec** - specifies write throughput limit in bytes per second.
- ▶ **--total-iops-sec** - specifies total I/O operations limit per second.
- ▶ **--read-iops-sec** - specifies read I/O operations limit per second.
- ▶ **--write-iops-sec** - specifies write I/O operations limit per second.

For more information refer to the blkdeviotune section of the virsh MAN page. For an example domain XML refer to [Figure 20.24, “Devices - Hard drives, floppy disks, CDROMs”](#).

14.19. Display or set block I/O parameters

blkiotune sets and or displays the I/O parameters for a specified guest virtual machine. The following format should be used:

```
# virsh blkiotune domain [--weight weight] [--device-weights device-weights]
[[--config] [--live] | [--current]]
```

The only required parameter is the domain name of the guest virtual machine, the **--config**, **--live**, and **--current** functions the same as in [Section 14.17, “Setting schedule parameters”](#). If no weight is specified, it will query current I/O limits setting. Otherwise, alter the weights with the following flags:

- ▶ **--weight** allows the range [100 - 1000].
- ▶ **--device-weights** is a single string listing one or more device/weight pairs, in the format of **/path/to/device,weight,/path/to/device,weight**. Each weight is within the range [100-1000], or the value 0 to remove that device from per-device listings. Only the devices listed in the string are modified; any existing per-device weights for other devices remain unchanged.

Refer to [Section 20.9, “Block I/O tuning”](#) for the Domain XML example.

14.20. Setting network interface bandwidth parameters

domiftune sets the guest virtual machine's network interface bandwidth parameters. The following format should be used:

```
#virsh domiftune domain interface-device [[--config] [--live] | [--current]]
[--inbound average,peak,burst] [--outbound average,peak,burst]
```

The only required parameter is the domain name and interface device of the guest virtual machine, the **--config**, **--live**, and **--current** functions the same as in [Section 14.17, “Setting schedule parameters”](#). If no limit is specified, it will query current network interface setting. Otherwise, alter the limits with the following flags:

- ▶ <interface-device> This is mandatory and it will set or query the domain's network interface's bandwidth parameters. **interface-device** can be the interface's target name (<target dev='name'>), or the MAC address.
- ▶ If no **--inbound** or **--outbound** is specified, this command will query and show the bandwidth settings. Otherwise, it will set the inbound or outbound bandwidth. **average,peak,burst** is the same as in command attach-interface.

14.21. Configuring memory Tuning

The element memtune provides details regarding the memory tunable parameters for the domain. If this is omitted, it defaults to the OS provided defaults. For QEMU/KVM, the parameters are applied to the

QEMU process as a whole. Thus, when counting them, one needs to add up guest virtual machine RAM, guest virtual machine video RAM, and some memory overhead of QEMU itself. The last piece is hard to determine so one needs guess and try. For each tunable, it is possible to designate which unit the number is in on input, using the same values as for <memory>. For backwards compatibility, output is always in KiB. units.

Here is an example XML with the memtune options used:

```
<domain>
  <memtune>
    <hard_limit unit='G'>1</hard_limit>
    <soft_limit unit='M'>128</soft_limit>
    <swap_hard_limit unit='G'>2</swap_hard_limit>
    <min_guarantee unit='bytes'>67108864</min_guarantee>
  </memtune>
  ...
</domain>
```

memtune has the following options:

- ▶ **hard_limit** - The optional `hard_limit` element is the maximum memory the guest virtual machine can use. The units for this value are kibibytes (i.e. blocks of 1024 bytes)
- ▶ **soft_limit** - The optional `soft_limit` element is the memory limit to enforce during memory contention. The units for this value are kibibytes (i.e. blocks of 1024 bytes)
- ▶ **swap_hard_limit** - The optional `swap_hard_limit` element is the maximum memory plus swap the guest virtual machine can use. The units for this value are kibibytes (i.e. blocks of 1024 bytes). This has to be more than `hard_limit` value provided
- ▶ **min_guarantee** - The optional `min_guarantee` element is the guaranteed minimum memory allocation for the guest virtual machine. The units for this value are kibibytes (i.e. blocks of 1024 bytes)

```
# virsh memtune vr-rhel6u1-x86_64-kvm --hard-limit 512000
# virsh memtune vr-rhel6u1-x86_64-kvm
hard_limit      : 512000 kB
soft_limit      : unlimited
swap_hard_limit: unlimited
```

`hard_limit` is 512000 kB, it is maximum memory the guest virtual machine domain can use.

14.22. Converting QEMU arguments to domain XML

The `virsh domxml-from-native` provides a way to convert an existing set of QEMU arguments into a guest description using libvirt Domain XML that can then be used by libvirt. Please note that this command is intended to be used only to convert existing qemu guests previously started from the command line in order to allow them to be managed through libvirt. The method described here should not be used to create new guests from scratch. New guests should be created using either virsh or virt-manager. Additional information can be found [here](#).

Suppose you have a QEMU guest with the following args file:

To convert this to a domain XML file so that the guest can be managed by libvirt, run:

```
$ virsh domxml-from-native qemu-argv demo.args
```

This command turns the args file above, into this domain XML file:

```
<domain type='qemu'>
  <uuid>00000000-0000-0000-000000000000</uuid>
  <memory>219136</memory>
  <currentMemory>219136</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='hd' />
  </os>
  <clock offset='utc' />
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/bin/qemu</emulator>
    <disk type='block' device='disk'>
      <source dev='/dev/HostVG/QEMUGuest1' />
      <target dev='hda' bus='ide' />
    </disk>
  </devices>
</domain>
```

Chapter 15. Managing guests with the Virtual Machine Manager (virt-manager)

This section describes the Virtual Machine Manager (**virt-manager**) windows, dialog boxes, and various GUI controls.

virt-manager provides a graphical view of hypervisors and guests on your host system and on remote host systems. **virt-manager** can perform virtualization management tasks, including:

- ▶ defining and creating guests,
- ▶ assigning memory,
- ▶ assigning virtual CPUs,
- ▶ monitoring operational performance,
- ▶ saving and restoring, pausing and resuming, and shutting down and starting guests,
- ▶ links to the textual and graphical consoles, and
- ▶ live and offline migrations.

15.1. Starting virt-manager

To start **virt-manager** session open the **Applications** menu, then the **System Tools** menu and select **Virtual Machine Manager (virt-manager)**.

The **virt-manager** main window appears.

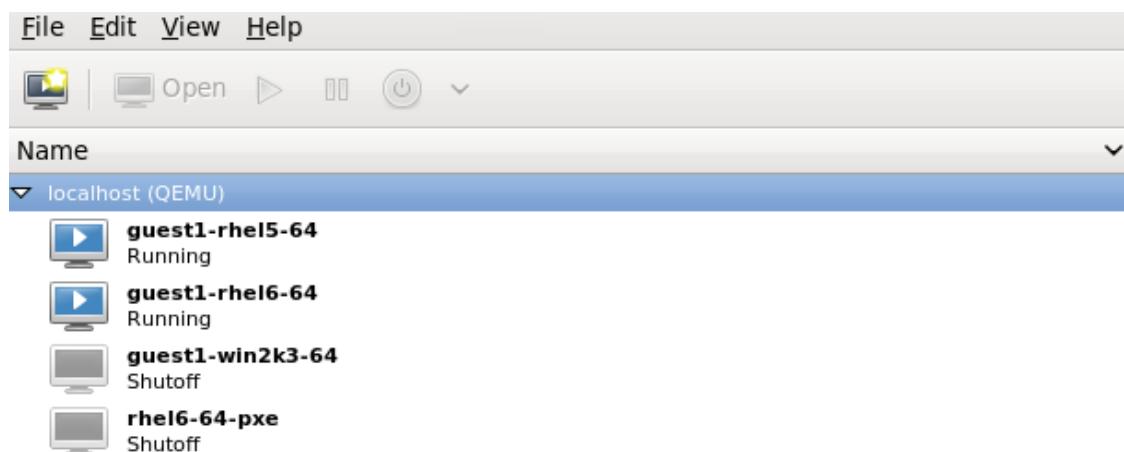


Figure 15.1. Starting virt-manager

Alternatively, **virt-manager** can be started remotely using ssh as demonstrated in the following command:

```
ssh -X host's address
[remotehost]# virt-manager
```

Using **ssh** to manage virtual machines and hosts is discussed further in [Section 6.1, “Remote management with SSH”](#).

15.2. 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.

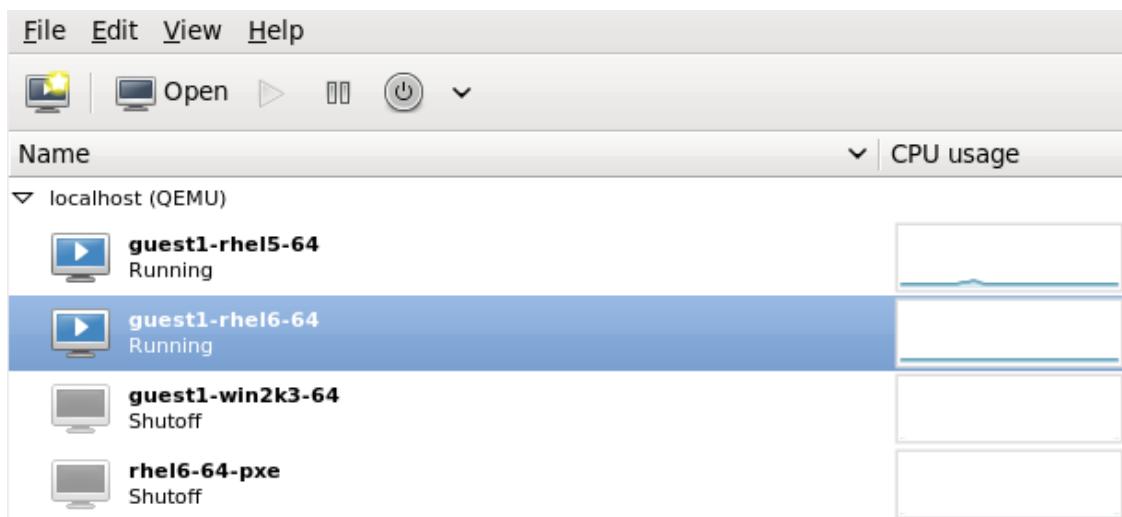


Figure 15.2. Virtual Machine Manager main window

15.3. The virtual hardware details window

The virtual hardware details window displays information about the virtual hardware configured for the guest. Virtual hardware resources can be added, removed and modified in this window. To access the virtual hardware details window, click on the icon in the toolbar.

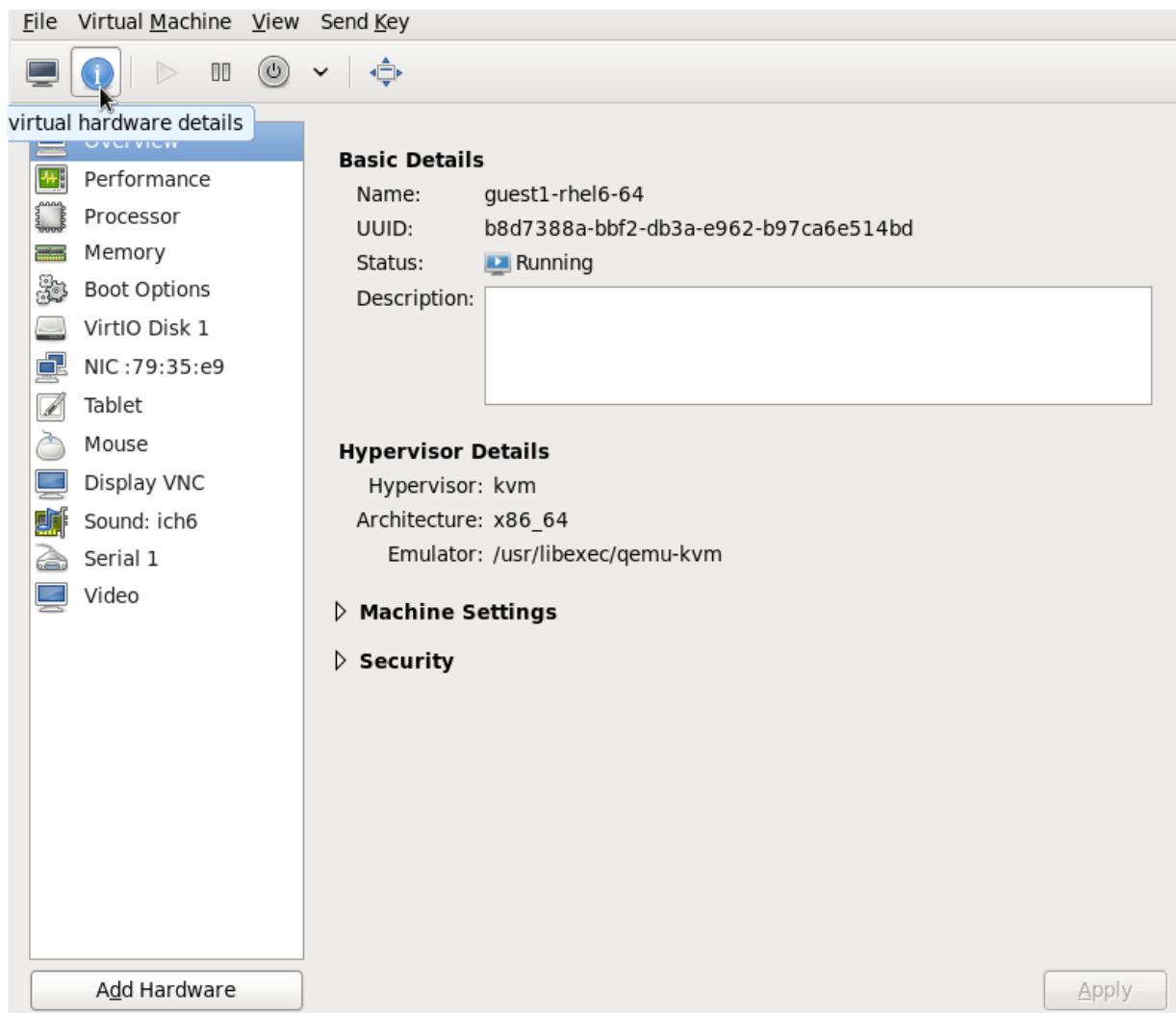


Figure 15.3. The virtual hardware details icon

Clicking the icon displays the virtual hardware details window.

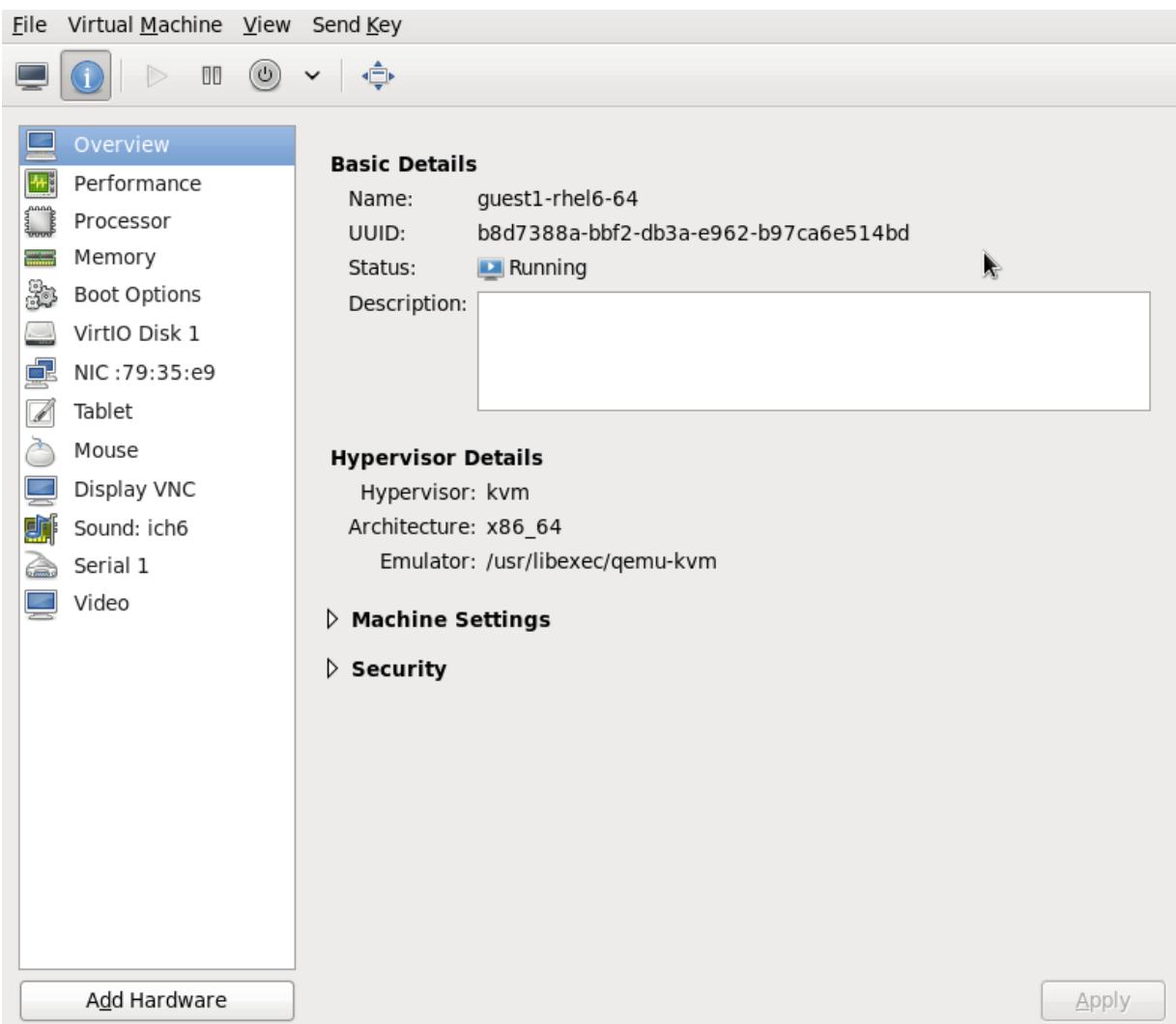


Figure 15.4. The virtual hardware details window

15.4. Virtual Machine graphical console

This window displays a guest's graphical console. Guests can use several different protocols to export their graphical framebuffers: **virt-manager** supports **VNC** and **SPICE**. If your virtual machine is set to require authentication, the Virtual Machine graphical console prompts you for a password before the display appears.

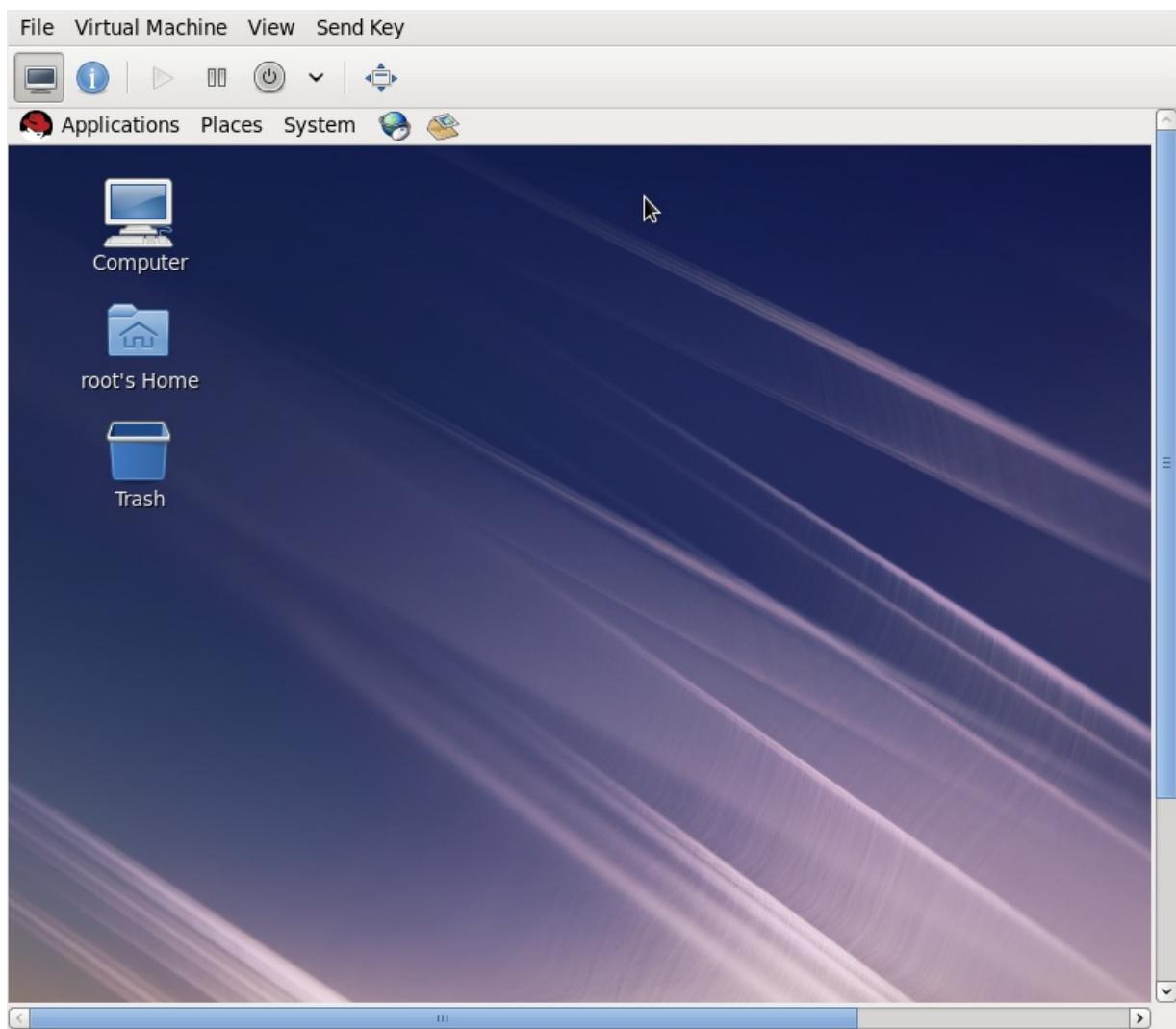


Figure 15.5. Graphical console window

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'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. Although virt-manager is configured to listen to other public network interfaces and alternative methods can be configured, it is not recommended.

Remote administration can be performed by tunneling over SSH which encrypts the traffic. Although VNC can be configured to access remotely without tunneling over SSH, for security reasons, it is not recommended. To remotely administer the guest follow the instructions in: [Chapter 6. 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+F1) to prevent them from being sent to the guest machine. You can use the **Send key** menu option to send these sequences. From the guest machine window, click the **Send key** menu and select the key sequence to send. In addition, from this menu you can also capture the screen output.

SPICE is an alternative to VNC available for Red Hat Enterprise Linux.

15.5. Adding a remote connection

This procedure covers how to set up a connection to a remote system using **virt-manager**.

1. To create a new connection open the **File** menu and select the **Add Connection...** menu item.
2. The **Add Connection** wizard appears. Select the hypervisor. For Red Hat Enterprise Linux 6 systems select **QEMU/KVM**. Select Local for the local system or one of the remote connection options and click **Connect**. This example uses Remote tunnel over SSH which works on default installations. For more information on configuring remote connections refer to [Chapter 6, Remote management of guests](#)

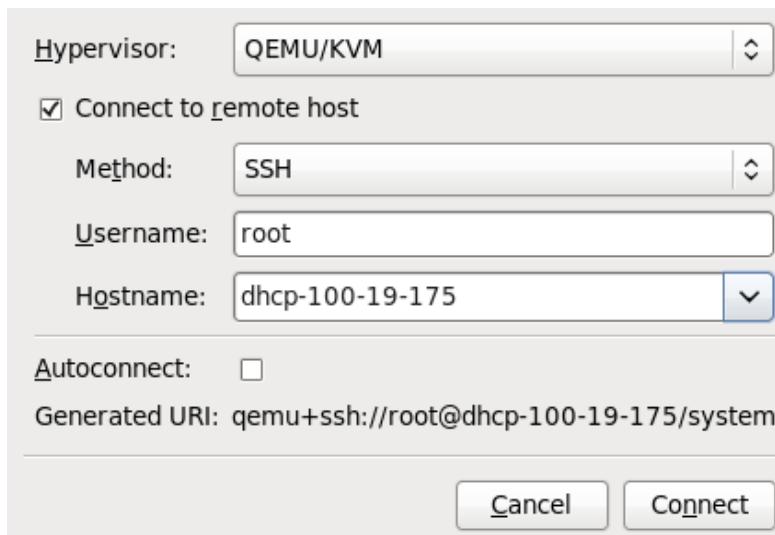


Figure 15.6. Add Connection

3. Enter the root password for the selected host when prompted.

A remote host is now connected and appears in the main **virt-manager** window.

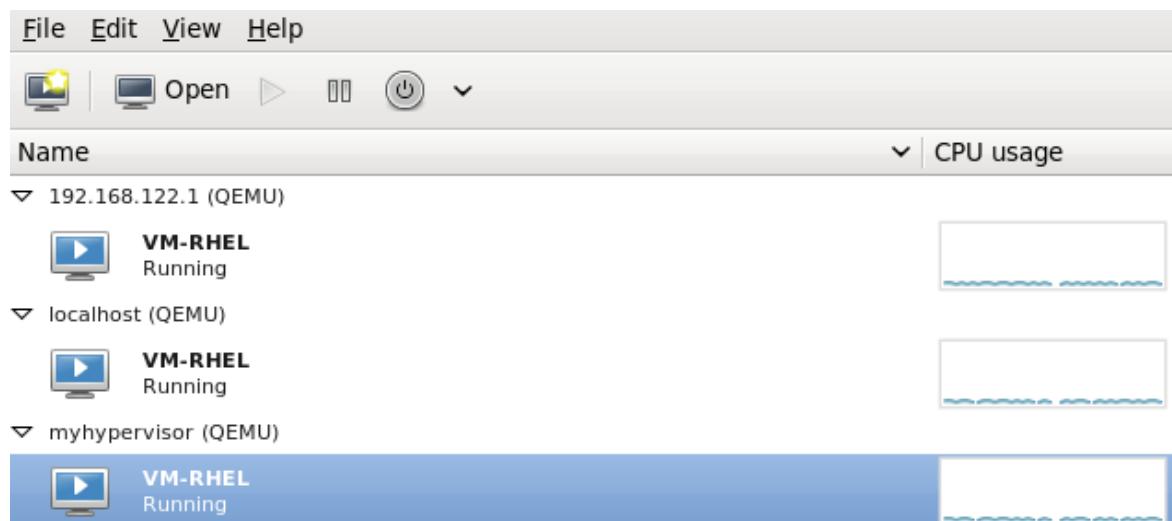


Figure 15.7. Remote host in the main virt-manager window

15.6. Displaying guest details

You can use the Virtual Machine Monitor to view activity information for any virtual machines on your system.

To view a virtual system's details:

1. In the Virtual Machine Manager main window, highlight the virtual machine that you want to view.

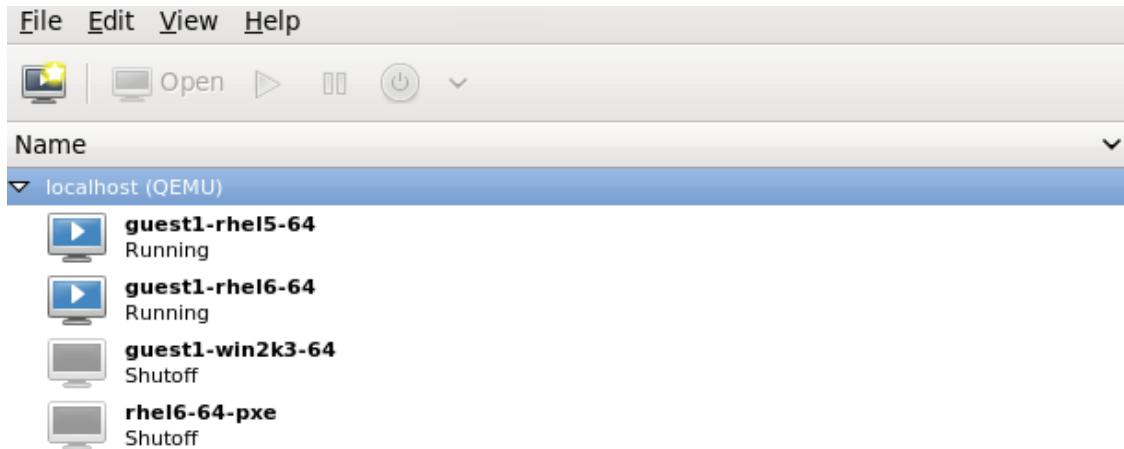


Figure 15.8. Selecting a virtual machine to display

2. From the Virtual Machine Manager **Edit** menu, select **Virtual Machine Details**.

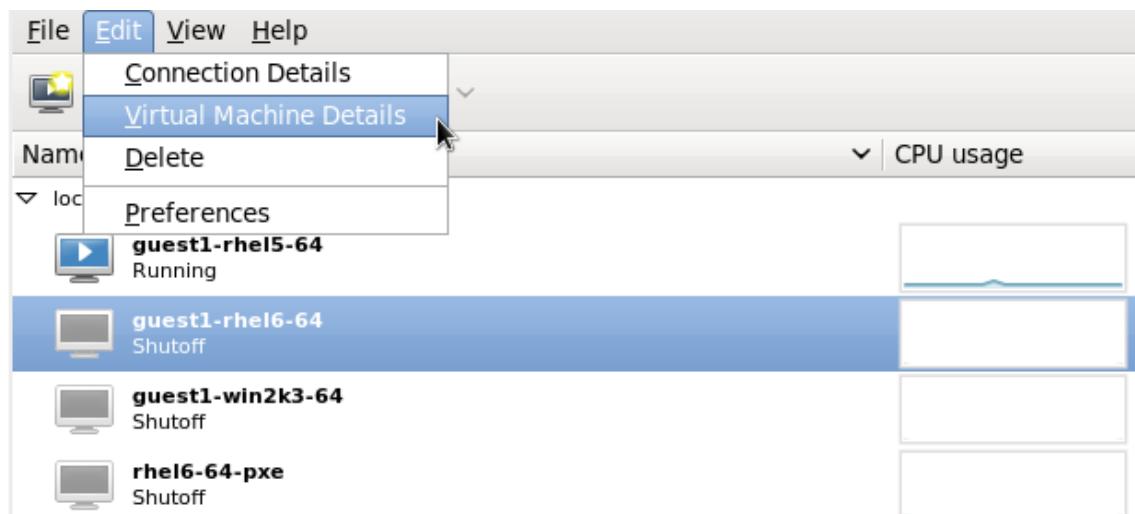


Figure 15.9. Displaying the virtual machine details

When the Virtual Machine details window opens, there may be a console displayed. Should this happen, click **View** and then select **Details**. The Overview window opens first by default. To go back to this window, select **Overview** from the navigation pane on the left hand side.

The **Overview** view shows a summary of configuration details for the guest.

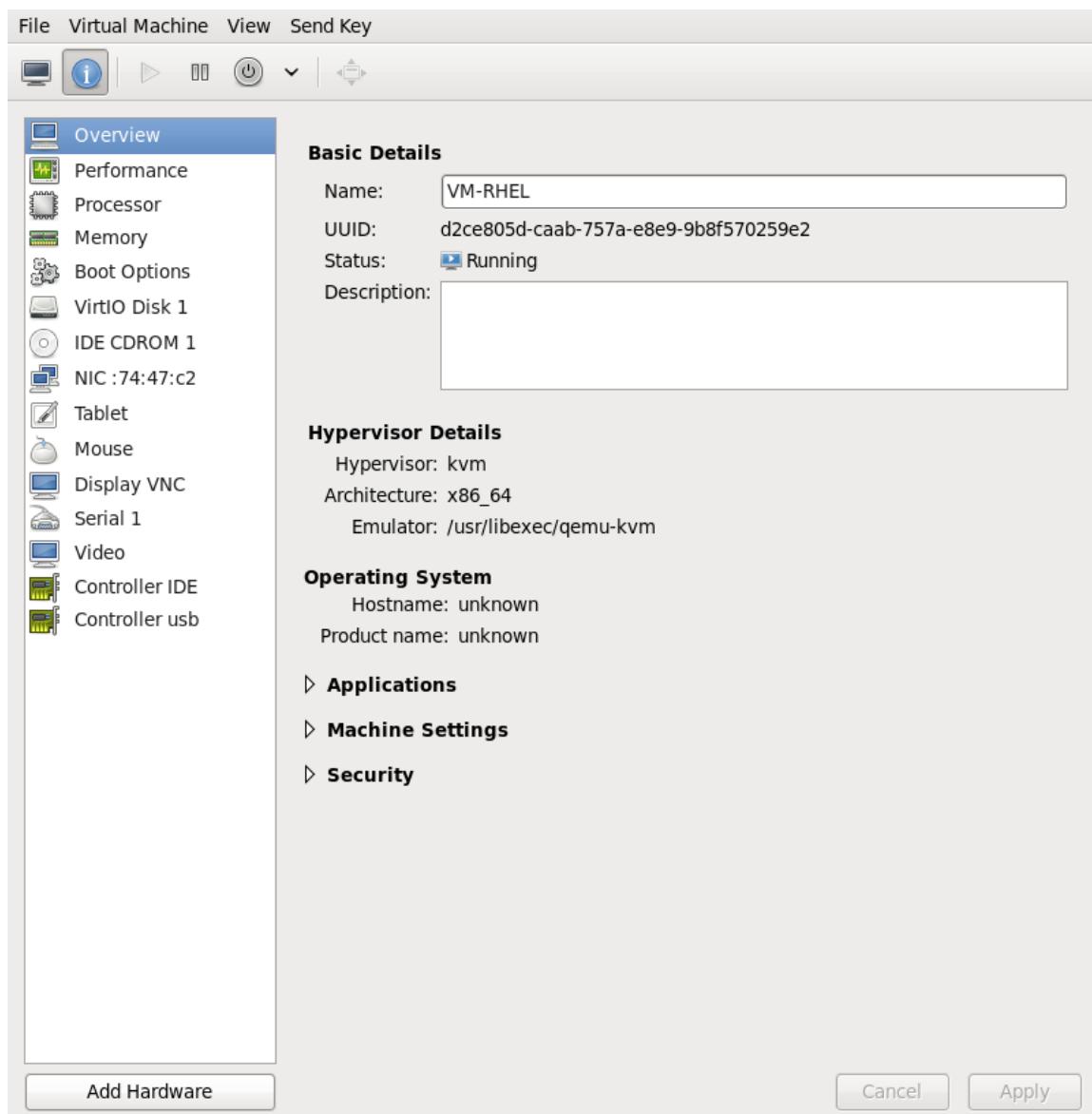


Figure 15.10. Displaying guest details overview

3. Select **Performance** from the navigation pane on the left hand side.

The **Performance** view shows a summary of guest performance, including CPU and Memory usage.

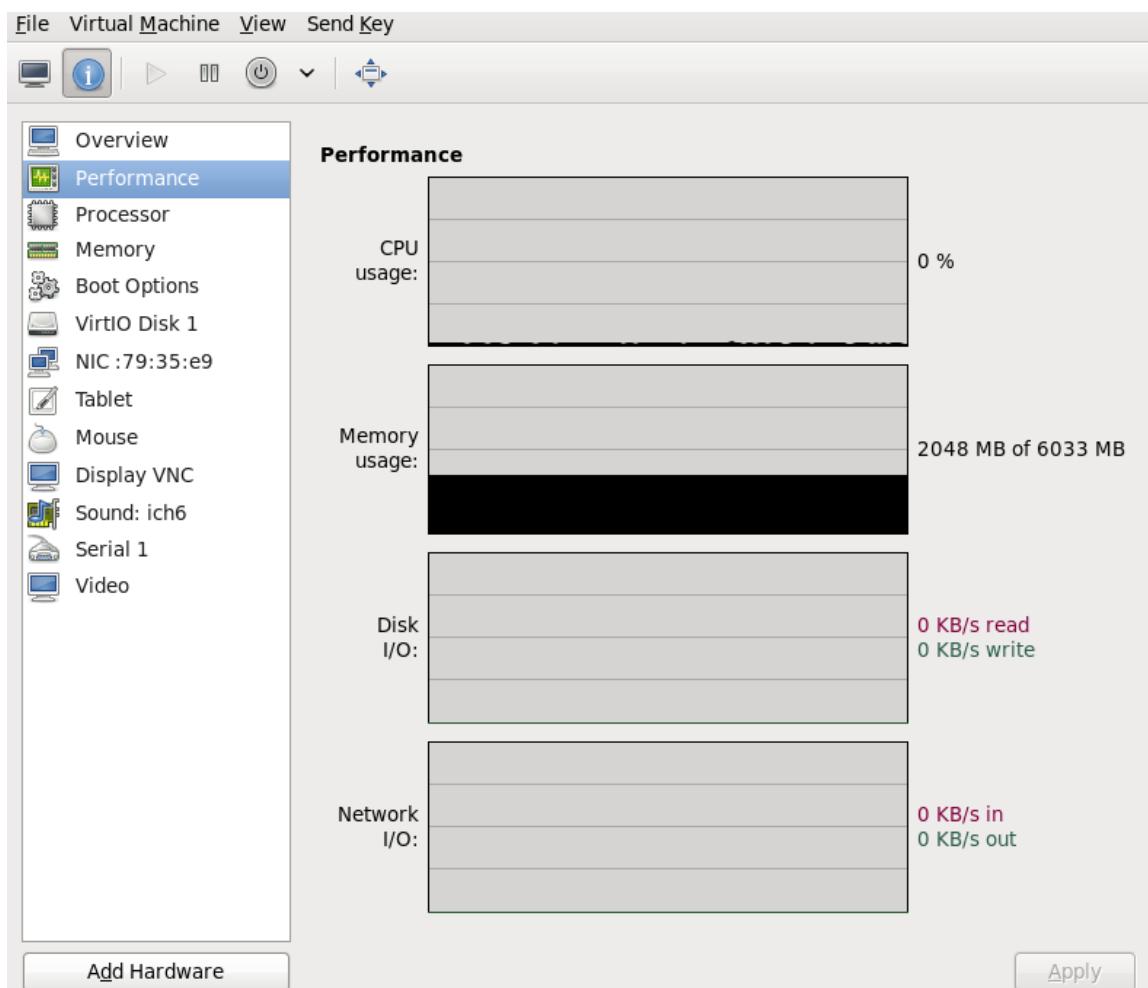


Figure 15.11. Displaying guest performance details

4. Select **Processor** from the navigation pane on the left hand side. The **Processor** view allows you to view or change the current processor allocation.

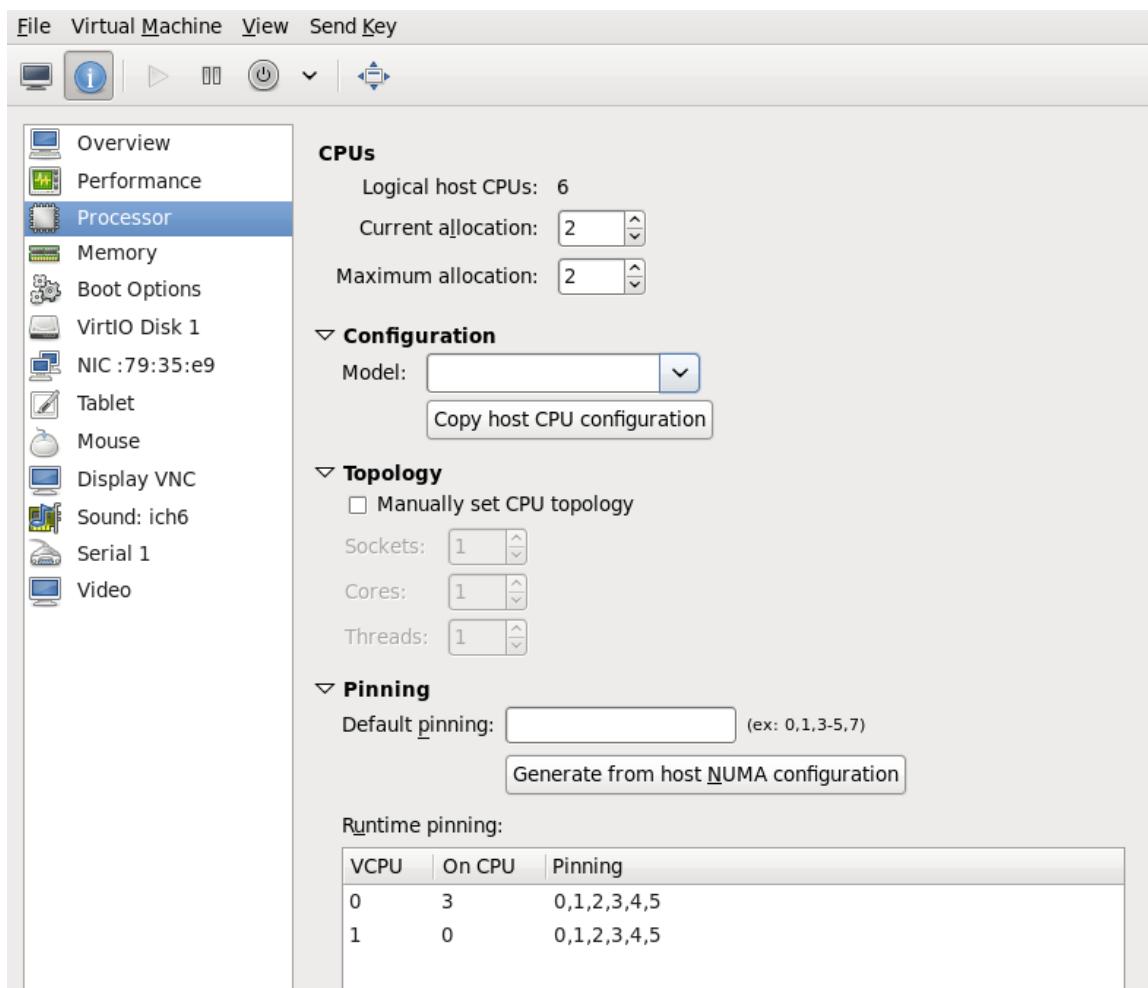


Figure 15.12. Processor allocation panel

5. Select **Memory** from the navigation pane on the left hand side. The **Memory** view allows you to view or change the current memory allocation.

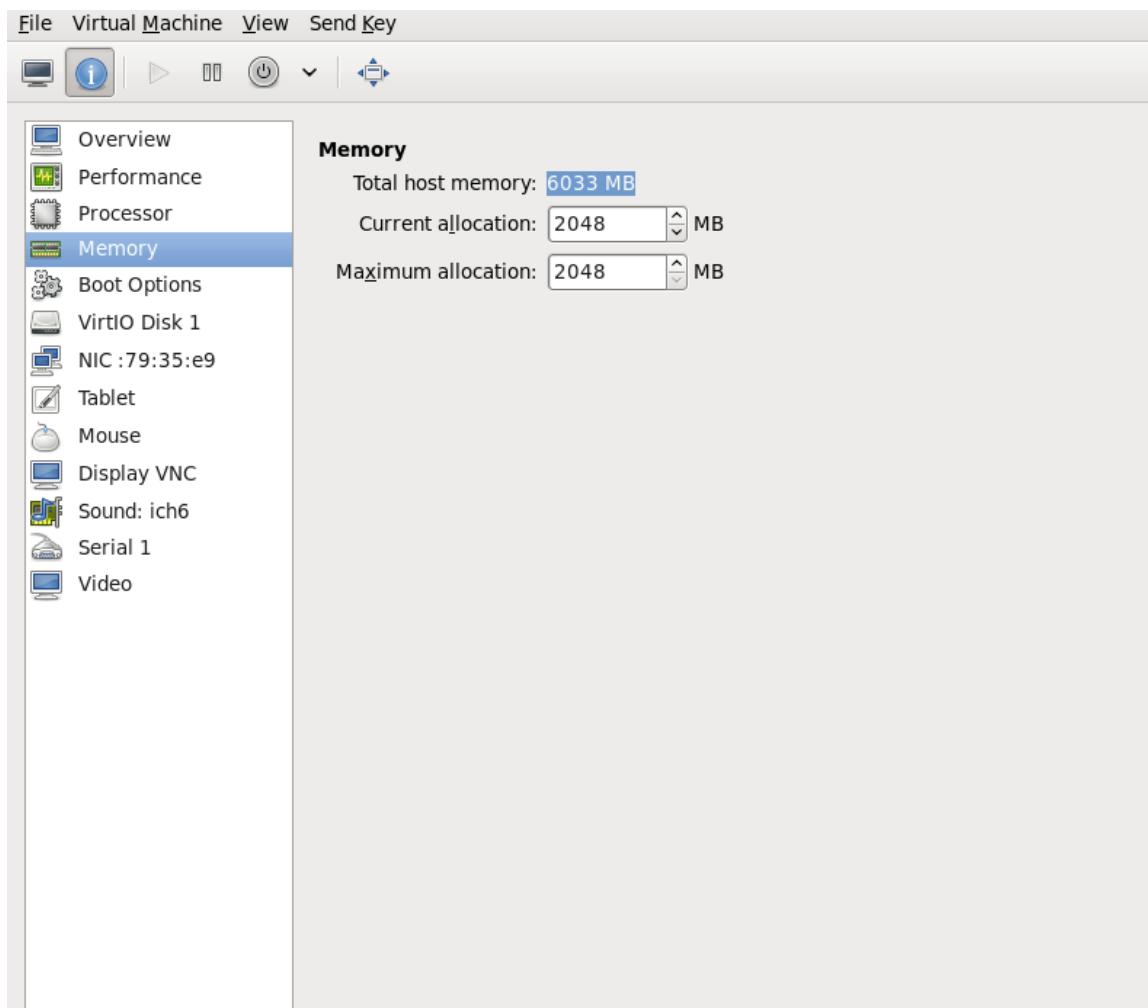


Figure 15.13. Displaying memory allocation

6. Each virtual disk attached to the virtual machine is displayed in the navigation pane. Click on a virtual disk to modify or remove it.

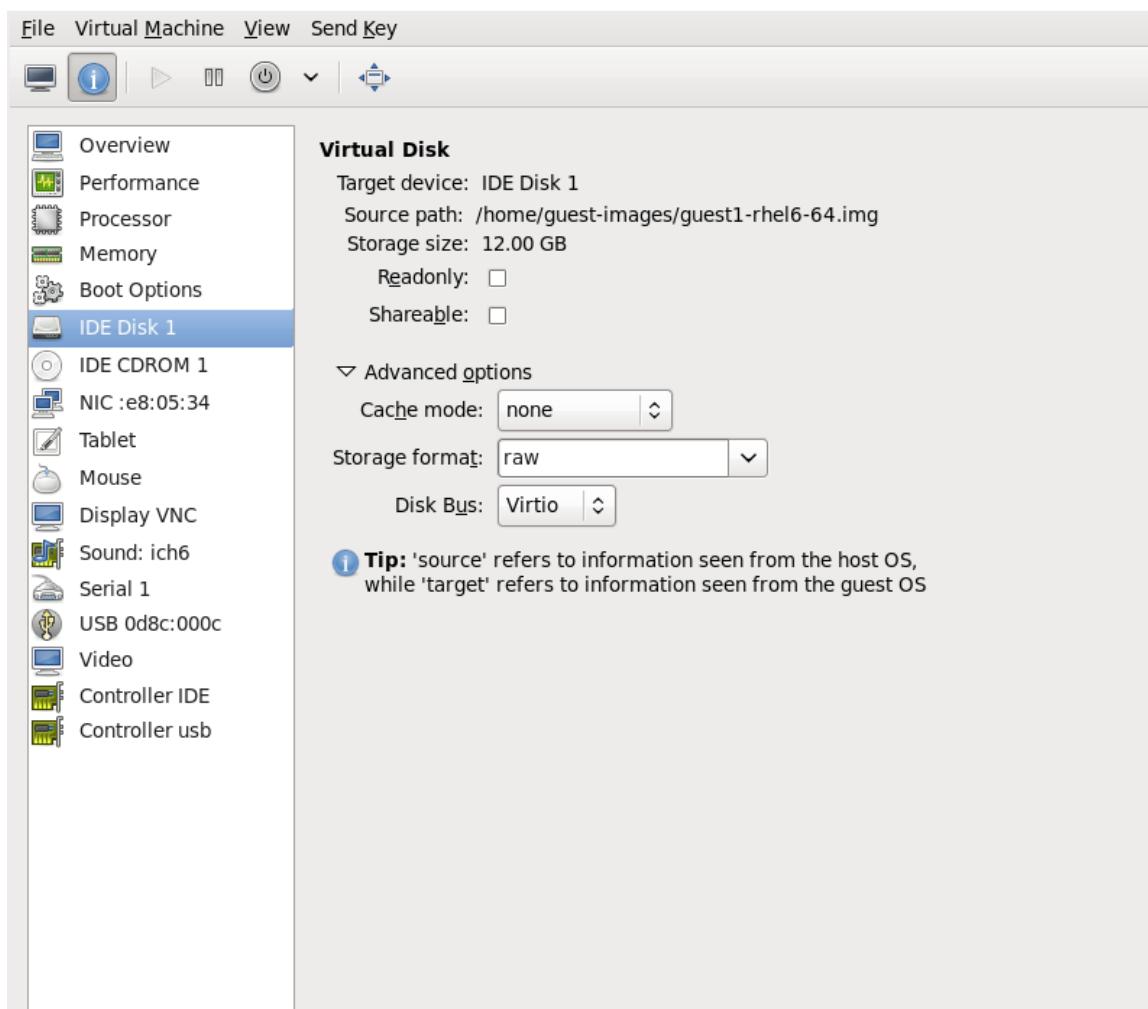


Figure 15.14. Displaying disk configuration

7. Each virtual network interface attached to the virtual machine is displayed in the navigation pane. Click on a virtual network interface to modify or remove it.

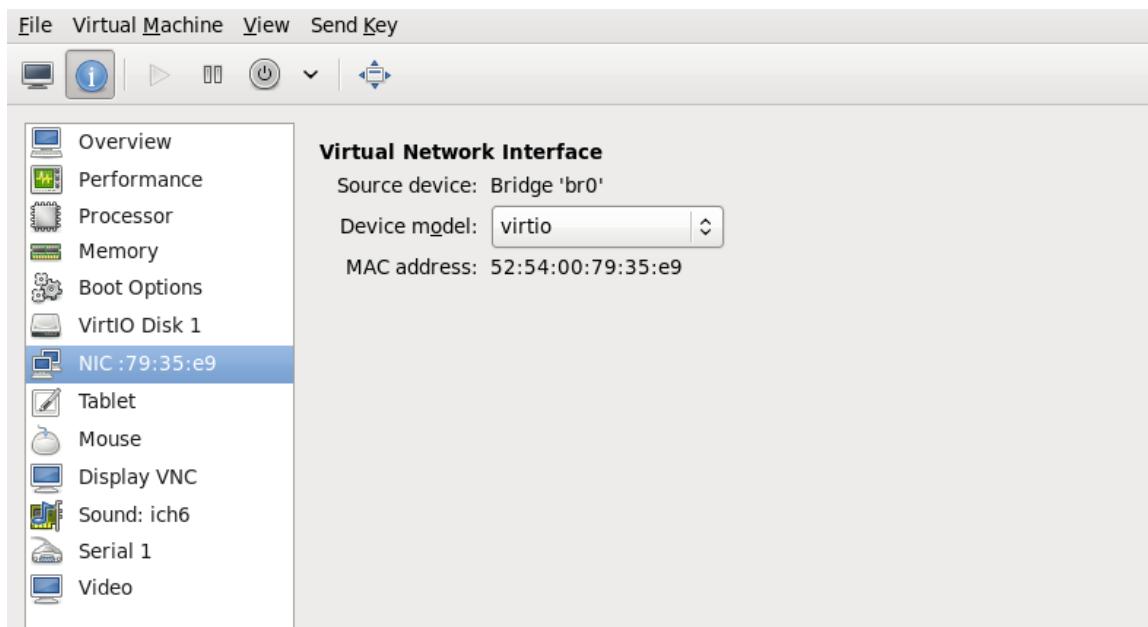


Figure 15.15. Displaying network configuration

15.7. Performance monitoring

Performance monitoring preferences can be modified with **virt-manager**'s preferences window.

To configure performance monitoring:

- From the **Edit** menu, select **Preferences**.

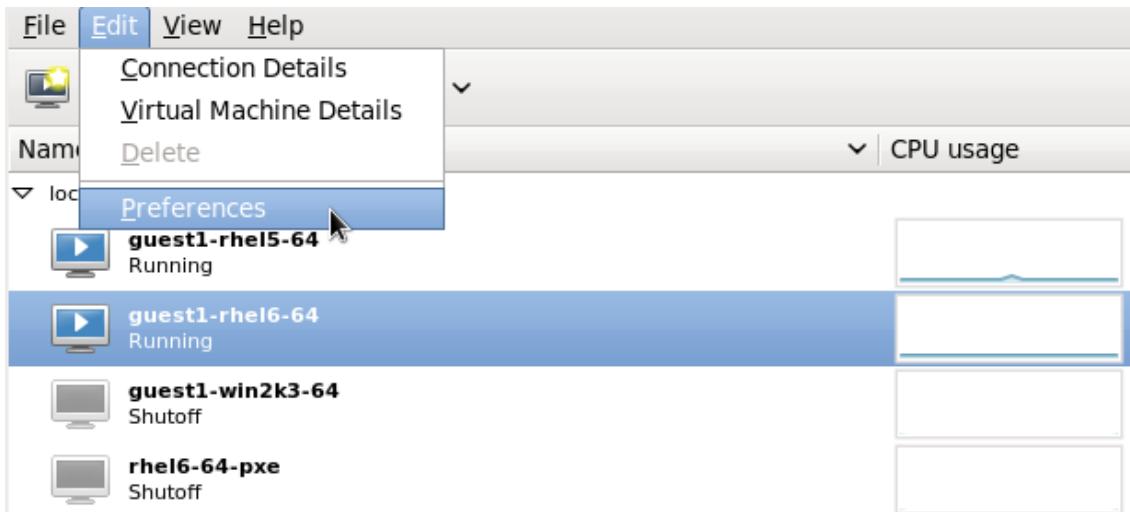


Figure 15.16. Modifying guest preferences

The **Preferences** window appears.

- From the **Stats** tab specify the time in seconds or stats polling options.

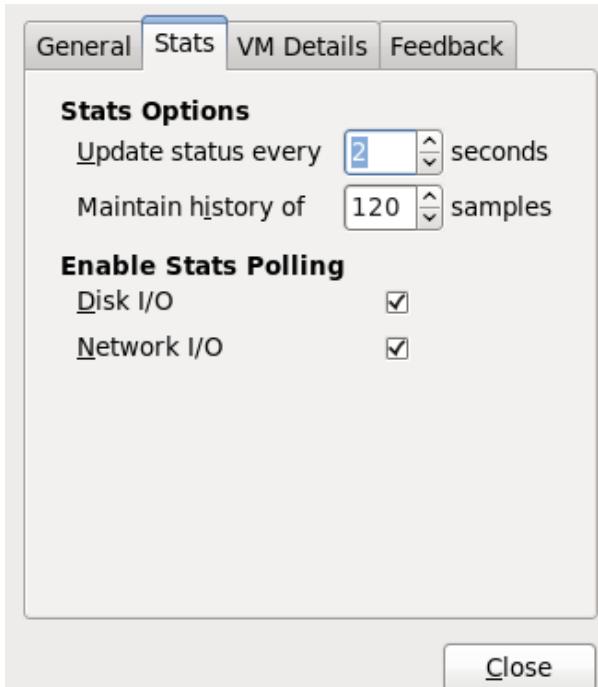


Figure 15.17. Configuring performance monitoring

15.8. Displaying CPU usage for guests

To view the CPU usage for all guests on your system:

- From the **View** menu, select **Graph**, then the **Guest CPU Usage** check box.

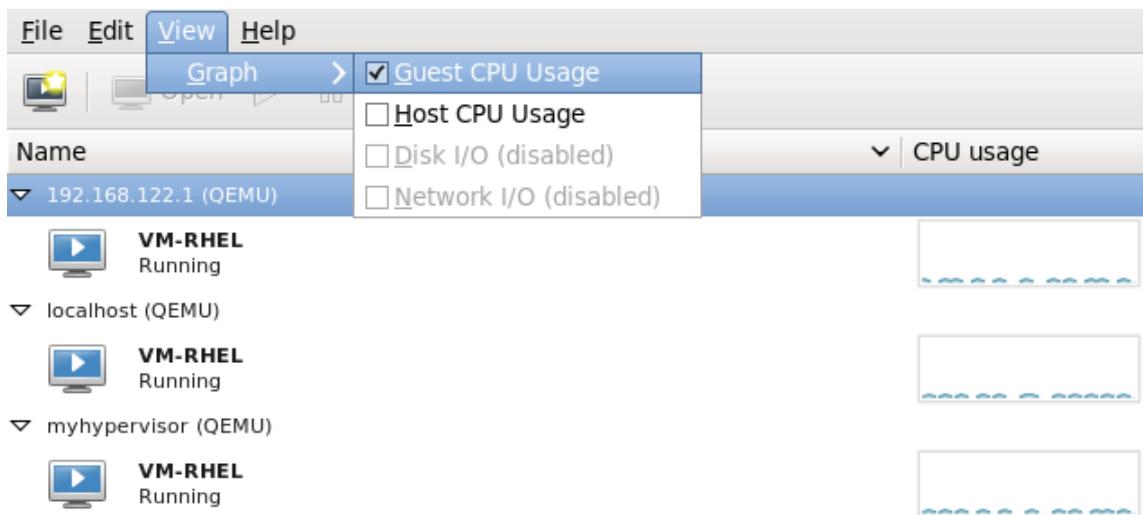


Figure 15.18. Enabling guest CPU usage statistics graphing

- The Virtual Machine Manager shows a graph of CPU usage for all virtual machines on your system.

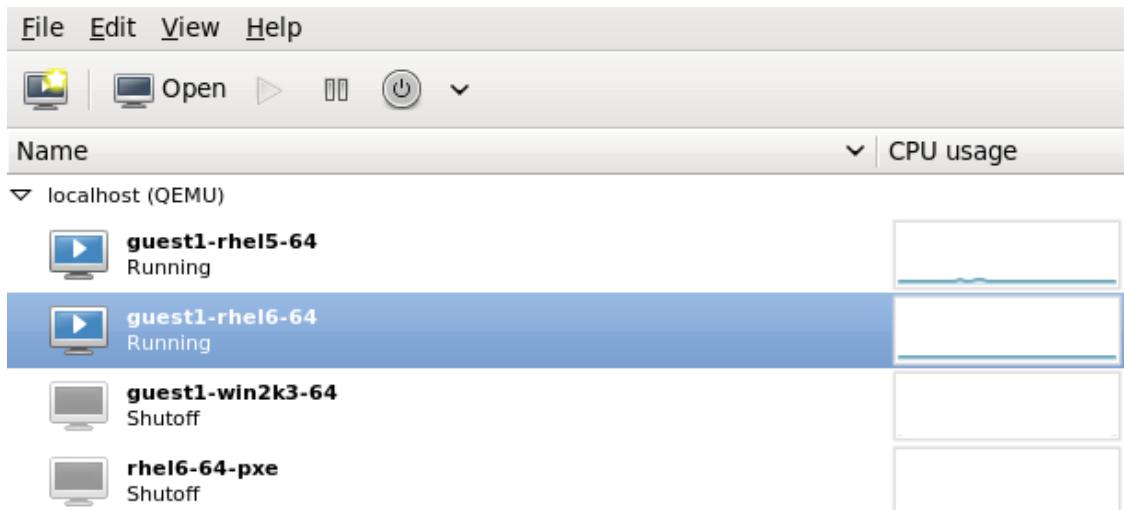


Figure 15.19. Guest CPU usage graph

15.9. Displaying CPU usage for hosts

To view the CPU usage for all hosts on your system:

- From the **View** menu, select **Graph**, then the **Host CPU Usage** check box.



Figure 15.20. Enabling host CPU usage statistics graphing

2. The Virtual Machine Manager shows a graph of host CPU usage on your system.



Figure 15.21. Host CPU usage graph

15.10. Displaying Disk I/O

To view the disk I/O for all virtual machines on your system:

1. Make sure that the Disk I/O statistics collection is enabled. To do this, from the **Edit** menu, select **Preferences** and click the **Stats** tab.
2. Select the **Disk I/O** checkbox.

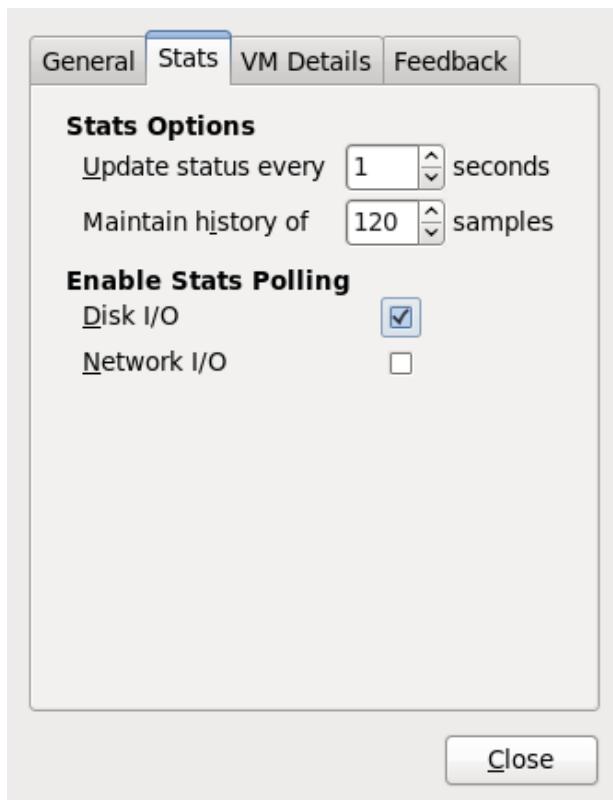


Figure 15.22. Enabling Disk I/O

3. To enable the Disk I/O display, from the **View** menu, select **Graph**, then the **Disk I/O** check box.

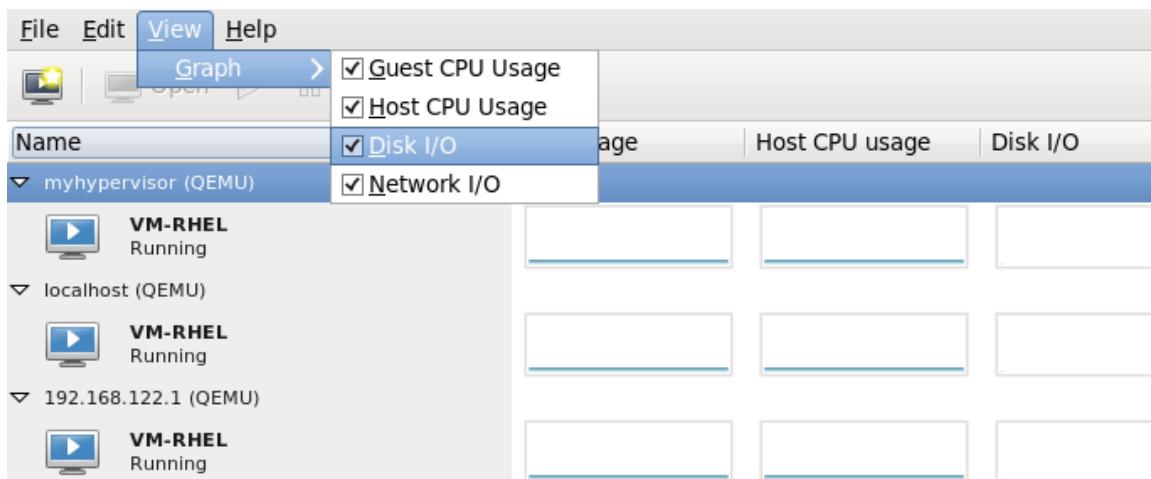


Figure 15.23. Selecting Disk I/O

4. The Virtual Machine Manager shows a graph of Disk I/O for all virtual machines on your system.

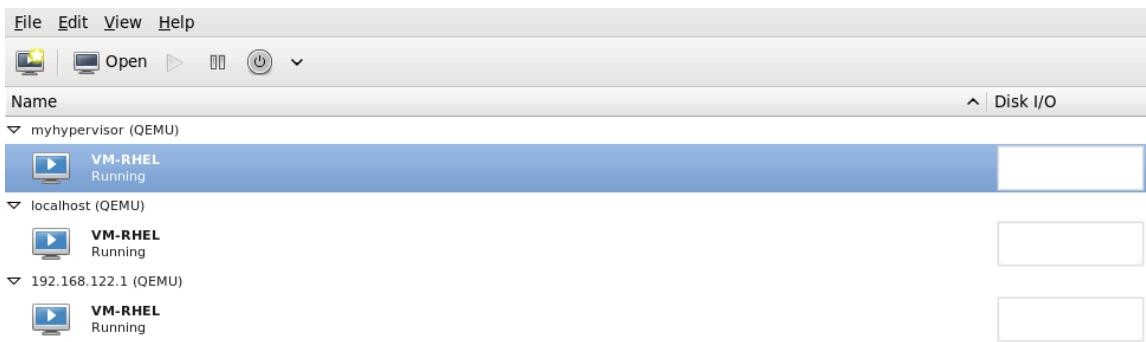


Figure 15.24. Displaying Disk I/O

15.11. Displaying Network I/O

To view the network I/O for all virtual machines on your system:

1. Make sure that the Network I/O statistics collection is enabled. To do this, from the **Edit** menu, select **Preferences** and click the **Stats** tab.
2. Select the **Network I/O** checkbox.

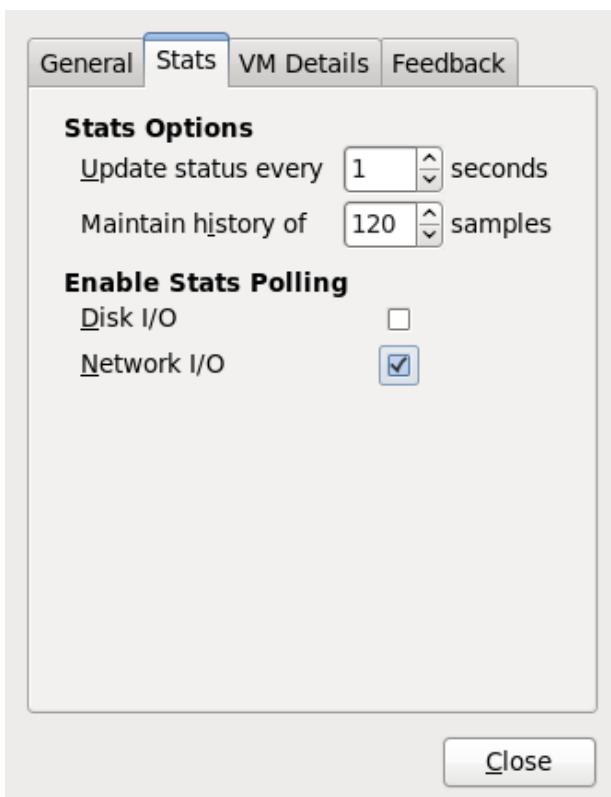
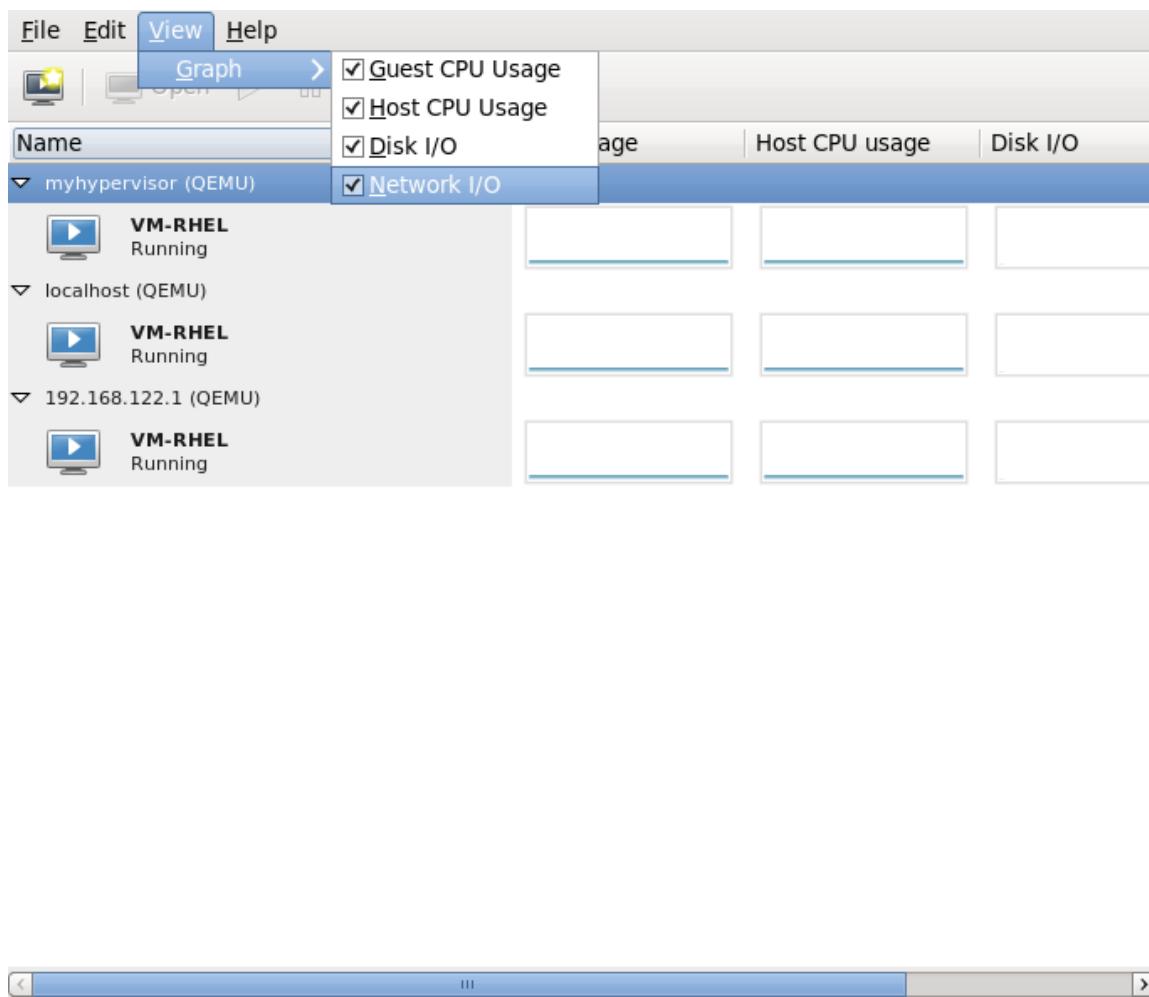


Figure 15.25. Enabling Network I/O

3. To display the Network I/O statistics, from the **View** menu, select **Graph**, then the **Network I/O** check box.

**Figure 15.26. Selecting Network I/O**

4. The Virtual Machine Manager shows a graph of Network I/O for all virtual machines on your system.

**Figure 15.27. Displaying Network I/O**

Chapter 16. Guest virtual machine disk access with offline tools

16.1. Introduction

Red Hat Enterprise Linux 6 comes with tools to access, edit and create host physical machine disks or other disk images. There are several uses for these tools, including:

- ▶ Viewing or downloading files located on a host physical machine disk.
- ▶ Editing or uploading files onto a host physical machine disk.
- ▶ Reading or writing host physical machine configuration.
- ▶ Reading or writing the Windows Registry in Windows host physical machines.
- ▶ Preparing new disk images containing files, directories, file systems, partitions, logical volumes and other options.
- ▶ Rescuing and repairing host physical machines that fail to boot or those that need boot configuration changes.
- ▶ Monitoring disk usage of host physical machines.
- ▶ Auditing compliance of host physical machines, for example to organizational security standards.
- ▶ Deploying host physical machines by cloning and modifying templates.
- ▶ Reading CD and DVD ISO and floppy disk images.



Warning

You must **never** use these tools to write to a host physical machine or disk image which is attached to a running virtual machine, not even to open such a disk image in write mode. Doing so will result in disk corruption of the guest virtual machine. The tools try to prevent you from doing this, however do not catch all cases. If there is any suspicion that a guest virtual machine might be running, it is strongly recommended that the tools not be used, or at least **always** use the tools in read-only mode.

16.2. Terminology

This section explains the terms used throughout this chapter.

- ▶ **libguestfs (GUEST FileSystem LIBrary)** - the underlying C library that provides the basic functionality for opening disk images, reading and writing files and so on. You can write C programs directly to this API, but it is quite low level.
- ▶ **guestfish (GUEST Filesystem Interactive SHell)** is an interactive shell that you can use from the command line or from shell scripts. It exposes all of the functionality of the libguestfs API.
- ▶ Various virt tools are built on top of libguestfs, and these provide a way to perform specific single tasks from the command line. Tools include **virt-df**, **virt-rescue**, **virt-resize** and **virt-edit**.
- ▶ **hiveX** and **Augeas** are libraries for editing the Windows Registry and Linux configuration files respectively. Although these are separate from libguestfs, much of the value of libguestfs comes from the combination of these tools.
- ▶ **guestmount** is an interface between libguestfs and FUSE. It is primarily used to mount file systems from disk images on your host physical machine. This functionality is not necessary, but can be useful.

16.3. Installation

To install libguestfs, guestfish, the libguestfs tools, guestmount and support for Windows guest virtual machines, subscribe to the RHEL V2WIN channel, go to the [Red Hat Website](#) and run the following

command:

```
# yum install libguestfs guestfish libguestfs-tools libguestfs-mount libguestfs-winsupport
```

To install every libguestfs-related package including the language bindings, run the following command:

```
# yum install '*guestf*'
```

16.4. The guestfish shell

guestfish is an interactive shell that you can use from the command line or from shell scripts to access guest virtual machine file systems. All of the functionality of the libguestfs API is available from the shell.

To begin viewing or editing a virtual machine disk image, run the following command, substituting the path to your desired disk image:

```
guestfish --ro -a /path/to/disk/image
```

--ro means that the disk image is opened read-only. This mode is always safe but does not allow write access. Only omit this option when you are **certain** that the guest virtual machine is not running, or the disk image is not attached to a live guest virtual machine. It is not possible to use **libguestfs** to edit a live guest virtual machine, and attempting to will result in irreversible disk corruption.

/path/to/disk/image is the path to the disk. This can be a file, a host physical machine logical volume (such as /dev/VG/LV), a host physical machine device (/dev/cdrom) or a SAN LUN (/dev/sdf3).



Note

libguestfs and guestfish do not require root privileges. You only need to run them as root if the disk image being accessed needs root to read and/or write.

When you start guestfish interactively, it will display this prompt:

```
guestfish --ro -a /path/to/disk/image
```

Welcome to guestfish, the libguestfs filesystem interactive shell for editing virtual machine filesystems.

```
Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell
```

```
><fs>
```

At the prompt, type **run** to initiate the library and attach the disk image. This can take up to 30 seconds the first time it is done. Subsequent starts will complete much faster.



Note

libguestfs will use hardware virtualization acceleration such as KVM (if available) to speed up this process.

Once the **run** command has been entered, other commands can be used, as the following section demonstrates.

16.4.1. Viewing file systems with guestfish

16.4.1.1. Manual listing and viewing

The **list-filesystems** command will list file systems found by libguestfs. This output shows a Red Hat Enterprise Linux 4 disk image:

```
><fs> run
><fs> list-filesystems
/dev/vda1: ext3
/dev/VolGroup00/LogVol00: ext3
/dev/VolGroup00/LogVol01: swap
```

This output shows a Windows disk image:

```
><fs> run
><fs> list-filesystems
/dev/vda1: ntfs
/dev/vda2: ntfs
```

Other useful commands are **list-devices**, **list-partitions**, **lvs**, **pvs**, **vfs-type** and **file**. You can get more information and help on any command by typing **help command**, as shown in the following output:

```
><fs> help vfs-type
NAME
  vfs-type - get the Linux VFS type corresponding to a mounted device

SYNOPSIS
  vfs-type device

DESCRIPTION
  This command gets the filesystem type corresponding to the filesystem on
  "device".

  For most filesystems, the result is the name of the Linux VFS module
  which would be used to mount this filesystem if you mounted it without
  specifying the filesystem type. For example a string such as "ext3" or
  "ntfs".
```

To view the actual contents of a file system, it must first be mounted. This example uses one of the Windows partitions shown in the previous output (**/dev/vda2**), which in this case is known to correspond to the **C:** drive:

```
><fs> mount-ro /dev/vda2 /
><fs> ll /
total 1834753
drwxrwxrwx  1 root root      4096 Nov  1 11:40 .
drwxr-xr-x 21 root root      4096 Nov 16 21:45 ..
lrwxrwxrwx  2 root root       60 Jul 14  2009 Documents and Settings
drwxrwxrwx  1 root root      4096 Nov 15 18:00 Program Files
drwxrwxrwx  1 root root      4096 Sep 19 10:34 Users
drwxrwxrwx  1 root root     16384 Sep 19 10:34 Windows
```

You can use guestfish commands such as **ls**, **ll**, **cat**, **more**, **download** and **tar-out** to view and download files and directories.



Note

There is no concept of a current working directory in this shell. Unlike ordinary shells, you cannot for example use the **cd** command to change directories. All paths must be fully qualified starting at the top with a forward slash (/) character. Use the **Tab** key to complete paths.

To exit from the guestfish shell, type **exit** or enter **Ctrl+d**.

16.4.1.2. Via guestfish inspection

Instead of listing and mounting file systems by hand, it is possible to let guestfish itself inspect the image and mount the file systems as they would be in the guest virtual machine. To do this, add the **-i** option on the command line:

```
guestfish --ro -a /path/to/disk/image -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
editing virtual machine filesystems.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

Operating system: Red Hat Enterprise Linux AS release 4 (Nahant Update 8)
/dev/VolGroup00/LogVol00 mounted on /
/dev/vda1 mounted on /boot

><fs> ll /
total 210
drwxr-xr-x. 24 root root 4096 Oct 28 09:09 .
drwxr-xr-x. 21 root root 4096 Nov 17 15:10 ..
drwxr-xr-x. 2 root root 4096 Oct 27 22:37 bin
drwxr-xr-x. 4 root root 1024 Oct 27 21:52 boot
drwxr-xr-x. 4 root root 4096 Oct 27 21:21 dev
drwxr-xr-x. 86 root root 12288 Oct 28 09:09 etc
[etc]
```

Because guestfish needs to start up the libguestfs back end in order to perform the inspection and mounting, the **run** command is not necessary when using the **-i** option. The **-i** option works for many common Linux and Windows guest virtual machines.

16.4.1.3. Accessing a guest virtual machine by name

A guest virtual machine can be accessed from the command line when you specify its name as known to libvirt (in other words, as it appears in **virsh list --all**). Use the **-d** option to access a guest virtual machine by its name, with or without the **-i** option:

```
guestfish --ro -d GuestName -i
```

16.4.2. Modifying files with guestfish

To modify files, create directories or make other changes to a guest virtual machine, first heed the warning at the beginning of this section: your guest virtual machine must be shut down. Editing or changing a running disk with guestfish **will** result in disk corruption. This section gives an example of editing the **/boot/grub/grub.conf** file. When you are sure the guest virtual machine is shut down you can omit the **--ro** flag in order to get write access via a command such as:

```
guestfish -d RHEL3 -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
editing virtual machine filesystems.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

Operating system: Red Hat Enterprise Linux AS release 3 (Taroon Update 9)
/dev/vda2 mounted on /
/dev/vda1 mounted on /boot

><fs> edit /boot/grub/grub.conf
```

Commands to edit files include **edit**, **vi** and **emacs**. Many commands also exist for creating files and directories, such as **write**, **mkdir**, **upload** and **tar-in**.

16.4.3. Other actions with guestfish

You can also format file systems, create partitions, create and resize LVM logical volumes and much more, with commands such as **mkfs**, **part-add**, **lvresize**, **lvcreate**, **vgcreate** and **pvccreate**.

16.4.4. Shell scripting with guestfish

Once you are familiar with using guestfish interactively, according to your needs, writing shell scripts with it may be useful. The following is a simple shell script to add a new MOTD (message of the day) to a guest:

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$guestname" -i <<'EOF'
  write /etc/motd "Welcome to Acme Incorporated."
  chmod 0644 /etc/motd
EOF
```

16.4.5. Augeas and libguestfs scripting

Combining libguestfs with Augeas can help when writing scripts to manipulate Linux guest virtual machine configuration. For example, the following script uses Augeas to parse the keyboard configuration of a guest virtual machine, and to print out the layout. Note that this example only works with guest virtual machines running Red Hat Enterprise Linux:

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$1" -i --ro <<'EOF'
  aug-init / 0
  aug-get /files/etc/sysconfig/keyboard/LAYOUT
EOF
```

Augeas can also be used to modify configuration files. You can modify the above script to **change** the keyboard layout:

```
#!/bin/bash -
set -e
guestname="$1"

guestfish -d "$1" -i <<'EOF'
aug-init / 0
aug-set /files/etc/sysconfig/keyboard/LAYOUT '"gb"'
aug-save
EOF
```

Note the three changes between the two scripts:

1. The **--ro** option has been removed in the second example, giving the ability to write to the guest virtual machine.
2. The **aug-get** command has been changed to **aug-set** to modify the value instead of fetching it. The new value will be "**gb**" (including the quotes).
3. The **aug-save** command is used here so Augeas will write the changes out to disk.

Note

More information about Augeas can be found on the website <http://augeas.net>.

guestfish can do much more than we can cover in this introductory document. For example, creating disk images from scratch:

```
guestfish -N fs
```

Or copying out whole directories from a disk image:

```
><fs> copy-out /home /tmp/home
```

For more information see the man page `guestfish(1)`.

16.5. Other commands

This section describes tools that are simpler equivalents to using guestfish to view and edit guest virtual machine disk images.

- ▶ **virt-cat** is similar to the guestfish **download** command. It downloads and displays a single file to the guest virtual machine. For example:

```
# virt-cat RHEL3 /etc/ntp.conf | grep ^server
server      127.127.1.0      # local clock
```

- ▶ **virt-edit** is similar to the guestfish **edit** command. It can be used to interactively edit a single file within a guest virtual machine. For example, you may need to edit the **grub.conf** file in a Linux-based guest virtual machine that will not boot:

```
# virt-edit LinuxGuest /boot/grub/grub.conf
```

virt-edit has another mode where it can be used to make simple non-interactive changes to a single file. For this, the **-e** option is used. This command, for example, changes the root password in a Linux guest virtual machine to having no password:

```
# virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?/:/root::/'
```

- ▶ **virt-ls** is similar to the guestfish **ls**, **ll** and **find** commands. It is used to list a directory or directories (recursively). For example, the following command would recursively list files and directories under /home in a Linux guest virtual machine:

```
# virt-ls -R LinuxGuest /home/ | less
```

16.6. virt-rescue: The rescue shell

16.6.1. Introduction

This section describes **virt-rescue**, which can be considered analogous to a rescue CD for virtual machines. It boots a guest virtual machine into a rescue shell so that maintenance can be performed to correct errors and the guest virtual machine can be repaired.

There is some overlap between **virt-rescue** and **guestfish**. It is important to distinguish their differing uses. **virt-rescue** is for making interactive, ad-hoc changes using ordinary Linux file system tools. It is particularly suited to rescuing a guest virtual machine that has failed. **virt-rescue** cannot be scripted.

In contrast, **guestfish** is particularly useful for making scripted, structured changes through a formal set of commands (the libguestfs API), although it can also be used interactively.

16.6.2. Running **virt-rescue**

Before you use **virt-rescue** on a guest virtual machine, make sure the guest virtual machine is not running, otherwise disk corruption will occur. When you are sure the guest virtual machine is not live, enter:

```
virt-rescue GuestName
```

(where **GuestName** is the guest name as known to libvirt), or:

```
virt-rescue /path/to/disk/image
```

(where the path can be any file, any logical volume, LUN, or so on) containing a guest virtual machine disk.

You will first see output scroll past, as **virt-rescue** boots the rescue VM. In the end you will see:

```
Welcome to virt-rescue, the libguestfs rescue shell.
```

Note: The contents of / are the rescue appliance.

You have to mount the guest virtual machine's partitions under /sysroot before you can examine them.

```
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
><rescue>
```

The shell prompt here is an ordinary bash shell, and a reduced set of ordinary Red Hat Enterprise Linux commands is available. For example, you can enter:

```
><rescue> fdisk -l /dev/vda
```

The previous command will list disk partitions. To mount a file system, it is suggested that you mount it under **/sysroot**, which is an empty directory in the rescue machine for the user to mount anything you like. Note that the files under / are files from the rescue VM itself:

```
><rescue> mount /dev/vda1 /sysroot/
EXT4-fs (vda1): mounted filesystem with ordered data mode. Opts: (null)
><rescue> ls -l /sysroot/grub/
total 324
-rw-r--r--. 1 root root    63 Sep 16 18:14 device.map
-rw-r--r--. 1 root root 13200 Sep 16 18:14 e2fs_stage1_5
-rw-r--r--. 1 root root 12512 Sep 16 18:14 fat_stage1_5
-rw-r--r--. 1 root root 11744 Sep 16 18:14 ffs_stage1_5
-rw-----. 1 root root   1503 Oct 15 11:19 grub.conf
[...]
```

When you are finished rescuing the guest virtual machine, exit the shell by entering **exit** or **Ctrl+d**.

virt-rescue has many command line options. The options most often used are:

- ▶ **--ro**: Operate in read-only mode on the guest virtual machine. No changes will be saved. You can use this to experiment with the guest virtual machine. As soon as you exit from the shell, all of your changes are discarded.
- ▶ **--network**: Enable network access from the rescue shell. Use this if you need to, for example, download RPM or other files into the guest virtual machine.

16.7. virt-df: Monitoring disk usage

16.7.1. Introduction

This section describes **virt-df**, which displays file system usage from a disk image or a guest virtual machine. It is similar to the Linux **df** command, but for virtual machines.

16.7.2. Running virt-df

To display file system usage for all file systems found in a disk image, enter the following:

```
# virt-df /dev/vg_guests/RHEL6
Filesystem           1K-blocks      Used   Available  Use%
RHEL6:/dev/sda1        101086     10233     85634    11%
RHEL6:/dev/VolGroup00/LogVol00 7127864  2272744   4493036   32%
```

(Where **/dev/vg_guests/RHEL6** is a Red Hat Enterprise Linux 6 guest virtual machine disk image. The path in this case is the host physical machine logical volume where this disk image is located.)

You can also use **virt-df** on its own to list information about all of your guest virtual machines (ie. those known to libvirt). The **virt-df** command recognizes some of the same options as the standard **df** such as **-h** (human-readable) and **-i** (show inodes instead of blocks).

virt-df also works on Windows guest virtual machines:

```
# virt-df -h
Filesystem           Size   Used  Available  Use%
F14x64:/dev/sda1      484.2M  66.3M    392.9M  14%
F14x64:/dev/vg_f14x64/lv_root  7.4G   3.0G     4.4G  41%
RHEL6brewx64:/dev/sda1  484.2M  52.6M    406.6M  11%
RHEL6brewx64:/dev/vg_rhel6brewx64/lv_root
                           13.3G   3.4G     9.2G  26%
Win7x32:/dev/sda1      100.0M  24.1M    75.9M  25%
Win7x32:/dev/sda2      19.9G   7.4G    12.5G  38%
```



Note

You can use **virt-df** safely on live guest virtual machines, since it only needs read-only access. However, you should not expect the numbers to be precisely the same as those from a **df** command running inside the guest virtual machine. This is because what is on disk will be slightly out of sync with the state of the live guest virtual machine. Nevertheless it should be a good enough approximation for analysis and monitoring purposes.

virt-df is designed to allow you to integrate the statistics into monitoring tools, databases and so on. This allows system administrators to generate reports on trends in disk usage, and alerts if a guest virtual machine is about to run out of disk space. To do this you should use the **--csv** option to generate machine-readable Comma-Separated-Values (CSV) output. CSV output is readable by most databases, spreadsheet software and a variety of other tools and programming languages. The raw CSV looks like the following:

```
# virt-df --csv WindowsGuest
Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
Win7x32,/dev/sda1,102396,24712,77684,24.1%
Win7x32,/dev/sda2,20866940,7786652,13080288,37.3%
```

For resources and ideas on how to process this output to produce trends and alerts, refer to the following URL: <http://virt-tools.org/learning/advanced-virt-df/>.

16.8. virt-resize: resizing guest virtual machines offline

16.8.1. Introduction

This section describes **virt-resize**, a tool for expanding or shrinking guest virtual machines. It only works for guest virtual machines which are offline (shut down). It works by copying the guest virtual machine image and leaving the original disk image untouched. This is ideal because you can use the original image as a backup, however there is a trade-off as you need twice the amount of disk space.

16.8.2. Expanding a disk image

This section demonstrates a simple case of expanding a disk image:

1. Locate the disk image to be resized. You can use the command **virsh dumpxml GuestName** for a libvirt guest virtual machine.
2. Decide on how you wish to expand the guest virtual machine. Run **virt-df -h** and **virt-list-partitions -lh** on the guest virtual machine disk, as shown in the following output:

```
# virt-df -h /dev/vg_guests/RHEL6
Filesystem          Size   Used  Available  Use%
RHEL6:/dev/sda1    98.7M  10.0M   83.6M   11%
RHEL6:/dev/VolGroup00/LogVol00  6.8G   2.2G   4.3G   32%

# virt-list-partitions -lh /dev/vg_guests/RHEL6
/dev/sda1 ext3 101.9M
/dev/sda2 pv 7.9G
```

This example will demonstrate how to:

- ▶ Increase the size of the first (boot) partition, from approximately 100MB to 500MB.
- ▶ Increase the total disk size from 8GB to 16GB.
- ▶ Expand the second partition to fill the remaining space.
- ▶ Expand **/dev/VolGroup00/LogVol00** to fill the new space in the second partition.

1. Make sure the guest virtual machine is shut down.
2. Rename the original disk as the backup. How you do this depends on the host physical machine storage environment for the original disk. If it is stored as a file, use the **mv** command. For logical volumes (as demonstrated in this example), use **lvrename**:

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. Create the new disk. The requirements in this example are to expand the total disk size up to 16GB. Since logical volumes are used here, the following command is used:

```
# lvcreate -L 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. The requirements from step 2 are expressed by this command:

```
# virt-resize \
    /dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 \
    --resize /dev/sda1=500M \
    --expand /dev/sda2 \
    --LV-expand /dev/VolGroup00/LogVol00
```

The first two arguments are the input disk and output disk. **--resize /dev/sda1=500M** resizes the first partition up to 500MB. **--expand /dev/sda2** expands the second partition to fill all remaining space. **--LV-expand /dev/VolGroup00/LogVol00** expands the guest virtual machine logical volume to fill the extra space in the second partition.

virt-resize describes what it is doing in the output:

```
Summary of changes:
/dev/sda1: partition will be resized from 101.9M to 500.0M
/dev/sda1: content will be expanded using the 'resize2fs' method
/dev/sda2: partition will be resized from 7.9G to 15.5G
/dev/sda2: content will be expanded using the 'pvresize' method
/dev/VolGroup00/LogVol00: LV will be expanded to maximum size
/dev/VolGroup00/LogVol00: content will be expanded using the 'resize2fs'
method
Copying /dev/sda1 ...
[#####
Copying /dev/sda2 ...
[#####
Expanding /dev/sda1 using the 'resize2fs' method
Expanding /dev/sda2 using the 'pvresize' method
Expanding /dev/VolGroup00/LogVol00 using the 'resize2fs' method
```

5. Try to boot the virtual machine. If it works (and after testing it thoroughly) you can delete the backup disk. If it fails, shut down the virtual machine, delete the new disk, and rename the backup disk back to its original name.
6. Use **virt-df** and/or **virt-list-partitions** to show the new size:

```
# virt-df -h /dev/vg_pin/RHEL6
Filesystem           Size   Used  Available Use%
RHEL6:/dev/sda1      484.4M 10.8M   448.6M   3%
RHEL6:/dev/VolGroup00/LogVol00 14.3G   2.2G   11.4G  16%
```

Resizing guest virtual machines is not an exact science. If **virt-resize** fails, there are a number of tips that you can review and attempt in the `virt-resize(1)` man page. For some older Red Hat Enterprise Linux guest virtual machines, you may need to pay particular attention to the tip regarding GRUB.

16.9. virt-inspector: inspecting guest virtual machines

16.9.1. Introduction

virt-inspector is a tool for inspecting a disk image to find out what operating system it contains.



Note

Red Hat Enterprise Linux 6.2 ships with two variations of this program: **virt-inspector** is the original program as found in Red Hat Enterprise Linux 6.0 and is now deprecated upstream. **virt-inspector2** is the same as the new upstream **virt-inspector** program.

16.9.2. Installation

To install virt-inspector and the documentation, enter the following command:

```
# yum install libguestfs-tools libguestfs-devel
```

To process Windows guest virtual machines you must also install *libguestfs-winsupport*. Refer to [Section 16.10.2, “Installation”](#) for details. The documentation, including example XML output and a Relax-NG schema for the output, will be installed in **/usr/share/doc/libguestfs-devel-*/** where “*” is replaced by the version number of libguestfs.

16.9.3. Running virt-inspector

You can run **virt-inspector** against any disk image or libvirt guest virtual machine as shown in the following example:

```
virt-inspector --xml disk.img > report.xml
```

Or as shown here:

```
virt-inspector --xml GuestName > report.xml
```

The result will be an XML report (**report.xml**). The main components of the XML file are a top-level `<operatingsystems>` element containing usually a single `<operatingsystem>` element, similar to the following:

```

<operatingsystems>
  <operatingsystem>

    <!-- the type of operating system and Linux distribution -->
    <name>linux</name>
    <distro>rhel</distro>
    <!-- the name, version and architecture -->
    <product_name>Red Hat Enterprise Linux Server release 6.4 </product_name>
    <major_version>6</major_version>
    <minor_version>4</minor_version>
    <package_format>rpm</package_format>
    <package_management>yum</package_management>
    <root>/dev/VolGroup/lv_root</root>
    <!-- how the filesystems would be mounted when live -->
    <mountpoints>
      <mountpoint dev="/dev/VolGroup/lv_root"/></mountpoint>
      <mountpoint dev="/dev/sda1">/boot</mountpoint>
      <mountpoint dev="/dev/VolGroup/lv_swap">swap</mountpoint>
    </mountpoints>

    <!-- filesystems-->
    <filesystem dev="/dev/VolGroup/lv_root">
      <label></label>
      <uuid>b24d9161-5613-4ab8-8649-f27a8a8068d3</uuid>
      <type>ext4</type>
      <content>linux-root</content>
      <spec>/dev/mapper/VolGroup-lv_root</spec>
    </filesystem>
    <filesystem dev="/dev/VolGroup/lv_swap">
      <type>swap</type>
      <spec>/dev/mapper/VolGroup-lv_swap</spec>
    </filesystem>
    <!-- packages installed -->
    <applications>
      <application>
        <name>firefox</name>
        <version>3.5.5</version>
        <release>1.fc12</release>
      </application>
    </applications>

  </operatingsystem>
</operatingsystems>

```

Processing these reports is best done using W3C standard XPath queries. Red Hat Enterprise Linux 6 comes with a command line program (**xpath**) which can be used for simple instances; however, for long-term and advanced usage, you should consider using an XPath library along with your favorite programming language.

As an example, you can list out all file system devices using the following XPath query:

```

virt-inspector --xml GuestName | xpath //filesystem/@dev
Found 3 nodes:
-- NODE --
dev="/dev/sda1"
-- NODE --
dev="/dev/vg_f12x64/lv_root"
-- NODE --
dev="/dev/vg_f12x64/lv_swap"

```

Or list the names of all applications installed by entering:

```
virt-inspector --xml GuestName | xpath //application/name  
[...long list...]
```

16.10. virt-win-reg: Reading and editing the Windows Registry

16.10.1. Introduction

virt-win-reg is a tool that manipulates the Registry in Windows guest virtual machines. It can be used to read out registry keys. You can also use it to make changes to the Registry, but you must **never** try to do this for live/running guest virtual machines, as it will result in disk corruption.

16.10.2. Installation

To use **virt-win-reg** you must run the following:

```
# yum install libguestfs-tools libguestfs-winsupport
```

16.10.3. Using virt-win-reg

To read out Registry keys, specify the name of the guest virtual machine (or its disk image) and the name of the Registry key. You must use single quotes to surround the name of the desired key:

```
# virt-win-reg WindowsGuest \  
'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall' \  
| less
```

The output is in the standard text-based format used by **.REG** files on Windows.

Note

Hex-quoting is used for strings because the format does not properly define a portable encoding method for strings. This is the only way to ensure fidelity when transporting **.REG** files from one machine to another.

You can make hex-quoted strings printable by piping the output of **virt-win-reg** through this simple Perl script:

```
perl -MEncode -pe's?hex\((\d+)\)(\S+)?  
$t=$1;$_= $2;s,\,,g;"str($t):\\"".decode(utf16le=>pack("H*",$_))."\\"?eg'
```

To merge changes into the Windows Registry of an offline guest virtual machine, you must first prepare a **.REG** file. There is a great deal of documentation about doing this available [here](#). When you have prepared a **.REG** file, enter the following:

```
# virt-win-reg --merge WindowsGuest input.reg
```

This will update the registry in the guest virtual machine.

16.11. Using the API from Programming Languages

The libguestfs API can be used directly from the following languages in Red Hat Enterprise Linux 6.2: C, C++, Perl, Python, Java, Ruby and OCaml.

- ▶ To install C and C++ bindings, enter the following command:

```
# yum install libguestfs-devel
```

- ▶ To install Perl bindings:

```
# yum install 'perl(Sys::Guestfs)'
```

- ▶ To install Python bindings:

```
# yum install python-libguestfs
```

- ▶ To install Java bindings:

```
# yum install libguestfs-java libguestfs-java-devel libguestfs-javadoc
```

- ▶ To install Ruby bindings:

```
# yum install ruby-libguestfs
```

- ▶ To install OCaml bindings:

```
# yum install ocaml-libguestfs ocaml-libguestfs-devel
```

The binding for each language is essentially the same, but with minor syntactic changes. A C statement:

```
guestfs_launch (g);
```

Would appear like the following in Perl:

```
$g->launch ()
```

Or like the following in OCaml:

```
g#launch ()
```

Only the API from C is detailed in this section.

In the C and C++ bindings, you must manually check for errors. In the other bindings, errors are converted into exceptions; the additional error checks shown in the examples below are not necessary for other languages, but conversely you may wish to add code to catch exceptions. Refer to the following list for some points of interest regarding the architecture of the libguestfs API:

- ▶ The libguestfs API is synchronous. Each call blocks until it has completed. If you want to make calls asynchronously, you have to create a thread.
- ▶ The libguestfs API is not thread safe: each handle should be used only from a single thread, or if you want to share a handle between threads you should implement your own mutex to ensure that two threads cannot execute commands on one handle at the same time.
- ▶ You should not open multiple handles on the same disk image. It is permissible if all the handles are read-only, but still not recommended.
- ▶ You should not add a disk image for writing if anything else could be using that disk image (eg. a live VM). Doing this will cause disk corruption.
- ▶ Opening a read-only handle on a disk image which is currently in use (eg. by a live VM) is possible; however, the results may be unpredictable or inconsistent particularly if the disk image is being heavily written to at the time you are reading it.

16.11.1. Interaction with the API via a C program

Your C program should start by including the <guestfs.h> header file, and creating a handle:

```
#include <stdio.h>
#include <stdlib.h>
#include <guestfs.h>

int
main (int argc, char *argv[])
{
    guestfs_h *g;

    g = guestfs_create ();
    if (g == NULL) {
        perror ("failed to create libguestfs handle");
        exit (EXIT_FAILURE);
    }

    /* ... */

    guestfs_close (g);

    exit (EXIT_SUCCESS);
}
```

Save this program to a file (**test.c**). Compile this program and run it with the following two commands:

```
gcc -Wall test.c -o test -lguestfs
./test
```

At this stage it should print no output. The rest of this section demonstrates an example showing how to extend this program to create a new disk image, partition it, format it with an ext4 file system, and create some files in the file system. The disk image will be called **disk.img** and be created in the current directory.

The outline of the program is:

- ▶ Create the handle.
- ▶ Add disk(s) to the handle.
- ▶ Launch the libguestfs back end.
- ▶ Create the partition, file system and files.
- ▶ Close the handle and exit.

Here is the modified program:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <guestfs.h>

int
main (int argc, char *argv[])
{
    guestfs_h *g;
    size_t i;

    g = guestfs_create ();
    if (g == NULL) {
        perror ("failed to create libguestfs handle");
        exit (EXIT_FAILURE);
    }

    /* Create a raw-format sparse disk image, 512 MB in size. */
    int fd = open ("disk.img", O_CREAT|O_WRONLY|O_TRUNC|O_NOCTTY, 0666);
    if (fd == -1) {
        perror ("disk.img");
        exit (EXIT_FAILURE);
    }
    if (ftruncate (fd, 512 * 1024 * 1024) == -1) {
        perror ("disk.img: truncate");
        exit (EXIT_FAILURE);
    }
    if (close (fd) == -1) {
        perror ("disk.img: close");
        exit (EXIT_FAILURE);
    }

    /* Set the trace flag so that we can see each libguestfs call. */
    guestfs_set_trace (g, 1);

    /* Set the autosync flag so that the disk will be synchronized
     * automatically when the libguestfs handle is closed.
     */
    guestfs_set_autosync (g, 1);

    /* Add the disk image to libguestfs. */
    if (guestfs_add_drive_opts (g, "disk.img",
        GUESTFS_ADD_DRIVE_OPTS_FORMAT, "raw", /* raw format */
        GUESTFS_ADD_DRIVE_OPTS_READONLY, 0, /* for write */
        -1 /* this marks end of optional arguments */ )
        == -1)
        exit (EXIT_FAILURE);

    /* Run the libguestfs back-end. */
    if (guestfs_launch (g) == -1)
        exit (EXIT_FAILURE);

    /* Get the list of devices. Because we only added one drive
     * above, we expect that this list should contain a single
     * element.
     */
    char **devices = guestfs_list_devices (g);
    if (devices == NULL)
        exit (EXIT_FAILURE);
    if (devices[0] == NULL || devices[1] != NULL) {
        fprintf (stderr,
            "error: expected a single device from list-devices\n");
        exit (EXIT_FAILURE);
    }
}

```

```

}

/* Partition the disk as one single MBR partition. */
if (guestfs_part_disk (g, devices[0], "mbr") == -1)
    exit (EXIT_FAILURE);

/* Get the list of partitions. We expect a single element, which
 * is the partition we have just created.
 */
char **partitions = guestfs_list_partitions (g);
if (partitions == NULL)
    exit (EXIT_FAILURE);
if (partitions[0] == NULL || partitions[1] != NULL) {
    fprintf (stderr,
             "error: expected a single partition from list-partitions\n");
    exit (EXIT_FAILURE);
}

/* Create an ext4 filesystem on the partition. */
if (guestfs_mkfs (g, "ext4", partitions[0]) == -1)
    exit (EXIT_FAILURE);

/* Now mount the filesystem so that we can add files. */
if (guestfs_mount_options (g, "", partitions[0], "/") == -1)
    exit (EXIT_FAILURE);

/* Create some files and directories. */
if (guestfs_touch (g, "/empty") == -1)
    exit (EXIT_FAILURE);

const char *message = "Hello, world\n";
if (guestfs_write (g, "/hello", message, strlen (message)) == -1)
    exit (EXIT_FAILURE);

if (guestfs_mkdir (g, "/foo") == -1)
    exit (EXIT_FAILURE);

/* This uploads the local file /etc/resolv.conf into the disk image. */
if (guestfs_upload (g, "/etc/resolv.conf", "/foo/resolv.conf") == -1)
    exit (EXIT_FAILURE);

/* Because 'autosync' was set (above) we can just close the handle
 * and the disk contents will be synchronized. You can also do
 * this manually by calling guestfs_umount_all and guestfs_sync.
 */
guestfs_close (g);

/* Free up the lists. */
for (i = 0; devices[i] != NULL; ++i)
    free (devices[i]);
free (devices);
for (i = 0; partitions[i] != NULL; ++i)
    free (partitions[i]);
free (partitions);

exit (EXIT_SUCCESS);
}

```

Compile and run this program with the following two commands:

```
gcc -Wall test.c -o test -lguestfs
./test
```

If the program runs to completion successfully then you should be left with a disk image called

disk.img, which you can examine with guestfish:

```
guestfish --ro -a disk.img -m /dev/sda1
><fs> ll /
><fs> cat /foo/resolv.conf
```

By default (for C and C++ bindings only), libguestfs prints errors to stderr. You can change this behavior by setting an error handler. The `guestfs(3)` man page discusses this in detail.

16.12. Troubleshooting

A test tool is available to check that libguestfs is working. Run the following command after installing libguestfs (root access not required) to test for normal operation:

```
$ libguestfs-test-tool
```

This tool prints a large amount of text to test the operation of libguestfs. If the test is successful, the following text will appear near the end of the output:

```
===== TEST FINISHED OK =====
```

16.13. Where to find further documentation

The primary source for documentation for libguestfs and the tools are the Unix man pages. The API is documented in `guestfs(3)`. `guestfish` is documented in `guestfish(1)`. The virt tools are documented in their own man pages (eg. `virt-df(1)`).

Chapter 17. Using simple tools for guest virtual machine management

In addition to virt-manager, there are other smaller more minimal tools that can allow you to have access to your guest virtual machine's console. The sections that follow describe and explain these tools.

17.1. Using virt-viewer

virt-viewer is a minimal tool for displaying the graphical console of a guest virtual machine. The console is accessed using the VNC or SPICE protocol. The guest can be referred to based on its name, ID, or UUID. If the guest is not already running, then the viewer can be told to wait until it starts before attempting to connect to the console. The viewer can connect to remote hosts to lookup the console information and then also connect to the remote console using the same network transport.

To install the *virt-viewer* tool, run:

```
# sudo yum install virt-viewer
```

The basic virt viewer commands are as follows:

```
# virt-viewer [OPTIONS] DOMAIN-NAME|ID|UUID
```

The following options may be used with *virt-viewer*

- ▶ **-h**, or **--help** - Displays the command line help summary.
- ▶ **-V**, or **--version** - Displays the *virt-viewer* version number.
- ▶ **-v**, or **--verbose** - Displays information about the connection to the guest virtual machine
- ▶ **-c [URI]**, or **--connect=URI** - Specifies the hypervisor connection URI
- ▶ **-w**, or **--wait** - Causes the domain to start up before attempting to connect to the console.
- ▶ **-r**, or **--reconnect** - Automatically reconnects to the domain if it shuts down and restarts
- ▶ **-z PCT**, or **--zoom=PCT** - Adjusts the zoom level of the display window in the specified percentage. Accepted range 10-200%.
- ▶ **-a**, or **--attach** - Uses *libvirt* to directly attach to a local display, instead of making a TCP/UNIX socket connection. This avoids the need to authenticate with the remote display, if authentication with *libvirt* is already allowed. This option does not work with remote displays.
- ▶ **-f**, or **--full-screen** - Starts with the command window maximized to its fullscreen size.
- ▶ **-h HOTKEYS**, or **--hotkeys HOTKEYS** - Overrides the default hotkey settings with the new specified hotkey. Refer to [Example 17.4, “Setting hot keys”](#).
- ▶ **--debug** - Prints debugging information

Example 17.1. Connecting to a guest virtual machine

If using a XEN hypervisor:

```
# virt-viewer guest-name
```

If using a KVM-QEMU hypervisor:

```
# virt-viewer --connect qemu:///system 7
```

Example 17.2. To wait for a specific guest to start before connecting

Enter the following command:

```
# virt-viewer --reconnect --wait 66ab33c0-6919-a3f7-e659-16c82d248521
```

Example 17.3. To connect to a remote console using TLS

Enter the following command:

```
#virt-viewer --connect xen://example.org/ demo
```

To connect to a remote host using SSH, lookup the guest configuration and then make a direct non-tunnelled connection to the console

Example 17.4. Setting hot keys

To create a customized hotkey, run the following command:

```
# virt-viewer --hotkeys=([action1]=[key-combination1]), ([action2]=[key-combination2])
```

The following actions can be assigned to a hotkey:

- ▶ toggle-fullscreen
- ▶ release-cursor
- ▶ smartcard-insert
- ▶ smartcard-remove

Key name combinations are case insensitive. Each hot key setting should have a unique key combination.

For example, to create a hotkey to change to full screen mode:

```
#virt-viewer --hotkeys=toggle-fullscreen=shift+f11 qemu:///system 7
```

17.2. remote-viewer

The *remote-viewer* is a simple remote desktop display client that supports SPICE and VNC.

To install the *virt-viewer* tool, run:

```
# sudo yum install remote-viewer
```

The basic remote viewer commands are as follows:**remote-viewer [OPTIONS] DOMAIN-NAME|ID|UUID**

The following options may be used with *remote-viewer*

- ▶ **-h**, or **--help** - Displays the command line help summary.
- ▶ **-V**, or **--version** - Displays the *remote-viewer* version number.
- ▶ **-v**, or **--verbose** - Displays information about the connection to the guest virtual machine
- ▶ **-z PCT**, or **--zoom=PCT** - Adjusts the zoom level of the display window in the specified percentage.

Accepted range 10-200%.

- ▶ **-f**, or **--full-screen** - Starts with the command window maximized to its fullscreen size.
- ▶ **-t TITLE**, or **--title TITLE** - Sets the window title to the string given
- ▶ **--spice-controller** - Uses the SPICE controller to initialize the connection with the SPICE server. This option is used by the SPICE browser addons to allow web page to start a client.
- ▶ **--debug** - Prints debugging information

For more information see the MAN page for the remote-viewer.

Chapter 18. Virtual Networking

This chapter introduces the concepts needed to create, start, stop, remove, and modify virtual networks with libvirt.

Additional information can be found in the libvirt reference chapter

18.1. Virtual network switches

Libvirt virtual networking uses the concept of a *virtual network switch*. A virtual network switch is a software construct that operates on a host physical machine server, to which virtual machines (guests) connect. The network traffic for a guest is directed through this switch:

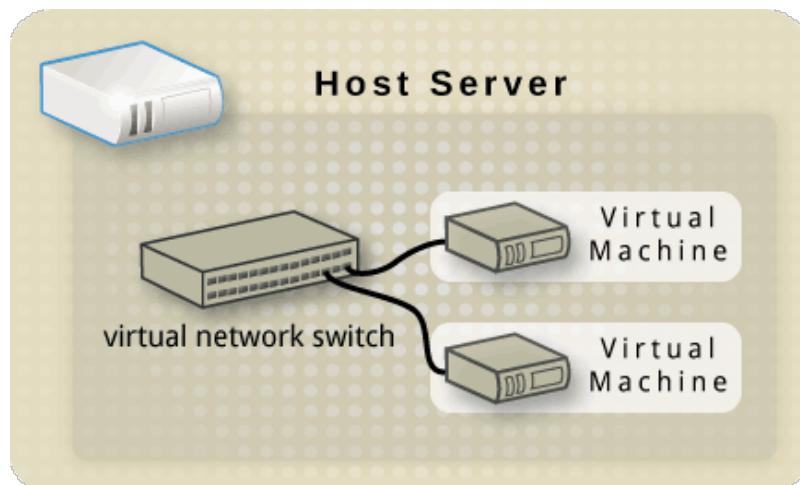


Figure 18.1. Virtual network switch with two guests

Linux host physical machine servers represent a virtual network switch as a network interface. When the libvirtd daemon (**libvirtd**) is first installed and started, the default network interface representing the virtual network switch is **virbr0**.

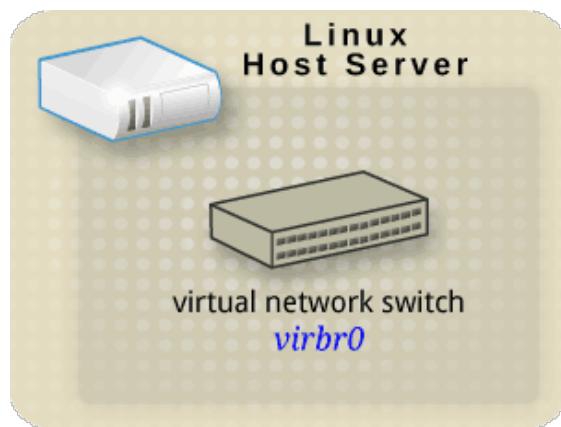


Figure 18.2. Linux host physical machine with an interface to a virtual network switch

This **virbr0** interface can be viewed with the **ifconfig** and **ip** commands like any other interface:

```
$ ifconfig virbr0
virbr0      Link encap:Ethernet  HWaddr 1B:C4:94:CF:FD:17
            inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
                  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                  RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:0
                  RX bytes:0 (0.0 b)  TX bytes:3097 (3.0 KiB)
```

```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
        inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

18.2. Network Address Translation

By default, virtual network switches operate in NAT mode. They use IP masquerading rather than SNAT (Source-NAT) or DNAT (Destination-NAT). IP masquerading enables connected guests to use the host physical machine IP address for communication to any external network. By default, computers that are placed externally to the host physical machine cannot communicate to the guests inside when the virtual network switch is operating in NAT mode, as shown in the following diagram:

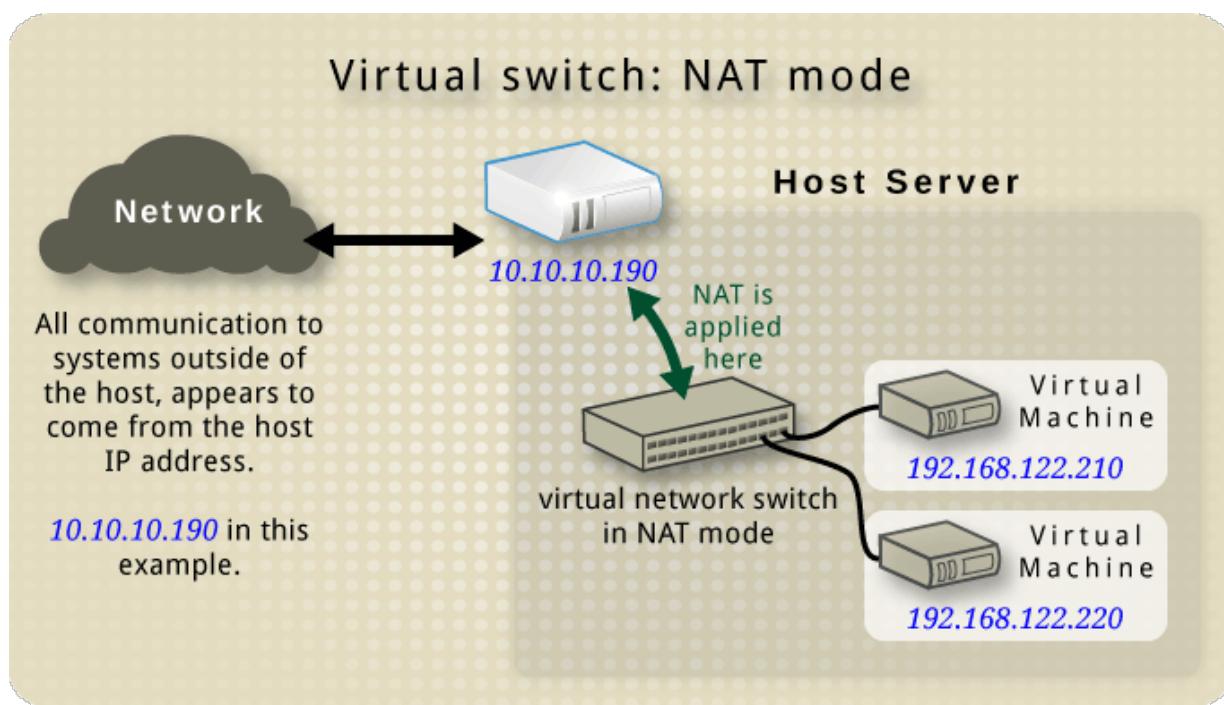


Figure 18.3. Virtual network switch using NAT with two guests



Warning

Virtual network switches use NAT configured by iptables rules. Editing these rules while the switch is running is not recommended, as incorrect rules may result in the switch being unable to communicate.

If the switch is not running, you can set the public IP range for forward mode NAT in order to create a port masquerading range by running:

```
# iptables -j SNAT --to-source [start]-[end]
```

18.3. Networking protocols

The following sections describe individual networking protocols and how they are used in libvirt

18.3.1. DNS and DHCP

IP information can be assigned to guests via DHCP. A pool of addresses can be assigned to a virtual network switch for this purpose. Libvirt uses the `dnsmasq` program for this. An instance of dnsmasq is automatically configured and started by libvirt for each virtual network switch that needs it.

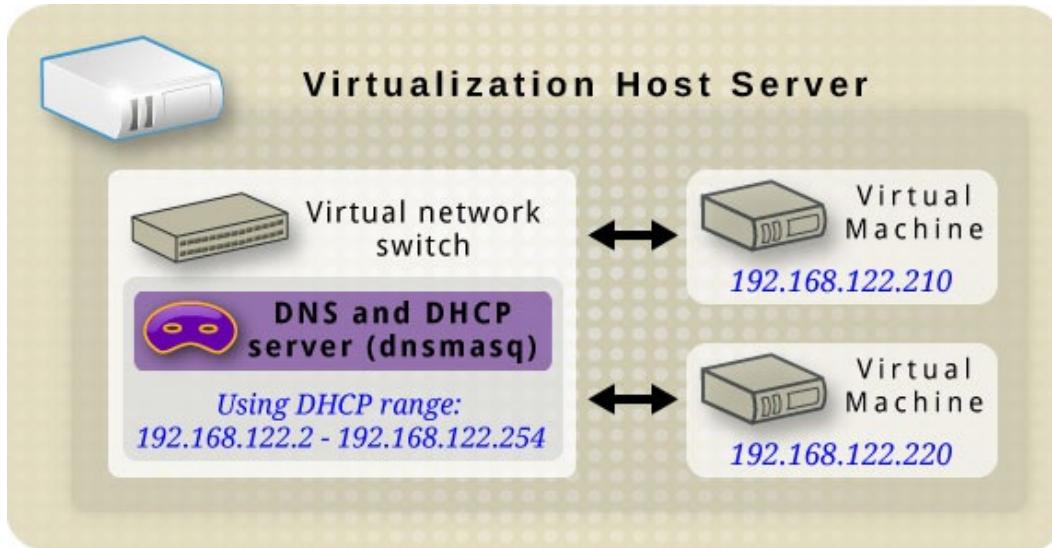


Figure 18.4. Virtual network switch running dnsmasq

18.3.2. Routed mode

When using *routed mode*, the virtual switch connects to the physical LAN connected to the host physical machine, passing traffic back and forth without the use of NAT. The virtual switch can examine all traffic and use the information contained within the network packets to make routing decisions. When using this mode, all of the virtual machines are in their own subnet, routed through a virtual switch. This situation is not always ideal as no other host physical machines on the physical network are aware of the virtual machines without manual physical router configuration, and cannot access the virtual machines. Routed mode operates at Layer 3 of the OSI networking model.

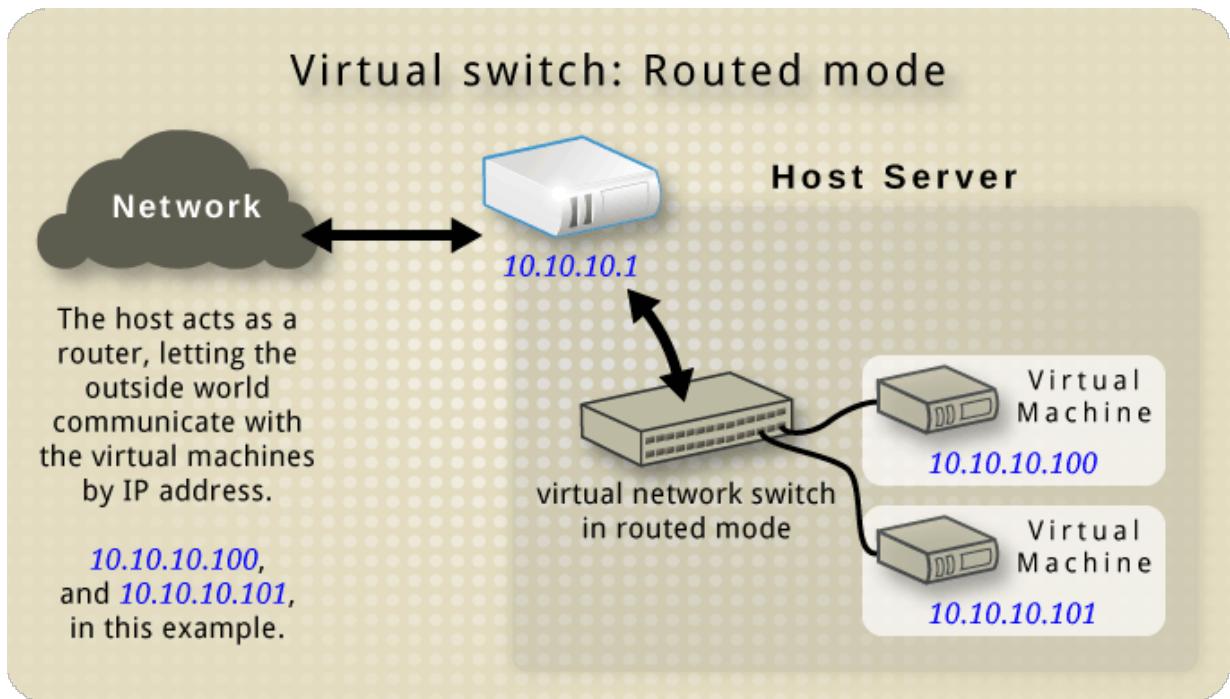


Figure 18.5. Virtual network switch in routed mode

18.3.3. Isolated mode

When using *Isolated mode*, guests connected to the virtual switch can communicate with each other, and with the host physical machine, but their traffic will not pass outside of the host physical machine, nor can they receive traffic from outside the host physical machine. Using dnsmasq in this mode is required for basic functionality such as DHCP. However, even if this network is isolated from any physical network, DNS names are still resolved. Therefore a situation can arise when DNS names resolve but ICMP echo request (ping) commands fail.

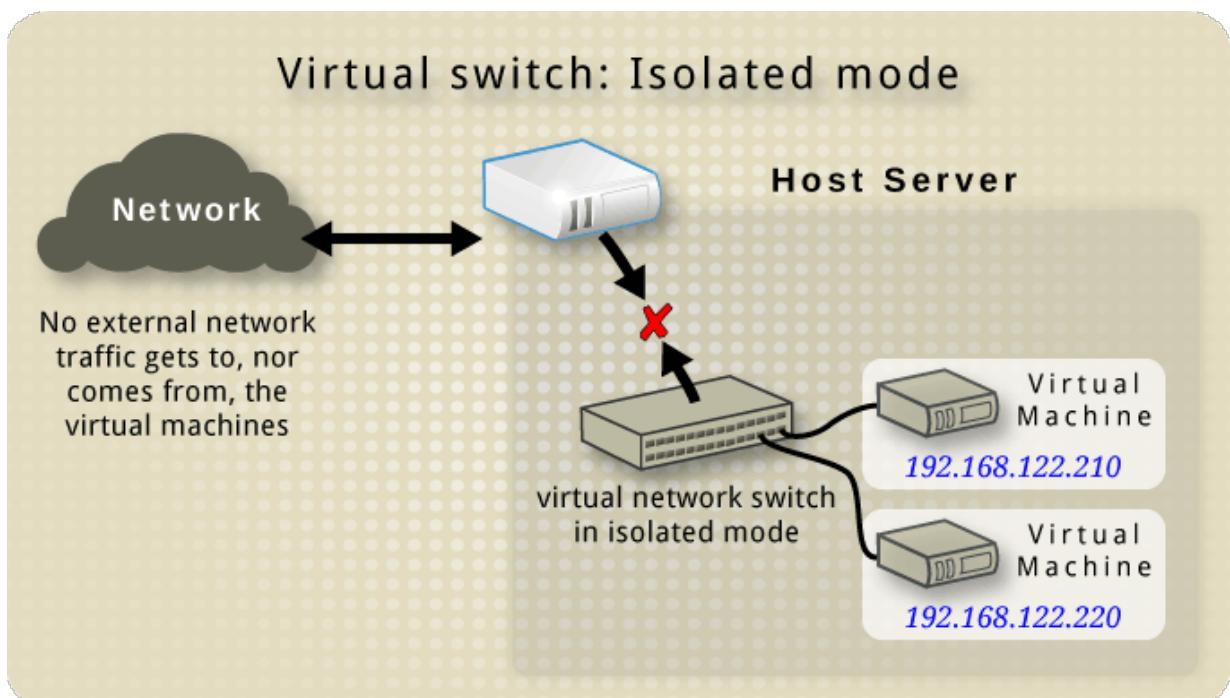


Figure 18.6. Virtual network switch in isolated mode

18.4. The default configuration

When the libvirtd daemon (**libvirtd**) is first installed, it contains an initial virtual network switch configuration in NAT mode. This configuration is used so that installed guests can communicate to the external network, through the host physical machine. The following image demonstrates this default configuration for **libvirtd**:

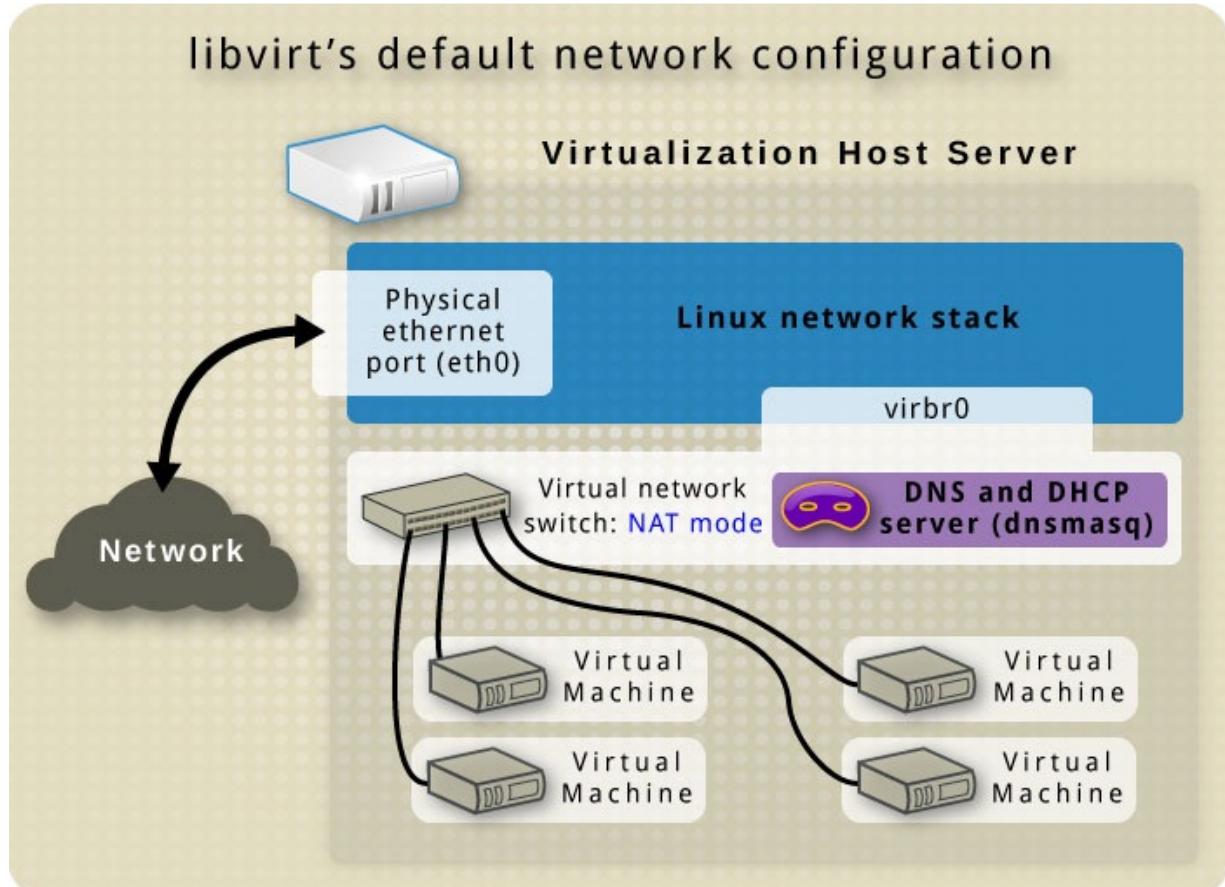


Figure 18.7. Default libvirt network configuration

Note

A virtual network can be restricted to a specific physical interface. This may be useful on a physical system that has several interfaces (for example, **eth0**, **eth1** and **eth2**). This is only useful in routed and NAT modes, and can be defined in the **dev=<interface>** option, or in **virt-manager** when creating a new virtual network.

18.5. Examples of common scenarios

This section demonstrates different virtual networking modes and provides some example scenarios.

18.5.1. Routed mode

DMZ

Consider a network where one or more nodes are placed in a controlled subnetwork for security reasons. The deployment of a special subnetwork such as this is a common practice, and the

subnetwork is known as a DMZ. Refer to the following diagram for more details on this layout:

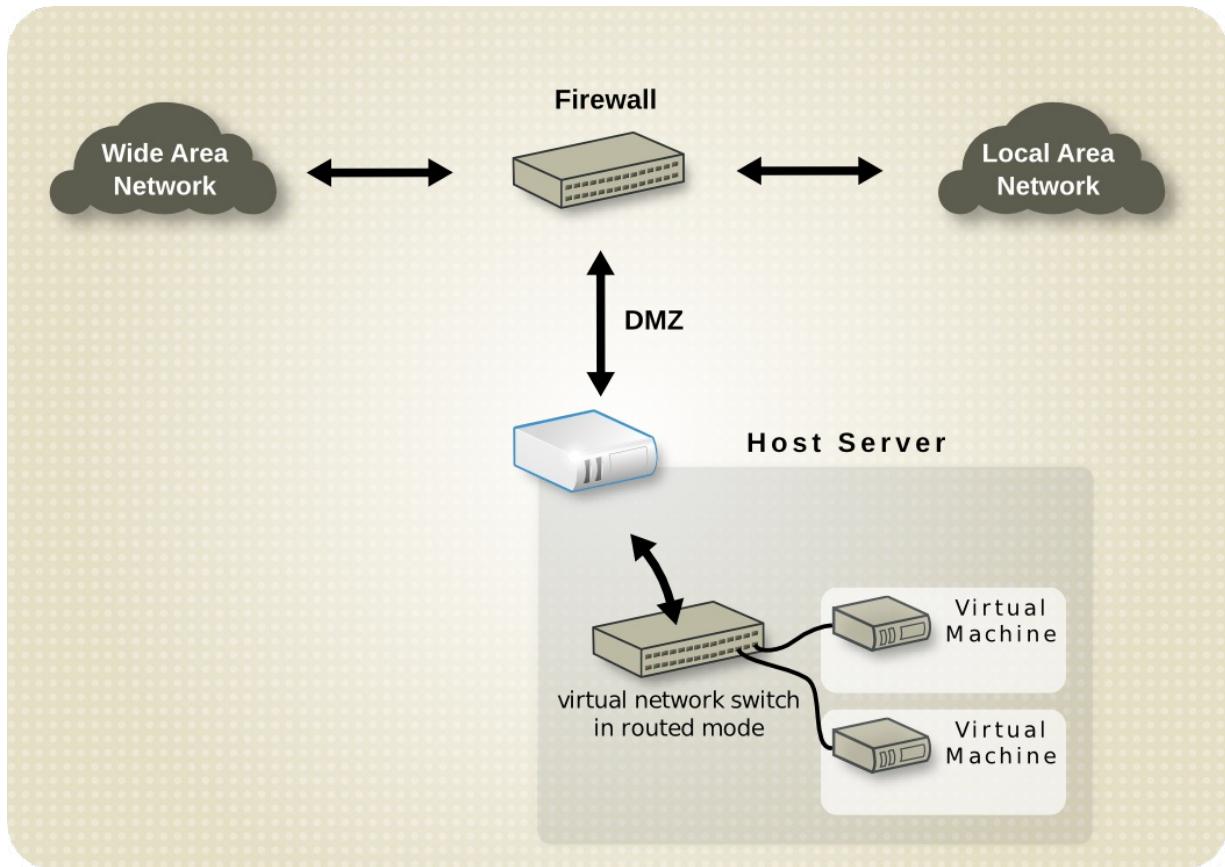


Figure 18.8. Sample DMZ configuration

Host physical machines in a DMZ typically provide services to WAN (external) host physical machines as well as LAN (internal) host physical machines. As this requires them to be accessible from multiple locations, and considering that these locations are controlled and operated in different ways based on their security and trust level, routed mode is the best configuration for this environment.

Virtual Server hosting

Consider a virtual server hosting company that has several host physical machines, each with two physical network connections. One interface is used for management and accounting, the other is for the virtual machines to connect through. Each guest has its own public IP address, but the host physical machines use private IP address as management of the guests can only be performed by internal administrators. Refer to the following diagram to understand this scenario:

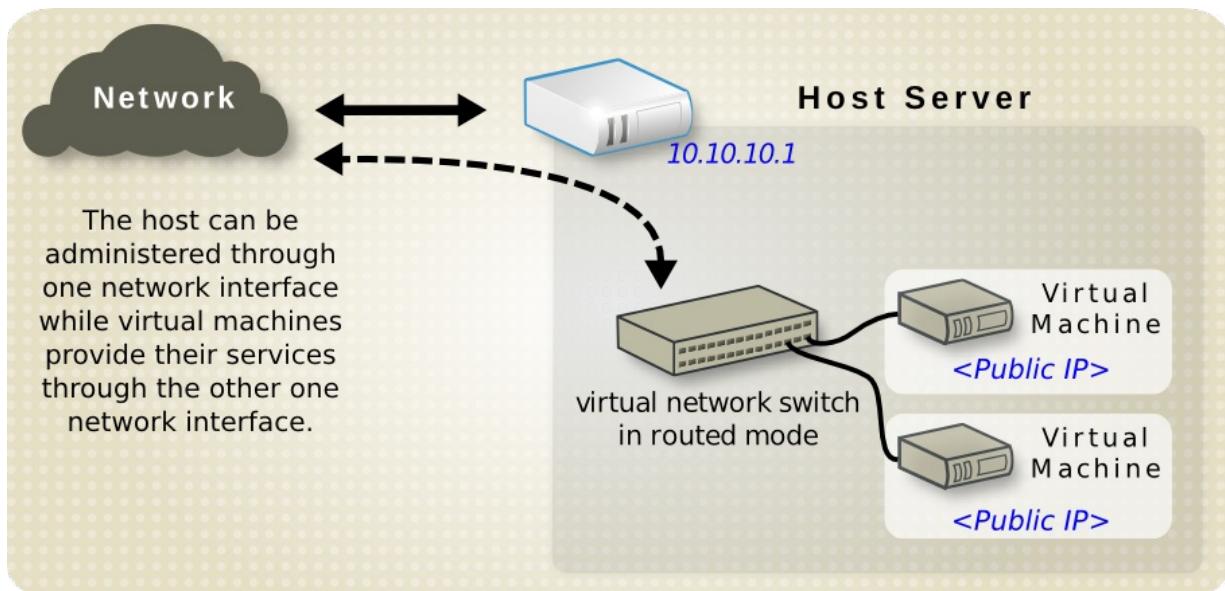


Figure 18.9. Virtual server hosting sample configuration

When the host physical machine has a public IP address and the virtual machines have static public IP addresses, bridged networking cannot be used, as the provider only accepts packets from the MAC address of the public host physical machine. The following diagram demonstrates this:

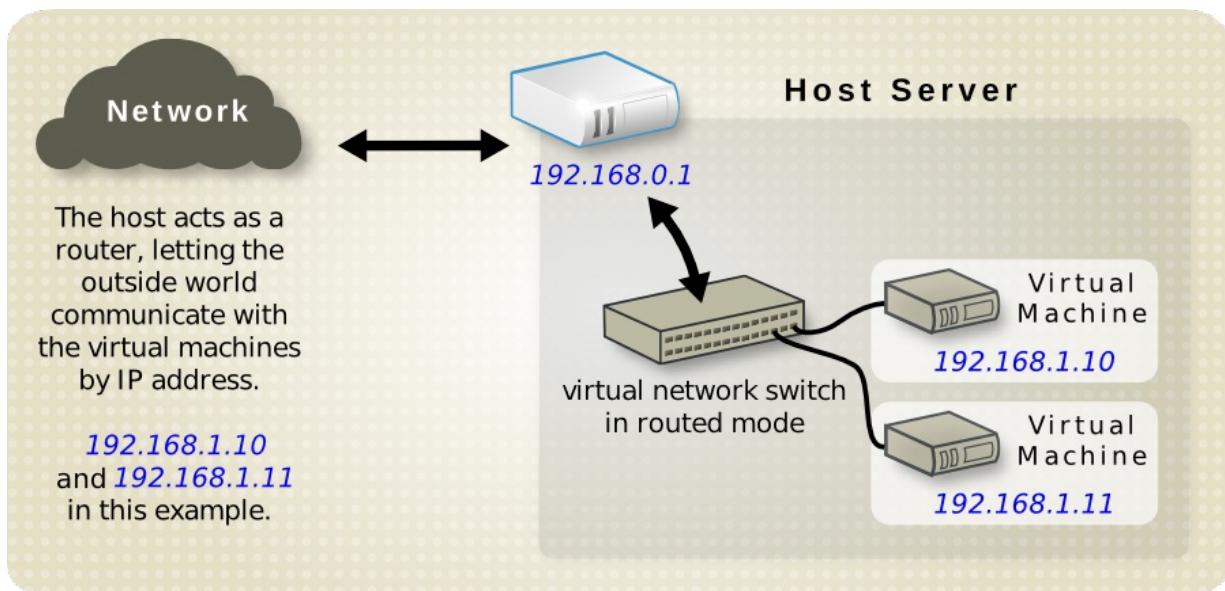


Figure 18.10. Virtual server using static IP addresses

18.5.2. NAT mode

NAT (Network Address Translation) mode is the default mode. It can be used for testing when there is no need for direct network visibility.

18.5.3. Isolated mode

Isolated mode allows virtual machines to communicate with each other only. They are unable to interact with the physical network.

18.6. Managing a virtual network

To configure a virtual network on your system:

- From the **Edit** menu, select **Connection Details**.

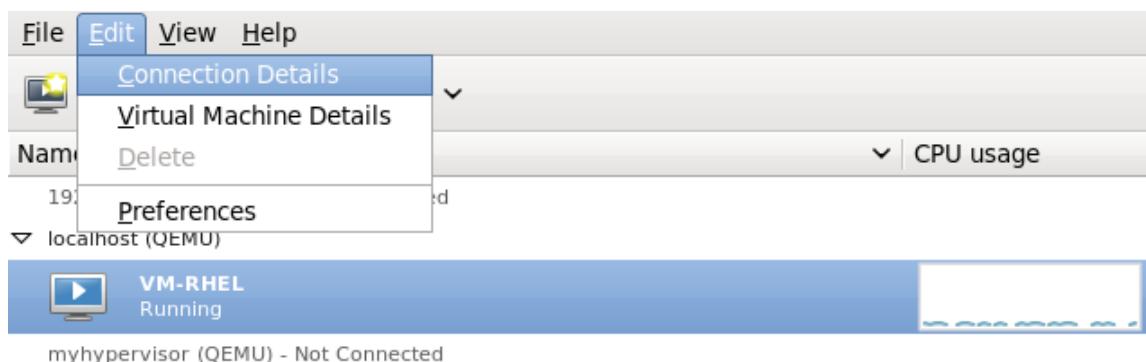


Figure 18.11. Selecting a host physical machine's details

- This will open the **Connection Details** menu. Click the **Virtual Networks** tab.

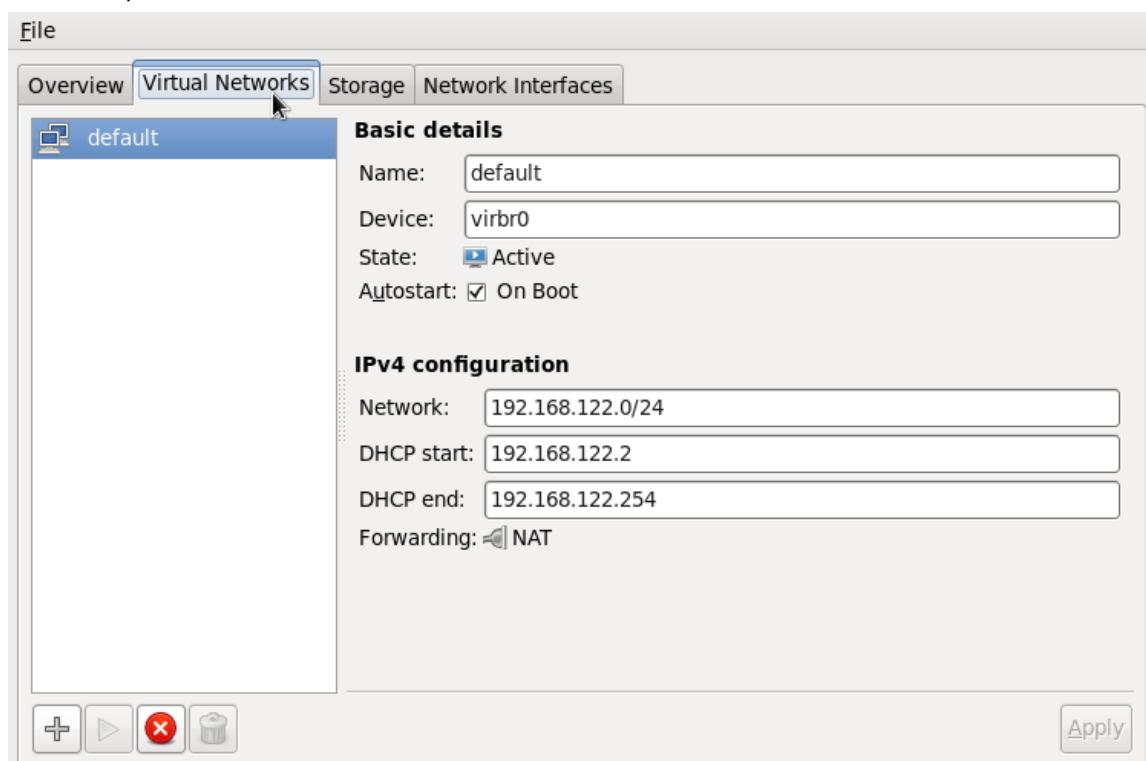


Figure 18.12. Virtual network configuration

- All available virtual networks are listed on the left-hand box of the menu. You can edit the configuration of a virtual network by selecting it from this box and editing as you see fit.

18.7. Creating a virtual network

To create a virtual network on your system:

1. Open the **Virtual Networks** tab from within the **Connection Details** menu. Click the **Add Network** button, identified by a plus sign (+) icon. For more information, refer to [Section 18.6, "Managing a virtual network"](#).

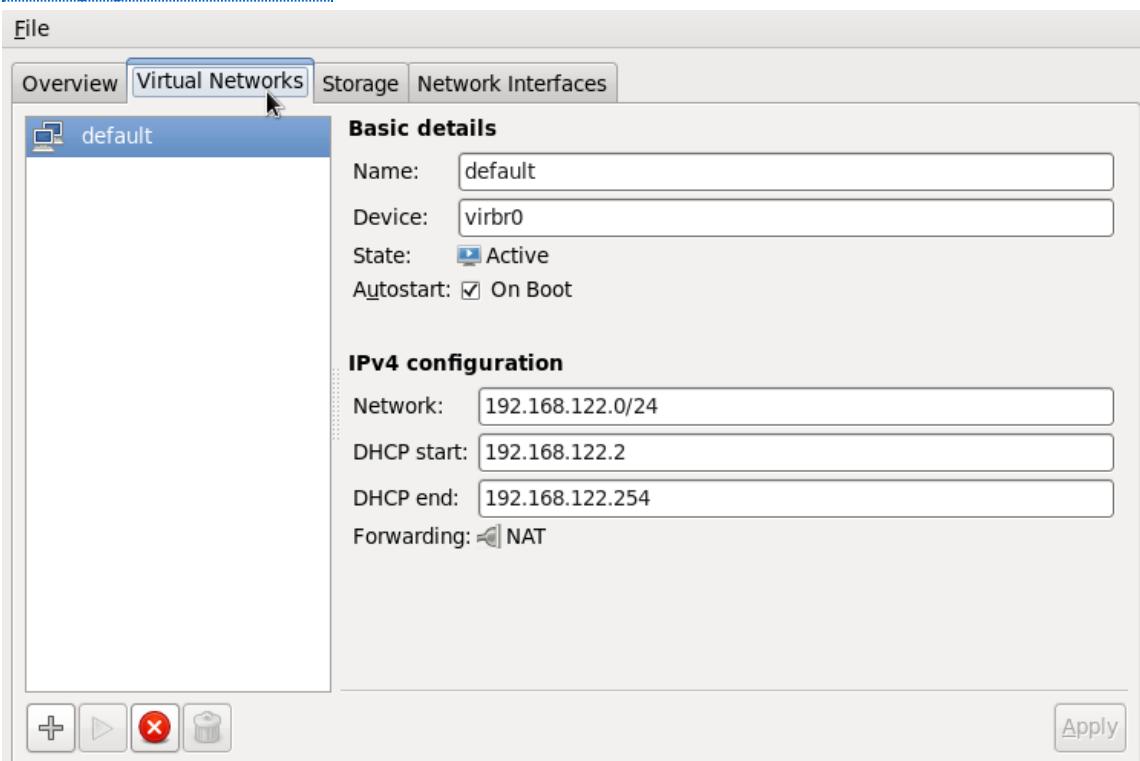


Figure 18.13. Virtual network configuration

This will open the **Create a new virtual network** window. Click **Forward** to continue.



Figure 18.14. Creating a new virtual network

2. Enter an appropriate name for your virtual network and click **Forward**.

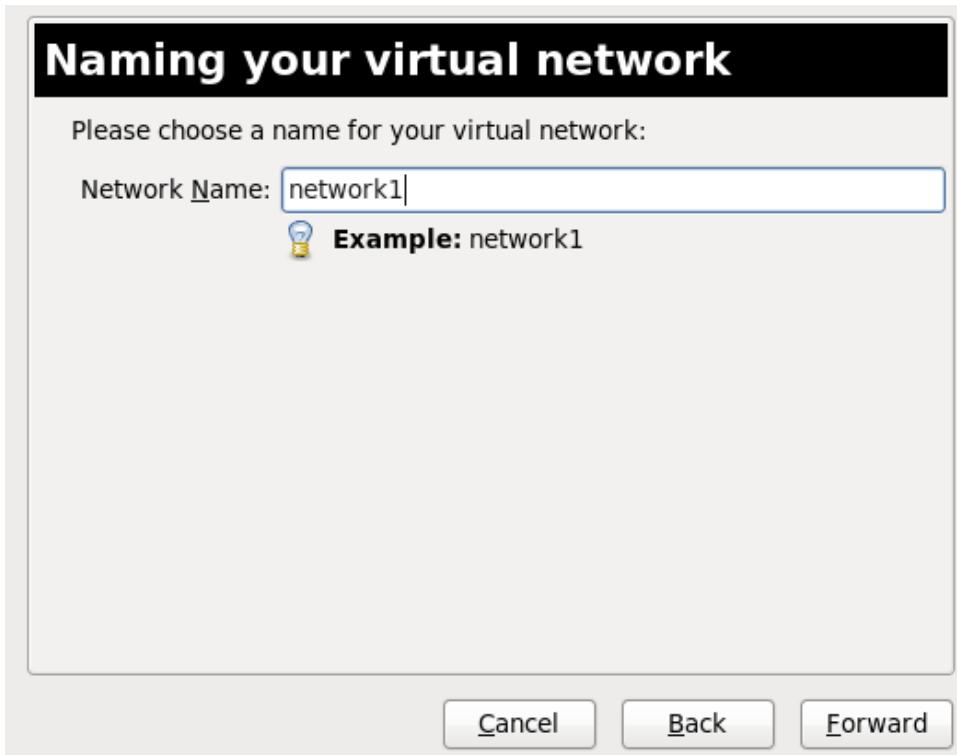


Figure 18.15. Naming your virtual network

3. Enter an IPv4 address space for your virtual network and click **Forward**.

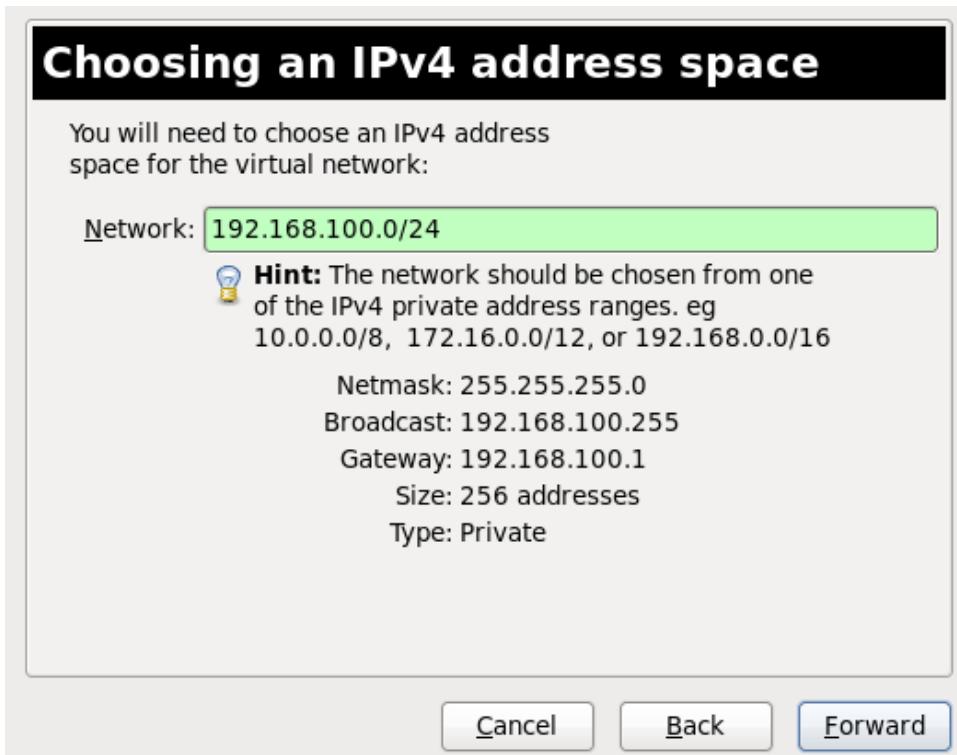


Figure 18.16. Choosing an IPv4 address space

4. Define the DHCP range for your virtual network by specifying a **Start** and **End** range of IP addresses. Click **Forward** to continue.

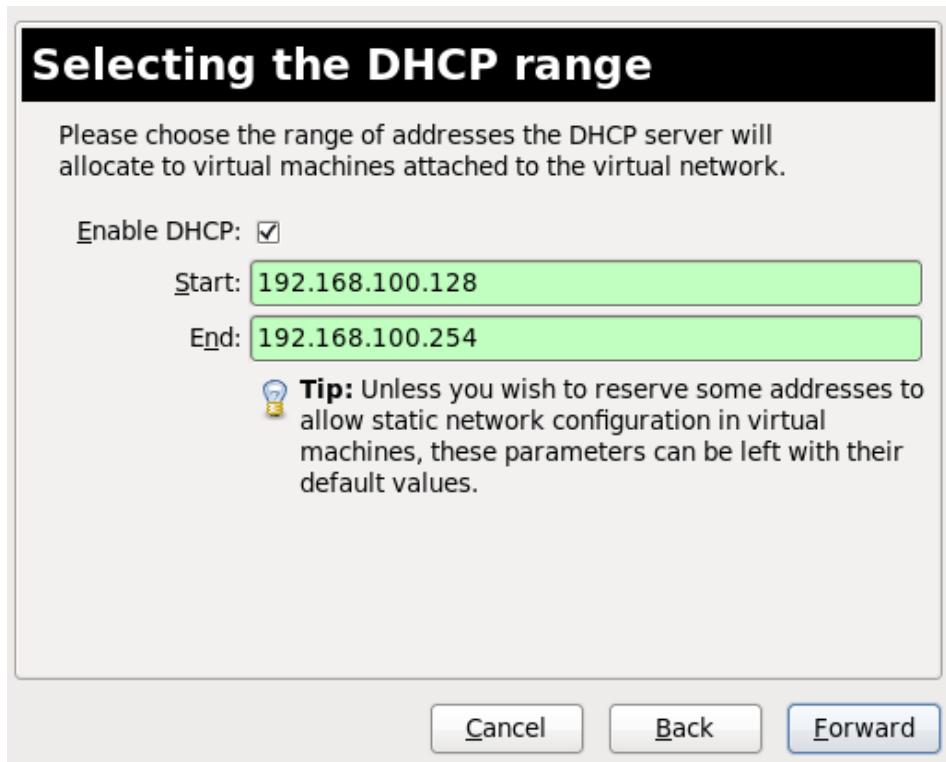


Figure 18.17. Selecting the DHCP range

5. Select how the virtual network should connect to the physical network.

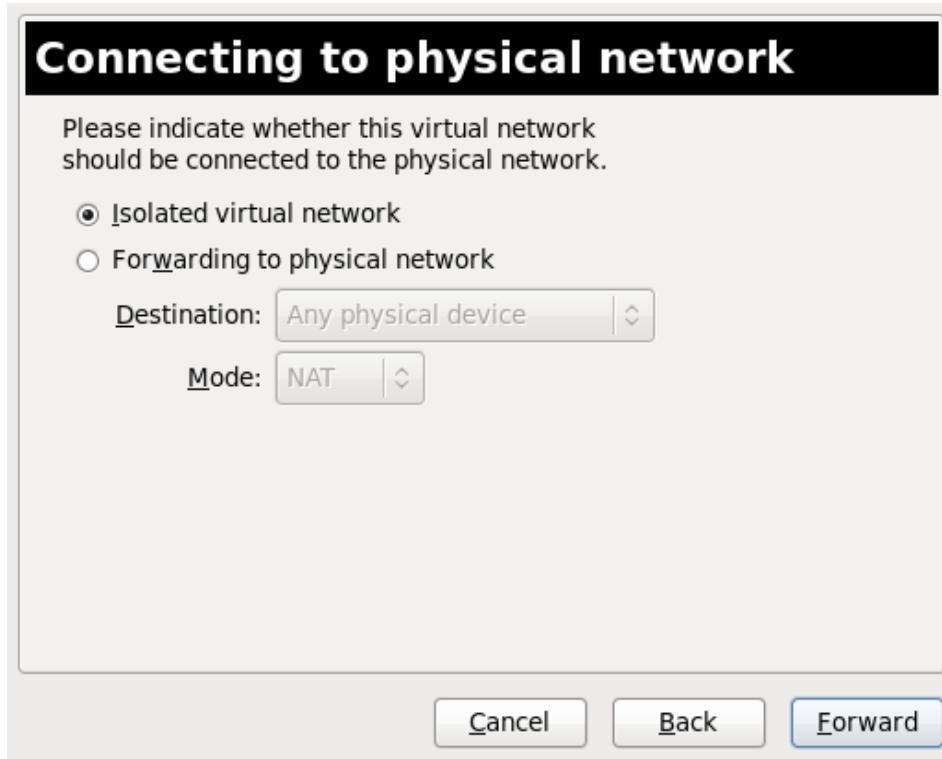


Figure 18.18. Connecting to physical network

If you select **Forwarding to physical network**, choose whether the **Destination** should be **Any physical device** or a specific physical device. Also select whether the **Mode** should be **NAT** or **Routed**.

- Click **Forward** to continue.
- You are now ready to create the network. Check the configuration of your network and click **Finish**.

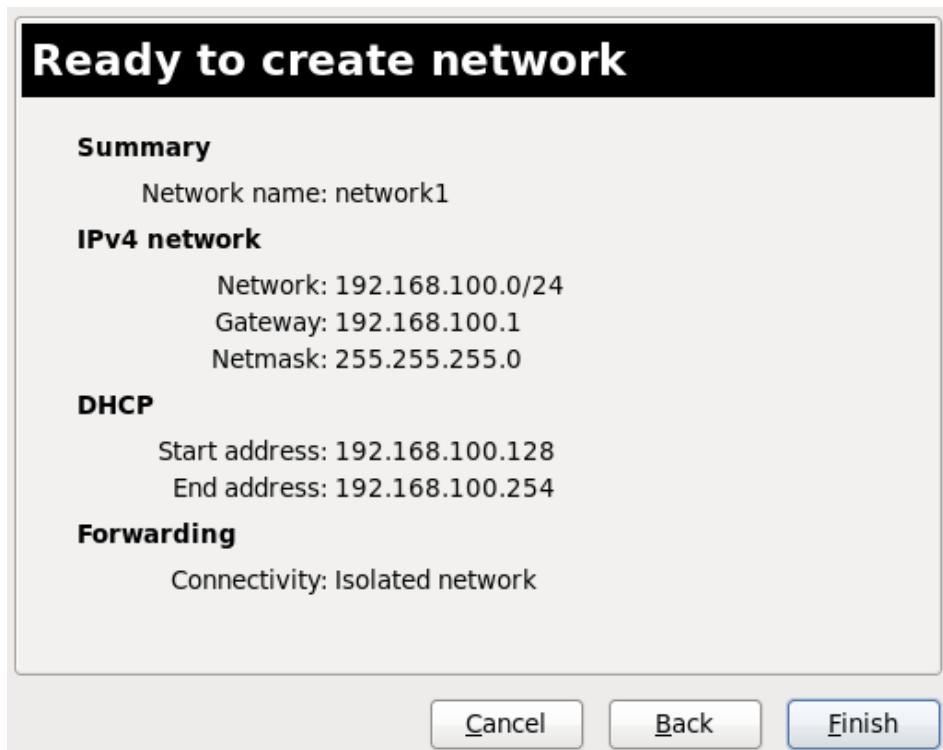


Figure 18.19. Ready to create network

- The new virtual network is now available in the **Virtual Networks** tab of the **Connection Details** window.

18.8. Attaching a virtual network to a guest

To attach a virtual network to a guest:

- In the **Virtual Machine Manager** window, highlight the guest that will have the network assigned.

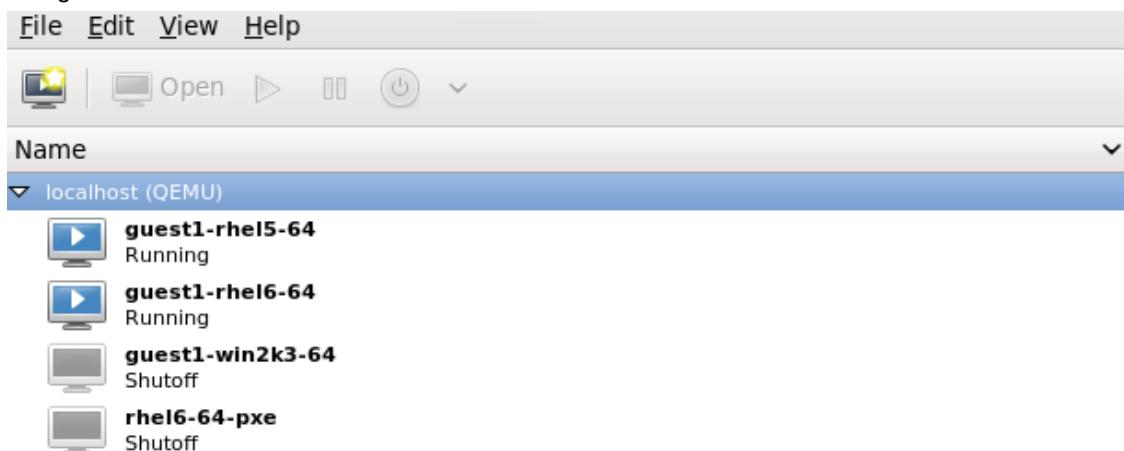


Figure 18.20. Selecting a virtual machine to display

- From the Virtual Machine Manager **Edit** menu, select **Virtual Machine Details**.

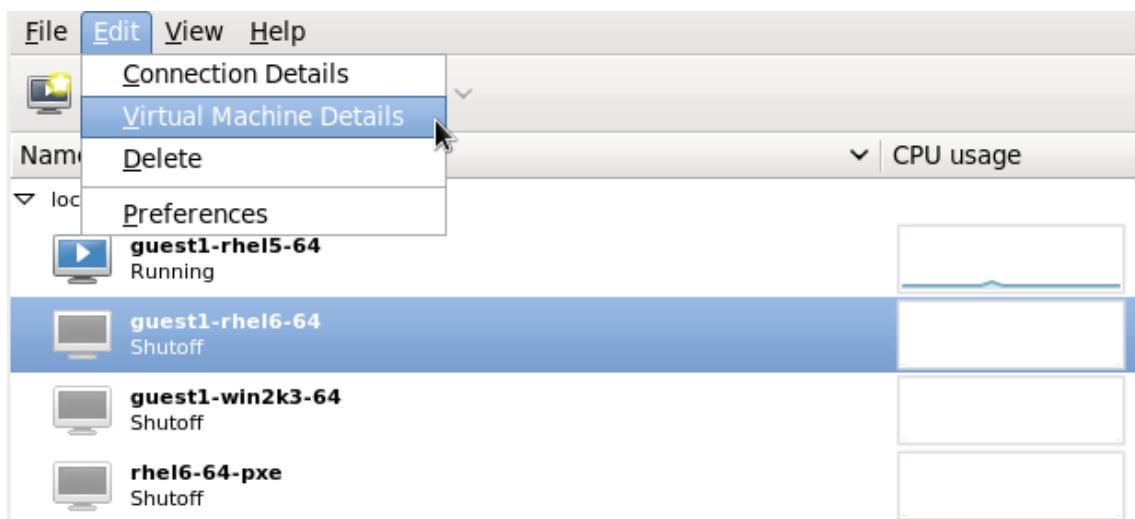


Figure 18.21. Displaying the virtual machine details

- Click the **Add Hardware** button on the Virtual Machine Details window.

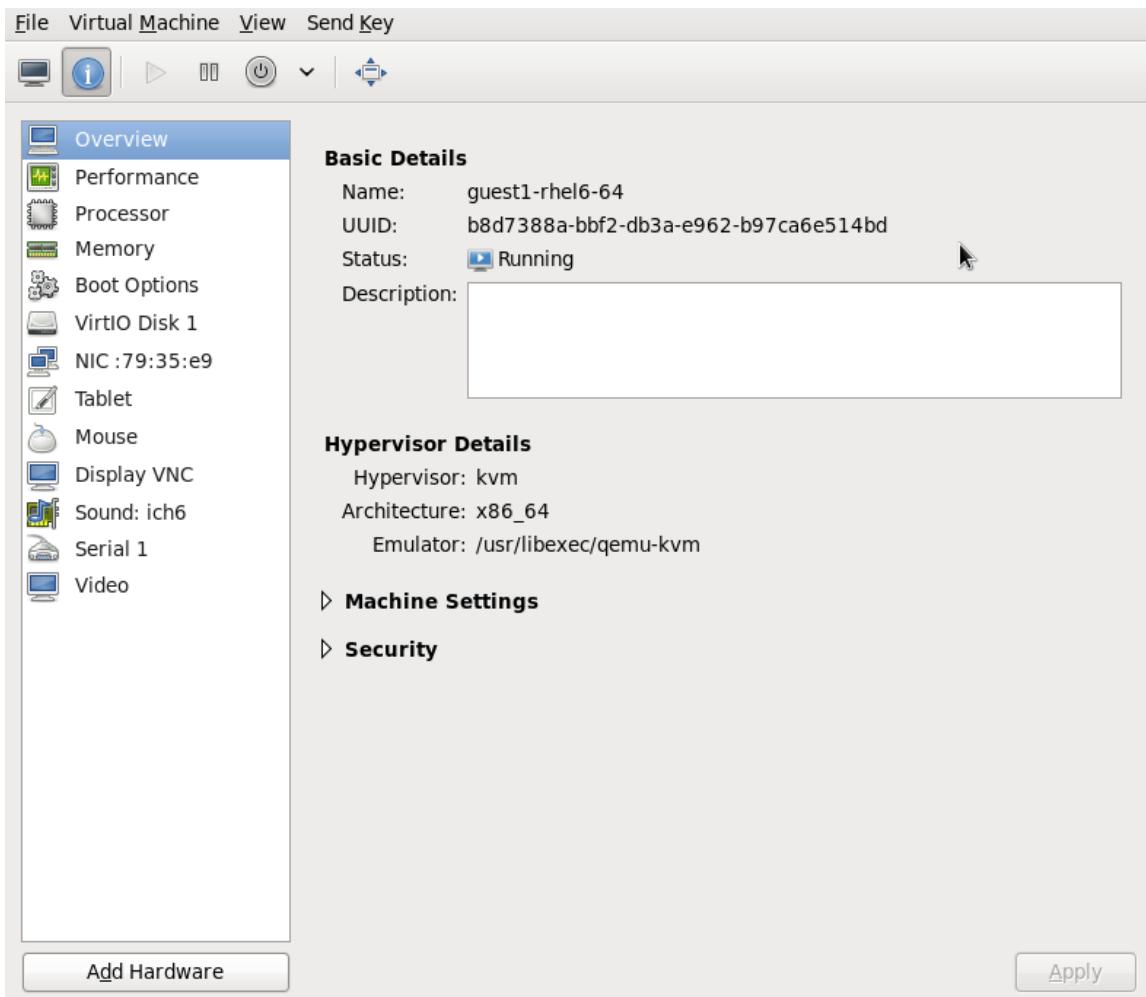


Figure 18.22. The Virtual Machine Details window

- In the **Add new virtual hardware** window, select **Network** from the left pane, and select your network name (*network1* in this example) from the **Host device** menu and click **Finish**.

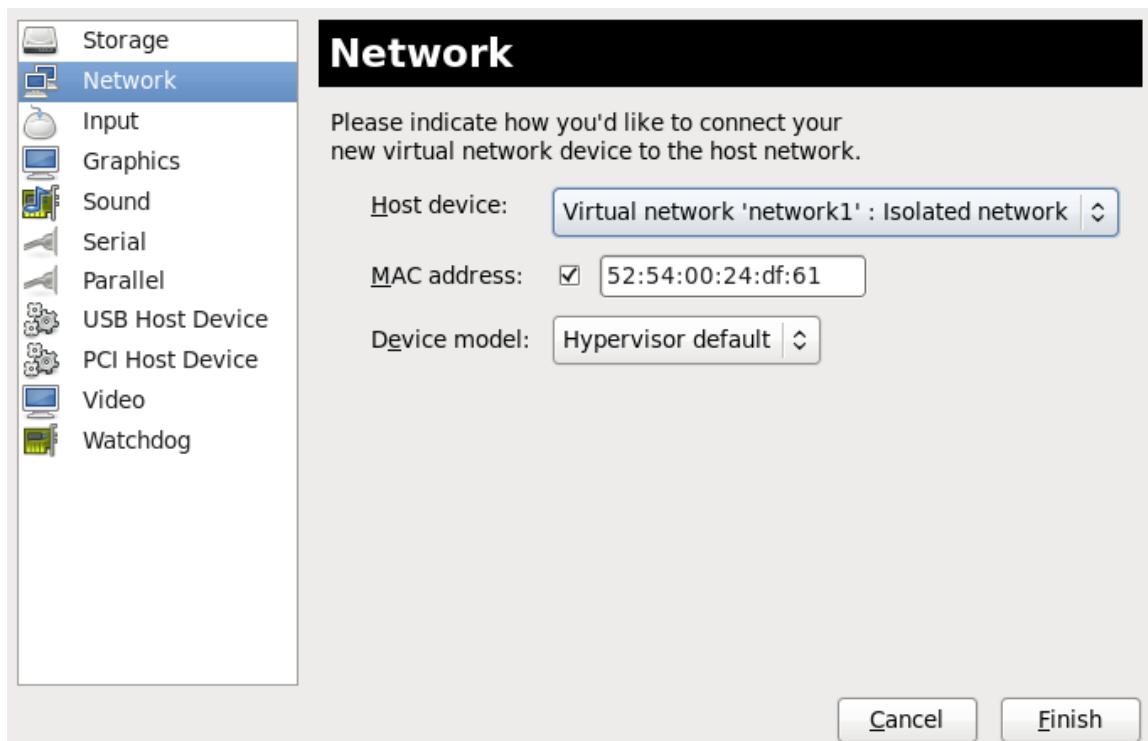


Figure 18.23. Select your network from the Add new virtual hardware window

- The new network is now displayed as a virtual network interface that will be presented to the guest upon launch.

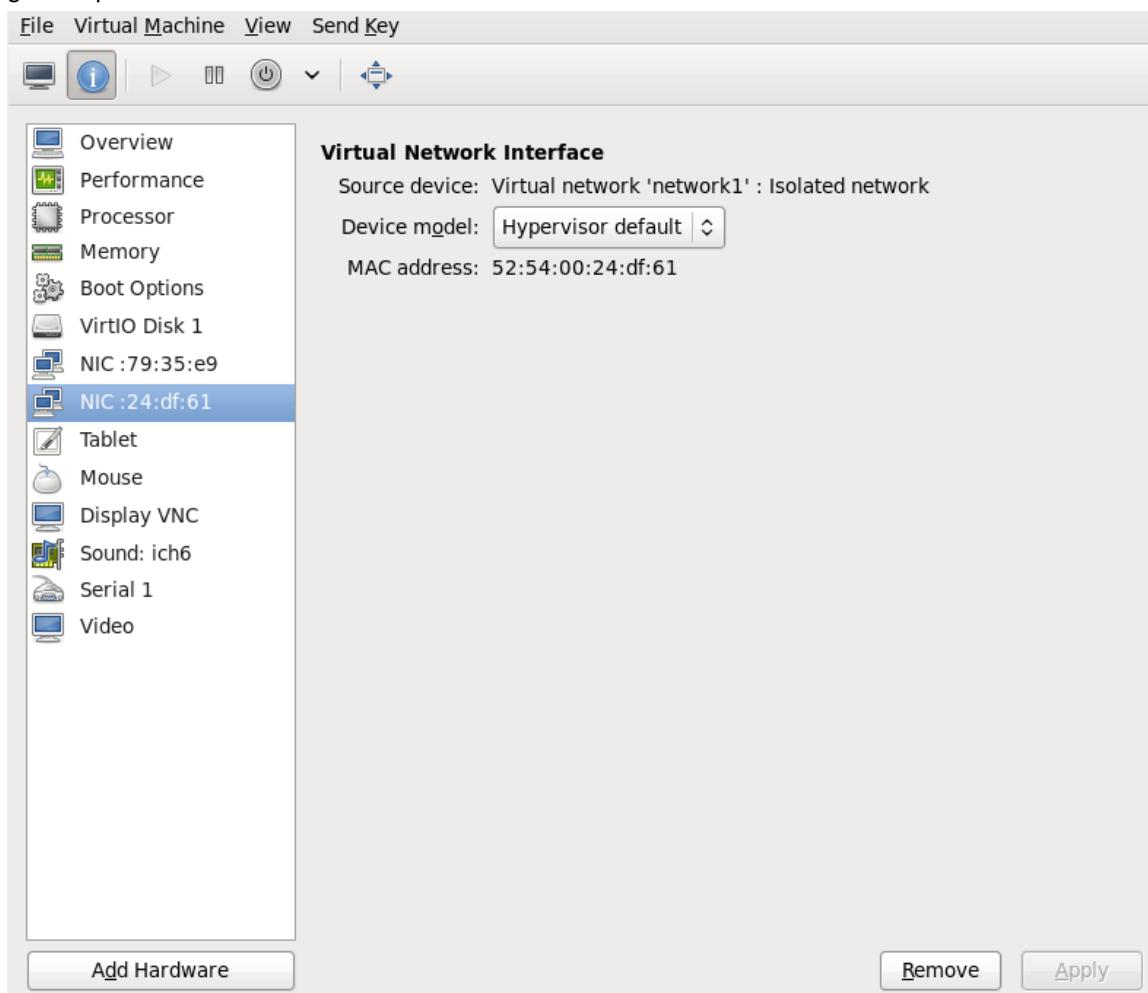


Figure 18.24. New network shown in guest hardware list

18.9. Directly attaching to physical interface

The instructions provided in this chapter will assist in the direct attachment of the virtual machine's NIC to the given physical interface of the host physical machine. This setup requires the Linux macvtap driver to be available. There are four modes that you can choose for the operation mode of the macvtap device, with 'vepa' being the default mode. Their behavior is as follows:

Physical interface delivery modes

vepa

All VMs' packets are sent to the external bridge. Packets whose destination is a VM on the same host physical machine as where the packet originates from are sent back to the host physical machine by the VEPA capable bridge (today's bridges are typically not VEPA capable).

bridge

Packets whose destination is on the same host physical machine as where they originate from are directly delivered to the target macvtap device. Both origin and destination devices need to be in bridge mode for direct delivery. If either one of them is in vepa mode, a VEPA capable bridge is required.

private

All packets are sent to the external bridge and will only be delivered to a target VM on the same host physical machine if they are sent through an external router or gateway and that device sends them back to the host physical machine. This procedure is followed if either the source or destination device is in private mode.

passthrough

This feature attaches a virtual function of a SRIOV capable NIC directly to a VM without losing the migration capability. All packets are sent to the VF/IF of the configured network device. Depending on the capabilities of the device additional prerequisites or limitations may apply; for example, on Linux this requires kernel 2.6.38 or newer.

Each of the four modes is configured by changing the domain xml file. Once this file is opened, change the mode setting as shown:

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
```

The network access of direct attached guest virtual machines can be managed by the hardware switch to which the physical interface of the host physical machine is connected to.

The interface can have additional parameters as shown below, if the switch is conforming to the IEEE 802.1Qbg standard. The parameters of the virtualport element are documented in more detail in the IEEE 802.1Qbg standard. The values are network specific and should be provided by the network administrator. In 802.1Qbg terms, the Virtual Station Interface (VSI) represents the virtual interface of a virtual machine.

Note that IEEE 802.1Qbg requires a non-zero value for the VLAN ID. Also if the switch is conforming to the IEEE 802.1Qbh standard, the values are network specific and should be provided by the network administrator.

Virtual Station Interface types

managerid

The VSI Manager ID identifies the database containing the VSI type and instance definitions. This is an integer value and the value 0 is reserved.

typeid

The VSI Type ID identifies a VSI type characterizing the network access. VSI types are typically managed by network administrator. This is an integer value.

typeidversion

The VSI Type Version allows multiple versions of a VSI Type. This is an integer value.

instanceid

The VSI Instance ID Identifier is generated when a VSI instance (i.e. a virtual interface of a virtual machine) is created. This is a globally unique identifier.

profileid

The profile ID contains the name of the port profile that is to be applied onto this interface. This name is resolved by the port profile database into the network parameters from the port profile, and those network parameters will be applied to this interface.

Each of the four types is configured by changing the domain xml file. Once this file is opened, change the mode setting as shown:

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0.2' mode='vepa' />
    <virtualport type="802.1Qbg">
      <parameters managerid="11" typeid="1193047" typeidversion="2"
instanceid="09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f" />
    </virtualport>
  </interface>
</devices>
```

The profile ID is shown here:

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private' />
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance' />
    </virtualport>
  </interface>
</devices>
...
```

18.10. Applying network filtering

This section provides an introduction to libvirt's network filters, their goals, concepts and XML format.

18.10.1. Introduction

The goal of the network filtering, is to enable administrators of a virtualized system to configure and enforce network traffic filtering rules on virtual machines and manage the parameters of network traffic that virtual machines are allowed to send or receive. The network traffic filtering rules are applied on the host physical machine when a virtual machine is started. Since the filtering rules cannot be circumvented from within the virtual machine, it makes them mandatory from the point of view of a virtual machine user.

From the point of view of the guest virtual machine, the network filtering system allows each virtual machine's network traffic filtering rules to be configured individually on a per interface basis. These rules are applied on the host physical machine when the virtual machine is started and can be modified while the virtual machine is running. The latter can be achieved by modifying the XML description of a network filter.

Multiple virtual machines can make use of the same generic network filter. When such a filter is modified, the network traffic filtering rules of all running virtual machines that reference this filter are updated. The machines that are not running will update on start.

As previously mentioned, applying network traffic filtering rules can be done on individual network interfaces that are configured for certain types of network configurations. Supported network types include:

- ▶ network
- ▶ ethernet -- must be used in bridging mode
- ▶ bridge

Example 18.1. An example of network filtering

The interface XML is used to reference a top-level filter. In the following example, the interface description references the filter clean-traffic.

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e' />
    <filterref filter='clean-traffic' />
  </interface>
</devices>
```

Network filters are written in XML and may either contain: references to other filters, rules for traffic filtering, or hold a combination of both. The above referenced filter clean-traffic is a filter that only contains references to other filters and no actual filtering rules. Since references to other filters can be used, a tree of filters can be built. The clean-traffic filter can be viewed using the command: # **virsh nwfilter-dumpxml clean-traffic**.

As previously mentioned, a single network filter can be referenced by multiple virtual machines. Since interfaces will typically have individual parameters associated with their respective traffic filtering rules, the rules described in a filter's XML can be generalized using variables. In this case, the variable name is used in the filter XML and the name and value are provided at the place where the filter is referenced.

Example 18.2. Description extended

In the following example, the interface description has been extended with the parameter IP and a dotted IP address as a value.

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1' />
    </filterref>
  </interface>
</devices>
```

In this particular example, the clean-traffic network traffic filter will be represented with the IP address parameter 10.0.0.1 and as per the rule dictates that all traffic from this interface will always be using 10.0.0.1 as the source IP address, which is one of the purpose of this particular filter.

18.10.2. Filtering chains

Filtering rules are organized in filter chains. These chains can be thought of as having a tree structure with packet filtering rules as entries in individual chains (branches).

Packets start their filter evaluation in the root chain and can then continue their evaluation in other chains, return from those chains back into the root chain or be dropped or accepted by a filtering rule in one of the traversed chains.

Libvirt's network filtering system automatically creates individual root chains for every virtual machine's network interface on which the user chooses to activate traffic filtering. The user may write filtering rules that are either directly instantiated in the root chain or may create protocol-specific filtering chains for efficient evaluation of protocol-specific rules.

The following chains exist:

- ▶ root
- ▶ mac
- ▶ stp (spanning tree protocol)
- ▶ vlan
- ▶ arp and rarp
- ▶ ipv4
- ▶ ipv6

Multiple chains evaluating the mac, stp, vlan, arp, rarp, ipv4, or ipv6 protocol can be created using the protocol name only as a prefix in the chain's name.

Example 18.3. ARP traffic filtering

This example allows chains with names arp-xyz or arp-test to be specified and have their ARP protocol packets evaluated in those chains.

The following filter XML shows an example of filtering ARP traffic in the arp chain.

```
<filter name='no-arp-spoofing' chain='arp' priority='-500'>
  <uuid>f88f1932-debf-4aa1-9fbe-f10d3aa4bc95</uuid>
  <rule action='drop' direction='out' priority='300'>
    <mac match='no' srcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='350'>
    <arp match='no' arpsrcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='400'>
    <arp match='no' arpsrcipaddr='$IP'/>
  </rule>
  <rule action='drop' direction='in' priority='450'>
    <arp opcode='Reply'/>
    <arp match='no' arpdstmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='in' priority='500'>
    <arp match='no' arpdstipaddr='$IP'/>
  </rule>
  <rule action='accept' direction='inout' priority='600'>
    <arp opcode='Request'/>
  </rule>
  <rule action='accept' direction='inout' priority='650'>
    <arp opcode='Reply'/>
  </rule>
  <rule action='drop' direction='inout' priority='1000'>
  </rule>
</filter>
```

The consequence of putting ARP-specific rules in the arp chain, rather than for example in the root chain, is that packets protocols other than ARP do not need to be evaluated by ARP protocol-specific rules. This improves the efficiency of the traffic filtering. However, one must then pay attention to only putting filtering rules for the given protocol into the chain since other rules will not be evaluated. For example, an IPv4 rule will not be evaluated in the ARP chain since IPv4 protocol packets will not traverse the ARP chain.

18.10.3. Filtering chain priorities

As previously mentioned, when creating a filtering rule, all chains are connected to the root chain. The order in which those chains are accessed is influenced by the priority of the chain. The following table shows the chains that can be assigned a priority and their default priorities.

Table 18.1. Filtering chain default priorities values

Chain (prefix)	Default priority
stp	-810
mac	-800
vlan	-750
ipv4	-700
ipv6	-600
arp	-500
rarp	-400



Note

A chain with a lower priority value is accessed before one with a higher value.

The chains listed in [Table 18.1, “Filtering chain default priorities values”](#) can be also be assigned custom priorities by writing a value in the range [-1000 to 1000] into the priority (XML) attribute in the filter node. [Section 18.10.2, “Filtering chains”](#) filter shows the default priority of -500 for arp chains, for example.

18.10.4. Usage of variables in filters

There are two variables that have been reserved for usage by the network traffic filtering subsystem: MAC and IP.

MAC is designated for the MAC address of the network interface. A filtering rule that references this variable will automatically be replaced with the MAC address of the interface. This works without the user having to explicitly provide the MAC parameter. Even though it is possible to specify the MAC parameter similar to the IP parameter above, it is discouraged since libvirt knows what MAC address an interface will be using.

The parameter **IP** represents the IP address that the operating system inside the virtual machine is expected to use on the given interface. The IP parameter is special in so far as the libvirt daemon will try to determine the IP address (and thus the IP parameter's value) that is being used on an interface if the parameter is not explicitly provided but referenced. For current limitations on IP address detection, consult the section on limitations [Section 18.10.12, “Limitations”](#) on how to use this feature and what to expect when using it. The XML file shown in [Section 18.10.2, “Filtering chains”](#) contains the filter **no-arp-spoofing**, which is an example of using a network filter XML to reference the MAC and IP variables.

Note that referenced variables are always prefixed with the character **\$**. The format of the value of a variable must be of the type expected by the filter attribute identified in the XML. In the above example, the **IP** parameter must hold a legal IP address in standard format. Failure to provide the correct structure will result in the filter variable not being replaced with a value and will prevent a virtual machine from starting or will prevent an interface from attaching when hotplugging is being used. Some of the types that are expected for each XML attribute are shown in the example [Example 18.4, “Sample variable types”](#).

Example 18.4. Sample variable types

As variables can contain lists of elements, (the variable IP can contain multiple IP addresses that are valid on a particular interface, for example), the notation for providing multiple elements for the IP variable is:

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e' />
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1' />
      <parameter name='IP' value='10.0.0.2' />
      <parameter name='IP' value='10.0.0.3' />
    </filterref>
  </interface>
</devices>
```

This XML file creates filters to enable multiple IP addresses per interface. Each of the IP addresses will result in a separate filtering rule. Therefore using the XML above and the the following rule, three individual filtering rules (one for each IP address) will be created:

```
<rule action='accept' direction='in' priority='500'>
  <tcp srchipaddr='$IP' />
</rule>
```

As it is possible to access individual elements of a variable holding a list of elements, a filtering rule like the following accesses the 2nd element of the variable **DSTPORTS**.

```
<rule action='accept' direction='in' priority='500'>
  <udp dstportstart='$DSTPORTS[1]' />
</rule>
```

Example 18.5. Using a variety of variables

As it is possible to create filtering rules that represent all possible combinations of rules from different lists using the notation `$VARIABLE[@<iterator id="x">]`. The following rule allows a virtual machine to receive traffic on a set of ports, which are specified in `DSTPORTS`, from the set of source IP address specified in `SRCPADDRESSES`. The rule generates all combinations of elements of the variable `DSTPORTS` with those of `SRCPADDRESSES` by using two independent iterators to access their elements.

```
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]' />
</rule>
```

Assign concrete values to `SRCPADDRESSES` and `DSTPORTS` as shown:

```
SRCPADDRESSES = [ 10.0.0.1, 11.1.2.3 ]
DSTPORTS = [ 80, 8080 ]
```

Assigning values to the variables using `$SRCPADDRESSES[@1]` and `$DSTPORTS[@2]` would then result in all combinations of addresses and ports being created as shown:

- ▶ 10.0.0.1, 80
- ▶ 10.0.0.1, 8080
- ▶ 11.1.2.3, 80
- ▶ 11.1.2.3, 8080

Accessing the same variables using a single iterator, for example by using the notation `$SRCPADDRESSES[@1]` and `$DSTPORTS[@1]`, would result in parallel access to both lists and result in the following combination:

- ▶ 10.0.0.1, 80
- ▶ 11.1.2.3, 8080



Note

`$VARIABLE` is short-hand for `$VARIABLE[@0]`. The former notation always assumes the role of iterator with `iterator id="0"` added as shown in the opening paragraph at the top of this section.

18.10.5. Automatic IP address detection and DHCP snooping

18.10.5.1. Introduction

The detection of IP addresses used on a virtual machine's interface is automatically activated if the variable `IP` is referenced but no value has been assigned to it. The variable `CTRL_IP_LEARNING` can be used to specify the IP address learning method to use. Valid values include: `any`, `dhcp`, or `none`.

The value `any` instructs libvirt to use any packet to determine the address in use by a virtual machine, which is the default setting if the variable `CTRL_IP_LEARNING` is not set. This method will only detect a single IP address per interface. Once a guest virtual machine's IP address has been detected, its IP network traffic will be locked to that address, if for example, IP address spoofing is prevented by one of its filters. In that case, the user of the VM will not be able to change the IP address on the interface inside the guest virtual machine, which would be considered IP address spoofing. When a guest virtual machine is migrated to another host physical machine or resumed after a suspend operation, the first packet sent by the guest virtual machine will again determine the IP address that the guest virtual

machine can use on a particular interface.

The value of ***dhcp*** instructs libvirt to only honor DHCP server-assigned addresses with valid leases. This method supports the detection and usage of multiple IP address per interface. When a guest virtual machine resumes after a suspend operation, any valid IP address leases are applied to its filters. Otherwise the guest virtual machine is expected to use DHCP to obtain a new IP addresses. When a guest virtual machine migrates to another physical host physical machine, the guest virtual machine is required to re-run the DHCP protocol.

If CTRL_IP_LEARNING is set to ***none***, libvirt does not do IP address learning and referencing IP without assigning it an explicit value is an error.

18.10.5.2. DHCP snooping

CTRL_IP_LEARNING=dhcp (DHCP snooping) provides additional anti-spoofing security, especially when combined with a filter allowing only trusted DHCP servers to assign IP addresses. To enable this, set the variable **DHCPSERVER** to the IP address of a valid DHCP server and provide filters that use this variable to filter incoming DHCP responses.

When DHCP snooping is enabled and the DHCP lease expires, the guest virtual machine will no longer be able to use the IP address until it acquires a new, valid lease from a DHCP server. If the guest virtual machine is migrated, it must get a new valid DHCP lease to use an IP address (e.g., by bringing the VM interface down and up again).



Note

Automatic DHCP detection listens to the DHCP traffic the guest virtual machine exchanges with the DHCP server of the infrastructure. To avoid denial-of-service attacks on libvirt, the evaluation of those packets is rate-limited, meaning that a guest virtual machine sending an excessive number of DHCP packets per second on an interface will not have all of those packets evaluated and thus filters may not get adapted. Normal DHCP client behavior is assumed to send a low number of DHCP packets per second. Further, it is important to setup appropriate filters on all guest virtual machines in the infrastructure to avoid them being able to send DHCP packets. Therefore guest virtual machines must either be prevented from sending UDP and TCP traffic from port 67 to port 68 or the DHCPSERVER variable should be used on all guest virtual machines to restrict DHCP server messages to only be allowed to originate from trusted DHCP servers. At the same time anti-spoofing prevention must be enabled on all guest virtual machines in the subnet.

Example 18.6. Activating IPs for DHCP snooping

The following XML provides an example for the activation of IP address learning using the DHCP snooping method:

```
<interface type='bridge'>
  <source bridge='virbr0' />
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp' />
  </filterref>
</interface>
```

18.10.6. Reserved Variables

[Table 18.2, “Reserved variables”](#) shows the variables that are considered reserved and are used by libvirt:

Table 18.2. Reserved variables

Variable Name	Definition
MAC	The MAC address of the interface
IP	The list of IP addresses in use by an interface
IPV6	Not currently implemented: the list of IPV6 addresses in use by an interface
DHCPSERVER	The list of IP addresses of trusted DHCP servers
DHCPSERVERV6	Not currently implemented: The list of IPv6 addresses of trusted DHCP servers
CTRL_IP_LEARNING	The choice of the IP address detection mode

18.10.7. Element and attribute overview

The root element required for all network filters is named `<filter>` with two possible attributes. The `name` attribute provides a unique name of the given filter. The `chain` attribute is optional but allows certain filters to be better organized for more efficient processing by the firewall subsystem of the underlying host physical machine. Currently the system only supports the following chains: `root`, `ipv4`, `ipv6`, `arp` and `rarp`.

18.10.8. References to other filters

Any filter may hold references to other filters. Individual filters may be referenced multiple times in a filter tree but references between filters must not introduce loops.

Example 18.7. An Example of a clean traffic filter

The following shows the XML of the clean-traffic network filter referencing several other filters.

```
<filter name='clean-traffic'>
  <uuid>6ef53069-ba34-94a0-d33d-17751b9b8cb1</uuid>
  <filterref filter='no-mac-spoofing'/>
  <filterref filter='no-ip-spoofing'/>
  <filterref filter='allow-incoming-ipv4'/>
  <filterref filter='no-arp-spoofing'/>
  <filterref filter='no-other-12-traffic'/>
  <filterref filter='qemu-announce-self'/>
</filter>
```

To reference another filter, the XML node `filterref` needs to be provided inside a `filter` node. This node must have the attribute `filter` whose value contains the name of the filter to be referenced.

New network filters can be defined at any time and may contain references to network filters that are not known to libvirt, yet. However, once a virtual machine is started or a network interface referencing a filter is to be hotplugged, all network filters in the filter tree must be available. Otherwise the virtual machine will not start or the network interface cannot be attached.

18.10.9. Filter rules

The following XML shows a simple example of a network traffic filter implementing a rule to drop traffic if the IP address (provided through the value of the variable `IP`) in an outgoing IP packet is not the expected one, thus preventing IP address spoofing by the VM.

Example 18.8. Example of network traffic filtering

```
<filter name='no-ip-spoofing' chain='ipv4'>
  <uuid>fce8ae33-e69e-83bf-262e-30786c1f8072</uuid>
  <rule action='drop' direction='out' priority='500'>
    <ip match='no' srcipaddr='$IP' />
  </rule>
</filter>
```

The traffic filtering rule starts with the rule node. This node may contain up to three of the following attributes:

- ▶ action is mandatory can have the following values:
 - drop (matching the rule silently discards the packet with no further analysis)
 - reject (matching the rule generates an ICMP reject message with no further analysis)
 - accept (matching the rule accepts the packet with no further analysis)
 - return (matching the rule passes this filter, but returns control to the calling filter for further analysis)
 - continue (matching the rule goes on to the next rule for further analysis)
- ▶ direction is mandatory can have the following values:
 - in for incoming traffic
 - out for outgoing traffic
 - inout for incoming and outgoing traffic
- ▶ priority is optional. The priority of the rule controls the order in which the rule will be instantiated relative to other rules. Rules with lower values will be instantiated before rules with higher values. Valid values are in the range of -1000 to 1000. If this attribute is not provided, priority 500 will be assigned by default. Note that filtering rules in the root chain are sorted with filters connected to the root chain following their priorities. This allows to interleave filtering rules with access to filter chains. Refer to [Section 18.10.3, “Filtering chain priorities”](#) for more information.
- ▶ statematch is optional. Possible values are '0' or 'false' to turn the underlying connection state matching off. The default setting is 'true' or 1

For more information see [Section 18.10.11, “Advanced Filter Configuration Topics”](#).

The above example [Example 18.7, “An Example of a clean traffic filter”](#) indicates that the traffic of **type ip** will be associated with the chain **ipv4** and the rule will have **priority=500**. If for example another filter is referenced whose traffic of **type ip** is also associated with the chain **ipv4** then that filter's rules will be ordered relative to the **priority=500** of the shown rule.

A rule may contain a single rule for filtering of traffic. The above example shows that traffic of type ip is to be filtered.

18.10.10. Supported protocols

The following sections list and give some details about the protocols that are supported by the network filtering subsystem. This type of traffic rule is provided in the rule node as a nested node. Depending on the traffic type a rule is filtering, the attributes are different. The above example showed the single attribute **srcipaddr** that is valid inside the ip traffic filtering node. The following sections show what attributes are valid and what type of data they are expecting. The following datatypes are available:

- ▶ **UINT8** : 8 bit integer; range 0-255
- ▶ **UINT16**: 16 bit integer; range 0-65535
- ▶ **MAC_ADDR**: MAC address in dotted decimal format, i.e., 00:11:22:33:44:55
- ▶ **MAC_MASK**: MAC address mask in MAC address format, i.e., FF:FF:FF:FC:00:00
- ▶ **IP_ADDR**: IP address in dotted decimal format, i.e., 10.1.2.3

- ▶ **IP_MASK**: IP address mask in either dotted decimal format (255.255.248.0) or CIDR mask (0-32)
- ▶ **IPV6_ADDR**: IPv6 address in numbers format, i.e., FFFF::1
- ▶ **IPV6_MASK**: IPv6 mask in numbers format (FFFF:FFFF:FC00::) or CIDR mask (0-128)
- ▶ **STRING**: A string
- ▶ **BOOLEAN**: 'true', 'yes', '1' or 'false', 'no', '0'
- ▶ **IPSETFLAGS**: The source and destination flags of the ipset described by up to 6 'src' or 'dst' elements selecting features from either the source or destination part of the packet header; example: src,src,dst. The number of 'selectors' to provide here depends on the type of ipset that is referenced

Every attribute except for those of type **IP_MASK** or **IPV6_MASK** can be negated using the match attribute with value **no**. Multiple negated attributes may be grouped together. The following XML fragment shows such an example using abstract attributes.

```
[...]
<rule action='drop' direction='in'>
  <protocol match='no' attribute1='value1' attribute2='value2' />
  <protocol attribute3='value3' />
</rule>
[...]
```

Rules behave evaluate the rule as well as look at it logically within the boundaries of the given protocol attributes. Thus, if a single attribute's value does not match the one given in the rule, the whole rule will be skipped during the evaluation process. Therefore, in the above example incoming traffic will only be dropped if: the protocol property **attribute1** does not match both **value1** and the protocol property **attribute2** does not match **value2** and the protocol property **attribute3** matches **value3**.

18.10.10.1. MAC (Ethernet)

Protocol ID: mac

Rules of this type should go into the root chain.

Table 18.3. MAC protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
protocolid	UINT16 (0x600-0xffff), STRING	Layer 3 protocol ID. Valid strings include [arp, rarp, ipv4, ipv6]
comment	STRING	text string up to 256 characters

The filter can be written as such:

```
[...]
<mac match='no' srcmacaddr='$MAC' />
[...]
```

18.10.10.2. VLAN (802.1Q)

Protocol ID: vlan

Rules of this type should go either into the root or vlan chain.

Table 18.4. VLAN protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
vlan-id	UINT16 (0x0-0xffff, 0 - 4095)	VLAN ID
encap-protocol	UINT16 (0x03c-0xffff), String	Encapsulated layer 3 protocol ID, valid strings are arp, ipv4, ipv6
comment	STRING	text string up to 256 characters

18.10.10.3. STP (Spanning Tree Protocol)

Protocol ID: stp

Rules of this type should go either into the root or stp chain.

Table 18.5. STP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
type	UINT8	Bridge Protocol Data Unit (BPDU) type
flags	UINT8	BPDU flagdstmacmask
root-priority	UINT16	Root priority range start
root-priority-hi	UINT16 (0x0-0xffff, 0 - 4095)	Root priority range end
root-address	MAC_ADDRESS	root MAC Address
root-address-mask	MAC_MASK	root MAC Address mask
root-cost	UINT32	Root path cost (range start)
root-cost-hi	UINT32	Root path cost range end
sender-priority-hi	UINT16	Sender priority range end
sender-address	MAC_ADDRESS	BPDU sender MAC address
sender-address-mask	MAC_MASK	BPDU sender MAC address mask
port	UINT16	Port identifier (range start)
port_hi	UINT16	Port identifier range end
msg-age	UINT16	Message age timer (range start)
msg-age-hi	UINT16	Message age timer range end
max-age-hi	UINT16	Maximum age time range end
hello-time	UINT16	Hello time timer (range start)
hello-time-hi	UINT16	Hello time timer range end
forward-delay	UINT16	Forward delay (range start)
forward-delay-hi	UINT16	Forward delay range end
comment	STRING	text string up to 256 characters

18.10.10.4. ARP/RARP

Protocol ID: arp or rarp

Rules of this type should either go into the root or arp/rarp chain.

Table 18.6. ARP and RARP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
hwtype	UINT16	Hardware type
protcoltype	UINT16	Protocol type
opcode	UINT16, STRING	Opcode valid strings are: Request, Reply, Request_Reverse, Reply_Reverse, DRARP_Request, DRARP_Reply, DRARP_Error, InARP_Request, ARP_NAK
arpsrcmacaddr	MAC_ADDR	Source MAC address in ARP/RARP packet
arpdstmacaddr	MAC _ADDR	Destination MAC address in ARP/RARP packet
arpsrcipaddr	IP_ADDR	Source IP address in ARP/RARP packet
arpdstipaddr	IP_ADDR	Destination IP address in ARP/RARP packet
gratututous	BOOLEAN	Boolean indicating whether to check for a gratuitous ARP packet
comment	STRING	text string up to 256 characters

18.10.10.5. IPv4

Protocol ID: ip

Rules of this type should either go into the root or ipv4 chain.

Table 18.7. IPv4 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
protocol	UINT8, STRING	Layer 4 protocol identifier. Valid strings for protocol are: tcp, udp, udplite, esp, ah, icmp, igmp, sctp
srcportstart	UINT16	Start of range of valid source ports; requires protocol
srcportend	UINT16	End of range of valid source ports; requires protocol
dstportstart	UNIT16	Start of range of valid destination ports; requires protocol
dstportend	UNIT16	End of range of valid destination ports; requires protocol
comment	STRING	text string up to 256 characters

18.10.10.6. IPv6

Protocol ID: ipv6

Rules of this type should either go into the root or ipv6 chain.

Table 18.8. IPv6 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
protocol	UINT8, STRING	Layer 4 protocol identifier. Valid strings for protocol are: tcp, udp, udplite, esp, ah, icmpv6, sctp
scrportstart	UNIT16	Start of range of valid source ports; requires protocol
srcportend	UNIT16	End of range of valid source ports; requires protocol
dstportstart	UNIT16	Start of range of valid destination ports; requires protocol
dstportend	UNIT16	End of range of valid destination ports; requires protocol
comment	STRING	text string up to 256 characters

18.10.10.7. TCP/UDP/SCTP

Protocol ID: tcp, udp, sctp

The chain parameter is ignored for this type of traffic and should either be omitted or set to root..

Table 18.9. TCP/UDP/SCTP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
scripto	IP_ADDR	Start of range of source IP address
srcipfrom	IP_ADDR	End of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
scrportstart	UNIT16	Start of range of valid source ports; requires protocol
srcportend	UINT16	End of range of valid source ports; requires protocol
dstportstart	UNIT16	Start of range of valid destination ports; requires protocol
dstportend	UNIT16	End of range of valid destination ports; requires protocol
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,I,NVALID or NONE
flags	STRING	TCP-only: format of mask/flags with mask and flags each being a comma separated list of SYN,ACK,URG,PSH,FIN,RST or NONE or ALL
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSET FLAGS	flags for the IPSet; requires ipset attribute

18.10.10.8. ICMP

Protocol ID: icmp

Note: The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.10. ICMP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to the MAC address of the sender
dstmacaddr	MAD_ADDR	MAC address of the destination
dstmacmask	MAC_MASK	Mask applied to the MAC address of the destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
type	UNIT16	ICMP type
code	UNIT16	ICMP code
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,I_NVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSET FLAGS	flags for the IPSet; requires ipset attribute

18.10.10.9. IGMP, ESP, AH, UDPLITE, 'ALL'

Protocol ID: igmp, esp, ah, udplite, all

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.11. IGMP, ESP, AH, UDPLITE, 'ALL'

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to the MAC address of the sender
dstmacaddr	MAD_ADDR	MAC address of the destination
dstmacmask	MAC_MASK	Mask applied to the MAC address of the destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.10.10.10. TCP/UDP/SCTP over IPV6

Protocol ID: tcp-ipv6, udp-ipv6, sctp-ipv6

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.12. TCP, UDP, SCTP over IPv6 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
srcportstart	UINT16	Start of range of valid source ports
srcportend	UINT16	End of range of valid source ports
dstportstart	UINT16	Start of range of valid destination ports
dstportend	UINT16	End of range of valid destination ports
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,I NVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.10.10.11. ICMPv6

Protocol ID: icmpv6

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.13. ICMPv6 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
type	UINT16	ICMPv6 type
code	UINT16	ICMPv6 code
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,I_NVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.10.10.12. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv6

Protocol ID: igmp-ipv6, esp-ipv6, ah-ipv6, udplite-ipv6, all-ipv6

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.14. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,I_NVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSET FLAGS	flags for the IPSet; requires ipset attribute

18.10.11. Advanced Filter Configuration Topics

The following sections discuss advanced filter configuration topics.

18.10.11.1. Connection tracking

The network filtering subsystem (on Linux) makes use of the connection tracking support of IP tables. This helps in enforcing the directionality of network traffic (state match) as well as counting and limiting the number of simultaneous connections towards a guest virtual machine. As an example, if a guest virtual machine has TCP port 8080 open as a server, clients may connect to the guest virtual machine on port 8080. Connection tracking and enforcement of directionality then prevents the guest virtual machine from initiating a connection from (TCP client) port 8080 to the host physical machine back to a remote host physical machine. More importantly, tracking helps to prevent remote attackers from establishing a connection back to a guest virtual machine. For example, if the user inside the guest virtual machine established a connection to port 80 on an attacker site, then the attacker will not be able to initiate a connection from TCP port 80 back towards the guest virtual machine. By default the connection state match that enables connection tracking and then enforcement of directionality of traffic is turned on.

Example 18.9. XML example for turning off connections to the TCP port

The following shows an example XML fragment where this feature has been turned off for incoming connections to TCP port 12345.

```
[...]
<rule direction='in' action='accept' statematch='false'>
    <cp dstportstart='12345' />
</rule>
[...]
```

This now allows incoming traffic to TCP port 12345, but would also enable the initiation from (client) TCP port 12345 within the VM, which may or may not be desirable.

18.10.11.2. Limiting Number of Connections

To limit the number of connections a guest virtual machine may establish, a rule must be provided that sets a limit of connections for a given type of traffic. If for example a VM is supposed to be allowed to only ping one other IP address at a time and is supposed to have only one active incoming ssh connection at a time.

Example 18.10. XML sample file that sets limits to connections

The following XML fragment can be used to limit connections

```
[...]
<rule action='drop' direction='in' priority='400'>
    <tcp connlimit-above='1' />
</rule>
<rule action='accept' direction='in' priority='500'>
    <tcp dstportstart='22' />
</rule>
<rule action='drop' direction='out' priority='400'>
    <icmp connlimit-above='1' />
</rule>
<rule action='accept' direction='out' priority='500'>
    <icmp />
</rule>
<rule action='accept' direction='out' priority='500'>
    <udp dstportstart='53' />
</rule>
<rule action='drop' direction='inout' priority='1000'>
    <all />
</rule>
[...]
```



Note

Limitation rules must be listed in the XML prior to the rules for accepting traffic. According to the XML file in [Example 18.10, “XML sample file that sets limits to connections”](#), an additional rule for allowing DNS traffic sent to port 22 go out the guest virtual machine, has been added to avoid ssh sessions not getting established for reasons related to DNS lookup failures by the ssh daemon. Leaving this rule out may result in the ssh client hanging unexpectedly as it tries to connect. Additional caution should be used in regards to handling timeouts related to tracking of traffic. An ICMP ping that the user may have terminated inside the guest virtual machine may have a long timeout in the host physical machine's connection tracking system and will therefore not allow another ICMP ping to go through.

The best solution is to tune the timeout in the host physical machine's **sysfs** with the following command:
`# echo 3 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout`. This command sets the ICMP connection tracking timeout to 3 seconds. The effect of this is that once one ping is terminated, another one can start after 3 seconds.

If for any reason the guest virtual machine has not properly closed its TCP connection, the connection to be held open for a longer period of time, especially if the TCP timeout value was set for a large amount of time on the host physical machine. In addition, any idle connection may result in a time out in the connection tracking system which can be re-activated once packets are exchanged.

However, if the limit is set too low, newly initiated connections may force an idle connection into TCP backoff. Therefore, the limit of connections should be set rather high so that fluctuations in new TCP connections don't cause odd traffic behavior in relation to idle connections.

18.10.11.3. Command line tools

virsh has been extended with life-cycle support for network filters. All commands related to the network filtering subsystem start with the prefix **nwfilter**. The following commands are available:

- ▶ **nwfilter-list** : lists UUIDs and names of all network filters
- ▶ **nwfilter-define** : defines a new network filter or updates an existing one (must supply a name)
- ▶ **nwfilter-undefine** : deletes a specified network filter (must supply a name). Do not delete a network filter currently in use.
- ▶ **nwfilter-dumpxml** : displays a specified network filter (must supply a name)
- ▶ **nwfilter-edit** : edits a specified network filter (must supply a name)

18.10.11.4. Pre-existing network filters

The following is a list of example network filters that are automatically installed with libvirt:

Table 18.15. ICMPv6 protocol types

Command Name	Description
no-arp-spoofing	Prevents a guest virtual machine from spoofing ARP traffic; this filter only allows ARP request and reply messages and enforces that those packets contain the MAC and IP addresses of the guest virtual machine.
allow-dhcp	Allows a guest virtual machine to request an IP address via DHCP (from any DHCP server)
allow-dhcp-server	Allows a guest virtual machine to request an IP address from a specified DHCP server. The dotted decimal IP address of the DHCP server must be provided in a reference to this filter. The name of the variable must be DHCPSERVER .
no-ip-spoofing	Prevents a guest virtual machine from sending IP packets with a source IP address different from the one inside the packet.
no-ip-multicast	Prevents a guest virtual machine from sending IP multicast packets.
clean-traffic	Prevents MAC, IP and ARP spoofing. This filter references several other filters as building blocks.

These filters are only building blocks and require a combination with other filters to provide useful network traffic filtering. The most used one in the above list is the **clean-traffic** filter. This filter itself can for example be combined with the **no-ip-multicast** filter to prevent virtual machines from sending IP multicast traffic on top of the prevention of packet spoofing.

18.10.11.5. Writing your own filters

Since libvirt only provides a couple of example networking filters, you may consider writing your own. When planning on doing so there are a couple of things you may need to know regarding the network filtering subsystem and how it works internally. Certainly you also have to know and understand the protocols very well that you want to be filtering on so that no further traffic than what you want can pass and that in fact the traffic you want to allow does pass.

The network filtering subsystem is currently only available on Linux host physical machines and only works for Qemu and KVM type of virtual machines. On Linux, it builds upon the support for ebtables, iptables and ip6tables and makes use of their features. Considering the list found in [Section 18.10.10, “Supported protocols”](#) the following protocols can be implemented using ebtables:

- ▶ mac
- ▶ stp (spanning tree protocol)
- ▶ vlan (802.1Q)
- ▶ arp, rarp
- ▶ ipv4
- ▶ ipv6

Any protocol that runs over IPv4 is supported using iptables, those over IPv6 are implemented using ip6tables.

Using a Linux host physical machine, all traffic filtering rules created by libvirt's network filtering subsystem first passes through the filtering support implemented by ebtables and only afterwards through iptables or ip6tables filters. If a filter tree has rules with the protocols including: mac, stp, vlan arp, rarp, ipv4, or ipv6; the ebttable rules and values listed will automatically be used first.

Multiple chains for the same protocol can be created. The name of the chain must have a prefix of one of

the previously enumerated protocols. To create an additional chain for handling of ARP traffic, a chain with name arp-test, can for example be specified.

As an example, it is possible to filter on UDP traffic by source and destination ports using the ip protocol filter and specifying attributes for the protocol, source and destination IP addresses and ports of UDP packets that are to be accepted. This allows early filtering of UDP traffic with ebtables. However, once an IP or IPv6 packet, such as a UDP packet, has passed the ebtables layer and there is at least one rule in a filter tree that instantiates iptables or ip6tables rules, a rule to let the UDP packet pass will also be necessary to be provided for those filtering layers. This can be achieved with a rule containing an appropriate `udp` or `udp-ipv6` traffic filtering node.

Example 18.11. Creating a custom filter

Suppose a filter is needed to fulfill the following list of requirements:

- ▶ prevents a VM's interface from MAC, IP and ARP spoofing
- ▶ opens only TCP ports 22 and 80 of a VM's interface
- ▶ allows the VM to send ping traffic from an interface but not let the VM be pinged on the interface
- ▶ allows the VM to do DNS lookups (UDP towards port 53)

The requirement to prevent spoofing is fulfilled by the existing ***clean-traffic*** network filter, thus the way to do this is to reference it from a custom filter.

To enable traffic for TCP ports 22 and 80, two rules are added to enable this type of traffic. To allow the guest virtual machine to send ping traffic a rule is added for ICMP traffic. For simplicity reasons, general ICMP traffic will be allowed to be initiated from the guest virtual machine, and will not be specified to ICMP echo request and response messages. All other traffic will be prevented to reach or be initiated by the guest virtual machine. To do this a rule will be added that drops all other traffic. Assuming the guest virtual machine is called **test** and the interface to associate our filter with is called **eth0**, a filter is created named **test-eth0**.

The result of these considerations is the following network filter XML:

```
<filter name='test-eth0'>
    <!-- This rule references the clean traffic filter to prevent MAC, IP and ARP
        spoofing. By not providing an IP address parameter, libvirt will detect the IP
        address the guest virtual machine is using. -->
    <filterref filter='clean-traffic'/>

    <!-- This rule enables TCP ports 22 (ssh) and 80 (http) to be reachable -->
    <rule action='accept' direction='in'>
        <tcp dstportstart='22' />
    </rule>

    <rule action='accept' direction='in'>
        <tcp dstportstart='80' />
    </rule>

    <!-- This rule enables general ICMP traffic to be initiated by the guest
        virtual machine including ping traffic -->
    <rule action='accept' direction='out'>
        <icmp />
    </rule>>

    <!-- This rule enables outgoing DNS lookups using UDP -->
    <rule action='accept' direction='out'>
        <udp dstportstart='53' />
    </rule>

    <!-- This rule drops all other traffic -->
    <rule action='drop' direction='inout'>
        <all />
    </rule>

</filter>
```

18.10.11.6. Sample custom filter

Although one of the rules in the above XML contains the IP address of the guest virtual machine as either a source or a destination address, the filtering of the traffic works correctly. The reason is that whereas the rule's evaluation occurs internally on a per-interface basis, the rules are additionally

evaluated based on which (tap) interface has sent or will receive the packet, rather than what their source or destination IP address may be.

Example 18.12. Sample XML for network interface descriptions

An XML fragment for a possible network interface description inside the domain XML of the test guest virtual machine could then look like this:

```
[...]
<interface type='bridge'>
    <source bridge='mybridge' />
    <filterref filter='test-eth0' />
</interface>
[...]
```

To more strictly control the ICMP traffic and enforce that only ICMP echo requests can be sent from the guest virtual machine and only ICMP echo responses be received by the guest virtual machine, the above ICMP rule can be replaced with the following two rules:

```
<!-- enable outgoing ICMP echo requests-->
<rule action='accept' direction='out'>
    <icmp type='8' />
</rule>
```

```
<!-- enable incoming ICMP echo replies-->
<rule action='accept' direction='in'>
    <icmp type='0' />
</rule>
```

Example 18.13. Second example custom filter

This example demonstrates how to build a similar filter as in the example above, but extends the list of requirements with an ftp server located inside the guest virtual machine. The requirements for this filter are:

- ▶ prevents a guest virtual machine's interface from MAC, IP, and ARP spoofing
- ▶ opens only TCP ports 22 and 80 in a guest virtual machine's interface
- ▶ allows the guest virtual machine to send ping traffic from an interface but does not allow the guest virtual machine to be pinged on the interface
- ▶ allows the guest virtual machine to do DNS lookups (UDP towards port 53)
- ▶ enables the ftp server (in active mode) so it can run inside the guest virtual machine

The additional requirement of allowing an FTP server to be run inside the guest virtual machine maps into the requirement of allowing port 21 to be reachable for FTP control traffic as well as enabling the guest virtual machine to establish an outgoing TCP connection originating from the guest virtual machine's TCP port 20 back to the FTP client (FTP active mode). There are several ways of how this filter can be written and two possible solutions are included in this example.

The first solution makes use of the state attribute of the TCP protocol that provides a hook into the connection tracking framework of the Linux host physical machine. For the guest virtual machine-initiated FTP data connection (FTP active mode) the RELATED state is used to enable detection that the guest virtual machine-initiated FTP data connection is a consequence of (or 'has a relationship with') an existing FTP control connection, thereby allowing it to pass packets through the firewall. The RELATED state, however, is only valid for the very first packet of the outgoing TCP connection for the FTP data path. Afterwards, the state is ESTABLISHED, which then applies equally to the incoming and outgoing direction. All this is related to the FTP data traffic originating from TCP port 20 of the guest virtual machine. This then leads to the following solution:

```

<filter name='test-eth0'>
  <!-- This filter (eth0) references the clean traffic filter to prevent MAC,
  IP, and ARP spoofing. By not providing an IP address parameter, libvirt will
  detect the IP address the guest virtual machine is using. -->
  <filterref filter='clean-traffic'/>

  <!-- This rule enables TCP port 21 (FTP-control) to be reachable -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21' />
  </rule>

  <!-- This rule enables TCP port 20 for guest virtual machine-initiated FTP
  data connection related to an existing FTP control connection -->
  <rule action='accept' direction='out'>
    <tcp srcportstart='20' state='RELATED,ESTABLISHED' />
  </rule>

  <!-- This rule accepts all packets from a client on the FTP data connection -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='20' state='ESTABLISHED' />
  </rule>

  <!-- This rule enables TCP port 22 (SSH) to be reachable -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22' />
  </rule>

  <!-- This rule enables TCP port 80 (HTTP) to be reachable -->
  <rule action='accept' direction='in'>
    <tcp dstportstart='80' />
  </rule>

  <!-- This rule enables general ICMP traffic to be initiated by the guest
  virtual machine, including ping traffic -->
  <rule action='accept' direction='out'>
    <icmp />
  </rule>

  <!-- This rule enables outgoing DNS lookups using UDP -->
  <rule action='accept' direction='out'>
    <udp dstportstart='53' />
  </rule>

  <!-- This rule drops all other traffic -->
  <rule action='drop' direction='inout'>
    <all />
  </rule>

</filter>
```

Before trying out a filter using the RELATED state, you have to make sure that the appropriate connection tracking module has been loaded into the host physical machine's kernel. Depending on the version of the kernel, you must run either one of the following two commands before the FTP connection with the guest virtual machine is established:

- ▶ **#modprobe nf_conntrack_ftp** - where available OR
- ▶ **#modprobe ip_conntrack_ftp** if above is not available

If protocols other than FTP are used in conjunction with the RELATED state, their corresponding module must be loaded. Modules are available for the protocols: ftp, tftp, irc, sip, sctp, and amanda.

The second solution makes use of the state flags of connections more than the previous solution did. This solution takes advantage of the fact that the NEW state of a connection is valid when the very

first packet of a traffic flow is detected. Subsequently, if the very first packet of a flow is accepted, the flow becomes a connection and thus enters into the ESTABLISHED state. Therefore a general rule can be written for allowing packets of ESTABLISHED connections to reach the guest virtual machine or be sent by the guest virtual machine. This is done writing specific rules for the very first packets identified by the NEW state and dictates the ports that the data is acceptable. All packets meant for ports that are not explicitly accepted are dropped, thus not reaching an ESTABLISHED state. Any subsequent packets sent from that port are dropped as well.

```

<filter name='test-eth0'>
    <!-- This filter references the clean traffic filter to prevent MAC, IP and
    ARP spoofing. By not providing and IP address parameter, libvirt will detect the
    IP address the VM is using. -->
    <filterref filter='clean-traffic'/>

    <!-- This rule allows the packets of all previously accepted connections to
    reach the guest virtual machine -->
    <rule action='accept' direction='in'>
        <all state='ESTABLISHED' />
    </rule>

    <!-- This rule allows the packets of all previously accepted and related
    connections be sent from the guest virtual machine -->
    <rule action='accept' direction='out'>
        <all state='ESTABLISHED,RELATED' />
    </rule>

    <!-- This rule enables traffic towards port 21 (FTP) and port 22 (SSH)-->
    <rule action='accept' direction='in'>
        <tcp dstportstart='21' dstportend='22' state='NEW' />
    </rule>

    <!-- This rule enables traffic towards port 80 (HTTP) -->
    <rule action='accept' direction='in'>
        <tcp dstportstart='80' state='NEW' />
    </rule>

    <!-- This rule enables general ICMP traffic to be initiated by the guest
    virtual machine, including ping traffic -->
    <rule action='accept' direction='out'>
        <icmp state='NEW' />
    </rule>

    <!-- This rule enables outgoing DNS lookups using UDP -->
    <rule action='accept' direction='out'>
        <udp dstportstart='53' state='NEW' />
    </rule>

    <!-- This rule drops all other traffic -->
    <rule action='drop' direction='inout'>
        <all />
    </rule>

</filter>
```

18.10.12. Limitations

The following is a list of the currently known limitations of the network filtering subsystem.

- ▶ VM migration is only supported if the whole filter tree that is referenced by a guest virtual machine's top level filter is also available on the target host physical machine. The network filter **clean-traffic** for example should be available on all libvirt installations and thus enable migration of guest virtual machines that reference this filter. To assure version compatibility is not a problem

- make sure you are using the most current version of libvirt by updating the package regularly.
- ▶ Migration must occur between libvirt installations of version 0.8.1 or later in order not to lose the network traffic filters associated with an interface.
 - ▶ VLAN (802.1Q) packets, if sent by a guest virtual machine, cannot be filtered with rules for protocol IDs arp, rarp, ipv4 and ipv6. They can only be filtered with protocol IDs, MAC and VLAN. Therefore, the example filter clean-traffic [Example 18.1, “An example of network filtering”](#) will not work as expected.

Chapter 19. qemu-kvm Whitelist

19.1. Introduction

The primary objective of this QEMU-KVM whitelist is to provide a complete list of the supported options of the **qemu-kvm** utility used as an emulator and a virtualizer in Red Hat Enterprise Linux 6. This is a comprehensive summary of the supported options. Red Hat Enterprise Linux 6 uses KVM as an underlying virtualization technology. The machine emulator and virtualizer used is a modified version of QEMU called qemu-kvm. This version does not support all configuration options of the original QEMU and it also adds some additional options.



Note

This chapter lists only the supported options of the **qemu-kvm** utility. Options **not** listed here are not supported by Red Hat.

Whitelist format

- ▶ <name> - When used in a syntax description, this string should be replaced by user-defined value.
- ▶ [a|b|c] - When used in a syntax description, only one of the strings separated by | is used.
- ▶ When no comment is present, an option is supported with all possible values.

19.2. Basic options

Emulated machine

-M <machine-type>
-machine <machine-type>[,<property>[=<value>][,...]]

Processor type

-cpu <model>[,<FEATURE>][...]

Additional models are visible by running **-cpu ?** command.

- ▶ **Opteron_G5** - AMD Opteron 63xx class CPU
- ▶ **Opteron_G4** - AMD Opteron 62xx class CPU
- ▶ **Opteron_G3** - AMD Opteron 23xx (AMD Opteron Gen 3)
- ▶ **Opteron_G2** - AMD Opteron 22xx (AMD Opteron Gen 2)
- ▶ **Opteron_G1** - AMD Opteron 240 (AMD Opteron Gen 1)
- ▶ **Westmere** - Westmere E56xx/L56xx/X56xx (Nehalem-C)
- ▶ **Haswell** - Intel Core Processor (Haswell)
- ▶ **SandyBridge** - Intel Xeon E312xx (Sandy Bridge)
- ▶ **Nehalem** - Intel Core i7 9xx (Nehalem Class Core i7)
- ▶ **Penryn** - Intel Core 2 Duo P9xxx (Penryn Class Core 2)
- ▶ **Conroe** - Intel Celeron_4x0 (Conroe/Merom Class Core 2)
- ▶ **cpu64-rhel5** - Red Hat Enterprise Linux 5 supported QEMU Virtual CPU version
- ▶ **cpu64-rhel6** - Red Hat Enterprise Linux 6 supported QEMU Virtual CPU version
- ▶ **default** - special option use default option from above.

Processor Topology

-smp <n>[,cores=<ncores>][,threads=<nthreads>][,sockets=<nsocks>][,maxcpus=<maxcpus>]

Hypervisor and guest operating system limits on processor topology apply.

NUMA system

-numa <nodes>[,mem=<size>][,cpus=<cpu[-cpu]>][,nodeid=<node>]

Hypervisor and guest operating system limits on processor topology apply.

Memory size

-m <megs>

Supported values are limited by guest minimal and maximal values and hypervisor limits.

Keyboard layout

-k <language>

Guest name

-name <name>

Guest UUID

-uuid <uuid>

19.3. Disk options

Generic drive

-drive <option>[,<option>[,<option>[...]]]

Supported with the following options:

- ▶ **readonly**[on|off]
- ▶ **werror**[enospcl|report|stop|ignore]
- ▶ **rerror**[report|stop|ignore]
- ▶ **id**=<id>

Id of the drive has the following limitation for if=none:

- IDE disk has to have <id> in following format: drive-ide0-<BUS>-<UNIT>

Example of correct format:

-drive if=none,id=drive-ide0-<BUS>-<UNIT>,... -device ide-drive,drive=drive-ide0-<BUS>-<UNIT>,bus=ide.<BUS>,unit=<UNIT>

- ▶ **file**=<file>

Value of <file> is parsed with the following rules:

- Passing floppy device as <file> is not supported.
- Passing cd-rom device as <file> is supported only with cdrom media type (media=cdrom) and only as IDE drive (either if=ide or if=none + -device ide-drive).
- If <file> is neither block nor character device, it must not contain ':'.

- ▶ **if**=<interface>

The following interfaces are supported: none, ide, virtio, floppy.

- ▶ **index**=<index>

- ▶ **media**=<media>

- ▶ **cache**=<cache>

Supported values: none, writeback or writethrough.

- ▶ **copy-on-read**=[on|off]

- ▶ **snapshot**=[yes|no]

- ▶ **serial**=<serial>
- ▶ **aio**=<aio>
- ▶ **format**=<format>

This option is not required and can be omitted. However, this is not recommended for raw images because it represents security risk. Supported formats are:

- **qcow2**
- **raw**

Boot option

-boot [order=<drives>][,menu=[on|off]]

Snapshot mode

-snapshot

19.4. Display options

Disable graphics

-nographic

VGA card emulation

-vga <type>

Supported types:

- ▶ **cirrus** - Cirrus Logic GD5446 Video card.
- ▶ **std** - Standard VGA card with Bochs VBE extensions.
- ▶ **qxl** - Spice paravirtual card.
- ▶ **none** - Disable VGA card.

VNC display

-vnc <display>[,<option>[,<option>[,...]]]

Supported display value:

- ▶ [<host>]:<port>
- ▶ unix:<path>
- ▶ **share**[allow-exclusive|force-shared|ignore]
- ▶ none - Supported with no other options specified.

Supported options are:

- ▶ **to**=<port>
- ▶ **reverse**
- ▶ **password**
- ▶ **tls**
- ▶ **x509**=</path/to/certificate/dir> - Supported when **tls** specified.
- ▶ **x509verify**=</path/to/certificate/dir> - Supported when **tls** specified.
- ▶ **sasl**
- ▶ **acl**

Spice desktop

-spice option[,option[,...]]

Supported options are:

- ▶ **port=<number>**
- ▶ **addr=<addr>**
- ▶ **ipv4**
- ▶ **ipv6**
- ▶ **password=<secret>**
- ▶ **disable-ticketing**
- ▶ **disable-copy-paste**
- ▶ **tls-port=<number>**
- ▶ **x509-dir=</path/to/certificate/dir>**
- ▶ **x509-key-file=<file>**
 - ▶ **x509-key-password=<file>**
 - ▶ **x509-cert-file=<file>**
 - ▶ **x509-cacert-file=<file>**
 - ▶ **x509-dh-key-file=<file>**
- ▶ **tls-cipher=<list>**
- ▶ **tls-channel[main|display|cursor|inputs|record|playback]**
- ▶ **plaintext-channel[main|display|cursor|inputs|record|playback]**
- ▶ **image-compression=<compress>**
- ▶ **jpeg-wan-compression=<value>**
- ▶ **zlib-glz-wan-compression=<value>**
- ▶ **streaming-video=[off|all|filter]**
- ▶ **agent-mouse=[on|off]**
- ▶ **playback-compression=[on|off]**
- ▶ **seamless-migratio=[on|off]**

19.5. Network options

TAP network

-netdev tap,id=<id>][,<options>...]

The following options are supported (all use name=value format):

- ▶ **ifname**
- ▶ **fd**
- ▶ **script**
- ▶ **downscript**
- ▶ **sndbuf**
- ▶ **vnet_hdr**
- ▶ **vhost**
- ▶ **vhostfd**
- ▶ **vhostforce**

19.6. Device options

General device

-device <driver>[,<prop>[=<value>]][,...]

All drivers support following properties

- ▶ id
- ▶ bus

Following drivers are supported (with available properties):

▶ **pci-assign**

- host
- bootindex
- configfd
- addr
- rombar
- romfile
- multifunction

If the device has multiple functions, all of them need to be assigned to the same guest.

▶ **rtl8139**

- mac
- netdev
- bootindex
- addr

▶ **e1000**

- mac
- netdev
- bootindex
- addr

▶ **virtio-net-pci**

- ioeventfd
- vectors
- indirect
- event_idx
- csum
- guest_csum
- gso
- guest_tso4
- guest_tso6
- guest_ecn
- guest_ufo
- host_tso4
- host_tso6
- host_ecn
- host_ufo
- mrg_rxbuf
- status
- ctrl_vq
- ctrl_rx
- ctrl_vlan
- ctrl_rx_extra
- mac

- netdev
 - bootindex
 - x-txtimer
 - x-txburst
 - tx
 - addr
- ▶ **qxl**
- ram_size
 - vram_size
 - revision
 - cmdlog
 - addr
- ▶ **ide-drive**
- unit
 - drive
 - physical_block_size
 - bootindex
 - ver
 - wwn
- ▶ **virtio-blk-pci**
- class
 - drive
 - logical_block_size
 - physical_block_size
 - min_io_size
 - opt_io_size
 - bootindex
 - ioeventfd
 - vectors
 - indirect_desc
 - event_idx
 - scsi
 - addr
- ▶ **virtio-scsi-pci** - tech-preview in 6.3, supported in 6.4.
- For Windows' guests, Windows Server 2003, which was tech-preview in 6.4, is no longer supported. However, Windows Server 2008 and 2012, and Windows desktop 7 and 8 are fully supported in 6.5.
- vectors
 - indirect_desc
 - event_idx
 - num_queues
 - addr
- ▶ **isa-debugcon**
- ▶ **isa-serial**
- index
 - iobase
 - irq
 - chardev
- ▶ **virtserialport**

- nr
- chardev
- name
- ▶ **virtconsole**
 - nr
 - chardev
 - name
- ▶ **virtio-serial-pci**
 - vectors
 - class
 - indirect_desc
 - event_idx
 - max_ports
 - flow_control
 - addr
- ▶ **ES1370**
 - addr
- ▶ **AC97**
 - addr
- ▶ **intel-hda**
 - addr
- ▶ **hda-duplex**
 - cad
- ▶ **hda-micro**
 - cad
- ▶ **hda-output**
 - cad
- ▶ **i6300esb**
 - addr
- ▶ **ib700** - no properties
- ▶ **sga** - no properties
- ▶ **virtio-balloon-pci**
 - indirect_desc
 - event_idx
 - addr
- ▶ **usb-tablet**
 - migrate
 - port
- ▶ **usb-kbd**
 - migrate
 - port
- ▶ **usb-mouse**
 - migrate
 - port
- ▶ **usb-ccid** - supported since 6.2
 - port
 - slot
- ▶ **usb-host** - tech preview since 6.2

- hostbus
 - hostaddr
 - hostport
 - vendorid
 - productid
 - isobufs
 - port
- ▶ **usb-hub** - supported since 6.2
- port
- ▶ **usb-ehci** - tech preview since 6.2
- freq
 - maxframes
 - port
- ▶ **usb-storage** - tech preview since 6.2
- drive
 - bootindex
 - serial
 - removable
 - port
- ▶ **usb-redir** - tech preview for 6.3, supported since 6.4
- chardev
 - filter
- ▶ **scsi-cd** - tech preview for 6.3, supported since 6.4
- drive
 - logical_block_size
 - physical_block_size
 - min_io_size
 - opt_io_size
 - bootindex
 - ver
 - serial
 - scsi-id
 - lun
 - channel-scsi
 - wwn
- ▶ **scsi-hd** - tech preview for 6.3, supported since 6.4
- drive
 - logical_block_size
 - physical_block_size
 - min_io_size
 - opt_io_size
 - bootindex
 - ver
 - serial
 - scsi-id
 - lun
 - channel-scsi
 - wwn

- ▶ **scsi-block** -tech preview for 6.3, supported since 6.4
 - drive
 - bootindex
- ▶ **scsi-disk** -tech preview for 6.3
 - drive=drive
 - logical_block_size
 - physical_block_size
 - min_io_size
 - opt_io_size
 - bootindex
 - ver
 - serial
 - scsi-id
 - lun
 - channel-scsi
 - wwn
- ▶ **piix3-usb-uhci**
- ▶ **piix4-usb-uhci**
- ▶ **ccid-card-passthru**

Global device setting

-global <device>.<property>=<value>

Supported devices and properties as in "General device" section with these additional devices:

- ▶ **isa-fdc**
 - driveA
 - driveB
 - bootindexA
 - bootindexB
- ▶ **qxl-vga**
 - ram_size
 - vram_size
 - revision
 - cmdlog
 - addr

Character device

-chardev backend,id=<id>[,<options>]

Supported backends are:

- ▶ **null**,id=<id> - null device
- ▶ **socket**,id=<id>,port=<port>[,host=<host>][,to=<to>][,ipv4][,ipv6][,nodelay][,server][,nowait][,telnet] - tcp socket
- ▶ **socket**,id=<id>,path=<path>[,server][,nowait][,telnet] - unix socket
- ▶ **file**,id=<id>,path=<path> - trafit to file.
- ▶ **stdio**,id=<id> - standard i/o
- ▶ **spicevmc**,id=<id>,name=<name> - spice channel

Enable USB

-usb

19.7. Linux/Multiboot boot

Kernel file

-kernel <bzImage>

Note: multiboot images are not supported

Ram disk

-initrd <file>

Command line parameter

-append <cmdline>

19.8. Expert options

KVM virtualization

-enable-kvm

QEMU-KVM supports only KVM virtualization and it is used by default if available. If -enable-kvm is used and KVM is not available, qemu-kvm fails. However, if -enable-kvm is not used and KVM is not available, **qemu-kvm** runs in TCG mode, which is not supported.

Disable kernel mode PIT reinjection

-no-kvm-pit-reinjection

No shutdown

-no-shutdown

No reboot

-no-reboot

Serial port, monitor, QMP

-serial <dev>

-monitor <dev>

-qmp <dev>

Supported devices are:

- ▶ **stdio** - standard input/output
- ▶ **null** - null device
- ▶ **file:<filename>** - output to file.
- ▶ **tcp:[<host>]:<port>[,server][,nowait][,nodelay]** - TCP Net console.
- ▶ **unix:<path>[,server][,nowait]** - Unix domain socket.
- ▶ **mon:<dev_string>** - Any device above, used to multiplex monitor too.
- ▶ **none** - disable, valid only for -serial.
- ▶ **chardev:<id>** - character device created with -chardev.

Monitor redirect

-mon <chardev_id>[,mode=[readline|control]][,default=[on|off]]

Manual CPU start**-S****RTC****-rtc [base=utc|localtime|date][,clock=host|vm][,driftfix=none|slew]****Watchdog****-watchdog model****Watchdog reaction****-watchdog-action <action>****Guest memory backing****-mem-prealloc -mem-path /dev/hugepages****SMBIOS entry****-smbios type=0[,vendor=<str>][,<version=str>][,date=<str>][,release=%d.%d]****-smbios****type=1[,manufacturer=<str>][,product=<str>][,version=<str>][,serial=<str>][,uuid=<uuid>][,sku=<str>][,family=<str>]**

19.9. Help and information options

Help**-h****-help****Version****-version****Audio help****-audio-help**

19.10. Miscellaneous options

Migration**-incoming****No default configuration****-nodefconfig****-nodefaults**

Running without -nodefaults is not supported

Device configuration file**-readconfig <file>****-writeconfig <file>****Loaded saved state**

-loadvm <file>

Chapter 20. Manipulating the domain xml

This section describes the XML format used to represent domains. Here the term *domain* refers to the root `<domain>` element required for all guest virtual machine. The domain XML has two attributes: **type** specifies the hypervisor used for running the domain. The allowed values are driver specific, but include **KVM** and others. **id** is a unique integer identifier for the running guest virtual machine. Inactive machines have no id value. The sections in this chapter will address the components of the domain XML. Additional chapters in this manual may refer to this chapter when manipulation of the domain XML is required.

20.1. General information and metadata

This information is in this part of the domain XML:

```
<domain type='xen' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <title>A short description - title - of the domain</title>
  <description>Some human readable description</description>
  <metadata>
    <app1:foo xmlns:app1="http://app1.org/app1/">..</app1:foo>
    <app2:bar xmlns:app2="http://app1.org/app2/">..</app2:bar>
  </metadata>
  ...
</domain>
```

Figure 20.1. Domain XML metadata

The components of this section of the domain XML are as follows:

Table 20.1. General metadata elements

Element	Description
<name>	Assigns a name for the virtual machine. This name should consist only of alpha-numeric characters and is required to be unique within the scope of a single host physical machine. It is often used to form the filename for storing the persistent configuration files.
<uuid>	assigns a globally unique identifier for the virtual machine. The format must be RFC 4122 compliant, eg 3e3fce45-4f53-4fa7-bb32-11f34168b82b . If omitted when defining/creating a new machine, a random UUID is generated. It is also possible to provide the UUID via a sysinfo specification.
<title>	title Creates space for a short description of the domain. The title should not contain any newlines.
<description>	Different from the title, This data is not used by libvirt in any way, it can contain any information the user wants to display.
<metadata>	Can be used by applications to store custom metadata in the form of XML nodes/trees. Applications must use custom namespaces on their XML nodes/trees, with only one top-level element per namespace (if the application needs structure, they should have sub-elements to their namespace element)

20.2. Operating system booting

There are a number of different ways to boot virtual machines each with their own pros and cons. Each one is described in the sub-sections that follow and include: BIOS bootloader, Host physical machine bootloader, direct kernel boot, and container boot.

20.2.1. BIOS bootloader

Booting via the BIOS is available for hypervisors supporting full virtualization. In this case the BIOS has a boot order priority (floppy, harddisk, cdrom, network) determining where to obtain/find the boot image. The OS section of the domain XML contains the information as follows:

```

...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <boot dev='hd' />
  <boot dev='cdrom' />
  <bootmenu enable='yes' />
  <smbios mode='sysinfo' />
  <bios useserial='yes' rebootTimeout='0' />
</os>
...

```

Figure 20.2. BOIS bootloader domain XML

The components of this section of the domain XML are as follows:

Table 20.2. BIOS bootloader elements

Element	Description
<type>	Specifies the type of operating system to be booted on the guest virtual machine. hvm indicates that the OS is one designed to run on bare metal, so requires full virtualization. linux refers to an OS that supports the Xen 3 hypervisor guest ABI. There are also two optional attributes, arch specifying the CPU architecture to virtualization, and machine referring to the machine type. Refer to Driver Capabilities for more information.
<loader>	refers to a piece of firmware that is used to assist the domain creation process. It is only needed for using Xen fully virtualized domains.
<boot>	takes one of the values: fd , hd , cdrom or network and is used to specify the next boot device to consider. The boot element can be repeated multiple times to setup a priority list of boot devices to try in turn. Multiple devices of the same type are sorted according to their targets while preserving the order of buses. After defining the domain, its XML configuration returned by libvirt (through virDomainGetXMLDesc) lists devices in the sorted order. Once sorted, the first device is marked as bootable. For more information see BIOS bootloader .
<bootmenu>	determines whether or not to enable an interactive boot menu prompt on guest virtual machine startup. The enable attribute can be either yes or no . If not specified, the hypervisor default is used
<smbios>	determines how SMBIOS information is made visible in the guest virtual machine. The mode attribute must be specified, as either emulate (lets the hypervisor generate all values), host (copies all of Block 0 and Block 1, except for the UUID, from the host physical machine's SMBIOS values; the virConnectGetSysinfo call can be used to see what values are copied), or sysinfo (uses the values in the sysinfo element). If not specified, the hypervisor default setting is used.
<bios>	This element has attribute useserial with possible values yes or no . The attribute enables or disables Serial Graphics Adapter which allows users to see BIOS messages on a serial port. Therefore, one needs to have serial port defined. Note there is another attribute, rebootTimeout that controls whether and after how long the guest virtual machine should start booting again in case the boot fails (according to BIOS). The value is in milliseconds with maximum of 65535 and special value -1 disables the reboot.

20.2.2. Host physical machine bootloader

Hypervisors employing paravirtualization do not usually emulate a BIOS, but instead the host physical machine is responsible for the operating system boot. This may use a pseudo-bootloader in the host physical machine to provide an interface to choose a kernel for the guest virtual machine. An example is pygrub with Xen.

```
...
<bootloader>/usr/bin/pygrub</bootloader>
<bootloader_args>--append single</bootloader_args>
...
```

Figure 20.3. Host physical machine bootloader domain XML

The components of this section of the domain XML are as follows:

Table 20.3. BIOS bootloader elements

Element	Description
<bootloader>	provides a fully qualified path to the bootloader executable in the host physical machine OS. This bootloader will choose which kernel to boot. The required output of the bootloader is dependent on the hypervisor in use.
<bootloader_args>	allows command line arguments to be passed to the bootloader (optional command)

20.2.3. Direct kernel boot

When installing a new guest virtual machine OS, it is often useful to boot directly from a kernel and initrd stored in the host physical machine OS, allowing command line arguments to be passed directly to the installer. This capability is usually available for both para and full virtualized guest virtual machines.

```
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-i386/os/</cmdline>
  <dtb>/root/ppc.dtb</dtb>
</os>
...
```

Figure 20.4. Direct kernel boot

The components of this section of the domain XML are as follows:

Table 20.4. Direct kernel boot elements

Element	Description
<type>	same as described in the BIOS boot section
<loader>	same as described in the BIOS boot section
<kernel>	specifies the fully-qualified path to the kernel image in the host physical machine OS
<initrd>	specifies the fully-qualified path to the (optional) ramdisk image in the host physical machine OS.
<cmdline>	specifies arguments to be passed to the kernel (or installer) at boot time. This is often used to specify an alternate primary console (eg serial port), or the installation media source / kickstart file

20.2.4. Container boot

When booting a domain using container based virtualization, instead of a kernel or boot image, a path to the init binary is required, using the init element. By default this will be launched with no arguments. To specify the initial argv, use the **initarg** element, repeated as many times as required. The **cmdline** element, provides an equivalent to **/proc/cmdline** but will not effect **<initarg>**.

```
>
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-i386/os/</cmdline>
  <dtb>/root/ppc.dtb</dtb>
</os>
...
```

Figure 20.5. Container boot

20.3. SMBIOS system information

Some hypervisors allow control over what system information is presented to the guest virtual machine (for example, SMBIOS fields can be populated by a hypervisor and inspected via the dmidecode command in the guest virtual machine). The optional sysinfo element covers all such categories of information.

```

...
<os>
  <smbios mode='sysinfo' />
  ...
</os>
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
  </bios>
  <system>
    <entry name='manufacturer'>Fedora</entry>
    <entry name='vendor'>Virt-Manager</entry>
  </system>
</sysinfo>
...

```

Figure 20.6. SMBIOS system information

The **<sysinfo>** element has a mandatory attribute **type** that determines the layout of sub-elements, and may be defined as follows:

- ▶ **smbios** - Sub-elements call out specific SMBIOS values, which will affect the guest virtual machine if used in conjunction with the **smbios** sub-element of the **<os>** element. Each sub-element of **sysinfo** names a SMBIOS block, and within those elements can be a list of entry elements that describe a field within the block. The following blocks and entries are recognized:
 - **bios** - This is block 0 of SMBIOS, with entry names drawn from **vendor**, **version**, **date**, and **release**.
 - **<system>** - This is block 1 of SMBIOS, with entry names drawn from **manufacturer**, **product**, **version**, **serial**, **uuid**, **sku**, and **family**. If a **uuid** entry is provided alongside a top-level **uuid** element, the two values must match.

20.4. CPU allocation

```

<domain>
  ...
  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
  ...
</domain>

```

Figure 20.7. CPU allocation

The **<cpu>** element defines the maximum number of virtual CPUs allocated for the guest virtual machine OS, which must be between 1 and the maximum supported by the hypervisor. This element can contain an optional **cpuset** attribute, which is a comma-separated list of physical CPU numbers that domain process and virtual CPUs can be pinned to by default.

Note that the pinning policy of domain process and virtual CPUs can be specified separately by using the **cputune** attribute. If attribute **emulatorpin** of **<cputune>** is specified, **cpuset** specified by **<vcpu>** will be ignored.

Similarly, virtual CPUs that have set a value for **vcpupin** cause **cpuset** settings to be ignored. For

virtual CPUs where **vcpupin** is not specified, it will be pinned to the physical CPUs specified by **cpuset**. Each element in the **cpuset** list is either a single CPU number, a range of CPU numbers, or a caret (^) followed by a CPU number to be excluded from a previous range. The attribute **current** can be used to specify whether fewer than the maximum number of virtual CPUs should be enabled.

The optional attribute **placement** can be used to indicate the CPU placement mode for domain process, its value can be either **static** or **auto**, which defaults to **placement**, or **numatune**, or **static** if **cpuset** is specified. **auto** indicates the domain process will be pinned to the advisory nodeset from querying numad, and the value of attribute **cpuset** will be ignored if it's specified. If both **cpuset** and **placement** are not specified, or if placement is **static**, but no **cpuset** is specified, the domain process will be pinned to all the available physical CPUs.

20.5. CPU tuning

```
<domain>
  ...
  <cputune>
    <vcpupin vcpu="0" cpuset="1-4,^2"/>
    <vcpupin vcpu="1" cpuset="0,1"/>
    <vcpupin vcpu="2" cpuset="2,3"/>
    <vcpupin vcpu="3" cpuset="0,4"/>
    <emulatorpin cpuset="1-3"/>
    <shares>2048</shares>
    <period>1000000</period>
    <quota>-1</quota>
    <emulator_period>1000000</emulator_period>
    <emulator_quota>-1</emulator_quota>
  </cputune>
  ...
</domain>
```

Figure 20.8. CPU tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.5. CPU tuning elements

Element	Description
<cputune>	Provides details regarding the CPU tunable parameters for the domain. This is optional.
<vcpu pin>	Specifies which of host physical machine's physical CPUs the domain VCPU will be pinned to. If this is omitted, and attribute cpuset of element <vcpu> is not specified, the vCPU is pinned to all the physical CPUs by default. It contains two required attributes, the attribute vcpu specifies id , and the attribute cpuset is same as attribute cpuset of element <vCPU>.
<emulatorpin>	Specifies which of the host physical machine CPUs, the "emulator", a subset of a domains not including vcpu, will be pinned to. If this is omitted, and attribute cpuset of element <vcpu> is not specified, the "emulator" is pinned to all the physical CPUs by default. It contains one required attribute cpuset specifying which physical CPUs to pin to. emulatorpin is not allowed if attribute placement of element <vcpu> is auto .
<shares>	Specifies the proportional weighted share for the domain. If this is omitted, it defaults to the OS provided defaults. If there is no unit for the value, it is calculated relative to the setting of other guest virtual machine. For example, if a guest virtual machine is configured with value 2048 will get twice as much CPU time as a guest virtual machine configured with value 1024.
<period>	Specifies the enforcement interval in microseconds. By using period , each of the domain's vcpu will not be allowed to consume more than its allotted quota worth of run time. This value should be within the following range: 1000 - 1000000 . A period with a value of 0 means no value.
<quota>	Specifies the maximum allowed bandwidth in microseconds. A domain with quota as any negative value indicates that the domain has infinite bandwidth, which means that it is not bandwidth controlled. The value should be within the following range: 1000 - 18446744073709551 or less than 0 . A quota with value of 0 means no value. You can use this feature to ensure that all vcpus run at the same speed.
<emulator_period>	Specifies the enforcement interval in microseconds. Within an <emulator_period>, emulator threads (those excluding vcpus) of the domain will not be allowed to consume more than the <emulator_quota> worth of run time. The <emulator_period> value should be in the following range: 1000 - 1000000 . An <emulator_period> with value of 0 , means no

<emulator_quota>	value. Specifies the maximum allowed bandwidth in microseconds for the domain's emulator threads (those excluding vcpus). A domain with an <emulator_quota> as a negative value indicates that the domain has infinite bandwidth for emulator threads (those excluding vcpus), which means that it is not bandwidth controlled. The value should be in the following range: 1000 - 18446744073709551 , or less than 0 . An <emulator_quota> with value 0 means no value.
-------------------------------	---

20.6. Memory backing

Memory backing allows the hypervisor to properly manage large pages within the guest virtual machine. Once confured the following domain XML is effected:

```
<domain>
  ...
  <memoryBacking>
    <hugepages/>
  </memoryBacking>
  ...
</domain>
```

Figure 20.9. Memory backing

The optional **<memoryBacking>** element, may have an **<hugepages>** element set within it. This tells the hypervisor that the guest virtual machine should have its memory allocated using hugepages instead of the normal native page size.

20.7. Memory tuning

```
<domain>
  ...
  <memtune>
    <hard_limit unit='G'>1</hard_limit>
    <soft_limit unit='M'>128</soft_limit>
    <swap_hard_limit unit='G'>2</swap_hard_limit>
    <min_guarantee unit='bytes'>67108864</min_guarantee>
  </memtune>
  ...
</domain>
```

Figure 20.10. Memory tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.6. Memory tuning elements

Element	Description
<memtune>	Provides details regarding the memory tunable parameters for the domain. If this is omitted, it defaults to the OS provided defaults. The parameters are applied to the process as a whole therefore when setting limits, one needs to add up guest virtual machine RAM, guest virtual machine video RAM, and allow for some memory overhead. The last piece is hard to determine so one uses trial and error. For each tunable, it is possible to designate which unit the number is in on input, using the same values as for <memory>. For backwards compatibility, output is always in KiB.
<hard_limit>	This is the maximum memory the guest virtual machine can use. The unit for this value is expressed in kibibytes (i.e. blocks of 1024 bytes)
<soft_limit>	This is the memory limit to enforce during memory contention. The unit for this value is expressed in kibibytes (i.e. blocks of 1024 bytes)
<swap_hard_limit>	This is the maximum memory plus swap the guest virtual machine can use. The unit for this value is expressed in kibibytes (i.e. blocks of 1024 bytes). This has to be more than <hard_limit> value provided
<min_guarantee>	This is the guaranteed minimum memory allocation for the guest virtual machine. The units for this value is expressed in kibibytes (i.e. blocks of 1024 bytes)

20.8. NUMA node tuning

Once NUMA node tuning is done using conventional management tools the following domain XML parameters are effected:

```
>
<domain>
  ...
  <numatune>
    <memory mode="strict" nodeset="1-4,^3"/>
  </numatune>
  ...
</domain>
```

Figure 20.11. NUMA node tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.7. NUMA node tuning elements

Element	Description
<numatune>	Provides details of how to tune the performance of a NUMA host physical machine via controlling NUMA policy for domain process.
<memory>	Specifies how to allocate memory for the domain process on a NUMA host physical machine. It contains several optional attributes. Attribute mode is either interleave , strict , or preferred . If no value is given it defaults to strict . Attribute nodeset specifies the NUMA nodes, using the same syntax as attribute cpuset of element < vcpu >. Attribute placement can be used to indicate the memory placement mode for the domain process. Its value can be either static or auto . If attribute < nodeset > is specified it defaults to the < placement > of < vcpu >, or static . auto indicates the domain process will only allocate memory from the advisory nodeset returned from querying numad and the value of attribute nodeset will be ignored if it's specified. If attribute placement of vcpu is auto , and attribute < numatune > is not specified, a default numatune with < placement > auto and mode strict will be added implicitly.

20.9. Block I/O tuning

```

<domain>
  ...
  <blkiotune>
    <weight>800</weight>
    <device>
      <path>/dev/sda</path>
      <weight>1000</weight>
    </device>
    <device>
      <path>/dev/sdb</path>
      <weight>500</weight>
    </device>
  </blkiotune>
  ...
</domain>

```

Figure 20.12. Block I/O tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.8. Block I/O tuning elements

Element	Description
<blkiotune>	This optional element provides the ability to tune Blkio cgroup tunable parameters for the domain. If this is omitted, it defaults to the OS provided defaults.
<weight>	This optional weight element is the overall I/O weight of the guest virtual machine. The value should be within the range 100 - 1000.
<device>	The domain may have multiple <device> elements that further tune the weights for each host physical machine block device in use by the domain. Note that multiple guest virtual machine disks can share a single host physical machine block device. In addition, as they are backed by files within the same host physical machine file system, this tuning parameter is at the global domain level, rather than being associated with each guest virtual machine disk device (contrast this to the <iotune> element which can be applied to a single <disk>). Each device element has two mandatory sub-elements, <path> describing the absolute path of the device, and <weight> giving the relative weight of that device, which has an acceptable range of 100 - 1000.

20.10. Resource partitioning

Hypervisors may allow for virtual machines to be placed into resource partitions, potentially with nesting of said partitions. The <resource> element groups together configuration related to resource partitioning. It currently supports a child element partition whose content defines the path of the resource partition in which to place the domain. If no partition is listed, then the domain will be placed in a default partition. It is the responsibility of the app/admin to ensure that the partition exists prior to starting the guest virtual machine. Only the (hypervisor specific) default partition can be assumed to exist by default.

```
<resource>
  <partition>/virtualmachines/production</partition>
</resource>
```

Figure 20.13. Resource partitioning

Resource partitions are currently supported by the QEMU and LXC drivers, which map partition paths to cgroups directories in all mounted controllers.

20.11. CPU model and topology

This section covers the requirements for CPU model. Note that every hypervisor has its own policy for which CPU features guest will see by default. The set of CPU features presented to the guest by QEMU/KVM depends on the CPU model chosen in the guest virtual machine configuration. **qemu32** and **qemu64** are basic CPU models but there are other models (with additional features) available. Each model and its topology is specified using the following elements from the domain XML:

```
<cpu match='exact'>
  <model fallback='allow'>core2duo</model>
  <vendor>Intel</vendor>
  <topology sockets='1' cores='2' threads='1' />
  <feature policy='disable' name='lahf_lm' />
</cpu>
```

Figure 20.14. CPU model and topology example 1

```
<cpu mode='host-model'>
  <model fallback='forbid' />
  <topology sockets='1' cores='2' threads='1' />
</cpu>
```

Figure 20.15. CPU model and topology example 2

```
<cpu mode='host-passthrough' />
```

Figure 20.16. CPU model and topology example 3

In cases where no restrictions are to be put on either the CPU model nor its features, a simpler cpu element such as the following may be used.

```
<cpu>
  <topology sockets='1' cores='2' threads='1' />
</cpu>
```

Figure 20.17. CPU model and topology example 4

The components of this section of the domain XML are as follows:

Table 20.9. CPU model and topology elements

Element	Description
<cpu>	This is the main container for describing guest virtual machine CPU requirements.
<match>	<p>Specifies how the virtual CPU is provided to the guest virtual machine match for these requirements. The match attribute can be omitted if topology is the only element within <cpu>. Possible values for the match attribute are:</p> <ul style="list-style-type: none"> ▶ minimum - the specified CPU model and features describes the minimum requested CPU. ▶ exact - the virtual CPU provided to the guest virtual machine will exactly match the specification ▶ strict - the guest virtual machine will not be created unless the host physical machine CPU exactly matches the specification. <p>Note that the match attribute can be omitted and will default to exact.</p>
<mode>	<p>This optional attribute may be used to make it easier to configure a guest virtual machine CPU to be as close to the host physical machine CPU as possible. Possible values for the mode attribute are:</p> <ul style="list-style-type: none"> ▶ custom - describes how the CPU is presented to the guest virtual machine. This is the default setting when the mode attribute is not specified. This mode makes it so that a persistent guest virtual machine will see the same hardware no matter what host physical machine the guest virtual machine is booted on. ▶ host-model - this is essentially a shortcut to copying host physical machine CPU definition from the capabilities XML into the domain XML. As the CPU definition is copied just before starting a domain, the same XML can be used on different host physical machines while still providing the best guest virtual machine CPU each host physical machine supports. Neither the match attribute nor any feature elements can be used in this mode. For more information see libvirt domain XML CPU models ▶ host-passthrough With this mode, the CPU visible to the guest virtual machine is exactly the same as the host physical machine CPU including elements that cause errors within libvirt. The obvious downside of this mode is that the guest virtual machine environment cannot be reproduced on different hardware

and therefore this mode is recommended with great caution. Neither **model** nor **feature** elements are allowed in this mode.

- ▶ Note that in both **host-model** and **host-passthrough** mode, the real (approximate in host-passthrough mode) CPU definition which would be used on current host physical machine can be determined by specifying `VIR_DOMAIN_XML_UPDATE_CPU` flag when calling `virDomainGetXMLDesc` API. When running a guest virtual machine that might be prone to operating system reactivation when presented with different hardware, and which will be migrated between host physical machines with different capabilities, you can use this output to rewrite XML to the custom mode for more robust migration.

<model>	Specifies CPU model requested by the guest virtual machine. The list of available CPU models and their definition can be found in cpu_map.xml file installed in libvirt's data directory. If a hypervisor is not able to use the exact CPU model, libvirt automatically falls back to a closest model supported by the hypervisor while maintaining the list of CPU features. An optional fallback attribute can be used to forbid this behavior, in which case an attempt to start a domain requesting an unsupported CPU model will fail. Supported values for fallback attribute are: allow (this is the default), and forbid . The optional vendor_id attribute can be used to set the vendor id seen by the guest virtual machine. It must be exactly 12 characters long. If not set, the vendor id of the host physical machine is used. Typical possible values are AuthenticAMD and GenuineIntel .
<vendor>	Specifies CPU vendor requested by the guest virtual machine. If this element is missing, the guest virtual machine runs on a CPU matching given features regardless of its vendor. The list of supported vendors can be found in cpu_map.xml .
<topology>	Specifies requested topology of virtual CPU provided to the guest virtual machine. Three non-zero values have to be given for sockets, cores, and threads: total number of CPU sockets, number of cores per socket, and number of threads per core, respectively.
<feature>	Can contain zero or more elements used to fine-tune features provided by the selected CPU model. The list of known feature names can be found in the same file as CPU models. The meaning of each feature element depends on its <code>policy</code> attribute, which has to be set to one of the following values:

- ▶ **force** - forces the virtual to be supported regardless of whether it is actually supported by host physical machine CPU.
- ▶ **require** - dictates that guest virtual machine creation will fail unless the feature is supported by host physical machine CPU. This is the default setting
- ▶ **optional** - this feature is supported by virtual CPU but and only if it is supported by host physical machine CPU.
- ▶ **disable** - this is not supported by virtual CPU.
- ▶ **forbid** - guest virtual machine creation will fail if the feature is supported by host physical machine CPU.

20.11.1. Guest virtual machine NUMA topology

Guest virtual machine NUMA topology can be specified using the `<numa>` element and the following from the domain XML:

```
<cpu>
  <numa>
    <cell cpus='0-3' memory='512000' />
    <cell cpus='4-7' memory='512000' />
  </numa>
</cpu>
...
```

Figure 20.18. Guest virtual machine NUMA topology

Each cell element specifies a NUMA cell or a NUMA node. **cpus** specifies the CPU or range of CPUs that are part of the node. **memory** specifies the node memory in kibibytes (i.e. blocks of 1024 bytes). Each cell or node is assigned **cellid** or **nodeid** in increasing order starting from 0.

20.12. Events configuration

Using the following sections of domain XML it is possible to override the default actions taken on various events.

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<on_lockfailure>poweroff</on_lockfailure>
```

Figure 20.19. Events Configuration

The following collections of elements allow the actions to be specified when a guest virtual machine OS

triggers a life cycle operation. A common use case is to force a reboot to be treated as a poweroff when doing the initial OS installation. This allows the VM to be re-configured for the first post-install bootup.

The components of this section of the domain XML are as follows:

Table 20.10. Event configuration elements

State	Description
<on_poweroff>	<p>Specifies the action that is to be executed when the guest virtual machine requests a poweroff. Four possible arguments are possible:</p> <ul style="list-style-type: none"> ▶ destroy - this action terminates the domain completely and releases all resources ▶ restart - this action terminates the domain completely and restarts it with the same configuration ▶ preserve - this action terminates the domain completely but and its resources are preserved to allow for future analysis. ▶ rename-restart - this action terminates the domain completely and then restarts it with a new name
<on_reboot>	<p>Specifies the action that is to be executed when the guest virtual machine requests a reboot. Four possible arguments are possible:</p> <ul style="list-style-type: none"> ▶ destroy - this action terminates the domain completely and releases all resources ▶ restart - this action terminates the domain completely and restarts it with the same configuration ▶ preserve - this action terminates the domain completely but and its resources are preserved to allow for future analysis. ▶ rename-restart - this action terminates the domain completely and then restarts it with a new name
<on_crash>	<p>Specifies the action that is to be executed when the guest virtual machine crashes. In addition, it supports these additional actions:</p> <ul style="list-style-type: none"> ▶ coredump-destroy - the crashed domain's core is dumped, domain is terminated completely, and all resources are released. ▶ coredump-restart - the crashed domain's core is dumped, and the domain is restarted with the same configuration settings <p>Four possible arguments are possible:</p> <ul style="list-style-type: none"> ▶ destroy - this action terminates the domain completely and releases all resources ▶ restart - this action terminates the domain completely and restarts it with the same configuration ▶ preserve - this action terminates the domain completely but and its resources are preserved to allow for future analysis. ▶ rename-restart - this action terminates the domain completely and then restarts it with a new name

	a new name
<on_lockfailure>	<p>Specifies what action should be taken when a lock manager loses resource locks. The following actions are recognized by libvirt, although not all of them need to be supported by individual lock managers. When no action is specified, each lock manager will take its default action. The following arguments are possible:</p> <ul style="list-style-type: none"> ▶ poweroff - forcefully powers off the domain ▶ restart - restarts the domain to reacquire its locks. ▶ pause - pauses the domain so that it can be manually resumed when lock issues are solved. ▶ ignore - keeps the domain running as if nothing happened.

20.13. Power Management

It is possible to forcibly enable or disable BIOS advertisements to the guest virtual machine OS using conventional management tools which effects the following section of the domain XML:

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

Figure 20.20. Power Management

The **<pm>** element can be enabled using the argument **yes** or disabled using the argument **no**. BIOS support can be implemented for S3 using the argument **suspend-to-disk** and S4 using the argument **suspend-to-mem** ACPI sleep states. If nothing is specified, the hypervisor will be left with its default value.

20.14. Hypervisor features

Hypervisors may allow certain CPU / machine features to be enabled (**state='on'**) or disabled (**state='off'**).

```

...
<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on' />
  </hyperv>
</features>
...

```

Figure 20.21. Hypervisor features

All features are listed within the `<features>` element, if a `<state>` is not specified it is disabled. The available features can be found by calling the `capabilities` XML, but a common set for fully virtualized domains are:

Table 20.11. Hypervisor features elements

State	Description
<code><pae></code>	Physical address extension mode allows 32-bit guest virtual machines to address more than 4 GB of memory.
<code><acpi></code>	Useful for power management, for example, with KVM guest virtual machines it is required for graceful shutdown to work.
<code><apic></code>	Allows the use of programmable IRQ management. For this element, there is an optional attribute <code>eoи</code> with values <code>on</code> and <code>off</code> which sets the availability of EOI (End of Interrupt) for the guest virtual machine.
<code><hap></code>	Enables the use of Hardware Assisted Paging if it is available in the hardware.
<code>hyperv</code>	Enables various features to improve the behavior of guest virtual machines running Microsoft Windows. Using the optional attribute <code>relaxed</code> with values <code>on</code> or <code>off</code> enables or disables the relax constraints on timers

20.15. Time keeping

The guest virtual machine clock is typically initialized from the host physical machine clock. Most operating systems expect the hardware clock to be kept in UTC, which is the default setting. Note that for Windows guest virtual machines the guest virtual machine must be set in `localtime`.

```
...
<clock offset='localtime'>
  <timer name='rtc' tickpolicy='catchup' track='guest'>
    <catchup threshold='123' slew='120' limit='10000' />
  </timer>
  <timer name='pit' tickpolicy='delay' />
</clock>
...
```

Figure 20.22. Time keeping

The components of this section of the domain XML are as follows:

Table 20.12. Time keeping elements

State	Description
<clock>	<p>The offset attribute takes four possible values, allowing for fine grained control over how the guest virtual machine clock is synchronized to the host physical machine. Note that hypervisors are not required to support all policies across all time sources</p> <ul style="list-style-type: none"> ▶ utc - Synchronizes the clock to UTC when booted. utc mode can be converted to variable mode, which can be controlled by using the adjustment attribute. If the value is reset, the conversion is not done. A numeric value forces the conversion to variable mode using the value as the initial adjustment. The default adjustment is hypervisor specific. ▶ localtime - Synchronizes the guest virtual machine clock with the host physical machine's configured timezone when booted. The adjustment attribute behaves the same as in 'utc' mode. ▶ timezone - Synchronizes the guest virtual machine clock to the requested timezone using the timezone attribute. ▶ variable - Gives the guest virtual machine clock an arbitrary offset applied relative to UTC or localtime, depending on the basis attribute. The delta relative to UTC (or localtime) is specified in seconds, using the adjustment attribute. The guest virtual machine is free to adjust the RTC over time and expect that it will be honored at next reboot. This is in contrast to utc and localtime mode (with the optional attribute adjustment='reset'), where the RTC adjustments are lost at each reboot. In addition the basis attribute can be either utc (default) or localtime. The clock element may have zero or more <timer> elements.
<timer>	See Note
<frequency>	This is an unsigned integer specifying the frequency at which name="tsc" runs.
<mode>	The mode attribute controls how the name="tsc" <timer> is managed, and can be set to: auto , native , emulate , paravirt , or smpsafe . Other timers are always emulated.
<present>	Specifies whether a particular timer is available to the guest virtual machine. Can be set to yes or no



Additional information about the <timer> element

Each **<timer>** element must contain a **name** attribute, and may have the following attributes depending on the name specified.

- ▶ **<name>** - selects which **timer** is being modified. The following values are acceptable:**hpet** (QEMU-KVM or XEN), **kvmclock** (QEMU-KVM), **pit**(QEMU-KVM), or **rtc**(QEMU-KVM).
- ▶ **track** - specifies the timer track. The following values are acceptable: **boot**, **guest**, or **wall**. **track** is only valid for **name="rtc"** or **name="platform"**.
- ▶ **tickpolicy** - determines what happens whens the deadline for injecting a tick to the guest virtual machine is missed. The following values can be assigned:
 - **delay** -will continue to deliver ticks at the normal rate. The guest virtual machine time will be delayed due to the late tick
 - **catchup** - delivers ticks at a higher rate in order to catch up with the missed tick. The guest virtual machine time is not displayed once catchup is complete. In addition, there can be three optional attributes, each a positive integer, as follows: threshold, slew, and limit.
 - **merge** - merges the missed tick(s) into one tick and injects them. The guest virtual machine time may be delayed, depending on how the merge is done.
 - **discard** - throws away the missed tick(s) and continues with future injection at its default interval setting. The guest virtual machine time may be delayed, unless there is an explicit statement for handling lost ticks

20.16. Devices

This set of XML elements are all used to describe devices provided to the guest virtual machine domain. All of the devices below are indicated as children of the main devices element.

```
...
<devices>
  <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
</devices>
...
```

Figure 20.23. Devices - child elements

The contents of the **<emulator>** element specify the fully qualified path to the device model emulator binary. The capabilities XML specifies the recommended default emulator to use for each particular domain type or architecture combination.

20.16.1. Hard drives, floppy disks, CDROMs

This section of the domain XML specifies any device that looks like a disk, be it a floppy, harddisk, cdrom, or paravirtualized driver is specified via the disk element.

```

...
<devices>
  <disk type='file' snapshot='external'>
    <driver name="tap" type="aio" cache="default"/>
    <source file='/var/lib/xen/images/fv0' startupPolicy='optional'>
      <seclabel relabel='no'/>
    </source>
    <target dev='hda' bus='ide'/>
    <iotune>
      <total_bytes_sec>100000000</total_bytes_sec>
      <read_iops_sec>400000</read_iops_sec>
      <write_iops_sec>100000</write_iops_sec>
    </iotune>
    <boot order='2' />
    <encryption type='... '>
      ...
    </encryption>
    <shareable/>
    <serial>
      ...
    </serial>
  </disk>
  ...
  <disk type='network'>
    <driver name="qemu" type="raw" io="threads" ioeventfd="on"
event_idx="off"/>
    <source protocol="sheepdog" name="image_name">
      <host name="hostname" port="7000"/>
    </source>
    <target dev="hdb" bus="ide"/>
    <boot order='1' />
    <transient/>
    <address type='drive' controller='0' bus='1' unit='0' />
  </disk>
  <disk type='network'>
    <driver name="qemu" type="raw"/>
    <source protocol="rbd" name="image_name2">
      <host name="hostname" port="7000"/>
    </source>
    <target dev="hdd" bus="ide"/>
    <auth username='myuser'>
      <secret type='ceph' usage='mypassid' />
    </auth>
  </disk>
  <disk type='block' device='cdrom'>
    <driver name='qemu' type='raw' />
    <target dev='hdc' bus='ide' tray='open' />
    <readonly/>
  </disk>
  <disk type='block' device='lun'>
    <driver name='qemu' type='raw' />
    <source dev='/dev/sda' />
    <target dev='sda' bus='scsi' />
    <address type='drive' controller='0' bus='0' target='3' unit='0' />
  </disk>
  <disk type='block' device='disk'>
    <driver name='qemu' type='raw' />
    <source dev='/dev/sda' />
    <geometry cyls='16383' heads='16' secs='63' trans='lba' />
    <blockio logical_block_size='512' physical_block_size='4096' />
    <target dev='hda' bus='ide' />
  </disk>
  <disk type='volume' device='disk'>
    <driver name='qemu' type='raw' />

```

```

<source pool='blk-pool0' volume='blk-pool0-vol0' />
<target dev='hda' bus='ide' />
</disk>
</devices>
...

```

Figure 20.24. Devices - Hard drives, floppy disks, CDROMs

20.16.1.1. Disk element

The `<disk>` element is the main container for describing disks. The attribute `type` can be used with the `<disk>` element. The following types are allowed:

- ▶ **file**
- ▶ **block**
- ▶ **dir**
- ▶ **network**

For more information, see [Disk Elements](#)

20.16.1.2. Source element

If the `<disk type='file' />`, then the `file` attribute specifies the fully-qualified path to the file holding the disk. If the `<disk type='block' />`, then the `dev` attribute specifies the path to the host physical machine device to serve as the disk. With both `file` and `block`, one or more optional sub-elements `seclabel`, described below, can be used to override the domain security labeling policy for just that source file. If the disk type is `dir`, then the `dir` attribute specifies the fully-qualified path to the directory to use as the disk. If the disk type is `network`, then the `protocol` attribute specifies the protocol to access to the requested image; possible values are `nbd`, `rbd`, `sheepdog` or `gluster`.

If the protocol attribute is `rbd`, `sheepdog` or `gluster`, an additional attribute `name` is mandatory to specify which volume and or image will be used. When the disk type is `network`, the `source` may have zero or more `host` sub-elements used to specify the host physical machines to connect, including: `type='dir'` and `type='network'`. For a `file` disk type which represents a cdrom or floppy (the `device` attribute), it is possible to define policy what to do with the disk if the source file is not accessible. This is done by manipulating the `startupPolicy` attribute, with the following values:

- ▶ **mandatory** causes a failure if missing for any reason. This is the default setting.
- ▶ **requisite** causes a failure if missing on boot up, drops if missing on migrate/restore/revert
- ▶ **optional** drops if missing at any start attempt

20.16.1.3. Mirror element

This element is present if the hypervisor has started a `BlockCopy` operation, where the `<mirror>` location in the attribute file will eventually have the same contents as the source, and with the file format in attribute format (which might differ from the format of the source). If an attribute `ready` is present, then it is known the disk is ready to pivot; otherwise, the disk is probably still copying. For now, this element only valid in output; it is ignored on input.

20.16.1.4. Target element

The `<target>` element controls the bus / device under which the disk is exposed to the guest virtual machine OS. The `dev` attribute indicates the logical device name. The actual device name specified is not guaranteed to map to the device name in the guest virtual machine OS. The optional `bus` attribute specifies the type of disk device to emulate; possible values are driver specific, with typical values being

ide, scsi, virtio, xen, usb or sata. If omitted, the bus type is inferred from the style of the device name. eg, a device named '**sda**' will typically be exported using a SCSI bus. The optional attribute **tray** indicates the tray status of the removable disks (i.e. CDROM or Floppy disk), the value can be either **open** or **closed**. The default setting is **closed**. For more information, see [target Elements](#)

20.16.1.5. iotune

The optional **<iotune>** element provides the ability to provide additional per-device I/O tuning, with values that can vary for each device (contrast this to the **blkiotune** element, which applies globally to the domain). This element has the following optional sub-elements. Note that any sub-element not specified or at all or specified with a value of **0** implies no limit.

- ▶ **<total_bytes_sec>** - the total throughput limit in bytes per second. This element cannot be used with **<read_bytes_sec>** or **<write_bytes_sec>**.
- ▶ **<read_bytes_sec>** - the read throughput limit in bytes per second.
- ▶ **<write_bytes_sec>** - the write throughput limit in bytes per second.
- ▶ **<total_iops_sec>** - the total I/O operations per second. This element cannot be used with **<read_iops_sec>** or **<write_iops_sec>**.
- ▶ **<read_iops_sec>** - the read I/O operations per second.
- ▶ **<write_iops_sec>** - the write I/O operations per second.

20.16.1.6. driver

The optional **<driver>** element allows specifying further details related to the hypervisor driver that is used to provide the disk. The following options may be used:

- ▶ If the hypervisor supports multiple backend drivers, then the **name** attribute selects the primary backend driver name, while the optional **type** attribute provides the sub-type. For a list of possible types refer to [Driver Elements](#).
- ▶ The optional **cache** attribute controls the cache mechanism, possible values are: **default, none, writethrough, writeback, directsync** (similar to **writethrough**, but it bypasses the host physical machine page cache) and **unsafe** (host physical machine may cache all disk io, and sync requests from guest virtual machine virtual machines are ignored).
- ▶ The optional **error_policy** attribute controls how the hypervisor behaves on a disk read or write error, possible values are **stop, report, ignore, and enospace**. The default setting of **error_policy** is **report**. There is also an optional **rerror_policy** that controls behavior for read errors only. If no **rerror_policy** is given, **error_policy** is used for both read and write errors. If **rerror_policy** is given, it overrides the **error_policy** for read errors. Also note that **enospace** is not a valid policy for read errors, so if **error_policy** is set to **enospace** and **no rerror_policy** is given, the read error the default setting, **report** will be used.
- ▶ The optional **io** attribute controls specific policies on I/O; **qemu** guest virtual machine virtual machines support **threads** and **native**. The optional **ioeventfd** attribute allows users to set domain I/O asynchronous handling for disk device. The default is left to the discretion of the hypervisor. Accepted values are **on** and **off**. Enabling this allows the guest virtual machine virtual machine to be executed while a separate thread handles I/O. Typically guest virtual machine virtual machines experiencing high system CPU utilization during I/O will benefit from this. On the other hand, an overloaded host physical machine can increase guest virtual machine virtual machine I/O latency. Unless you are absolutely certain that the **io** needs to be manipulated, it is highly recommended that you not change the default setting and allow the hypervisor to dictate the setting.
- ▶ The optional **event_idx** attribute controls some aspects of device event processing and can be set to either **on** or **off** - if it is on, it will reduce the number of interrupts and exits for the guest virtual machine virtual machine. The default is determined by the hypervisor and the default setting is **on**. In cases that there is a situation where this behavior is suboptimal, this attribute provides a way to force the feature **off**. Unless you are absolutely certain that the **event_idx** needs to be manipulated, it is highly recommended that you not change the default setting and allow the hypervisor to dictate the setting.

- ▶ The optional **copy_on_read** attribute controls whether to copy the read backing file into the image file. The accepted values can be either **on** or **<off>**. **copy-on-read** avoids accessing the same backing file sectors repeatedly and is useful when the backing file is over a slow network. By default **copy-on-read** is **off**.

20.16.1.7. Additional Device Elements

The following attributes may be used within the **device** element:

- ▶ **<boot>** - Specifies that the disk is bootable.

Additional boot values

- **<order>** - Determines the order in which devices will be tried during boot sequence.
- **<per-device>** boot elements cannot be used together with general boot elements in BIOS bootloader section
- ▶ **<encryption>** - Specifies how the volume is encrypted. See the Storage Encryption page for more information.
- ▶ **<readonly>** - Indicates the device cannot be modified by the guest virtual machine virtual machine. This setting is the default for disks with **attribute device='cdrom'**.
- ▶ **shareable** Indicates the device is expected to be shared between domains (as long as hypervisor and OS support this). If **shareable** is used, **cache='no'** should be used for that device.
- ▶ **<transient>**- Indicates that changes to the device contents should be reverted automatically when the guest virtual machine virtual machine exits. With some hypervisors, marking a disk **transient** prevents the domain from participating in migration or snapshots.
- ▶ **<serial>**- Specifies the serial number of guest virtual machine virtual machine's hard drive. For example, **<serial>WD-WMAP9A966149</serial>**.
- ▶ **<wwn>** - Specifies the WWN (World Wide Name) of a virtual hard disk or CD-ROM drive. It must be composed of 16 hexadecimal digits.
- ▶ **<vendor>** - Specifies the vendor of a virtual hard disk or CD-ROM device. It must not be longer than 8 printable characters.
- ▶ **<product>** - Specifies the product of a virtual hard disk or CD-ROM device. It must not be longer than 16 printable characters
- ▶ **<host>** - Supports 4 attributes: **viz, name, port, transport** and **socket**, which specify the hostname, the port number, transport type and path to socket, respectively. The meaning of this element and the number of the elements depend on the **protocol** attribute as shown here:

additional host attributes

- **nbd** - Specifies a server running nbd-server and may only be used for only one host physical machine
- **rbd** - Monitors servers of RBD type and may be used for one or more host physical machines
- **sheepdog** - Specifies one of the sheepdog servers (default is localhost:7000) and can be used one or none of the host physical machines
- **gluster** - Specifies a server running a glusterd daemon and may be used for only one host physical machine. The valid values for transport attribute are **tcp**, **rdma** or **unix**. If nothing is specified, **tcp** is assumed. If transport is **unix**, the **socket** attribute specifies path to unix socket.
- ▶ **<address>** - Ties the disk to a given slot of a controller. The actual **<controller>** device can often be inferred by but it can also be explicitly specified. The **type** attribute is mandatory, and is typically **pci** or **drive**. For a **pci** controller, additional attributes for **bus**, **slot**, and **function** must be present, as well as optional **domain** and **multifunction**. **multifunction** defaults to **off**. For a **drive** controller, additional attributes **controller**, **bus**, **target**, and **unit** are available, each with a default setting of **0**.
- ▶ **auth** - Provides the authentication credentials needed to access the source. It includes a mandatory attribute **username**, which identifies the username to use during authentication, as well as a sub-element **secret** with mandatory attribute **type**. More information can be found here at [Device](#)

Elements

- ▶ **geometry** - Provides the ability to override geometry settings. This mostly useful for S390 DASD-disks or older DOS-disks.
- ▶ **cyls** - Specifies the number of cylinders.
- ▶ **heads** - Specifies the number of heads.
- ▶ **secs** - Specifies the number of sectors per track.
- ▶ **trans** - Specifies the BIOS-Translation-Modus and can have the following values:**none**, **1ba** or **auto**
- ▶ **blockio** - Allows the block device to be overridden with any of the block device properties listed below:

blockio options

- **logical_block_size**- reports to the guest virtual machine virtual machine OS and describes the smallest units for disk I/O.
- **physical_block_size** - reports to the guest virtual machine virtual machine OS and describes the disk's hardware sector size which can be relevant for the alignment of disk data.

20.16.2. Filesystems

A filesystems directory on the host physical machine that can be accessed directly from the guest virtual machine virtual machine

```
...
<devices>
  <filesystem type='template'>
    <source name='my-vm-template' />
    <target dir='/' />
  </filesystem>
  <filesystem type='mount' accessmode='passthrough'>
    <driver type='path' wrpolicy='immediate' />
    <source dir='/export/to/guest' />
    <target dir='/import/from/host' />
    <readonly>
  </filesystem>
  ...
</devices>
...
```

Figure 20.25. Devices - filesystems

The **filesystem** attribute has the following possible values:

- ▶ **type='mount'** - Specifies the host physical machine directory to mount in the guest virtual machine. This is the default type if one is not specified. This mode also has an optional sub-element **driver**, with an attribute **type='path'** or **type='handle'**. The driver block has an optional attribute **wrpolicy** that further controls interaction with the host physical machine page cache; omitting the attribute reverts to the default setting, while specifying a value **immediate** means that a host physical machine writeback is immediately triggered for all pages touched during a guest virtual machine file write operation
- ▶ **type='template'** - Specifies the OpenVZ filesystem template and is only used by OpenVZ driver.
- ▶ **type='file'** - Specifies that a host physical machine file will be treated as an image and mounted in the guest virtual machine. This filesystem format will be autodetected and is only used by LXC

driver.

- ▶ **type='block'** - Specifies the host physical machine block device to mount in the guest virtual machine. The filesystem format will be autodetected and is only used by LXC driver.
- ▶ **type='ram'** - Specifies that an in-memory filesystem, using memory from the host physical machine OS will be used. The source element has a single attribute **usage** which gives the memory usage limit in kibibytes and is only used by LXC driver.
- ▶ **type='bind'** - Specifies a directory inside the guest virtual machine which will be bound to another directory inside the guest virtual machine. This element is only used by LXC driver.
- ▶ **accessmode** which specifies the security mode for accessing the source. Currently this only works with type='mount' for the QEMU/KVM driver. The possible values are:
 - **passthrough** - Specifies that the source is accessed with the User's permission settings that are set from inside the guest virtual machine. This is the default accessmode if one is not specified.
 - **mapped** - Specifies that the source is accessed with the permission settings of the hypervisor.
 - **squash** - Similar to '**passthrough**', the exception is that failure of privileged operations like **chown** are ignored. This makes a passthrough-like mode usable for people who run the hypervisor as non-root.
- ▶ **<source>** - Specifies that the resource on the host physical machine that is being accessed in the guest virtual machine. The **name** attribute must be used with **<type='template'>**, and the **dir** attribute must be used with **<type='mount'>**. The **usage** attribute is used with **<type='ram'>** to set the memory limit in KB.
- ▶ **target** - Dictates where the source drivers can be accessed in the guest virtual machine. For most drivers this is an automatic mount point, but for QEMU-KVM this is merely an arbitrary string tag that is exported to the guest virtual machine as a hint for where to mount.
- ▶ **readonly** - Enables exporting the filesystem as a readonly mount for guest virtual machine, by default **read-write** access is given.
- ▶ **space_hard_limit** - Specifies the maximum space available to this guest virtual machine's filesystem
- ▶ **space_soft_limit** - Specifies the maximum space available to this guest virtual machine's filesystem. The container is permitted to exceed its soft limits for a grace period of time. Afterwards the hard limit is enforced.

20.16.3. Device addresses

Many devices have an optional **<address>** sub-element to describe where the device placed on the virtual bus is presented to the guest virtual machine. If an address (or any optional attribute within an address) is omitted on input, libvirt will generate an appropriate address; but an explicit address is required if more control over layout is required. See below for device examples including an address element.

Every address has a mandatory attribute **type** that describes which bus the device is on. The choice of which address to use for a given device is constrained in part by the device and the architecture of the guest virtual machine. For example, a disk device uses **type='disk'**, while a console device would use **type='pci'** on i686 or x86_64 guest virtual machines, or **type='spapr-vio'** on PowerPC64 pseries guest virtual machines. Each address **<type>** has additional optional attributes that control where on the bus the device will be placed. The additional attributes are as follows:

- ▶ **type='pci'** - PCI addresses have the following additional attributes:
 - **domain** (a 2-byte hex integer, not currently used by qemu)
 - **bus** (a hex value between 0 and 0xff, inclusive)
 - **slot** (a hex value between 0x0 and 0x1f, inclusive)
 - **function** (a value between 0 and 7, inclusive)
 - Also available is the **multifunction** attribute, which controls turning on the multifunction bit for a particular slot/function in the PCI control register. This multifunction attribute defaults to '**off**'.

- but should be set to '**on**' for function 0 of a slot that will have multiple functions used.
- ▶ **type='drive'** - drive addresses have the following additional attributes:
 - **controller** - (a 2-digit controller number)
 - **bus** - (a 2-digit bus number)
 - **target** - (a 2-digit bus number)
 - **unit** - (a 2-digit unit number on the bus)
- ▶ **type='virtio-serial'** - Each virtio-serial address has the following additional attributes:
 - **controller** - (a 2-digit controller number)
 - **bus** - (a 2-digit bus number)
 - **slot** - (a 2-digit slot within the bus)
- ▶ **type='ccid'** - A CCID address, used for smart-cards, has the following additional attributes:
 - **bus** - (a 2-digit bus number)
 - **slot** attribute - (a 2-digit slot within the bus)
- ▶ **type='usb'** - USB addresses have the following additional attributes:
 - **bus** - (a hex value between 0 and 0xffff, inclusive)
 - **port** - (a dotted notation of up to four octets, such as 1.2 or 2.1.3.1)
- ▶ **type='spapr-vio'** - On PowerPC pseries guest virtual machines, devices can be assigned to the SPAPR-VIO bus. It has a flat 64-bit address space; by convention, devices are generally assigned at a non-zero multiple of 0x1000, but other addresses are valid and permitted by libvirt. The additional attribute: **reg** (the hex value address of the starting register) can be assigned to this attribute.

20.16.4. Controllers

Depending on the guest virtual machine architecture, some device busses can appear more than once, with a group of virtual devices tied to a virtual controller. Normally, libvirt can automatically infer such controllers without requiring explicit XML markup, but sometimes it is necessary to provide an explicit controller element.

```
...
<devices>
  <controller type='ide' index='0' />
  <controller type='virtio-serial' index='0' ports='16' vectors='4' />
  <controller type='virtio-serial' index='1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0' />
  </controller>
  ...
</devices>
...
```

Figure 20.26. Devices - controllers

Each controller has a mandatory attribute **type**, which must be one of "**ide**", "**fdc**", "**scsi**", "**sata**", "**usb**", "**ccid**", or "**virtio-serial**", and a mandatory attribute **index** which is the decimal integer describing in which order the bus controller is encountered (for use in controller attributes of **address** elements). The "virtio-serial" controller has two additional optional attributes **ports** and **vectors**, which control how many devices can be connected through the controller. A "scsi" controller has an optional attribute **model**, which is one of "**auto**", "**buslogic**", "**ibmvscsi**", "**lsilogic**", "**lsias1068**", "**virtio-scsi**" or "**vmpvscsi**". A "usb" controller has an optional attribute **model**, which is one of "**piix3-uhci**", "**piix4-uhci**", "**ehci**", "**ich9**"

`ehci1`, `ich9-uhci1`, `ich9-uhci2`, `ich9-uhci3`, `vt82c686b-uhci`, `pci-ohci` or `nec-xhci`. Additionally, if the USB bus needs to be explicitly disabled for the guest virtual machine, `model='none'` may be used. The PowerPC64 "spapr-vio" addresses do not have an associated controller.

For controllers that are themselves devices on a PCI or USB bus, an optional sub-element `address` can specify the exact relationship of the controller to its master bus, with semantics given above.

USB companion controllers have an optional sub-element `master` to specify the exact relationship of the companion to its master controller. A companion controller is on the same bus as its master, so the companion index value should be equal.

```
...
<devices>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0' bus='0' slot='4' function='7' />
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0' />
    <address type='pci' domain='0' bus='0' slot='4' function='0'
multifunction='on' />
  </controller>
  ...
</devices>
...
```

Figure 20.27. Devices - controllers - USB

20.16.5. Device leases

When using a lock manager, you have the option to record device leases against a guest virtual machine. The lock manager will ensure that the guest virtual machine doesn't start unless the leases can be acquired. When configured using conventional management tools, the following section of the domain xml is effected:

```
...
<devices>
  ...
  <lease>
    <lockspace>somearea</lockspace>
    <key>somekey</key>
    <target path='/some/lease/path' offset='1024' />
  </lease>
  ...
</devices>
...
```

Figure 20.28. Devices - device leases

The `lease` section can have the following arguments:

- ▶ **lockspace** - an arbitrary string that identifies lockspace within which the key is held. Lock managers may impose extra restrictions on the format, or length of the lockspace name.
- ▶ **key** - an arbitrary string, that uniquely identifies the lease to be acquired. Lock managers may impose extra restrictions on the format, or length of the key.
- ▶ **target** - the fully qualified path of the file associated with the lockspace. The offset specifies where the lease is stored within the file. If the lock manager does not require an offset, set this value to **0**.

20.16.6. Host physical machine device assignment

20.16.6.1. USB / PCI devices

The host physical machine's USB and PCI devices can be passed through to the guest virtual machine using the **hostdev** element, by modifying the host physical machine using a management tool the following section of the domain xml file is configured:

```
...
<devices>
  <hostdev mode='subsystem' type='usb'>
    <source startupPolicy='optional'>
      <vendor id='0x1234' />
      <product id='0xbeef' />
    </source>
    <boot order='2' />
  </hostdev>
</devices>
...
```

Figure 20.29. Devices - host physical machine device assignment

Alternatively the following can also be done:

```
...
<devices>
  <hostdev mode='subsystem' type='pci' managed='yes'>
    <source>
      <address bus='0x06' slot='0x02' function='0x0' />
    </source>
    <boot order='1' />
    <rom bar='on' file='/etc/fake/boot.bin' />
  </hostdev>
</devices>
...
```

Figure 20.30. Devices - host physical machine device assignment alternative

The components of this section of the domain XML are as follows:

Table 20.13. Host physical machine device assignment elements

Parameter	Description
hostdev	<p>This is the main container for describing host physical machine devices. For USB device passthrough mode is always subsystem and type is usb for a USB device and pci for a PCI device. When managed is yes for a PCI device, it is detached from the host physical machine before being passed on to the guest virtual machine, and reattached to the host physical machine after the guest virtual machine exits. If managed is omitted or no for PCI and for USB devices, the user is responsible to use the argument virNodeDeviceDetach (or virsh nodedev-detach) before starting the guest virtual machine or hot-plugging the device, and virNodeDeviceReAttach (or virsh nodedev-reattach) after hot-unplug or stopping the guest virtual machine.</p>
source	<p>Describes the device as seen from the host physical machine. The USB device can either be addressed by vendor / product id using the vendor and product elements or by the device's address on the host physical machines using the address element. PCI devices on the other hand can only be described by their address. Note that the source element of USB devices may contain a startupPolicy attribute which can be used to define a rule for what to do if the specified host physical machine USB device is not found. The attribute accepts the following values:</p> <ul style="list-style-type: none"> ▶ mandatory - fails if missing for any reason (the default) ▶ requisite - fails if missing on boot up, drops if missing on migrate/restore/revert ▶ optional - drops if missing at any start attempt
vendor, product	<p>These elements each have an id attribute that specifies the USB vendor and product id. The IDs can be given in decimal, hexadecimal (starting with 0x) or octal (starting with 0) form.</p>
boot	<p>Specifies that the device is bootable. The attribute's order determines the order in which devices will be tried during boot sequence. The per-device boot elements cannot be used together with general boot elements in BIOS bootloader section.</p>
rom	<p>Used to change how a PCI device's ROM is presented to the guest virtual machine. The optional bar attribute can be set to on or off, and determines whether or not the device's ROM will be visible in the guest virtual machine's</p>

	memory map. (In PCI documentation, the rombar setting controls the presence of the Base Address Register for the ROM). If no rom bar is specified, the default setting will be used. The optional file attribute is used to point to a binary file to be presented to the guest virtual machine as the device's ROM BIOS. This can be useful, for example, to provide a PXE boot ROM for a virtual function of an sr-iov capable ethernet device (which has no boot ROMs for the VFs).
address	Also has a bus and device attribute to specify the USB bus and device number the device appears at on the host physical machine. The values of these attributes can be given in decimal, hexadecimal (starting with 0x) or octal (starting with 0) form. For PCI devices the element carries 3 attributes allowing to designate the device as can be found with lspci or with virsh nodedev-list

20.16.6.2. Block / character devices

The host physical machine's block / character devices can be passed through to the guest virtual machine by using management tools to modify the domain xml **hostdev** element. Note that this is only possible with container based virtualization.

```
...
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
```

Figure 20.31. Devices - host physical machine device assignment block character devices

An alternative approach is this:

```
...
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
```

Figure 20.32. Devices - host physical machine device assignment block character devices alternative 1

Another alternative approach is this:

```
...
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
```

Figure 20.33. Devices - host physical machine device assignment block character devices alternative 2

The components of this section of the domain XML are as follows:

Table 20.14. Block / character device elements

Parameter	Description
hostdev	This is the main container for describing host physical machine devices. For block/character devices passthrough mode is always capabilities and type is block for a block device and char for a character device.
source	This describes the device as seen from the host physical machine. For block devices, the path to the block device in the host physical machine OS is provided in the nested block element, while for character devices the char element is used

20.16.7. Redirected devices

USB device redirection through a character device is supported by configuring it with management tools that modify the following section of the domain xml:

```

...
<devices>
  <redirdev bus='usb' type='tcp'>
    <source mode='connect' host='localhost' service='4000' />
    <boot order='1' />
  </redirdev>
  <redirfilter>
    <usbdev class='0x08' vendor='0x1234' product='0xbeef' version='2.00'
allow='yes' />
    <usbdev allow='no' />
  </redirfilter>
</devices>
...

```

Figure 20.34. Devices - redirected devices

The components of this section of the domain XML are as follows:

Table 20.15. Redirected device elements

Parameter	Description
redirdev	This is the main container for describing redirected devices. bus must be usb for a USB device. An additional attribute type is required, matching one of the supported serial device types, to describe the host physical machine side of the tunnel; type='tcp' or type='spicevmc' (which uses the usredir channel of a SPICE graphics device) are typical. The redirdev element has an optional sub-element address which can tie the device to a particular controller. Further sub-elements, such as source , may be required according to the given type , although atarget sub-element is not required (since the consumer of the character device is the hypervisor itself, rather than a device visible in the guest virtual machine).
boot	Specifies that the device is bootable. The order attribute determines the order in which devices will be tried during boot sequence. The per-device boot elements cannot be used together with general boot elements in BIOS bootloader section.
redirfilter	This is used for creating the filter rule to filter out certain devices from redirection. It uses sub-element usbdev to define each filter rule. The class attribute is the USB Class code.

20.16.8. Smartcard devices

A virtual smartcard device can be supplied to the guest virtual machine via the **smartcard** element. A USB smartcard reader device on the host physical machine cannot be used on a guest virtual machine

with simple device passthrough, as it cannot be made available to both the host physical machine and guest virtual machine and can possibly lock the host physical machine computer when it is removed from the guest virtual machine. Therefore, some hypervisors provide a specialized virtual device that can present a smartcard interface to the guest virtual machine, with several modes for describing how the credentials are obtained from the host physical machine or even a from a channel created to a third-party smartcard provider. To set these parameters use a management tool that will edit the following section of the domain XML:

USB device redirection through a character device is supported by configuring it with management tools that modify the following section of the domain xml:

```
...
<devices>
  <smartcard mode='host' />
  <smartcard mode='host-certificates'>
    <certificate>cert1</certificate>
    <certificate>cert2</certificate>
    <certificate>cert3</certificate>
    <database>/etc/pki/nssdb/</database>
  </smartcard>
  <smartcard mode='passthrough' type='tcp'>
    <source mode='bind' host='127.0.0.1' service='2001' />
    <protocol type='raw' />
    <address type='ccid' controller='0' slot='0' />
  </smartcard>
  <smartcard mode='passthrough' type='spicevmc' />
</devices>
...
```

Figure 20.35. Devices - smartcard devices

The **smartcard** element has a mandatory attribute **mode**. The following modes are supported; in each mode, the guest virtual machine sees a device on its USB bus that behaves like a physical USB CCID (Chip/Smart Card Interface Device) card.

The mode attributes are as follows:

Table 20.16. Smartcard mode elements

Parameter	Description
<code>mode='host'</code>	In this mode, the hypervisor relays all requests from the guest virtual machine into direct access to the host physical machine's smartcard via NSS. No other attributes or sub-elements are required. See below about the use of an optional address sub-element.
<code>mode='host-certificates'</code>	This mode allows you to provide three NSS certificate names residing in a database on the host physical machine, rather than requiring a smartcard to be plugged into the host physical machine. These certificates can be generated via the command <code>certutil -d /etc/pki/nssdb -x -t CT,CT,CT -S -s CN=cert1 -n cert1</code> , and the resulting three certificate names must be supplied as the content of each of three certificate sub-elements. An additional sub-element database can specify the absolute path to an alternate directory (matching the -d flag of the certutil command when creating the certificates); if not present, it defaults to <code>/etc/pki/nssdb</code> .
<code>mode='passthrough'</code>	Using this mode allows you to tunnel all requests through a secondary character device to a third-party provider (which may in turn be talking to a smartcard or using three certificate files, rather than having the hypervisor directly communicate with the host physical machine. In this mode of operation, an additional attribute type is required, matching one of the supported serial device types, to describe the host physical machine side of the tunnel; type='tcp' or type='spicevmc' (which uses the smartcard channel of a SPICE graphics device) are typical. Further sub-elements, such as source , may be required according to the given type, although a target sub-element is not required (since the consumer of the character device is the hypervisor itself, rather than a device visible in the guest virtual machine).

Each mode supports an optional sub-element **address**, which fine-tunes the correlation between the smartcard and a ccid bus controller (Refer to [Section 20.16.3, “Device addresses”](#)).

20.16.9. Network interfaces

The network interface devices are modified using management tools that will configure the following part of the Domain XML:

```
...
<devices>
  <interface type='bridge'>
    <source bridge='xenbr0' />
    <mac address='00:16:3e:5d:c7:9e' />
    <script path='vif-bridge' />
    <boot order='1' />
    <rom bar='off' />
  </interface>
</devices>
...
```

Figure 20.36. Devices - network interfaces

There are several possibilities for specifying a network interface visible to the guest virtual machine. Each subsection below provides more details about common setup options. Additionally, each **<interface>** element has an optional **<address>** sub-element that can tie the interface to a particular pci slot, with attribute **type='pci'** (Refer to [Section 20.16.3, “Device addresses”](#)).

20.16.9.1. Virtual networks

This is the recommended configuration for general guest virtual machine connectivity on host physical machines with dynamic / wireless networking configurations (or multi-host physical machine environments where the host physical machine hardware details are described separately in a **<network>** definition). In addition, it provides a connection whose details are described by the named network definition. Depending on the virtual network's **forward mode** configuration, the network may be totally isolated (no **<forward>** element given), NAT'ing to an explicit network device or to the default route (**forward mode='nat'**), routed with no NAT (**forward mode='route'**), or connected directly to one of the host physical machine's network interfaces (via macvtap) or bridge devices (**forward mode='bridge|private|vepa|passthrough'**)

For networks with a forward mode of bridge, private, vepa, and passthrough, it is assumed that the host physical machine has any necessary DNS and DHCP services already setup outside the scope of libvirt. In the case of isolated, nat, and routed networks, DHCP and DNS are provided on the virtual network by libvirt, and the IP range can be determined by examining the virtual network config with **virsh net-dumpxml [networkname]**. There is one virtual network called 'default' setup out of the box which does NAT'ing to the default route and has an IP range of 192.168.122.0/255.255.255.0. Each guest virtual machine will have an associated tun device created with a name of vnetN, which can also be overridden with the **<target>** element (refer to [Section 20.16.9.11, “Overriding the target element”](#)).

When the source of an interface is a network, a portgroup can be specified along with the name of the network; one network may have multiple portgroups defined, with each portgroup containing slightly different configuration information for different classes of network connections. Also, similar to **<direct>** network connections (described below), a connection of type **network** may specify a **<virtualport>** element, with configuration data to be forwarded to a vepa (802.1Qbg) or 802.1Qbh compliant switch, or to an Open vSwitch virtual switch.

Since the actual type of switch may vary depending on the configuration in the **<network>** on the host physical machine, it is acceptable to omit the virtualport type attribute, and specify attributes from multiple different virtualport types (and also to leave out certain attributes); at domain startup time, a complete **<virtualport>** element will be constructed by merging together the type and attributes defined in the network and the portgroup referenced by the interface. The newly-constructed virtualport is a combination of both. The attributes from lower virtualport can't make changes on the ones defined in higher virtualport. Interfaces take the highest priority, portgroup is lowest priority.

For example, in order to work properly with both an 802.1Qbh switch and an Open vSwitch switch, you may choose to specify no type, but both an **profileid** (in case the switch is 802.1Qbh) and an **interfaceid** (in case the switch is Open vSwitch) (you may also omit the other attributes, such as **managerid**, **typeid**, or **profileid**, to be filled in from the network's **virtualport**). If you want to limit a guest virtual machine to connecting only to certain types of switches, you can specify the virtualport type, but still omit some/all of the parameters - in this case if the host physical machine's network has a different type of virtualport, connection of the interface will fail. The virtual network parameters are defined using management tools that modify the following part of the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
  </interface>
  ...
  <interface type='network'>
    <source network='default' portgroup='engineering' />
    <target dev='vnet7' />
    <mac address="00:11:22:33:44:55" />
    <virtualport>
      <parameters instanceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f' />
    </virtualport>
  </interface>
</devices>
...
```

Figure 20.37. Devices - network interfaces- virtual networks

20.16.9.2. Bridge to LAN

Note that this is the recommended configuration setting for general guest virtual machine connectivity on host physical machines with static wired networking configurations.

Bridge to LAN provides a bridge from the guest virtual machine directly onto the LAN. This assumes there is a bridge device on the host physical machine which has one or more of the host physical machines physical NICs enslaved. The guest virtual machine will have an associated **tun** device created with a name of **<vnetN>**, which can also be overridden with the **<target>** element (refer to [Section 20.16.9.11, “Overriding the target element”](#)). The **<tun>** device will be enslaved to the bridge. The IP range / network configuration is whatever is used on the LAN. This provides the guest virtual machine full incoming and outgoing net access just like a physical machine.

On Linux systems, the bridge device is normally a standard Linux host physical machine bridge. On host physical machines that support Open vSwitch, it is also possible to connect to an open vSwitch bridge device by adding a **virtualport type='openvswitch'** to the interface definition. The Open vSwitch type virtualport accepts two parameters in its **parameters** element - an **interfaceid** which is a standard uuid used to uniquely identify this particular interface to Open vSwitch (if you do not specify one, a random **interfaceid** will be generated for you when you first define the interface), and an optional **profileid** which is sent to Open vSwitch as the interfaces **<port-profile>**. To set the bridge to LAN settings, use a management tool that will configure the following part of the domain XML:

```

...
<devices>
...
<interface type='bridge'>
  <source bridge='br0' />
</interface>
<interface type='bridge'>
  <source bridge='br1' />
  <target dev='vnet7' />
  <mac address="00:11:22:33:44:55" />
</interface>
<interface type='bridge'>
  <source bridge='ovsbr' />
  <virtualport type='openvswitch'>
    <parameters profileid='menial' interfaceid='09b11c53-8b5c-4eeb-8f00-
d84eaa0aaa4f' />
  </virtualport>
</interface>
...
</devices>

```

Figure 20.38. Devices - network interfaces- bridge to LAN

20.16.9.3. Setting a port masquerading range

In cases where you want to set the port masquerading range, the port can be set as follows:

```

<forward mode='nat'>
  <address start='1.2.3.4' end='1.2.3.10' />
</forward> ...

```

Figure 20.39. Port Masquerading Range

These values should be set using the **iptables** commands as shown in [Section 18.2, “Network Address Translation”](#).

20.16.9.4. Userspace SLIRP stack

Setting the userspace SLIRP stack parameters provides a virtual LAN with NAT to the outside world. The virtual network has DHCP and DNS services and will give the guest virtual machine an IP addresses starting from 10.0.2.15. The default router will be 10.0.2.2 and the DNS server will be 10.0.2.3. This networking is the only option for unprivileged users who need their guest virtual machines to have outgoing access.

The userspace SLIP stack parameters are defined in the following part of the domain XML::

```

...
<devices>
  <interface type='user' />
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55"/>
  </interface>
</devices>
...

```

Figure 20.40. Devices - network interfaces- Userspace SLIRP stack

20.16.9.5. Generic Ethernet connection

Provides a means for the administrator to execute an arbitrary script to connect the guest virtual machine's network to the LAN. The guest virtual machine will have a **tun** device created with a name of **vnetN**, which can also be overridden with the **target** element. After creating the **tun** device a shell script will be run which is expected to do whatever host physical machine network integration is required. By default this script is called **/etc/qemu-ifup** but can be overridden (refer to [Section 20.16.9.11, "Overriding the target element"](#)).

The generic Ethernet connection parameters are defined in the following part of the domain XML:

```

...
<devices>
  <interface type='ethernet' />
  ...
  <interface type='ethernet'>
    <target dev='vnet7' />
    <script path='/etc/qemu-ifup-mynet' />
  </interface>
</devices>
...

```

Figure 20.41. Devices - network interfaces- generic Ethernet connection

20.16.9.6. Direct attachment to physical interfaces

Manipulating the direct attachment to physical interfaces provides direct attachment of the guest virtual machine's NIC to the given taht the physial interface of the host physical machine is specified.

This setup requires the Linux macvtap driver to be available. One of the modes **vepa** ('Virtual Ethernet Port Aggregator'), **bridge** or **private** can be chosen for the operation mode of the macvtap device, **vepa** being the default mode.

Manipulating direct attachment to physical interfaces involves setting the following parameters in the following part of the domain XML.

```

...
<devices>
  ...
    <interface type='direct'>
      <source dev='eth0' mode='vepa'/>
    </interface>
  </devices>
...

```

Figure 20.42. Devices - network interfaces- direct attachment to physical interfaces

The individual modes cause the delivery of packets to behave as shown in [Table 20.17, “Direct attachment to physical interface elements”](#):

Table 20.17. Direct attachment to physical interface elements

Element	Description
vepa	All of the guest virtual machines' packets are sent to the external bridge. Packets whose destination is a guest virtual machine on the same host physical machine as where the packet originates from are sent back to the host physical machine by the VEPA capable bridge (today's bridges are typically not VEPA capable).
bridge	Packets whose destination is on the same host physical machine as where they originate from are directly delivered to the target macvtap device. Both origin and destination devices need to be in bridge mode for direct delivery. If either one of them is in vepa mode, a VEPA capable bridge is required.
private	All packets are sent to the external bridge and will only be delivered to a target VM on the same host physical machine if they are sent through an external router or gateway and that device sends them back to the host physical machine. This procedure is followed if either the source or destination device is in private mode.
passthrough	This feature attaches a virtual function of a SRIOV capable NIC directly to a guest virtual machine without losing the migration capability. All packets are sent to the VF/IF of the configured network device. Depending on the capabilities of the device additional prerequisites or limitations may apply; for example, this requires kernel 2.6.38 or newer.

The network access of direct attached virtual machines can be managed by the hardware switch to which the physical interface of the host physical machine machine is connected to.

The interface can have additional parameters as shown below, if the switch is conforming to the IEEE 802.1Qbg standard. The parameters of the virtualport element are documented in more detail in the IEEE

802.1Qbg standard. The values are network specific and should be provided by the network administrator. In 802.1Qbg terms, the Virtual Station Interface (VSI) represents the virtual interface of a virtual machine.

Note that IEEE 802.1Qbg requires a non-zero value for the VLAN ID.

Additional elements that can be manipulated are described in [Table 20.18, “Direct attachment to physical interface additional elements”](#):

Table 20.18. Direct attachment to physical interface additional elements

Element	Description
managerid	The VSI Manager ID identifies the database containing the VSI type and instance definitions. This is an integer value and the value 0 is reserved.
typeid	The VSI Type ID identifies a VSI type characterizing the network access. VSI types are typically managed by network administrator. This is an integer value.
typeidversion	The VSI Type Version allows multiple versions of a VSI Type. This is an integer value.
instanceid	The VSI Instance ID Identifier is generated when a VSI instance (i.e. a virtual interface of a virtual machine) is created. This is a globally unique identifier.
profileid	The profile ID contains the name of the port profile that is to be applied onto this interface. This name is resolved by the port profile database into the network parameters from the port profile, and those network parameters will be applied to this interface.

Additional parameters in the domain XML include:

```

...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0.2' mode='vepa' />
    <virtualport type="802.1Qbg">
      <parameters managerid="11" typeid="1193047" typeidversion="2"
instanceid="09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f" />
    </virtualport>
  </interface>
</devices>
...

```

Figure 20.43. Devices - network interfaces- direct attachment to physical interfaces additional parameters

The interface can have additional parameters as shown below if the switch is conforming to the IEEE 802.1Qbh standard. The values are network specific and should be provided by the network

administrator.

Additional parameters in the domain XML include:

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private' />
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance' />
    </virtualport>
  </interface>
</devices>
...
```

Figure 20.44. Devices - network interfaces- direct attachment to physical interfaces more additional parameters

The **profileid** attribute, contains the name of the port profile that is to be applied to this interface. This name is resolved by the port profile database into the network parameters from the port profile, and those network parameters will be applied to this interface.

20.16.9.7. PCI passthrough

A PCI network device (specified by the **source** element) is directly assigned to the guest virtual machine using generic device passthrough, after first optionally setting the device's MAC address to the configured value, and associating the device with an 802.1Qbh capable switch using an optionally specified **virtualport** element (see the examples of virtualport given above for type='direct' network devices). Note that - due to limitations in standard single-port PCI ethernet card driver design - only SR-IOV (Single Root I/O Virtualization) virtual function (VF) devices can be assigned in this manner; to assign a standard single-port PCI or PCIe ethernet card to a guest virtual machine, use the traditional **hostdev** device definition

Note that this "intelligent passthrough" of network devices is very similar to the functionality of a standard **hostdev** device, the difference being that this method allows specifying a MAC address and **virtualport** for the passed-through device. If these capabilities are not required, if you have a standard single-port PCI, PCIe, or USB network card that doesn't support SR-IOV (and hence would anyway lose the configured MAC address during reset after being assigned to the guest virtual machine domain), or if you are using a version of libvirt older than 0.9.11, you should use standard **hostdev** to assign the device to the guest virtual machine instead of **interface type='hostdev'**.

```

...
<devices>
  <interface type='hostdev'>
    <driver name='vfio'/>
    <source>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
function='0x0' />
    </source>
    <mac address='52:54:00:6d:90:02'>
      <virtualport type='802.1Qbh'>
        <parameters profileid='finance' />
      </virtualport>
    </interface>
</devices>
...

```

Figure 20.45. Devices - network interfaces- PCI passthrough

20.16.9.8. Multicast tunnel

A multicast group may be used to represent a virtual network. Any guest virtual machine whose network devices are within the same multicast group will talk to each other, even if they reside across multiple physical host physical machines. This mode may be used as an unprivileged user. There is no default DNS or DHCP support and no outgoing network access. To provide outgoing network access, one of the guest virtual machines should have a second NIC which is connected to one of the first 4 network types in order to provide appropriate routing. The multicast protocol is compatible with protocols used by **user mode** linux guest virtual machines as well. Note that the source address used must be from the multicast address block. A multicast tunnel is created by manipulating the **interface type** using a management tool and setting/changing it to **mcast**, and providing a mac and source address. The result is shown in changes made to the domain XML:

```

...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
      <source address='230.0.0.1' port='5558' />
    </interface>
</devices>
...

```

Figure 20.46. Devices - network interfaces- multicast tunnel

20.16.9.9. TCP tunnel

Creating a TCP client/server architecture is another way to provide a virtual network where one guest virtual machine provides the server end of the network and all other guest virtual machines are configured as clients. All network traffic between the guest virtual machines is routed via the guest virtual machine that is configured as the server. This model is also available for use to unprivileged users. There is no default DNS or DHCP support and no outgoing network access. To provide outgoing

network access, one of the guest virtual machines should have a second NIC which is connected to one of the first 4 network types thereby providing the appropriate routing. A TCP tunnel is created by manipulating the **interface type** using a management tool and setting/changing it to **server** or **client**, and providing a mac and source address. The result is shown in changes made to the domain XML:

```
...
<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
      <source address='192.168.0.1' port='5558' />
    </interface>
  ...
  <interface type='client'>
    <mac address='52:54:00:8b:c9:51'>
      <source address='192.168.0.1' port='5558' />
    </interface>
</devices>
...
```

Figure 20.47. Devices - network interfaces- TCP tunnel

20.16.9.10. Setting NIC driver-specific options

Some NICs may have tunable driver-specific options. These options are set as attributes of the **driver** sub-element of the interface definition. These options are set by using management tools to configuring the following sections of the domain XML:

```
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet1' />
    <model type='virtio' />
    <driver name='vhost' txmode='iothread' ioeventfd='on' event_idx='off' />
  </interface>
</devices>
...
```

Figure 20.48. Devices - network interfaces- setting NIC driver-specific options

Currently the following attributes are available for the "virtio" NIC driver:

Table 20.19. virtio NIC driver elements

Parameter	Description
name	The optional name attribute forces which type of backend driver to use. The value can be either qemu (a user-space backend) or vhost (a kernel backend, which requires the vhost module to be provided by the kernel); an attempt to require the vhost driver without kernel support will be rejected. The default setting is vhost if the vhost driver present, but will silently fall back to qemu if not.
txmode	Specifies how to handle transmission of packets when the transmit buffer is full. The value can be either iothread or timer . If set to iothread , packet tx is all done in an iothread in the bottom half of the driver (this option translates into adding " tx=bh " to the qemu commandline -device virtio-net-pci option). If set to timer , tx work is done in qemu, and if there is more tx data than can be sent at the present time, a timer is set before qemu moves on to do other things; when the timer fires, another attempt is made to send more data. In general you should leave this option alone, unless you are very certain you that changing it is an absolute necessity.
ioeventfd	Allows users to set domain I/O asynchronous handling for interface device. The default is left to the discretion of the hypervisor. Accepted values are on and off . Enabling this option allows qemu to execute a guest virtual machine while a separate thread handles I/O. Typically guest virtual machines experiencing high system CPU utilization during I/O will benefit from this. On the other hand, overloading the physical host physical machine may also increase guest virtual machine I/O latency. Therefore, you should leave this option alone, unless you are very certain you that changing it is an absolute necessity.
event_idx	The event_idx attribute controls some aspects of device event processing. The value can be either on or off . Choosing on , reduces the number of interrupts and exits for the guest virtual machine. The default is on . In case there is a situation where this behavior is suboptimal, this attribute provides a way to force the feature off. You should leave this option alone, unless you are very certain you that changing it is an absolute necessity.

20.16.9.11. Overriding the target element

To override the target element, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet1' />
  </interface>
</devices>
...
```

Figure 20.49. Devices - network interfaces- overriding the target element

If no target is specified, certain hypervisors will automatically generate a name for the created tun device. This name can be manually specified, however the name must not start with either 'vnet' or 'vif', which are prefixes reserved by libvirt and certain hypervisors. Manually specified targets using these prefixes will be ignored.

20.16.9.12. Specifying boot order

To specify the boot order, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet1' />
    <boot order='1' />
  </interface>
</devices>
...
```

Figure 20.50. Specifying boot order

For hypervisors which support it, you can set a specific NIC to be used for the network boot. The order of attributes determine the order in which devices will be tried during boot sequence. Note that the per-device boot elements cannot be used together with general boot elements in BIOS bootloader section.

20.16.9.13. Interface ROM BIOS configuration

To specify the ROM BIOS configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet1' />
    <rom bar='on' file='/etc/fake/boot.bin' />
  </interface>
</devices>
...
```

Figure 20.51. Interface ROM BIOS configuration

For hypervisors which support it, you can change how a PCI Network device's ROM is presented to the guest virtual machine. The **bar** attribute can be set to **on** or **off**, and determines whether or not the device's ROM will be visible in the guest virtual machine's memory map. (In PCI documentation, the "rombar" setting controls the presence of the Base Address Register for the ROM). If no rom bar is specified, the qemu default will be used (older versions of qemu used a default of **off**, while newer qemus have a default of **on**). The optional **file** attribute is used to point to a binary file to be presented to the guest virtual machine as the device's ROM BIOS. This can be useful to provide an alternative boot ROM for a network device.

20.16.9.14. Quality of service

This section of the domain XML provides setting quality of service. Incoming and outgoing traffic can be shaped independently. The **bandwidth** element can have at most one inbound and at most one outbound child elements. Leaving any of these children element out results in no QoS being applied on that traffic direction. Therefore, when you want to shape only domain's incoming traffic, use inbound only, and vice versa.

Each of these elements has one mandatory attribute **average** (or **floor** as described below). **average** specifies average bit rate on the interface being shaped. Then there are two optional attributes: **peak**, which specifies maximum rate at which interface can send data, and **burst**, which specifies the amount of bytes that can be burst at peak speed. Accepted values for attributes are integer numbers.

The units for **average** and **peak** attributes are kilobytes per second, whereas **burst** is only set in kilobytes. In addition, inbound traffic can optionally have a **floor** attribute. This guarantees minimal throughput for shaped interfaces. Using the **floor** requires that all traffic goes through one point where QoS decisions can take place. As such it may only be used in cases where the **interface type='network'** / with a **forward** type of **route**, **nat**, or no forward at all). It should be noted that within a virtual network, all connected interfaces are required to have at least the inbound QoS set (**average** at least) but the floor attribute doesn't require specifying **average**. However, **peak** and **burst** attributes still require **average**. At the present time, ingress qdiscs may not have any classes, and therefore **floor** may only be applied only on inbound and not outbound traffic.

To specify the QoS configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet0' />
    <bandwidth>
      <inbound average='1000' peak='5000' floor='200' burst='1024' />
      <outbound average='128' peak='256' burst='256' />
    </bandwidth>
  </interface>
</devices>
...
```

Figure 20.52. Quality of service

20.16.9.15. Setting VLAN tag (on supported network types only)

To specify the VLAN tag configuration settings, use a management tool to make the following changes to the domain XML:

```

...
<devices>
  <interface type='bridge'>
    <vlan>
      <tag id='42'/>
    </vlan>
    <source bridge='ovsbr0' />
    <virtualport type='openvswitch'>
      <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f' />
    </virtualport>
  </interface>
<devices>
...

```

Figure 20.53. Setting VLAN tag (on supported network types only)

If (and only if) the network connection used by the guest virtual machine supports vlan tagging transparent to the guest virtual machine, an optional **vlan** element can specify one or more **vlan** tags to apply to the guest virtual machine's network traffic (openvswitch and **type='hostdev'** SR-IOV interfaces do support transparent vlan tagging of guest virtual machine traffic; everything else, including standard Linux bridges and libvirt's own virtual networks, do not support it. 802.1Qbh (vn-link) and 802.1Qbg (VEPA) switches provide their own way (outside of libvirt) to tag guest virtual machine traffic onto specific vlans.) To allow for specification of multiple tags (in the case of vlan trunking), a subelement, **tag**, specifies which vlan tag to use (for example: **tag id='42'**/. If an interface has more than one **vlan** element defined, it is assumed that the user wants to do VLAN trunking using all the specified tags. In the case that vlan trunking with a single tag is desired, the optional attribute **trunk='yes'** can be added to the toplevel **vlan** element.

20.16.9.16. Modifying virtual link state

This element provides means of setting state of the virtual network link. Possible values for attribute **state** are **up** and **down**. If **down** is specified as the value, the interface behaves as if it had the network cable disconnected. Default behavior if this element is unspecified is to have the link state **up**.

To specify the virtual link state configuration settings, use a management tool to make the following changes to the domain XML:

```

...
<devices>
  <interface type='network'>
    <source network='default' />
    <target dev='vnet0' />
    <link state='down' />
  </interface>
<devices>
...

```

Figure 20.54. Modifying virtual link state

20.16.10. Input devices

Input devices allow interaction with the graphical framebuffer in the guest virtual machine virtual machine. When enabling the framebuffer, an input device is automatically provided. It may be possible to add additional devices explicitly, for example, to provide a graphics tablet for absolute cursor movement.

To specify the input devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <input type='mouse' bus='usb' />
</devices>
...
```

Figure 20.55. Input devices

The **<input>** element has one mandatory attribute: **type** which can be set to: **mouse** or **tablet**. The latter provides absolute cursor movement, while the former uses relative movement. The optional **bus** attribute can be used to refine the exact device type and can be set to: **xen** (paravirtualized), **ps2**, and **usb**.

The input element has an optional sub-element **<address>**, which can tie the device to a particular PCI slot, as documented above.

20.16.11. Hub devices

A hub is a device that expands a single port into several so that there are more ports available to connect devices to a host physical machine system.

To specify the hub devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <hub type='usb' />
</devices>
...
```

Figure 20.56. Hub devices

The hub element has one mandatory attribute, the type whose value can only be **usb**. The hub element has an optional sub-element **address** with **type='usb'** which can tie the device to a particular controller.

20.16.12. Graphical framebuffers

A graphics device allows for graphical interaction with the guest virtual machine OS. A guest virtual machine will typically have either a framebuffer or a text console configured to allow interaction with the admin.

To specify the graphical framebuffer devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <graphics type='sdl' display=':0.0' />
  <graphics type='vnc' port='5904'>
    <listen type='address' address='1.2.3.4' />
  </graphics>
  <graphics type='rdp' autoport='yes' multiUser='yes' />
  <graphics type='desktop' fullscreen='yes' />
  <graphics type='spice'>
    <listen type='network' network='rednet' />
  </graphics>
</devices>
...
```

Figure 20.57. Graphical framebuffers

The **graphics** element has a mandatory **type** attribute which takes the value **sdl**, **vnc**, **rdp** or **desktop** as explained below:

Table 20.20. Graphical framebuffer elements

Parameter	Description
sdl	This displays a window on the host physical machine desktop, it can take 3 optional arguments: a display attribute for the display to use, an xauth attribute for the authentication identifier, and an optional fullscreen attribute accepting values yes or no
vnc	Starts a VNC server. The port attribute specifies the TCP port number (with -1 as legacy syntax indicating that it should be auto-allocated). The autoport attribute is the new preferred syntax for indicating autoallocation of the TCP port to use. The listen attribute is an IP address for the server to listen on. The passwd attribute provides a VNC password in clear text. The keymap attribute specifies the keymap to use. It is possible to set a limit on the validity of the password by giving an timestamp passwdValidTo='2010-04-09T15:51:00' assumed to be in UTC. The connected attribute allows control of connected client during password changes. VNC accepts keep value only and note that it may not be supported by all hypervisors. Rather than using listen/port , QEMU supports a socket attribute for listening on a unix domain socket path.
spice	Starts a SPICE server. The port attribute specifies the TCP port number (with -1 as legacy syntax indicating that it should be auto-allocated), while tlsPort gives an alternative secure port number. The autoport attribute is the new preferred syntax for indicating auto-allocation of both port numbers. The listen attribute is an IP address for the server to listen on. The passwd attribute provides a SPICE password in clear text. The keymap attribute specifies the keymap to use. It is possible to set a limit on the validity of the password by giving an timestamp passwdValidTo='2010-04-09T15:51:00' assumed to be in UTC. The connected attribute allows control of connected client during password changes. SPICE accepts keep to keep client connected, disconnect to disconnect client and fail to fail changing password. Note it is not supported by all hypervisors. The defaultMode attribute sets the default channel security policy, valid values are secure , insecure and the default any (which is secure if possible, but falls back to insecure rather than erroring out if no secure path is available).

When SPICE has both a normal and TLS secured TCP port configured, it may be desirable to restrict what channels can be run on each port. This is achieved by adding one or more **channel** elements

inside the main **graphics** element. Valid channel names include **main**, **display**, **inputs**, **cursor**, **playback**, **record**; **smartcard**; and **usbredir**.

To specify the SPICE configuration settings, use a management tool to make the following changes to the domain XML:

```
<graphics type='spice' port='-1' tlsPort='-1' autoport='yes'>
  <channel name='main' mode='secure'/>
  <channel name='record' mode='insecure'/>
  <image compression='auto_glz'/>
  <streaming mode='filter'/>
  <clipboard copypaste='no'/>
  <mouse mode='client'/>
</graphics>
```

Figure 20.58. SPICE configuration

SPICE supports variable compression settings for audio, images and streaming. These settings are accessible via the **compression** attribute in all following elements: **image** to set image compression (accepts `auto_glz`, `auto_lz`, `quic`, `glz`, `lz`, `off`), **jpeg** for JPEG compression for images over wan (accepts `auto`, `never`, `always`), **zlib** for configuring wan image compression (accepts `auto`, `never`, `always`) and **playback** for enabling audio stream compression (accepts `on` or `off`).

Streaming mode is set by the **streaming** element, setting its **mode** attribute to one of **filter**, **all** or **off**.

In addition, Copy and paste functionality (via the SPICE agent) is set by the **clipboard** element. It is enabled by default, and can be disabled by setting the **copypaste** property to **no**.

Mouse mode is set by the **mouse** element, setting its **mode** attribute to one of **server** or **client**. If no mode is specified, the qemu default will be used (**client** mode).

Additional elements include:

Table 20.21. Additional graphical framebuffer elements

Parameter	Description
rdp	Starts a RDP server. The port attribute specifies the TCP port number (with -1 as legacy syntax indicating that it should be auto-allocated). The autoport attribute is the new preferred syntax for indicating autoallocation of the TCP port to use. The replaceUser attribute is a boolean deciding whether multiple simultaneous connections to the VM are permitted. The multiUser attribute is a boolean deciding whether the existing connection must be dropped and a new connection must be established by the VRDP server, when a new client connects in single connection mode.
desktop	This value is reserved for VirtualBox domains for the moment. It displays a window on the host physical machine desktop, similarly to "sdl", but uses the VirtualBox viewer. Just like "sdl", it accepts the optional attributes display and fullscreen.
listen	Rather than putting the address information used to set up the listening socket for graphics types vnc and spice in the graphics , the listen attribute, a separate subelement of graphics , called listen can be specified (see the examples above). listen accepts the following attributes: <ul style="list-style-type: none"> ▶ type - Set to either address or network. This tells whether this listen element is specifying the address to be used directly, or by naming a network (which will then be used to determine an appropriate address for listening). ▶ address - this attribute will contain either an IP address or hostname (which will be resolved to an IP address via a DNS query) to listen on. In the "live" XML of a running domain, this attribute will be set to the IP address used for listening, even if type='network'. ▶ network - if type='network', the network attribute will contain the name of a network in libvirt's list of configured networks. The named network configuration will be examined to determine an appropriate listen address. For example, if the network has an IPv4 address in its configuration (e.g. if it has a forward type of route, nat, or no forward type (isolated)), the first IPv4 address listed in the network's configuration will be used. If the network is describing a host physical machine bridge, the first IPv4 address associated with that bridge device will be used, and if the network is describing one of the 'direct' (macvtap) modes, the first IPv4 address of the first

forward dev will be used.

20.16.13. Video devices

A video device.

To specify the video devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <video>
    <model type='vga' vram='8192' heads='1'>
      <acceleration accel3d='yes' accel2d='yes'/>
    </model>
  </video>
</devices>
...
```

Figure 20.59. Video devices

The **graphics** element has a mandatory **type** attribute which takes the value "sdl", "vnc", "rdp" or "desktop" as explained below:

Table 20.22. Graphical framebuffer elements

Parameter	Description
video	The video element is the container for describing video devices. For backwards compatibility, if no video is set but there is a graphics element in domain xml, then libvirt will add a default video according to the guest virtual machine type. If "ram" or "vram" are not supplied a default value is used.
model	This has a mandatory type attribute which takes the value vga , cirrus , vmvga , xen , vbox , or qxl depending on the hypervisor features available. You can also provide the amount of video memory in kibibytes (blocks of 1024 bytes) using vram and the number of figure with heads .
acceleration	If acceleration is supported it should be enabled using the accel3d and accel2d attributes in the acceleration element.
address	The optional address sub-element can be used to tie the video device to a particular PCI slot.

20.16.14. Consoles, serial, parallel, and channel devices

A character device provides a way to interact with the virtual machine. Paravirtualized consoles, serial ports, parallel ports and channels are all classed as character devices and so represented using the same syntax.

To specify the consols, channel and other devices configuration settings, use a management tool to

make the following changes to the domain XML:

```
...
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2' />
    <target port='0' />
  </parallel>
  <serial type='pty'>
    <source path='/dev/pts/3' />
    <target port='0' />
  </serial>
  <console type='pty'>
    <source path='/dev/pts/4' />
    <target port='0' />
  </console>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd' />
    <target type='guestfwd' address='10.0.2.1' port='4600' />
  </channel>
</devices>
...
```

Figure 20.60. Consoles, serial, parallel, and channel devices

In each of these directives, the top-level element name (parallel, serial, console, channel) describes how the device is presented to the guest virtual machine. The guest virtual machine interface is configured by the target element. The interface presented to the host physical machine is given in the type attribute of the top-level element. The host physical machine interface is configured by the source element. The source element may contain an optional seclabel to override the way that labelling is done on the socket path. If this element is not present, the security label is inherited from the per-domain setting. Each character device element has an optional sub-element **address** which can tie the device to a particular controller or PCI slot.

20.16.15. Guest virtual machine interfaces

A character device presents itself to the guest virtual machine as one of the following types.

To set the parallel port, use a management tool to make the following change to the domain XML

```
...
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2' />
    <target port='0' />
  </parallel>
</devices>
...
```

Figure 20.61. Guest virtual machine interface Parallel Port

<target> can have a **port** attribute, which specifies the port number. Ports are numbered starting from 0. There are usually 0, 1 or 2 parallel ports.

To set the serial port use a management tool to make the following change to the domain XML:

```
...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3' />
    <target port='0' />
  </serial>
</devices>
...
```

Figure 20.62. Guest virtual machine interface serial port

<target> can have a **port** attribute, which specifies the port number. Ports are numbered starting from 0. There are usually 0, 1 or 2 serial ports. There is also an optional **type** attribute, which has two choices for its value, one is **isa-serial**, the other is **usb-serial**. If **type** is missing, **isa-serial** will be used by default. For usb-serial an optional sub-element **<address>** with **type='usb'** can tie the device to a particular controller, documented above.

The **<console>** element is used to represent interactive consoles. Depending on the type of guest virtual machine in use, the consoles might be paravirtualized devices, or they might be a clone of a serial device, according to the following rules:

- ▶ If no **targetType** attribute is set, then the default device **type** is according to the hypervisor's rules. The default **type** will be added when re-querying the XML fed into libvirt. For fully virtualized guest virtual machines, the default device type will usually be a serial port.
- ▶ If the **targetType** attribute is **serial**, and if no **<serial>** element exists, the console element will be copied to the **<serial>** element. If a **<serial>** element does already exist, the console element will be ignored.
- ▶ If the **targetType** attribute is not **serial**, it will be treated normally.
- ▶ Only the first **<console>** element may use a **targetType** of **serial**. Secondary consoles must all be paravirtualized.
- ▶ On s390, the console element may use a targetType of **sclp** or **sclplm** (line mode). SCLP is the native console type for s390. There's no controller associated to SCLP consoles.

In the example below, a virtio console device is exposed in the guest virtual machine as `/dev/hvc[0-7]` (for more information, see <http://fedoraproject.org/wiki/Features/VirtioSerial>):

```

...
<devices>
  <console type='pty'>
    <source path='/dev/pts/4'/>
    <target port='0' />
  </console>

  <!-- KVM virtio console -->
  <console type='pty'>
    <source path='/dev/pts/5' />
    <target type='virtio' port='0' />
  </console>
</devices>
...

...
<devices>
  <!-- KVM s390 sclp console -->
  <console type='pty'>
    <source path='/dev/pts/1' />
    <target type='sclp' port='0' />
  </console>
</devices>
...

```

Figure 20.63. Guest virtual machine interface - virtio console device

If the console is presented as a serial port, the `<target>` element has the same attributes as for a serial port. There is usually only one console.

20.16.16. Channel

This represents a private communication channel between the host physical machine and the guest virtual machine and is manipulated by making changes to your guest virtual machine virtual machine using a management tool that results in changes made to the following section of the domain xml

```

...
<devices>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd' />
    <target type='guestfwd' address='10.0.2.1' port='4600' />
  </channel>

  <!-- KVM virtio channel -->
  <channel type='pty'>
    <target type='virtio' name='arbitrary.virtio.serial.port.name' />
  </channel>
  <channel type='unix'>
    <source mode='bind' path='/var/lib/libvirt/qemu/f16x86_64.agent' />
    <target type='virtio' name='org.qemu.guest_agent.0' />
  </channel>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0' />
  </channel>
</devices>
...

```

Figure 20.64. Channel

This can be implemented in a variety of ways. The specific type of `<channel>` is given in the `type` attribute of the `<target>` element. Different channel types have different target attributes as follows:

- ▶ **guestfwd** - Dictates that TCP traffic sent by the guest virtual machine to a given IP address and port is forwarded to the channel device on the host physical machine. The `target` element must have address and port attributes.
- ▶ **virtio** - Paravirtualized virtio channel. `<channel>` is exposed in the guest virtual machine under `/dev/vport*`, and if the optional element `name` is specified, `/dev/virtio-ports/$name` (for more info, please see <http://fedoraproject.org/wiki/Features/VirtioSerial>). The optional element `address` can tie the channel to a particular `type='virtio-serial'` controller, documented above. With QEMU, if name is "org.qemu.guest_agent.0", then libvirt can interact with a guest virtual machine agent installed in the guest virtual machine, for actions such as guest virtual machine shutdown or file system quiescing.
- ▶ **spicevmc** - Paravirtualized SPICE channel. The domain must also have a SPICE server as a graphics device, at which point the host physical machine piggy-backs messages across the main channel. The `target` element must be present, with attribute `type='virtio'`; an optional attribute `name` controls how the guest virtual machine will have access to the channel, and defaults to `name='com.redhat.spice.0'`. The optional `<address>` element can tie the channel to a particular `type='virtio-serial'` controller.

20.16.17. Host physical machine interface

A character device presents itself to the host physical machine as one of the following types:

Table 20.23. Character device elements

Parameter	Description	XML snippet
Domain logfile	Disables all input on the character device, and sends output into the virtual machine's logfile	<pre><devices> <console type='stdio'> <target port='1' /> </console> </devices></pre>
Device logfile	A file is opened and all data sent to the character device is written to the file.	<pre><devices> <serial type="file"> <source path="/var/log/vm/vm-serial.log"/> <target port="1" /> </serial> </devices></pre>
Virtual console	Connects the character device to the graphical framebuffer in a virtual console. This is typically accessed via a special hotkey sequence such as "ctrl+alt+3"	<pre><devices> <serial type='vc'> <target port="1" /> </serial> </devices></pre>
Null device	Connects the character device to the void. No data is ever provided to the input. All data written is discarded.	<pre><devices> <serial type='null'> <target port="1" /> </serial> </devices></pre>
Pseudo TTY	A Pseudo TTY is allocated	

• ८०००० •

using `/dev/ptmx`. A suitable client such as `virsh console` can connect to interact with the serial port locally.

```
<devices>  
  
<serial type="pty">  
  
<source  
path="/dev/pts/3"/>  
  
<target port="1"/>  
  
</serial>  
  
</devices>
```

NB Special case	<p>NB special case if <code><console type='pty'></code>, then the TTY path is also duplicated as an attribute <code>tty='/dev/pts/3'</code> on the top level <code><console></code> tag. This provides compat with existing syntax for <code><console></code> tags.</p>
Host physical machine device proxy	<p>The character device is passed through to the underlying physical character device. The device types must match, eg the emulated serial port should only be connected to a host physical machine serial port - don't connect a serial port to a parallel port.</p>
	<pre><devices> <serial type="dev"> <source path="/dev/ttys0"/> <target port="1"/> </serial> </devices></pre>
Named pipe	<p>The character device writes output to a named pipe. See <code>pipe(7)</code> MAN page for more info.</p>
	<pre><devices> <serial type="pipe"> <source path="/tmp/mypipe"/> <target port="1"/> </serial> </devices></pre>

For `oncharserver`:

The character device acts as a TCP client connecting to a remote server.

```
<devices>
<serial type="tcp">
<source mode="connect" host="0.0.0.0" service="2445"/>
<protocol type="raw"/>
<target port="1"/>
</serial>
</devices>
```

Or as a TCP server waiting for a client connection.

```
<devices>
<serial type="tcp">
<source mode="bind" host="127.0.0.1" service="2445"/>
<protocol type="raw"/>
<target port="1"/>
</serial>
</devices>
```

Alternatively you can use telnet instead of raw TCP. In addition, you can also use telnets (secure telnet) and tls.

UDP network console

The character device acts as a UDP netconsole service, sending and receiving packets. This is a lossy service.

```
<devices>
  <serial type="tcp">
    <source
      mode="connect"
      host="0.0.0.0"
      service="2445"/>
    <protocol
      type="telnet"/>
    <target port="1"/>
  </serial>
  <serial type="tcp">
    <source mode="bind"
      host="127.0.0.1"
      service="2445"/>
    <protocol
      type="telnet"/>
    <target port="1"/>
  </serial>
</devices>
```

UNIX domain socket

The character device acts as a

```
<devices>
  <serial type="udp">
    <source mode="bind"
      host="0.0.0.0"
      service="2445"/>
    <source
      mode="connect"
      host="0.0.0.0"
      service="2445"/>
    <target port="1"/>
  </serial>
</devices>
```

UNIX domain socket
client/server

The character device acts as a
UNIX domain socket server,
accepting connections from local
clients.

```
<devices>
  <serial type="unix">
    <source mode="bind"
      path="/tmp/foo"/>
    <target port="1"/>
  </serial>
</devices>
```

20.17. Sound devices

A virtual sound card can be attached to the host physical machine via the sound element.

```
...
<devices>
  <sound model='es1370' />
</devices>
...
```

Figure 20.65. Virtual sound card

The sound element has one mandatory attribute, **model**, which specifies what real sound device is emulated. Valid values are specific to the underlying hypervisor, though typical choices are '**es1370**', '**sb16**', '**ac97**', and '**ich6**'. In addition, a sound element with ich6 model can have optional sub-elements **codec** to attach various audio codecs to the audio device. If not specified, a default codec will be attached to allow playback and recording. Valid values are '**duplex**' (advertises a line-in and a line-out) and '**micro**' (advertises a speaker and a microphone).

```
...
<devices>
  <sound model='ich6'>
    <codec type='micro'/>
  </sound>
</devices>
...
```

Figure 20.66. Sound devices

Each sound element has an optional sub-element **<address>** which can tie the device to a particular PCI slot, documented above.

20.18. Watchdog device

A virtual hardware watchdog device can be added to the guest virtual machine via the **<watchdog>** element. The watchdog device requires an additional driver and management daemon in the guest virtual machine. As merely enabling the watchdog in the libvirt configuration does not do anything useful

on its own. Currently there is no support notification when the watchdog fires.

```
...
<devices>
  <watchdog model='i6300esb' />
</devices>
...

...
<devices>
  <watchdog model='i6300esb' action='poweroff' />
</devices>
</domain>
```

Figure 20.67. Watchdog device

The following attributes are declared in this XML:

- ▶ **model** - The required **model** attribute specifies what real watchdog device is emulated. Valid values are specific to the underlying hypervisor.
- ▶ The **model** attribute may take the following values:
 - **i6300esb** — the recommended device, emulating a PCI Intel 6300ESB
 - **ib700** — emulates an ISA iBase IB700
- ▶ **action** - The optional **action** attribute describes what action to take when the watchdog expires. Valid values are specific to the underlying hypervisor. The **action** attribute can have the following values:
 - **reset** — default setting, forcefully resets the guest virtual machine
 - **shutdown** — gracefully shuts down the guest virtual machine (not recommended)
 - **poweroff** — forcefully powers off the guest virtual machine
 - **pause** — pauses the guest virtual machine
 - **none** — does nothing
 - **dump** — automatically dumps the guest virtual machine.

Note that the 'shutdown' action requires that the guest virtual machine is responsive to ACPI signals. In the sort of situations where the watchdog has expired, guest virtual machines are usually unable to respond to ACPI signals. Therefore using 'shutdown' is not recommended. In addition, the directory to save dump files can be configured by `auto_dump_path` in file `/etc/libvirt/qemu.conf`.

20.19. Memory balloon device

A virtual memory balloon device is added to all Xen and KVM/QEMU guest virtual machines. It will be seen as `<memballoon>` element. It will be automatically added when appropriate, so there is no need to explicitly add this element in the guest virtual machine XML unless a specific PCI slot needs to be assigned. Note that if the memballoon device needs to be explicitly disabled, `model='none'` may be used.

The following example automatically added device with KVM

```
...
<devices>
  <memballoon model='virtio' />
</devices>
...
```

Figure 20.68. Memory balloon device

Here is an example where the device is added manually with static PCI slot 2 requested

```
...
<devices>
    <memballoon model='virtio'>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />
    </memballoon>
</devices>
</domain>
```

Figure 20.69. Memory balloon device added manually

The required **model** attribute specifies what type of balloon device is provided. Valid values are specific to the virtualization platform are: '**virtio**' which is the default setting with the KVM hypervisor or '**xen**' which is the default setting with the Xen hypervisor.

20.20. Random number generator device

The virtual random number generator device allows the host physical machine to pass through entropy to guest virtual machine operating systems.

```
...
<devices>
    <rng model='virtio'>
        <rate period="2000" bytes="1234"/>
        <backend model='random'>/dev/random</backend>
        <!-- OR -->
        <backend model='egd' type='udp'>
            <source mode='bind' service='1234'>
            <source mode='connect' host physical machine='1.2.3.4' service='1234'>
        </backend>
    </rng>
</devices>
...
```

Figure 20.70. Random number generator device

The random number generator device allows the following attributes/elements:

- ▶ **model** - The required **model** attribute specifies what type of RNG device is provided. '**virtio**' — supported by qemu and virtio-rng kernel module
- ▶ **<backend>** - The **<backend>** element specifies the source of entropy to be used for the domain. The source model is configured using the **model** attribute. Supported source models include '**random**' — **/dev/random** (default setting) or similar device as source and '**egd**' which sets a EGD protocol backend.
- ▶ **backend type='random'** - This **<backend>** type expects a non-blocking character device as input. Examples of such devices are **/dev/random** and **/dev/urandom**. The file name is specified as contents of the **<backend>** element. When no file name is specified the hypervisor default is used.

- ▶ <**backend type='egd'**> - This backend connects to a source using the EGD protocol. The source is specified as a character device. Refer to character device host physical machine interface for more information.

20.21. TPM devices

The TPM device enables a QEMU guest virtual machine to have access to TPM functionality. The TPM passthrough device type provides access to the host physical machine's TPM for one QEMU guest virtual machine. No other software may be using the TPM device, typically `/dev/tpm0`, at the time the QEMU guest virtual machine is started. The following domain XML example shows the usage of the TPM passthrough device

```
...
<devices>
  <tpm model='tpm-tis'>
    <backend type='passthrough'>
      <backend path='/dev/tpm0' />
    </backend>
  </tpm>
</devices>
...
```

Figure 20.71. TPM devices

The **model** attribute specifies what device model QEMU provides to the guest virtual machine. If no model name is provided, tpm-tis will automatically be chosen. The **<backend>** element specifies the type of TPM device. The following types are supported: '**passthrough**' — uses the host physical machine's TPM device and '**passthrough**'. This backend type requires exclusive access to a TPM device on the host physical machine. An example for such a device is `/dev/tpm0`. The filename is specified as path attribute of the source element. If no file name is specified then `/dev/tpm0` is automatically used.

20.22. Security label

The **<seclabel>** element allows control over the operation of the security drivers. There are three basic modes of operation, '**dynamic**' where libvirt automatically generates a unique security label, '**static**' where the application/administrator chooses the labels, or '**none**' where confinement is disabled. With dynamic label generation, libvirt will always automatically relabel any resources associated with the virtual machine. With static label assignment, by default, the administrator or application must ensure labels are set correctly on any resources, however, automatic relabeling can be enabled if desired.

If more than one security driver is used by libvirt, multiple seclabel tags can be used, one for each driver and the security driver referenced by each tag can be defined using the attribute **model**. Valid input XML configurations for the top-level security label are:

```

<seclabel type='dynamic' model='selinux'/>

<seclabel type='dynamic' model='selinux'>
  <baselabel>system_u:system_r:my_svirt_t:s0</baselabel>
</seclabel>

<seclabel type='static' model='selinux' relabel='no'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='static' model='selinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='none'/>

```

Figure 20.72. Security label

If no '**type**' attribute is provided in the input XML, then the security driver default setting will be used, which may be either '**none**' or '**dynamic**'. If a **<baselabel>** is set but no '**type**' is set, then the type is presumed to be '**dynamic**'. When viewing the XML for a running guest virtual machine with automatic resource relabeling active, an additional XML element, **<imagedlabel>**, will be included. This is an output-only element, so will be ignored in user supplied XML documents.

The following elements can be manipulated with the following values:

- ▶ **type** - Either **static**, **dynamic** or **none** to determine whether libvirt automatically generates a unique security label or not.
- ▶ **model** - A valid security model name, matching the currently activated security model
- ▶ **relabel** - Either **yes** or **no**. This must always be **yes** if dynamic label assignment is used. With static label assignment it will default to **no**.
- ▶ **<label>** - If static labelling is used, this must specify the full security label to assign to the virtual domain. The format of the content depends on the security driver in use:
 - **SELinux**: a SELinux context.
 - **AppArmor**: an AppArmor profile.
 - **DAC**: owner and group separated by colon. They can be defined both as user/group names or uid/gid. The driver will first try to parse these values as names, but a leading plus sign can be used to force the driver to parse them as uid or gid.
- ▶ **<baselabel>** - If dynamic labelling is used, this can optionally be used to specify the base security label. The format of the content depends on the security driver in use.
- ▶ **<imagedlabel>** - This is an output only element, which shows the security label used on resources associated with the virtual domain. The format of the content depends on the security driver in use. When relabeling is in effect, it is also possible to fine-tune the labeling done for specific source file names, by either disabling the labeling (useful if the file lives on NFS or other file system that lacks security labeling) or requesting an alternate label (useful when a management application creates a special label to allow sharing of some, but not all, resources between domains). When a seclabel element is attached to a specific path rather than the top-level domain assignment, only the attribute relabel or the sub-element label are supported.

20.23. Example domain XML configuration

QEMU emulated guest virtual machine on x86_64

```

<domain type='qemu'>
  <name>QEmu-fedora-i686</name>
  <uuid>c7a5fdbd-cdaf-9455-926a-d65c16db1809</uuid>
  <memory>219200</memory>
  <currentMemory>219200</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='cdrom' />
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='cdrom'>
      <source file='/home/user/boot.iso' />
      <target dev=' hdc' />
      <readonly/>
    </disk>
    <disk type='file' device='disk'>
      <source file='/home/user/fedora.img' />
      <target dev='hda' />
    </disk>
    <interface type='network'>
      <source network='default' />
    </interface>
    <graphics type='vnc' port=' -1' />
  </devices>
</domain>

```

Figure 20.73. Example domain XML config

KVM hardware accelerated guest virtual machine on i686

```

<domain type='kvm'>
  <name>demo2</name>
  <uuid>4dea24b3-1d52-d8f3-2516-782e98a23fa0</uuid>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
  </os>
  <clock sync="localtime" />
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/demo2.img' />
      <target dev='hda' />
    </disk>
    <interface type='network'>
      <source network='default' />
      <mac address='24:42:53:21:52:45' />
    </interface>
    <graphics type='vnc' port=' -1' keymap='de' />
  </devices>
</domain>

```

Figure 20.74. Example domain XML config

Chapter 21. Troubleshooting

This chapter covers common problems and solutions for Red Hat Enterprise Linux 6 virtualization issues.

Read this chapter to develop an understanding of some of the common problems associated with virtualization technologies. Troubleshooting takes practice and experience which are difficult to learn from a book. It is recommended that you experiment and test virtualization on Red Hat Enterprise Linux 6 to develop your troubleshooting skills.

If you cannot find the answer in this document there may be an answer online from the virtualization community. Refer to [Section B.1, “Online resources”](#) for a list of Linux virtualization websites.

21.1. Debugging and troubleshooting tools

This section summarizes the System Administrator applications, the networking utilities, and debugging tools. You can employ these standard System administration tools and logs to assist with troubleshooting:

- ▶ **kvm_stat** - refer to [Section 21.3, “kvm_stat”](#)
- ▶ **trace-cmd**
- ▶ **ftrace** Refer to the *Red Hat Enterprise Linux Developer Guide*
- ▶ **vmstat**
- ▶ **iostat**
- ▶ **lsof**
- ▶ **systemtap**
- ▶ **crash**
- ▶ **sysrq**
- ▶ **sysrq t**
- ▶ **sysrq w**

These networking tools can assist with troubleshooting virtualization networking problems:

- ▶ **ifconfig**
- ▶ **tcpdump**
The **tcpdump** command 'sniffs' network packets. **tcpdump** is useful for finding network abnormalities and problems with network authentication. There is a graphical version of **tcpdump** named **wireshark**.
- ▶ **brctl**
brctl is a networking tool that inspects and configures the Ethernet bridge configuration in the Linux kernel. You must have root access before performing these example commands:

```
# brctl show
bridge-name      bridge-id          STP   enabled   interfaces
-----
virtbr0          8000.firebaseio.com    yes     eth0

# brctl showmacs virtbr0
port-no          mac-addr           local?    aging timer
1                fe:ff:ff:ff:ff:ff:   yes      0.00
2                fe:ff:ff:fe:ff:   yes      0.00
# brctl showstp virtbr0
virtbr0
bridge-id        8000.firebaseio.com
designated-root  8000.firebaseio.com
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.

- ▶ **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.

21.2. Creating virsh dump files

Executing a **virsh dump** command sends a request to dump the core of a guest virtual machine to a file so errors in the virtual machine can be diagnosed. Running this command may require you to manually ensure proper permissions on file and path specified by the argument **corefilepath**. The **virsh dump** command is similar to a coredump (or the **crash** utility). To create the **virsh dump** file, run:

```
#virsh dump <domain> <corefilepath> [--bypass-cache] { [--live] | [--crash] | [--reset] } [--verbose] [--memory-only]
```

While the domain (guest virtual machine domain name) and corefilepath (location of the newly created core dump file) are mandatory, the following arguments are optional:

- ▶ **--live** creates a dump file on a running machine and doesn't pause it.
- ▶ **--crash** stops the guest virtual machine and generates the dump file. The main difference is that the guest virtual machine will not be listed as Stopped, with the reason as Crashed. Note that in **virt-manager** the status will be listed as Paused.
- ▶ **--reset** will reset the guest virtual machine following a successful dump. Note, these three switches are mutually exclusive.
- ▶ **--bypass-cache** uses O_DIRECT to bypass the file system cache.
- ▶ **--memory-only** the dump file will be saved as an elf file, and will only include domain's memory and cpu common register value. This option is very useful if the domain uses host devices directly.
- ▶ **--verbose** displays the progress of the dump

The entire dump process may be monitored using **virsh domjobinfo** command and can be canceled by running **virsh domjobabort**.

21.3. **kvm_stat**

The **kvm_stat** command is a python script which retrieves runtime statistics from the **kvm** kernel module. The **kvm_stat** command can be used to diagnose guest behavior visible to **kvm**. In particular, performance related issues with guests. Currently, the reported statistics are for the entire system; the behavior of all running guests is reported. To run this script you need to install the *qemu-kvm-tools* package. Refer to the *Red Hat Enterprise Linux Virtualization Host Configuration and Guest Installation Guide*.

The **kvm_stat** command requires that the **kvm** kernel module is loaded and **debugfs** is mounted. If either of these features are not enabled, the command will output the required steps to enable **debugfs** or the **kvm** module. For example:

```
# kvm_stat  
Please mount debugfs ('mount -t debugfs debugfs /sys/kernel/debug')  
and ensure the kvm modules are loaded
```

Mount **debugfs** if required:

```
# mount -t debugfs debugfs /sys/kernel/debug
```

kvm_stat output

The **kvm_stat** command outputs statistics for all guests and the host. The output is updated until the command is terminated (using **Ctrl+c** or the **q** key).

```
# kvm_stat

kvm statistics

efer_reload          94      0
exits                4003074  31272
fpu_reload           1313881  10796
halt_exits           14050    259
halt_wakeup          4496    203
host_state_reload   1638354  24893
hypercalls           0       0
insn_emulation       1093850  1909
insn_emulation_fail  0       0
invlpg               75569    0
io_exits             1596984  24509
irq_exits            21013    363
irq_injections       48039    1222
irq_window           24656    870
largepages            0       0
mmio_exits           11873    0
mmu_cache_miss       42565    8
mmu_flooded          14752    0
mmu_pde_zapped       58730    0
mmu_pte_updated      6       0
mmu_pte_write        138795   0
mmu_recycled          0       0
mmu_shadow_zapped    40358    0
mmu_unsync            793     0
nmi_injections        0       0
nmi_window            0       0
pf_fixed              697731   3150
pf_guest              279349   0
remote_tlb_flush      5       0
request_irq           0       0
signal_exits          1       0
tlb_flush             200190   0
```

Explanation of variables:

efer_reload

The number of Extended Feature Enable Register (EFER) reloads.

exits

The count of all **VMEXIT** calls.

fpu_reload

The number of times a **VMENTRY** reloaded the FPU state. The **fpu_reload** is incremented when a guest is using the Floating Point Unit (FPU).

halt_exits

Number of guest exits due to **halt** calls. This type of exit is usually seen when a guest is idle.

halt_wakeup

Number of wakeups from a **halt**.

host_state_reload

Count of full reloads of the host state (currently tallies MSR setup and guest MSR reads).

hypcalls

Number of guest hypervisor service calls.

insn_emulation

Number of guest instructions emulated by the host.

insn_emulation_fail

Number of failed **insn_emulation** attempts.

io_exits

Number of guest exits from I/O port accesses.

irq_exits

Number of guest exits due to external interrupts.

irq_injections

Number of interrupts sent to guests.

irq_window

Number of guest exits from an outstanding interrupt window.

largepages

Number of large pages currently in use.

mmio_exits

Number of guest exits due to memory mapped I/O (MMIO) accesses.

mmu_cache_miss

Number of KVM MMU shadow pages created.

mmu_flooded

Detection count of excessive write operations to an MMU page. This counts detected write operations not of individual write operations.

mmu_pde_zapped

Number of page directory entry (PDE) destruction operations.

mmu_pte_updated

Number of page table entry (PTE) destruction operations.

mmu_pte_write

Number of guest page table entry (PTE) write operations.

mmu_recycled

Number of shadow pages that can be reclaimed.

mmu_shadow_zapped

Number of invalidated shadow pages.

mmu_unsync

Number of non-synchronized pages which are not yet unlinked.

nmi_injections

Number of Non-maskable Interrupt (NMI) injections to the guest.

nmi_window

Number of guest exits from (outstanding) Non-maskable Interrupt (NMI) windows.

pf_fixed

Number of fixed (non-paging) page table entry (PTE) maps.

pf_guest

Number of page faults injected into guests.

remote_tlb_flush

Number of remote (sibling CPU) Translation Lookaside Buffer (TLB) flush requests.

request_irq

Number of guest interrupt window request exits.

signal_exits

Number of guest exits due to pending signals from the host.

tlb_flush

Number of **tlb_flush** operations performed by the hypervisor.

 **Note**

The output information from the **kvm_stat** command is exported by the KVM hypervisor as pseudo files located in the **/sys/kernel/debug/kvm/** directory.

21.4. 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 host physical machines using the KVM hypervisor.

This section covers how to enable serial console output for fully virtualized guests.

Fully virtualized guest serial console output can be viewed with the **virsh console** command.

Be aware fully virtualized guest serial consoles have some limitations. Present limitations include:

- ▶ output data may be dropped or scrambled.

The serial port is called **ttyS0** on Linux or **COM1** on Windows.

You must configure the virtualized operating system to output information to the virtual serial port.

To output kernel information from a fully virtualized Linux guest into the domain, modify the **/boot/grub/grub.conf** file. Append the following to the **kernel** line: **console=tty0**
console=ttyS0,115200.

```
title Red Hat Enterprise Linux Server (2.6.32-36.x86-64)
root (hd0,0)
kernel /vmlinuz-2.6.32-36.x86-64 ro root=/dev/volgroup00/logvol00 \
console=tty0 console=ttyS0,115200
initrd /initrd-2.6.32-36.x86-64.img
```

Reboot the guest.

On the host, access the serial console with the following command:

```
# virsh console
```

You can also use **virt-manager** to display the virtual text console. In the guest console window, select **Serial 1** in **Text Consoles** from the **View** menu.

21.5. Virtualization log files

- ▶ Each fully virtualized guest log is in the **/var/log/libvirt/qemu/** directory. Each guest log is named as **GuestName.log** and will be periodically compressed once a size limit is reached.

If you encounter any errors with the Virtual Machine Manager, you can review the generated data in the **virt-manager.log** file that resides in the **\$HOME/.virt-manager** directory.

21.6. Loop device errors

If file-based guest images are used you may have to increase the number of configured loop devices. The default configuration allows up to eight active loop devices. If more than eight file-based guests or loop devices are needed the number of loop devices configured can be adjusted in the **/etc/modprobe.d/** directory. Add the following line:

```
options loop max_loop=64
```

This example uses 64 but you can specify another number to set the maximum loop value. You may also have to implement loop device backed guests on your system. To use a loop device backed guests for a full virtualized system, use the **phy: device** or **file: file** commands.

21.7. Live Migration Errors

There may be cases where a live migration causes the memory contents to be re-transferred over and over again. This process causes the guest to be in a state where it is constantly writing to memory and therefore will slow down migration. If this should occur, and the guest is writing more than several tens of MBs per second, then live migration may fail to finish (converge). This issue is not scheduled to be resolved at the moment for Red Hat Enterprise Linux 6, and is scheduled to be fixed in Red Hat Enterprise Linux 7.

The current live-migration implementation has a default migration time configured to 30ms. This value determines the guest pause time at the end of the migration in order to transfer the leftovers. Higher values increase the odds that live migration will converge

21.8. Enabling Intel VT-x and AMD-V virtualization hardware extensions in BIOS

This section describes how to identify hardware virtualization extensions and enable them in your BIOS if they are disabled.

The Intel VT-x extensions can be disabled in the BIOS. Certain laptop vendors have disabled the Intel VT-x extensions by default in their CPUs.

The virtualization extensions cannot be disabled in the BIOS for AMD-V.

Refer to the following section for instructions on enabling disabled virtualization extensions.

Verify the virtualization extensions are enabled in BIOS. The BIOS settings for Intel VT or AMD-V are usually in the **Chipset** or **Processor** menus. The menu names may vary from this guide, the virtualization extension settings may be found in **Security Settings** or other non standard menu names.

Procedure 21.1. Enabling virtualization extensions in BIOS

1. Reboot the computer and open the system's BIOS menu. This can usually be done by pressing the **delete** key, the **F1** key or **Alt** and **F4** keys depending on the system.

2. Enabling the virtualization extensions in BIOS



Note

Many of the steps below may vary depending on your motherboard, processor type, chipset and OEM. Refer to your system's accompanying documentation for the correct information on configuring your system.

- a. Open the **Processor** submenu. The processor settings menu may be hidden in the **Chipset**, **Advanced CPU Configuration** or **Northbridge**.
 - b. Enable **Intel Virtualization Technology** (also known as Intel VT-x). **AMD-V** extensions cannot be disabled in the BIOS and should already be enabled. The virtualization extensions may be labeled **Virtualization Extensions**, **Vanderpool** or various other names depending on the OEM and system BIOS.
 - c. Enable Intel VT-d or AMD IOMMU, if the options are available. Intel VT-d and AMD IOMMU are used for PCI device assignment.
 - d. Select **Save & Exit**.
3. Reboot the machine.
 4. When the machine has booted, run `cat /proc/cpuinfo |grep -E "vmx|svm"`. Specifying **-color** is optional, but useful if you want the search term highlighted. 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.

21.9. KVM networking performance

By default, KVM virtual machines are assigned a virtual Realtek 8139 (rtl8139) NIC (network interface controller). Whereas Red Hat Enterprise Linux guests are assigned a virtio NIC by default, Windows guests or the guest type is not specified.

The rtl8139 virtualized NIC works fine in most environments, but this device can suffer from performance degradation problems on some networks, such as a 10 Gigabit Ethernet.

To improve performance, you can switch to the para-virtualized network driver.

**Note**

Note that the virtualized Intel PRO/1000 (**e1000**) driver is also supported as an emulated driver choice. To use the **e1000** driver, replace **virtio** in the procedure below with **e1000**. For the best performance it is recommended to use the **virtio** driver.

Procedure 21.2. Switching to the virtio driver

1. Shutdown the guest operating system.
2. Edit the guest's configuration file with the **virsh** command (where **GUEST** is the guest's name):

```
# virsh edit GUEST
```

The **virsh edit** command uses the **\$EDITOR** shell variable to determine which editor to use.

3. Find the network interface section of the configuration. This section resembles the snippet below:

```
<interface type='network'>
    [output truncated]
    <model type='rtl8139' />
</interface>
```

4. Change the type attribute of the model element from '**rtl8139**' to '**virtio**'. This will change the driver from the rtl8139 driver to the e1000 driver.

```
<interface type='network'>
    [output truncated]
    <model type='virtio' />
</interface>
```

5. Save the changes and exit the text editor
6. Restart the guest operating system.

Creating new guests using other network drivers

Alternatively, new guests can be created with a different network driver. This may be required if you are having difficulty installing guests over a network connection. This method requires you to have at least one guest already created (possibly installed from CD or DVD) to use as a template.

1. Create an XML template from an existing guest (in this example, named **Guest1**):

```
# virsh dumpxml Guest1 > /tmp/guest-template.xml
```

2. Copy and edit the XML file and update the unique fields: virtual machine name, UUID, disk image, MAC address, and any other unique parameters. Note that you can delete the UUID and MAC address lines and virsh will generate a UUID and MAC address.

```
# cp /tmp/guest-template.xml /tmp/new-guest.xml
# vi /tmp/new-guest.xml
```

Add the model line in the network interface section:

```
<interface type='network'>
    [output truncated]
    <model type='virtio' />
</interface>
```

3. Create the new virtual machine:

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

21.10. Workaround for creating external snapshots with libvirt

There are two classes of snapshots for QEMU guests. Internal snapshots are contained completely within a qcow2 file, and fully supported by *libvirt*, allowing for creating, deleting, and reverting of snapshots. This is the default setting used by *libvirt* when creating a snapshot, especially when no option is specified. Although this file type takes a bit longer than others in creating the the snapshot, it is required by *libvirt* to use qcow2 disks. Another drawback to this file type is that qcow2 disks are not subject to receive improvements from QEMU.

External snapshots, on the other hand work with any type of original disk image, can be taken with no guest downtime, and are able to receive active improvements from QEMU. In *libvirt*, they are created when using the **--disk-only** or **--memspec** option to **snapshot-create-as** (or when specifying an explicit XML file to **snapshot-create** that does the same). At the moment external snapshots are a one-way operation as *libvirt* can create them but can't do anything further with them. A workaround is described [here](#).

21.11. Missing characters on guest console with Japanese keyboard

On a Red Hat Enterprise Linux 6 host, connecting a Japanese keyboard locally to a machine may result in typed characters such as the underscore (the _ character) not being displayed correctly in guest consoles. This occurs because the required keymap is not set correctly by default.

With Red Hat Enterprise Linux 3 and Red Hat Enterprise Linux 6 guests, there is usually no error message produced when pressing the associated key. However, Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5 guests may display an error similar to the following:

```
atkdb.c: Unknown key pressed (translated set 2, code 0x0 on isa0060/serio0).
atkbd.c: Use 'setkeycodes 00 <keycode>' to make it known.
```

To fix this issue in **virt-manager**, perform the following steps:

- ▶ Open the affected guest in **virt-manager**.
- ▶ Click **View → Details**.
- ▶ Select **Display VNC** in the list.
- ▶ Change **Auto** to **ja** in the **Keypad** pull-down menu.
- ▶ Click the **Apply** button.

Alternatively, to fix this issue using the **virsh edit** command on the target guest:

- ▶ Run **virsh edit <target guest>**
- ▶ Add the following attribute to the **<graphics>** tag: **keymap='ja'**. For example:

```
<graphics type='vnc' port=''-1' autoport='yes' keymap='ja'/>
```

21.12. Known Windows XP guest issues

If you perform device-add quickly followed by device-del using a Windows XP guest, the guest does not eject the device and instead it displays the following error: "The device (device name) cannot be stopped because of an unknown error. Since the device is still being used, do not remove it". It should be noted that newer Windows OS version guests as well as all known Linux guests do not experience this problem. To prevent this issue from happening, wait to delete a device that you just added.

The Virtual Host Metrics Daemon (vhostmd)

vhostmd (the Virtual Host Metrics Daemon) allows virtual machines to see limited information about the host they are running on. This daemon is only supplied with Red Hat Enterprise Linux for SAP.

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

The system administrator chooses which metrics are available for use on a per guest virtual machine basis. In addition, the system administrator may block one or more guest virtual machines from having any access to metric configurations.

Customers who want to use **vhostmd** and **vm-dump-metrics** therefore need subscriptions for "RHEL for SAP Business Applications" to be able to subscribe their RHEL systems running SAP to the "RHEL for SAP" channel on RHN/Customer Portal to install the packages. The following kbase article in the customer portal describes the setup of vhostmd on RHEL:

<https://access.redhat.com/knowledge/solutions/41566>

Additional resources

To learn more about virtualization and Red Hat Enterprise Linux, refer to the following resources.

B.1. Online resources

- ▶ <http://www.libvirt.org/> is the official website for the **libvirt** virtualization API.
- ▶ <http://virt-manager.et.redhat.com/> is the project website for the **Virtual Machine Manager** (virt-manager), the graphical application for managing virtual machines.
- ▶ Open Virtualization Center
<http://www.openvirtualization.com>
- ▶ Red Hat Documentation
<http://www.redhat.com/docs/>
- ▶ Virtualization technologies overview
<http://virt.kernelnewbies.org>
- ▶ Red Hat Emerging Technologies group
<http://et.redhat.com>

B.2. Installed documentation

- ▶ **man virsh** and **/usr/share/doc/libvirt-<version-number>** — Contains sub commands and options for the **virsh** virtual machine management utility as well as comprehensive information about the **libvirt** virtualization library API.
- ▶ **/usr/share/doc/gnome-applet-vm-<version-number>** — Documentation for the GNOME graphical panel applet that monitors and manages locally-running virtual machines.
- ▶ **/usr/share/doc/libvirt-python-<version-number>** — Provides details on the Python bindings for the **libvirt** library. The **libvirt-python** package allows python developers to create programs that interface with the **libvirt** virtualization management library.
- ▶ **/usr/share/doc/python-virtinst-<version-number>** — Provides documentation on the **virt-install** command that helps in starting installations of Fedora and Red Hat Enterprise Linux related distributions inside of virtual machines.
- ▶ **/usr/share/doc/virt-manager-<version-number>** — Provides documentation on the Virtual Machine Manager, which provides a graphical tool for administering virtual machines.

Revision History

Revision 1-323.404	Mon Nov 25 2013	Rüdiger Landmann
Rebuild with Publican 4.0.0		
Revision 1-323	Mon Nov 18 2013	Laura Novich
Fixed BZ1012905 Release Candidate		
Revision 1-322	Mon Nov 11 2013	Laura Novich
Re-Build Beta		
Revision 1-321	Sun Nov 10 2013	Laura Novich
Final build for Beta		
Revision 1-320	Sun Nov 10 2013	Laura Novich
Final build		
Revision 1-319	Sun Nov 10 2013	Laura Novich
Final beta build		
Revision 1-318	Sun Nov 10 2013	Laura Novich
Fixed multiple errors/typos BZ982109		
Revision 1-317	Sun Nov 10 2013	Laura Novich
Fixed whitelist		
Revision 1-316	Sun Nov 10 2013	Laura Novich
Supplied documentation for BZ1015979		
Revision 1-315	Thu Nov 7 2013	Laura Novich
Fixed BZ1012905 again		
Revision 1-314	Thu Nov 7 2013	Laura Novich
Fixed BZ1012905 again		
Revision 1-313	Mon Nov 4 2013	Laura Novich
Fixed BZ1012905 Fixed BZ1012427		
Revision 1-312	Sun Sep 29 2013	Laura Novich
Preparing for Beta		
Revision 1-311	Thu Sep 26 2013	Laura Novich
Fixed BZ969779		
Revision 1-310	Tue Sep 24 2013	Laura Novich
Fixed BZ977262		
Revision 1-309	Tue Sep 24 2013	Laura Novich
Fixed BZ974323		
Revision 1-308	Mon Sep 23 2013	Laura Novich

Fixed note in storage pools chapter

Revision 1-307	Mon Sep 23 2013	Laura Novich
Fixed some whitespace issues in the Domain XML chapter		
Revision 1-306	Mon Sep 23 2013	Laura Novich
Fixed BZ954246		
Revision 1-305	Mon Sep 23 2013	Laura Novich
Fixed URL to Customer Portal		
Revision 1-304	Sun Sep 22 2013	Laura Novich
Fixed BZ841160		
Revision 1-303	Sun Sep 22 2013	Laura Novich
Fixed BZ869585		
Revision 1-302	Sun Sep 22 2013	Laura Novich
Fixed BZ869239		
Revision 1-301	Sun Sep 22 2013	Laura Novich
Fixed BZ852432		
Revision 1-300	Sun Sep 22 2013	Laura Novich
Fixed BZ851047		
Revision 1-299	Sat Sep 21 2013	Laura Novich
BZ997089		
Revision 1-298	Sat Sep 21 2013	Laura Novich
BZ982930		
Revision 1-297	Sat Sep 21 2013	Laura Novich
BZ982930		
Revision 1-296	Sat Sep 21 2013	Laura Novich
ran spellcheck to find typos. Changes to multiple files		
Revision 1-295	Wed Sep 18 2013	Laura Novich
added qemu32 as a cpu type to the whitelist. This fixes BZ1008992		
Revision 1-294	Wed Sep 18 2013	Laura Novich
changed Storage Concepts chapter to reflect non-support of multi-path storage pools. BZ905765		
Revision 1-293	Tue Sep 17 2013	Laura Novich
changed domain XML chapter to add sentence about CPU models. BZ841160		
Revision 1-292	Tue Sep 17 2013	Laura Novich
changed iSCSI storage pool directions and examples and screenshots and made them more consistent to fix BZ972594		
Revision 1-291	Tue Sep 17 2013	Laura Novich
changed iSCSI storage pool directions and examples and screenshots and made them more consistent to fix BZ972594		

Revision 1-290	Mon Sep 16 2013	Laura Novich
added virsh snapshot section as per BZ851047		
Revision 1-289	Sun Sep 15 2013	Laura Novich
added qemu-image new information as per BZ977089		
Revision 1-288	Sun Sep 15 2013	Laura Novich
added port masquerading information as per BZ977241		
Revision 1-287	Thu Sep 12 2013	Laura Novich
removed lines from table 4.1 in sVirt chapter for BZ915131		
Revision 1-286	Thu Sep 12 2013	Laura Novich
added virsh dump command to Troubleshooting chapter BZ969779		
Revision 1-285	Thu Sep 12 2013	Laura Novich
added warning message to Storage Pool Chapter as per BZ974323		
Revision 1-284	Thu Sep 12 2013	Laura Novich
added disk size limit as per 96980		
Revision 1-283	Thu Sep 12 2013	Laura Novich
added disk size limit as per 96980		
Revision 1-282	Wed Sep 11 2013	Laura Novich
added virsh domxml-from-native as per BZ869585		
Revision 1-281	Wed Sep 11 2013	Laura Novich
added cgroup information as specified in BZ958796		
Revision 1-280	Tue Sep 10 2013	Laura Novich
Fixed Storage Volume chapter as per BZ905318		
Revision 1-279	Tue Sep 10 2013	Laura Novich
Fixed Live Migration section as per BZ973844		
Revision 1-278	Tue Sep 10 2013	Laura Novich
Fixed Troubleshooting section as per BZ852432		
Revision 1-277	Tue Sep 10 2013	Laura Novich
typos BZ954246		
Revision 1-276	Tue Sep 10 2013	Laura Novich
removed majority of content for vhostmd as there is a kbase article for it and the package is only used with Red Hat Enterprise Linux for SAP		
Revision 1-275	Tue Sep 10 2013	Laura Novich
multiple edits for multiple chapters BZ982109		
Revision 1-274	Tue Sep 10 2013	Laura Novich
Added virt-tools chapter hot key information		

Revision 1-273	Mon Sep 9 2013	Laura Novich
Added virt-tools chapter		
Revision 1-272	Mon Sep 2 2013	Laura Novich
Added sentence to LVM storage pools procedure		
Revision 1-271	Sun Sep 1 2013	Laura Novich
Added some cross-reference information in the Troubleshooting section to answer BZ 508607		
Revision 1-270	Thu Aug 29 2013	Laura Novich
Changed wording to fix BZ924028 Added section for RHEL guests and freeze thaw hooks - BZ 9282930 Added note for the same function on Windows Guests		
Revision 1-269	Thu Aug 22 2013	Laura Novich
changed file extension as per BZ979833		
Revision 1-268	Wed Aug 21 2013	Laura Novich
added change to Misc. Admin chapter for further development of libvirt-guests service		
Revision 1-267	Wed Aug 21 2013	Laura Novich
added change to Misc. Admin chapter for further development of libvirt-guests service		
Revision 1-266	Wed Aug 21 2013	Scott Radvan
added change to Misc. Admin chapter for further development of libvirt-guests service		
Revision 1-265	Thu Aug 15 2013	Laura Novich
added change to Misc. Admin chapter for further development of libvirt-guests added change as proposal to change graphics in Svirt chapter		
Revision 1-264	Tue Aug 13 2013	Laura Novich
changed KSM chapter to reflect auto NUMA support, added note BZ987795		
Revision 1-263	Mon Aug 12 2013	Laura Novich
changed KSM chapter to reflect auto NUMA support BZ987795 fixed virsh reference to reflect change needed to fix BZ988101 fixed Misc. Administration chapter to reflect change needed in instructions for shutting down guests using libvirt-guests to fix BZ989996		
Revision 1-262	Wed Aug 7 2013	Laura Novich
changed wording globally to meet request in BZ 813619		
Revision 1-261	Sun Jul 28 2013	Laura Novich
changed URL in introduction chapter BZ982464		
Revision 1-260	Tue Jul 16 2013	Laura Novich
BZ980129 - changed ID for intro chapter ID		
Revision 1-259	Tue Jul 16 2013	Laura Novich
changed introduction to include enhanced documentation list BZ980129		
Revision 1-258	Sun Jul 14 2013	Laura Novich
BZ717482 - added information about ksmtuned		

Revision 1-257	Sun Jul 14 2013	Laura Novich
BZ926947 - fixed 2 syntax errors in certtool directions		
Revision 1-256	Wed Feb 20 2013	Laura Novich
Submitting Version 6.4 GA Release Candidate		
Revision 1-254	Wed Feb 20 2013	Laura Novich
modifying example in committing section for BZ902405		
Revision 1-253	Tue Feb 19 2013	Laura Novich
6.4 GA Release Candidate Version		
Revision 1-252	Mon Feb 18 2013	Laura Novich
BZ902405 - fixed organization		
Revision 1-251	Mon Feb 18 2013	Laura Novich
BZ902405 - fixed some typos		
Revision 1-250	Sun Feb 17 2013	Laura Novich
BZ902405 - changed the entire section		
Revision 1-249	Wed Feb 6 2013	Laura Novich
BZ902405 - changed the example again following feedback		
Revision 1-248	Wed Feb 6 2013	Laura Novich
BZ814540 - added MaxStartups parameter to KVM Migration chapter BZ840921 - removed LiveBlock Copy from Virsh chapter 902405 - fixed example for Swap calculation in KVM overcommitting chapter		
Revision 1-247	Thu Jan 24 2013	Laura Novich
corrected spelling inconsistency for hugepages		
Revision 1-246	Thu Jan 24 2013	Laura Novich
fixed hugepages support in chapter 8, BZ903016		
Revision 1-245	Tue Jan 15 2013	Laura Novich
fixed KSM.xml - to reflect change in NUMA		
Revision 1-244	Tue Jan 15 2013	Laura Novich
fixed typos for BZ824328		
Revision 1-243	Thu Dec 20 2012	Laura Novich
fixed typos for BZ514841		
Revision 1-242	Thu Dec 20 2012	Laura Novich
changed core dump directions BZ856563		
Revision 1-241	Wed Dec 19 2012	Laura Novich
fixed typos for BZ856563 abd BZ514841		
Revision 1-240	Tue Dec 18 2012	Laura Novich
Changed coreDump details BZ 856563		
Revision 1-239	Mon Dec 17 2012	Laura Novich

changed storage volume directions for storage volume chapter BZ840827

Revision 1-238	Mon Dec 17 2012	Laura Novich
changed virsh commands for deleting a storage pool in Storage Pool chapter - BZ 514841		
Revision 1-237	Wed Dec 12 2012	Laura Novich
added guest core memory libvirt commands		
Revision 1-236	Sun Dec 2 2012	Laura Novich
fixed typos in Storage Pools chapter		
Revision 1-235	Wed Nov 28 2012	Laura Novich
fixed typos globally		
Revision 1-234	Tue Nov 27 2012	Laura Novich
fixed spelling globally		
Revision 1-233	Sun Nov 25 2012	Laura Novich
fixed whitelist to remove portions of usb-storage as dictated in https://bugzilla.redhat.com/show_bug.cgi?id=878885		
Revision 1-232	Thu Nov 22 2012	Laura Novich
fixed typo in Virtual Networking chapter, bz720309 fixed typo in Storage Pools chapter, bz615141		
Revision 1-231	Wed Nov 21 2012	Laura Novich
added open vSwitch management to the virtual networking chapter. bz853127		
Revision 1-230	Wed Nov 21 2012	Laura Novich
added dynamic vNIC management to the Misc. Administration chapter. BZ853130		
Revision 1-229	Wed Nov 21 2012	Laura Novich
added to KSM chapter that KSM is NUMA aware bz852657		
Revision 1-228	Wed Nov 21 2012	Laura Novich
added usb-redirect section to the Misc. Administration chapter. https://bugzilla.redhat.com/show_bug.cgi?id=795929		
Revision 1-227	Wed Nov 21 2012	Laura Novich
added qemu-ga section to the Misc. Administration chapter		
Revision 1-226	Tue Nov 20 2012	Laura Novich
Changed vPMU section in the Misc. Administration chapter. https://bugzilla.redhat.com/show_bug.cgi?id=840927		
Revision 1-225	Tue Nov 20 2012	Laura Novich
Added vPMU and S3/S4 to Misc. admin tasks chapter. https://bugzilla.redhat.com/show_bug.cgi?id=840816 and https://bugzilla.redhat.com/show_bug.cgi?id=840927		
Revision 1-224	Tue Nov 20 2012	Laura Novich
Fixed timer mode attribute table in the Libvirt Managed Timers section of the Misc. Administration chapter. This fix is against https://bugzilla.redhat.com/show_bug.cgi?id=832419		
Revision 1-223	Mon Nov 19 2012	Laura Novich

fixed troubleshooting chapter

Revision 1-222	Mon Nov 19 2012	Laura Novich
fixed chapter 11		
Revision 1-221	Mon Nov 19 2012	Laura Novich
fixed chapter 10		
Revision 1-220	Mon Nov 19 2012	Laura Novich
fixed chapter 12 and 14 virtio-scsi typos		
Revision 1-219	Mon Nov 19 2012	Laura Novich
fixed chapter 12 and 14 virtio-scsi		
Revision 1-218	Mon Nov 19 2012	Laura Novich
fixed chapter 17 - added macvtap/sr-iov and VEPA tech		
Revision 1-217	Mon Nov 19 2012	Laura Novich
fixed chapter 11 - added introduction		
Revision 1-216	Sun Nov 18 2012	Laura Novich
fixed chapter 16		
Revision 1-215	Sun Nov 18 2012	Laura Novich
fixed chapter 16		
Revision 1-214	Sun Nov 18 2012	Laura Novich
fixed chapters 12 and 14		
Revision 1-213	Sun Nov 18 2012	Laura Novich
fixed ch 11		
Revision 1-212	Sun Nov 18 2012	Laura Novich
fixed chapter 4 and chapter 16		
Revision 1-211	Thu Nov 15 2012	Laura Novich
fixed whitelist		
Revision 1-210	Thu Nov 15 2012	Laura Novich
fixed KSM deactivation		
Revision 1-209	Mon Nov 12 2012	Laura Novich
fixed chapter 12		
Revision 1-208	Mon Nov 12 2012	Laura Novich
fixed chapter 17		
Revision 1-207	Mon Nov 12 2012	Laura Novich
fixed chapter 12		
Revision 1-206	Mon Nov 12 2012	Laura Novich
fixed chapter 4		

Revision 1-205	Mon Nov 12 2012	Laura Novich
fixed chapter 5 fixed chapter 12		
Revision 1-204	Thu Nov 8 2012	Laura Novich
fixed whitelist		
Revision 1-203	Tue Oct 30 2012	Laura Novich
changed whitelist to add CPU new info		
Revision 1-202	Sun Oct 28 2012	Laura Novich
added blockresize to chapter 14		
Revision 1-201	Sun Oct 28 2012	Laura Novich
fixed typo in chapter 14		
Revision 1-200	Tue Oct 23 2012	Laura Novich
fixed virsh Reference chapter - live block copy		
Revision 1-199	Tue Oct 23 2012	Laura Novich
fixed typos in chapters 4 and 10		
Revision 1-198	Tue Oct 23 2012	Laura Novich
fixed virsh Reference chapter 14		
Revision 1-197	Tue Oct 23 2012	Laura Novich
fixed chapter 4 - KVM Migration		
Revision 1-196	Mon Oct 22 2012	Laura Novich
fixed ch 12		
Revision 1-195	Mon Oct 22 2012	Laura Novich
fixed whitelist to include -machine dump-guest-core		
Revision 1-194	Mon Oct 22 2012	Laura Novich
fixed virsh reference, added memory tuning and allocation commands		
Revision 1-193	Mon Oct 22 2012	Laura Novich
added more cross-references to the Virtual Networking chapter		
Revision 1-192	Mon Oct 22 2012	Laura Novich
added network filtering and DHCP snooping libvirt (virsh) commands		
Revision 1-191	Thu Oct 18 2012	Laura Novich
added block copy, block pull and block commit to the virsh reference chapter		
Revision 1-190	Wed Oct 17 2012	Laura Novich
fixed BZ748422 - changed TLS and SSL Management sections		
Revision 1-189	Tue Oct 16 2012	Laura Novich
fixed storage volume chapter		
Revision 1-188	Sun Oct 14 2012	Laura Novich
fixed whitelist		

Revision 1-187	Sun Oct 14 2012	Laura Novich
fixed storage pools and volumes		
Revision 1-186	Sun Oct 14 2012	Laura Novich
fixed live migration section added USB support		
Revision 1-185	Thu Oct 11 2012	Laura Novich
fixed live migration section		
Revision 1-184	Thu Oct 11 2012	Laura Novich
fixed Troubleshooting section		
Revision 1-183	Wed Oct 10 2012	Laura Novich
fixed BZ831453		
Revision 1-182	Wed Oct 10 2012	Laura Novich
fixed BZ831453		
Revision 1-181	Wed Oct 10 2012	Laura Novich
fixed virsh ref		
Revision 1-180	Wed Oct 10 2012	Laura Novich
re-run		
Revision 1-179	Wed Oct 10 2012	Laura Novich
fixed troubleshootig section		
Revision 1-178	Wed Oct 10 2012	Laura Novich
fixed chapters 4 and 17		
Revision 1-177	Tue Oct 9 2012	Laura Novich
BZ831453		
Revision 1-176	Tue Oct 9 2012	Laura Novich
BZ753026		
Revision 1-175	Tue Oct 9 2012	Laura Novich
BZ835572		
Revision 1-174	Tue Oct 9 2012	Laura Novich
fixed-screenshots		
Revision 1-173	Tue Oct 9 2012	Laura Novich
bz753049		
Revision 1-172	Tue Oct 9 2012	Laura Novich
ch4 fixes		
Revision 1-171	Tue Oct 9 2012	Laura Novich
ch4 fixes		
Revision 1-170	Tue Oct 9 2012	Laura Novich

BZ753049

Revision 1-169 BZ829125	Thu Oct 4 2012	Laura Novich
Revision 1-168 whitelist edited	Thu Oct 4 2012	Laura Novich
Revision 1-167 changed NFS example in ch 4	Thu Oct 4 2012	Laura Novich
Revision 1-166 BZ753026 BZ753049 BZ826360 BZ824328 BZ830659 BZ831897	Thu Oct 4 2012	Laura Novich
Revision 1-165 edited some screenshots	Wed Oct 3 2012	Laura Novich
Revision 1-164 fixed ch 12	Wed Oct 3 2012	Laura Novich
Revision 1-163 fixed chapter 9	Wed Oct 3 2012	Laura Novich
Revision 1-162 BZ829146	Wed Oct 3 2012	Laura Novich
Revision 1-161 BZ835572	Tue Oct 2 2012	Laura Novich
Revision 1-160 changed ch 12	Tue Oct 2 2012	Laura Novich
Revision 1-159 BZ814135	Tue Oct 2 2012	Laura Novich
Revision 1-158 BZ829131	Tue Oct 2 2012	Laura Novich
Revision 1-157 fixed BZ832132	Tue Oct 2 2012	Laura Novich
Revision 1-156 changed --persistent to --config across the guide	Tue Oct 2 2012	Laura Novich
Revision 1-155 fixed BZ829146	Mon Oct 1 2012	Laura Novich
Revision 1-154 deleted chapter acc to BZ829512	Mon Oct 1 2012	Laura Novich

Revision 1-153	Mon Oct 1 2012	Laura Novich
fixed Storage pools with virsh BZ829131		
Revision 1-152	Fri Sep 28 2012	Laura Novich
fixed whitelist		
Revision 1-151	Fri Sep 28 2012	Laura Novich
storage formats		
Revision 1-150	Thu Sep 27 2012	Laura Novich
new screens		
Revision 1-149	Thu Sep 27 2012	Laura Novich
BZ829519		
Revision 1-148	Thu Sep 27 2012	Laura Novich
BZ829519		
Revision 1-147	Thu Sep 27 2012	Laura Novich
BZ829526		
Revision 1-146	Thu Sep 27 2012	Laura Novich
BZ854592 - fixed syntax		
Revision 1-145	Thu Sep 27 2012	Laura Novich
fixed storage pools BZ831952		
Revision 1-144	Tue Sep 25 2012	Laura Novich
fixed ch 9 BZ822444		
Revision 1-143	Mon Sep 24 2012	Laura Novich
fixed storage pools added storage pools deletion instructions fixed BZ831952		
Revision 1-142	Mon Sep 24 2012	Laura Novich
fixed whitelist BZ814100		
Revision 1-141	Mon Sep 24 2012	Laura Novich
fixed whitelist BZ831602		
Revision 1-140	Mon Sep 24 2012	Laura Novich
fixed whitelist BZ831602		
Revision 1-139	Mon Sep 24 2012	Laura Novich
fixed whitelist BZ814180,814135,814100,814106		
Revision 1-138	Mon Sep 24 2012	Laura Novich
fixed three bugs:BZ854592,BZ832410,BZ850972		
Revision 1-137	Mon Sep 24 2012	Laura Novich
fixed three bugs:BZ854592,BZ832410,BZ850972		

Revision 1-136	Thurs Sept 20 2012	Laura Novich
Added all fixes for chapter 12 and ch 15		
Revision 1-135	Tues Sept 04 2012	Laura Novich
Added all fixes for chapter 17 Bug 829527		
Revision 1-134	Mon Aug 27 2012	Scott Radvan
Add fixes from 850792		
Revision 1-133	Sun Aug 26 2012	Scott Radvan
bump to avoid SVN conflict.		
Revision 1-132	Sun Aug 26 2012	Scott Radvan
Modified overcommit example: BZ#804397		
Revision 1-131	Mon Jun 17 2012	Laura Novich
Version for 6.3 GA release		
Revision 0.1-129	Sun Jun 17 2012	Laura Novich
Test version for 6.3 GA release		
Revision 0.1-128	Wed Jun 13 2012	Laura Novich
Test version for 6.3 GA release		
Revision 0.1-125	Wed Jun 13 2012	Laura Novich
fixes to chapters chs 17 and 18(BZ#831554, 829527)		
Revision 0.1-124	Wed Jun 13 2012	Laura Novich
fixes to chapters 14 and 15(BZ#829513)		
Revision 0.1-123	Wed Jun 13 2012	Laura Novich
Removed what was chapters 19 and 20, fixed chs 15-17 (BZ#829879, 829513)		
Revision 0.1-122	Tues Jun 12 2012	Laura Novich
Made many screen shot changes to chapter 11 (ref- BZ#829134).		
Revision 0.1-120	Tues Jun 12 2012	Laura Novich
Made many screen shot changes to chapter 5 (ref- BZ#829523).		
Revision 0.1-117	Tues Jun 12 2012	Laura Novich
Made many screen shot changes to chapter 11 (ref- BZ#82918).		
Revision 0.1-111	Sun Jun 10 2012	Laura Novich
Made many changes to fix multiple issues (BZ# 829124, 829525, 829524, 829521).		
Revision 0.1-109	Tue Mar 27 2012	Laura Novich
Made many changes to fix issue (BZ# 514841).		
Revision 0.1-106	Thu Mar 19 2012	Laura Novich
Added new changes required from QE (BZ# 794798).		
Revision 0.1-101	Thu Mar 19 2012	Laura Novich
Added new information 12.3.2 (BZ# 794798).		

Revision 0.1-100	Thu Mar 19 2012	Laura Novich
Added new information 12.3.1 (BZ# 794798).		
Revision 0.1-99	Thu Mar 15 2012	Laura Novich
Added new Section to Chapter 4 (BZ# 788002).		
Revision 0.1-98	Mon Feb 27 2012	Scott Radvan
Add the `shut off' state for the `virsh list --all' command (BZ#752349).		
Revision 0.1-97	Mon Feb 27 2012	Scott Radvan
Add procedure for dealing with Japanese keymap issues (BZ#729148).		
Revision 0.1-96	Mon Feb 27 2012	Scott Radvan
"none" is no longer a valid tickpolicy (BZ#742082).		
Revision 0.1-95	Mon Feb 27 2012	Scott Radvan
Revise KSM service details (BZ#769478).		
Revision 0.1-94	Wed Feb 22 2012	Scott Radvan
Remove manually-added 'title' tags from admonitions throughout guide.		
Revision 0.1-93	Wed Feb 22 2012	Scott Radvan
First publish with new tool.		
Revision 0.1-88	Mon Jan 09 2012	Scott Radvan
Resolves BZ#769122.		
Revision 0.1-87	Mon Nov 28 2011	Scott Radvan
Resolves BZ#753068.		
Revision 0.1-86	Mon Nov 28 2011	Scott Radvan
Resolves BZ#755487.		
Revision 0.1-85	Mon Nov 21 2011	Scott Radvan
Resolves BZ#754910.		
Revision 0.1-84	Tue Nov 15 2011	Scott Radvan
Resolves BZ#753041.		
Revision 0.1-83	Tue Nov 15 2011	Scott Radvan
Further typographical errors.		
Revision 0.1-82	Tue Nov 15 2011	Scott Radvan
Minor typographical errors.		
Revision 0.1-81	Tue Nov 15 2011	Scott Radvan
Resolves BZ#753018.		
Revision 0.1-80	Tue Nov 15 2011	Scott Radvan
Resolves BZ#753711.		
Revision 0.1-79	Thu Nov 03 2011	Scott Radvan
Resolves BZ#746336.		

Revision 0.1-78	Thu Nov 03 2011	Scott Radvan
Resolves BZ#74490.		
Revision 0.1-76	Tue Oct 25 2011	Scott Radvan
Resolves BZ#74490.		
Revision 0.1-75	Tue Oct 25 2011	Scott Radvan
Resolves BZ#748310.		
Revision 0.1-73	Tue Oct 18 2011	Scott Radvan
Add script for migrating non-running guests.		
Revision 0.1-69	Mon Sep 19 2011	Scott Radvan
QEMU/KVM whitelist chapter updated with minor fixes.		
Revision 0.1-68	Fri Sep 16 2011	Scott Radvan
Chapter 15 restructure, add Guest CPU model configuration details (BZ#737109)		
Revision 0.1-67	Thu Sep 15 2011	Scott Radvan
Add qemu-kvm whitelist		
Revision 0.1-66	Tue Sep 13 2011	Scott Radvan
BZ#735224		
Revision 0.1-65	Mon Sep 12 2011	Scott Radvan
BZ#735207 BZ#735209 BZ#735211 BZ#735219 BZ#735220 BZ#735223		
Revision 0.1-64	Tue Sep 06 2011	Scott Radvan
BZ#734998		
Revision 0.1-03	Thu Jun 2 2011	Scott Radvan
Add draft watermark		
Revision 0.1-02	Thu Jun 2 2011	Scott Radvan
Import majority of text, add alpha channels to PNGs, bump for publish.		

Index

F

feedback

- contact information for this manual, [We Need Feedback!](#)

H

help

- getting help, [Do You Need Help?](#)

V**virtualization**

- additional resources
 - useful websites, [Online resources](#)

Virtualization

- additional resources
 - installed documentation, [Installed documentation](#)