Abstract

This guide included information on how to configure a Fedora machine as a virtualization host, and install and configure virtual machines under Fedora virtualization.

Note: This document is under development, is subject to substantial change, and is provided only as a preview. The included information and instructions should not be considered complete, and should be used with caution.

Preface

1. Document Conventions
   1.1. Typographic Conventions
   1.2. Pull-quote Conventions
   1.3. Notes and Warnings

2. We Need Feedback!

I. Deployment

1. Introduction
   1.1. What is in this guide?
   1.2. Virtualization Documentation Suite

2. System requirements

3. KVM guest virtual machine compatibility
   3.1. Fedora 6 support limits
   3.2. Supported CPU Models
      3.2.1. Guest CPU models

4. Virtualization restrictions
   4.1. KVM restrictions
   4.2. Application restrictions
   4.3. Other restrictions

5. Installing the virtualization packages
   5.1. Configuring a Virtualization Host installation
   5.2. Installing virtualization packages on an existing Fedora system

6. Guest virtual machine installation overview
   6.1. Guest virtual machine prerequisites and considerations
   6.2. Creating guests with virt-install
   6.3. Creating guests with virt-manager
   6.4. Installing guest virtual machines with PXE

7. Installing a Red Hat Enterprise Linux 6 guest virtual machine on a Red Hat Enterprise Linux 6 host
   7.1. Creating a Red Hat Enterprise Linux 6 guest with local installation media
   7.2. Creating a Red Hat Enterprise Linux 6 guest with a network installation tree
   7.3. Creating a Red Hat Enterprise Linux 6 guest with PXE

8. Virtualizing Fedora on Other Platforms
   8.1. On VMWare
   8.2. On Hyper-V

9. Installing a fully-virtualized Windows guest
   9.1. Using virt-install to create a guest

10. KVM Para-virtualized Drivers
    10.1. Installing the KVM Windows para-virtualized drivers
    10.2. Installing the drivers on an installed Windows guest virtual machine
    10.3. Installing drivers during the Windows installation
    10.4. Using the para-virtualized drivers with Red Hat Enterprise Linux 3.9 guests
    10.5. Using KVM para-virtualized drivers for existing devices
    10.6. Using KVM para-virtualized drivers for new devices

11. Network configuration
    11.1. Network Address Translation (NAT) with libvirt
    11.2. Disabling vhost-net
    11.3. Bridged networking with libvirt

12. PCI device configuration
12. PCI device configuration
12.1. Assigning a PCI device with virsh
12.2. Assigning a PCI device with virt-manager
12.3. PCI device assignment with virt-install
12.4. Detaching an assigned PCI device

13. SR-IOV
13.1. Introduction
13.2. Using SR-IOV
13.3. Troubleshooting SR-IOV

14. KVM guest timing management

15. Network booting with libvirt
15.1. Preparing the boot server
15.1.1. Setting up a PXE boot server on a private libvirt network
15.2. Booting a guest using PXE
15.2.1. Using bridged networking
15.2.2. Using a private libvirt network

16. QEMU Guest Agent
16.1. Set Up Communication between Guest Agent and Host

17. Server best practices

18. Security for virtualization
18.1. Storage security issues
18.2. SELinux and virtualization
18.3. SELinux
18.4. Virtualization firewall information

19. sVirt
19.1. Security and Virtualization
19.2. sVirt labeling

20. KVM live migration
20.1. Live migration requirements
20.2. Live migration and Fedora version compatibility
20.3. Shared storage example: NFS for a simple migration
20.4. Live KVM migration with virsh
20.4.1. Additional tips for migration with virsh
20.4.2. Additional options for the virsh migrate command
20.5. Migrating with virt-manager

21. Remote management of guests
21.1. Remote management with SSH
21.2. Remote management over TLS and SSL
21.3. Transport modes

22. Overcommitting with KVM

23. KSM

24. Advanced virtualization administration
24.1. Control Groups (cgroups)
24.2. Hugepage support

25. Miscellaneous administration tasks
25.1. Automatically starting guests
25.2. Guest memory allocation
25.3. Using qemu-img
25.4. Verifying virtualization extensions
25.5. Setting KVM processor affinities
25.6. Generating a new unique MAC address
25.7. Improving guest response time
25.8. Disable SMART disk monitoring for guests
25.9. Configuring a VNC Server
25.10. Gracefully shutting down guests
25.11. Virtual machine timer management with libvirt
25.12. Using PMU to monitor guest performance
25.13. Guest virtual machine power management
25.14. QEMU Guest Agent Protocol
25.14.1. guest-sync
25.14.2. guest-sync-delimited
25.15. Setting a limit on device redirection
25.16. Dynamically changing a host or a network bridge that is attached to a virtual NIC

26. Storage concepts
   26.1. Storage pools
   26.2. Volumes

27. Storage pools
   27.1. Creating storage pools
      27.1.1. Disk-based storage pools
      27.1.2. Partition-based storage pools
      27.1.3. Directory-based storage pools
      27.1.4. LVM-based storage pools
      27.1.5. iSCSI-based storage pools
      27.1.6. NFS-based storage pools

28. Volumes
   28.1. Creating volumes
   28.2. Cloning volumes
   28.3. Adding storage devices to guests
      28.3.1. Adding file based storage to a guest
      28.3.2. Adding hard drives and other block devices to a guest
      28.3.3. Managing storage controllers in a guest
   28.4. Deleting and removing volumes

29. The Virtual Host Metrics Daemon (vhostmd)
   29.1. Installing vhostmd on the host
   29.2. Configuration of vhostmd
   29.3. Starting and stopping the daemon
   29.4. Verifying that vhostmd is working from the host
   29.5. Configuring guests to see the metrics
   29.6. Using vm-dump-metrics in Fedora guests to verify operation

III. Appendicies
A. Troubleshooting
   A.1. Debugging and troubleshooting tools
   A.2. kvm_stat
   A.3. Troubleshooting with serial consoles
   A.4. Virtualization log files
   A.5. Loop device errors
   A.6. Live Migration Errors
   A.7. Enabling Intel VT-x and AMD-V virtualization hardware extensions in BIOS
   A.8. KVM networking performance
   A.9. Missing characters on guest console with Japanese keyboard
   A.10. Known Windows XP guest issues
B. Common libvirt errors and troubleshooting
   B.1. libvirtd failed to start
   B.2. The URI failed to connect to the hypervisor
      B.2.1. Cannot read CA certificate
      B.2.2. Failed to connect socket ...: Permission denied
      B.2.3. Other connectivity errors
   B.3. The guest virtual machine cannot be started: internal error
      guest CPU is not compatible with host CPU
   B.4. Guest starting fails with error: monitor socket did not show up
   B.5. Internal error cannot find character device (null)
   B.6. Guest virtual machine booting stalls with error: No boot device
   B.7. Virtual network default has not been started
   B.8. PXE boot (or DHCP) on guest failed
   B.9. Guest can reach outside network, but cannot reach host when using macvtap interface
   B.10. Could not add rule to fixup DHCP response checksums on network 'default'
   B.11. Unable to add bridge br0 port vnet0: No such device
   B.12. Guest is unable to start with error: warning: could not open /dev/net/tun
   B.13. Migration fails with Error: unable to resolve address
   B.14. Migration fails with Unable to allow access for disk path: No such file or directory
   B.15. No guest virtual machines are present when libvirtd is started
   B.16. Unable to connect to server at host:16509: Connection refused ... error: failed to connect to the hypervisor
   B.17. Common XML errors
B.17.1. Editing domain definition
B.17.2. XML syntax errors
B.17.3. Logic and configuration errors

C. NetKVM Driver Parameters
   C.1. Configurable parameters for NetKVM

D. qemu-kvm Whitelist
   D.1. Introduction
   D.2. Basic options
   D.3. Disk options
   D.4. Display options
   D.5. Network options
   D.6. Device options
   D.7. Linux/Multiboot boot
   D.8. Expert options
   D.9. Help and information options
   D.10. Miscellaneous options

E. Managing guests with virsh
   E.1. virsh command quick reference
   E.2. Attaching and updating a device with virsh
   E.3. Connecting to the hypervisor
   E.4. Creating a virtual machine XML dump (configuration file)
      E.4.1. Adding multifunction PCI devices to KVM guests
   E.5. Suspending, resuming, saving and restoring a guest
   E.6. Shutting down, rebooting and force-shutdown of a guest
   E.7. Retrieving guest information
   E.8. Retrieving node information
   E.9. Storage pool information
   E.10. Displaying per-guest information
   E.11. Managing virtual networks
   E.12. Migrating guests with virsh
   E.13. Disk image management with live block copy
      E.13.1. Using blockcommit to shorten a backing chain
      E.13.2. Using blockpull to shorten a backing chain
      E.13.3. Using blockresize to change the size of a domain path
   E.14. Guest CPU model configuration
      E.14.1. Introduction
      E.14.2. Learning about the host CPU model
      E.14.3. Determining a compatible CPU model to suit a pool of hosts
      E.14.4. Configuring the guest CPU model

F. Managing guests with the Virtual Machine Manager (virt-manager)
   F.1. Starting virt-manager
   F.2. The Virtual Machine Manager main window
   F.3. The virtual hardware details window
   F.4. Virtual Machine graphical console
   F.5. Adding a remote connection
   F.6. Displaying guest details
   F.7. Performance monitoring
   F.8. Displaying CPU usage for guests
   F.9. Displaying CPU usage for hosts
   F.10. Displaying Disk I/O
   F.11. Displaying Network I/O

G. Guest disk access with offline tools
   G.1. Introduction
   G.2. Terminology
   G.3. Installation
   G.4. The guestfish shell
      G.4.1. Viewing file systems with guestfish
      G.4.2. Modifying files with guestfish
      G.4.3. Other actions with guestfish
      G.4.4. Shell scripting with guestfish
      G.4.5. Augeas and libguestfs scripting
   G.5. Other commands
   G.6. virt-rescue: The rescue shell
      G.6.1. Introduction
      G.6.2. Running virt-rescue
   G.7. virt-df: Monitoring disk usage
      G.7.1. Introduction
      G.7.2. Running virt-df
   G.8. virt-resize: resizing guests offline
J.16.1. Hard drives, floppy disks, CDROMs
J.16.2. Filesystems
J.16.3. Device addresses
J.16.4. Controllers
J.16.5. Device leases
J.16.6. Host device assignment
J.16.7. Redirected devices
J.16.8. Smartcard devices
J.16.9. Network interfaces
J.16.10. Input devices
J.16.11. Hub devices
J.16.12. Graphical framebuffers
J.16.13. Video devices
J.16.15. Guest interfaces
J.16.16. Channel
J.16.17. Host interface
J.16.18. Sound devices
J.16.19. Watchdog device
J.17. Memory balloon device
J.18. Random number generator device
J.19. TPM devices
J.20. Security label
J.21. Example domain XML configuration

K. Revision History
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 includes the Liberation Fonts set by default.

1.1. Typographic Conventions

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

Mono-spaced Bold

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

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

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context. Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

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

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

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

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

Proportional Bold

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

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

```text
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:

```java
package org.jboss.book.jca.ex1;
import javax.naming.InitialContext;
public class ExClient {
    public static void main(String args[]) throws Exception {
        InitialContext iniCtx = new InitialContext();
        Object ref = iniCtx.lookup("EchoBean");
        EchoHome home = (EchoHome) ref;
        Echo echo = home.create();
        System.out.println("Created Echo");
        System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
    }
}
```

### 1.3. Notes and Warnings

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

![Note](image)

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](image)

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](image)

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

### 2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a
Part I. Deployment

Table of Contents

1. Introduction
   1.1. What is in this guide?
   1.2. Virtualization Documentation Suite

2. System requirements

3. KVM guest virtual machine compatibility
   3.1. Fedora 6 support limits
   3.2. Supported CPU Models
      3.2.1. Guest CPU models

4. Virtualization restrictions
   4.1. KVM restrictions
   4.2. Application restrictions
   4.3. Other restrictions

5. Installing the virtualization packages
   5.1. Configuring a Virtualization Host installation
   5.2. Installing virtualization packages on an existing Fedora system

6. Guest virtual machine installation overview
   6.1. Guest virtual machine prerequisites and considerations
   6.2. Creating guests with virt-install
   6.3. Creating guests with virt-manager
   6.4. Installing guest virtual machines with PXE

7. Installing a Red Hat Enterprise Linux 6 guest virtual machine on a Red Hat Enterprise Linux 6 host
   7.1. Creating a Red Hat Enterprise Linux 6 guest with local installation media
   7.2. Creating a Red Hat Enterprise Linux 6 guest with a network installation tree
   7.3. Creating a Red Hat Enterprise Linux 6 guest with PXE

8. Virtualizing Fedora on Other Platforms
   8.1. On VMWare
   8.2. On Hyper-V

9. Installing a fully-virtualized Windows guest
   9.1. Using virt-install to create a guest

10. KVM Para-virtualized Drivers
    10.1. Installing the KVM Windows para-virtualized drivers
    10.2. Installing the drivers on an installed Windows guest virtual machine
    10.3. Installing drivers during the Windows installation
    10.4. Using the para-virtualized drivers with Red Hat Enterprise Linux 3.9 guests
    10.5. Using KVM para-virtualized drivers for existing devices
    10.6. Using KVM para-virtualized drivers for new devices

11. Network configuration
    11.1. Network Address Translation (NAT) with libvirt
    11.2. Disabling vhost-net
    11.3. Bridged networking with libvirt

12. PCI device configuration
    12.1. Assigning a PCI device with virsh
    12.2. Assigning a PCI device with virt-manager
    12.3. PCI device assignment with virt-install
    12.4. Detaching an assigned PCI device
13. SR-IOV
   13.1. Introduction
   13.2. Using SR-IOV
   13.3. Troubleshooting SR-IOV

14. KVM guest timing management

15. Network booting with libvirt
   15.1. Preparing the boot server
      15.1.1. Setting up a PXE boot server on a private libvirt network
   15.2. Booting a guest using PXE
      15.2.1. Using bridged networking
      15.2.2. Using a private libvirt network

16. QEMU Guest Agent
   16.1. Set Up Communication between Guest Agent and Host
Chapter 1. Introduction

1.1. What is in this guide?

1.2. Virtualization Documentation Suite

1.1. What is in this guide?

The Virtualization Deployment and Administration Guide, introduced in Fedora 19, resulted from the merger of the Virtualization Host Installation and Guest Configuration Guide and the Virtualization Administration Guide. This new guide provides complete information on both deploying a virtual setup on a Fedora virtualization host as well as how to administer and maintain the system. As such, this guide has 2 main parts: Deployment and Administration. The appendix sections contain references and troubleshooting.

The initial chapters in this guide outline the prerequisites to enable a Fedora host machine to deploy virtualization. System requirements, compatible hardware, support and product restrictions are covered in detail. The first part of this guide (Deployment) covers basic host configuration, including mandatory and optional virtualization packages, which are covered in Chapter 5, Installing the virtualization packages. Later chapters in this part include guest virtual machine installation, which is covered in detail starting from Chapter 6, Guest virtual machine installation overview, with procedures for installing fully virtualized Fedora guests and Windows para-virtualized guests using virt-manager and virsh. The part concludes with More detailed information on networking, PCI device configuration, SR-IOV, KVM guest timing management, and troubleshooting help for libvirt and SR-IOV is included later in the guide.

The second part of this guide (Administration) covers more advanced configuration tactics, creating various storage pools and volumes, manipulating and finetuning memory and other resources, as well as administration tasks that can be performed using virsh and virt-manager to administer both hosts and guests.

The last part of this guide (Appendix) contains troubleshooting information with steps that should be done before getting technical support (if possible), detailed reference guides for using QEMU-KVM flags, describing the Domain XML, and a complete CLI description of the commands used in Virsh and the screens used in Virt-manager. Additional tools that may be useful are also described in this part as well.

1.2. Virtualization Documentation Suite

Fedora offers a wealth of documentation solutions across its various virtualization products. Coverage of Fedora and its inbuilt virtualization products include:

- Fedora — 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.
- Fedora — Virtualization Deployment and Administration Guide covers the installation, configuration, and maintenance of virtualization hosts and virtual machines.
- Fedora — Virtualization Security Guide: This guide provides an overview of virtualization security technologies provided by Red Hat. Also included are recommendations for securing hosts, guests, and shared infrastructure and resources in virtualized environments.
- Fedora — 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.
- Fedora — V2V Guide describes importing virtual machines from KVM, Xen and VMware ESX/ESXi hypervisors to Red Hat Enterprise Virtualization and KVM managed by libvirt.

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

- oVirt — 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 hosts.
- oVirt — Command Line Shell Guide contains information for installing and using the Red Hat Enterprise Virtualization Manager command line shell.
- oVirt — 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.
- oVirt — Evaluation Guide enables prospective customers to evaluate the features of Red Hat Enterprise Virtualization. Use this guide if you have an evaluation license.
- oVirt — Installation Guide describes the installation prerequisites and procedures. Read this if you need to install Red Hat Enterprise...
Virtualization. The installation of hosts, 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.

- oVirt — Power User Portal Guide describes how power users can create and manage virtual machines from the Red Hat Enterprise Virtualization User Portal.
- oVirt — Quick Start Guide provides quick and simple instructions for first time users to set up a basic Red Hat Enterprise Virtualization environment.
- oVirt — Technical Notes describe the changes made between the current release and the previous one.
- 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.
Chapter 2. System requirements

This chapter lists system requirements for successfully running virtual machines, referred to as VMs on Fedora. Virtualization is available for Fedora on the Intel 64 and AMD64 architecture.

The KVM hypervisor is provided with Fedora 16.

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

Minimum system requirements

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

Recommended system requirements

- One processor core or hyper-thread for the maximum number of virtualized CPUs in a guest virtual machine and one for the host.
- 2 GB of RAM plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for each virtual machine.

Most guest operating systems will require at least 6GB of disk space, but the additional storage space required for each guest depends on its image format.

For guest virtual machines using raw images, the guest's total required space (total for raw format) is equal to or greater than the sum of the space required by the guest's raw image files (images), the 6GB space required by the host operating system (host), and the swap space that guest will require (swap).

Equation 2.1. Calculating required space for guest virtual machines using raw images

\[
\text{total for raw format} = \text{images} + \text{host} + \text{swap}
\]

For qcow images, you must also calculate the expected maximum storage requirements of the guest (total for qcow format), as qcow and qcow2 images grow as required. To allow for this expansion, first multiply the expected maximum storage requirements of the guest (expected maximum guest storage) by 1.01, and add to this the space required by the host (host), and the necessary swap space (swap).

Equation 2.2. Calculating required space for guest virtual machines using qcow images

\[
\text{total for qcow format} = (\text{expected maximum guest storage} \times 1.01) + \text{host} + \text{swap}
\]

For more information on Guest virtual machine requirements refer to Chapter 22, Overcommitting with KVM.

Calculating swap space

Using swap space can provide additional memory beyond the available physical memory. The swap partition is used for swapping underused memory to the hard drive to speed up memory performance. The default size of the swap partition is calculated from the physical RAM of the host.

KVM requirements

The KVM hypervisor requires:

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

Storage support

The guest virtual machine storage methods are:

- files on local storage,
- physical disk partitions,
- locally connected physical LUNs,
- LVM partitions,
- NFS shared file systems,
- iSCSI,
- GFS2 clustered file systems,
- Fibre Channel-based LUNs, and
- Fibre Channel over Ethernet (FCoE).
Chapter 3. KVM guest virtual machine compatibility

3.1. Fedora 6 support limits

3.2. Supported CPU Models

3.2.1. Guest CPU models

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

3.1. Fedora 6 support limits

Fedora 19 servers have certain support limits.

The following URLs explain the processor and memory amount limitations for Fedora 19:

- For host systems:
  

- For hypervisors:
  

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

[http://www.redhat.com/resourcelibrary/articles/enterprise-linux-virtualization-support](http://www.redhat.com/resourcelibrary/articles/enterprise-linux-virtualization-support)

3.2. Supported CPU Models

Fedora 19 supports the use of the following QEMU CPU model definitions:

- **Opteron_64**: AMD Opteron 62xx (Gen 4 Class Opteron)
- **Opteron_63**: AMD Opteron 23xx (Gen 3 Class Opteron)
- **Opteron_62**: AMD Opteron 22xx (Gen 2 Class Opteron)
- **Opteron_61**: AMD Opteron 240 (Gen 1 Class Opteron)
- **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)
- **Westmere**: Westmere E56xx/L56xx/X56xx (Nehalem-C)

3.2.1. Guest CPU models

Historically, CPU model definitions were hard-coded in qemu. This method of defining CPU models was inflexible, and made it difficult to create virtual CPUs with feature sets that matched existing physical CPUs. Typically, users modified a basic CPU model definition with feature flags in order to provide the CPU characteristics required by a virtual machine. Unless these feature sets were carefully controlled, safe migration — which requires feature sets between current and prospective hosts to match — was difficult to support.

qemu-kvm has now replaced most hard-wired definitions with configuration file based CPU model definitions. Definitions for a number of current processor models are now included by default, allowing users to specify features more accurately and migrate more safely.

A list of supported guest CPU models can be viewed with the
The four CPUID fields, `feature_edx`, `feature_ecx`, `extfeature_edx` and `extfeature_ecx`, accept named flag values from the corresponding feature sets listed by the `/usr/libexec/qemu-kvm -cpu ?cpuid` command, as shown:

```bash
# /usr/libexec/qemu-kvm -cpu Nehalem,check
warning: host cpuid 0000_0001 lacks requested flag 'sse4.2|sse4_2' [0x00100000]
warning: host cpuid 0000_0001 lacks requested flag 'popcnt' [0x00800000]
```

If a defined feature is not available, those features will fail silently by default.
Chapter 4. Virtualization restrictions

4.1. KVM restrictions
4.2. Application restrictions
4.3. Other restrictions

This chapter covers additional support and product restrictions of the virtualization packages in Fedora 19.

4.1. KVM restrictions

The following restrictions apply to the KVM hypervisor:

Maximum vCPUs per guest
Guest virtual machines support up to a maximum of 160 virtual CPUs in Fedora 19.

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

Memory overcommit
KVM supports memory overcommit and can store the memory of guest virtual machines in swap. A virtual machine will run slower if it is swapped frequently. Fedora Knowledgebase has an article on safely and efficiently determining the size of the swap partition, available here: https://access.redhat.com/knowledge/solutions/15244. When KSM is used for memory overcommitting, make sure that the swap size follows the recommendations described in this article.

Important
When device assignment is in use, all virtual machine memory must be statically pre-allocated to enable DMA with the assigned device. Memory overcommit is therefore not supported with device assignment.

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

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

Refer to the Fedora Virtualization Administration Guide for tips and recommendations on overcommitting CPUs.

Virtualized SCSI devices
SCSI emulation is not supported with KVM in Fedora.

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

Para-virtualized devices
Para-virtualized devices are also known as Virtio devices. They are purely virtual devices designed to work optimally in a virtual machine.

Fedora 19 supports 32 PCI device slots per virtual machine, and 8 PCI functions per device slot. This gives a theoretical maximum of 256 PCI functions per guest when multi-function capabilities are enabled.

However, this theoretical maximum is subject to the following limitations:

- Each virtual machine supports a maximum of 8 assigned device functions.
- 4 PCI device slots are configured with emulated devices by default. However, users can explicitly remove 2 of the emulated devices that are configured by default (the video adapter device in slot 2, and the memory balloon driver device in slot 3). This gives...
users a supported functional maximum of 30 PCI device slots per virtual machine.

Migration restrictions
Device assignment refers to physical devices that have been exposed to a virtual machine, for the exclusive use of that virtual machine. Because device assignment uses hardware on the specific host where the virtual machine runs, migration and save/restore are not supported when device assignment is in use. If the guest operating system supports hot-plugging, assigned devices can be removed prior to the migration or save/restore operation to enable this feature.

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

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

For migration to work, cache=none must be specified for all block devices opened in write mode.

Warning
Failing to include the cache=none option can result in disk corruption.

Storage restrictions
There are risks associated with giving guest virtual machines write access to entire disks or block devices (such as /dev/sdb). If a guest virtual machine has access to an entire block device, it can share any volume label or partition table with the host machine. If bugs exist in the host system’s partition recognition code, this can create a security risk. Avoid this risk by configuring the host machine to ignore devices assigned to a guest virtual machine.

Warning
Failing to adhere to storage restrictions can result in risks to security.

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

- Intel® 82576NS Gigabit Ethernet Controller (igb driver)
- Intel® 82576EB Gigabit Ethernet Controller (igb driver)
- Intel® 82599ES 10 Gigabit Ethernet Controller (ixgbe driver)
- Intel® 82599EB 10 Gigabit Ethernet Controller (ixgbe driver)

Core dumping restrictions
Because core dumping is currently implemented on top of migration, it is not supported when device assignment is in use.

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

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

Multiple PCIe endpoints connected through a non-root PCIe switch require ACS support in the PCIe bridges of the PCIe switch. To disable this restriction, edit the `/etc/libvirt/qemu.conf` file and insert the line:

```
relaxed_acs_check=1
```

Fedora 19 has limited PCI configuration space access by guest device drivers. This limitation could cause drivers that are dependent on PCI configuration space to fail configuration.

Fedora 17 introduced interrupt remapping as a requirement for PCI device assignment. If your platform does not provide support for interrupt remapping, circumvent the KVM check for this support with the following command as the root user at the command line prompt:

```
# echo 1 > /sys/module/kvm/parameters/allow_unsafe_assigned_interrupts
```
4.2. Application restrictions

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

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

The following applications should be avoided due to high I/O requirements:

- kdump server
- netdump server

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

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

4.3. Other restrictions

For the list of all other restrictions and issues affecting virtualization read the **Fedora 19 Release Notes**. The **Fedora 19 Release Notes** cover the present new features, known issues and restrictions as they are updated or discovered.
Chapter 5. Installing the virtualization packages

5.1. Configuring a Virtualization Host installation

5.2. Installing virtualization packages on an existing Fedora system

Before you can use virtualization, the virtualization packages must be installed on your computer. Virtualization packages can be installed either during the host installation sequence or after host installation using the **yum** command and the Fedora Project download page.

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

5.1. Configuring a Virtualization Host installation

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

Procedure 5.1. Installing the virtualization package group

1. **Launch the Fedora installation program**
   Start an interactive Fedora installation from the Fedora Installation CD-ROM, DVD or PXE.

2. **Continue installation up to package selection**
   Complete the other steps up to the package selection step.

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

If you wish to create virtual machines in a graphical user interface (virt-manager) later, you should also select the General Purpose Desktop package group.

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

![Figure 5.3. The Fedora package selection screen](image)

Click on the Close button, then the Next button to continue the installation.

When the installation is complete, reboot the system.

Important

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

Installing KVM packages with Kickstart files

Kickstart files allow for large, automated installations without a user manually installing each individual host system. This section describes how to create and use a Kickstart file to install Fedora with the Virtualization packages.

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

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

For more information about Kickstart files, refer to the Fedora Installation Guide, available from [https://access.redhat.com/knowledge/docs/Red_Hat_Enterprise_Linux/](https://access.redhat.com/knowledge/docs/Red_Hat_Enterprise_Linux/).

5.2. Installing virtualization packages on an existing Fedora system

This section describes the steps for installing the KVM hypervisor on a working Fedora 16 or newer system.

To install the packages, your machines must be registered. There are two methods of registering an unregistered installation of Fedora:

1. To register via RHN Classic, run the `rhn_register` command and follow the prompts.
2. To register via Fedora, run the `subscription-manager register` command and follow the prompts.

If you do not have a valid Fedora subscription, visit the Fedora online store to obtain one.

Installing the virtualization packages with yum

To use virtualization on Fedora you require at least the `qemu-kvm` and `qemu-img` packages. These packages provide the user-level KVM emulator and disk image manager on the host Fedora system.

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

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

Several additional virtualization management packages are also available:

**Recommended virtualization packages**

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

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

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

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

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

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

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

### Installing Virtualization package groups

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

**Table 5.1. Virtualization Package Groups**

<table>
<thead>
<tr>
<th>Package Group</th>
<th>Description</th>
<th>Mandatory Packages</th>
<th>Optional Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtualization</td>
<td>Provides an environment for hosting virtual machines</td>
<td>qemu-kvm</td>
<td>qemu-guest-agent, qemu-kvm-tools</td>
</tr>
<tr>
<td>Virtualization Client</td>
<td>Clients for installing and managing virtualization instances</td>
<td>python-virtinst, virt-manager, virt-viewer</td>
<td>virt-top</td>
</tr>
<tr>
<td>Virtualization Tools</td>
<td>Tools for offline virtual image management</td>
<td>libguestfs</td>
<td>libguestfs-pava, libguestfs-tools, virt-v2v</td>
</tr>
</tbody>
</table>

To install a package group, run the `yum groupinstall <groupname>` command. For instance, to install the Virtualization Tools package group, run the `yum groupinstall "Virtualization Tools"` command.
Tools\textsuperscript{\textregistered} command.
Chapter 6. Guest virtual machine installation overview

6.1. Guest virtual machine prerequisites and considerations

6.2. Creating guests with virt-install

6.3. Creating guests with virt-manager

6.4. Installing guest virtual machines with PXE

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

Detailed installation instructions are available in the following chapters for specific versions of Fedora and Microsoft Windows.

6.1. Guest virtual machine prerequisites and considerations

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

Performance

Guest virtual machines should be deployed and configured based on their intended tasks. Some guest systems (for instance, guests running a database server) may require special performance considerations. Guests may require more assigned CPUs or memory based on their role and projected system load.

Input/Output requirements and types of Input/Output

Some guest virtual machines may have a particularly high I/O requirement or may require further considerations or projections based on the type of I/O (for instance, typical disk block size access, or the amount of clients).

Storage

Some guest virtual machines may require higher priority access to storage or faster disk types, or may require exclusive access to areas of storage. The amount of storage used by guests should also be regularly monitored and taken into account when deploying and maintaining storage.

Networking and network infrastructure

Depending upon your environment, some guest virtual machines could require faster network links than other guests. Bandwidth or latency are often factors when deploying and maintaining guests, especially as requirements or load changes.

Request requirements

SCSI requests can only be issued to guest virtual machines on virtio drives if the virtio drives are backed by whole disks, and the disk device parameter is set to lun, as shown in the following example:

```
<devices>
<emulator>/usr/libexec/qemu-kvm</emulator>
<disk type='block' device='lun'>
</disk>
</devices>
```

6.2. Creating guests with virt-install

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

The virt-install tool provides a number of options that can be passed on the command line. To see a complete list of options run the following command:

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

Note that you need root privileges in order for virt-install commands to complete successfully. The virt-install man page also documents each command option and important variables.

qemu-img is a related command which may be used before virt-install to configure storage options.
An important option is the `--graphics` option which allows graphical installation of a virtual machine.

**Example 6.1. Using virt-install to install a Fedora 19 guest virtual machine**

This example creates a Fedora 19 guest:

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

Ensure that you select the correct `os-type` for your operating system when running this command.

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

**Note**

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

### 6.3. Creating guests with virt-manager

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

**Procedure 6.1. Creating a guest virtual machine with virt-manager**

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

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

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

![Figure 6.1. Virtual Machine Manager window](image)

The New VM wizard breaks down the virtual machine creation process into five steps:

- a. Naming the guest virtual machine and choosing the installation type
- b. Locating and configuring the installation media
- c. Configuring memory and CPU options
- d. Configuring the virtual machine’s storage
- e. Configuring networking, architecture, and other hardware settings

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

4. **Specify name and installation type**

   The guest virtual machine creation process starts with the selection of a name and installation type. Virtual machine names can have underscores (_), periods (.), and hyphens (-).
Figure 6.2. Name virtual machine and select installation method

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

Local install media (ISO image or CDROM)
This method uses a CD-ROM, DVD, or image of an installation disk (for example, .iso).

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

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

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

Click Forward to continue.

5. Configure installation
Next, configure the OS type and Version of the installation. Ensure that you select the appropriate OS type for your virtual machine. Depending on the method of installation, provide the install URL or existing storage path.
Configure CPU and memory

The next step involves configuring the number of CPUs and amount of memory to allocate to the virtual machine. The wizard shows the number of CPUs and amount of memory you can allocate; configure these settings and click Forward.
Figure 6.5. Configuring CPU and Memory

7. Configure storage
Assign storage to the guest virtual machine.

Figure 6.6. Configuring virtual storage

If you chose to import an existing disk image during the first step, virt-manager will skip this step. Assign sufficient space for your virtual machine and any applications it requires, then click Forward to continue.

8. Final configuration
Verify the settings of the virtual machine and click Finish when you are satisfied; doing so will create the virtual machine with default networking settings, virtualization type, and architecture.
If you prefer to further configure the virtual machine's hardware first, check the Customise configuration before install box first before clicking Finish. Doing so will open another wizard that will allow you to add, remove, and configure the virtual machine's hardware settings. After configuring the virtual machine's hardware, click Apply. virt-manager will then create the virtual machine with your specified hardware settings.

6.4. Installing guest virtual machines with PXE

Requirements

PXE guest installation requires a PXE server running on the same subnet as the guest virtual machines you wish to install. The method of accomplishing this depends on how the virtual machines are connected to the network. Contact Support if you require assistance setting up a PXE server.

PXE installation with virt-install

virt-install PXE installations require both the --network=bridge:installation parameter, where Installation is the name of your bridge, and the --pxe parameter.

By default, if no network is found, the guest virtual machine attempts to boot from alternative bootable devices. If there is no other bootable device found, the guest pauses. You can use the qemu-kvm boot parameter reboot-timeout to allow the guest to retry booting if no bootable device is found, like so:

```
# qemu-kvm -boot reboot-timeout=1000
```

Example 6.2. Fully-virtualized PXE installation with virt-install

```
# virt-install --hvm --connect qemu:///system \
   --network=bridge:installation --pxe --graphics spice \
   --name rhel6-machine --ram=756 --vcpus=4 \
   --os-type=linux --os-variant=rhel6 \
   --disk path=/var/lib/libvirt/images/rhel6-machine.img,size=10
```

Note that the command above cannot be executed in a text-only environment. A fully-virtualized (--hvm) guest can only be installed in a text-only environment if the --location and --extra-args "console=console_type" are provided instead of the --graphics spice parameter.
Procedure 6.2. PXE installation with virt-manager

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

   ![Figure 6.8. Selecting the installation method](image1)

   ![Figure 6.9. Selecting the installation type](image2)
2. **Start the installation**
   The installation is ready to start.
Figure 6.12. Finalizing virtual machine details

A DHCP request is sent and if a valid PXE server is found the guest virtual machine’s installation processes will start.
Chapter 7. Installing a Red Hat Enterprise Linux 6 guest virtual machine on a Red Hat Enterprise Linux 6 host

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

This procedure covers creating a Red Hat Enterprise Linux 6 guest virtual machine with a locally stored installation DVD or DVD image. DVD images are available from http://access.redhat.com for Red Hat Enterprise Linux 6.

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

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

   **Important**
   Various storage types may be used for storing guest virtual machines. However, for a virtual machine to be able to use migration features the virtual machine must be created on networked storage.

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

2. **Open virt-manager and start the wizard**
   Open virt-manager by executing the `virt-manager` command as root or opening Applications → System Tools → Virtual Machine Manager.
Click on the **Create a new virtual machine** button to start the new virtualized guest wizard.

![Create a new virtual machine button](image)

**Figure 7.2. The Create a new virtual machine button**

The **New VM** window opens.

3. **Name the virtual machine**

   Virtual machine names can contain letters, numbers and the following characters: ‘:’, ‘.’ and ‘-’. Virtual machine names must be unique for migration and cannot consist only of numbers.

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

   ![New VM window](image)

   **Figure 7.3. The New VM window - Step 1**

   Click **Forward** to continue.

4. **Select the installation media**

   Select the appropriate radio button for your installation media.

   ![New VM window](image)

   **Figure 7.4. Locate your install media**

   If you wish to install from a CD-ROM or DVD, select the **Use CDROM or DVD** radio button, and select the appropriate disk drive from the drop-down list of drives available.

   If you wish to install from an ISO image, select **Use ISO image**, and then click the **Browse...** button to open the **Locate media volume** window.

   Select the installation image you wish to use, and click **Choose Volume**.
If no images are displayed in the Locate media volume window, click on the Browse Local button to browse the host machine for the installation image or DVD drive containing the installation disk. Select the installation image or DVD drive containing the installation disk and click Open; the volume is selected for use and you are returned to the Create a new virtual machine wizard.

**Important**

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

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

![New VM](image)

**Locate your install media**

- Use CDROM or DVD
- Use ISO image: /var/lib/libvirt/images/guest2-rhel6-64.iso

**Choose an operating system type and version**

OS type: Linux
Version: Red Hat Enterprise Linux 6

Click Forward to continue.

5. **Set RAM and virtual CPUs**

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

Virtual machines require sufficient physical memory (RAM) to run efficiently and effectively. Red Hat supports a minimum of 512MB of RAM for a virtual machine. Red Hat recommends at least 1024MB of RAM for each logical core.

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

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

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

*Note*

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

a. With the default local storage

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

b. **With a storage pool**

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

**Locate or create storage volume**

<table>
<thead>
<tr>
<th>Storage Pools</th>
<th>Name</th>
<th>Size</th>
<th>Format</th>
<th>Used By</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot-scratch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boot-scratch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.8. The Locate or create storage volume window

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

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

Choose a format option from the **Format** dropdown menu. Format options include raw, cow, qcow, qcow2, qed, vmdk, and vpc. Adjust other fields as desired.

**New Storage Volume**

Create a storage unit that can be used directly by a virtual machine.

- **Name**: guest2-rhel6-64
- **Format**: raw

**Storage Volume Quota**

Boot-scratch's available space: 17.91 GB

- **Max Capacity**: 8000 MB
- **Allocation**: 0 MB

![New Storage Volume window](image)

Figure 7.9. The Add a Storage Volume window

Click **Finish** to continue.

7. **Verify and finish**

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

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

Click on the **Advanced options** down arrow to inspect and modify advanced options. For a standard Red Hat Enterprise Linux 6 installation, none of these options require modification.
Click **Finish** to continue into the Red Hat Enterprise Linux installation sequence. For more information on installing Red Hat Enterprise Linux 6 refer to the Red Hat Enterprise Linux 6 Installation Guide.

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

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

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

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

   > Various storage types may be used for storing guest virtual machines. However, for a virtual machine to be able to use migration features the virtual machine must be created on networked storage.

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

2. **Open virt-manager and start the wizard**
   
   Open virt-manager by executing the `virt-manager` command as root or opening `Applications → System Tools → Virtual Machine Manager.`
Click on the Create a new virtual machine button to start the new virtual machine wizard.

The Create a new virtual machine window opens.

3. Name the virtual machine
   Virtual machine names can contain letters, numbers and the following characters: ‘‘,’ ‘’ and ‘’. Virtual machine names must be unique for migration and cannot consist only of numbers.
   Choose the installation method from the list of radio buttons.

Click Forward to continue.

4. Provide the installation URL, and the Kickstart URL and Kernel options if required.
5. The remaining steps are the same as the ISO installation procedure. Continue from Step 5 of the ISO installation procedure.

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

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

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

   **Important**
   Various storage types may be used for storing guest virtual machines. However, for a virtual machine to be able to use migration features the virtual machine must be created on networked storage.

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

2. **Open virt-manager and start the wizard**
   Open virt-manager by executing the `virt-manager` command as root or opening **Applications → System Tools → Virtual Machine Manager**.
Click on the *Create new virtualized guest* button to start the new virtualized guest wizard.

The *New VM* window opens.

3. Name the virtual machine

Virtual machine names can contain letters, numbers and the following characters: `'`, `.` and `-`. Virtual machine names must be unique for migration and cannot consist only of numbers.

Choose the installation method from the list of radio buttons.

Click **Forward** to continue.

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

From this point, the only difference in this PXE procedure is on the final *New VM* screen, which shows the **Install: PXE Install** field.
Chapter 8. Virtualizing Fedora on Other Platforms

8.1. On VMWare

Fedora 17 and onwards provide the `vmxnet3` driver, a para-virtualized network adapter used when running Red Hat Enterprise Linux on VMWare hosts. For further information about this driver, refer to http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1001805.

Fedora 18 and onwards provide the `vmw_pvscsi` driver, a para-virtualized SCSI adapter used when running Red Hat Enterprise Linux on VMWare hosts. For further information about this driver, refer to http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1010398.

8.2. On Hyper-V

Fedora 18 and onwards provide Microsoft's Linux Integration Services, a set of drivers that enable synthetic device support in supported virtualized operating systems. Fedora is a supported virtualized operating system under Linux Integration Services version 3.4. Further details about the drivers provided are available from http://www.microsoft.com/en-us/download/details.aspx?id=34603.
Chapter 9. Installing a fully-virtualized Windows guest

9.1. Using virt-install to create a guest

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

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

Installing a Windows operating system involves two major steps:

1. Creating the guest virtual machine, using either `virt-install` or `virt-manager`.
2. Installing the Windows operating system on the guest virtual machine, using `virt-viewer`.

Refer to Chapter 6, Guest virtual machine installation overview for details about creating a guest virtual machine with `virt-install` or `virt-manager`.

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

9.1. Using virt-install to create a guest

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

Important

Before creating the guest, consider first if the guest needs to use KVM Windows para-virtualized drivers. If it does, keep in mind that you can do so during or after installing the Windows operating system on the guest. For more information about para-virtualized drivers, refer to Chapter 10, KVM Para-virtualized Drivers. For instructions on how to install KVM para-virtualized drivers, refer to Section 10.1, “Installing the KVM Windows para-virtualized drivers”.

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

```
# virt-install \\
  --name=guest-name \\
  --os-type=windows \\
  --network network=default \\
  --disk path=,size= \\
  --cdrom= \\
  --graphics spice --ram=1024
```

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

Important

All image files are stored in `/var/lib/libvirt/images/` by default. Other directory locations for file-based images are possible, but may require SELinux configuration. If you run SELinux in enforcing mode.

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

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

Once the fully-virtualized guest is created, `virt-viewer` will launch the guest and run the operating system's installer. Refer to the relevant Microsoft installation documentation for instructions on how to install the operating system.
Chapter 10. KVM Para-virtualized Drivers

10.1. Installing the KVM Windows para-virtualized drivers

10.2. Installing the drivers on an installed Windows guest virtual machine

10.3. Installing drivers during the Windows installation

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

10.5. Using KVM para-virtualized drivers for existing devices

10.6. Using KVM para-virtualized drivers for new devices

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

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

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

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

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

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

**Note**
PCI devices are limited by the virtualized system architecture. Refer to Chapter 12, **PCI device configuration** for additional limitations when using assigned devices.

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

- Windows XP Service Pack 3 and newer (32-bit only)
- Windows Server 2003 (32-bit and 64-bit versions)
- Windows Server 2008 (32-bit and 64-bit versions)
- Windows Server 2008 R2 (64-bit only)
- Windows 7 (32-bit and 64-bit versions)

10.1. Installing the KVM Windows para-virtualized drivers

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

You can install the para-virtualized drivers on a guest virtual machine using one of the following methods:

- Hosting the installation files on a network accessible to the virtual machine,
- Using a virtualized CD-ROM device of the driver installation disk .iso file, or
- Using a virtualized floppy device to install the drivers during boot time.

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

1. **Download the drivers**
   The virtio-win package contains the para-virtualized block and network drivers for all supported Windows guest virtual machines.
The virtio-win package can be found here in RHN:
https://rhn.redhat.com/rhn/software/packages/details/Overview.do?pid=602010. It requires access to one of the following channels:
- RHEL Client Supplementary (v. 6)
- RHEL Server Supplementary (v. 6)
- RHEL Workstation Supplementary (v. 6)

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

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

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


2. Install the para-virtualized drivers
   It is recommended to install the drivers on the virtual machine before attaching or modifying a device to use the para-virtualized drivers.
   For block devices storing root file systems or other block devices required for booting the virtual machine, the drivers must be installed before the device is modified. If the drivers are not installed on the virtual machine and the driver is set to the virtio driver the virtual machine will not boot.

10.2. Installing the drivers on an installed Windows guest virtual machine
This procedure covers installing the para-virtualized drivers with a virtualized CD-ROM after Windows is installed.

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

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

1. Open virt-manager and the guest virtual machine
   Open `virt-manager`, then open the guest virtual machine from the list by double-clicking the guest name.

2. Open the hardware window
   Click the lightbulb icon on the toolbar at the top of the window to view virtual hardware details.

   ![Figure 10.1. The virtual hardware details button](image)

   Then click the Add Hardware button at the bottom of the new view that appears.

   ![Figure 10.2. The virtual machine hardware information window](image)

   This opens a wizard for adding the new device.

3. Select the device type — for Red Hat Enterprise Linux 6 versions prior to 6.2
   Skip this step if you are using Red Hat Enterprise Linux 6.2 or later.
   On Red Hat Enterprise Linux 6 versions prior to version 6.2, you must select the type of device you wish to add. In this case, select Storage from the dropdown menu.
Click the Finish button to proceed.

4. Select the ISO file
   Ensure that the Select managed or other existing storage radio button is selected, and browse to the para-virtualized driver's .iso image file. The default location for the latest version of the drivers is /usr/share/virtio-win/virtio-win.iso.
   Change the Device type to IDE cdrom and click the Forward button to proceed.

5. Finish adding virtual hardware — for Red Hat Enterprise Linux 6 versions prior to 6.2
   If you are using Red Hat Enterprise Linux 6.2 or later, skip this step.
   On Red Hat Enterprise Linux 6 versions prior to version 6.2, click on the Finish button to finish adding the virtual hardware and close the wizard.
6. **Reboot**

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

Once the CD-ROM with the drivers is attached and the virtual machine has started, proceed with Procedure 10.2, "Windows installation on a Windows 7 virtual machine."

**Procedure 10.2. Windows installation on a Windows 7 virtual machine**

This procedure installs the drivers on a Windows 7 virtual machine as an example. Adapt the Windows installation instructions to your guest's version of Windows.

1. **Open the Computer Management window**

   On the desktop of the Windows virtual machine, click the **Windows** icon at the bottom corner of the screen to open the Start menu. Right-click on **Computer** and select **Manage** from the pop-up menu.

2. **Open the Device Manager**

   Select the **Device Manager** from the left-most pane. This can be found under **Computer Management > System Tools**.
3. Start the driver update wizard
   a. View available system devices
      Expand **System devices** by clicking on the arrow to its left.

   b. Locate the appropriate device
      There are up to four drivers available: the balloon driver, the serial driver, the network driver, and the block driver.
      - **Balloon**, the balloon driver, affects the **PCI standard RAM Controller** in the **System devices** group.
      - **viserial**, the serial driver, affects the **PCI Simple Communication Controller** in the **System devices** group.
      - **NetKVM**, the network driver, affects the **Network adapters** group. This driver is only available if a virtio NIC is configured. Configurable parameters for this driver are documented in **Appendix C, NetKVM Driver Parameters**.
      - **viostor**, the block driver, affects the **Disk drives** group. This driver is only available if a virtio disk is configured.
      Right-click on the device whose driver you wish to update, and select **Update Driver**... from the pop-up menu.
      This example installs the balloon driver, so right-click on **PCI standard RAM Controller**.

   c. Open the driver update wizard
      From the drop-down menu, select
**Update Driver Software**... to access the driver update wizard.

---

**Figure 10.10. Opening the driver update wizard**

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

---

**Figure 10.11. The driver update wizard**

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

---

**Figure 10.12. The driver update wizard**

b. **Browse to the location of the driver**
   A separate driver is provided for each of the various combinations of operating system and architecture. The drivers are arranged hierarchically according to their driver type, the operating system, and the architecture on which they will be installed: `driver_type/os/arch/`. For example, the Balloon driver for a Windows 7 operating system with an x86 (32-bit) architecture, resides in the `Balloon/w7/x86` directory.
Once you have navigated to the correct location, click OK.

c. Click Next to continue.

The following screen is displayed while the driver installs:

6. Close the installer
The following screen is displayed when installation is complete:
7. Reboot
Reboot the virtual machine to complete the driver installation.

**10.3. Installing drivers during the Windows installation**

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

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

**Procedure 10.3. Installing para-virtualized drivers during the Windows installation**

1. Install the virtio-win package:

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

   **Note**

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

2. Creating the guest virtual machine

   **Important**

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

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

   a. **Creating the guest virtual machine with virsh**

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

      If the virtual machine is created from an XML definition file with `virsh`, use the `virsh define` command not the `virsh create` command.

      a. Create, but do not start, the virtual machine. Refer to the Red Hat Enterprise Linux Virtualization Administration Guide for details on creating virtual machines with the `virsh` command.

      b. Add the driver disk as a virtualized floppy disk with the `virsh` command. This example can be copied and used if there are no other virtualized floppy devices attached to the guest virtual machine. Note that `vm_name` should be replaced with the name of the virtual machine.

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

      You can now continue with Step 3.

   b. **Creating the guest virtual machine with virt-manager and changing the disk type**
a. At the final step of the virt-manager guest creation wizard, check the **Customize configuration before install** checkbox.

![Create a new virtual machine](image1)

Figure 10.17. The virt-manager guest creation wizard

Click on the **Finish** button to continue.

b. Open the Add Hardware wizard

Click the **Add Hardware** button in the bottom left of the new panel.

![Add Hardware button](image2)

Figure 10.18. The Add Hardware button

c. Select storage device

**Storage** is the default selection in the **Hardware type** list.

![Storage](image3)

Figure 10.19. The Add new virtual hardware wizard

Ensure the **Select managed or other existing storage** radio button is selected. Click **Browse**...  

![Select managed or other existing storage](image4)

Figure 10.20. Select managed or existing storage

In the new window that opens, click **Browse Local**. Navigate to `/usr/share/virtio-win/virtio-win.vfd`, and click **Select** to confirm. Change **Device type** to **Floppy disk**, and
click Finish to continue.

Figure 10.21. Change the Device type

d. Confirm settings

Review the device settings.

Figure 10.22. The virtual machine hardware information window

You have now created a removable device accessible by your virtual machine.

e. Change the hard disk type

To change the hard disk type from IDE Disk to Virtio Disk, we must first remove the existing hard disk, Disk 1. Select the disk and click on the Remove button.

Figure 10.23. The virtual machine hardware information window

Add a new virtual storage device by clicking Add Hardware. Then, change the Device type from IDE disk to Virtio Disk. Click Finish to confirm the operation.
f. Ensure settings are correct

Review the settings for *VirtIO Disk 1*.

When you are satisfied with the configuration details, click the **Begin Installation** button.

You can now continue with **Step 3**.

c. Creating the guest virtual machine with *virt-install*

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

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

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

- For Windows 7 or 8, append:

  ```
  --boot=1
  ```

  Important

  If the device you wish to add is a **disk** (that is, not a **floppy** or a **cdrom**), you will also need to add the **bus=virtio** option to the end of the *--disk* parameter, like so:

  ```
  --disk path=/usr/share/virtio-win/virtio-win.vfd,device=disk,bus=virtio
  ```
3. Additional steps for driver installation
   During the installation, additional steps are required to install drivers, depending on the type of Windows guest.
   
a. Windows Server 2003 and Windows XP
   Before the installation blue screen repeatedly press **F6** for third party drivers.

   Press **S** to install additional device drivers.

   Press **Enter** to continue the installation.

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

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

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

Note

To use the network device driver only, load the virtio, virtio_net and virtio_pci modules. To use the block device driver only, load the virtio_virtio_ring, virtio_blk and virtio_pci modules.

Important

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

Loading the para-virtualized drivers with the sysinit script

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

The drivers must be loaded in the following order:

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

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

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

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

Append the following lines after that section:

```
if [-f /etc/rc.modules ]; then
  /etc/rc.modules
fi
modprobe virtio
modprobe virtio_ring # Comment this out if you do not need block driver
modprobe virtio_blk # Comment this out if you do not need block driver
modprobe virtio_net # Comment this out if you do not need net driver
modprobe virtio_pci
```

Reboot the guest virtual machine to load the kernel modules.

Adding the para-virtualized drivers to the initrd RAM disk

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

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

AMD64 and Intel 64 issues

Use the x86_64 version of the virtio package for AMD64 systems.
Use the ia32e version of the virtio package for Intel 64 systems. Using the x86_64 version of the virtio may cause a ‘Unresolved symbol’ error during the boot sequence on Intel 64 systems.

Network performance issues

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

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

```
# ethtool -k interface
```

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

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

Para-virtualized driver swap partition issue

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

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

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

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

Save the changes and reboot the guest virtual machine. The virtual machine should now have swap partitions.

10.5. Using KVM para-virtualized drivers for existing devices

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

Procedure 10.4. Using KVM para-virtualized drivers for existing devices

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

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

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

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

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

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

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

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

Refer to the libvirt website for more details on using Virtio:
10.6. Using KVM para-virtualized drivers for new devices

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

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

**Important**

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

Procedure 10.5. Starting the new device wizard

1. Open the guest virtual machine by double clicking on the name of the guest in `virt-manager`.
2. Open the `Show virtual hardware details` tab by clicking the lightbulb button.

![Figure 10.30. The Show virtual hardware details tab](image)

3. In the `Show virtual hardware details` tab, click on the `Add Hardware` button.
4. In the Adding Virtual Hardware tab select `Storage` or `Network` for the type of device. The storage and network device wizards are covered in procedures Procedure 10.6, “Adding a storage device using the para-virtualized storage driver” and Procedure 10.7, “Adding a network device using the para-virtualized network driver”.

Procedure 10.6. Adding a storage device using the para-virtualized storage driver

1. Open the guest virtual machine by double clicking on the name of the guest in `virt-manager`.
2. Open the `Show virtual hardware details` tab by clicking the lightbulb button.

![Figure 10.31. The Show virtual hardware details tab](image)

3. In the `Show virtual hardware details` tab, click on the `Add Hardware` button.
4. Select `Network` as the `Hardware type`.
5. Select the network device and driver

Create a new disk image or select a storage pool volume.
Set the Device type to Virtio Disk to use the para-virtualized drivers. Choose the desired Host device.

Figure 10.33. The Add new virtual hardware wizard

Click Finish to complete the procedure.

Procedure 10.7. Adding a network device using the para-virtualized network driver

1. Select hardware type
   - Select Network as the Hardware type.

   ![Figure 10.34. The Add new virtual hardware wizard](image)

   - Click Forward to continue.

2. Select the network device and driver
   - Select the network device from the Host device list.
   - Create a custom MAC address or use the one provided.
   - Set the Device model to virtio to use the para-virtualized drivers.
Figure 10.35. The Add new virtual hardware wizard

Click **Forward** to continue.

3. **Finish the procedure**

Confirm the details for the new device are correct.

![Add new virtual hardware wizard](image)

Figure 10.36. The Add new virtual hardware wizard

Click **Finish** to complete the procedure.

Once all new devices are added, reboot the virtual machine. Windows virtual machines may not recognize the devices until the guest is rebooted.
Chapter 11. Network configuration

11.1. Network Address Translation (NAT) with libvirt

11.2. Disabling vhost-net

11.3. Bridged networking with libvirt

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

Fedora supports the following networking setups for virtualization:

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

You must enable NAT, network bridging or directly assign a PCI device to allow external hosts access to network services on guest virtual machines.

### 11.1. Network Address Translation (NAT) with libvirt

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

**Host configuration**

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

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

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

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

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

Mark the default network to automatically start:

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

Start the default network:

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

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

```
# brctl show
bridge name     bridge id               STP enabled
interfaces
virbr0          8000.000000000000       yes
```

**Guest virtual machine configuration**

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

```
<interface type='network'>
  <source network='default'/>
</interface>
```
Defining a MAC address is optional. If you do not define one, a MAC address is automatically generated and used as the MAC address of the bridge device used by the network. Manually setting the MAC address may be useful to maintain consistency or easy reference throughout your environment, or to avoid the very small chance of a conflict.

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

### 11.2. Disabling vhost-net

The **vhost-net** module is a kernel-level backend forvirtio networking that reduces virtualization overhead by moving virtio packet processing tasks out of user space (the qemu process) and into the kernel (the **vhost-net** driver). vhost-net is only available for virtio network interfaces. If the vhost-net kernel module is loaded, it is enabled by default for all virtio interfaces, but can be disabled in the interface configuration in the case that a particular workload experiences a degradation in performance when vhost-net is in use.

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

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

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

Setting the driver name to **qemu** forces packet processing into qemu user space, effectively disabling vhost-net for that interface.

### 11.3. Bridged networking with libvirt

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

To create a bridge (**br0**) based on the **eth0** interface, execute the following command on the host:

```
# virsh iface-bridge eth0 br0
```

**Important**

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

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

If you do not want to disable NetworkManager entirely, add "NM_CONTROLLED=no" to the `ifcfg-*` network script being used for the bridge.
Chapter 12. PCI device configuration

12.1. Assigning a PCI device with virsh
12.2. Assigning a PCI device with virt-manager
12.3. PCI device assignment with virt-install
12.4. Detaching an assigned PCI device

Red Hat Enterprise Linux 6 exposes three classes of device to its virtual machines:

- **Emulated devices** are purely virtual devices that mimic real hardware, allowing unmodified guest operating systems to work with them using their standard in-box drivers.
- **Virtio devices** are purely virtual devices designed to work optimally in a virtual machine. Virtio devices are similar to emulated devices, however, non-Linux virtual machines do not include the drivers they require by default. Virtualization management software like the Virtual Machine Manager (virt-manager) and the Red Hat Enterprise Virtualization Hypervisor install these drivers automatically for supported non-Linux guest operating systems.
- **Assigned devices** are physical devices that are exposed to the virtual machine. This method is also known as ‘passthrough’. Device assignment allows virtual machines exclusive access to PCI devices for a range of tasks, and allows PCI devices to appear and behave as if they were physically attached to the guest operating system. Device assignment is supported on PCI Express devices, except graphics cards. Parallel PCI devices may be supported as assigned devices, but they have severe limitations due to security and system configuration conflicts.

Red Hat Enterprise Linux 6 supports 32 PCI device slots per virtual machine, and 8 PCI functions per device slot. This gives a theoretical maximum of 256 configurable PCI functions per guest. However, this theoretical maximum is subject to the following limitations:

- Each virtual machine supports a maximum of 8 assigned device functions.
- 4 PCI device slots are configured with emulated devices by default. However, users can explicitly remove 2 of the emulated devices that are configured by default (the video adapter device in slot 2, and the memory balloon driver device in slot 3). This gives users a supported functional maximum of 30 PCI device slots per virtual machine.

Red Hat Enterprise Linux 6.0 and newer supports hot plugging assigned PCI devices into virtual machines. However, PCI device hot plugging operates at the slot level and therefore does not support multi-function PCI devices. Multi-function PCI devices are recommended for static device configuration only.

Red Hat Enterprise Linux 6.0 limited guest operating system driver access to a device’s standard and extended configuration space. Limitations that were present in Red Hat Enterprise Linux 6.0 are significantly reduced in Red Hat Enterprise Linux 6.1, and enable a much larger set of PCI Express devices to be successfully assigned to KVM guests.

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

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

Procedure 12.1. Preparing an Intel system for PCI device assignment

1. **Enable the Intel VT-d specifications**
   The Intel VT-d specifications provide hardware support for directly assigning a physical device to a virtual machine. These specifications are required to use PCI device assignment with Red Hat Enterprise Linux.
   The Intel VT-d specifications must be enabled in the BIOS. Some system manufacturers disable these specifications by default. The terms used to refer to these specifications can differ between manufacturers; consult your system manufacturer’s documentation for the appropriate terms.

2. **Activate Intel VT-d in the kernel**
   Activate Intel VT-d in the kernel by adding the intel_iommu=on parameter to the kernel line in the /boot/grub/grub.conf file.
The example below is a modified `grub.conf` file with Intel VT-d activated.

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.32-330.0.64S.0)
  root (hd0,0)
  kernel /vmlinuz-2.6.32-330.x86_64 ro
  root=/dev/VolGroup00/LogVol00 rhgb quiet
  intel_iommu=on
  initrd /initrd-2.6.32-330.0.64S.0.img
```

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

Procedure 12.2. Preparing an AMD system for PCI device assignment

1. Enable the AMD IOMMU specifications
   The AMD IOMMU specifications are required to use PCI device assignment in Red Hat Enterprise Linux. These specifications must be enabled in the BIOS. Some system manufacturers disable these specifications by default.

2. Enable IOMMU kernel support
   Append `amd_iommu=on` to the kernel command line in `/boot/grub/grub.conf` so that AMD IOMMU specifications are enabled at boot.

12.1. Assigning a PCI device with virsh
These steps cover assigning a PCI device to a virtual machine on a KVM hypervisor.

This example uses a PCIe network controller with the PCI identifier code, `pci_0000_01_00_0`, and a fully virtualized guest machine named `guest1-F19`.

Procedure 12.3. Assigning a PCI device to a guest virtual machine with virsh

1. Identify the device
   First, identify the PCI device designated for device assignment to the virtual machine. Use the `lspci` command to list the available PCI devices. You can refine the output of `lspci` with `grep`.
   This example uses the Ethernet controller highlighted in the following output:

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

   This Ethernet controller is shown with the short identifier `00:19.0`. We need to find out the full identifier used by `virsh` in order to assign this PCI device to a virtual machine.

   To do so, combine the `virsh nodetdev-list` command with the `grep` command to list all devices of a particular type (`pci`) that are attached to the host machine. Then look at the output for the string that maps to the short identifier of the device you wish to use.

   This example highlights the string that maps to the Ethernet controller with the short identifier `00:19.0`. Note that the : and . characters are replaced with underscores in the full identifier.
Record the PCI device number that maps to the device you want to use; this is required in other steps.

2. Review device information

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

```xml
<device>
  <name>pci_0000_00_19_0</name>
  <parent>computer</parent>
  <driver>
    <name>e1000e</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>0</bus>
    <slot>25</slot>
    <function>0</function>
    <product id='0x1502'>82579LM Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
      <capability>
      </capability>
    </capability>
  </capability>
</device>
```

3. Determine required configuration details

Refer to the output from the `virsh nodedev-dumpxml pci_0000_00_19_0` command for the values required for the configuration file.

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

The example device has the following values: bus = 0, slot = 25 and function = 0. The decimal configuration uses those three values:

```xml
bus='0'
slot='25'
function='0'
```

If you want to convert to hexadecimal values, you can use the `printf` utility to convert from decimal values, as shown in the following example:

```
$ printf %x 8
0
$ printf %x 25
19
$ printf %x 0
0
```

The example device would use the following hexadecimal values in the configuration file:

```xml
bus='0x0'
slot='0x19'
function='0x0'
```

4. Add configuration details

Run `virsh edit`, specifying the virtual machine name, and add a device entry in the `<source>` section to assign the PCI device.
to the guest virtual machine.

```
# virsh edit guest1-F19
<hostdev mode='subsystem' type='pci'
   managed='yes'>
  <source>
    <address domain='0x0' bus='0x0' slot='0x19'
      function='0x0'/>
  </source>
</hostdev>
```

Alternately, run `virsh attach-device`, specifying the virtual machine name and the guest's XML file:

```
virsh attach-device guest1-F19 file.xml
```

5. **Allow device management**

Set an SELinux boolean to allow the management of the PCI device from the virtual machine:

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

6. **Start the virtual machine**

```
# virsh start guest1-F19
```

The PCI device should now be successfully assigned to the virtual machine, and accessible to the guest operating system.

### 12.2. Assigning a PCI device with virt-manager

PCI devices can be added to guest virtual machines using the graphical `virt-manager` tool. The following procedure adds a Gigabit Ethernet controller to a guest virtual machine.

**Procedure 12.4. Assigning a PCI device to a guest virtual machine using virt-manager**

1. **Open the hardware settings**

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

![Figure 12.1. The virtual machine hardware information window](image)

2. **Select a PCI device**

Select **PCI Host Device** from the **Hardware** list on the left. Select an unused PCI device. Note that selecting PCI devices presently in use on the host causes errors. In this example, a spare 82576 network device is used. Click **Finish** to complete setup.
Figure 12.2. The Add new virtual hardware wizard

3. Add the new device

The setup is complete and the guest virtual machine now has direct access to the PCI device.

Figure 12.3. The virtual machine hardware information window

12.3. PCI device assignment with virt-install

To use virt-install to assign a PCI device, use the --host-device parameter.

Procedure 12.5. Assigning a PCI device to a virtual machine with virt-install

1. Identify the device

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

```bash
# lspci | grep Ethernet
00:10.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit Network Connection
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection [rev 01]
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection [rev 01]
```

The `virsh nodedev-list` command lists all devices attached to the system, and identifies each PCI device with a string. To limit output to only PCI devices, run the following command:
```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_00_1f_4
pci_0000_00_1f_7
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

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

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

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

2. **Add the device**

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

```
virt-install \
  --name=guest1-F19 \
  --disk path=/var/lib/libvirt/images/guest1-F19.img,size=8 \
  --nonsparse --graphics spice \
  --vcpus=2 --ram=2048 \
  --location=http://example1.com/installation_tree/F19-Server-x86_64/os \
  --nonetworks \
  --os-type=linux \
  --os-variant=fedora \
  --host-device=pci_0000_01_00_0
```

3. **Complete the installation**

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

### 12.4. Detaching an assigned PCI device

When a host PCI device has been assigned to a guest machine, the host can no longer use the device. Read this section to learn how to detach the device from the guest with `virsh` or `virt-manager` so it is available for host use.

Procedure 12.6. Detaching a PCI device from a guest with `virsh`

1. **Detach the device**

Use the following command to detach the PCI device from the guest by removing it in the guest’s XML file:

```
# virsh detach-device name_of_guest file.xml
```

2. **Re-attach the device to the host (optional)**
If the device is in managed mode, skip this step. The device will be returned to the host automatically.

If the device is not using managed mode, use the following command to re-attach the PCI device to the host machine:

```
# virsh nodedev-reattach device
```

For example, to re-attach the `pci_0000_01_00_0` device to the host:

```
virsh nodedev-reattach pci_0000_01_00_0
```

The device is now available for host use.

**Procedure 12.7. Detaching a PCI Device from a guest with virt-manager**

1. **Open the virtual hardware details screen**
   In virt-manager, double-click on the virtual machine that contains the device. Select the Show virtual hardware details button to display a list of virtual hardware.

   ![Figure 12.4. The virtual hardware details button](image)

2. **Select and remove the device**
   Select the PCI device to be detached from the list of virtual devices in the left panel.

   ![Figure 12.5. Selecting the PCI device to be detached](image)

   Click the Remove button to confirm. The device is now available for host use.
Chapter 13. SR-IOV

13.1. Introduction

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

Figure 13.1. How SR-IOV works

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

SR-IOV uses two PCI functions:

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

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

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

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

Advantages of SR-IOV

SR-IOV devices can share a single physical port with multiple virtual machines.

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

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

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

13.2. Using SR-IOV

This section covers the use of PCI passthrough to assign a Virtual Function of an SR-IOV capable multiport network card to a virtual machine as a network device.

SR-IOV Virtual Functions (VFs) can be assigned to virtual machines by adding a device entry in `<hostdev>` with the `virsh edit` or `virsh attach-device` command. However, this can be problematic because unlike a regular network device, an SR-IOV VF network device does not have a permanent unique MAC address, and is assigned a new MAC address each time the host is rebooted. Because of this, even if the guest is assigned the same VF after a reboot, when the host is rebooted the guest determines its network adapter to have a new MAC address. As a result, the guest believes there is new hardware connected each time, and will usually require re-configuration of the guest’s network settings.

`libvirt-0.9.10` and newer contains the `<interface type='hostdev'>` interface device. Using this interface device, `libvirt` will first perform any network-specific hardware/switch initialization indicated (such as setting the MAC address, VLAN tag, or 802.1Qbh virtualport parameters), then perform the PCI device assignment to the guest.
Using the `<interface type='hostdev'>` interface device requires:

1. an SR-IOV-capable network card,
2. host hardware that supports either the Intel VT-d or the AMD IOMMU extensions, and
3. the PCI address of the VF to be assigned.

**Important**

Assignment of an SR-IOV device to a virtual machine requires that the host hardware supports the Intel VT-d or the AMD IOMMU specification.

To attach an SR-IOV network device on an Intel or an AMD system, follow this procedure:

**Procedure 13.1. Attach an SR-IOV network device on an Intel or AMD system**

1. **Enable Intel VT-d or the AMD IOMMU specifications in the BIOS and kernel**
   
   On an Intel system, enable Intel VT-d in the BIOS if it is not enabled already. Refer to **Procedure 12.1, "Preparing an Intel system for PCI device assignment"** for procedural help on enabling Intel VT-d in the BIOS and kernel.
   
   Skip this step if Intel VT-d is already enabled and working.
   
   On an AMD system, enable the AMD IOMMU specifications in the BIOS if they are not enabled already. Refer to **Procedure 12.2, "Preparing an AMD system for PCI device assignment"** for procedural help on enabling IOMMU in the BIOS.

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

   ```
   # lspci
   03:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
   03:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
   ```
   
   Note that the output has been modified to remove all other devices.

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

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

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

   ```
   # modprobe -r igb
   # modprobe igb max_vfs=7
   ```

5. **Make the Virtual Functions persistent**
   
   Add the line `options igb max_vfs=7` to any file in `/etc/modprobe.d` to make the Virtual Functions persistent. For example:

   ```
   # echo "options igb max_vfs=7" >>/etc/modprobe.d/igb.conf
   ```

6. **Inspect the new Virtual Functions**
   
   Using the `lspci` command, list the newly added Virtual Functions attached to the Intel 82576 network device. (Alternatively, use `grep` to search for `Virtual Function`, to search for devices that support Virtual Functions.)
The identifier for the PCI device is found with the `-n` parameter of the `lspci` command. The Physical Functions correspond to `0b:00.0` and `0b:00.1`. All Virtual Functions have `Virtual Function` in the description.

7. Verify devices exist with `virsh`

The `libvirt` service must recognize the device before adding a device to a virtual machine. `libvirt` uses a similar notation to the `lspci` output. All punctuation characters, ; and ., in `lspci` output are changed to underscores (_).

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

```
# virsh nodedev-list | grep 0b
pci_0000_0b_00_0
pci_0000_0b_00_1
pci_0000_0b_10_0
pci_0000_0b_10_1
pci_0000_0b_10_2
pci_0000_0b_10_3
pci_0000_0b_10_4
pci_0000_0b_10_5
pci_0000_0b_10_6
pci_0000_0b_10_7
pci_0000_0b_11_0
pci_0000_0b_11_1
pci_0000_0b_11_2
pci_0000_0b_11_3
pci_0000_0b_11_4
pci_0000_0b_11_5
```

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

8. Get device details with `virsh`

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

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

```
# virsh nodedev-dumpxml pci_0000_0b_10_0
<device>
  <name>pci_0000_0b_10_0</name>
  <parent>/pci_0000_0b_00_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>8</domain>
    <bus>0</bus>
    <slot>8</slot>
    <function>0</function>
    <product id='0x8086'>Intel Corporation</product>
    <vendor id='0x8086'>Intel 82576 Gigabit Network Connection</vendor>
  </capability>
</device>
```
virsh nodedev-dumpxml pci_0000_0b_10_0
<device>
  <name>pci_0000_0b_10_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igbvf</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>11</bus>
    <slot>16</slot>
    <function>0</function>
    <product id='0x10ca'>Intel Corporation</product>
    <vendor id='0x8086'>82576 Virtual Function</vendor>
  </capability>
</device>

This example adds the Virtual Function pci_0000_0b_10_0 to the virtual machine in Step 9. Note the bus, slot and function parameters of the Virtual Function: these are required for adding the device.

Copy these parameters into a temporary XML file, such as /tmp/new-interface.xml for example.

<interface type='hostdev' managed='yes'>
  <source>
    <address type='pci' domain='0' bus='11' slot='16' function='0'/>
  </source>
</interface>

Note

If you do not specify a MAC address, one will be automatically generated. The <virtualport> element is only used when connecting to an 802.11Qbh hardware switch. The <vlan> element was introduced for Fedora 18 and this will transparently put the guest's device on the VLAN tagged 42.

When the virtual machine starts, it should see a network device of the type provided by the physical adapter, with the configured MAC address. This MAC address will remain unchanged across host and guest reboots.

The following <interface> example shows the syntax for the optional <mac address>, <virtualport>, and <vlan> elements. In practice, use either the <vlan> or <virtualport> element, not both simultaneously as shown in the example:

...<devices>
...
  <interface type='hostdev' managed='yes'>
    <source>
      <address type='pci' domain='0' bus='11' slot='16' function='0'/>
    </source>
    <mac address='52:54:00:6d:90:02'>
      <vlan>
        <tag id='42'/>
      </vlan>
      <virtualport type='802.1Qbh'>
        <parameters profileid='finance'/>
      </virtualport>
    </mac>
  </interface>
...
</devices>

9. Add the Virtual Function to the virtual machine

Add the Virtual Function to the virtual machine using the following command with the temporary file created in the previous step. This attaches the new device immediately and saves it for subsequent guest restarts.

virsh attach-device MyGuest /tmp/new-interface.xml --live --config

Specifying the --live option with virsh attach-device attaches the new device to the running guest. Using the --config option ensures the new device is available after future guest restarts.

Note

The --live option is only accepted when the guest is running. virsh will return an error if the --live option is used on a non-running guest.

The virtual machine detects a new network interface card. This new card is the Virtual Function of the SR-IOV device.
13.3. Troubleshooting SR-IOV

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

Error starting the guest

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

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

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

Error migrating, saving, or dumping the guest

Attempts to migrate and dump the virtual machine cause an error similar to the following:

```
# virsh dump --crash 5 /tmp/vmcore
error: Failed to core dump domain 5 to /tmp/vmcore
error: internal error unable to execute QEMU
command 'migrate': An undefined error has occurred
```

Because device assignment uses hardware on the specific host where the virtual machine was started, guest migration and save are not supported when device assignment is in use. Currently, the same limitation also applies to core-dumping a guest; this may change in the future.
Chapter 14. KVM guest timing management

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

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

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

Note

Fedora 17 and newer, uses `kvm-clock` as their default clock source. Running without `kvm-clock` requires special configuration, and is not recommended.

Important

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

```
# service ntpd start
```

Add the ntpd service to the default startup sequence:

```
# chkconfig ntpd on
```

The `ntpd` service will correct the effects of clock skew as long as the clock runs no more than 0.05% faster or slower than the reference time source. The `ntp` startup script adjusts the clock offset from the reference time by adjusting the system clock at startup time, if required.

Constant Time Stamp Counter (TSC)

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

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

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

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

Configuring hosts without a constant Time Stamp Counter

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

Important

These instructions are for AMD revision F CPUs only.

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

```
title Fedora (2.6.32-330.x86_64)
r
```
frequency available. Valid limits can be found in the
/sys/devices/system/cpu/cpu*/cpufreq/scaling_available_frequencies files.

Required parameters for Fedora Linux guests

For certain Fedroa guest virtual machines, additional kernel parameters
are required. These parameters can be set by appending them to the end
of the /kernel line in the /boot/grub/grub.conf file of the guest
virtual machine.

Note

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

Warning

The divider kernel parameter was previously recommended for
Fedora guest virtual machines that did not have high
responsiveness requirements, or exist on systems with high guest
density. It is no longer recommended for use with guests running
Fedora versions prior to version 16.

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

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

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

```
/usepmtimer
```

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

Using the Real-Time Clock with Windows Server 2008, Windows
Server 2008 R2, and Windows 7 guests

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

The boot.ini file is no longer used as of Windows Server 2008 and
do not use the TSC as a time source if the hypervisor-present bit is
set. The Fedora KVM hypervisor enables this CPUID bit by default, so it is
no longer necessary to use the Boot Configuration Data Editor
(bcdedit.exe) to modify the Windows boot parameters.

1. Open the Windows guest virtual machine.
2. Open the Accessories menu of the start menu. Right click on the
   Command Prompt application, select
   Run as Administrator.
3. Confirm the security exception, if prompted.
4. Set the boot manager to use the platform clock. This should
   instruct Windows to use the PM timer for the primary clock source.
   The system UUID ({default} in the example below) should be
   changed if the system UUID is different than the default boot
device.

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

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

Steal time accounting

Steal time is the amount of CPU time desired by a guest virtual machine
that is not provided by the host. Steal time occurs when the host allocates
these resources elsewhere: for example, to another guest.

Steal time is reported in the CPU time fields in /proc/stat as st. It is
automatically reported by utilities such as top and vmstat, and cannot
be switched off.
Large amounts of steal time indicate CPU contention, which can reduce guest performance. To relieve CPU contention, increase the guest’s CPU priority or CPU quota, or run fewer guests on the host.
Chapter 15. Network booting with libvirt

15.1. Preparing the boot server

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

15.2. Booting a guest using PXE

15.2.1. Using bridged networking

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

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

**Warning**

These procedures are provided only as an example. Ensure that you have sufficient backups before proceeding.

15.1. Preparing the boot server

To perform the steps in this chapter you will need:

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

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

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

Procedure 15.1. Configuring the PXE boot server

1. Place the PXE boot images and configuration in `/var/lib/tftp`.
2. Run the following commands:
   ```
   # virsh net-destroy default
   # virsh net-edit default
   ```
3. Edit the `<ip>` element in the configuration file for the default network to include the appropriate address, network mask, DHCP address range, and boot file, where `BOOT_FILENAME` represents the file name you are using to boot the guest virtual machine.
   ```
   <ip address='192.168.122.1' netmask='255.255.255.0'>
     <tftp root='/var/lib/tftp' />
   <dhcp>
     <range start='192.168.122.2' end='192.168.122.254' />
     <bootp file='BOOT_FILENAME' />
   </dhcp>
   </ip>
   ```
4. Boot the guest using PXE (refer to Section 15.2, "Booting a guest using PXE").

15.2. Booting a guest using PXE

This section demonstrates how to boot a guest virtual machine with PXE.

15.2.1. Using bridged networking

Procedure 15.2. Booting a guest using PXE and bridged networking

1. Ensure bridging is enabled such that the PXE boot server is available on the network.
2. Boot a guest virtual machine with PXE booting enabled. You can use the `virt-install` command to create a new virtual machine with PXE booting enabled, as shown in the following example command:
   ```
   virt-install --pxe --network bridge=breth0 --prompt
   ```
   Alternatively, ensure that the guest network is configured to use your bridged network, and that the XML guest configuration file has a `<boot dev='network'/>` element inside the `<os>` element, as shown in the following example:
15.2.2. Using a private libvirt network

Procedure 15.3. Using a private libvirt network

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

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

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

Alternatively, ensure that the guest network is configured to use your bridged network, and that the XML guest configuration file has a `<boot dev='network'/>` element inside the `<os>` element, as shown in the following example:

```
<os>
  <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
  <boot dev='network'/>  
  <boot dev='hd'/>   
</os>
```

Also ensure that the guest virtual machine is connected to the private network:

```
<interface type='network'>
  <mac address='52:54:80:66:79:14'/>
  <source network='default'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```
Chapter 16. QEMU Guest Agent

16.1. Set Up Communication between Guest Agent and Host

The QEMU Guest Agent allows the host machine to issue commands to the guest operating system. The guest operating system then responds to those commands asynchronously.

This section covers the options and commands available to the guest agent in detail. It also covers how to run the guest agent in the foreground, or as a daemon in the background.

16.1. Set Up Communication between Guest Agent and Host

The host machine communicates with the guest agent through a VirtIO serial connection between the host and guest machines. A VirtIO serial channel is connected to the host via a character device driver (typically a Unix socket), and the guest listens on this serial channel. The following procedure shows how to set up the host and guest machines for guest agent use.

Procedure 16.1. Set Up Host-Agent Communication

1. **Launch QEMU with a character device driver**
   
   Launch QEMU as usual, with additional definitions for the character device driver required to communicate with the guest agent.

   The following example launches QEMU to communicate over the Unix socket `/tmp/qga.sock`.

   ```bash
   /usr/libexec/qemu-kvm [...] -chardev socket,path=/tmp/qga.sock,server,nowait,id=qga0 
   -device virtio-serial 
   -device virtserialport,chardev=qga0,name=org.qemu.guest_agent.0
   ```

2. **Start the Guest Agent**

   On the guest, run the following command to start the Guest Agent:

   ```bash
   qemu-ga --path device_path --method method
   ```

   The guest agent now parses incoming QMP messages for commands, and acts upon them if valid.

   If no other method or path is specified with the `--method` or `--path` options respectively, the Guest Agent listens over virtio-serial, through the `/dev/virtio-ports/org.qemu.guest_agent.0` device path.

You can now communicate with the guest by sending valid QMP commands over the established character device driver.

Part II. Administration

Table of Contents

17. Server best practices
18. Security for virtualization
   18.1. Storage security issues
   18.2. SELinux and virtualization
   18.3. SELinux
   18.4. Virtualization firewall information
19. sVirt
   19.1. Security and Virtualization
   19.2. sVirt labeling
20. KVM live migration
   20.1. Live migration requirements
   20.2. Live migration and Fedora version compatibility
   20.3. Shared storage example: NFS for a simple migration
   20.4. Live KVM migration with virsh
      20.4.1. Additional tips for migration with virsh
      20.4.2. Additional options for the virsh migrate command
   20.5. Migrating with virt-manager
21. Remote management of guests
   21.1. Remote management with SSH
   21.2. Remote management over TLS and SSL
   21.3. Transport modes
22. Overcommitting with KVM

23. KSM

24. Advanced virtualization administration
   24.1. Control Groups (cgroups)
   24.2. Hugepage support

25. Miscellaneous administration tasks
   25.1. Automatically starting guests
   25.2. Guest memory allocation
   25.3. Using qemu-ling
   25.4. Verifying virtualization extensions
   25.5. Setting KVM processor affinities
   25.6. Generating a new unique MAC address
   25.7. Improving guest response time
   25.8. Disable SMART disk monitoring for guests
   25.9. Configuring a VNC Server
   25.10. Gracefully shutting down guests
   25.11. Virtual machine timer management with libvirt
   25.12. Using PMU to monitor guest performance
   25.13. Guest virtual machine power management
   25.14. QEMU Guest Agent Protocol
      25.14.1. guest-sync
      25.14.2. guest-sync-delimited
   25.15. Setting a limit on device redirection
   25.16. Dynamically changing a host or a network bridge that is attached to a virtual NIC

26. Storage concepts
   26.1. Storage pools
   26.2. Volumes

27. Storage pools
   27.1. Creating storage pools
      27.1.1. Disk-based storage pools
      27.1.2. Partition-based storage pools
      27.1.3. Directory-based storage pools
      27.1.4. LVM-based storage pools
      27.1.5. iSCSI-based storage pools
      27.1.6. NFS-based storage pools

28. Volumes
   28.1. Creating volumes
   28.2. Cloning volumes
   28.3. Adding storage devices to guests
      28.3.1. Adding file based storage to a guest
      28.3.2. Adding hard drives and other block devices to a guest
      28.3.3. Managing storage controllers in a guest
   28.4. Deleting and removing volumes

29. The Virtual Host Metrics Daemon (vhostmd)
   29.1. Installing vhostmd on the host
   29.2. Configuration of vhostmd
   29.3. Starting and stopping the daemon
   29.4. Verifying that vhostmd is working from the host
   29.5. Configuring guests to see the metrics
   29.6. Using vm-dump-metrics in Fedora guests to verify operation
Chapter 17. Server best practices

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

1. Run SELinux in enforcing mode. Set SELinux to run in enforcing mode with the `setenforce` command.

   ```bash
   # setenforce 1
   ```

2. Remove or disable any unnecessary services such as `AutoFS`, `NFS`, `FTP`, `HTTP`, `NIS`, `telnetd`, `sendmail` and so on.

3. Only add the minimum number of user accounts needed for platform management on the server and remove unnecessary user accounts.

4. 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.

5. 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 18. Security for virtualization

18.1. Storage security issues

18.2. SELinux and virtualization

18.3. SELinux

18.4. Virtualization firewall information

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

- Run only necessary services on hosts. The fewer processes and services running on the host, the higher the level of security and performance.
- Enable SELinux on the hypervisor. Read Section 18.2, "SELinux and virtualization" for more information on using SELinux and virtualization.
- Use a firewall to restrict traffic to the host. You can setup a firewall with default-reject rules that will help secure the host from attacks. It is also important to limit network-facing services.
- Do not allow normal users to access the host. The host is privileged, and granting access to unprivileged accounts may compromise the level of security.

18.1. Storage security issues

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

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

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

18.2. 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 Fedora systems should run with SELinux enabled and in enforcing mode.

Adding LVM based storage with SELinux in enforcing mode

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

Procedure 18.1. Creating and mounting a logical volume on a virtualized guest with SELinux enabled

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

   # lvcreate -n NewVolumeName -L 5G volumegroup

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

   # mke2fs -j /dev/volumegroup/NewVolumeName

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

   # mkdir /virtstorage

4. Mount the logical volume.

   # mount /dev/volumegroup/NewVolumeName/virtstorage

5. Set the correct SELinux type for the libvirt image location.
# semanage fcontext -a -t virt_image_t

"/virtstorage(/.*)?"

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

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

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

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

---

18.3. SELinux

This section contains topics to consider when using SELinux with your virtualization deployment. When you deploy system changes or add devices, you must update your SELinux policy accordingly. To configure an LVM volume for a guest, you must modify the SELinux context for the respective underlying block device and volume group. 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 -R /dev/sda2

KVM and SELinux

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

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

18.4. Virtualization firewall information

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

Note

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

- ICMP requests must be accepted. ICMP packets are used for network testing. You cannot ping guests if ICMP packets are blocked.
- Port 22 should be open for SSH access and the initial installation.
- Ports 80 or 443 (depending on the security settings on the RHEV Manager) are used by the vdsm-reg service to communicate information about the host.
- Ports 5634 to 6166 are used for guest console access with the SPICE
protocol.

3. 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.

3. Enabling IP forwarding (`net.ipv4.ip_forward = 1`) is also required for shared bridges and the default bridge. Note that installing libvirt enables this variable so it will be enabled when the virtualization packages are installed unless it was manually disabled.

```
Note

Note that enabling IP forwarding is not required for physical bridge devices. When a guest is connected through a physical bridge, traffic only operates at a level that does not require IP configuration such as IP forwarding.
```
Chapter 19. sVirt

19.1. Security and Virtualization

19.2. sVirt labeling

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

This chapter describes how sVirt integrates with virtualization technologies in Fedora.

Non-virtualized environments

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

Virtualized environments

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

19.1. Security and Virtualization

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

sVirt is an effort to isolate guests and limit their ability to launch further attacks if exploited. This is demonstrated in the following image, where an attack can not break out of the virtualized guest and extend to another guest instance:
SELinux introduces a pluggable security framework for virtualized instances in its implementation of Mandatory Access Control (MAC). The sVirt framework allows guests and their resources to be uniquely labeled. Once labeled, rules can be applied which can reject access between different guests.

19.2. sVirt labeling

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

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

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

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

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

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

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

It is also possible to perform static labeling when using sVirt. Static labels allow the administrator to select a specific label, including the MCS/MLS field, for a virtualized guest. Administrators who run statically-labeled virtualized guests are responsible for setting the correct label on the image files. The virtualized guest will always be started with that label, and the sVirt system will never modify the label of a statically-labeled virtual machine’s content. This allows the svirt component to run in an MLS environment. You can also run multiple virtualized guests with different sensitivity levels on a system, depending on your requirements.
Chapter 20. KVM live migration

20.1. Live migration requirements

20.2. Live migration and Fedora version compatibility

20.3. Shared storage example: NFS for a simple migration

20.4. Live KVM migration with virsh

20.4.1. Additional tips for migration with virsh

20.4.2. Additional options for the virsh migrate command

20.5. Migrating with virt-manager

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

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

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

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

Migrations can be performed live or not.

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

A migration that is not performed live, suspends the guest, then moves an image of the guest's memory to the destination host. The guest is then resumed on the destination host and the memory the guest used on the source host is freed. The time 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 modifies pages faster than KVM can transfer them to the destination host, offline migration must be used, as live migration would never complete.

20.1. Live migration requirements

Migrating guests requires the following:

Migration requirements

- A guest 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

- Both systems must have the appropriate TCP/IP ports open.

- A separate system exporting the shared storage medium. Storage should not reside on either of the two hosts being used for migration.

- Shared storage must mount at the same location on source and destination systems. The mounted directory 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`. For more information on prehooks, refer to [libvirt.org](http://libvirt.org), and for more information on the XML option, see the [virsh](https://www.redhat.com/en/docs/virsh) manual.

When migration is attempted on an existing guest in a public bridge+tap network, the source and destination hosts must be located in the same network. Otherwise, the guest network will not operate after migration.

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

Procedure 20.1. Configuring `libvirtd.conf`

1. Opening the `libvirtd.conf` requires running the command as root:
   ```
   # vim /etc/libvirt/libvirtd.conf
   ```
2. Change the parameters as needed and save the file.
3. Restart the `libvirtd` service:
   ```
   # service libvirtd restart
   ```

20.2. Live migration and Fedora version compatibility

Live Migration should only be performed in cases where the guest virtual machine and the host physical machine are running the same version of Fedora. Should you perform a live migration using different versions be forewarned that the migration may fail.

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 on the shared storage. Alternatively, use the NFS example in Section 20.3, "Shared storage example: NFS for a simple migration".

20.3. Shared storage example: NFS for a simple migration

### Important

This example uses NFS to share guest images with other KVM hosts. 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 guests. iSCSI storage is a better choice for large deployments. Refer to Section 27.1.5, "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
   a. Install the NFS packages if they are not yet installed:
      ```
      # yum install nfs
      ```
   b. Make sure that the ports for NFS in `iptables` (2049, for example) are opened and add NFS to the `/etc/hosts.allow` file.
   c. Start the NFS service:
# service nfs start

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

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

**Warning**

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

---

### 20.4. Live KVM migration with virsh

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

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

Note that the `--live` option may be eliminated when live migration is not desired. Additional options are listed in [Section 20.4.2, "Additional options for the virsh migrate command"](https://example.com/virsh/migrate).

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

The `DestinationURL` parameter is the connection URL of the destination host. The destination system must run the same version of Fedora, 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 as seen from the source guest.
- peer2peer migration: `DestinationURL` is the URL of the target host as seen from the source host.

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

**Important**

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

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

---

**Example: live migration with virsh**

This example migrates from `host1.example.com` to `host2.example.com`. Change the host names for your environment.

This example migrates a virtual machine named `guest1-F19`.

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

1. **Verify the guest is running**
   
   From the source system, `host1.example.com`, verify `guest1-F19` is running:

   ```
   [root@host1 ~]# virsh list
   Id Name State
   ------------------------------
   10 guest1-F19 running
   ```

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

   ```
   # virsh migrate --live guest1-F19
   qemu+ssh://host2.example.com/system
   ```

   Once the command is entered you will be prompted for the root password of the destination system.
3. **Wait**
   The migration may take some time depending on load and the size of the guest. `virsh` only reports errors. The guest continues to run on the source host until fully migrated.

4. **Verify the guest has arrived at the destination host**
   From the destination system, `host2.example.com`, verify `guest1-F19` is running:
   ```
   [root@host2 ~]# virsh list
   Id  Name                 State
   ----------------------------------
   10  guest1-F19     running
   ```
   The live migration is now complete.

**Note**
libvirtd supports a variety of networking methods including TLS/SSL, UNIX sockets, SSH, and unencrypted TCP. Refer to [Chapter 21, Remote management of guests](#) for more information on using other methods.

**Note**
Non-running guests cannot be migrated with the `virsh migrate` command. To migrate a non-running guest, the following script should be used:
```
virsh dumpxml Guest1 > Guest1.xml
virsh -c qemu+ssh://<target-system-FQDN> define Guest1.xml
virsh undefine Guest1
```

### 20.4.1. 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 20.1, Configuring libvirtd.conf`.

1. Open the `libvirtd.conf` file as described in `Procedure 20.1, Configuring libvirtd.conf`.
2. Look for the Processing controls section.
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.

```
# 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, up to 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 up to 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
```

**Important**

The `max_clients` and `max_workers` parameters settings are effected by all guest connections to the libvirtd service. This means that any user that is using the same guest 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.

20.4.2. 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
- `--tunnelled` - used for tunnelled migration
- `--persistent` - leaves the domain persistent on destination host
- `--undefinesource` - undefines the domain on the source host
- `--suspend` - leaves the domain paused on the destination host
- `--copy-storage-all` - indicates migration with non-shared storage with full disk copy
- `--copy-storage-inc` - 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, the --copy-storage- option only tell libvirt to transfer data from the images on source host to the images found at the same place on the destination host.

- --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 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.

- dname - is used for renaming the domain to new name during migration, which also usually can be omitted.

- --xml file 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.

### 20.5. Migrating with virt-manager

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

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

   ![Figure 20.1. Virt-Manager main menu](image)

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

   ![Figure 20.2. Open Add Connection window](image)

3. **Add connection**
   
   The **Add Connection** window appears.
Figure 20.3. Adding a connection to the target host

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

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

Figure 20.4. Enter password

4. **Migrate guest**

Right-click on the host to be migrated (*guest1-F19* in this example) and click **Migrate**.

Select the host you wish to migrate to and click **Migrate**.
5. View the storage details for the host
   In the Edit menu, click Connection Details, the Connection Details window appears.
   Click the Storage tab. The iSCSI target details for this host is shown.
This host was defined by the following XML configuration:

```xml
<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>
```
Chapter 21. Remote management of guests

21.1. Remote management with SSH

21.2. Remote management over TLS and SSL

21.3. Transport modes

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

21.1. Remote management with SSH

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

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

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

Note

Fedora enables remote management of large numbers of virtual machines. Refer to the oVirt 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).
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 Fedora 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.

![Add connection menu](image)

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 Username and Hostname, then click Connect.

21.2. Remote management over TLS and SSL

You can manage virtual machines using TLS and SSL. TLS and SSL provides greater scalability but is more complicated than ssh (refer to Section 21.1, “Remote management with SSH”). TLS and SSL is the same technology used by web browsers for secure connections. The libvirt management connection opens a TCP port for incoming connections, which is securely encrypted and authenticated based on x509 certificates. The procedures that follow provide instructions on creating and deploying authentication certificates for TLS and SSL management.

Procedure 21.1. Creating a certificate authority (CA) key for TLS management

1. Before you begin, confirm that `certtool` is installed. If not:
1. Run the command:
   ```
   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 `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 21.2. 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:
   ```
   vim server.info
   ```
   ```
   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 21.3. 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:
This process is quite similar to Procedure 21.2, “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 = Fedora
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
```

### 21.3. Transport modes

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

**Transport Layer Security (TLS)**

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

**UNIX sockets**

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

**SSH**

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

**ext**

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

**TCP**

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

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

**Remote URIs**

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

Libvirt URIs take the general form (content in square brackets, “[” represents optional functions):

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

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

**Examples of remote management parameters**

1. Connect to a remote KVM host named `host2`, using SSH transport and the SSH username `virtuser`.
   
   ```
   qemu+ssh://virtuser@host2/
   ```

2. Connect to a remote KVM hypervisor on the host named `host2` using TLS.
   
   ```
   qemu:///host2/
   ```

**Testing examples**

3. 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
   ```

4. 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 cover 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 21.1. Extra URI parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Transport mode</th>
<th>Description</th>
<th>Example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>all modes</td>
<td>The name passed to the <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.</td>
<td>name=qemu:///system</td>
</tr>
<tr>
<td>command</td>
<td>ssh and ext</td>
<td>The external command. For ext transport this is required. For ssh the default is <code>ssh</code>. The PATH is searched for the command.</td>
<td>command=/opt/openssh/bin/ssh</td>
</tr>
<tr>
<td>socket</td>
<td>unix and ssh</td>
<td>The path to the UNIX domain socket, which overrides the default. For ssh transport, this is passed to the remote <code>netcat</code> command (see <code>netcat</code>).</td>
<td>socket=/opt/libvirtunix/libvirt/libvirt-sock</td>
</tr>
<tr>
<td>netcat</td>
<td>ssh</td>
<td>The <code>netcat</code> command can be used to connect to remote systems. The default <code>netcat</code> parameter uses the <code>nc</code> command. For SSH transport, <code>libvirt</code> constructs an SSH command using the form below:</td>
<td>netcat=/opt/netcat/bin/nc</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>command -p port [-l username] hostname</code> <code>netcat -U socket</code></td>
<td></td>
</tr>
<tr>
<td>no_verify</td>
<td>ssh</td>
<td>If set to a non-zero value, this disables client checks of the server's certificate. Note that to disable server checks of the client's certificate or IP address you</td>
<td>no_verify=1</td>
</tr>
</tbody>
</table>
must change the `libvirtd` configuration.

<table>
<thead>
<tr>
<th>no_tty</th>
<th>ssh</th>
<th>if set to a non-zero value, this stops ssh from asking for a password if it cannot log in to the remote machine automatically (for using ssh-agent or similar). Use this when you do not have access to a terminal - for example in graphical programs which use libvirt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_tty=1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 22. Overcommitting with KVM

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

Overcommitting memory

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

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

Warning

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

Configuring swap for overcommitting memory

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

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

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

Important

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

Example 22.1. Memory overcommit example

ExampleServer1 has 32GB of physical RAM. The system is being configured to run 56 guests, each with 1GB of virtualized memory. The host system itself needs a maximum of 3GB (apart from the guests).

The total maximum memory consumption is 56GB + 3GB = 59GB. The system's physical RAM is 32GB, which leaves 27GB. Therefore, the minimum amount of swap that the host should have configured is 27GB.

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

For more information on KSM and overcommitting, refer to Chapter 23, KSM.

Overcommitting virtualized CPUs

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

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

You cannot overcommit symmetric multiprocessing guests on more than the physical number of processing cores. For example a guest with four VCPUs should not be run on a host with a dual core processor. Overcommitting symmetric multiprocessing guests in over the physical number of processing cores will cause significant performance
Assigning guests VCPUs up to the number of physical cores is appropriate and works as expected. For example, running guests with four VCPUs on a quad core host. Guests with less than 100% loads should function effectively in this setup.

**Important**

Do not overcommit memory or CPUs in a production environment without extensive testing. Applications which use 100% of memory or processing resources may become unstable in overcommitted environments. Test before deploying.
Chapter 23. KSM

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

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

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

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

Starting in Fedora 18, KSM is NUMA aware

Fedora 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 Fedora. When using Fedora as a KVM host, 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 system's main memory. Start the ksm service to enable KSM to share more memory.

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

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

```
# chkconfig ksm on
```

The KSM tuning service

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

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

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

The /etc/ksmtuned.conf file is the configuration file for the ksmtuned service. The file output below is the default ksmtuned.conf file.
# Configuration file for ksmtuned.
# How long ksmtuned should sleep between tuning
# adjustments
# KSM_MONITOR_INTERVAL=60
# Millisecond sleep between ksm scans for 16Gb server.
# Smaller servers sleep more, bigger sleep less.
# KSM_SLEEP_MSEC=30
# KSM_NPAGES_BOOST=300
# KSM_NPAGES_DECAY=-50
# KSM_NPAGES_MIN=64
# KSM_NPAGES_MAX=1250
# KSM_THRES_COEF=28
# KSM_THRES_CONST=2048
# uncomment the following to enable ksmtuned debug
# information
# LOGFILE=/var/log/ksmtuned
# DEBUG=1

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

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

The /sys/kernel/mm/ksm/ files

full_scans
Full scans run.

pages_shared
Total pages shared.

pages_sharing
Pages presently shared.

pages_to_scan
Pages not scanned.

pages_unshared
Pages no longer shared.

pages_volatile
Number of volatile pages.

run
Whether the KSM process is running.

sleep_millisecs
Sleep milliseconds.

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

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

KSM can be deactivated by stopping the 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
Chapter 24. Advanced virtualization administration

24.1. Control Groups (cgroups)

Fedora 19 provides a new kernel feature: control groups, which are often referred to as cgroups. Cgroups allow you to allocate resources such as CPU time, system memory, network bandwidth, or combinations of these resources among user-defined groups of tasks (processes) running on a system. You can monitor the cgroups you configure, deny cgroups access to certain resources, and even reconfigure your cgroups dynamically on a running system.

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

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

24.2. Hugepage support

Introduction

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

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

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

Using Transparent Hugepage Support

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

Transparent Hugepage Support does not prevent the use of hugetlbfs. However, when hugetlbfs is not used, KVM will use transparent hugepages instead of the regular 4kB page size.
Chapter 25. Miscellaneous administration tasks

25.1. Automatically starting guests
25.2. Guest memory allocation
25.3. Using qemu-img
25.4. Verifying virtualization extensions
25.5. Setting KVM processor affinities
25.6. Generating a new unique MAC address
25.7. Improving guest response time
25.8. Disable SMART disk monitoring for guests
25.9. Configuring a VNC Server
25.10. Gracefully shutting down guests
25.11. Virtual machine timer management with libvirt
25.12. Using PMU to monitor guest performance
25.13. Guest virtual machine power management
25.14. QEMU Guest Agent Protocol
  25.14.1. guest-sync
  25.14.2. guest-sync-delimited
25.15. Setting a limit on device redirection
25.16. Dynamically changing a host or a network bridge that is attached to a virtual NIC

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

25.1. Automatically starting guests

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

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

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

The guest now automatically starts with the host.

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

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

The guest no longer automatically starts with the host.

25.2. Guest memory allocation

The following procedure shows how to allocate memory for a guest. 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.

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)
- `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)
- `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)
- `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 x $2^{10}$ 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 crashes the optional attribute `dumpCore` can be used to control whether the guest'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 memory will be included in the coredump file.

The currentMemory attribute determines the actual memory allocation for a guest. This value can be less than the maximum allocation, to allow for ballooning up the guest’s 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's memory to be included in the generated coredump file -->
  <currentMemory unit='KiB'>524288</currentMemory>
  <!-- makes the current memory unit 524288 KiB -->
...</domain>
```

25.3. 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:

```
```

The `-p` parameter shows the progress of the command (optional and not for every command) and `-S` indicates the consecutive number of bytes that must contain only zeros for `qemu-img` to create a sparse image during conversion.

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

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

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

**Create**

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

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

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

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

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

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

Rebase

Changes the backing file of an image.

```
```

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`.

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

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

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

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

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

Resize

Change the disk image `filename` as if it had been created with size `size`. Only images in raw format can be resized regardless of version. Fedora 17 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 or host 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.

wdi
Oracle VM VirtualBox hard disk image format.

vmdk
VMware 3 and 4 compatible image format.

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

vvfat
Virtual VFAT disk image format.

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

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

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

   ```
   flags : fpu tsc msr pae mce cx8 apic mtrr mca cmov pat pse36 ccf
   ds cpl nx xsave osxsave pdxst mxr sf wrmsr crm cmp_legacy svm cmov
   est tm tsc modrm sf xtpr lahf_lm
   ```

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

   ```
   flags : fpu tsc msr pae mce cx8 apic mtrr mca cmov pat pse36 ccf
   mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt lm 3nowext 3now np1 cx8
   lahf_lm cmp_legacy svm cr8legacy ts fid vtd 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 A.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.

### 25.5. 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 guests.

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

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

#### Identifying CPU and NUMA topology

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

```
# virsh nodetree
CPU model: x86_64
CPU(s): 8
CPU frequency: 1880 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.

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

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 with four virtual CPUs, it is optimal to lock the guest to physical CPUs 0 to 3, or 4 to 7, to avoid accessing non-local memory, which is significantly slower than accessing local memory.

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

**Decide which NUMA node can run the guest**

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

```
# virsh freecell --all
0: 2283620 kB
1: 3354784 kB
```

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

**Lock a guest to a NUMA node or physical CPU set**

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

1. Extract from the `virsh capabilities` output.
2. Observe that the node 1, `<cell id='1'>`, uses physical CPUs 4 to 7.

3. The guest can be locked to a set of CPUs by appending the `cpuset` attribute to the configuration file.
   a. While the guest is offline, open the configuration file with `virsh edit`.
   b. Locate the guest’s virtual CPU count, defined in the `vcpus` element.

   ```xml
   <vcpus>4</vcpus>
   ``
   The guest in this example has four CPUs.
   c. Add a `cpuset` attribute with the CPU numbers for the relevant NUMA cell.

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

4. Save the configuration file and restart the guest.

The guest has been locked to CPUs 4 to 7.

Automatically locking guests to CPUs with `virt-install`

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

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

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

Tuning CPU affinity on running guests

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

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

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

```bash
# 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: 3
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.

```bash
# 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
```

25.6. Generating a new unique MAC address

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

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

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

Another method to generate a new MAC for your guest

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

```
# echo 'import virtinst.util; print\nvirtinst.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 """New UUID:"""
import virtinst.util ; print virtinst.util.uuidToString(virtinst.util.randomUUID())
print "New MAC:" ; import virtinst.util ; print virtinst.util.randomMAC()
print ""
```

25.7. Improving guest response time

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

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

These types of workload may cause guests to appear slow or unresponsive. Usually, the guest's memory is eventually fully loaded into the host's main memory from swap. Once the guest is loaded in main memory, the guest will perform normally. Note, the process of loading a guest from swap to main memory may take several seconds per gigabyte of RAM assigned to the guest, depending on the type of storage used for swap and the performance of the components.
KVM guests function as Linux processes. Linux processes are not permanently kept in main memory (physical RAM). The kernel scheduler swaps process memory into virtual memory (swap). Swap, with conventional hard disk drives, is thousands of times slower than main memory in modern computers. If a guest is inactive for long periods of time, the guest may be placed into swap by the kernel.

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

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

For more information on overcommitting with KVM, refer to Chapter 22, Overcommitting with KVM.

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 guests. Overcommitting guests without any swap can cause guests or the host system to crash.

Turning off swap

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

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

```
# swapoff -a
```

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

Using SSDs for swap

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

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

25.8. Disable SMART disk monitoring for guests

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

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

25.9. 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 Fedora Installation Guide.

25.10. Gracefully shutting down guests

Installing virtualized Fedora 19 guests with the Minimal installation option will not install the `acpid` package.

Without the `acpid` package, the Fedora guest does not shut down when the `virsh shutdown` command is executed. The `virsh shutdown` command is designed to gracefully shut down guests.

Using `virsh shutdown` is easier and safer for system administration. Without graceful shut down with the `virsh shutdown` command a system administrator must log into a guest manually or send the `Ctrl-Alt-Del` key combination to each guest.
Other virtualized operating systems may be affected by this issue. The `virsh shutdown` command requires that the guest operating system is configured to handle ACPI shut down requests. Many operating systems require additional configuration on the guest operating system to accept ACPI shut down requests.

Procedure 25.1. Workaround for Fedora

1. **Install the acpid package**
   The `acpid` service listens and processes ACPI requests.
   Log into the guest and install the `acpid` package on the guest:
   ```
   # yum install acpid
   ```

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

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

### 25.11. Virtual machine timer management with libvirt

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

#### <clock>

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

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

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

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

3 adjustment

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

Example 25.1. Always synchronize to UTC

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

Example 25.2. Always synchronize to the host timezone

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

Example 25.3. Synchronize to an arbitrary timezone

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

Example 25.4. Synchronize to UTC + arbitrary offset

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

`<timer>`

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

3 name

The name of the time source to use.

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

3 track

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot</td>
<td>Corresponds to old host option, this is an unsupported tracking option.</td>
</tr>
<tr>
<td>guest</td>
<td>RTC always tracks guest time.</td>
</tr>
<tr>
<td>wall</td>
<td>RTC always tracks host time.</td>
</tr>
</tbody>
</table>

3 tickpolicy

The policy used to pass ticks on to the guest.
Table 25.4. tickpolicy attribute values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay</td>
<td>Continue to deliver at normal rate (i.e. ticks are delayed).</td>
</tr>
<tr>
<td>catchup</td>
<td>Deliver at a higher rate to catch up.</td>
</tr>
<tr>
<td>merge</td>
<td>Ticks merged into one single tick.</td>
</tr>
<tr>
<td>discard</td>
<td>All missed ticks are discarded.</td>
</tr>
</tbody>
</table>

3. frequency

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

3. mode

Determines how the time source is exposed to the guest. This attribute is only relevant for a name value of tsc. All other timers are always emulated. Command is as follows: `<timer name='tsc' frequency='NNN' mode='auto|native|emulate|smpsafe'/>`. Mode definitions are given in the table.

Table 25.5. mode attribute values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Native if TSC is unstable, otherwise allow native TSC access.</td>
</tr>
<tr>
<td>native</td>
<td>Always allow native TSC access.</td>
</tr>
<tr>
<td>emulate</td>
<td>Always emulate TSC.</td>
</tr>
<tr>
<td>smpsafe</td>
<td>Always emulate TSC and interlock SMP</td>
</tr>
</tbody>
</table>

3. present

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

Table 25.6. present attribute values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Force this timer to be visible to the guest.</td>
</tr>
<tr>
<td>no</td>
<td>Force this timer to not be visible to the guest.</td>
</tr>
</tbody>
</table>

Example 25.5. Clock synchronizing to local time with RTC and PIT timers, and the HPET timer disabled

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

25.12. Using PMU to monitor guest performance

In Fedora 18 and onward, 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 guests running Fedora 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.

### 25.13. 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:

```xml
...<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.

### 25.14. QEMU Guest Agent Protocol

The QEMU guest agent protocol (QEMU-ga), uses the same protocol as QMP, `qemu-ga`. There are a couple issues regarding its isa-serial/virtio-serial transport, and the following caveats have been noted:

1. There is no way for `qemu-ga` to detect whether or not a client has connected to the channel.
2. There is no way for a client to detect whether or not `qemu-ga` has disconnected or reconnected to the backend.
3. If the virtio-serial device resets and `qemu-ga` has not connected to the channel as a result, (generally caused by a reboot or hotplug), data from the client will be dropped.
4. If `qemu-ga` 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-ga` is still running/connected.

`qemu-ga` uses the guest-sync or guest-sync-delimited command to address the problem of re-synchronizing the channel after re-connection or client-side timeouts. These are described below.

#### 25.14.1. 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-ga`’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-ga` 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-ga`’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-ga` 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-ga` wiki page on [wiki.qemu.org](http://wiki.qemu.org).

#### 25.14.2. guest-sync-delimited

If `qemu-ga` attempts to communicate with a client, and the client receives a partial response from a previous `qemu-ga` 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-ga` 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.
25.15. 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 25.6. An example of limiting redirection with a windows guest virtual machine

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

   ```xml
   <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 and confirm the setting changes by running the following:

   ```bash
   ps -ef | grep $guest_name
   -device usb-redir,chardev=charredir0,id=redir0,
   /filter=0x08:0x1234:0xBEEF:0x0200:1|-1:-1:-1:-1:0,bus=usb.0,port=3
   ```
4. Plug a USB device into host, and use virt-viewer to connect to the guest.
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.
6. To make sure that the filter captures properly check the USB device vendor and product, then make the following changes in the host's domain XML to allow for USB redirection.

   ```xml
   <redirfilter>
     <usbdev class='0x08' vendor='0x0951' product='0x1625' version='2.0' allow='yes'/>
     <usbdev allow='no'/>
   </redirfilter>
   ```
7. Restart the guest, then use virt-viewer to connect to the guest. The USB device will now redirect traffic to the guest.

25.16. Dynamically changing a host or a network bridge that is attached to a virtual NIC

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

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

   ```xml
   <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, confirm the guest’s network functionality, and check that the guest’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     vnet0
```

4. Update the guest’s network with the new interface parameters with the following command:

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

5. On the guest, run `service network restart`. The guest gets a new IP address for virbr1. Check the guest’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 26. Storage concepts

26.1. Storage pools

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.

26.1. Storage pools

A storage pool is a file, directory, or storage device managed by libvirt for the purpose of providing storage to guests. The storage pool can be local or it can be shared over a network.

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 server. Local storage pools include: local directories, directly attached disks, physical partitions, and LVM volume groups. These storage volumes store guest images or are attached to guests as additional storage. As local storage pools are directly attached to the host server, they are useful for development, testing and small deployments that do not require migration or large numbers of guests. Local storage pools are not suitable for many production environments as local storage pools do not support live migration.

Networked (shared) storage pools - Networked storage pools include storage devices shared over a network using standard protocols. Networked storage is required when migrating virtual machines between hosts 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.

26.2. Volumes

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

Referencing volumes

To reference a specific volume, three approaches are possible:

The name of the volume and the storage pool

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

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

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

virsh #
```

The full path to the storage on the host system

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

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

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

```

The unique volume key

When a volume is first created in the virtualization system, a unique identifier is generated and assigned to it. The unique identifier is termed the volume key. The format of this volume key varies upon the
storage used.
When used with block based storage such as LVM, the volume key
may follow this format:

```
c3pKz4-qPVc-X7FM-7W9W-Wjc8-qSiz-mtv9Gm
```
When used with file based storage, the volume key may instead be a
copy of the full path to the volume storage:

```
 TwistImage.secondimage.img
```
For example, a volume with the volume key of

```
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```

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

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

**vol-name**

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

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

**vol-path**

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

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

**The vol-key command**

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

```
# virsh vol-key /dev/guest_images/firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
# virsh vol-key --pool guest_images firstimage
Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr
```
Chapter 27. Storage pools

27.1. Creating storage pools

27.1.1. Disk-based storage pools

27.1.2. Partition-based storage pools

27.1.3. Directory-based storage pools

27.1.4. LVM-based storage pools

27.1.5. iSCSI-based storage pools

27.1.6. NFS-based 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 27.1. 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 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 /vmdata`. 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's fstab so that the share is mounted at boot time.

27.1. Creating storage pools

27.1.1. 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 to detect these partitions or LVM groups and cause errors.

27.1.1.1. 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:
   
   ```xml
   <name>guest_images_disk</name>
   <device path='/dev/sdb'/>
   <target><path>/dev</path></target>
   <format type='gpt'/>
   ```
   
   The `name` parameter determines the name of the storage pool. This example uses the name `guest_images_disk` in the example below.
   
   The `device` parameter with the `path` attribute specifies the device path of the storage device. This example uses the device `/dev/sdb`.
   
   The `target` parameter with the `path` sub-parameter determines the location on the host file system to attach volumes created with this storage pool. For example, `sdb1`, `sdb2`, `sdb3`. Using `/dev/`, as in the example below, means volumes created from this storage pool can be accessed as `/dev/sdb1`, `/dev/sdb2`, `/dev/sdb3`.
   
   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 27.2. Disk based storage device storage pool**
   
   ```xml
   <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.
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
```

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.

### 27.1.1.2. Deleting a storage pool using virsh

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

1. To avoid any issues with other 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. Remove the storage pool's definition

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

### 27.1.2. Partition-based storage pools

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

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

#### 27.1.2.1. Creating a partition-based storage pool using virt-manager

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

Procedure 27.1. Creating a partition-based storage pool with virt-manager

1. **Open the storage pool settings**

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

      *Open the Edit menu and select Connection Details*

      ![Figure 27.1. Connection Details](image)

   b. Click on the **Storage** tab of the **Connection Details** window.
2. Create the new storage pool
   a. Add a new pool (part 1)
      Press the + button (the add pool button). The
      Add a New Storage Pool wizard appears.
      Choose a Name for the storage pool. This example uses
      the name guest_images_fs. Change the Type to fs: Pre-Formatted Block Device.

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

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

      Format
      Select a format from the Format list. The device is
      formatted with the selected format.
      This example uses the ext4 file system, the default
      Fedora 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.

![Storage list confirmation](image)

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

27.1.2.2. Deleting a storage pool using virt-manager
This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other 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.

![Stop Icon](image)

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

27.1.2.3. 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 27.2. 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.

```bash
# 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.

```bash
# 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.

```bash
# virsh pool-build guest_images_fs
Pool guest_images_fs built
```

```bash
# ls -la /guest_images
total 8
drwx------.  2 root root 4096 May 31 19:38 .
```

```bash
dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

```bash
# 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.

```bash
# virsh pool-start guest_images_fs
Pool guest_images_fs started
```

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

5. **Turn on autostart**

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

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

```bash
# 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.
27.1.2.4. Deleting a storage pool using virsh

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.

   ```
   # 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
   ```

27.1.3. Directory-based storage pools

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

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

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

1. Create the local directory

   a. Optional: Create a new directory for the storage pool
   
   Create the directory on the host for the storage pool. This example uses a directory named `/guest_images`.

   ```
   # mkdir /guest_images
   ```

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

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

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

   ```
   # chmod 700 /guest_images
   ```

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

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

2. Configure SELinux file contexts

   Configure the correct SELinux context for the new directory. 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 labelling 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.

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

3. Open the storage pool settings

   a. In the `virt-manager` graphical interface, select the host from the main window.
   
   Open the `Edit` menu and select `Connection Details`
b. Click on the Storage tab of the Connection Details window.

4. Create the new storage pool
   a. Add a new pool (part1)
      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.

      Press the Forward button to continue.
   b. Add a new pool (part2)
      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 27.10. Verify the storage pool information

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

27.1.3.2. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other 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.

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

27.1.3.3. 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:
   
   1. The **name** of the storage pool.
      
      This example uses the name `guest_images`. All further `virsh` commands used in this example use this name.
   2. 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:
27.1.3.4. Deleting a storage pool using virsh

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

1. To avoid any issues with other 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

27.1.4. 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

Please refer to the Fedora 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.

---

### 27.1.4.1. 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.
- **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 (1-409, 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
```

**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**
3. Create the new storage pool
   a. Start the Wizard
      Press the + button (the add pool button). The Add a New Storage Pool wizard appears.
      Choose a Name for the storage pool. We use guest_images_lvm for this example. Then change the Type to logical: LVM Volume Group, and
      Press the Forward button to continue.
   b. Add a new pool (part 2)
      Change the Target Path field. This example uses /guest_images.
      Now fill in the Target Path and Source Path fields, then tick the Build Pool check box.
      Use the Target Path field to either select an existing LVM volume group or as the name for a new volume group. The default format is /dev/storage_pool_name. This example uses a new volume group named /dev/guest_images_lvm.
      The Source Path field is optional if an existing LVM volume group is used in the Target Path.
      For new LVM volume groups, input the location of a storage device in the Source Path field. This example uses a blank partition /dev/sdc.
      The Build Pool checkbox instructs virt-manager to create a new LVM volume group. If you are using an existing volume group you should not select the Build Pool checkbox. This example is using a blank partition to create a new volume group so the Build Pool checkbox must be selected.
Verify the details and press the Finish button format the LVM volume group and create the storage pool.

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

![Warning message](image)

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.

![Confirm LVM storage pool details](image)

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

### 27.1.4.2. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other 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.
2. Delete the storage pool by clicking the Trash can icon. This icon is only enabled if you stop the storage pool first.

27.1.4.3. 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 27.3. Creating an LVM-based storage pool with virsh**

1. Define the pool name `guest_images_lvm`.
   
   ```bash
   # 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.
   
   ```bash
   # virsh pool-build guest_images_lvm
   Pool guest_images_lvm built
   ```

3. Initialize the new pool.
   
   ```bash
   # virsh pool-start guest_images_lvm
   Pool guest_images_lvm started
   ```

4. Show the volume group information with the `vgs` command.
   
   ```bash
   # 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.
   
   ```bash
   # virsh pool-autostart guest_images_lvm
   Pool guest_images_lvm marked as autostarted
   ```

6. List the available pools with the `virsh` command.
   
   ```bash
   # 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.
   
   ```bash
   # 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.
   
   ```bash
   # 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.
27.1.4. Deleting a storage pool using virsh

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

1. To avoid any issues with other 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
   ```

27.1.5. iSCSI-based storage pools

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

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

27.1.5.1. Configuring a software iSCSI target

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

Procedure 27.4. Creating an iSCSI target

1. Install the required packages

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

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

2. Start the `tgtd` service

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

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

3. Optional: Create LVM volumes

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

   a. Create the RAID array

      Creating software RAID5 arrays is covered by the Fedora 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 image named `virtimage2.img` for an iSCSI target.

   a. Create a new directory for the image

   ```
   # mkdir virtimage2
   ```
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:

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

Where:

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

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

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

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

**Important**

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

6. **Restart the tgtd service**

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

```
# service tgtd restart
```

7. **iptables configuration**

Open port 3260 for iSCSI access with `iptables`.

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

8. **Verify the new targets**

View the new targets to ensure the setup was successful with the `tgt-admin --show` command.
# tgt-admin --show
Target 1: iqn.2010-05.com.example.server1:trial
System information:
Driver: iscsi
State: ready
I_T nexus information:
LUN information:
LUN: 0
  Type: controller
  SCSI ID: IET 00810000
  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 00810001
  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 00810002
  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 access ACLs for production environments.

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

```
# iscsiadm --mode discovery --type sendtargets --portal server1.example.com
```

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
```

Detaching the device.

```
# iscsiadm -d2 -m node --logout
```

An iSCSI device is now ready to use for virtualization.

### 27.1.5.2. Adding an iSCSI target to virt-manager
This procedure covers creating a storage pool with an iSCSI target in virt-manager.

#### Procedure 27.5. Adding an iSCSI device to virt-manager

1. **Open the host storage tab**
   Open the Storage tab in the Host Details window.
   a. Open virt-manager.
   b. Select a host from the main virt-manager window. Click Edit menu and select Connection Details.
c. Click on the Storage tab.

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

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

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

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

   Enter the source path, for the iSCSI target. This example uses demo-target.

   Check the IQN checkbox to enter the IQN. This example uses iqn.2010-05.com.example.server1:iscsirhel6guest.
Figure 27.22. Create an iSCSI storage pool

Press Finish to create the new storage pool.

27.1.5.3. Deleting a storage pool using virt-manager

This procedure demonstrates how to delete a storage pool.

1. To avoid any issues with other 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.

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

27.1.5.4. 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

The example below creates the same iSCSI-based storage pool as the previous step.
```bash
# virsh pool-define-as --name iscsi --
type iscsi
--source-host server1.example.com
--source-dev iqn.2010-05.com.example.server1:iscsi
--target /dev/disk/by-path
Pool iscsi defined
```

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

```bash
# virsh pool-list --all
Name                State      Autostart
------------------- -------- ------------
default             active     yes
iscsi               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 guests.

```bash
# virsh pool-start guest_images_disk
Pool guest_images_disk started
```

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

```bash
# virsh pool-autostart iscsi
Pool iscsi marked as autostarted
```

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

```bash
# virsh pool-info iscsi
Name: iscsi
UUID: afcc5367-6770-e151-bcb3-8478c36c5e28
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.

### 27.1.5.5. Deleting a storage pool using virsh

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

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

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

2. Remove the storage pool’s definition

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

### 27.1.6. NFS-based storage pools

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

#### 27.1.6.1. Creating a NFS-based storage pool with `virt-manager`

1. Open the host storage tab
Open the Storage tab in the Host Details window.
   a. Open `virt-manager`.
   b. Select a host from the main `virt-manager` window. Click Edit menu and select Connection Details.
Figure 27.24. Connection details

c. Click on the Storage tab.

Figure 27.25. 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 27.26. 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 hosts for migration. Enter the hostname or IP address of the NFS server. This example uses server1.example.com.

Enter the NFS path. This example uses /nfstrial.

Figure 27.27. Create an NFS storage pool

Press Finish to create the new storage pool.
27.1.6.2. 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.

![Stop Icon](image)

2. Delete the storage pool by clicking the Trash can icon. This icon is only enabled if you stop the storage pool first.
Chapter 28. Volumes

28.1. 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)
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
```

28.2. Cloning volumes

The new volume will be allocated from storage in the same storage pool as the volume being cloned. The `virsh vol-clone` command with 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)
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Number  Start   End     Size    File system  Name  Flags
----------  ------  -------  ------  --------------  -------
2  8590MB  17.2GB  8590MB  primary
3  8590MB  17.2GB  8590MB  primary
1  21.5GB  30.1GB  8590MB  primary
4  30.0GB  38.6GB  8590MB  primary
```

28.3. Adding storage devices to guests

This section covers adding storage devices to a guest. Additional storage can only be added as needed.

28.3.1. Adding file based storage to a guest

File-based storage is a collection of files that are stored on the host's file system.
system that act as virtualized hard drives for guests. To add file-based storage, perform the following steps:

Procedure 28.1. Adding file-based storage

1. Create a storage file or use an existing file (such as an ISO 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:

   ```bash
   # dd if=/dev/zero of=/var/lib/libvirt/images/FileName.iso 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.

   ```bash
   # dd if=/dev/zero of=/var/lib/libvirt/images/FileName.iso 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`:

   ```xml
   <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:

   ```xml
   <disk type='file' device='cdrom'>
       <driver name='qemu' type='raw' cache='none'/>
       <source file='/var/lib/libvirt/images/FileName.iso'/>
       <readonly/>
       <target dev='hdc'/>
   </disk>
   ```

4. Add the device defined in `NewStorage.xml` with your guest (Guest1):

   ```bash
   # 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:

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

```bash
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.iso 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):

```bash
# virsh start Guest1
```

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.

   a. Start `fdisk` for the new device:

   ```bash
   # fdisk /dev/vdb
   Command (m for help):
   
   Type `n` for a new partition.
   
   c. The following appears:

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

   Type `p` for a primary partition.

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

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

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

   ```bash
   First cylinder (1-409, default 1):
   ```

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

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

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

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

   h. Enter `83` for a Linux partition.

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

   j. Enter `w` to write changes and quit.

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

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

   ```bash
   # 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`.

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

Procedure 28.2. Adding physical block devices to guests

1. This procedure describes how to add a hard drive on the host to a guest. It applies to all physical block devices, including CD-ROM, DVD and floppy devices. Physically attach the hard disk device to the host. Configure the host 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 partition /dev/sr0:

   ```xml
   <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. Alternatively, you can use the virsh attach-disk command, as shown:

   ```bash
   # 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:

   ```bash
   # 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 now has a new hard disk device called /dev/vdc on Linux (or something similar, depending on what the guest OS chooses) or D: drive (for example) on Windows. You can now initialize the disk from the guest, following the standard procedures for the guest’s operating system. Refer to Procedure 28.1, “Adding file-based storage” and Step 6 for an example.

**Warning**

The host 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 guests, have write access to whole partitions or LVM volumes, because a guest could potentially write a filesystem label belonging to the host, to its own block device storage. Upon reboot of the host, the host could then mistakenly use the guest's disk as a system disk, which would compromise the host system.

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

**Important**

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

28.3.3. Managing storage controllers in a guest

Starting from Red Hat Enterprise Linux 6.3, SCSI devices are also supported inside guests.

Unlike virtio disks, SCSI devices require the presence of a controller in the guest.
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.

**Procedure 28.3. Creating a virtual SCSI controller**

1. Display the configuration of the guest (**Guest1**) and look for a pre-existing SCSI controller:
   ```bash
   # virsh dumpxml Guest1 | grep controller.*scsi
   ```
   If a controller is present, the command will output one or more lines similar to the following:
   ```xml
   <controller type='scsi' model='virtio-scsi' index='0'/>
   ```

2. If the previous step did not show a controller, create the description for one in a new file and add it to the virtual machine, using the following steps:
   a. Create the controller by writing a `<controller>` element in a new file and save this file with an XML extension. **NewHBA.xml**, for example.
   ```xml
   <controller type='scsi' model='virtio-scsi'/>
   ```
   b. Associate the device in the **NewHBA.xml** you just created with your guest:
   ```bash
   # virsh attach-device --config Guest1 ~/${}/NewHBA.xml
   ```
   In this example the `--config` option behaves the same as it does for disks. Refer to Procedure 28.2, "Adding physical block devices to guests" for more information.

3. Add a new SCSI disk or CD-ROM. The new disk can be added using the methods in sections Section 28.3.1, "Adding file based storage to a guest" and Section 28.3.2, "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`.
   ```bash
   # virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.iso sdb --cache none
   ```
   Depending on the version of the driver in the guest, the new disk may not be detected immediately by a running guest. Follow the steps in the Red Hat Enterprise Linux Storage Administration Guide.

**28.4. 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**.

```bash
# virsh vol-delete --pool guest_images volume1
Vol volume1 deleted
```
Chapter 29. The Virtual Host Metrics Daemon (vhostmd)

29.1. Installing vhostmd on the host

29.2. Configuration of vhostmd

29.3. Starting and stopping the daemon

29.4. Verifying that vhostmd is working from the host

29.5. Configuring guests to see the metrics

29.6. Using vm-dump-metrics in Fedora guests to verify operation

vhostmd (the Virtual Host Metrics Daemon) allows virtual machines to see limited information about the host they are running on. In the host, a daemon (vhostmd) runs which writes metrics periodically into a disk image. This disk image is exported read-only to guests. Guests can read the disk image to see metrics. Simple synchronization stops guests from seeing out of date or corrupt metrics.

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

29.1. Installing vhostmd on the host

The vhostmd package is available from RHN and is located in the Downloads area. It must be installed on each host where guests are required to get host metrics.

29.2. Configuration of vhostmd

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

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

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

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

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

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

```
# man vhostmd
```

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

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

29.3. Starting and stopping the daemon

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

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

To start the daemon running, do:

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

To stop the daemon running, do:

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

To disable the daemon from being started at boot, do:
# /sbin/chkconfig vhostmd off

## 29.4. Verifying that vhostmd is working from the host

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

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

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

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

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

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

## 29.5. Configuring guests to see the metrics

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

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

### Important

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

**Procedure 29.1. Configuring KVM guests**

1. Shut down the guest.
2. Do:

   ```bash
   # virsh edit GuestName
   and add the following section into `<devices>`:
   ```

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

3. Reboot the guest.

**Procedure 29.2. Configuring Xen guests**

1. Shut down the guest.
2. Do:

   ```bash
   # virsh edit GuestName
   and add the following section into `<devices>`:
   ```

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

3. Reboot the guest.

## 29.6. Using vm-dump-metrics in Fedora guests to verify operation

Optionally, the `vm-dump-metrics` package from the RHN Downloads area may be installed in Fedora guests. This package provides a simple command line tool (also called `vm-dump-metrics`) which allows host metrics to be displayed in the guest.

This is useful for verifying correct operation of `vhostmd` from a guest.

In the guest, run the following command as root:
# vm-dump-metrics

If everything is working, this should print out a long XML document starting with `metrics`.

If this does not work, then verify that the metrics disk has appeared in the guest. It should appear as `/dev/vd*` (for example, `/dev/vdb`, `/dev/vdd`).

On the host, verify that the libvirt configuration changes have been made by using the command:

```
# virsh dumpxml GuestName
```

Verify that vhostmd is running on the host and the `/dev/shm/vhostmd0` file exists.

## Part III. Appendicies

### Part III. Appendicies

**Table of Contents**

A. Troubleshooting

A.1. Debugging and troubleshooting tools
A.2. kvm_stat
A.3. Troubleshooting with serial consoles
A.4. Virtualization log files
A.5. Loop device errors
A.6. Live Migration Errors
A.7. Enabling Intel VT-x and AMD-V virtualization hardware extensions in BIOS
A.8. KVM networking performance
A.9. Missing characters on guest console with Japanese keyboard
A.10. Known Windows XP guest issues

B. Common libvirt errors and troubleshooting

B.1. libvirtd failed to start
B.2. The URI failed to connect to the hypervisor
   B.2.1. Cannot read CA certificate
   B.2.2. Failed to connect socket ... : Permission denied
   B.2.3. Other connectivity errors
B.3. The guest virtual machine cannot be started: **internal error**
   guest CPU is not compatible with host CPU
B.4. Guest starting fails with error: **monitor socket did not show up**
B.5. **Internal error** cannot find character device (null)
B.6. Guest virtual machine booting stalls with error: No boot device
B.7. Virtual network default has not been started
B.8. PXE boot (or DHCP) on guest failed
B.9. Guest can reach outside network, but cannot reach host when using macvtap interface
B.10. Could not add rule to fixup DHCP response checksums on network ‘default’
B.11. Unable to add bridge br0 port vnet0: No such device
B.12. Guest is unable to start with error: warning: could not open
   /dev/net/tun
B.13. Migration fails with **Error**: unable to resolve address
B.14. Migration fails with Unable to allow access for disk path: No such file or directory
B.15. No guest virtual machines are present when libvirtd is started
B.16. Unable to connect to server at host:16509: Connection refused
B.17. Common XML errors
   B.17.1. Editing domain definition
   B.17.2. XML syntax errors
   B.17.3. Logic and configuration errors

C. NetKVM Driver Parameters

C.1. Configurable parameters for NetKVM

D. qemu-kvm Whitelist

D.1. Introduction
D.2. Basic options
D.3. Disk options
D.4. Display options
D.5. Network options
D.6. Device options
D.7. Linux/Multiboot boot
D.8. Expert options
D.9. Help and information options
D.10. Miscellaneous options

E. Managing guests with virsh
   E.1. virsh command quick reference
   E.2. Attaching and updating a device with virsh
   E.3. Connecting to the hypervisor
   E.4. Creating a virtual machine XML dump (configuration file)
      E.4.1. Adding multifunction PCI devices to KVM guests
   E.5. Suspending, resuming, saving and restoring a guest
   E.6. Shutting down, rebooting and force-shutdown of a guest
   E.7. Retrieving guest information
   E.8. Retrieving node information
   E.9. Storage pool information
   E.10. Displaying per-guest information
   E.11. Managing virtual networks
   E.12. Migrating guests with virsh
   E.13. Disk image management with live block copy
      E.13.1. Using blockcommit to shorten a backing chain
      E.13.2. Using blockpull to shorten a backing chain
      E.13.3. Using blockresize to change the size of a domain path
   E.14. Guest CPU model configuration
      E.14.1. Introduction
      E.14.2. Learning about the host CPU model
      E.14.3. Determining a compatible CPU model to suit a pool of hosts
      E.14.4. Configuring the guest CPU model

F. Managing guests with the Virtual Machine Manager (virt-manager)
   F.1. Starting virt-manager
   F.2. The Virtual Machine Manager main window
   F.3. The virtual hardware details window
   F.4. Virtual Machine graphical console
   F.5. Adding a remote connection
   F.6. Displaying guest details
   F.7. Performance monitoring
   F.8. Displaying CPU usage for guests
   F.9. Displaying CPU usage for hosts
   F.10. Displaying Disk I/O
   F.11. Displaying Network I/O

G. Guest disk access with offline tools
   G.1. Introduction
   G.2. Terminology
   G.3. Installation
   G.4. The guestfish shell
      G.4.1. Viewing file systems with guestfish
      G.4.2. Modifying files with guestfish
      G.4.3. Other actions with guestfish
      G.4.4. Shell scripting with guestfish
      G.4.5. Augeas and libguestfs scripting
   G.5. Other commands
   G.6. virt-rescue: The rescue shell
      G.6.1. Introduction
      G.6.2. Running virt-rescue
   G.7. virt-df: Monitoring disk usage
      G.7.1. Introduction
      G.7.2. Running virt-df
   G.8. virt-resize: resizing guests offline
      G.8.1. Introduction
      G.8.2. Expanding a disk image
   G.9. virt-inspector: inspecting guests
      G.9.1. Introduction
      G.9.2. Installation
      G.9.3. Running virt-inspector
   G.10. virt-win-reg: Reading and editing the Windows Registry
      G.10.1. Introduction
      G.10.2. Installation
      G.10.3. Using virt-win-reg
   G.11. Using the API from Programming Languages
      G.11.1. Interaction with the API via a C program
   G.12. Troubleshooting
G.13. Where to find further documentation

H. Virtual Networking
  H.1. Virtual network switches
  H.2. Network Address Translation
  H.3. Networking protocols
    H.3.1. DNS and DHCP
    H.3.2. Routed mode
    H.3.3. Isolated mode
  H.4. The default configuration
  H.5. Examples of common scenarios
    H.5.1. Routed mode
    H.5.2. NAT mode
    H.5.3. Isolated mode
  H.6. Managing a virtual network
  H.7. Creating a virtual network
  H.8. Attaching a virtual network to a guest
  H.9. Directly attaching to physical interface
  H.10. Applying network filtering
    H.10.1. Introduction
    H.10.2. Filtering chains
    H.10.3. Filtering chain priorities
    H.10.4. Usage of variables in filters
    H.10.5. Automatic IP address detection and DHCP snooping
    H.10.6. Reserved Variables
    H.10.7. Element and attribute overview
    H.10.8. References to other filters
    H.10.9. Filter rules
    H.10.10. Supported protocols
    H.10.11. Advanced Filter Configuration Topics
    H.10.12. Limitations

I. Additional resources
  I.1. Online resources
  I.2. Installed documentation

J. Manipulating the domain xml
  J.1. General information and metadata
  J.2. Operating system booting
    J.2.1. BIOS bootloader
    J.2.2. Host bootloader
    J.2.3. Direct kernel boot
    J.2.4. Container boot
  J.3. SMBIOS system information
  J.4. CPU allocation
  J.5. CPU tuning
  J.6. Memory backing
  J.7. Memory tuning
  J.8. NUMA node tuning
  J.9. Block iO tuning
  J.10. Resource partitioning
  J.11. CPU model and topology
    J.11.1. Guest NUMA topology
  J.12. Events configuration
  J.13. Power Management
  J.14. Hypervisor features
  J.15. Time keeping
  J.16. Devices
    J.16.1. Hard drives, floppy disks, CDROMs
    J.16.2. Filesystems
    J.16.3. Device addresses
    J.16.4. Controllers
    J.16.5. Device leases
    J.16.6. Host device assignment
    J.16.7. Redirected devices
    J.16.8. Smartcard devices
    J.16.9. Network interfaces
    J.16.10. Input devices
    J.16.11. Hub devices
    J.16.12. Graphical framebuffers
    J.16.13. Video devices
    J.16.15. Guest interfaces
    J.16.16. Channel
    J.16.17. Host interface
  J.17. Sound devices
Troubleshooting

This chapter covers common problems and solutions for Fedora 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 Fedora 19 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 I.1, “Online resources” for a list of Linux virtualization websites.

A.1. Debugging and troubleshooting tools

This section summarizes the System Administrator applications, the networking utilities, and debugging tools. You can employ these standard System administration tools and logs to assist with troubleshooting:

- `kvm_stat`
- `trace-cmd`
- `vmstat`
- `iostat`
- `lsipf`
- `systat`
- `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:
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** connects to a VNC server running on your server or a virtual machine. Install vncviewer using the `yum install vnc` command.
- **vncserver** starts 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.

### A.2. kvm_stat

The **kvm_stat** command is a python script which retrieves runtime statistics from the **kvm** kernel module. The **kvm_stat** command can be used to diagnose guest behavior visible to **kvm**. In particular, performance related issues with guests. Currently, the reported statistics are for the entire system; the behavior of all running guests is reported.

The **kvm_stat** command requires that the **kvm** kernel module is loaded and **debugfs** is mounted. If either of these features are not enabled, the command will output the required steps to enable **debugfs** or the **kvm** module. For example:

```
# kvm_stat
Please mount debugfs ('mount -t debugfs /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).
<table>
<thead>
<tr>
<th>Variable</th>
<th>Count</th>
<th>guest Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>efer_reload</td>
<td>94</td>
<td>0</td>
</tr>
<tr>
<td>exits</td>
<td>4083674</td>
<td>31272</td>
</tr>
<tr>
<td>fpu_reload</td>
<td>1313881</td>
<td>10796</td>
</tr>
<tr>
<td>halt_exits</td>
<td>14050</td>
<td>259</td>
</tr>
<tr>
<td>halt_wakeup</td>
<td>4496</td>
<td>203</td>
</tr>
<tr>
<td>host_state_reload</td>
<td>1638354</td>
<td>24893</td>
</tr>
<tr>
<td>hypercalls</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>insn_emulation</td>
<td>1093858</td>
<td>1089</td>
</tr>
<tr>
<td>insn_emulation_fail</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>invlpg</td>
<td>75569</td>
<td>0</td>
</tr>
<tr>
<td>io_exits</td>
<td>1596984</td>
<td>24509</td>
</tr>
<tr>
<td>irq_exits</td>
<td>21013</td>
<td>363</td>
</tr>
<tr>
<td>irq_injections</td>
<td>48039</td>
<td>1222</td>
</tr>
<tr>
<td>irq_window</td>
<td>24656</td>
<td>878</td>
</tr>
<tr>
<td>largepages</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mmio_exits</td>
<td>11873</td>
<td>0</td>
</tr>
<tr>
<td>mmu_cache_miss</td>
<td>42365</td>
<td>0</td>
</tr>
<tr>
<td>mmu_flooded</td>
<td>14792</td>
<td>0</td>
</tr>
<tr>
<td>mmu_pde_zapped</td>
<td>58730</td>
<td>0</td>
</tr>
<tr>
<td>mmu_pte_updated</td>
<td>138795</td>
<td>0</td>
</tr>
<tr>
<td>mmu_pte_write</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mmu_recycled</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mmu_shadow_zapped</td>
<td>40358</td>
<td>0</td>
</tr>
<tr>
<td>mmu_unsync</td>
<td>793</td>
<td>0</td>
</tr>
<tr>
<td>nmi_injections</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nmi_window</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pf_fixed</td>
<td>697731</td>
<td>3159</td>
</tr>
<tr>
<td>pf_guest</td>
<td>279340</td>
<td>0</td>
</tr>
<tr>
<td>remote_tlb_flush</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>request_irq</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>signal_exits</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tlb_flush</td>
<td>208190</td>
<td>0</td>
</tr>
</tbody>
</table>

**Explanation of variables:**

**efer_reload**
The number of Extended Feature Enable Register (EFER) reloads.

**exits**
The count of all VMEXIT calls.

**fpu_reload**
The number of times a VMENTRY reloaded the FPU state. The fpu_reload is incremented when a guest is using the Floating Point Unit (FPU).

**halt_exits**
Number of guest exits due to halt calls. This type of exit is usually seen when a guest is idle.

**halt_wakeup**
Number of wakeups from a halt.

**host_state_reload**
Count of full reloads of the host state (currently tallies MSR setup and guest MSR reads).

**hypercalls**
Number of guest hypervisor service calls.

**insn_emulation**
Number of guest instructions emulated by the host.

**insn_emulation_fail**
Number of failed insn_emulation attempts.

**io_exits**
Number of guest exits from I/O port accesses.

**irq_exits**
Number of guest exits due to external interrupts.

**irq_injections**
Number of interrupts sent to guests.

**irq_window**
Number of guest exits from an outstanding interrupt window.

**largepages**
Number of large pages currently in use.
**mmio_exits**
Number of guest exits due to memory mapped I/O (MMIO) accesses.

**mmu_cache_miss**
Number of KVM MMU shadow pages created.

**mmu_flooded**
Detection count of excessive write operations to an MMU page. This counts detected write operations not of individual write operations.

**mmu_pde_zapped**
Number of page directory entry (PDE) destruction operations.

**mmu_pte_updated**
Number of page table entry (PTE) destruction operations.

**mmu_pte_write**
Number of guest page table entry (PTE) write operations.

**mmu_recycled**
Number of shadow pages that can be reclaimed.

**mmu_shadow_zapped**
Number of invalidated shadow pages.

**mmu_unsync**
Number of non-synchronized pages which are not yet unlinked.

**nmi_injections**
Number of Non-maskable Interrupt (NMI) injections to the guest.

**nmi_window**
Number of guest exits from (outstanding) Non-maskable Interrupt (NMI) windows.

**pf_fixed**
Number of fixed (non-paging) page table entry (PTE) maps.

**pf_guest**
Number of page faults injected into guests.

**remote_tlb_flush**
Number of remote (sibling CPU) Translation Lookaside Buffer (TLB) flush requests.

**request_irq**
Number of guest interrupt window request exits.

**signal_exits**
Number of guest exits due to pending signals from the host.

**tlb_flush**
Number of tlb_flush operations performed by the hypervisor.

---

**Note**
The output information from the `kvm_stat` command is exported by the KVM hypervisor as pseudo files located in the `/sys/kernel/debug/kvm/` directory.

---

**A.3. Troubleshooting with serial consoles**

Linux kernels can output information to serial ports. This is useful for debugging kernel panics and hardware issues with video devices or headless servers. The subsections in this section cover setting up serial console output for machines running Fedora virtualization kernels and their guests. This section covers how to enable serial console output for fully virtualized guests.
Fully virtualized guest serial console output can be viewed with the **virsh console** command.

Be aware fully virtualized guest serial consoles have some limitations. Present limitations include:

- output data may be dropped or scrambled.

The serial port is called **ttyS0** on Linux or **COM1** on Windows.

You must configure the virtualized operating system to output information to the virtual serial port.

To output kernel information from a fully virtualized Linux guest into the domain, modify the `/boot/grub/grub.conf` file. Append the following to the `kernel` line: `console=tty0 console=ttyS0,115200`.

```
title Fedora 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.

### A.4. Virtualization log files

Each fully virtualized guest log is in the `/var/log/libvirt/qemu/` directory. Each guest log is named as **GuestName.log** and will be periodically compressed once a size limit is reached.

If you encounter any errors with the Virtual Machine Manager, you can review the generated data in the **virt-manager.log** file that resides in the `$HOME/.virt-manager` directory.

### A.5. Loop device errors

If file-based guest images are used you may have to increase the number of configured loop devices. The default configuration allows up to eight active loop devices. If more than eight file-based guests or loop devices are needed the number of loop devices configured can be adjusted in the `/etc/modprobe.d/` directory. Add the following line:

```
options loop max_loop=64
```

This example uses 64 but you can specify another number to set the maximum loop value. You may also have to implement loop device backed guests on your system. To use a loop device backed guests for a full virtualized system, use the **phy: device** or **file: file** commands.

### A.6. Live Migration Errors

There may be cases where a live migration causes the memory contents to be re-transfered 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 scheduled to be fixed in Fedora 19 or 20.

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.

### A.7. 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 A.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.

**A.8. KVM networking performance**

By default, KVM virtual machines are assigned a virtual Realtek 8139 (rtl8139) NIC (network interface controller) if they are Windows guests or the guest type is not specified. Fedora guests are assigned a virtio NIC by default.

The rtl8139 virtualized NIC works fine in most environments. However, this device can suffer from performance degradation problems on some networks, for example, a 10 Gigabit Ethernet network.

To improve performance switch to the para-virtualized network driver.

- **Note**

  Note that the virtualized Intel PRO/1000 (**e1000**) driver is also supported as an emulated driver choice. To use the e1000 driver, replace **virtio** in the procedure below with **e1000**. For the best performance it is recommended to use the **virtio** driver.

**Procedure A.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:
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'>
    <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'>
    <model type='virtio' />
</interface>
```

3. Create the new virtual machine:
```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

A.9. Missing characters on guest console with Japanese keyboard

On a Fedora 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.

When Using Fedora with 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:

```
atkbd.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:

1. Open the affected guest in `virt-manager`.
2. Click `View` → `Details`.
3. Select `Display VNC` in the list.
4. Change `Auto` to `ja` in the `Keymap` pull-down menu.
5. Click the `Apply` button.

Alternatively, to fix this issue using the `virsh edit` command on the target guest:

1. Run `virsh edit <target guest>`
2. Add the following attribute to the `<graphics>` tag: `keymap='ja'`. For example:
```
<graphics type='vnc' port='-1' autoport='yes' keymap='ja'/>
```
A.10. 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.

Common libvirt errors and troubleshooting

This appendix documents common libvirt-related problems and errors along with instructions for dealing with them.

Locate the error on the table below and follow the corresponding link under Solution for detailed troubleshooting information.
### Table B.1. Common libvirt errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description of problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>libvirtd failed to start</strong></td>
<td>The libvirtd daemon failed to start. However, there is no information about this error in <code>/var/log/messages</code>.</td>
<td>Section B.1, &quot;libvirtd failed to start&quot;</td>
</tr>
<tr>
<td><strong>Cannot read CA certificate</strong></td>
<td>This is one of several errors that occur when the URI fails to connect to the hypervisor.</td>
<td>Section B.2, &quot;The URI failed to connect to the hypervisor&quot;</td>
</tr>
<tr>
<td><strong>Failed to connect socket ...: Permission denied</strong></td>
<td>This is one of several errors that occur when the URI fails to connect to the hypervisor.</td>
<td>Section B.2, &quot;The URI failed to connect to the hypervisor&quot;</td>
</tr>
<tr>
<td><strong>Other connectivity errors</strong></td>
<td>These are other errors that occur when the URI fails to connect to the hypervisor.</td>
<td>Section B.2, &quot;The URI failed to connect to the hypervisor&quot;</td>
</tr>
<tr>
<td><strong>Internal error guest CPU is not compatible with host CPU</strong></td>
<td>The guest virtual machine cannot be started because the host and guest processors are different.</td>
<td>Section B.3, &quot;The guest virtual machine cannot be started. Internal error guest CPU is not compatible with host CPU&quot;</td>
</tr>
<tr>
<td><strong>Failed to create domain from vm.xml error: monitor socket did not show up.: Connection refused</strong></td>
<td>The guest virtual machine (or domain) starting fails and returns this error or similar.</td>
<td>Section B.4, &quot;Guest starting fails with error: monitor socket did not show up. Connection refused&quot;</td>
</tr>
<tr>
<td><strong>Internal error cannot find character device (null)</strong></td>
<td>This error can occur when attempting to connect a guest's console. It reports that there is no serial console configured for the guest virtual machine.</td>
<td>Section B.5, &quot;Internal error cannot find character device (null)&quot;</td>
</tr>
<tr>
<td><strong>No boot device</strong></td>
<td>After building a guest virtual machine from an existing disk image, the guest booting stalls. However, the guest can start successfully using the QEMU command directly.</td>
<td>Section B.6, &quot;Guest virtual machine booting stalls with error: No boot device&quot;</td>
</tr>
<tr>
<td><strong>The virtual network &quot;default&quot; has not been started</strong></td>
<td>If the default network (or other locally-created network) is unable to start, any virtual machine configured to use that network for its connectivity will also fail to start.</td>
<td>Section B.7, &quot;Virtual network default has not been started&quot;</td>
</tr>
<tr>
<td><strong>PXE boot (or DHCP) on guest failed</strong></td>
<td>A guest virtual machine starts successfully, but is unable to acquire an IP address from DHCP. Boot using the PXE protocol, or both. This is often a result of a long forward delay time set for the bridge, or when the ipables package and kernel do not support checksum mangling rules.</td>
<td>Section B.8, &quot;PXE boot (or DHCP) on guest failed&quot;</td>
</tr>
<tr>
<td><strong>Guest can reach outside network, but cannot reach host when using macvtap interface</strong></td>
<td>A guest can communicate with other guests, but cannot connect to the host machine after being configured to use a macvtap (or type='direct') network interface. This is actually not an error — it is the defined behavior of macvtap.</td>
<td>Section B.9, &quot;Guest can reach outside network, but cannot reach host when using macvtap&quot;</td>
</tr>
<tr>
<td><strong>Could not add rule to fixup DHCP response checksums on network 'default'</strong></td>
<td>This warning message is almost always harmless, but is often mistakenly seen as evidence of a problem.</td>
<td>Section B.10, &quot;Could not add rule to fixup DHCP response checksums on network 'default'&quot;</td>
</tr>
<tr>
<td><strong>Unable to add bridge br0 port vnet0: No such device</strong></td>
<td>This error message or the similar Failed to add tap interface to bridge 'br0': No such device reveal that the bridge device specified in the guest's (or domain's) <code>&lt;interface&gt;</code> definition does not exist.</td>
<td>Section B.11, &quot;Failed to add bridge br0 port vnet0: No such device&quot;</td>
</tr>
<tr>
<td><strong>Warning: could not open /dev/net/tun: no virtual network emulation qemu-kvm: -netdev tap,script=/etc/my-qemu-emulation qemu-kvm: -netdev /dev/net/tun: no virtual network emulation</strong></td>
<td>The guest virtual machine does not start after configuring a type='ethernet' (or generic ethernet) interface in the host system. This error or similar appears either in <code>libvirtd.log</code>, <code>/var/log/libvirt/qemu/name_of_guest.log</code>, or in both.</td>
<td>Section B.12, &quot;Guest is unable to start with error warning: could not open /dev/net/tun&quot;</td>
</tr>
<tr>
<td><strong>Unable to resolve address name_of_host service '49155': Name or service not known</strong></td>
<td>QEMU guest migration fails and this error message appears with an unfamiliar hostname.</td>
<td>Section B.13, &quot;Migration fails with Error: unable to resolve address name_of_host service '49155': Name or service not known&quot;</td>
</tr>
<tr>
<td>Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or directory</td>
<td>A guest virtual machine cannot be migrated because libvirt cannot access the disk image(s).</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>No guest virtual machines are present when libvirtd is started</td>
<td>The libvirt daemon is successfully started, but no guest virtual machines appear to be present when running virsh list --all.</td>
<td></td>
</tr>
<tr>
<td>Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor</td>
<td>While libvirt should listen on TCP ports for connections, the connection to the hypervisor fails.</td>
<td></td>
</tr>
<tr>
<td>Common XML errors</td>
<td>libvirt uses XML documents to store structured data. Several common errors occur with XML documents when they are passed to libvirt through the API. This entry provides instructions for editing guest XML definitions, and details common errors in XML syntax and configuration.</td>
<td></td>
</tr>
</tbody>
</table>

### B.1. libvirtd failed to start

**Symptom**

The libvirt daemon does not start automatically. Starting the libvirt daemon manually fails as well:

```bash
# /etc/init.d/libvirtd start
* Caching service dependencies ... [ ok ]
* Starting libvirtd ... /usr/sbin/libvirtd: error: Unable to initialize network sockets. Check /var/log/messages or run without --daemon for more info.
* start-stop-daemon: failed to start '/usr/sbin/libvirtd'
  [ !! ]
* ERROR: libvirtd failed to start
```

Moreover, there is not 'more info' about this error in /var/log/messages.

**Investigation**

Change libvirt's logging in `/etc/libvirt/libvirtd.conf` by uncommenting the line below. To uncomment the line, open the `/etc/libvirt/libvirtd.conf` file in a text editor, remove the hash (or #) symbol from the beginning of the following line, and save the change:

```bash
log_outputs="3:syslog:libvirtd"
```

This line is commented out by default to prevent libvirt from producing excessive log messages. After diagnosing the problem, it is recommended to comment this line again in the `/etc/libvirt/libvirtd.conf` file.

**Note**

Restart libvirt to determine if this has solved the problem.

```bash
# libvirtd still does not start successfully, an error similar to the following will be shown in the /var/log/messages file:
Feb 6 17:22:09 bart libvirtd: 17576: info : libvirt version: 0.9.9
Feb 6 17:22:09 bart libvirtd: 17576: error : virNetTLSContextCheckCertFile(92): cannot read CA certificate '/etc/pki/CA/cacert.pem': No such file or directory
Feb 6 17:22:09 bart /etc/init.d/libvirtd[17573]: start-stop-daemon: failed to start '/usr/sbin/libvirtd'
Feb 6 17:22:09 bart /etc/init.d/libvirtd[17565]: ERROR: libvirtd failed to start
```

The libvirt man page shows that the missing cacert.pem file is used as TLS authority when libvirt is run in Listen for TCP/IP.
connections mode. This means the \texttt{--listen} parameter is being passed.

\textbf{Solution}

Configure the \texttt{libvirt} daemon’s settings with one of the following methods:

\begin{itemize}
  \item Install a CA certificate.
  \item Do not use TLS; use bare TCP instead. In \texttt{/etc/libvirt/libvirtd.conf} set \texttt{listen_tls = 0} and \texttt{listen_tcp = 1}. The default values are \texttt{listen_tls = 1} and \texttt{listen_tcp = 0}.
  \item Do not pass the \texttt{--listen} parameter. In \texttt{/etc/sysconfig/libvirtd.conf} change the \texttt{LIBVIRTD_ARGS} variable.
\end{itemize}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image}
\caption{Diagram of a network connection setting}
\end{figure}

B.2. The URI failed to connect to the hypervisor

Several different errors can occur when connecting to the server (for example, when running \texttt{virsh}).

B.2.1. Cannot read CA certificate

\textbf{Symptom}

When running a command, the following error (or similar) appears:

\begin{verbatim}
$ virsh -c name_of_uri list
error: Cannot read CA certificate
'/etc/pki/CA/cacert.pem': No such file or directory
error: failed to connect to the hypervisor
\end{verbatim}

\textbf{Investigation}

The error message is misleading about the actual cause. This error can be caused by a variety of factors, such as an incorrectly specified URI, or a connection that is not configured.

\textbf{Solution}

\begin{itemize}
  \item \textbf{Incorrectly specified URI}
    
    When specifying \texttt{qemu://system} or \texttt{qemu://session} as a connection URI, \texttt{virsh} attempts to connect to hostnames \texttt{system} or \texttt{session} respectively. This is because \texttt{virsh} recognizes the text after the second forward slash as the host.
    
    Use three forward slashes to connect to the local host. For example, specifying \texttt{qemu:///system} instructs \texttt{virsh} connect to the \texttt{system} instance of \texttt{libvirtd} on the local host.
    
    When a hostname is specified, the \texttt{QEMU} transport defaults to \texttt{TLS}. This results in certificates.
  \item \textbf{Connection is not configured}
    
    The URI is correct (for example, \texttt{qemu[+tls]://server/system}) but the certificates are not set up properly on your machine. For information on configuring \texttt{TLS}, see Setting up \texttt{libvirt for TLS} available from the \texttt{libvirt} website.
\end{itemize}

B.2.2. Failed to connect socket ... : Permission denied

\textbf{Symptom}

When running a \texttt{virsh} command, the following error (or similar) appears:

\begin{verbatim}
$ virsh -c qemu:///system list
error: Failed to connect socket to
'/var/run/libvirtd/libvirtd.sock': Permission denied
error: failed to connect to the hypervisor
\end{verbatim}
Investigation
Without any hostname specified, the connection to QEMU uses UNIX sockets by default. If there is no error running this command as root, the UNIX socket options in `/etc/libvirt/libvirtd.conf` are likely misconfigured.

Solution
To connect as a non-root user using UNIX sockets, configure the following options in `/etc/libvirt/libvirtd.conf`:

```
unix_sock_group = <group>
unix_sock_rw_perms = <perms>
unix_sock_ro_perms = <perms>
```

Note
The user running `virsh` must be a member of the `group` specified in the `unix_sock_group` option.

B.2.3. Other connectivity errors

Unable to connect to server at server:port: Connection refused
The daemon is not running on the server or is configured not to listen, using configuration option `listen_tcp` or `listen_tls`.

End of file while reading data: nc: using stream socket: Input/output error
If you specified `ssh` transport, the daemon is likely not running on the server. Solve this error by verifying that the daemon is running on the server.

B.3. The guest virtual machine cannot be started: internal error guest CPU is not compatible with host CPU

Symptom
Running on an Intel Core i7 processor (which `virt-manager` refers to as `Nehalem`, or the older Core 2 Duo, referred to as `Penryn`), a KVM guest (or domain) is created using `virt-manager`. After installation, the guest's processor is changed to match the host's CPU. The guest is then unable to start and reports this error:

```
2012-02-06 17:49:15.985+0000: 20757: error : qemuBuildCpuArgStr:3565 : internal error guest CPU is not compatible with host CPU
```

Additionally, clicking `Copy host CPU configuration` in `virt-manager` shows Pentium III instead of `Nehalem` or `Penryn`.

Investigation
The `/usr/share/libvirt/cpu_map.xml` file lists the flags that define each CPU model. The `Nehalem` and `Penryn` definitions contain this:

```xml
<feature name='nx'/>
```

As a result, the `NX` (or `No eXecute`) flag needs to be presented to identify the CPU as `Nehalem` or `Penryn`. However, in `/proc/cpuinfo`, this flag is missing.

Solution
Nearly all new BIOSes allow enabling or disabling of the `No eXecute` bit. However, if disabled, some CPUs do not report this flag and thus `libvirt` detects a different CPU. Enabling this functionality instructs `libvirt` to report the correct CPU. Refer to your hardware documentation for further instructions on this subject.
B.4. Guest starting fails with error: monitor socket did not show up

**Symptom**
The guest virtual machine (or domain) starting fails with this error (or similar):

```
# virsh -c qemu:///system create name_of_guest.xml
error: Failed to create domain from name_of_guest.xml error: monitor socket did not show up.: Connection refused
```

**Investigation**
This error message shows:

1. `libvirt` is working;
2. The QEMU process failed to start up; and
3. `libvirt` quits when trying to connect QEMU or the QEMU agent monitor socket.

To understand the error details, examine the guest log:

```
# cat /var/log/libvirt/qemu/name_of_guest.log
LC_ALL=C PATH=/sbin:/usr/sbin:/bin:/usr/bin
QEMU_AUDIO_DRV=none /usr/bin/qemu-kvm -S -M pc -enable-kvm -m 768 -smp 1,sockets=1,cores=1,threads=1 -name name_of_guest -uuid ebfaadbe-e908-ba92-fdb8-3fa2d8575a42 -nodefaults -chardev=socket,id=monitor,path=/var/lib/libvirt/qemu/name_of_guest.monitor,server,nowait -mon chardev=monitor,path=/var/lib/libvirt/qemu/name_of_guest.monitor,server,nowait -mon chardev=monitor,mode=readline -no-reboot -boot c -kernel /var/lib/libvirt/boot/vmlinuz -initrd /var/lib/libvirt/boot/initrd.img -append method=http://www.example.com/pub/product/release/version/x86_64/os/ -drive file=/var/lib/libvirt/images/name_of_guest.img,if=n,one,id=drive-ide0-0-0,boot=on -device ide-drive,bus=ide.0,unit=0,drive=drive-ide0-0-0-id=drive-ide0-0-0-0 -drive virtio-net-pci,vlan=9,id=net0,mac=52:48:08:ff:0a:ba,bus=pci.0,addr=0x4 -device virtio-net-pci,vlan=9,id=net0,mac=52:48:08:ff:0a:ba,bus=pci.0,addr=0x4 -device virtio-net-pci,vlan=9,id=net0,mac=52:48:08:ff:0a:ba,bus=pci.0,addr=0x4 -device virtio-net-pci,vlan=9,id=net0,mac=52:48:08:ff:0a:ba,bus=pci.0,addr=0x4 -device virtio-net-pci,vlan=9,id=net0,mac=52:48:08:ff:0a:ba,bus=pci.0,addr=0x4
device redirected to /dev/pts/1
qemu: could not load kernel
'/var/lib/libvirt/boot/vmlinuz': Permission denied
```

**Solution**
The guest log contains the details needed to fix the error.

If a host is shut down while the guest is still running a `libvirt` version prior to 0.9.5, the `libvirt`-guest's init script attempts to perform a managed save of the guest. If the managed save was incomplete (for example, due to loss of power before the managed save image was flushed to disk), the save image is corrupted and will not be loaded by QEMU. The older version of `libvirt` does not recognize the corruption, making the problem perpetual. In this case, the guest log will show an attempt to use `-incoming` as one of its arguments, meaning that `libvirt` is trying to start QEMU by migrating in the saved state file.

This problem can be fixed by running `virsh managedsave -remove name_of_guest` to remove the corrupted managed save image. Newer versions of `libvirt` take steps to avoid the corruption in the first place, as well as adding `virsh start --force-boot name_of_guest` to bypass any managed save image.

B.5. Internal error cannot find character device (null)

**Symptom**
This error message appears when attempting to connect to a guest virtual machine's console:

```
# virsh console test2 Connected to domain test2
Escape character is ']' error: internal error cannot find character device (null)
```

**Investigation**
This error message shows that there is no serial console configured for the guest virtual machine.

**Solution**

Set up a serial console in the guest's XML file.

**Procedure B.1. Setting up a serial console in the guest's XML**

1. Add the following XML to the guest virtual machine's XML using `virsh edit`:

   ```xml
   <serial type='pty'>
     <target port='0'/>
   </serial>
   <console type='pty'>
     <target type='serial' port='0'/>
   </console>
   ```

2. Set up the console in the guest kernel command line.
   To do this, either log in to the guest virtual machine to edit the `/boot/grub/grub.conf` file directly, or use the `virt-edit` command line tool. Add the following to the guest kernel command line:

   ```
   console=ttyS0,115200
   ```

3. Run the following command:

   ```
   # virsh start vm && virsh console vm
   ```

**B.6. Guest virtual machine booting stalls with error: No boot device**

**Symptom**

After building a guest virtual machine from an existing disk image, the guest booting stalls with the error message **No boot device**.

However, the guest virtual machine can start successfully using the QEMU command directly.

**Investigation**

The disk's bus type is not specified in the command for importing the existing disk image:

```
# virt-install \
   --connect qemu:///system \
   --ram 2048 -n rhel_64 \
   --os-type=linux --os-variant=rhel5 \
   --disk path=/root/RHEL-Server-5.8-64- \
   virtio.qcow2,device=disk,format=qcow2 \ 
   --vcpus=2 --graphics spice --noautoconsole --import
```

However, the command line used to boot up the guest virtual machine using QEMU directly shows that it uses virtio for its bus type:

````
# ps -ef | grep qemu 
/usr/libexec/qemu-kvm -monitor stdio -drive file=/root/RHEL-Server-5.8-32-virtio.qcow2,index=0,if=virtio,mediadisk,cache=no,format=qcow2 -net nic,vlan=0,model=rtl8139,macaddr=00:30:91:aa:04:74 -net tap,vlan=0,script=/etc/qemu-ifup,downscript=no -m 2048 -smp 2,cores=1,threads=1 -cpu qemu64,+sse2 -soundhw ac97 -rtc-tick-hack -M rhel5.6.0 -usbdevice tablet -vnc :18 -boot c -no-kvm-pit-reinjection
```

Note the `bus=` in the guest's XML generated by `libvirt` for the imported guest.
The bus type for the disk is set as ide, which is the default value set by libvirt. This is the incorrect bus type, and has caused the unsuccessful boot for the imported guest.

Solution

Procedure B.2. Correcting the disk bus type

1. Undefine the imported guest, then re-import it with bus=virtio and the following:

   ```
   # virsh destroy rhel_64
   # virsh undefine rhel_64
   # virt-install:
   --connect qemu:///system
   --ram 1024
   --os-type=linux --os-variant=rhel5
   --disk path=/root/RHEL-Server-5.8-64-virtio.qcow2,device=disk,
   --vcpus=2 --graphics spice --noautoconsole --import
   ```

2. Edit the imported guest's XML using virsh edit and correct the disk bus type.

B.7. Virtual network default has not been started

Symptom

Normally, the configuration for a virtual network named default is installed as part of the libvirt package, and is configured to autostart when libvirtd is started.

If the default network (or any other locally-created network) is unable to start, any virtual machine configured to use that network for its connectivity will also fail to start, resulting in this error message:

```
Virtual network default has not been started
```
Investigation

One of the most common reasons for a libvirt virtual network's failure to start is that the dnsmasq instance required to serve DHCP and DNS requests from clients on that network has failed to start.

To determine if this is the cause, run `virsh net-start default` from a root shell to start the default virtual network.

If this action does not successfully start the virtual network, open `/var/log/libvirt/libvirtd.log` to view the complete error log message.

If a message similar to the following appears, the problem is likely a systemwide dnsmasq instance that is already listening on libvirt's bridge, and is preventing libvirt's own dnsmasq instance from doing so. The most important parts to note in the error message are `dnsmasq` and `exit status 2`:

```
Could not start virtual network default: internal error
Child process (/usr/sbin/dnsmasq --strict-order --
  bind-interfaces
  --pid-file=/var/run/libvirt/network/default.pid --
  conf-file=  
  --except-interface lo --listen-address
  192.168.122.1
  - - dhcp-range 192.168.122.2,192.168.122.254
  - - dhcp-leasefile=/var/lib/libvirt/dnsmasq/default.leases
  - - dhcp-lease-max=253 --dhcp-no-override) status unexpected: exit status 2
```

Solution

If the machine is not using dnsmasq to serve DHCP for the physical network, disable dnsmasq completely.

If it is necessary to run dnsmasq to serve DHCP for the physical network, edit the `/etc/dnsmasq.conf` file. Add or uncomment the first line, as well as one of the two lines following that line. Do not add or uncomment all three lines:

```
bind-interfaces
interface=interface_name
listen-address=chosen_IP_address
```

After making this change and saving the file, restart the systemwide dnsmasq service.

Next, start the `default` network with the `virsh net-start default` command.

Start the virtual machines.

---

B.8. PXE boot (or DHCP) on guest failed

Symptom

A guest virtual machine starts successfully, but is then either unable to acquire an IP address from DHCP or boot using the PXE protocol, or both. There are two common causes of this error: having a long forward delay time set for the bridge, and when the iptables package and kernel do not support checksum mangling rules.

Long forward delay time on bridge

Investigation

This is the most common cause of this error. If the guest network interface is connecting to a bridge device that has STP (Spanning Tree Protocol) enabled, as well as a long forward delay set, the bridge will not forward network packets from the guest virtual machine onto the bridge until at least that number of forward delay seconds have elapsed since the guest connected to the bridge. This delay allows the bridge time to watch traffic from the interface and determine the MAC addresses behind it, and prevent forwarding loops in the network topology.

If the forward delay is longer than the timeout of the guest's PXE or DHCP client, then the client's operation will fail, and the guest will either fail to boot (in the case of PXE) or fail to acquire an IP address (in the case of DHCP).

Solution

If this is the case, change the forward delay on the bridge to 0, disable STP on the bridge, or both.
If the guest has interfaces connecting to a libvirt-managed virtual network, edit the definition for the network, and restart it. For example, edit the default network with the following command:

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

Add the following attributes to the `<bridge>` element:

```
<name_of_bridge='virbr0' delay='0' stp='on'/>
```

**Note**

`delay='0'` and `stp='on'` are the default settings for virtual networks, so this step is only necessary if the configuration has been modified from the default.

If the guest interface is connected to a host bridge that was configured outside of libvirt, change the delay setting.

Add or edit the following lines in the `/etc/sysconfig/network-scripts/ifcfg-name_of_bridge` file to turn STP on with a 0 second delay:

```
STP=on
DELAY=0
```

After changing the configuration file, restart the bridge device:

```
/sbin/ifdown name_of_bridge
/sbin/ifup name_of_bridge
```

**Note**

If `name_of_bridge` is not the root bridge in the network, that bridge's delay will eventually reset to the delay time configured for the root bridge. In this case, the only solution is to disable STP completely on `name_of_bridge`.

### The `iptables` package and kernel do not support checksum mangling rules

**Investigation**

This message is only a problem if all four of the following conditions are true:

- The guest is using virtio network devices. If so, the configuration file will contain `model type='virtio'`.
- The host has the `vhost-net` module loaded. This is true if `ls /dev/vhost-net` does not return an empty result.
- The guest is attempting to get an IP address from a DHCP server that is running directly on the host.
- The `iptables` version on the host is older than 1.4.10. `iptables` 1.4.10 was the first version to add the `libxt_CHECKSUM` extension. This is the case if the following message appears in the `libvirtd` logs:

```
warning: Could not add rule to fixup DHCP response checksums on network default
```

**Important**

Unless all of the other three conditions in this list are also true, the above warning message can be disregarded, and is not an indicator of any other problems.
When these conditions occur, UDP packets sent from the host to the guest have uncomputed checksums. This makes the host's UDP packets seem invalid to the guest's network stack.

Solution

To solve this problem, invalidate any of the four points above. The best solution is to update the host's iptables and kernel to iptables-1.4.10 or newer where possible. Otherwise, the most specific fix is to disable the vhost-net driver for this particular guest. To do this, edit the guest configuration with this command:

```
virsh edit name_of_guest
```

Add or change a `<driver>` line in the `<interface>` section:

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

Save the changes, shut down the guest, and then restart it.

If this problem is still not resolved, the issue may be due to a conflict between firewalld and the default libvirt network.

To fix this, stop firewalld with the `service firewalld stop` command, then restart libvirt with the `service libvirtd restart` command.

B.9. Guest can reach outside network, but cannot reach host when using macvtap interface

Symptom

A guest virtual machine can communicate with other guests, but cannot connect to the host machine after being configured to use a macvtap (also known as `type='direct'`) network interface.

Investigation

Even when not connecting to a Virtual Ethernet Port Aggregator (VEPA) or VN-Link capable switch, macvtap interfaces can be useful. Setting the mode of such an interface to `bridge` allows the guest to be directly connected to the physical network in a very simple manner without the setup issues (or NetworkManager incompatibility) that can accompany the use of a traditional host bridge device.

However, when a guest virtual machine is configured to use a `type='direct'` network interface such as macvtap, despite having the ability to communicate with other guests and other external hosts on the network, the guest cannot communicate with its own host.

This situation is actually not an error — it is the defined behavior of macvtap. Due to the way in which the host's physical Ethernet is attached to the macvtap bridge, traffic into that bridge from the guests that is forwarded to the physical interface cannot be bounced back up to the host's IP stack. Additionally, traffic from the host's IP stack that is sent to the physical interface cannot be bounced back up to the macvtap bridge for forwarding to the guests.

Solution

Use `libvirt` to create an isolated network, and create a second interface for each guest virtual machine that is connected to this network. The host and guests can then directly communicate over this isolated network, while also maintaining compatibility with NetworkManager.

Procedure B.3. Creating an isolated network with libvirt

1. Add and save the following XML in the `/tmp/isolated.xml` file. If the 192.168.254.0/24 network is already in use elsewhere on your network, you can choose a different network.
2. Create the network with this command: `virsh net-define /tmp/isolated.xml`  
3. Set the network to autostart with the `virsh net-autostart isolated` command.  
4. Start the network with the `virsh net-start isolated` command.  
5. Using `virsh edit name_of_guest`, edit the configuration of each guest that uses macvtap for its network connection and add a new `<interface>` in the `<devices>` section similar to the following (note the `<model type='virtio'/>` line is optional to include):

   ```xml
   <interface type='network'>
   <source network='isolated'/>
   <model type='virtio'/>
   </interface>
   ```  
6. Shut down, then restart each of these guests.

The guests are now able to reach the host at the address 192.168.254.1, and the host will be able to reach the guests at the IP address they acquired from DHCP (alternatively, you can manually configure the IP addresses for the guests). Since this new network is isolated to only the host and guests, all other communication from the guests will use the macvtap interface.

B.10. Could not add rule to fixup DHCP response checksums on network ‘default’

**Symptom**

This message appears:

```
Could not add rule to fixup DHCP response checksums on network 'default'
```

**Investigation**

Although this message appears to be evidence of an error, it is almost always harmless.

**Solution**

Unless the problem you are experiencing is that the guest virtual machines are unable to acquire IP addresses through DHCP, this message can be ignored.

If this is the case, refer to Section B.8, ‘PXE boot (or DHCP) on guest failed’ for further details on this situation.

B.11. Unable to add bridge br0 port vnet0: No such device

**Symptom**

The following error message appears:

```
Unable to add bridge name_of_bridge port vnet0: No such device
```

For example, if the bridge name is `br0`, the error message will appear as:

```
Unable to add bridge br0 port vnet0: No such device
```

In *libvirt* versions 0.9.6 and earlier, the same error appears as:

```
Failed to add tap interface to bridge name_of_bridge: No such device
```

Or for example, if the bridge is named `br0`:  

```
Failed to add tap interface to bridge br0: No such device
```
Failed to add tap interface to bridge 'br0': No such device

Investigation
Both error messages reveal that the bridge device specified in the guest's (or domain's) `<interface>` definition does not exist.

To verify the bridge device listed in the error message does not exist, use the shell command `ifconfig br0`.

A message similar to this confirms the host has no bridge by that name:

```
br0: error fetching interface information: Device not found
```

If this is the case, continue to the solution.

However, if the resulting message is similar to the following, the issue exists elsewhere:

```
br0 Link encap:Ethernet  HWaddr 00:00:5A:11:70:48
inet addr:10.22.1.5 Bcast:10.255.255.255  Mask:255.0.0.0
UP BROADCAST RUNNING MULTICAST MTU:1500  Metric:1
RX packets:249841 errors:0 dropped:0 overruns:0 frame:0
TX packets:281948 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:106327234 (101.4 MiB)  TX bytes:21182634 (20.2 MiB)
```

Solution
Edit the existing bridge or create a new bridge with `virsh`

Use `virsh` to either edit the settings of an existing bridge or network, or to add the bridge device to the host system configuration.

Edit the existing bridge settings using `virsh`

Use `virsh edit name_of_guest` to change the `<interface>` definition to use a bridge or network that already exists.

For example, change `type='bridge'` to `type='network'`, and `<source bridge='br0'/>` to `<source network='default'/>`.

Create a host bridge using `virsh`

For `libvirt` version 0.9.8 and later, a bridge device can be created with the `virsh iface-bridge` command. This will create a bridge device `br0` with `eth0`, the physical network interface which is set as part of a bridge, attached:

```
virsh iface-bridge eth0 br0
```

Optional: If desired, remove this bridge and restore the original `eth0` configuration with this command:

```
virsh iface-unbridge br0
```

Create a host bridge manually

For older versions of `libvirt`, it is possible to manually create a bridge device on the host. Refer to `Section 11.3, Bridged networking with libvirt` for instructions.

B.12. Guest is unable to start with error: warning: could not open /dev/net/tun

Symptom
The guest virtual machine does not start after configuring a `type='ethernet'` (also known as 'generic ethernet') interface in the host system. An error appears either in `libvirtd.log`, `var/log/libvirtd/gemu/name_of_guest.log`, or in both, similar to the below message:
Investigation

Use of the generic ethernet interface type (<interface type='ethernet'>) is discouraged, because using it requires lowering the level of host protection against potential security flaws in QEMU and its guests. However, it is sometimes necessary to use this type of interface to take advantage of some other facility that is not yet supported directly in libvirt. For example, openswitch was not supported in libvirt until libvirt-0.9.11, so in older versions of libvirt, <interface type='ethernet'> was the only way to connect a guest to an openswitch bridge.

However, if you configure a <interface type='ethernet'> interface without making any other changes to the host system, the guest virtual machine will not start successfully.

The reason for this failure is that for this type of interface, a script called by QEMU needs to manipulate the tap device. However, with type='ethernet' configured, in an attempt to lock down QEMU, libvirt and SELinux have put in place several checks to prevent this. (Normally, libvirt performs all of the tap device creation and manipulation, and passes an open file descriptor for the tap device to QEMU.)

Solution

Reconfigure the host system to be compatible with the generic ethernet interface.

Procedure B.4. Reconfiguring the host system to use the generic ethernet interface

1. Set SELinux to permissive by configuring

   ```
   SELINUX=permissive
   # This file controls the state of SELinux on the system.
   # SELINUX= can take one of these three values:
   # enforcing - SELinux security policy is enforced.
   # permissive - SELinux prints warnings instead of enforcing.
   # disabled - No SELinux policy is loaded.
   SELINUXTYPE=targeted
   # SELINUXTYPE= can take one of these two values:
   # targeted - Targeted processes are protected,
   # mls - Multi Level Security protection.
   SELINUXTYPE=targeted
   ```

2. From a root shell, run the command `setenforce permissive`.

3. In `/etc/libvirt/qemu.conf` add or edit the following lines:

   ```
   clear_emulator_capabilities = 0
   user = "root"
   group = "root"
   ```


Important

Since each of these steps significantly decreases the host’s security protections against QEMU guest domains, this configuration should only be used if there is no alternative to using <interface type='ethernet'>.
B.13. Migration fails with Error: unable to resolve address

Symptom
QEMU guest migration fails and this error message appears:

```
# virsh migrate qemu
qemu+tcp://192.168.122.12/system
error: Unable to resolve address name_of_host
service '49155': Name or service not known
```

For example, if the destination hostname is "newyork", the error message will appear as:

```
# virsh migrate qemu
qemu+tcp://192.168.122.12/system
error: Unable to resolve address 'newyork'
service '49155': Name or service not known
```

However, this error looks strange as we did not use "newyork" hostname anywhere.

Investigation
During migration, libvirtd running on the destination host creates a URI from an address and port where it expects to receive migration data and sends it back to libvirtd running on the source host.

In this case, the destination host (192.168.122.12) has its name set to "newyork". For some reason, libvirtd running on that host is unable to resolve the name to an IP address that could be sent back and still be useful. For this reason, it returned the "newyork" hostname hoping the source libvirtd would be more successful with resolving the name. This can happen if DNS is not properly configured or /etc/hosts has the hostname associated with local loopback address (127.0.0.1).

Note that the address used for migration data cannot be automatically determined from the address used for connecting to destination libvirtd (for example, from qemu+tcp://192.168.122.12/system). This is because to communicate with the destination libvirtd, the source libvirtd may need to use network infrastructure different from that which virsh (possibly running on a separate machine) requires.

Solution
The best solution is to configure DNS correctly so that all hosts involved in migration are able to resolve all host names.

If DNS cannot be configured to do this, a list of every host used for migration can be added manually to the /etc/hosts file on each of the hosts. However, it is difficult to keep such lists consistent in a dynamic environment.

If the host names cannot be made resolvable by any means, virsh migrate supports specifying the migration host:

```
# virsh migrate qemu
qemu+tcp://192.168.122.12/system
tcp://192.168.122.12
```

Destination libvirtd will take the tcp://192.168.122.12 URI and append an automatically generated port number. If this is not desirable (because of firewall configuration, for example), the port number can be specified in this command:

```
# virsh migrate qemu
qemu+tcp://192.168.122.12/system
tcp://192.168.122.12:12345
```

Another option is to use tunnelled migration. Tunnelled migration does not create a separate connection for migration data, but instead tunnels the data through the connection used for communication with destination libvirtd (for example, qemu+tcp://192.168.122.12/system):

```
# virsh migrate qemu
qemu+tcp://192.168.122.12/system --p2p --tunnelled
```
B.14. Migration fails with Unable to allow access for disk path: No such file or directory

Symptom
A guest virtual machine (or domain) cannot be migrated because libvirt cannot access the disk image(s):

```
# virsh migrate qemu qemu+tcp://name_of_host/system
error: Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or directory
```

For example, if the destination hostname is "newyork", the error message will appear as:

```
# virsh migrate qemu qemu+tcp://newyork/system
error: Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or directory
```

Investigation
By default, migration only transfers the in-memory state of a running guest (such as memory or CPU state). Although disk images are not transferred during migration, they need to remain accessible at the same path by both hosts.

Solution
Set up and mount shared storage at the same location on both hosts. The simplest way to do this is to use NFS:

Procedure B.5. Setting up shared storage

1. Set up an NFS server on a host serving as shared storage. The NFS server can be one of the hosts involved in the migration, as long as all hosts involved are accessing the shared storage through NFS.

```
# mkdir -p /exports/images
# cat >>/etc/exports <<EOF
/exports/images 192.168.122.0/24(rw,no_root_squash)
EOF
```

2. Mount the exported directory at a common location on all hosts running libvirt. For example, if the IP address of the NFS server is 192.168.122.1, mount the directory with the following commands:

```
# cat >>/etc/fstab <<EOF
192.168.122.1:/exports/images /var/lib/libvirt/images nfs auto 0 0
EOF
# mount /var/lib/libvirt/images
```

Note
It is not possible to export a local directory from one host using NFS and mount it at the same path on another host — the directory used for storing disk images must be mounted from shared storage on both hosts. If this is not configured correctly, the guest virtual machine may lose access to its disk images during migration, because the source host's libvirt daemon may change the owner, permissions, and SELinux labels on the disk images after it successfully migrates the guest to its destination. If libvirt detects that the disk images are mounted from a shared storage location, it will not make these changes.

B.15. No guest virtual machines are present when libvirtd is started

Symptom
The libvirt daemon is successfully started, but no guest virtual machines appear to be present.
Investigation

There are various possible causes of this problem. Performing these tests will help to determine the cause of this situation:

Verify KVM kernel modules

Verify that KVM kernel modules are inserted in the kernel:

```bash
# lsmod | grep kvm
kvm_intel             121346  0
kvm                   328927  1 kvm_intel
```

If you are using an AMD machine, verify the `kvm_amd` kernel modules are inserted in the kernel instead, using the similar command `lsmod | grep kvm_amd` in the root shell.

If the modules are not present, insert them using the `modprobe <modulename>` command.

Note

Although it is uncommon, KVM virtualization support may be compiled into the kernel. In this case, modules are not needed.

Verify virtualization extensions

Verify that virtualization extensions are supported and enabled on the host:

```bash
# egrep "(vmx|svm)" /proc/cpuinfo
flags  : fpu vme de pse tsc ... svm ... skinit
wdt npt lbrv sve_lock nrip_save
flags  : fpu vme de pse tsc ... svm ... skinit
wdt npt lbrv sve_lock nrip_save
```

Enable virtualization extensions in your hardware's firmware configuration within the BIOS setup. Refer to your hardware documentation for further details on this.

Verify client URI configuration

Verify that the URI of the client is configured as desired:

```bash
# virsh uri
vbox:///system
```

For example, this message shows the URI is connected to the VirtualBox hypervisor, not QEMU, and reveals a configuration error for a URI that is otherwise set to connect to a QEMU hypervisor. If the URI was correctly connecting to QEMU, the same message would appear instead as:

```bash
# virsh uri
qemu:///system
```

This situation occurs when there are other hypervisors present, which libvirt may speak to by default.

Solution

After performing these tests, use the following command to view a list of guest virtual machines:

```bash
# virsh list --all
```

---

B.16. Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor

Symptom

While `libvirtd` should listen on TCP ports for connections, the connections fail:
# virsh -c qemu+tcp://host/system
error: unable to connect to server at 'host:16509':
Connection refused
error: failed to connect to the hypervisor

The libvirt daemon is not listening on TCP ports even after changing
configuration in /etc/libvirt/libvirtd.conf:

# grep listen /etc/libvirt/libvirtd.conf
listen_tls = 1
listen_tcp = 1
listen_addr = "0.0.0.0"

However, the TCP ports for libvirt are still not open after changing
configuration:

# netstat -lntp | grep libvirtd
#

Investigation
The libvirt daemon was started without the --listen option. Verify
this by running this command:

# ps aux | grep libvirtd
root 27314  0.0  0.0 1000920 18304 ?       Sl
Feb16   1:19 libvirtd --daemon

The output does not contain the --listen option.

Solution
Start the daemon with the --listen option.
To do this, modify the /etc/sysconfig/libvirtd file and
uncomment the following line:

#LIBVIRTDARGS="--listen"

Then restart the libvirtd service with this command:

# /etc/init.d/libvirtd restart

B.17. Common XML errors
The libvirt tool uses XML documents to store structured data. A variety
of common errors occur with XML documents when they are passed to
libvirt through the API. Several common XML errors — including
misformatted XML, inappropriate values, and missing elements — are
detailed below.

B.17.1. Editing domain definition
Although it is not recommended, it is sometimes necessary to edit a guest
virtual machine's (or a domain's) XML file manually. To access the guest's
XML for editing, use the following command:

# virsh edit
name_of_guest.xml

This command opens the file in a text editor with the current definition of
the guest virtual machine. After finishing the edits and saving the
changes, the XML is reloaded and parsed by libvirt. If the XML is correct,
the following message is displayed:

# virsh edit
name_of_guest.xml

Domain name_of_guest.xml XML configuration edited.

Important
When using the edit command in virsh to edit an XML document,
save all changes before exiting the editor.

After saving the XML file, use the xmllint command to validate that the
XML is well-formed, or the virt-xml-validate command to check for
usage problems:

# xmllint --noout config.xml

# virt-xml-validate config.xml

If no errors are returned, the XML description is well-formed and matches
the libvirt schema. While the schema does not catch all constraints, fixing
any reported errors will further troubleshooting.

XML documents stored by libvirt
These documents contain definitions of states and configurations for
the guests. These documents are automatically generated and
should not be edited manually. Errors in these documents contain the
file name of the broken document. The file name is valid only on the
host machine defined by the URI, which may refer to the machine the
command was run on.

Errors in files created by libvirt are rare. However, one possible
source of these errors is a downgrade of libvirt — while newer
versions of libvirt can always read XML generated by older versions,
older versions of libvirt may be confused by XML elements added in
a newer version.

B.17.2. XML syntax errors
Syntax errors are caught by the XML parser. The error message contains
information for identifying the problem.

This example error message from the XML parser consists of three lines —
the first line denotes the error message, and the two following lines
contain the context and location of the XML code containing the error. The
third line contains an indicator showing approximately where the error lies
on the line above it:

```
error: (name_of_guest.xml):6: StartTag: invalid
element name
<vcpu>2</vcpu><

-----------------^  
```

Information contained in this message:

```
(name_of_guest.xml)  
This is the file name of the document that contains the error. File
names in parentheses are symbolic names to describe XML
documents parsed from memory, and do not directly correspond
to files on disk. File names that are not contained in parentheses
are local files that reside on the target of the connection.
```

```
6  
This is the line number in the XML file that contains the error.
```

```
StartTag: invalid element name  
This is the error message from the libxml2 parser, which
describes the specific XML error.
```

B.17.2.1. Stray < in the document
Symptom
The following error occurs:

```
error: (name_of_guest.xml):6: StartTag: invalid
element name
<vcpu>2</vcpu><

-----------------^  
```

Investigation
This error message shows that the parser expects a new element
name after the < symbol on line 6 of a guest's XML file.

Ensure line number display is enabled in your text editor. Open the
XML file, and locate the text on line 6:

```
<domain type='kvm'>
  <name name_of_guest/>
  <memory>524288</memory>
  <vcpu>2</vcpu>
</domain>
```

This snippet of a guest's XML file contains an extra < in the
document:

Solution
Remove the extra < or finish the new element.

B.17.2.2. Unterminated attribute
Symptom
The following error occurs:

```
error: (name_of_guest.xml):2: Unescaped '<' not
allowed in attributes values

<name name_of_guest/>

```

Investigation
This snippet of a guest's XML file contains an unterminated element
attribute value:

```xml
<domain type='kvm'>
 <name>name_of_guest</name>
</domain>
```

In this case, `'kvm'` is missing a second quotation mark. Strings of attribute values, such as quotation marks and apostrophes, must be opened and closed, similar to XML start and end tags.

**Solution**
Correctly open and close all attribute value strings.

### B.17.2.3. Opening and ending tag mismatch

**Symptom**

The following error occurs:

```none
error: (name_of_guest.xml):61: Opening and ending tag mismatch: clock line 16 and domain
```

**Investigation**

The error message above contains three clues to identify the offending tag:

- The message following the last colon, `clock line 16 and domain`, reveals that `<clock>` contains a mismatched tag on line 16 of the document. The last hint is the pointer in the context part of the message, which identifies the second offending tag.

Unpaired tags must be closed with `>`.

```xml
<domain type='kvm'>
  ...
</domain>
```

This error is caused by mismatched XML tags in the file. Every XML tag must have a matching start and end tag.

**Other examples of mismatched XML tags**

- The following examples produce similar error messages and show variations of mismatched XML tags.

  This snippet contains an unended pair tag for `<features>`:

  ```xml
  <domain type='kvm'>
    ...
    <features>
      <acpi/>
      <pae/>
    </features>
    ...
  </domain>
  ```

  This snippet contains an end tag (`</name>`) without a corresponding start tag:

  ```xml
  <domain type='kvm'>
    ...
  </domain>
  ```

**Solution**

Ensure all XML tags start and end correctly.

### B.17.2.4. Typographical errors in tags

**Symptom**

The following error message appears:

```none
error: (name_of_guest.xml):1: Specification mandate value for attribute ty
```

**Investigation**

XML errors are easily caused by a simple typographical error. This error message highlights the XML error — in this case, an extra white space within the word `type` — with a pointer.

```xml
<domain ty pe='kvm'>
```

These XML examples will not parse correctly because of typographical errors such as a missing special character, or an
Solution
To identify the problematic tag, read the error message for the context of the file, and locate the error with the pointer. Correct the XML and save the changes.

B.17.3. Logic and configuration errors
A well-formatted XML document can contain errors that are correct in syntax but libvirt cannot parse. Many of these errors exist, with two of the most common cases outlined below.

B.17.3.1. Vanishing parts
Symptom
Parts of the change you have made do not show up and have no effect after editing or defining the domain. The define or edit command works, but when dumping the XML once again, the change disappears.

Investigation
This error likely results from a broken construct or syntax that libvirt does not parse. The libvirt tool will generally only look for constructs it knows but ignore everything else, resulting in some of the XML changes vanishing after libvirt parses the input.

Solution
Validate the XML input before passing it to the edit or define commands. The libvirt developers maintain a set of XML schemas bundled with libvirt which define the majority of the constructs allowed in XML documents used by libvirt.

Validate libvirt XML files using the following command:

```bash
# virt-xml-validate libvirt.xml
```

If this command passes, libvirt will likely understand all constructs from your XML, except if the schemas cannot detect options which are valid only for a given hypervisor. Any XML generated by libvirt as a result of a virsh dump command, for example, should validate without error.

B.17.3.2. Incorrect drive device type
Symptom
The definition of the source image for the CD-ROM virtual drive is not present, despite being added:

```xml
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
</disk>
```

Solution
Correct the XML by adding the missing <source> parameter as follows:

```xml
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source file='/path/to/image.iso'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
</disk>
```

A type='block' disk device expects that the source is a physical device. To use the disk with an image file, use type='file' instead.
NetKVM Driver Parameters

After the NetKVM driver is installed, you can configure it to better suit your environment. The parameters listed in this section can be configured in the Windows Device Manager (devmgmt.msc).

**Important**
Modifying the driver's parameters causes Windows to re-load that driver. This interrupts existing network activity.

**Procedure C.1. Configuring NetKVM Parameters**

1. **Open Device Manager**
   - Click on the Start button. In the right-hand pane, right-click on Computer, and click Manage. If prompted, click Continue on the User Account Control window. This opens the Computer Management window.
   - In the left-hand pane of the Computer Management window, click Device Manager.

2. **Locate the correct device**
   - In the central pane of the Computer Management window, click on the + symbol beside Network adapters.
   - Under the list of Fedora VirtIO Ethernet Adapter devices, double-click on NetKVM. This opens the Properties window for that device.

3. **View device parameters**
   - In the Properties window, click on the Advanced tab.

4. **Modify device parameters**
   - Click on the parameter you wish to modify to display the options for that parameter.
   - Modify the options as appropriate, then click on OK to save your changes.

**C.1. Configurable parameters for NetKVM**

**Logging parameters**

**Logging.Enable**
- A Boolean value that determines whether logging is enabled. The default value is 1 (enabled).

**Logging.Level**
- An integer that defines the logging level. As the integer increases, so does the verbosity of the log. The default value is 0 (errors only). 1-2 adds configuration messages. 3-4 adds packet flow information. 5-6 adds interrupt and DPC level trace information.

**Important**
High logging levels will slow down your guest virtual machine.

**Logging.Statistics(sec)**
- An integer that defines whether log statistics are printed, and the time in seconds between each periodical statistics printout. The default value is 0 (no logging statistics).

**Initial parameters**

**Assign MAC**
- A string that defines the locally-administered MAC address for the para-virtualized NIC. This is not set by default.

**Init.ConnectionRate(Mb)**
- An integer that represents the connection rate in megabytes. The default value for Windows 2008 and later is 10000.

**Init.Do802.1PQ**
- A Boolean value that enables Priority/VLAN tag population and
removal support. The default value is 1 (enabled).

**Init.UseMergedBuffers**
A Boolean value that enables merge-able RX buffers. The default value is 1 (enabled).

**Init.UsePublishEvents**
A Boolean value that enables published event use. The default value is 1 (enabled).

**Init.MTUSize**
An integer that defines the maximum transmission unit (MTU). The default value is 1500. Any value from 500 to 65500 is acceptable.

**Init.IndirectTx**
Controls whether indirect ring descriptors are in use. The default value is Disable, which disables use of indirect ring descriptors. Other valid values are Enable, which enables indirect ring descriptor usage; and Enable*, which enables conditional use of indirect ring descriptors.

**Init.MaxTxBuffers**
An integer that represents the amount of TX ring descriptors that will be allocated. The default value is 1024. Valid values are: 16, 32, 64, 128, 256, 512, or 1024.

**Init.MaxRxBuffers**
An integer that represents the amount of RX ring descriptors that will be allocated. The default value is 256. Valid values are: 16, 32, 64, 128, 256, 512, or 1024.

**Offload.Tx.Checksum**
Specifies the TX checksum offloading mode.

In Fedora 18 and onward, the valid values for this parameter are All (the default), which enables IP, TCP and UDP checksum offloading for both IPv4 and IPv6; TCP/UDP(v4, v6), which enables TCP and UDP checksum offloading for both IPv4 and IPv6; TCP/UDP(v4), which enables TCP and UDP checksum offloading for IPv4 only; and TCP(v4), which enables only TCP checksum offloading for IPv4 only.

In Fedora 17 and earlier, the valid values for this parameter are TCP/UDP (the default value), which enables TCP and UDP checksum offload; TCP, which enables only TCP checksum offload; or Disable, which disables TX checksum offload.

**Offload.Tx.LSO**
A Boolean value that enables TX TCP Large Segment Offload (LSO). The default value is 1 (enabled).

**Offload.Rx.Checksum**
Specifies the RX checksum offloading mode.

In Fedora 18 and onward, the valid values for this parameter are All (the default), which enables IP, TCP and UDP checksum offloading for both IPv4 and IPv6; TCP/UDP(v4, v6), which enables TCP and UDP checksum offloading for both IPv4 and IPv6; TCP/UDP(v4), which enables TCP and UDP checksum offloading for IPv4 only; and TCP(v4), which enables only TCP checksum offloading for IPv4 only.

In Fedora 17 and earlier, the valid values are Disable (the default), which disables RX checksum offloading; All, which enables TCP, UDP, and IP checksum offloading; TCP/UDP, which enables TCP and UDP checksum offloading; and TCP, which enables only TCP checksum offloading.

**Test and debug parameters**

**TestOnly.DelayConnect(ms)**
The period for which to delay connection upon startup, in milliseconds. The default value is 6.

**TestOnly.DPCChecking**
Sets the DPC checking mode. 0 (the default) disables DPC checking. 1 enables DPC checking; each hang test verifies DPC activity and acts as if the DPC was spawned. 2 clears the device interrupt status and is otherwise identical to 1.

**TestOnly.Scatter-Gather**
A Boolean value that determines whether scatter-gather functionality is enabled. The default value is 1 (enabled). Setting this value to 0 disables scatter-gather functionality and all dependent capabilities.

**TestOnly.InterruptRecovery**
A Boolean value that determines whether interrupt recovery is enabled. The default value is 1 (enabled).

**TestOnly.PacketFilter**
A Boolean value that determines whether packet filtering is enabled. The default value is 1 (enabled).

**TestOnly.BatchReceive**
A Boolean value that determines whether packets are received in batches, or singularly. The default value is 1, which enables bached packet receipt.

**TestOnly.Promiscuous**
A Boolean value that determines whether promiscuous mode is enabled. The default value is 0 (disabled).

**TestOnly.AnalyzeIPPackets**
A Boolean value that determines whether the checksum fields of outgoing IP packets are tested and verified for debugging purposes. The default value is 0 (no checking).

**TestOnly.RXThrottle**
An integer that determines the number of receive packets handled in a single DPC. The default value is 1000.

**TestOnly.UseSwTxChecksum**
A Boolean value that determines whether hardware checksumming is enabled. The default value is 0 (disabled).

---

**qemu-kvm Whitelist**

**D.1. Introduction**
**Product identification**
Fedora

**Objectives**
The primary objective of this whitelist is to provide a complete list of the supported options of the `qemu-kvm` utility used as an emulator and a virtualizer in Fedora. This is a comprehensive summary of the supported options.

**Background**
Fedora uses KVM as an underlying virtualization technology. The machine emulator and virtualizer used is a modified version of QEMU called qemu-kvm. This version does not support all configuration options of the original QEMU and it adds some additional options.

**Scope of the chapter**

**Used 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.

**D.2. Basic options**
**Emulated machine**
-M `<machine-type>`
Processor type

-cpu <model>[,<FEATURE>][...]  

We support exposing additional features and placing restrictions. Supported models are:

- **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 P9xx (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  

<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>][,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>

D.3. Disk options

Generic drive

-drive <option>[,<option>][,...]

Supported with the following options:

- **readonly** [on|off]
- **werror** [enospc|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>
D.4. Display options

Disabled graphics

-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.
- x509-verify=<path/to/certificate/dir> - Supported when tls specified.
- sasl
- acl

Spice desktop

-spice 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-ca-cert-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]
D.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
- vhostfdd
- vhostforce

D.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_nb
  - status
  - ctrl_vq
  - ctrl_rx
  - ctrl_vlan

agent-mouse=[on|off]
playback-compression=[on|off]
seamless-migration=[on|off]
- ctrl_rx_extra
- mac
- netdev
- bootindex
- x-drimer
- 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
  - bcache
  - loevntfd
  - vectors
  - indirect_desc
  - event_id
  - scsi
  - 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_id
  - max_ports
  - flow_control
  - addr

- ES1370
  - addr
- AC97
  - addr
- intel-hda
  - addr
- hda-duplex
  - cad
- hda-micro
  - cad
- hda-output
  - cad
- i8300esb
  - 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
  producid
  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 since 6.3
  chardev
  filter

scsi-cd - tech preview for 6.3
  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
  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
  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>],[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

D.7. Linux/Multiboot boot

Kernel file
-kernel <bzImage>

Note: multiboot images are not supported

Ram disk
-initrd <file>

Command line parameter
-append <cmdline>

D.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>]`

D.9. Help and information options

Help
- `-h`
- `-help`

Version
- `-version`

Audio help
- `-audio-help`

D.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>`

Managing guests with virsh

virsh is a command line interface tool for managing guests and the hypervisor. The virsh command-line tool is built on the libvirt
management API and operates as an alternative to the qemu-kvm command and the graphical virt-manager application. The virsh command can be used in read-only mode by unprivileged users or, with root access, full administration functionality. The virsh command is ideal for scripting virtualization administration.

### E.1. virsh command quick reference

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>help</td>
<td>Prints basic help information.</td>
</tr>
<tr>
<td>list</td>
<td>Lists all guests.</td>
</tr>
<tr>
<td>dumpxml</td>
<td>Outputs the XML configuration file for the guest.</td>
</tr>
<tr>
<td>create</td>
<td>Creates a guest from an XML configuration file and starts the new guest.</td>
</tr>
<tr>
<td>start</td>
<td>Starts an inactive guest.</td>
</tr>
<tr>
<td>destroy</td>
<td>Forces a guest to stop.</td>
</tr>
<tr>
<td>define</td>
<td>Creates a guest from an XML configuration file without starting the new guest.</td>
</tr>
<tr>
<td>domid</td>
<td>Displays the guest's ID.</td>
</tr>
<tr>
<td>domuuid</td>
<td>Displays the guest's UUID.</td>
</tr>
<tr>
<td>dominfo</td>
<td>Displays guest information.</td>
</tr>
<tr>
<td>domname</td>
<td>Displays the guest's name.</td>
</tr>
<tr>
<td>domstate</td>
<td>Displays the state of a guest.</td>
</tr>
<tr>
<td>quit</td>
<td>Quits the interactive terminal.</td>
</tr>
<tr>
<td>reboot</td>
<td>Reboots a guest.</td>
</tr>
<tr>
<td>restore</td>
<td>Restores a previously saved guest stored in a file.</td>
</tr>
<tr>
<td>resume</td>
<td>Resumes a paused guest.</td>
</tr>
<tr>
<td>save</td>
<td>Saves the present state of a guest to a file.</td>
</tr>
<tr>
<td>shutdown</td>
<td>Gracefully shuts down a guest.</td>
</tr>
<tr>
<td>suspend</td>
<td>Pauses a guest.</td>
</tr>
<tr>
<td>undefine</td>
<td>Deletes all files associated with a guest.</td>
</tr>
<tr>
<td>migrate</td>
<td>Migrates a guest to another host.</td>
</tr>
</tbody>
</table>

The following virsh command options manage guest and hypervisor resources:
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setmem</td>
<td>Sets the allocated memory for a guest. Refer to the <code>virsh</code> manpage for more details.</td>
</tr>
<tr>
<td>setmaxmem</td>
<td>Sets maximum memory limit for the hypervisor. Refer to the <code>virsh</code> manpage for more details.</td>
</tr>
<tr>
<td>setvcpus</td>
<td>Changes number of virtual CPUs assigned to a guest. Refer to the <code>virsh</code> manpage for more details.</td>
</tr>
<tr>
<td>vcpuinfo</td>
<td>Displays virtual CPU information about a guest.</td>
</tr>
<tr>
<td>vcpupin</td>
<td>Controls the virtual CPU affinity of a guest.</td>
</tr>
<tr>
<td>domblkstat</td>
<td>Displays block device statistics for a running guest.</td>
</tr>
<tr>
<td>domifstat</td>
<td>Displays network interface statistics for a running guest.</td>
</tr>
<tr>
<td>attach-device</td>
<td>Attach a device to a guest, using a device definition in an XML file.</td>
</tr>
<tr>
<td>attach-disk</td>
<td>Attaches a new disk device to a guest.</td>
</tr>
<tr>
<td>attach-interface</td>
<td>Attaches a new network interface to a guest.</td>
</tr>
<tr>
<td>update-device</td>
<td>Detaches a disk image from a guests CD-ROM drive. See Section E.2, “Attaching and updating a device with virsh” for more details.</td>
</tr>
<tr>
<td>detach-device</td>
<td>Detaches a device from a guest, takes the same kind of XML descriptions as command <code>attach-device</code>.</td>
</tr>
<tr>
<td>detach-disk</td>
<td>Detaches a disk device from a guest.</td>
</tr>
<tr>
<td>detach-interface</td>
<td>Detach a network interface from a guest.</td>
</tr>
</tbody>
</table>

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](http://libvirt.org/formatstorage.html)
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find-storage-pool-sources</td>
<td>Returns the XML definition for all storage pools of a given type that could be found.</td>
</tr>
<tr>
<td>find-storage-pool-sources host port</td>
<td>Returns data on all storage pools of a given type that could be found as XML. If the host and port are provided, this command can be run remotely.</td>
</tr>
<tr>
<td>pool-autostart</td>
<td>Sets the storage pool to start at boot time.</td>
</tr>
<tr>
<td>pool-build</td>
<td>The pool-build command builds a defined pool. This command can format disks and create partitions.</td>
</tr>
<tr>
<td>pool-create</td>
<td>pool-create creates and starts a storage pool from the provided XML storage pool definition file.</td>
</tr>
<tr>
<td>pool-create-as name</td>
<td>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.</td>
</tr>
<tr>
<td>pool-define</td>
<td>pool-define creates a storage pool from an XML definition file but does not start the new storage pool.</td>
</tr>
<tr>
<td>pool-define-as name</td>
<td>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.</td>
</tr>
<tr>
<td>pool-destroy</td>
<td>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.</td>
</tr>
<tr>
<td>pool-delete</td>
<td>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.</td>
</tr>
<tr>
<td>pool-dumpxml</td>
<td>Prints the XML definition for a storage pool.</td>
</tr>
<tr>
<td>pool-edit</td>
<td>Opens the XML definition file for a storage pool in the users default text editor.</td>
</tr>
<tr>
<td>pool-info</td>
<td>Returns information about a storage pool.</td>
</tr>
<tr>
<td>pool-list</td>
<td>Lists storage pools known to libvirt. By default, pool-list lists pools in use by active guests. The --inactive parameter lists inactive pools and the --all parameter lists all pools.</td>
</tr>
<tr>
<td>pool-undefine</td>
<td>Deletes the definition for an inactive storage pool.</td>
</tr>
<tr>
<td>pool-uuid</td>
<td>Returns the UUID of the named pool.</td>
</tr>
<tr>
<td>pool-name</td>
<td>Prints a storage pool's name when provided the UUID of a storage pool.</td>
</tr>
<tr>
<td>pool-refresh</td>
<td>Refreshes the list of volumes contained in a storage pool.</td>
</tr>
<tr>
<td>pool-start</td>
<td>Starts a storage pool that is defined but inactive.</td>
</tr>
</tbody>
</table>
Table E.4. Volume options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vol-create</td>
<td>Create a volume from an XML file.</td>
</tr>
<tr>
<td>vol-create-from</td>
<td>Create a volume using another volume as input.</td>
</tr>
<tr>
<td>vol-create-as</td>
<td>Create a volume from a set of arguments.</td>
</tr>
<tr>
<td>vol-clone</td>
<td>Clone a volume.</td>
</tr>
<tr>
<td>vol-delete</td>
<td>Delete a volume.</td>
</tr>
<tr>
<td>vol-wipe</td>
<td>Wipe a volume.</td>
</tr>
<tr>
<td>vol-dumpxml</td>
<td>Show volume information in XML.</td>
</tr>
<tr>
<td>vol-info</td>
<td>Show storage volume information.</td>
</tr>
<tr>
<td>vol-list</td>
<td>List volumes.</td>
</tr>
<tr>
<td>vol-pool</td>
<td>Returns the storage pool for a given volume key or path.</td>
</tr>
<tr>
<td>vol-path</td>
<td>Returns the volume path for a given volume name or key.</td>
</tr>
<tr>
<td>vol-name</td>
<td>Returns the volume name for a given volume key or path.</td>
</tr>
<tr>
<td>vol-key</td>
<td>Returns the volume key for a given volume name or path.</td>
</tr>
</tbody>
</table>

Table E.5. Secret options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret-define</td>
<td>Define or modify a secret from an XML file.</td>
</tr>
<tr>
<td>secret-dumpxml</td>
<td>Show secret attributes in XML.</td>
</tr>
<tr>
<td>secret-set-value</td>
<td>Set a secret value.</td>
</tr>
<tr>
<td>secret-get-value</td>
<td>Output a secret value.</td>
</tr>
<tr>
<td>secret-undefine</td>
<td>Undefine a secret.</td>
</tr>
<tr>
<td>secret-list</td>
<td>List secrets.</td>
</tr>
</tbody>
</table>

Table E.6. Network filter options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nwfilter-define</td>
<td>Define or update a network filter from an XML file.</td>
</tr>
<tr>
<td>nwfilter-undefine</td>
<td>Undefine a network filter.</td>
</tr>
<tr>
<td>nwfilter-dumpxml</td>
<td>Show network filter information in XML.</td>
</tr>
<tr>
<td>nwfilter-list</td>
<td>List network filters.</td>
</tr>
<tr>
<td>nwfilter-edit</td>
<td>Edit XML configuration for a network filter.</td>
</tr>
</tbody>
</table>

This table contains virsh command options for snapshots:

Table E.7. Snapshot options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshot-create</td>
<td>Create a snapshot.</td>
</tr>
<tr>
<td>snapshot-current</td>
<td>Get the current snapshot.</td>
</tr>
<tr>
<td>snapshot-delete</td>
<td>Delete a domain snapshot.</td>
</tr>
<tr>
<td>snapshot-dumpxml</td>
<td>Dump XML for a domain snapshot.</td>
</tr>
<tr>
<td>snapshot-list</td>
<td>List snapshots for a domain.</td>
</tr>
<tr>
<td>snapshot-revert</td>
<td>Revert a domain to a snapshot.</td>
</tr>
</tbody>
</table>

This table contains miscellaneous virsh commands:

Table E.8. Miscellaneous options

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Displays the version of virsh.</td>
</tr>
<tr>
<td>nodeinfo</td>
<td>Outputs information about the hypervisor.</td>
</tr>
</tbody>
</table>

E.2. Attaching and updating a device with virsh

For information on this procedure refer to Section 28.3.1, “Adding file based storage to a guest”

E.3. Connecting to the hypervisor

Connect to a hypervisor session with virsh:
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`.

### E.4. Creating a virtual machine

#### XML dump (configuration file)

Output a guest’s XML configuration file with `virsh`:

```
# virsh dumpxml [guest-id, guestname or uuid]
```

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

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

This file `guest.xml` can recreate the guest (refer to Editing a guest’s configuration file. You can edit this XML configuration file to configure additional devices or to deploy additional guests.

An example of `virsh dumpxml` output:

```
<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='guest1-rhel6-64.img'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <interface type='bridge'>
      <mac address='52:54:00:b9:35:a9'/>
      <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 serialport='0'/>
  </console>
  <input type='tablet' bus='usb'/>
  <input type='ps2'/>
  <graphics type='vnc' port='-1' autoport='yes'/>
  <sound model='ich6'/>
  <video model='cirrus' vram='9216' heads='1'/>
  <memballoon model='virtio'/>
</domain>
```

Note that the `<shareable/>` flag is set. This indicates the device is expected to be shared between domains (assuming the hypervisor and OS support this), which means that caching should be deactivated for that device.

**Creating a guest from a configuration file**

Guests can be created from XML configuration files. You can copy
existing XML from previously created guests or use the `dumpxml` option (refer to Section E.4, “Creating a virtual machine XML dump (configuration file)”). To create a guest with `virsh` from an XML file:

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

**Editing a guest’s configuration file**

Instead of using the `dumpxml` option (refer to Section E.4, “Creating a virtual machine XML dump (configuration file)”) guests can be edited either while they run or while they are offline. The `virsh edit` command provides this functionality. For example, to edit the guest named `softwaretesting`:

```
# virsh edit softwaretesting
```

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

### E.4.1. Adding multifunction PCI devices to KVM guests

This section will demonstrate how to add multi-function PCI devices to KVM guests.

1. Run the `virsh edit [guestname]` command to edit the XML configuration file for the guest.
2. In the address type tag, add a `multifunction='on'` entry for `function='0x0'`.

   ```xml
   <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:

   ```xml
   <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 shows:

   ```
   $ lspci
   00:05.0 SCSI storage controller: Fedora, Inc Virtio block device
   00:05.1 SCSI storage controller: Fedora, Inc Virtio block device
   ```

### E.5. Suspending, resuming, saving and restoring a guest

#### Suspending a guest

Suspend a guest with `virsh`:

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

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

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

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

**Save a guest**

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

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

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

**Restore a guest**

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

```
# virsh restore filename
```

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

---

### E.6. Shutting down, rebooting and force-shutdown of a guest

**Shut down a guest**

Shut down a guest using the `virsh` command:

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

You can control the behavior of the rebooting guest by modifying the `on_shutdown` parameter in the guest's configuration file.

**Rebooting a guest**

Reboot a guest using `virsh` command:

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

You can control the behavior of the rebooting guest by modifying the `on_reboot` element in the guest's configuration file.

**Forcing a guest to stop**

Force a guest to stop with the `virsh` command:

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

This command does an immediate ungraceful shutdown and stops the specified guest. Using `virsh destroy` can corrupt guest file systems. Use the `destroy` option only when the guest is unresponsive.

---

### E.7. Retrieving guest information

**Getting the domain ID of a guest**

To get the domain ID of a guest:

```
# virsh domid {domain-name or domain-uuid}
```

**Getting the domain name of a guest**

To get the domain name of a guest:

```
# virsh domname {domain-id or domain-uuid}
```

**Getting the UUID of a guest**

To get the Universally Unique Identifier (UUID) for a guest:

```
# virsh domuuid {domain-id or domain-name}
```
An example of `virsh domuuid` output:
```
# virsh domuuid r5b2-mySQL01
4a4c5a0a7-e3f-c781-96e4-288f2862f811
```

Displaying guest information

Using `virsh` with the guest's domain ID, domain name or UUID you can display information on the specified guest:
```
# virsh dominfo (domain-id, domain-name or domain-uuid)
```

This is an example of `virsh dominfo` output:
```
# virsh dominfo vr-rhel6u1-x86_64-kvm
Id:             9
Name:           vr-rhel6u1-x86_64-kvm
UUID:           a03093a1-5da6-a2a2-3baf-a845db2f1b09
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)
```

E.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                2985 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.

E.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`.

E.10. Displaying per-guest information

Displaying the guests
To display the guest list and their current states with `virsh`:

```
# virsh list
```

Other options available include:

- the `--inactive` option to list inactive guests (that is, guests that have been defined but are not currently active), and
- the `--all` option lists all guests. For example:

```
# virsh list --all
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Domain-0</td>
<td>running</td>
</tr>
<tr>
<td>1</td>
<td>Domain202</td>
<td>paused</td>
</tr>
<tr>
<td>2</td>
<td>Domain10</td>
<td>inactive</td>
</tr>
<tr>
<td>3</td>
<td>Domain960</td>
<td>crashed</td>
</tr>
</tbody>
</table>

There are seven states that can be visible using this command:

- **Running** - The *running* state refers to guests 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`, `xm pause` or `virsh suspend`. When a guest is paused it consumes memory and other resources but it is ineligible for scheduling and CPU resources from the hypervisor.
- **Shutdown** - The `shutdown` state is for guests in the process of shutting down. The guest is sent a shutdown signal and should be in the process of stopping its operations gracefully. This may not work with all guest operating systems; some operating systems do not respond to these signals.
- **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 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 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:       0.8s
CPU Affinity:   yy
```

**Configuring virtual CPU affinity**

To configure the affinity of virtual CPUs with physical CPUs:

```
# virsh vcpupin domain-id vcpu cpulist
```

The `domain-id` parameter is the guest's ID number or name.

The `vcpu` parameter denotes the number of virtualized CPUs allocated to the guest. The `vcpu` parameter must be provided.

The `cpulist` parameter is a list of physical CPU identifier numbers separated by commas. The `cpulist` parameter determines which physical CPUs the VCPUs can run on.

**Configuring virtual CPU count**

To modify the number of CPUs assigned to a guest with `virsh`:

```
# virsh setvcpus (domain-name, domain-id or domain-uuid) count
```

This `count` value cannot exceed the number of CPUs that were assigned to the guest when it was created.

**Configuring memory allocation**

To modify a guest'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. Values lower than 64 MB are unlikely to work with most guest operating systems. A higher maximum memory value does not affect active guests. If the new value is lower than the available memory, it will shrink possibly causing the guest 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

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 RAM, guest 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:

```xml
<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 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 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. The units for this value are kibibytes (i.e. blocks of 1024 bytes)

```bash
# 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 domain can use.

Displaying guest block device information

Use `virsh domblkstat` to display block device statistics for a running guest.

```bash
# virsh domblkstat GuestName block-device
```

Displaying guest network device information

Use `virsh domifstat` to display network interface statistics for a running guest.

```bash
# virsh domifstat GuestName interface-device
```

E.11. Managing virtual networks
This section covers managing virtual networks with the `virsh` command. To list virtual networks:

```
# virsh net-list
```

This command generates output similar to:

```
# virsh net-list
Name                State  Autostart
-----------------------------------------
default              active  yes
vnet1                active  yes
vnet2                active  yes
```

To view network information for a specific virtual network:

```
# virsh net-dumpxml NetworkName
```

This displays information about a specified virtual network in XML format:

```
# virsh net-dumpxml vnet1
<network>
  <name>vnet1</name>
  <uuid>98361b46-1581-acb7-1643-85a412626e70</uuid>
  <forward dev='eth0'/>
  <bridge name='vnet0' stp='on' forwardDelay='0' />
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254' />
    </dhcp>
  </ip>
</network>
```

Other `virsh` commands used in managing virtual networks are:

- `virsh net-autostart network-name` — Autostart a network specified as `network-name`.
- `virsh net-create XMLfile` — generates and starts a new network using an existing XML file.
- `virsh net-define XMLfile` — generates a new network device from an existing XML file without starting it.
- `virsh net-destroy network-name` — destroys a network specified as `network-name`.
- `virsh net-name networkUUID` — converts a specified `networkUUID` to a network name.
- `virsh net-uuid network-name` — converts 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.

### E.12. Migrating guests with virsh

Information on migration using `virsh` is located in the section entitled Live KVM Migration with `virsh`. Refer to Section 20.4, “Live KVM migration with `virsh`”.

### E.13. Disk image management with live block copy

Live block copy allows you to copy an in use guest disk image to a destination image and switches the guest disk image to the destination guest image while the guest is running. Whilst live migration moves the memory and registry state of the host, the guest is kept in shared storage. Live block copy allows you to move the entire guest contents to another host on the fly while the guest is running. Live block copy may also be used for live migration without requiring permanent shared storage. In this method the disk image is copied to the destination host after migration, but while the guest is running.

Live block copy is especially useful for the following applications:

- moving the guest image from local storage to a central location
- when maintenance is required, guests can be transferred to another location, with no loss of performance
- allows for management of guest images for speed and efficiency
- image format conversions can be done without having to shut down the guest
Example E.1. Example (live block copy)

This example shows what happens when live block copy is performed. The example has a backing file (base) that is shared between a source and destination. It also has two overlays (sn1 and sn2) that are only present on the source and must be copied.

1. The backing file chain at the beginning looks like this:
   
   base ← sn1 ← sn2
   
   The components are as follows:
   
   - base - the original disk image
   - sn1 - the first snapshot that was taken of the base disk image
   - sn2 - the most current snapshot
   - active - the copy of the disk

2. When a copy of the image is created as a new image on top of sn2 the result is this:
   
   base ← sn1 ← sn2 ← active

3. At this point the read permissions are all in the correct order and are set automatically. To make sure write permissions are set properly, a mirror mechanism redirects all writes to both sn2 and active, so that sn2 and active read the same at any time (and this mirror mechanism is the essential difference between live block copy and image streaming).

4. A background task that loops over all disk clusters is executed. For each cluster, there are the following possible cases and actions:
   
   - The cluster is already allocated in active and there is nothing to do.
   - Use `bdrv_is_allocated()` to follow the backing file chain. If the cluster is read from base (which is shared) there is nothing to do.
   - If `bdrv_is_allocated()` variant is not feasible, rebase the image and compare the read data with write data in base in order to decide if a copy is needed.
   - In all other cases, copy the cluster into active

5. When the copy has completed, the backing file of active is switched to base (similar to rebase)

To reduce the length of a backing chain after a series of snapshots, the following commands are helpful: `blockcommit` and `blockpull`. See Section E.13.1, "Using blockcommit to shorten a backing chain" for more information.

E.13.1. Using blockcommit to shorten a backing chain

This section demonstrates how to use `blockcommit` to shorten a backing chain. For more background on backing chains, see Section E.13, "Disk image management with live block copy".

`blockcommit` copies data from one part of the chain down into a backing file, allowing you to pivot the rest of the chain in order to bypass the committed portions. For example, suppose this is the current state:

```
base ← snap1 ← snap2 ← active
```

Using `blockcommit` moves the contents of snap2 into snap1, allowing you to delete snap2 from the chain, making backups much quicker.

Procedure E.1. virsh blockcommit

Run the following command:

```
# virsh blockcommit $dom $disk --base snap1 --top snap2 --wait --verbose
```

The contents of snap2 are moved into snap1, resulting in:

```
base ← snap1 ← active.
```

Snap2 is no longer valid and can be deleted.

**Warning**

`blockcommit` will corrupt any file that depends on the `--base` argument (other than files that depended on the `--top` argument, as those files now point to the base). To prevent this, do not commit changes into files shared by more than one guest. The `--verbose` option will allow the progress to be printed on the screen.

E.13.2. Using blockpull to shorten a backing chain

`blockpull` can be used in the following applications:

- Flattens an image by populating it with data from its backing image chain. This makes the image file self-contained so that it no longer depends on backing images and looks like this:
Before: base.img ← Active
After: base.img is no longer used by the guest and Active contains all of the data.

Flattens part of the backing image chain. This can be used to flatten snapshots into the top-level image and looks like this:
Before: base ← sn1 ← sn2 ← active
After: base.img ← active. Note that active now contains all data from sn1 and sn2 and neither sn1 nor sn2 are used by the guest.

Moves the disk image to a new file system on the host. This is allows image files to be moved while the guest is running and looks like this:
Before (The original image file): /fs1/base.vm.img
After: /fs2/active.vm.qcow2 is now the new file system and /fs1/base.vm.img is no longer used.

Useful in live migration with post-copy storage migration. The disk image is copied from the source host to the destination host after live migration completes.
In short this is what happens: Before /source-host/base.vm.img
After /destination-host/active.vm.qcow2 /source-host/base.vm.img is no longer used.

Procedure E.2. Using blockpull to shorten a backing chain

1. It may be helpful to run this command prior to running blockpull:

```
# virsh snapshot-create-as $dom $name -- disk-only
```

2. If the chain looks like this: base ← snap1 ← snap2 ← active run the following:

```
# virsh blockpull $dom $disk snap1
```
This command makes ‘snap1’ the backing file of active, by pulling data from snap2 into active resulting in: base ← snap1 ← active.

3. Once the blockpull is complete, the libvirt tracking of the snapshot that created the extra image in the chain is no longer useful. Delete the tracking on the outdated snapshot with this command:

```
# virsh snapshot-delete $dom $name -- metadata
```

Additional applications of blockpull can be done as follows:

3. To flatten a single image and populate it with data from its backing image chain:

```
# virsh blockpull example-domain vda -- wait
```

3. To flatten part of the backing image chain:

```
# virsh blockpull example-domain vda -- base /path/to/base.img -- wait
```

3. To move the disk image to a new file system on the host:

```
# virsh snapshot-create example-domain -- xmlfile /path/to/new.xml -- disk-only followed by # virsh blockpull example-domain vda -- wait
```

3. To use live migration with post-copy storage migration:

   - On the destination run:

```
# qemu-img create -f qcow2 -o backing_file=/source-host/vm.img /destination-host/vm.qcow2
```

   - On the source run:

```
# virsh migrate example-domain
```

   - On the destination run:

```
# virsh blockpull example-domain vda -- wait
```

E.13.3. Using blockresize to change the size of a domain path

blockresize can be used to re-size a block device of a domain while the domain is running, using the absolute path of the block device which also corresponds to a unique target name (<target dev="name"/> or source file (<source files="name"/>). This can be applied to one of the disk devices attached to domain (you can use the command domblklist to print a table showing the brief information of all block devices associated with a given domain).
Note

Live image re-sizing will always resize the image, but may not immediately be picked up by guests. With recent guest kernels, the size of virtio-blk devices is automatically updated (older kernels require a guest reboot). With SCSI devices, it is required to manually trigger a re-scan in the guest with the command, `echo > /sys/class/scsi_device/0:0:0:0/device/rescan`. In addition, with IDE it is required to reboot the guest before it picks up the new size.

Run the following command: `blockresize [domain] [path size]` where:
- Domain is the unique target name or source file of the domain whose size you want to change
- Path size is a scaled integer which defaults to KiB (blocks of 1024 bytes) if there is no suffix. You must use a suffix of "B" to for bytes.

E.14. Guest CPU model configuration

E.14.1. Introduction

Every hypervisor has its own policy for what a guest will see for its CPUs by default. Whereas some hypervisors decide which CPU host features will be available for the guest, QEMU/KVM presents the guest 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. Such behavior enables the safe migration of guests between hosts, provided they all have physical CPUs that classify into the same group. libvirt does not typically enforce policy itself, rather it provides the mechanism on which the higher layers define their own desired policy. Understanding how to obtain CPU model information and define a suitable guest CPU model is critical to ensure guest migration is successful between hosts. 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.

E.14.2. Learning about the host CPU model

The `virsh capabilities` command displays an XML document describing the capabilities of the hypervisor connection and host. The XML schema displayed has been extended to provide information about the host CPU model. One of the big challenges in describing a CPU model is that every architecture has a different approach to exposing their capabilities. On x86, the capabilities of a modern CPU are exposed via the CPUID instruction. Essentially this comes down to a set of 32-bit integers with each bit given a specific meaning. Fortunately AMD and Intel agree on common semantics for these bits. Other hypervisors expose the notion of CPUID masks directly in their guest configuration format. However, QEMU/KVM supports far more than just the x86 architecture, so CPUID is clearly not suitable as the canonical configuration format. QEMU ended up using a scheme which combines a CPU model name string, with a set of named flags. On x86, the CPU model maps to a baseline CPUID mask, and the flags can be used to then toggle bits in the mask on or off. Libvirt decided to follow this lead and uses a combination of a model name and flags. Here is an example of what libvirt reports as the capabilities on a development workstation:
It is not practical to have a database listing all known CPU models, so libvirt has a small list of baseline CPU model names. It chooses the one that shares the greatest number of CPUID bits with the actual host CPU and then lists the remaining bits as named features. Notice that libvirt does not display which features the baseline CPU contains. This might seem like a flaw at first, but as will be explained in this section, it is not actually necessary to know this information.

E.14.3. Determining a compatible CPU model to suit a pool of hosts

Now that it is possible to find out what CPU capabilities a single host has, the next step is to determine what CPU capabilities are best to expose to the guest. If it is known that the guest will never need to be migrated to another host, the host CPU model can be passed straight through unmodified. A virtualized data center may have a set of configurations that can guarantee all servers will have 100% identical CPUs. Again the host CPU model can be passed straight through unmodified. The more common case, though, is where there is variation in CPUs between hosts. In this mixed CPU environment, the lowest common denominator CPU must be determined. This is not entirely straightforward, so libvirt provides an API for exactly this task. If libvirt is provided a list of XML documents, each describing a CPU model for a host, libvirt will internally convert these to CPUID masks, calculate their intersection, and convert the CPUID mask result back into an XML CPU description. Taking the CPU description from a server:

```
# virsh capabilities
<capabilities>
  <host>
    <uuid>c4a68e53-3f41-6d9e-baaf-d33a181ccfa0</uuid>
    <cpu>
      <arch>x86_64</arch>
      <model>core2duo</model>
      <topology sockets='1' cores='4' threads='1'/>
      <feature name='lahf_lm'/>
      <feature name='sse4.1'/>
      <feature name='xtpr'/>
      <feature name='cx16'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='acpi'/>
      <feature name='ds'/>
    </cpu>
  ... snip ...
</host>
</capabilities>
```

A quick check can be made to see whether this CPU description is compatible with the previous workstation CPU description, using the `virsh cpu-compare` command. To do so, the `virsh capabilities > virsh-caps-workstation-full.xml` command was executed on the workstation. The file `virsh-caps-workstation-full.xml` was edited and reduced to just the following content:

```
# virsh capabilities
<capabilities>
  <host>
    <uuid>8e8e4e67-9df4-9117-bf29-ffc31f6b6abb</uuid>
    <cpu>
      <arch>x86_64</arch>
      <vendor>Intel</vendor>
      <model>Westmere</model>
      <topology sockets='2' cores='4' threads='2'/>
      <feature name='rdtscp'/>
      <feature name='pdenlb'/>
      <feature name='dca'/>
      <feature name='txpr'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='monitor'/>
      <feature name='pbe'/>
      <feature name='te'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='acpi'/>
      <feature name='ds'/>
      <feature name='vme'/>
    </cpu>
  ... snip ...
</host>
</capabilities>
```
The reduced content was stored in a file named `virsh-caps-workstation-cpu-only.xml` and the `virsh cpu-compare` command can be executed using this file:

```
virsh cpu-compare virsh-caps-workstation-cpu-only.xml
```

As seen in this output, libvirt is correctly reporting the CPUs are not strictly compatible, because there are several features in the server CPU that are missing in the workstation CPU. To be able to migrate between the workstation and the server, it will be necessary to mask out some features, but to determine which ones, libvirt provides an API for this, shown via the `virsh cpu-baseline` command:

```
# virsh cpu-baseline virsh-cap-weybridge-strictly-cpu-only.xml
<cpu match='exact'>
  <model>Penryn</model>
  <arch>x86_64</arch>
  <feature policy='require' name='xtpr'/>
  <feature policy='require' name='tm2'/>
  <feature policy='require' name='est'/>
  <feature policy='require' name='vmx'/>
  <feature policy='require' name='ds_cpl'/>
  <feature policy='require' name='monitor'/>
  <feature policy='require' name='pbe'/>
  <feature policy='require' name='tm'/>
  <feature policy='require' name='ht'/>
  <feature policy='require' name='ss'/>
  <feature policy='require' name='acpi'/>
  <feature name='ds'/>
</cpu>
```

Similarly, if the two `<cpu>` elements are put into a single file named `both-cpus.xml`, the following command would generate the same result:

```
# virsh cpu-baseline both-cpus.xml
```

In this case, libvirt has determined that in order to safely migrate a guest between the workstation and the server, it is necessary to mask out 3 features from the XML description for the server, and 3 features from the XML description for the workstation.

### E.14.4. Configuring the guest CPU model

For simple defaults, the guest CPU configuration accepts the same basic XML representation as the host capabilities XML exposes. In other words, the XML from the `cpu-baseline` `virsh` command can now be copied directly into the guest XML at the top level under the `<domain>` element.

As the observant reader will have noticed from the previous XML snippet, there are a few extra attributes available when describing a CPU in the guest XML. These can mostly be ignored, but for the curious here is a quick description of what they do. The top level `<cpu>` element has an attribute called `match` with possible values of:

- `match='minimum'` - the host CPU must have at least the CPU features described in the guest XML. If the host has additional features beyond the guest configuration, these will also be exposed to the guest.
- `match='exact'` - the host CPU must have at least the CPU features described in the guest XML. If the host has additional features beyond the guest configuration, these will be masked out from the guest.
- `match='strict'` - the host CPU must have exactly the same CPU features described in the guest XML.

The next enhancement is that the `<feature>` elements can each have an extra `policy` attribute with possible values of:

- `policy='force'` - expose the feature to the guest even if the host does not have it. This is usually only useful in the case of software emulation.
- `policy='require'` - expose the feature to the guest and fail if the host does not have it. This is the sensible default.
- `policy='optional'` - expose the feature to the guest if it happens to support it.
policy='disable' - if the host has this feature, then hide it from the guest.

policy='forbid' - if the host has this feature, then fail and refuse to start the guest.

The 'forbid' policy is for a niche scenario where an incorrectly functioning application will try to use a feature even if it is not in the CPUID mask, and you wish to prevent accidentally running the guest on a host with that feature. The 'optional' policy has special behavior with respect to migration. When the guest is initially started the flag is optional, but when the guest is live migrated, this policy turns into 'require', since you cannot have features disappearing across migration.

Managing guests with the Virtual Machine Manager (virt-manager)

This section describes the Virtual Machine Manager (virt-manager) windows, dialog boxes, and various GUI controls.

virt-manager provides a graphical view of hypervisors and guests on your host system and on remote host systems. virt-manager can perform virtualization management tasks, including:

- defining and creating guests,
- assigning memory,
- assigning virtual CPUs,
- monitoring operational performance,
- saving and restoring, pausing and resuming, and shutting down and starting guests,
- links to the textual and graphical consoles, and
- live and offline migrations.

F.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 F.1. Starting virt-manager](image)

Alternatively, virt-manager can be started remotely using ssh as demonstrated in the following command:

```bash
ssh -X host's address [remotehost]# virt-manager
```

Using ssh to manage virtual machines and hosts is discussed further in Section 21.1, “Remote management with SSH”.

F.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.
F.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.
F.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.

![Virtual Machine graphical console](image)

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 Fedora. 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 21, 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 Fedora.

F.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 Fedora 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 21, Remote management of guests.
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.

---

### F.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.

2. From the Virtual Machine Manager `Edit` menu, select `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.

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.

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.
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.

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.

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.

F.7. Performance monitoring

Performance monitoring preferences can be modified with **virt-manager**’s preferences window.

To configure performance monitoring:

1. From the **Edit** menu, select **Preferences**.

The **Preferences** window appears.

2. From the **Stats** tab specify the time in seconds or stats polling options.

F.8. Displaying CPU usage for guests

To view the CPU usage for all guests on your system:

1. From the **View** menu, select **Graph**, then the **Guest CPU Usage** check box.
F.9. Displaying CPU usage for hosts

To view the CPU usage for all hosts on your system:

1. From the View menu, select Graph, then the Host CPU Usage check box.

![Host CPU Usage Graph](image)

Figure F.20. Enabling host CPU usage statistics graphing

2. The Virtual Machine Manager shows a graph of host CPU usage on your system.

![Host CPU Usage Graph](image)

Figure F.21. Host CPU usage graph

F.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.
To enable the Disk I/O display, from the View menu, select Graph, then the Disk I/O check box.

The Virtual Machine Manager shows a graph of Disk I/O for all virtual machines on your system.

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.
3. To display the Network I/O statistics, from the View menu, select Graph, then the Network I/O check box.

4. The Virtual Machine Manager shows a graph of Network I/O for all virtual machines on your system.
Guest disk access with offline tools

G.1. Introduction
Fedora comes with tools to access, edit and create guest disks or other disk images. There are several uses for these tools, including:

- Viewing or downloading files located on a guest disk.
- Editing or uploading files onto a guest disk.
- Reading or writing guest configuration.
- Reading or writing the Windows Registry in Windows guests.
- Preparing new disk images containing files, directories, file systems, partitions, logical volumes and other options.
- Rescuing and repairing guests that fail to boot or those that need boot configuration changes.
- Monitoring disk usage of guests.
- Auditing compliance of guests, for example to organizational security standards.
- Deploying guests by cloning and modifying templates.
- Reading CD and DVD ISO and floppy disk images.

**Warning**
You must never use these tools to write to a guest or disk image which is attached to a running virtual machine, not even to open such a disk image in write mode. Doing so will result in disk corruption of the guest. The tools try to prevent you from doing this, however do not catch all cases. If there is any suspicion that a guest might be running, it is strongly recommended that the tools not be used, or at least always use the tools in read-only mode.

G.2. Terminology
This section explains the terms used throughout this chapter.

- libguestfs (GUEST FileSystem LIBrary) - the underlying C library that provides the basic functionality for opening disk images, reading and writing files and so on. You can write C programs directly to this API, but it is quite low level.
- guestfish (GUEST Filesystem Interactive SHell) is an interactive shell that you can use from the command line or from shell scripts. It exposes all of the functionality of the libguestfs API.
- Various virt tools are built on top of libguestfs, and these provide a way to perform specific single tasks from the command line. Tools include virt-df, virt-rescue, virt-resize and virt-edit.
- hivex and Augeas are libraries for editing the Windows Registry and Linux configuration files respectively. Although these are separate from libguestfs, much of the value of libguestfs comes from the combination of these tools.
- guestmount is an interface between libguestfs and FUSE. It is primarily used to mount file systems from disk images on your host. This functionality is not necessary, but can be useful.

G.3. Installation
To install libguestfs, guestfish, the libguestfs tools, guestmount and support for Windows guests, subscribe to the RHEL V2WIN channel, go to the [Fedora Website](https://www.fedoraproject.org) and run the following command:

```bash
# yum install libguestfs guestfish libguestfs-tools
libguestfs-mount libguestfs-winsupport
```

To install every libguestfs-related package including the language bindings, run the following command:

```bash
# yum install "*guestfs*"
```

G.4. The guestfish shell
**guestfish** is an interactive shell that you can use from the command line or from shell scripts to access guest file systems. All of the functionality of the libguestfs API is available from the shell.

To begin viewing or editing a virtual machine disk image, run the following command, substituting the path to your desired disk image:

```bash
guestfish --ro -a /path/to/disk/image
```

`--ro` means that the disk image is opened read-only. This mode is always safe but does not allow write access. Only omit this option when you are certain that the guest is not running, or the disk image is not attached to a live guest. It is not possible to use libguestfs to edit a live guest, and attempting to will assuredly result in irreversible disk corruption.

`/path/to/disk/image` is the path to the disk. This can be a file, a host logical volume (such as `/dev/VG/LV`), a host device (`/dev/cdrom`) or a SAN LUN (`/dev/sdf3`).

**Note**
libguestfs and guestfish do not require root privileges. You only need to run them as root if the disk image being accessed needs root to read and/or write.

When you start guestfish interactively, it will display this prompt:

```
guestfish --ro -a /path/to/disk/image
Welcome to guestfish, the libguestfs filesystem interactive shell for editing virtual machine filesystems.
Type: 'help' for help on commands
'man' to read the manual
'quit' to quit the shell
> <fs>
```

At the prompt, type `run` to initiate the library and attach the disk image. This can take up to 30 seconds the first time it is done. Subsequent starts will complete much faster.

**Note**
libguestfs will use hardware virtualization acceleration such as KVM (if available) to speed up this process.

Once the `run` command has been entered, other commands can be used, as the following section demonstrates.

**G.4.1. Viewing file systems with guestfish**

**G.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:
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:

```bash
> <fs> mount-ro /dev/vda2 /
> <fs> ll /
```

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.

G.4.1.2. Via guestfish inspection

Instead of listing and mounting file systems by hand, it is possible to let guestfish itself inspect the image and mount the file systems as they would be in the guest. To do this, add the `-i` option on the command line:

```bash
<fs> ll /
```

Because guestfish needs to start up the libguestfs back end in order to perform the inspection and mounting, the `run` command is not necessary when using the `-i` option. The `-i` option works for many common Linux and Windows guests.

G.4.1.3. Accessing a guest by name

A guest can be accessed from the command line when you specify its name as known to libvirt (in other words, as it appears in `virsh list -all`). Use the `-d` option to access a guest by its name, with or without the `-i` option:

```bash
guestfish --ro -a /path/to/disk/image -i
```

To exit from the guestfish shell, type `exit` or enter Ctrl+d.
G.4.2. Modifying files with guestfish

To modify files, create directories or make other changes to a guest, first heed the warning at the beginning of this section: your guest must be shut down. Editing or changing a running disk with guestfish will result in disk corruption. This section gives an example of editing the /boot/grub/grub.conf file. When you are sure the guest is shut down you can omit the --ro flag in order to get write access via a command such as:

```
guestfish -d RHEL3 -i
```

Welcome to guestfish, the libguestfs filesystem interactive shell for editing virtual machine filesystems.

Type: 'help' for help on commands
'man' to read the manual
'quit' to quit the shell

Operating system: Red Hat Enterprise Linux AS release 3 (Taroon Update 9)

```
/dev/vda2 mounted on /
/dev/vda1 mounted on /boot
```

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.

G.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, parted, lvresize, lvcreate, vgcreate and pvcreate.

G.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 "$1" -i -i

write /etc/motd "Welcome to Acme Incorporated."
chmod 0644 /etc/motd
```

G.4.5. Augeas and libguestfs scripting

Combining libguestfs with Augeas can help when writing scripts to manipulate Linux guest configuration. For example, the following script uses Augeas to parse the keyboard configuration of a guest, and to print out the layout. Note that this example only works with guests running Red Hat Enterprise Linux:

```
#!/bin/bash -
set -e

guestname="$1"

guestfish -d "$1" -i --ro

aug-init / 0
aug-get /files/etc/sysconfig/keyboard/LAYOUT
```

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 --ro

aug-init / 0
aug-get /files/etc/sysconfig/keyboard/LAYOUT "gb"
aug-save
```

Note the three changes between the two scripts:

1. The --ro option has been removed in the second example, giving the ability to write to the guest.
2. The aug-get command has been changed to aug-set to modify the value instead of fetching it. The new value will be "gb" (including the quotes).
3. The aug-save command is used here so Augeas will write the changes out to disk.
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).

G.5. Other commands

This section describes tools that are simpler equivalents to using guestfish to view and edit guest disk images.

1. **virt-cat** is similar to the guestfish download command. It downloads and displays a single file to the guest. For example:

```
# virt-cat RHEL3 /etc/ntp.conf | grep ^server
server     127.127.1.0       # local clock
```

2. **virt-edit** is similar to the guestfish edit command. It can be used to interactively edit a single file within a guest. For example, you may need to edit the grub.conf file in a Linux-based guest that will not boot:

```
# virt-edit LinuxGuest /boot/grub/grub.conf
```

**virt-edit** has another mode where it can be used to make simple non-interactive changes to a single file. For this, the -e option is used. This command, for example, changes the root password in a Linux guest to having no password:

```
# virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/
```

3. **virt-ls** is similar to the guestfish ls, ll and find commands. It is used to list a directory or directories (recursively). For example, the following command would recursively list files and directories under /home in a Linux guest:

```
# virt-ls -R LinuxGuest /home/ | less
```

G.6. virt-rescue: The rescue shell

G.6.1. Introduction

This section describes **virt-rescue**, which can be considered analogous to a rescue CD for virtual machines. It boots a guest into a rescue shell so that maintenance can be performed to correct errors and the guest can be repaired.

There is some overlap between virt-rescue and guestfish. It is important to distinguish their differing uses. virt-rescue is for making interactive, ad-hoc changes using ordinary Linux file system tools. It is particularly suited to rescuing a guest that has gone wrong. virt-rescue cannot be scripted.

In contrast, guestfish is particularly useful for making scripted, structured changes through a formal set of commands (the libguestfs API), although it can also be used interactively.

G.6.2. Running virt-rescue

Before you use **virt-rescue** on a guest, make sure the guest is not running, otherwise disk corruption will occur. When you are sure the guest is not live, enter:

```
virt-rescue GuestName
```
(or where GuestName is the guest name as known to libvirt), or:

```
virt-rescue /path/to/disk/image
```

(where the path can be any file, any logical volume, LUN, or so on) containing a guest disk.

You will first see output scroll past, as virt-rescue boots the rescue VM. In the end you will see:
Welcome to virt-rescue, the libguestfs rescue shell.

Note: The contents of / are the rescue appliance.
You have to mount the guest's partitions under /sysroot before you can examine them.

bash: cannot set terminal process group (-1):
Inappropriate ioctl for device
bash: no job control in this shell

The shell prompt here is an ordinary bash shell, and a reduced set of ordinary Fedora commands is available. For example, you can enter:

```bash
<rescue>
```

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:

```bash
<rescue> mount /dev/vda1 /sysroot/
EXT4-fs (vda1): mounted filesystem with ordered data mode. Opts: (null)
```

When you are finished rescuing the guest, exit the shell by entering `exit` or `Ctrl+d`.

virt-rescue has many command line options. The options most often used are:

- `--ro`: Operate in read-only mode on the guest. No changes will be saved. You can use this to experiment with the guest. As soon as you exit from the shell, all of your changes are discarded.
- `--network`: Enable network access from the rescue shell. Use this if you need to, for example, download RPM or other files into the guest.

## G.7. virt-df: Monitoring disk usage

### G.7.1. Introduction

This section describes `virt-df`, which displays file system usage from a disk image or a guest. It is similar to the Linux `df` command, but for virtual machines.

### G.7.2. Running virt-df

To display file system usage for all file systems found in a disk image, enter the following:

```bash
# virt-df /dev/vg_guests/RHEL6
```

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>1K-blocks</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL6:/dev/sda1</td>
<td>181086</td>
<td>10233</td>
</tr>
<tr>
<td>85634</td>
<td>11%</td>
<td></td>
</tr>
<tr>
<td>RHEL6:/dev/VolGroup00/LogVol00</td>
<td>7127864</td>
<td>2272744</td>
</tr>
<tr>
<td>4493036</td>
<td>32%</td>
<td></td>
</tr>
</tbody>
</table>

(Where `/dev/vg_guests/RHEL6` is a Red Hat Enterprise Linux 4 guest disk image. The path in this case is the host logical volume where this disk image is located.)

You can also use `virt-df` on its own to list information about all of your guests (i.e. those known to libvirt). The `virt-df` command recognizes some of the same options as the standard `df` such as `-h` (human-readable) and `-i` (show inodes instead of blocks).

`virt-df` also works on Windows guests:
```bash
# 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       3.9G  4.4G   41%
RHEL6brewx64:/dev/sda1         484.2M      52.6M
RHEL6brewx64:/dev/vg_rhel6brewx64/lv_root  11.3G  3.4G
9.2G  26%  9.2G  26%
Win7x32:/dev/sda1              108.0M  24.1M  79.9M   25%
Win7x32:/dev/sda2              19.9G      7.4G  12.5G  38%
```

**Note**

You can use `virt-df` safely on live guests, since it only needs read-only access. However, you should not expect the numbers to be precisely the same as those from a `df` command running inside the guest. This is because what is on disk will be slightly out of synch with the state of the live guest. Nevertheless it should be a good enough approximation for analysis and monitoring purposes.

`virt-df` is designed to allow you to integrate the statistics into monitoring tools, databases and so on. This allows system administrators to generate reports on trends in disk usage, and alerts if a guest is about to run out of disk space. To do this you should use the `--csv` option to generate machine-readable Comma-Separated-Values (CSV) output. CSV output is readable by most databases, spreadsheet software and a variety of other tools and programming languages. The raw CSV looks like the following:

```bash
# 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/](http://virt-tools.org/learning/advanced-virt-df/).

### G.8. virt-resize: resizing guests offline

#### G.8.1. Introduction

This section describes `virt-resize`, a tool for expanding or shrinking guests. It only works for guests which are offline (shut down). It works by copying the guest image and leaving the original disk image untouched. This is ideal because you can use the original image as a backup, however there is a trade-off as you need twice the amount of disk space.

#### G.8.2. Expanding a disk image

This section demonstrates a simple case of expanding a disk image:

1. Locate the disk image to be resized. You can use the command `virsh dumpxml GuestName` for a libvirt guest.
2. Decide on how you wish to expand the guest. Run `virt-df -h` and `virt-list-partitions -lh` on the guest disk, as shown in the following output:

```bash
# 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 161.9M
/dev/sda2 pv 7.9G
```

This example will demonstrate how to:

- Increase the size of the first (boot) partition, from approximately 100MB to 500MB.
- Increase the total disk size from 8GB to 16GB.
- Expand the second partition to fill the remaining space.
- Expand `/dev/VolGroup00/LogVol00` to fill the new space in the second partition.

1. Make sure the guest is shut down.
2. Rename the original disk as the backup. How you do this depends on the host storage environment for the original disk. If it is stored as a file, use the `mv` command. For logical volumes (as demonstrated in this example), use `lvrename`:

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. Create the new disk. The requirements in this example are to expand the total disk size up to 16GB. Since logical volumes are used here, the following command is used:

```
# lvcreate -l 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. The requirements from step 2 are expressed by this command:

```
# virt-resize /dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 --resize /dev/sda1=500M --expand /dev/sda2 --LV-expand /dev/VolGroup00/LogVol00
```

The first two arguments are the input disk and output disk. `--resize /dev/sda1=500M` resizes the first partition up to 500MB, `--expand /dev/sda2` expands the second partition to fill all remaining space, `--LV-expand /dev/VolGroup00/LogVol00` expands the guest logical volume to fill the extra space in the second partition.

`virt-resize` describes what it is doing in the output:

```
Summary of changes:
/dev/sda1: partition will be resized from 101.9M to 500.0M
/dev/sda1: content will be expanded using the 'resize2fs' method
/dev/sda2: partition will be resized from 7.9G to 15.5G
/dev/sda2: content will be expanded using the 'pvresize' method
/dev/VolGroup00/LogVol00: LV will be expanded to maximum size
/dev/VolGroup00/LogVol00: content will be expanded using the 'resize2fs' method

Copying /dev/sda1 ... 
[.................................................................]
Copying /dev/sda2 ...
[.................................................................]
Expanding /dev/sda1 using the 'resize2fs' method
Expanding /dev/sda2 using the 'pvresize' method
Expanding /dev/VolGroup00/LogVol00 using the 'resize2fs' method
```

5. Try to boot the virtual machine. If it works (and after testing it thoroughly) you can delete the backup disk. If it fails, shut down the virtual machine, delete the new disk, and rename the backup disk back to its original name.

6. Use `virt-df` and/or `virt-list-partitions` to show the new size:

```
# virt-df -h /dev/vg_pin/RHEL6
Filesystem                  Size
RHEL6:/dev/sda1              484.4M
10.8M   448.6M    3%
RHEL6:/dev/VolGroup00/LogVol00 14.3G
2.2G   11.4G   16%
```

Resizing guests is not an exact science. If `virt-resize` fails, there are a number of tips that you can review and attempt in the `virt-resize(1)` man page. For some older Red Hat Enterprise Linux guests, you may need to pay particular attention to the tip regarding GRUB.

G.9. virt-inspector: inspecting guests

G.9.1. Introduction

`virt-inspector` is a tool for inspecting a disk image to find out what operating system it contains.

G.9.2. Installation

To install `virt-inspector` and the documentation, enter the following command:

```
# virt-df -h /dev/vg_pin/RHEL6
Filesystem                  Size
RHEL6:/dev/sda1              484.4M
10.8M   448.6M    3%
RHEL6:/dev/VolGroup00/LogVol00 14.3G
2.2G   11.4G   16%
```
G.9.3. Running virt-inspector

You can run `virt-inspector` against any disk image or libvirt guest as shown in the following example:

```
virt-inspector --xml disk.img > report.xml
```

Or as shown here:

```
virt-inspector --xml GuestName > report.xml
```

The result will be an XML report (`report.xml`). The main components of the XML file are a top-level `<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 installed -->
    <filesystem dev="/dev/VolGroup/lv_root">
      <label></label>
      <uuid>b24d9161-5613-4ab8-8649-f27a8a868d3f</uuid>
      <type>ext4</type>
      <content>linux-root</content>
      </filesystem>
    <filesystem dev="/dev/VolGroup/lv_swap">
      <type>swap</type>
      <spec>/dev/mapper/VolGroup-lv_swap</spec>
      </filesystem>
    <!-- packages installed -->
    <applications>
      <application name="firefox">
        <version>3.5.5</version>
        <release>1.fc12</release>
      </application>
    </applications>
  </operatingsystem>
</operatingsystems>
```

Processing these reports is best done using W3C standard XPath queries. Fedora 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...
G.10. virt-win-reg: Reading and editing the Windows Registry

G.10.1. Introduction

virt-win-reg is a tool that manipulates the Registry in Windows guests. It can be used to read out registry keys. You can also use it to make changes to the Registry, but you must never try to do this for live/running guests, as it will result in disk corruption.

G.10.2. Installation

To use virt-win-reg you must run the following:

```bash
# yum install libguestfs-tools libguestfs-winsupport
```

G.10.3. Using virt-win-reg

To read out Registry keys, specify the name of the guest (or its disk image) and the name of the Registry key. You must use single quotes to surround the name of the desired key:

```bash
# virt-win-reg WindowsGuest \ 'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current Version\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
perl -MEncode -pe 's?hex\((\d+)\):(.+)?$t=$1;$_=$2;s,\",\"\",g;"str($t):".".decode(utf16le=>pack("H",$_))."\""?
```

To merge changes into the Windows Registry of an offline guest, you must first prepare a .REG file. There is a great deal of documentation about doing this available from MSDN, and there is a good summary in the following Wikipedia page:


When you have prepared a .REG file, enter the following:

```bash
# virt-win-reg --merge WindowsGuest input.reg
```

This will update the registry in the guest.

G.11. Using the API from Programming Languages

The libguestfs API can be used directly from the following languages in Fedora 19: C, C++, Perl, Python, Java, Ruby and OCaml.

- To install C and C++ bindings, enter the following command:
  ```bash
  # yum install libguestfs-devel
  ```

- To install Perl bindings:
  ```bash
  # yum install 'perl(Sys::Guestfs)'
  ```

- To install Python bindings:
  ```bash
  # yum install python-libguestfs
  ```

- To install Java bindings:
  ```bash
  # yum install libguestfs-java libguestfs-java-devel
  ```

- To install Ruby bindings:
  ```bash
  # yum install ruby-libguestfs
  ```

- To install OCaml bindings:
  ```bash
  # yum install ocaml-libguestfs ocaml-libguestfs-devel
  ```
The binding for each language is essentially the same, but with minor syntactic changes. A C statement:

```c
guestfs_launch (g);
```

Would appear like the following in Perl:

```perl
$g->launch ()
```

Or like the following in OCaml:

```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 (e.g. a live VM). Doing this will cause disk corruption.
- Opening a read-only handle on a disk image which is currently in use (e.g. 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.

**G.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:

```c
#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:

```bash
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:
```c
#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
     * automatically synchronized
     * 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);
    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.

## G.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 =====
```

## G.13. Where to find further documentation

The primary source for documentation for libguestfs and the tools are the Unix man pages. The API is documented in guestfs(3). guestfish is documented in guestfish(1). The virt tools are documented in their own man pages (eg. virt-df(1)).

### Virtual Networking

This chapter introduces the concepts needed to create, start, stop, remove, and modify virtual networks with libvirt.

Additional information can be found in the libvirt reference chapter
H.1. Virtual network switches

Libvirt virtual networking uses the concept of a virtual network switch. A virtual network switch is a software construct that operates on a host server, to which virtual machines (guests) connect. The network traffic for a guest is directed through this switch:

Linux host 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`.

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
        RX packets:0 errors:0 dropped:0 overruns: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 scope global virbr0
```

H.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 IP address for communication to any external network. By default, computers that are placed externally to the host cannot communicate to the guests inside when the virtual network switch is operating in NAT mode, as shown in the following diagram:
Figure H.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.

H.3. Networking protocols

The following sections describe individual networking protocols and how they are used in libvirt.

H.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 H.4. Virtual network switch running dnsmasq

H.3.2. Routed mode

When using **routed mode**, the virtual switch connects to the physical LAN connected to the host, passing traffic back and forth without the use of NAT. The virtual switch can examine all traffic and use the information contained within the network packets to make routing decisions. When using this mode, all of the virtual machines are in their own subnet, routed through a virtual switch. This situation is not always ideal as no other hosts on the physical network are aware of the virtual machines without manual physical router configuration, and cannot access the virtual machines. Routed mode operates at Layer 3 of the OSI networking model.

Figure H.5. Virtual network switch in routed mode
### H.3.3. Isolated mode

When using *isolated mode*, guests connected to the virtual switch can communicate with each other, and with the host, but their traffic will not pass outside of the host, nor can they receive traffic from outside the host. Using dnsmasq in this mode is required for basic functionality such as DHCP. However, even if this network is isolated from any physical network, DNS names are still resolved. Therefore a situation can arise when DNS names resolve but ICMP echo request (ping) commands fail.

![Figure H.6. Virtual network switch in isolated mode](image)

### H.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. The following image demonstrates this default configuration for *libvirtd*:

![Figure H.7. Default libvirt network configuration](image)

**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.

### H.5. Examples of common scenarios

This section demonstrates different virtual networking modes and provides some example scenarios.

#### H.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:

![Sample DMZ configuration diagram](image)

Hosts in a DMZ typically provide services to WAN (external) hosts as well as LAN (internal) hosts. As this requires them to be accessible from multiple locations, and considering that these locations are controlled and operated in different ways based on their security and trust level, routed mode is the best configuration for this environment.

**Virtual Server hosting**

Consider a virtual server hosting company that has several hosts, each with two physical network connections. One interface is used for management and accounting, the other is for the virtual machines to connect through. Each guest has its own public IP address, but the hosts use private IP address as management of the guests can only be performed by internal administrators. Refer to the following diagram to understand this scenario:

![Virtual server hosting sample configuration](image)

When the host 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. The following diagram demonstrates this:

![Virtual server using static IP addresses](image)

**H.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.

**H.5.3. Isolated mode**

Isolated mode allows virtual machines to communicate with each other only. They are unable to interact with the physical network.
H.6. Managing a virtual network

To configure a virtual network on your system:

1. From the Edit menu, select Connection Details.

   ![Figure H.11. Selecting a host's details](image)

   This will open the Connection Details menu. Click the Virtual Networks tab.

   ![Figure H.12. Virtual network configuration](image)

2. 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.

H.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 H.6, "Managing a virtual network".

   ![Figure H.13. Virtual network configuration](image)

   This will open the Create a new virtual network window. Click Forward to continue.
2. Enter an appropriate name for your virtual network and click Forward.

3. Enter an IPv4 address space for your virtual network and click Forward.

4. Define the DHCP range for your virtual network by specifying a Start and End range of IP addresses. Click Forward to continue.
5. Select how the virtual network should connect to the physical network.

![Selecting the DHCP range](image)

**Selecting the DHCP range**

Select how the virtual network should connect to the physical network.

- Enable DHCP
  - Start: 192.168.100.128
  - End: 192.168.100.254

**Tip:** Unless you wish to reserve some addresses to allow static network configuration in virtual machines, these parameters can be left with their default values.

Click **Forward** to continue.

6. You are now ready to create the network. Check the configuration of your network and click **Finish**.

![Connecting to physical network](image)

**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.

7. The new virtual network is now available in the **Virtual Networks** tab of the **Connection Details** window.
H.8. Attaching a virtual network to a guest

To attach a virtual network to a guest:

1. In the Virtual Machine Manager window, highlight the guest that will have the network assigned.

![Figure H.20. Selecting a virtual machine to display](image)

2. From the Virtual Machine Manager Edit menu, select Virtual Machine Details.

![Figure H.21. Displaying the virtual machine details](image)

3. Click the Add Hardware button on the Virtual Machine Details window.

![Figure H.22. The Virtual Machine Details window](image)

4. In the Add new virtual hardware window, select Network from the left pane, and select your network name (network1 in this example) from the Host device menu and click Finish.
H.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. 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 as where the packet originates from are sent back to the host by the VEPA capable bridge (today's bridges are typically not VEPA capable).

**bridge**

Packets whose destination is on the same host 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 if they are sent through an external router or gateway and that device sends them back to the host. 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 directly attached guest virtual machines can be managed by the hardware switch to which the physical interface of the host 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="1193847" typeidversion="2" instanceid="09b11c53-8b5c-4eeb-8f00-d84eaa5a2a4f"/>
    </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>
```

**H.10. Applying network filtering**
This section provides an introduction to libvirt's network filters, their goals, concepts and XML format.

H.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 when a virtual machine is started. Since the filtering rules cannot be circumvented from within the virtual machine, they make them mandatory from the point of view of a virtual machine user.

From the point of view of the guest, 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 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 H.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.

```xml
<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 H.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.

```xml
<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.

H.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 H.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-f1d3aa4bc95</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'/>;
</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.

H.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>
<thead>
<tr>
<th>Chain (prefix)</th>
<th>Default priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>stp</td>
<td>-810</td>
</tr>
<tr>
<td>mac</td>
<td>-800</td>
</tr>
<tr>
<td>vlan</td>
<td>-750</td>
</tr>
<tr>
<td>ipv4</td>
<td>-700</td>
</tr>
<tr>
<td>ipv6</td>
<td>-600</td>
</tr>
<tr>
<td>arp</td>
<td>-500</td>
</tr>
<tr>
<td>rarp</td>
<td>-400</td>
</tr>
</tbody>
</table>
A chain with a lower priority value is accessed before one with a higher value.

The chains listed in Table H.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 H.10.2, "Filtering chains" filter shows the default priority of -500 for arp chains, for example.

H.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 H.10.12, "Limitations," on how to use this feature and what to expect when using it. The XML file shown in Section H.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.

Example H.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:

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 following rule, three individual filtering rules (one for each IP address) will be created:

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.

Fedora Draft Documentation UNUSED. Virtualization Deployment and Administration Guide 247
Example H.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 SRCIPADDRESSES. The rule generates all combinations of elements of the variable DSTPORTS with those of SRCIPADDRESSES by using two independent iterators to access their elements.

```
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]'/>
</rule>
```

Assign concrete values to SRCIPADDRESSES and DSTPORTS as shown:

<table>
<thead>
<tr>
<th>SRCIPADDRESSES</th>
<th>DSTPORTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10.0.0.1, 11.1.2.3]</td>
<td>[80, 8080]</td>
</tr>
</tbody>
</table>

Assigning values to the variables using $SRCIPADDRESSES[@1]$ 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 $SRCIPADDRESSES[@1]$ and $DSTPORTS[@1]$, would result in parallel access to both lists and result in the following combinations:

| 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.

H.10.5. Automatic IP address detection and DHCP snooping

H.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 TRL_IP_LEARNING is not set. This method will only detect a single IP address per interface. Once a guest’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, which would be considered IP address spoofing. When a guest is migrated to another host or resumed after a suspend operation, the first packet sent by the guest will again determine the IP address that the guest can use on a particular interface.

The value 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 resumes after a suspend operation, any valid IP address leases are applied to its filters. Otherwise the guest is expected to use DHCP to obtain a new IP addresses. When a guest migrates to another physical host, the guest 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.

H.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 will no longer be able to use the IP address until it acquires a new, valid lease from a DHCP server. If the guest 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 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 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 guests in the infrastructure to avoid them being able to send DHCP packets. Therefore guests 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 guests 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 guests in the subnet.

**Example H.6. Activating IPs for DHCP snooping**
The following XML provides an example for the activation of IP address learning using the DHCP snooping method:

```xml
<interface type='bridge'>
  <source bridge='virbr0'/>
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp'/>
  </filterref>
</interface>
```

**H.10.6. Reserved Variables**

Table H.2, "Reserved variables" shows the variables that are considered reserved and are used by libvirt:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>The MAC address of the interface</td>
</tr>
<tr>
<td>IP</td>
<td>The list of IP addresses in use by an interface</td>
</tr>
<tr>
<td>IPV6</td>
<td>Not currently implemented: the list of IPv6 addresses in use by an interface</td>
</tr>
<tr>
<td>DHCPSERVER</td>
<td>The list of IP addresses of trusted DHCP servers</td>
</tr>
<tr>
<td>DHCPSERVERV6</td>
<td>Not currently implemented: The list of IPv6 addresses of trusted DHCP servers</td>
</tr>
<tr>
<td>CTRL_IP_LEARNING</td>
<td>The choice of the IP address detection mode</td>
</tr>
</tbody>
</table>

**H.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. Currently the system only supports the chains root, ipv4, ipv6, arp and rarp.

**H.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 H.7. An Example of a clean traffic filter

The following shows the XML of the clean-traffic network filter referencing several other filters.

```xml
<filter name='clean-traffic'>
  <uuid>6ef53069-ba34-94a0-d33d-177519b8cc6c</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-l2-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.

H.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 H.8. Example of network traffic filtering

```xml
<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 and 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 and 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 H.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 H.10.11, "Advanced Filter Configuration Topics".

The above example 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.

H.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 dotted decimal format (255.255.255.0) 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.

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`.

### H.10.10.1. MAC (Ethernet)

**Protocol ID:** mac

Rules of this type should go into the root chain.

#### Table H.3. MAC protocol types

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of sender</td>
</tr>
<tr>
<td>dstmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of destination</td>
</tr>
<tr>
<td>dstmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of destination</td>
</tr>
<tr>
<td>protocolid</td>
<td>UINT16 (0x600-0xffff), STRING</td>
<td>Layer 3 protocol ID. Valid strings include [arp, rarp, ipv4, ipv6]</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>Text string up to 256 characters</td>
</tr>
</tbody>
</table>

The filter can be written as such:

```
[...]
<mac match='no' srcmacaddr='$MAC'/>
[...]
```

### H.10.10.2. VLAN (802.1Q)

**Protocol ID:** vlan

Rules of this type should go either into the root or vlan chain.
### Table H.4. VLAN protocol types

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of sender</td>
</tr>
<tr>
<td>dstmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of destination</td>
</tr>
<tr>
<td>dstmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of destination</td>
</tr>
<tr>
<td>vlan-id</td>
<td>UINT16 (0x0-0xfff, 0 - 4095)</td>
<td>VLAN ID</td>
</tr>
<tr>
<td>encap-protocol</td>
<td>UINT16 (0x03c-0xfff), String</td>
<td>Encapsulated layer 3 protocol ID, valid strings are arp, ipv4, ipv6</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>text string up to 256 characters</td>
</tr>
</tbody>
</table>

### H.10.10.3. STP (Spanning Tree Protocol)

Protocol ID: stp

Rules of this type should go either into the root or stp chain.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of sender</td>
</tr>
<tr>
<td>type</td>
<td>UINT8</td>
<td>Bridge Protocol Data Unit (BPDU) type</td>
</tr>
<tr>
<td>flags</td>
<td>UINT8</td>
<td>BPDU flagdstmacmask</td>
</tr>
<tr>
<td>root-priority</td>
<td>UINT16</td>
<td>Root priority range start</td>
</tr>
<tr>
<td>root-priority-hi</td>
<td>UINT16 (0x0-0xfff, 0 - 4095)</td>
<td>Root priority range end</td>
</tr>
<tr>
<td>root-address</td>
<td>MAC_ADDRESS</td>
<td>root MAC Address</td>
</tr>
<tr>
<td>root-address-mask</td>
<td>MAC_MASK</td>
<td>root MAC Address mask</td>
</tr>
<tr>
<td>root-cost</td>
<td>UINT32</td>
<td>Root path cost (range start)</td>
</tr>
<tr>
<td>root-cost-hi</td>
<td>UINT32</td>
<td>Root path cost range end</td>
</tr>
<tr>
<td>sender-priority-hi</td>
<td>UINT16</td>
<td>Sender priority range end</td>
</tr>
<tr>
<td>sender-address</td>
<td>MAC_ADDRESS</td>
<td>BPDU sender MAC address</td>
</tr>
<tr>
<td>sender-address-mask</td>
<td>MAC_MASK</td>
<td>BPDU sender MAC address mask</td>
</tr>
<tr>
<td>port</td>
<td>UINT16</td>
<td>Port identifier (range start)</td>
</tr>
<tr>
<td>port_hi</td>
<td>UINT16</td>
<td>Port identifier range end</td>
</tr>
<tr>
<td>msg-age</td>
<td>UINT16</td>
<td>Message age timer (range start)</td>
</tr>
<tr>
<td>msg-age-hi</td>
<td>UINT16</td>
<td>Message age timer range end</td>
</tr>
<tr>
<td>max-age-hi</td>
<td>UINT16</td>
<td>Maximum age timer range end</td>
</tr>
<tr>
<td>hello-time</td>
<td>UINT16</td>
<td>Hello time timer (range start)</td>
</tr>
<tr>
<td>hello-time-hi</td>
<td>UINT16</td>
<td>Hello time timer range end</td>
</tr>
<tr>
<td>forward-delay</td>
<td>UINT16</td>
<td>Forward delay (range start)</td>
</tr>
<tr>
<td>forward-delay-hi</td>
<td>UINT16</td>
<td>Forward delay range end</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>text string up to 256 characters</td>
</tr>
</tbody>
</table>

### H.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 H.6. ARP and RARP protocol types

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of sender</td>
</tr>
<tr>
<td>dstmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of destination</td>
</tr>
<tr>
<td>dstmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of destination</td>
</tr>
<tr>
<td>hwtype</td>
<td>UINT16</td>
<td>Hardware type</td>
</tr>
<tr>
<td>protocoltype</td>
<td>UINT16</td>
<td>Protocol type</td>
</tr>
<tr>
<td>opcode</td>
<td>UINT16, STRING</td>
<td>Opcode valid strings are: Request, Reply, Request_Reverse, Reply_Reverse, DRARP_Request, DRARP_Reply, DRARP_Error, InARP_Request, ARP_NAK</td>
</tr>
<tr>
<td>arpsrcmacaddr</td>
<td>MAC_ADDR</td>
<td>Source MAC address in ARP/RARP packet</td>
</tr>
<tr>
<td>arpdstmacaddr</td>
<td>MAC_ADDR</td>
<td>Destination MAC address in ARP/RARP packet</td>
</tr>
<tr>
<td>arpsrcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address in ARP/RARP packet</td>
</tr>
<tr>
<td>arpdstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address in ARP/RARP packet</td>
</tr>
<tr>
<td>gratuitous</td>
<td>BOOLEAN</td>
<td>Boolean indicating whether to check for a gratuitous ARP packet</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>Text string up to 256 characters</td>
</tr>
</tbody>
</table>

### H.10.10.5. IPv4

Protocol ID: ip

Rules of this type should either go into the root or ipv4 chain.

### Table H.7. IPv4 protocol types

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of sender</td>
</tr>
<tr>
<td>dstmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of destination</td>
</tr>
<tr>
<td>dstmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of destination</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>protocol</td>
<td>UINT8, STRING</td>
<td>Layer 4 protocol identifier. Valid strings for protocol are: tcp, udp, udplite, esp, ah, icmp, igmp, sctp</td>
</tr>
<tr>
<td>srcportstart</td>
<td>UINT16</td>
<td>Start of range of valid source ports; requires protocol</td>
</tr>
<tr>
<td>srcportend</td>
<td>UINT16</td>
<td>End of range of valid source ports; requires protocol</td>
</tr>
<tr>
<td>dstportstart</td>
<td>UINT16</td>
<td>Start of range of valid destination ports; requires protocol</td>
</tr>
<tr>
<td>dstportend</td>
<td>UINT16</td>
<td>End of range of valid destination ports; requires protocol</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>Text string up to 256 characters</td>
</tr>
</tbody>
</table>

### H.10.10.6. IPv6
Protocol ID: ipv6

Rules of this type should either go into the root or ipv6 chain.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of sender</td>
</tr>
<tr>
<td>dstmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of destination</td>
</tr>
<tr>
<td>dstmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to MAC address of destination</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>protocol</td>
<td>UINT8, STRING</td>
<td>Layer 4 protocol identifier. Valid strings for protocol are: tcp, udp, udp-lite, esp, ah, icmpv6, scvp</td>
</tr>
<tr>
<td>scrportstart</td>
<td>UNIT16</td>
<td>Start of range of valid source ports; requires protocol</td>
</tr>
<tr>
<td>srcportend</td>
<td>UNIT16</td>
<td>End of range of valid source ports; requires protocol</td>
</tr>
<tr>
<td>dstportstart</td>
<td>UNIT16</td>
<td>Start of range of valid destination ports; requires protocol</td>
</tr>
<tr>
<td>dstportend</td>
<td>UNIT16</td>
<td>End of range of valid destination ports; requires protocol</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>Text string up to 256 characters</td>
</tr>
</tbody>
</table>

H.10.10.7. TCP/UDP/SCTP

Protocol ID: tcp, udp, scvp

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>scripto</td>
<td>IP_ADDR</td>
<td>Start of range of source IP address</td>
</tr>
<tr>
<td>srcipfrom</td>
<td>IP_ADDR</td>
<td>End of range of source IP address</td>
</tr>
<tr>
<td>dstipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of destination IP address</td>
</tr>
<tr>
<td>dstip</td>
<td>IP_ADDR</td>
<td>End of range of destination IP address</td>
</tr>
<tr>
<td>scrportstart</td>
<td>UNIT16</td>
<td>Start of range of valid source ports; requires protocol</td>
</tr>
<tr>
<td>srcportend</td>
<td>UNIT16</td>
<td>End of range of valid source ports; requires protocol</td>
</tr>
<tr>
<td>dstportstart</td>
<td>UNIT16</td>
<td>Start of range of valid destination ports; requires protocol</td>
</tr>
<tr>
<td>dstportend</td>
<td>UNIT16</td>
<td>End of range of valid destination ports; requires protocol</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>Text string up to 256 characters</td>
</tr>
<tr>
<td>state</td>
<td>STRING</td>
<td>Comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE</td>
</tr>
<tr>
<td>flags</td>
<td>STRING</td>
<td>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</td>
</tr>
<tr>
<td>ipset</td>
<td>STRING</td>
<td>The name of an IPSet managed outside of libvirt</td>
</tr>
<tr>
<td>ipsetflags</td>
<td>IPSETFLAGS</td>
<td>Flags for the IPSet; requires ipset attribute</td>
</tr>
</tbody>
</table>
H.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 H.10. ICMP protocol types

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to the MAC address of the sender</td>
</tr>
<tr>
<td>dstmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of the destination</td>
</tr>
<tr>
<td>dstmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to the MAC address of the destination</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>srcipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of source IP address</td>
</tr>
<tr>
<td>srcipto</td>
<td>IP_ADDR</td>
<td>End of range of source IP address</td>
</tr>
<tr>
<td>dstipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of destination IP address</td>
</tr>
<tr>
<td>dstipto</td>
<td>IP_ADDR</td>
<td>End of range of destination IP address</td>
</tr>
<tr>
<td>type</td>
<td>UNIT16</td>
<td>ICMP type</td>
</tr>
<tr>
<td>code</td>
<td>UNIT16</td>
<td>ICMP code</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>text string up to 256 characters</td>
</tr>
<tr>
<td>state</td>
<td>STRING</td>
<td>comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE</td>
</tr>
<tr>
<td>ipset</td>
<td>STRING</td>
<td>The name of an IPSet managed outside of libvirt</td>
</tr>
<tr>
<td>ipsetflags</td>
<td>IPSETFLAGS</td>
<td>flags for the IPSet; requires ipset attribute</td>
</tr>
</tbody>
</table>

H.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 H.11. IGMP, ESP, AH, UDPLITE, 'ALL'

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to the MAC address of the sender</td>
</tr>
<tr>
<td>dstmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of the destination</td>
</tr>
<tr>
<td>dstmacmask</td>
<td>MAC_MASK</td>
<td>Mask applied to the MAC address of the destination</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>srcipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of source IP address</td>
</tr>
<tr>
<td>srcipto</td>
<td>IP_ADDR</td>
<td>End of range of source IP address</td>
</tr>
<tr>
<td>dstipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of destination IP address</td>
</tr>
<tr>
<td>dstipto</td>
<td>IP_ADDR</td>
<td>End of range of destination IP address</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>text string up to 256 characters</td>
</tr>
<tr>
<td>state</td>
<td>STRING</td>
<td>comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE</td>
</tr>
<tr>
<td>ipset</td>
<td>STRING</td>
<td>The name of an IPSet managed outside of libvirt</td>
</tr>
<tr>
<td>ipsetflags</td>
<td>IPSETFLAGS</td>
<td>flags for the IPSet; requires ipset attribute</td>
</tr>
</tbody>
</table>

H.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 H.12. TCP, UDP, SCTP over IPv6 protocol types

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>srcipfrom</td>
<td>IP_ADDR</td>
<td>start of range of source IP address</td>
</tr>
<tr>
<td>scripto</td>
<td>IP_ADDR</td>
<td>end of range of source IP address</td>
</tr>
<tr>
<td>dstipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of destination IP address</td>
</tr>
<tr>
<td>dstipto</td>
<td>IP_ADDR</td>
<td>End of range of destination IP address</td>
</tr>
<tr>
<td>srcportstart</td>
<td>UINT16</td>
<td>Start of range of valid source ports</td>
</tr>
<tr>
<td>srcportend</td>
<td>UINT16</td>
<td>End of range of valid source ports</td>
</tr>
<tr>
<td>dstportstart</td>
<td>UINT16</td>
<td>Start of range of valid destination ports</td>
</tr>
<tr>
<td>dstportend</td>
<td>UINT16</td>
<td>End of range of valid destination ports</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>text string up to 256 characters</td>
</tr>
<tr>
<td>state</td>
<td>STRING</td>
<td>comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE</td>
</tr>
<tr>
<td>ipset</td>
<td>STRING</td>
<td>The name of an IPSet managed outside of libvirt</td>
</tr>
<tr>
<td>ipsetflags</td>
<td>IPSETFLAGS</td>
<td>flags for the IPSet; requires ipset attribute</td>
</tr>
</tbody>
</table>

### H.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>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>srcipfrom</td>
<td>IP_ADDR</td>
<td>start of range of source IP address</td>
</tr>
<tr>
<td>scripto</td>
<td>IP_ADDR</td>
<td>end of range of source IP address</td>
</tr>
<tr>
<td>dstipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of destination IP address</td>
</tr>
<tr>
<td>dstipto</td>
<td>IP_ADDR</td>
<td>End of range of destination IP address</td>
</tr>
<tr>
<td>type</td>
<td>UINT16</td>
<td>ICMPv6 type</td>
</tr>
<tr>
<td>code</td>
<td>UINT16</td>
<td>ICMPv6 code</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>text string up to 256 characters</td>
</tr>
<tr>
<td>state</td>
<td>STRING</td>
<td>comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE</td>
</tr>
<tr>
<td>ipset</td>
<td>STRING</td>
<td>The name of an IPSet managed outside of libvirt</td>
</tr>
<tr>
<td>ipsetflags</td>
<td>IPSETFLAGS</td>
<td>flags for the IPSet; requires ipset attribute</td>
</tr>
</tbody>
</table>

### H.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>
<thead>
<tr>
<th>Attribute Name</th>
<th>Datatype</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcmacaddr</td>
<td>MAC_ADDR</td>
<td>MAC address of sender</td>
</tr>
<tr>
<td>srcipaddr</td>
<td>IP_ADDR</td>
<td>Source IP address</td>
</tr>
<tr>
<td>srcipmask</td>
<td>IP_MASK</td>
<td>Mask applied to source IP address</td>
</tr>
<tr>
<td>dstipaddr</td>
<td>IP_ADDR</td>
<td>Destination IP address</td>
</tr>
<tr>
<td>dstipmask</td>
<td>IP_MASK</td>
<td>Mask applied to destination IP address</td>
</tr>
<tr>
<td>srcipfrom</td>
<td>IP_ADDR</td>
<td>start of range of source IP address</td>
</tr>
<tr>
<td>scripto</td>
<td>IP_ADDR</td>
<td>end of range of source IP address</td>
</tr>
<tr>
<td>dstipfrom</td>
<td>IP_ADDR</td>
<td>Start of range of destination IP address</td>
</tr>
<tr>
<td>dstipto</td>
<td>IP_ADDR</td>
<td>End of range of destination IP address</td>
</tr>
<tr>
<td>comment</td>
<td>STRING</td>
<td>text string up to 256 characters</td>
</tr>
<tr>
<td>state</td>
<td>STRING</td>
<td>comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE</td>
</tr>
<tr>
<td>ipset</td>
<td>STRING</td>
<td>The name of an IPSet managed outside of libvirt</td>
</tr>
<tr>
<td>ipsetflags</td>
<td>IPSETFLAGS</td>
<td>flags for the IPSet; requires ipset attribute</td>
</tr>
</tbody>
</table>
H.10.11. Advanced Filter Configuration Topics

The following sections discuss advanced filter configuration topics.

H.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. As an example, if a guest has TCP port 8080 open as a server, clients may connect to the guest on port 8080. Connection tracking and enforcement of directionality then prevents the guest from initiating a connection from (TCP client) port 8080 to the host back to a remote host. More importantly, tracking helps to prevent remote attackers from establishing a connection back to a guest. For example, if the user inside the guest 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. By default the connection state match that enables connection tracking and then enforcement of directionality of traffic is turned on.

Example H.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.

H.10.11.2. Limiting Number of Connections

To limit the number of connections a guest 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 H.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>
[...]
```
Limitation rules must be listed in the XML prior to the rules for accepting traffic. According to the XML file in Example H.10. "XML sample file that sets limits to connections", an additional rule for allowing DNS traffic sent to port 22 go out the guest, 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 may have a long timeout in the guest’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’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 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. 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.

H.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). In addition, it cannot currently be in use
- `nwfilter-dumpxml` : displays a specified network filter (must supply a name)
- `nwfilter-edit` : edits a specified network filter (must supply a name)

H.10.11.4. Pre-existing network filters

The following is a list of example network filters that are automatically installed with `libvirt`:

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-arp-spoofing</td>
<td>Prevents a guest 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.</td>
</tr>
<tr>
<td>allow-dhcp</td>
<td>Allows a guest to request an IP address via DHCP (from any DHCP server)</td>
</tr>
<tr>
<td>allow-dhcp-server</td>
<td>Allows a guest 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 <code>DHCPSERVER</code>.</td>
</tr>
<tr>
<td>no-ip-spoofing</td>
<td>Prevents a guest from sending IP packets with a source IP address different from the one inside the packet.</td>
</tr>
<tr>
<td>no-ip-multicast</td>
<td>Prevents a guest from sending IP multicast packets.</td>
</tr>
<tr>
<td>clean-traffic</td>
<td>Prevents MAC, IP and ARP spoofing. This filter references several other filters as building blocks.</td>
</tr>
</tbody>
</table>

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.

H.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 hosts 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 H.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, 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 ebtatable 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 ebtatable 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 H.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 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, 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. To do this a rule will be added that drops all other traffic. Assuming the guest 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:

```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 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 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>
```

H.10.11.6. Sample custom filter

Although one of the rules in the above XML contains the IP address of the guest 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 H.12. Sample XML for network interface descriptions

An XML fragment for a possible network interface description inside the domain XML of the test guest could then look like this:

```xml
<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 and only ICMP echo responses be received by the guest, the above ICMP rule can be replaced with the following two rules:

```xml
<!-- 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 H.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. The requirements for this filter are:

- Prevents a guest's interface from MAC, IP, and ARP spoofing
- Opens only TCP ports 22 and 80 in a guest's interface
- Allows the guest to send ping traffic from an interface but does not allow the guest to be pinged on the interface
- Allows the guest to do DNS lookups (UDP towards port 53)
- Enables the ftp server (in active mode) so it can run inside the guest

The additional requirement of allowing an FTP server to be run inside the guest maps into the requirement of allowing port 21 to be reachable for FTP control traffic as well as enabling the guest to establish an outgoing TCP connection originating from the guest'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. For the guest-initiated FTP data connection (FTP active mode) the RELATED state is used to enable detection that the guest-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. 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 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-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'/>
</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, 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'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 is established:

```
# modprobe nf_conntrack_ftp - where available OR
```
modprobe ip_conntrack_ftp

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 or be sent by the guest. 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.

```xml
<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 - ->
  <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 - ->
  <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, 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>
```

H.10.12. Limitations

The following is a list of the currently known limitations of the network filtering subsystem.

1. VM migration is only supported if the whole filter tree that is referenced by a guest's top level filter is also available on the target host. The network filter clean-traffic for example should be available on all libvirt installations and thus enable migration of guests 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.

2. 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.

3. VLAN (802.1Q) packets, if sent by a guest, 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 H.1.2. “An example of network filtering” will not work as expected.
Additional resources

To learn more about virtualization and Fedora, refer to the following resources.

I.1. Online resources

- [http://www.libvirt.org/](http://www.libvirt.org/) is the official website for the libvirt virtualization API.
- [http://virt-manager.et.redhat.com/](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.opervirtualization.com](http://www.opervirtualization.com)
- Fedora Documentation [http://docs.fedoraproject.org](http://docs.fedoraproject.org)
- Virtualization technologies overview [http://virt.kernelnewbies.org](http://virt.kernelnewbies.org)

I.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/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 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.

Manipulating the domain xml

This section describes the XML format used to represent one or more libvirt domains. Whereas a libvirt domain refers to a specified guest virtual machine, here the term domain refers to the root `<domain>` element required for defining the parameters of all guest virtual machines. The `<domain>` element 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 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.

J.1. General information and metadata

This information is in this part of the domain XML:
The components of this section of the domain XML are as follows:

Table J.1. General metadata elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt;</td>
<td>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. It is often used to form the filename for storing the persistent configuration files.</td>
</tr>
<tr>
<td>&lt;uuid&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>title Creates space for a short description of the domain. The title should not contain any newlines.</td>
</tr>
<tr>
<td>&lt;description&gt;</td>
<td>Different from the title, this data is not used by libvirt in any way, it can contain any information the user wants to display.</td>
</tr>
<tr>
<td>&lt;metadata&gt;</td>
<td>Can be used by applications to store custom metadata in the form of XML nodes/trees. Applications must use custom namespaces on their XML node/trees, with only one top-level element per namespace (if the application needs structure, they should have sub-elements to their namespace element).</td>
</tr>
</tbody>
</table>

J.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 bootloader, direct kernel boot, and container boot.

J.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>
...
```
The components of this section of the domain XML are as follows:

### Table J.2. BIOS bootloader elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;type&gt;</code></td>
<td>Specifies the type of operating system to be booted on the guest virtual machine. <code>hvm</code> indicates that the OS is one designed to run on bare metal, so requires full virtualization. <code>linux</code> refers to an OS that supports the Xen 3 hypervisor guest ABI. There are also two optional attributes, <code>arch</code> specifying the CPU architecture to virtualization, and <code>machine</code> referring to the machine type. Refer to <a href="#">Driver Capabilities</a> for more information.</td>
</tr>
<tr>
<td><code>&lt;loader&gt;</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>&lt;boot&gt;</code></td>
<td>Takes one of the values: <code>fd</code>, <code>hd</code>, <code>cdrom</code> or <code>network</code> 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 <code>virDomainGetXMLDesc</code>) lists devices in the sorted order. Once sorted, the first device is marked as bootable. For more information see <a href="#">BIOS bootloader</a>.</td>
</tr>
<tr>
<td><code>&lt;bootmenu&gt;</code></td>
<td>Determines whether or not to enable an interactive boot menu prompt on guest startup. The <code>enable</code> attribute can be either <code>yes</code> or <code>no</code>. If not specified, the hypervisor default is used.</td>
</tr>
<tr>
<td><code>&lt;smbios&gt;</code></td>
<td>Determines how SMBIOS information is made visible in the guest. The <code>mode</code> attribute must be specified, as either <code>emulate</code> (lets the hypervisor generate all values), <code>host</code> (copies all of Block 0 and Block 1, except for the UUID, from the host's SMBIOS values; the <code>virConnectGetSysinfo</code> call can be used to see what values are copied), or <code>sysinfo</code> (uses the values in the <code>sysinfo</code> element). If not specified, the hypervisor default setting is used.</td>
</tr>
<tr>
<td><code>&lt;bios&gt;</code></td>
<td>This element has attribute <code>useserial</code> with possible values <code>yes</code> or <code>no</code>. 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, <code>rebootTimeout</code> that controls whether and after how long the guest 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.</td>
</tr>
</tbody>
</table>

### J.2.2. Host bootloader

Hypervisors employing paravirtualization do not usually emulate a BIOS, but instead the host is responsible for the operating system boot. This may use a pseudo-bootloader in the host to provide an interface to choose a kernel for the guest. An example is pygrub with Xen.
### J.3. Host bootloader domain XML

The components of this section of the domain XML are as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;bootloader&gt;</td>
<td>provides a fully qualified path to the bootloader executable in the host OS. This bootloader will choose which kernel to boot. The required output of the bootloader is dependent on the hypervisor in use.</td>
</tr>
<tr>
<td>&lt;bootloader_args&gt;</td>
<td>allows command line arguments to be passed to the bootloader (optional command)</td>
</tr>
</tbody>
</table>

### J.2.3. Direct kernel boot

When installing a new guest OS, it is often useful to boot directly from a kernel and initrd stored in the host OS, allowing command line arguments to be passed directly to the installer. This capability is usually available for both para and full virtualized guests.

The components of this section of the domain XML are as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;type&gt;</td>
<td>same as described in the BIOS boot section</td>
</tr>
<tr>
<td>&lt;loader&gt;</td>
<td>same as described in the BIOS boot section</td>
</tr>
<tr>
<td>&lt;kernel&gt;</td>
<td>specifies the fully-qualified path to the kernel image in the host OS</td>
</tr>
<tr>
<td>&lt;initrd&gt;</td>
<td>specifies the fully-qualified path to the (optional) ramdisk image in the host OS</td>
</tr>
<tr>
<td>&lt;cmdline&gt;</td>
<td>specifies arguments to be passed to the kernel (or installer) at boot time. This is often used to specify an alternate primary console (e.g., serial port), or the installation media source / kickstart file</td>
</tr>
</tbody>
</table>

### J.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.
J.3. SMBIOS system information

Some hypervisors allow control over what system information is presented to the guest (for example, SMBIOS fields can be populated by a hypervisor and inspected via the dmidecode command in the guest). The optional sysinfo element covers all such categories of information.

```xml
...<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>
...<os>
```

Figure J.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:

3. `smbios` - Sub-elements call out specific SMBIOS values, which will affect the guest 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.

J.4. CPU allocation

The `<cpu>` element defines the maximum number of virtual CPUs allocated for the guest 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 `sgmltagal` 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 noderset 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.
J.5. CPU tuning

Figure J.8. CPU tuning

Although all are optional, the components of this section of the domain XML are as follows:
Table J.5. CPU tuning elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cputune&gt;</code></td>
<td>Provides details regarding the CPU tunable parameters for the domain. This is optional.</td>
</tr>
<tr>
<td><code>&lt;vcpupin&gt;</code></td>
<td>Specifies which of host’s physical CPUs the domain VCPU will be pinned to. If this is omitted, and attribute <code>cpuset</code> of element <code>&lt;vcpu&gt;</code> is not specified, the VCPU is pinned to all the physical CPUs by default. It contains two required attributes, the attribute <code>vcpu</code> specifies <code>id</code>, and the attribute <code>cpuset</code> is same as attribute <code>cpuset</code> of element <code>&lt;vCPU&gt;</code>..</td>
</tr>
<tr>
<td><code>&lt;emulatorpin&gt;</code></td>
<td>Specifies which of the host physical CPUs, the “emulator”, a subset of a domains not including vcpu, will be pinned to. If this is omitted, and attribute <code>cpuset</code> of element <code>&lt;vcpu&gt;</code> is not specified, the “emulator” is pinned to all the physical CPUs by default. It contains one required attribute <code>cpuset</code> specifying which physical CPUs to pin to. <code>&lt;emulatorpin&gt;</code> is not allowed if attribute <code>placement</code> of element <code>&lt;vcpu&gt;</code> is <code>auto</code>.</td>
</tr>
<tr>
<td><code>&lt;shares&gt;</code></td>
<td>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 machines. 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.</td>
</tr>
<tr>
<td><code>&lt;period&gt;</code></td>
<td>Specifies the enforcement interval in microseconds. By using <code>period</code>, each of the domain’s vcpu will not be allowed to consume more than its allotted quota worth of runtime. This value should be within the following range: <code>1000-1000000</code>. A <code>period</code> with a value of <code>0</code> means no value.</td>
</tr>
<tr>
<td><code>&lt;quota&gt;</code></td>
<td>Specifies the maximum allowed bandwidth in microseconds. A domain with <code>quota</code> 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: <code>1000 - 18446744073709551</code> or less than <code>0</code>. A <code>quota</code> with value of <code>0</code> means no value. You can use this feature to ensure that all vcpus run at the same speed.</td>
</tr>
<tr>
<td><code>&lt;emulator_period&gt;</code></td>
<td>Specifies the enforcement interval in microseconds. Within an <code>&lt;emulator_period&gt;</code>, emulator threads (those excluding vcpus) of the domain will not be allowed to consume more than the <code>&lt;emulator_quota&gt;</code> worth of runtime. The <code>&lt;emulator_period&gt;</code> value should be in the following range: <code>1000 - 1000000</code>. An <code>&lt;emulator_period&gt;</code> with value of <code>0</code> means no value.</td>
</tr>
</tbody>
</table>
| `<emulator_quota>` | 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.

J.6. Memory backing

Memory backing allows the hypervisor to properly manage large pages
within the guest. Once configured the following domain XML is effected:

```xml
<domain>
    ...
    <memoryBacking>
        <hugepages/>
    </memoryBacking>
    ...
</domain>
```

Figure J.9. Memory backing

The optional <memoryBacking> element, may have an
<hugepages> element set within it. This tells the hypervisor that the
guest should have its memory allocated using hugepages instead of the
normal native page size.

J.7. Memory tuning

```xml
<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'>67208864</min_guarantee>
    </memtune>
    ...
</domain>
```

Figure J.10. Memory tuning

Although all are optional, the components of this section of the domain
XML are as follows:
Table J.6. Memory tuning elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;memtune&gt;</code></td>
<td>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 RAM, guest video RAM, and allow for some memory overhead. The last piece is hard to determine so one use 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 <code>&lt;memory&gt;</code>. For backwards compatibility, output is always in KiB.</td>
</tr>
<tr>
<td><code>&lt;hard_limit&gt;</code></td>
<td>This is the maximum memory the guest can use. The unit for this value is expressed in kibibytes (i.e. blocks of 1024 bytes)</td>
</tr>
<tr>
<td><code>&lt;soft_limit&gt;</code></td>
<td>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)</td>
</tr>
<tr>
<td><code>&lt;swap_hard_limit&gt;</code></td>
<td>This is the maximum memory plus swap the guest can use. The unit for this value is expressed in kibibytes (i.e. blocks of 1024 bytes). This has to be more than <code>&lt;hard_limit&gt;</code> value provided</td>
</tr>
<tr>
<td><code>&lt;min_guarantee&gt;</code></td>
<td>This is the guaranteed minimum memory allocation for the guest. The units for this value is expressed in kibibytes (i.e. blocks of 1024 bytes)</td>
</tr>
</tbody>
</table>

J.8. NUMA node tuning

Once NUMA node tuning is done using conventional management tools the following domain XML parameters are effected:

```xml
<domain>
  ...
  <numatune>
    <memory mode="strict" nodeset="1-4,^3"/>
  </numatune>
  ...
</domain>
```

Figure J.11. NUMA node tuning

Although all are optional, the components of this section of the domain XML are as follows:
Table J.7. NUMA node tuning elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;numatune&gt;</td>
<td>Provides details of how to tune the performance of a NUMA host via controlling NUMA policy for domain process.</td>
</tr>
<tr>
<td>&lt;memory&gt;</td>
<td>Specifies how to allocate memory for the domain process on a NUMA host. It contains several optional attributes. Attribute <strong>mode</strong> is either <strong>interleave</strong>, <strong>strict</strong>, or <strong>preferred</strong>. If no value is given it defaults to <strong>strict</strong>. Attribute <strong>nodeset</strong> specifies the NUMA nodes, using the same syntax as attribute <strong>cpuset</strong> of element <strong>&lt;vcpu&gt;</strong>. Attribute <strong>placement</strong> can be used to indicate the memory placement mode for the domain process. Its value can be either <strong>static</strong> or <strong>auto</strong>. If attribute <strong>nodeset</strong> is specified it defaults to the <strong>&lt;placement&gt;</strong> of <strong>&lt;vcpu&gt;</strong>, or <strong>static</strong>. <strong>auto</strong> indicates the domain process will only allocate memory from the advisory nodeset returned from querying numad and the value of attribute <strong>nodeset</strong> will be ignored if it’s specified. If attribute <strong>placement</strong> of <strong>&lt;vcpu&gt;</strong> is <strong>auto</strong>, and attribute <strong>&lt;numatune&gt;</strong> is not specified, a default numatune with <strong>&lt;placement&gt;</strong> auto and mode <strong>strict</strong> will be added implicitly.</td>
</tr>
</tbody>
</table>

J.9. Block I/O tuning

```xml
<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 J.12. Block I/O tuning

Although all are optional, the components of this section of the domain XML are as follows:
Table J.8. Block I/O tuning elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;blkiotune&gt;</code></td>
<td>This optional element provides the ability to tune block I/O cgroup tunable parameters for the domain. If this is omitted, it defaults to the OS provided defaults.</td>
</tr>
<tr>
<td><code>&lt;weight&gt;</code></td>
<td>This optional weight element is the overall I/O weight of the guest. The value should be within the range 100 - 1000.</td>
</tr>
<tr>
<td><code>&lt;device&gt;</code></td>
<td>The domain may have multiple <code>&lt;device&gt;</code> elements that further tune the weights for each host block device in use by the domain. Note that multiple guest disks can share a single host block device. In addition, as they are backed by files within the same host file system, this tuning parameter is at the global domain level, rather than being associated with each guest disk device (contrast this to the <code>&lt;irotune&gt;</code> element which can be applied to a single <code>&lt;disk&gt;</code>). Each device element has two mandatory sub-elements, <code>&lt;path&gt;</code> describing the absolute path of the device, and <code>&lt;weight&gt;</code> giving the relative weight of that device, which has an acceptable range of 100 - 1000.</td>
</tr>
</tbody>
</table>

J.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. Only the (hypervisor specific) default partition can be assumed to exist by default.

```
...<resource>
  <partition>/virtualmachines/production</partition>
</resource>
...
```

Figure J.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.

J.11. CPU model and topology

This section covers the requirements for CPU model. Its features and 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' thread='1'/>
  <feature policy='disable' name='lahf_lm'/>
</cpu>
...
```

Figure J.14. CPU model and topology example 1
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:

```xml
<cpu mode='host-passthrough'/>
```

Figure J.16. CPU model and topology example 3

The components of this section of the domain XML are as follows:
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cpu&gt;</code></td>
<td>This is the main container for describing guest CPU requirements.</td>
</tr>
<tr>
<td><code>&lt;match&gt;</code></td>
<td>Specifies how the virtual CPU is provided to the guest match for these requirements. The <code>match</code> attribute can be omitted if topology is the only element within <code>&lt;cpu&gt;</code>. Possible values for the <code>match</code> attribute are:</td>
</tr>
<tr>
<td></td>
<td>- <code>minimum</code> - the specified CPU model and features describes the minimum requested CPU.</td>
</tr>
<tr>
<td></td>
<td>- <code>exact</code> - the virtual CPU provided to the guest will exactly match the specification</td>
</tr>
<tr>
<td></td>
<td>- <code>strict</code> - the guest will not be created unless the host CPU exactly matches the specification.</td>
</tr>
<tr>
<td></td>
<td>Note that the <code>match</code> attribute can be omitted and will default to <code>exact</code>.</td>
</tr>
<tr>
<td><code>&lt;mode&gt;</code></td>
<td>This optional attribute may be used to make it easier to configure a guest CPU to be as close to the host CPU as possible. Possible values for the <code>mode</code> attribute are:</td>
</tr>
<tr>
<td></td>
<td>- <code>custom</code> - describes how the CPU is presented to the guest. This is the default setting when the <code>mode</code> attribute is not specified. This mode makes it so that a persistent guest will see the same hardware no matter what host the guest is booted on.</td>
</tr>
<tr>
<td></td>
<td>- <code>host-model</code> - this is essentially a shortcut to copying host 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 hosts while still providing the best guest CPU each host supports. Neither the <code>match</code> attribute nor any feature elements can be used in this mode. For more information see <a href="https://www.libvirt.org/formatdomain.html#definitions">libvirt domain XML CPU models</a>.</td>
</tr>
<tr>
<td></td>
<td>- <code>host-passthrough</code> - With this mode, the CPU visible to the guest is exactly the same as the host CPU including elements that cause errors within libvirt. The obvious downside of this mode is that the guest environment cannot be reproduced on different hardware and therefore this mode is recommended with great caution. Neither <code>model</code> nor <code>feature</code> elements are allowed in this mode.</td>
</tr>
<tr>
<td></td>
<td>Note that in both <code>host-model</code> and <code>host-passthrough</code> mode, the real (approximate in <code>host-passthrough</code> mode) CPU definition which would be used on current host can be determined by specifying <code>VIR_DOMAIN_XML_UPDATE_CPU</code> flag when calling <code>virDomainGetXMLDesc</code> API. When running a guest that might be prone to operating system reactivation when presented with different hardware, and which will be migrated between hosts with different capabilities, you can use this output to rewrite XML to the custom mode for more robust migration.</td>
</tr>
<tr>
<td><code>&lt;model&gt;</code></td>
<td>Specifies CPU model requested by the guest. The list of available CPU models and their definition can be found in <code>cpu_map.xml</code> 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...</td>
</tr>
</tbody>
</table>
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. It must be exactly 12 characters long. If not set, the vendor id of the host is used. Typical possible values are AuthenticAMD and GenuineIntel.

```xml
<vendor> Specifies CPU vendor requested by the guest. If this element is missing, the guest 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. 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 policy 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 CPU.
  - require - dictates that guest creation will fail unless the feature is supported by host CPU. This is the default setting.
  - optional - this feature is supported by virtual CPU but and only if it is supported by host CPU.
  - disable - this is not supported by virtual CPU.
  - forbid - guest creation will fail if the feature is supported by host CPU.
```

**J.11.1. Guest NUMA topology**

Guest NUMA topology can be specified using the numa element and the following from the domain XML:

```xml
... <cpu> ...
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
... </cpu> ...
```

**Figure J.18. Guest 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.

**J.12. Events configuration**

Using the following sections of domain XML it is possible to override the default actions taken on various events.
The following collections of elements allow the actions to be specified when a guest OS triggers a lifecycle 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 J.10. Event configuration elements

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;on_poweroff&gt;</code></td>
<td>Specifies the action that is to be executed when the guest requests a power off. Four possible arguments are possible:</td>
</tr>
<tr>
<td></td>
<td>- <code>destroy</code> - this action terminates the domain completely and releases all resources</td>
</tr>
<tr>
<td></td>
<td>- <code>restart</code> - this action terminates the domain completely and restarts it with the same configuration</td>
</tr>
<tr>
<td></td>
<td>- <code>preserve</code> - this action terminates the domain completely but and its resources are preserved to allow for future analysis.</td>
</tr>
<tr>
<td></td>
<td>- <code>rename-restart</code> - this action terminates the domain completely and then restarts it with a new name</td>
</tr>
<tr>
<td><code>&lt;on_reboot&gt;</code></td>
<td>Specifies the action that is to be executed when the guest requests a reboot. Four possible arguments are possible:</td>
</tr>
<tr>
<td></td>
<td>- <code>destroy</code> - this action terminates the domain completely and releases all resources</td>
</tr>
<tr>
<td></td>
<td>- <code>restart</code> - this action terminates the domain completely and restarts it with the same configuration</td>
</tr>
<tr>
<td></td>
<td>- <code>preserve</code> - this action terminates the domain completely but and its resources are preserved to allow for future analysis.</td>
</tr>
<tr>
<td></td>
<td>- <code>rename-restart</code> - this action terminates the domain completely and then restarts it with a new name</td>
</tr>
<tr>
<td><code>&lt;on_crash&gt;</code></td>
<td>Specifies the action that is to be executed when the guest crashes. In addition, it supports these additional actions:</td>
</tr>
<tr>
<td></td>
<td>- <code>coredump-destroy</code> - the crashed domain's core is dumped, domain is terminated completely, and all resources are released.</td>
</tr>
<tr>
<td></td>
<td>- <code>coredump-restart</code> - the crashed domain's core is dumped, and the domain is restarted with the same configuration settings</td>
</tr>
<tr>
<td><code>&lt;on_lockfailure&gt;</code></td>
<td>Specifies what action should be taken when a lock manager loses resource locks. The following</td>
</tr>
<tr>
<td></td>
<td>- <code>destroy</code> - this action terminates the domain completely and releases all resources</td>
</tr>
<tr>
<td></td>
<td>- <code>restart</code> - this action terminates the domain completely and restarts it with the same configuration</td>
</tr>
<tr>
<td></td>
<td>- <code>preserve</code> - this action terminates the domain completely but and its resources are preserved to allow for future analysis.</td>
</tr>
<tr>
<td></td>
<td>- <code>rename-restart</code> - this action terminates the domain completely and then restarts it with a new name</td>
</tr>
</tbody>
</table>
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:

- **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.

### J.13. Power Management

It is possible to forcibly enable or disable BIOS advertisements to the guest OS using conventional management tools which affects the following section of the domain XML:

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

Figure J.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.

### J.14. Hypervisor features

Hypervisors may allow certain CPU / machine features to be enabled (`state='on'`) or disabled (`state='off'`).

```xml
...<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
  </hyperv>
</features>...
```

Figure J.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 J.11. Hypervisor features elements

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pae&gt;</td>
<td>Physical address extension mode allows 32-bit guests to address more than 4 GB of memory.</td>
</tr>
<tr>
<td>&lt;acpi&gt;</td>
<td>Useful for power management, for example, with KVM guests it is required for graceful shutdown to work.</td>
</tr>
<tr>
<td>&lt;apic&gt;</td>
<td>Allows the use of programmable IRQ management. For this element, there is an optional attribute <code>eoi</code> with values <code>on</code> and <code>off</code> which sets the availability of EOI (End of Interrupt) for the guest.</td>
</tr>
<tr>
<td>&lt;hap&gt;</td>
<td>Enables the use of Hardware Assisted Paging if it is available in the hardware.</td>
</tr>
<tr>
<td>hyperv</td>
<td>Enables various features to improve the behavior of guests 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.</td>
</tr>
</tbody>
</table>

J.15. Time keeping

The guest clock is typically initialized from the host clock. Most operating systems expect the hardware clock to be kept in UTC, which is the default setting. Note that for Windows guests the guest must be set in `localtime`.

```xml
...<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 J.22. Time keeping

The components of this section of the domain XML are as follows:
## Table J.12. Time keeping elements

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
</table>
| <clock> | The offset attribute takes four possible values, allowing for fine grained control over how the guest clock is synchronized to the host. Note that hypervisors are not required to support all policies across all time sources.  
  - 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 clock with the host's configured timezone when booted. The adjustment attribute behaves the same as in 'utc' mode.  
  - timezone - Synchronizes the guest clock to the requested timezone using the timezone attribute.  
  - variable - Gives the guest 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 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. Can be set to yes or no.                                                                                                                   |
J.16. Devices

This set of XML elements are all used to describe devices provided to the guest 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 J.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.

J.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.
Figure J.24. Devices - Hard drives, floppy disks, CDROMs

J.16.1. 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.

J.16.1.2. Source element

If the `<disk type='file'>` is used, then the `file` attribute specifies the fully-qualified path to the file holding the disk. If the `<disk type='block'>` is used, then the `dev` attribute specifies the path to the host 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 hosts 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|event`
- `optional` drops if missing at any start attempt

### J.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.

### J.16.1.4. Target element

The `<target>` element controls the bus / device under which the disk is exposed to the guest 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 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, e.g., 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 Element`

### J.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 `bliiotune` 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.

### J.16.1.6. driver

The optional `<driver>` element allows specifying further details related to the hypervisor 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 page cache) and `unsafe` (host may cache all disk io, and sync requests from guest 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 `enspace`. 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 `error_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 `error_policy` is given, the read error the default setting, `report` will be used.

- The optional `io` attribute controls specific policies on I/O. QEMU guests support threads and native. The optional `loeventfd` 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 to be executed while a separate thread handles I/O. Typically guests experiencing high system CPU utilization during I/O will benefit from this. On the other hand, an overloaded host can increase guest 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. 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.

J.16.1.7. Additional Device Elements

The following attributes may be used within the `device` element:

- `<boot>` - Specified 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. This setting is the default for disks with `attribute devices="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 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's hard drive. For example, `<serial>WD-WMP9A966149</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 one host.
  - `rbd` - Monitors servers of RBD type and may be used for one or more hosts.
  - `sheepdog` - Specifies one of the sheepdog servers (default is localhost:7000) and can be used one or none of the hosts.
  - `gluster` - Specifies a server running a glusterd daemon and may be used for only one host. 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 is 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, lba` or `auto`
- `blockio` - Allows the block device to be overridden with any of the block device properties listed below:
  - `logical_block_size` - reports to the guest OS and describes the smallest units for disk I/O.
  - `physical_block_size` - reports to the guest OS and describes the disk's hardware sector size which can be relevant for the alignment of disk data.

### J.16.2. Filesystems

A filesystems directory on the host that can be accessed directly from the guest.

```
...<device>
  <filesystem type='template'>
    <source name='my-vm-template'/>
    <target dir=''/></filesystem>
  <filesystem type='mount' accessmode='passthrough'>
    <driver type='path' wrpolicy='immediate'/>
    <source dir='/import/from/host'/>  
    <readonly/> 
  </filesystem>
...</device>
```

**Figure J.25. Devices - filesystems**

The `filesystem` attribute has the following possible values:

- `type='mount'` - Specifies the host directory to mount in the guest. 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 page cache; omitting the attribute reverts to the default setting, while specifying a value immediate means that a host writeback is immediately triggered for all pages touched during a guest file write operation.

- `type='template'` - Specifies the OpenVZ filesystem template and is only used by OpenVZ driver.

- `type='file'` - Specifies that a host file will be treated as an image and mounted in the guest. This filesystem format will be autodetected and is only used by LXC driver.

- `type='block'` - Specifies the host block device to mount in the guest. 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 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 which will be bound to another directory inside the guest. 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 virtual guest 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 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 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's filesystem.
- **space_soft_limit** - Specifies the maximum space available to this guest's filesystem. The container is permitted to exceed its soft limits for a grace period of time. Afterwards the hard limit is enforced.

### J.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. 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. For example, a disk device uses `type='disk'`, while a console device would use `type='pci'` on i686 or x86_64 guests, or `type='spapr-vio'` on PowerPC64 pseries guests. 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 0xff, 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 guests, 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.

### J.16.4. Controllers

Depending on the guest 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.
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", "lsislogic", "lsisastor", "virtio-scsi" or "vmpvscsi". A "usb" controller has an optional attribute `model`, which is one of "piix3-uhci", "piix4-uhci", "ehci", "ich9-ehci1", "ich9-ehci2", "ich9-uhci2", "v82c660-uhci", "pci-ohci" or "nec-xhci". Additionally, if the USB bus needs to be explicitly disabled for the guest, `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.

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.

J.16.6. Host device assignment

J.16.6.1. USB / PCI devices
The host's USB and PCI devices can be passed through to the guest
using the hostdev element, by modifying the host 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>
  <hostdev mode='subsystem'
           type='pci'>
    <source>
      <address bus='0x06' slot='0x02' function='0x0'/>
    </source>
    <boot order='1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </hostdev>
</devices>
```

Figure J.29. Devices - host device assignment

Alternatively the following can also be done:

```
<devices>
  <hostdev mode='subsystem'
           type='pci'
           managed='yes'>
    <source>
      <address bus='0x06' slot='0x02' functions='0x0'/>
    </source>
    <boot order='1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </hostdev>
</devices>
```

Figure J.30. Devices - host device assignment alternative

The components of this section of the domain XML are as follows:
Table J.13. Host device assignment elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostdev</td>
<td>This is the main container for describing host 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 before being passed on to the guest, and reattached to the host after the guest exits. If managed is omitted or no for PCI and for USB devices, the user is responsible to use the argument virNodeDeviceDettach (or virsh nodedev-dettach) before starting the guest or hot-plugging the device, and virNodeDeviceReAttach (or virsh nodedev-reattach) after hot-unplug or stopping the guest.</td>
</tr>
<tr>
<td>source</td>
<td>Describes the device as seen from the host. 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 hosts 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 USB device is not found. The attribute accepts the following values:</td>
</tr>
<tr>
<td></td>
<td>- mandatory - fails if missing for any reason (the default)</td>
</tr>
<tr>
<td></td>
<td>- requisite - fails if missing on boot up, drops if missing on migrate/restore/revert</td>
</tr>
<tr>
<td></td>
<td>- optional - drops if missing at any start attempt</td>
</tr>
<tr>
<td>vendor, product</td>
<td>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.</td>
</tr>
<tr>
<td>boot</td>
<td>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.</td>
</tr>
<tr>
<td>rom</td>
<td>Used to change how a PCI device's ROM is presented to the guest. 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'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 default setting will be used. The optional file attribute is used to point to a binary file to be presented to the guest 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).</td>
</tr>
<tr>
<td>address</td>
<td>Also has a bus and device attribute to specify the USB bus and device number the device</td>
</tr>
</tbody>
</table>
appears at on the host. 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 \texttt{lspci} or with \texttt{virsh nodedev-list}

\section*{J.16.6.2. Block / character devices}

The host’s block / character devices can be passed through to the guest by using management tools to modify the domain xml \texttt{hostdev} element. Note that this is only possible with container based virtualization.

\begin{figure}[h]
\centering
\begin{verbatim}
... 
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
\end{verbatim}
\caption{Devices - host device assignment block character devices}
\end{figure}

An alternative approach is this:

\begin{figure}[h]
\centering
\begin{verbatim}
... 
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
\end{verbatim}
\caption{Devices - host device assignment block character devices alternative 1}
\end{figure}

Another alternative approach is this:

\begin{figure}[h]
\centering
\begin{verbatim}
... 
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
\end{verbatim}
\caption{Devices - host device assignment block character devices alternative 2}
\end{figure}

The components of this section of the domain XML are as follows:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Parameter} & \textbf{Description} \\
\hline
hostdev & This is the main container for describing host devices. For block/character devices passthrough \texttt{mode} is always \texttt{capabilities} and \texttt{type} is \texttt{block} for a block device and \texttt{char} for a character device. \\
source & This describes the device as seen from the host. For block devices, the path to the block device in the host OS is provided in the nested \texttt{block} element, while for character devices the \texttt{char} element is used. \\
\hline
\end{tabular}
\caption{Block / character device elements}
\end{table}

\section*{J.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:
The components of this section of the domain XML are as follows:

### Table J.15. Redirected device elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>redirdev</td>
<td>This is the main container for describing redirected devices. <code>bus</code> must be <code>usb</code> for a USB device. An additional attribute type is required, matching one of the supported serial device types, to describe the host side of the tunnel; <code>type='tcp'</code> or <code>type='spicevmc'</code> (which uses the usbredir channel of a SPICE graphics device) are typical. The redirdev element has an optional sub-element <code>address</code> which can tie the device to a particular controller. Further sub-elements, such as <code>source</code>, may be required according to the given <code>type</code>, although <code>target</code> sub-element is not required (since the consumer of the character device is the hypervisor itself, rather than a device visible in the guest).</td>
</tr>
<tr>
<td>boot</td>
<td>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.</td>
</tr>
<tr>
<td>redirfilter</td>
<td>This is used for creating the filter rule to filter out certain devices from redirection. It uses sub-element <code>usbddev</code> to define each filter rule. The <code>class</code> attribute is the USB Class code.</td>
</tr>
</tbody>
</table>

### J.16.8. Smartcard devices

A virtual smartcard device can be supplied to the guest via the `smartcard` element. A USB smartcard reader device on the host cannot be used on a guest with simple device passthrough, as it cannot be made available to both the host and guest and can possibly lock the host computer when it is removed from the guest. Therefore, some hypervisors provide a specialized virtual device that can present a smartcard interface to the guest, with several modes for describing how the credentials are obtained from the host 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:

```
<device>
  <redirdev bus='usb' type='tcp'>
    <source mode='connect' host='localhost' service='4000'/>
    <boot order='1'/>
  </redirdev>
  <redirfilter>
    <usbddev class='0x08' vendor='0x1234' product='0xbeef' version='2.00' allow='yes'/>
    <usbddev allow='no'/>
  </redirfilter>
</device>
```

USB device redirection through a character device is supported by configuring it with management tools that modify the following section of the domain xml:
The **smartcard** element has a mandatory attribute **mode**. The following modes are supported; in each mode, the guest 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode='host'</td>
<td>In this mode, the hypervisor relays all requests from the guest into direct access to the host's smartcard via NSS. No other attributes or sub-elements are required. See below about the use of an optional address sub-element.</td>
</tr>
<tr>
<td>mode='host-certificates'</td>
<td>This mode allows you to provide three NSS certificate names residing in a database on the host, rather than requiring a smartcard to be plugged into the host. 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>.</td>
</tr>
<tr>
<td>mode='passthrough'</td>
<td>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. In this mode of operation, an additional attribute <strong>type</strong> is required, matching one of the supported serial device types, to describe the host side of the tunnel; <strong>type='tcp'</strong> or <strong>type='spicevmc'</strong> (which uses the smartcard channel of a SPICE graphics device) are typical. Further sub-elements, such as <strong>source</strong>, may be required according to the given type, although a <strong>target</strong> sub-element is not required (since the consumer of the character device is the hypervisor itself, rather than a device visible in the guest).</td>
</tr>
</tbody>
</table>

Each mode supports an optional sub-element **address**, which fine-tunes the correlation between the smartcard and a ccid bus controller (Refer to Section J.16.3, "Device addresses").

---

Figure J.35. Devices - smartcard devices
J.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 J.36. Devices - network interfaces

There are several possibilities for specifying a network interface visible to the guest. 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 J.16.3, "Device addresses").

J.16.9.1. Virtual networks

This is the recommended configuration for general guest connectivity on hosts with dynamic / wireless networking configurations (or multi-host environments where the host 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'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 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 will have an associated tun device created with a name of vnetN, which can also be overridden with the `<target>` element (refer to Section J.16.9.10, "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, 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 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'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:

...
J.16.9.2. Bridge to LAN

Note that this is the recommended configuration setting for general guest connectivity on hosts 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 which has one or more of the hosts 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 J.16.9.10, "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 bridge. On hosts 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='network'>
    <source network='default' portgroup='engineering'/>
  </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 J.37. Devices - network interfaces- virtual networks

```
<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='00b11c53-8b5c-4eeb-8f00-d84eaa8aaa4f'/>
    </virtualport>
  </interface>
  ...
</devices>
```

Figure J.38. Devices - network interfaces- bridge to LAN

J.16.9.3. 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>
```

```
J.16.9.4. Generic Ethernet connection

Provides a means for the administrator to execute an arbitrary script to connect the guest’s network to the LAN. The guest will have a \texttt{tun} device created with a name of \texttt{vnetN}, which can also be overridden with the \texttt{target} element. After creating the \texttt{tun} device a shell script will be run which is expected to do whatever host network integration is required. By default this script is called \texttt{/etc/qemu-ifup} but can be overridden (refer to \texttt{Section J.16.9.14, “Overiding 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 J.40. Devices - network interfaces- generic Ethernet connection

J.16.9.5. 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 physical interface of the host is specified.

This setup requires the Linux macvtap driver to be available. One of the modes \texttt{vepa (‘Virtual Ethernet Port Aggregator’), bridge} or \texttt{private} can be chosen for the operation mode of the macvtap device, \texttt{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 J.41. Devices - network interfaces- direct attachment to physical interfaces

The individual modes cause the delivery of packets to behave as shown in \textit{Table J.17, “Direct attachment to physical interface elements”}.
Table J.17. Direct attachment to physical interface elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vepa</td>
<td>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 as where the packet originates from are sent back to the host by the VEPA capable bridge (today's bridges are typically not VEPA capable).</td>
</tr>
<tr>
<td>bridge</td>
<td>Packets whose destination is on the same host 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.</td>
</tr>
<tr>
<td>private</td>
<td>All packets are sent to the external bridge and will only be delivered to a target VM on the same host if they are sent through an external router or gateway and that device sends them back to the host. This procedure is followed if either the source or destination device is in private mode.</td>
</tr>
<tr>
<td>passthrough</td>
<td>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.</td>
</tr>
</tbody>
</table>

The network access of direct attached virtual machines can be managed by the hardware switch to which the physical interface of the host 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 J.18, "Direct attachment to physical interface additional elements".

Table J.18. Direct attachment to physical interface additional elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>managerid</td>
<td>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.</td>
</tr>
<tr>
<td>typeid</td>
<td>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.</td>
</tr>
<tr>
<td>typeidversion</td>
<td>The VSI Type Version allows multiple versions of a VSI Type. This is an integer value.</td>
</tr>
<tr>
<td>instanceid</td>
<td>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.</td>
</tr>
<tr>
<td>profileid</td>
<td>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.</td>
</tr>
</tbody>
</table>
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>
```

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>
```

**J.16.9.6. PCI passthrough**

A PCI network device (specified by the `source` element) is directly assigned to the guest 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, 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 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 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>
      </source>
    </source>
  </interface>
...<devices>
```

**J.16.9.7. 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 hosts. 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 guests 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:

```xml
...<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
      <source address='230.0.0.1' port='5558'/>
    </interface>
  </interface>
...</devices>
```

Figure J.45. Devices - network interfaces - multicast tunnel

### J.16.9.8. 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 by 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:

```xml
...<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
      <source address='192.168.0.1' port='5558'/>
    </interface>
  </interface>
  <interface type='client'>
    <mac address='52:54:00:8b:c9:51'>
      <source address='192.168.0.1' port='5558'/>
    </interface>
  </interface>
...</devices>
```

Figure J.46. Devices - network interfaces - TCP tunnel

### J.16.9.9. 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:

```xml
<device>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <model type='virtio'/>
    <driver name='vhost' txmode='iothread'
ioeventfd='on' event_idx='off'/>
  </interface>
</device>
```

Figure J.47. Devices - network interfaces - setting NIC driver-specific options

Currently the following attributes are available for the "virtio" NIC driver:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>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.</td>
</tr>
<tr>
<td>txmode</td>
<td>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 &quot;tx=io&quot; to the qemu commandline -device virtio-net-pci option). If set to timer, tx work is performed 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.</td>
</tr>
<tr>
<td>ioeventfd</td>
<td>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 may also increase guest I/O latency. Therefore, you should leave this option alone, unless you are very certain you that changing it is an absolute necessity.</td>
</tr>
<tr>
<td>event_idx</td>
<td>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.</td>
</tr>
</tbody>
</table>

### J.16.9.10. Overriding the target element

To override the target element, use a management tool to make the following changes to the domain XML:

```xml
... 
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
  </interface>
</devices>
...
```

Figure J.48. 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.

J.16.9.11. 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 J.49. 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.

J.16.9.12. 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 J.50. 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’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 versions have a default of on). The optional file attribute is used to point to a binary file to be presented to the guest as the device’s ROM BIOS. This can be useful to provide an alternative boot ROM for a network device.

J.16.9.13. 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 attribute 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:
3.16.9.14. 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:

```xml
<devices>
  <interface type='bridge'>
    <vlan>
      <tag id='42'/>
    </vlan>
    <source bridges='ovsbr0'/>
    <parameters interfaceid='8f08-d84ea0aa4f'/>
  </interface>
</devices>
```

Figure J.52. Setting VLAN tag (on supported network types only)

If (and only if) the network connection used by the guest supports vlan tagging transparent to the guest, an optional `vlan` element can specify one or more vlan tags to apply to the guest's network traffic (openvswitch and `type='hostdev'` SR-IOV interfaces do support transparent vlan tagging of guest traffic; everything else, including standard linux bridges and libvirt's own virtual networks, do not support it. 802.1Qbg (VEPA) switches provide their own way (outside of libvirt) to tag guest 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.

3.16.9.15. 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:

```xml
<devices>
  <interface type='network'>
    <source network='default'/>
    <link state='down'/>
  </interface>
</devices>
```

Figure J.53. Modifying virtual link state

3.16.10. Input devices

Input devices allow interaction with the graphical framebuffer in the guest 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:
Figure J.54. 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.

J.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 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 J.55. 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.

J.16.12. Graphical framebuffers

A graphics device allows for graphical interaction with the guest OS. A guest 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 J.56. Graphical framebuffers

The `<graphics>` element has a mandatory `<type>` attribute which takes the value `<sdl>`, `<vnc>`, `<rdp>` or `<desktop>` as explained below:
Table J.20. Graphical framebuffer elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdl</td>
<td>This displays a window on the host desktop, it can take 3 optional arguments: a <code>display</code> attribute for the display to use, an <code>xauth</code> attribute for the authentication identifier, and an optional <code>fullscreen</code> attribute accepting values <code>yes</code> or <code>no</code></td>
</tr>
<tr>
<td>vnc</td>
<td>Starts a VNC server. The <code>port</code> attribute specifies the TCP port number (with <code>-1</code> as legacy syntax indicating that it should be auto-allocated). The <code>autoport</code> attribute is the new preferred syntax for indicating autoallocation of the TCP port to use. The <code>listen</code> attribute is an IP address for the server to listen on. The <code>passwd</code> attribute provides a VNC password in clear text. The <code>keymap</code> attribute specifies the keymap to use. It is possible to set a limit on the validity of the password by giving an <code>timestamp</code>. <code>passwdValidTo='2010-04-09T15:51:00'</code> is assumed to be in UTC. The <code>connected</code> attribute allows control of connected client during password changes. VNC accepts <code>keep</code> 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.</td>
</tr>
<tr>
<td>spice</td>
<td>Starts a SPICE server. The <code>port</code> attribute specifies the TCP port number (with <code>-1</code> as legacy syntax indicating that it should be auto-allocated), while <code>tlsPort</code> gives an alternative secure port number. The <code>autoport</code> attribute is the new preferred syntax for indicating autoallocation of both port numbers. The <code>listen</code> attribute is an IP address for the server to listen on. The <code>passwd</code> attribute provides a SPICE password in clear text. The <code>keymap</code> attribute specifies the keymap to use. It is possible to set a limit on the validity of the password by giving an <code>timestamp</code>. <code>passwdValidTo='2010-04-09T15:51:00'</code> is assumed to be in UTC. The <code>connected</code> attribute allows control of connected client during password changes. SPICE accepts <code>keep</code> to keep client connected, disconnect to disconnect client and <code>fail</code> to fail changing password. Note it is not be supported by all hypervisors. The <code>defaultMode</code> attribute sets the default channel security policy, valid values are <code>secure</code>, <code>insecure</code> and the default <code>any</code> (which is <code>secure</code> if possible, but falls back to <code>insecure</code> rather than erroring out if no secure path is available).</td>
</tr>
</tbody>
</table>

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:
Figure J.57. 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 J.21. Additional graphical framebuffer elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdp</td>
<td>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 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.</td>
</tr>
<tr>
<td>desktop</td>
<td>This value is reserved for VirtualBox domains for the moment. It displays a window on the host desktop, similarly to &quot;sdl&quot;, but uses the VirtualBox viewer. Just like &quot;sdl&quot;, it accepts the optional attributes display and fullscreen.</td>
</tr>
<tr>
<td>listen</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• <strong>type</strong> - 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).</td>
</tr>
<tr>
<td></td>
<td>• <strong>address</strong> - 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 &quot;live&quot; XML of a running domain, this attribute will be set to the IP address used for listening, even if <strong>type='network'</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>network</strong> - if <strong>type='network'</strong>, the network attribute will contain the name of a network in libvit'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 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.</td>
</tr>
</tbody>
</table>

### J.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:
The **graphics** element has a mandatory type attribute which takes the value "sdl", "vnc", "rdp" or "desktop" as explained below:

Table J.22. Graphical framebuffer elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>video</strong></td>
<td>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 type. If &quot;ram&quot; or &quot;vram&quot; are not supplied a default value is used.</td>
</tr>
<tr>
<td><strong>model</strong></td>
<td>This has a mandatory type attribute which takes the value {vga, cirrus, vmvga, xen, vbox, 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 heads.</td>
</tr>
<tr>
<td><strong>acceleration</strong></td>
<td>If acceleration is supported it should be enabled using the accel3d and accel2d attributes in the acceleration element.</td>
</tr>
<tr>
<td><strong>address</strong></td>
<td>The optional address sub-element can be used to tie the video device to a particular PCI slot.</td>
</tr>
</tbody>
</table>


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 consoles, channel and other devices configuration settings, use a management tool to make the following changes to the domain XML:

```xml
...<devices>
<serial type='pty'>
<source path='/dev/pts/3'/>
<target port='0'/>
</serial>
<channel type='unix'>
<source mode='bind' path='/tmp/guestfwd'/>
<target type='guestfwd' address='10.0.2.1' port='4600'/>
</channel>
...</devices>
...```

Figure J.59. 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. The guest interface is configured by the target element. The interface presented to the host is given in the type attribute of the top-level element. The host 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 be used to tie the device to a particular controller or PCI slot.
J.16.15. Guest interfaces

A character device presents itself to the guest as one of the following types.

To set the parallel port, use a management tool to make the following change to the domain XML:

```xml
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2'/>
    <target port='0'/>
  </parallel>
</devices>
```

*Figure J.60. Guest interface Parallel Port*

A `<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:

```xml
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3'/>
    <target port='0'/>
  </serial>
</devices>
```

*Figure J.61. Guest Interface Serial port*

A `<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 in use, the consoles might be paravirtualized devices, or they might be a clone of a serial device, according to the following rules:

1. 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 guests, the default device type will usually be a serial port.
2. 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.
3. If the `targetType` attribute is not `serial`, it will be treated normally.
4. Only the first `<console>` element may use a `targetType` of `serial`. Secondary consoles must all be paravirtualized.
5. 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 as `/dev/hvc[0-7]` (for more information, see http://fedoraproject.org/wiki/Features/VirtioSerial):
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.

**J.16.16. Channel**

This represents a private communication channel between the host and the guest and is manipulated by making changes to your guest virtual machine using a management tool that results in changes made to the following section of the domain xml.

```
...<device>
    <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>
    <channel type='spicevmc'>
        <target type='virtio'>
            name='com.redhat.spice.0'/>
        </channel>
</device>
...```

**Figure J.63. 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 to a given IP address and port is forwarded to the channel device on the host. The target element must have address and port attributes.

- **virtio** - Paravirtualized virtio channel. <channel> is exposed in the guest 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 agent installed in the guest, for actions such as guest 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 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 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.

**J.16.17. Host interface**

A character device presents itself to the host as one of the following types:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>XML snippet</th>
</tr>
</thead>
</table>
| Domain logfile     | Disables all input on the character device, and sends output into the virtual machine’s logfile | `<devices>
  <console type='stdio'>
  <target port='1'/>
  </console>
</devices>`                                                                                                                            |
| Device logfile     | A file is opened and all data sent to the character device is written to the file. | `<devices>
  <serial type="file">
  <source path="/var/log/vm/vm-serial.log"/>
  <target port='1'/>
  </serial>
</devices>`                                                                                                                            |
| 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” | `<devices>
  <serial type='vc'>
  <target port='1'/>
  </serial>
</devices>`                                                                                                                            |
| Null device        | Connects the character device to the void. No data is ever provided to the input. All data written is discarded. | `<devices>
  <serial type='null'>
  <target port='1'/>
  </serial>
</devices>`                                                                                                                            |
| Pseudo TTY         | A Pseudo TTY is allocated 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    | NB special case if `<console type='pty'>`, then the TTY path is also duplicated as an attribute ttyc='/dev/pts/3' on the top level `<console>` tag. This provides compat with existing syntax for `<console>` tags. | `<devices>
  <serial type='pty'>
  <source path="/dev/pts/3"/>
  <target port='1'/>
  </serial>
</devices>`                                                                                                                            |
| Host device proxy  | 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 serial port - don’t connect a serial port to a parallel port. | `<devices>
  <serial type="dev">
  <source path="/dev/ttyS0"/>
  <target port='1'/>
  </serial>
</devices>`                                                                                                                            |
<table>
<thead>
<tr>
<th>Device Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Named pipe</td>
<td>The character device writes output to a named pipe. See pipe(7) MAN page for more info.</td>
</tr>
<tr>
<td>TCP client/server</td>
<td>The character device acts as a TCP client connecting to a remote server.</td>
</tr>
<tr>
<td>TCP client/server</td>
<td>Or as a TCP server waiting for a client connection.</td>
</tr>
<tr>
<td>UDP</td>
<td>The character device</td>
</tr>
</tbody>
</table>

Alternatively you can use telnet instead of raw TCP. In addition, you can also use telnets (secure telnet) and tls.

```xml
<devices>
  <serial type="pipe">
    <source path="/tmp/mypipe"/>
    <target port="1"/>
  </serial>
</devices>
```

```xml
<devices>
  <serial type="tcp">
    <source mode="connect" host="0.0.0.0" service="2445"/>
    <protocol type="raw"/>
    <target port="1"/>
  </serial>
</devices>
```

```xml
<devices>
  <serial type="tcp">
    <source mode="bind" host="127.0.0.1" service="2445"/>
    <protocol type="raw"/>
    <target port="1"/>
  </serial>
</devices>
```

```xml
<devices>
  <serial type="tcp">
    <source mode="connect" host="0.0.0.0" service="2445"/>
    <protocol type="telnet"/>
    <target port="1"/>
  </serial>
</devices>
```

```xml
<devices>
  <serial type="tcp">
    <source mode="bind" host="127.0.0.1" service="2445"/>
    <protocol type="telnet"/>
    <target port="1"/>
  </serial>
</devices>
```
### J.17. Sound devices

A virtual sound card can be attached to the host via the sound element.

#### Figure J.64. Virtual sound card

The sound element has one mandatory attribute, \texttt{model}, which specifies what real sound device is emulated. Valid values are specific to the underlying hypervisor, though typical choices are \texttt{`es1370'}, \texttt{`sb16'}, \texttt{`ac97'}, and \texttt{`ich6'}. In addition, a sound element with \texttt{ich6} model can have optional sub-elements \texttt{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 \texttt{`duplex'} (advertises a line-in and a line-out) and \texttt{`micro'} (advertises a speaker and a microphone).

#### Figure J.65. Sound devices

Each sound element has an optional sub-element \texttt{<address>} which can tie the device to a particular PCI slot, documented above.

### J.18. Watchdog device

A virtual hardware watchdog device can be added to the guest via the \texttt{<watchdog>} element. The watchdog device requires an additional driver and management daemon in the guest. As merely enabling the watchdog in the \texttt{libvirt} configuration does not do anything useful on its own. Currently there is no support notification when the watchdog fires.
J.19. Memory balloon device

A virtual memory balloon device is added to all Xen and KVM/QEMU guests. 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 XML unless a specific PCI slot needs to be assigned. Note that if the membaloop device needs to be explicitly disabled, `<model>'none'</model>` may be used.

The following example automatically added device with KVM

```
...<devices>
  <memballoon model='virtio'/>
</devices>
...
```

Figure J.67. 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>
```

Figure J.68. 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.

J.20. Random number generator device

The virtual random number generator device allows the host to pass through entropy to guest operating systems.
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 interface for more information.

### J.21. TPM devices

The TPM device enables a QEMU guest to have access to TPM functionality. The TPM passthrough device type provides access to the host's TPM for one QEMU guest. No other software may be is using the TPM device, typically `/dev/tpm0`, at the time the QEMU guest 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>
...
```

The `model` attribute specifies what device model QEMU provides to the guest. 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's TPM device and `'passthrough'`. This backend type requires exclusive access to a TPM device on the host. 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.

### J.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:
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 with automatic resource relabeling active, an additional XML element, imagelabel, 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 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.
- **imagelabel** - 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.

### J.23. Example domain XML configuration

QEMU emulated guest on x86_64
Figure J.72. Example domain XML config
KVM hardware accelerated guest 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'>$hv</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 J.73. Example domain XML config

Revision History

Revision 0.2-0  Tue May 28 2013  Laura Novich
Completed merger of guides and prepared for final publishing.

Revision 0.1-0  Tue Jan 15 2013  Laura Bailey