



SUSE Linux Enterprise Server 11 SP4

Virtualization with KVM for IBM System z

Virtualization with KVM for IBM System z

SUSE Linux Enterprise Server 11 SP4

Publication Date: September 08, 2025

SUSE LLC
1800 South Novell Place
Provo, UT 84606
USA

<https://documentation.suse.com> 

Copyright © 2006– 2025 SUSE LLC and contributors. All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or (at your option) version 1.3; with the Invariant Section being this copyright notice and license. A copy of the license version 1.2 is included in the section entitled “GNU Free Documentation License”.

For SUSE trademarks, see <http://www.suse.com/company/legal/> . All other third party trademarks are the property of their respective owners. A trademark symbol (®, [™] etc.) denotes a SUSE or Novell trademark; an asterisk (*) denotes a third party trademark.

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither SUSE LLC, its affiliates, the authors nor the translators shall be held liable for possible errors or the consequences thereof.

Contents

About This Manual **x**

- 1 Available Documentation **x**
- 2 Feedback **xii**
- 3 Documentation Conventions **xiii**

I REQUIREMENTS, LIMITATIONS, AND SUPPORT STATUS 1

1 KVM Installation and Requirements 2

- 1.1 Supported Guest Operating Systems **2**
- 1.2 The kvm package **2**
- 1.3 Installing KVM **3**

2 KVM Limitations 4

- 2.1 General Limitations **4**
- 2.2 Hardware Limitations **4**

3 KVM Support Status 6

- 3.1 Supported Features and Tools **6**
- 3.2 Unsupported Features and Tools **8**

4 I/O Virtualization 10

II MANAGING VIRTUAL MACHINES WITH libvirt 12

5 Overview 13

6 Guest Installation 16

6.1 Guest Installation with Virtual Machine Manager 16

Customizing the Default Settings 17

6.2 Installing from the Command Line with **vm-install** 22

Defining a VM Guest without Starting the Installation 23

6.3 Advanced Guest Installation Scenarios 23

Including Add-On Products in the Installation 23

7 Basic VM Guest Management 24

7.1 Listing VM Guests 24

Listing VM Guests with Virtual Machine Manager 24 • Listing VM Guests with **virsh** 24

7.2 Opening a Graphical Console 25

Opening a Graphical Console with Virtual Machine Manager 25 • Opening a Graphical Console with **virt-viewer** 26

7.3 Changing a VM Guest's State: Start, Stop, Pause 26

Changing a VM Guest's State with Virtual Machine Manager 27 • Changing a VM Guest's State with **virsh** 28

7.4 Saving and Restoring VM Guests 29

Saving / Restoring with Virtual Machine Manager 29 • Saving / Restoring with **virsh** 30

7.5 Deleting a VM Guest 30

Deleting a VM Guest with Virtual Machine Manager 31 • Deleting a VM Guest with **virsh** 31

8 Connecting and Authorizing 32

8.1 Authentication 32

libvirtd Authentication 33 • VNC Authentication 38

8.2 Configuring Remote Connections 41

Remote Tunnel over SSH (qemu+ssh) 41 • Remote TLS/SSL Connection with x509 Certificate (qemu+tls) 42

8.3 Connecting to a VM Host Server 50

“system” Access for Non-Privileged Users 51 • Managing Connections with Virtual Machine Manager 52

9 Managing Storage 54

9.1 Managing Storage with Virtual Machine Manager 56

Adding a Storage Pool 57 • Managing Storage Pools 60

9.2 Managing Storage with **virsh** 62

Listing Pools and Volumes 62 • Starting, Stopping and Deleting Pools 63 • Adding Volumes to a Storage Pool 64 • Deleting Volumes from a Storage Pool 65

9.3 Locking Disk Files and Block Devices with **virtlockd** 66

Enable Locking 66 • Configure Locking 67

9.4 Online Resizing of Guest Block Devices 68

10 Configuring Virtual Machines 70

10.1 Enabling Seamless and Synchronized Cursor Movement 70

10.2 Adding a PCI Device with Virtual Machine Manager 71

10.3 Adding a PCI Device with **virsh** 72

10.4 Adding SR-IOV Devices 75

Requirements 75 • Load and configure the SR-IOV Host drivers 76 • Adding a VF network device to an Existing VM Guest 78

10.5 Clock Settings 80

Using **kvm_clock** 81 • Other Time Keeping Methods 81

11 Administrating VM Guests 83

11.1 Migrating VM Guests 83

Migrating with **virt-manager** 84 • Migrating with **virsh** 84

11.2 Monitoring 86

Monitoring with Virtual Machine Manager 86 • Monitoring with **kvm_stat** 87

III MANAGING VIRTUAL MACHINES WITH QEMU 89

12 QEMU Overview 90

13 Guest Installation 91

13.1 Basic Installation with **qemu-kvm** 91

13.2 Managing Disk Images with **qemu-img** 93

General Information on qemu-img Invocation 93 • Creating, Converting and Checking Disk Images 94 • Managing Snapshots of Virtual Machines with qemu-img 97 • Manipulate Disk Images Effectively 100

14 Running Virtual Machines with **qemu-kvm** 105

14.1 Basic **qemu-kvm** Invocation 105

14.2 General **qemu-kvm** Options 105

Basic Virtual Hardware 106 • Storing and Reading Configuration of Virtual Devices 108 • Guest Real-time Clock 109

14.3 Using Devices in QEMU 110

Block Devices 110 • Graphic Devices and Display Options 113 • Character Devices 114

14.4 Networking in QEMU 117

Defining a Network Interface Card 117 • User-mode Networking 118 • Bridged Networking 120 • Accelerated Networking with vhost-net 123

14.5 Viewing a VM Guest with VNC 123

Secure VNC Connections 124

14.6	VirtFS: Sharing Folders between Host and Guests	127
	Implementation	128
14.7	KSM: Sharing Memory Pages between Guests	128
15	KVM Disk Cache Modes	130
15.1	Disk Interface Cache Modes	130
15.2	Description of Cache Modes	130
15.3	Data Integrity Implications of Cache Modes	132
15.4	Performance Implications of Cache Modes	133
15.5	Effect of Cache Modes on Live Migration	133
16	Administrating Virtual Machines with QEMU Monitor	134
16.1	Accessing Monitor Console	134
16.2	Getting Information about the Guest System	134
16.3	Changing VNC Password	137
16.4	Managing Devices	137
16.5	Controlling Keyboard and Mouse	137
16.6	Changing Available Memory	138
16.7	Dumping Virtual Machine Memory	138
16.8	Managing Virtual Machine Snapshots	139
16.9	Suspending and Resuming Virtual Machine Execution	141
16.10	Live Migration	141
A	Appendix	143
A.1	Generating x509 Client/Server Certificates	143
A.2	QEMU Command Line Options	144
	Supported qemu-kvm Command Line Options	144 • Unsupported qemu-kvm Command Line Options
	Supported qemu-kvm monitor Command	147 • Supported qemu-kvm monitor Command

Line Options 149 • Unsupported **qemu-kvm** monitor Command Line Options 150

A.2.2 Unsupported **qemu-kvm** Command Line Options 147

A.2.3 Supported **qemu-kvm** monitor Command Line Options 149

A.2.4 Unsupported **qemu-kvm** monitor Command Line Options 150

B GNU Licenses 154

About This Manual

This manual offers an introduction to setting up and managing virtualization with KVM (Kernel-based Virtual Machine) on SUSE Linux Enterprise Server. The first part introduces KVM by describing its requirements and SUSE's support status. The second part deals with managing KVM with `libvirt`, while the last part covers management with QEMU.

Many chapters in this manual contain links to additional documentation resources. This includes additional documentation that is available on the system as well as documentation available on the Internet.

For an overview of the documentation available for your product and the latest documentation updates, refer to <http://www.suse.com/doc>.

1 Available Documentation

We provide HTML and PDF versions of our books in different languages. The following manuals for users and administrators are available for this product:

Book "Deployment Guide"

Shows how to install single or multiple systems and how to exploit the product inherent capabilities for a deployment infrastructure. Choose from various approaches, ranging from a local installation or a network installation server to a mass deployment using a remote-controlled, highly-customized, and automated installation technique.

Book "Administration Guide"

Covers system administration tasks like maintaining, monitoring, and customizing an initially installed system.

Book "Security Guide"

Introduces basic concepts of system security, covering both local and network security aspects. Shows how to make use of the product inherent security software like AppArmor (which lets you specify per program which files the program may read, write, and execute), and the auditing system that reliably collects information about any security-relevant events.

Book “Security and Hardening Guide”

Deals with the particulars of installing and setting up a secure SUSE Linux Enterprise Server, and additional post-installation processes required to further secure and harden that installation. Supports the administrator with security-related choices and decisions.

Book “System Analysis and Tuning Guide”

An administrator's guide for problem detection, resolution and optimization. Find how to inspect and optimize your system by means of monitoring tools and how to efficiently manage resources. Also contains an overview of common problems and solutions, and of additional help and documentation resources.

Book “Virtualization with Xen”

Offers an introduction to virtualization technology of your product. It features an overview of the various fields of application and installation types of each of the platforms supported by SUSE Linux Enterprise Server as well as a short description of the installation procedure.

Book “Virtualization with KVM for IBM System z”

Offers an introduction to setting up and managing virtualization with KVM (Kernel-based Virtual Machine) on SUSE Linux Enterprise Server. Learn how to manage KVM with libvirt or QEMU. The guide also contains detailed information about requirements, limitations, and support status.

Book “AutoYaST”

AutoYaST is a system for installing one or more SUSE Linux Enterprise systems automatically and without user intervention, using an AutoYaST profile that contains installation and configuration data. The manual guides you through the basic steps of auto-installation: preparation, installation, and configuration.

Book “Storage Administration Guide”

Provides information about how to manage storage devices on a SUSE Linux Enterprise Server.

In addition to the comprehensive manuals, several quick start guides are available:

Article “Installation Quick Start”

Lists the system requirements and guides you step-by-step through the installation of SUSE Linux Enterprise Server from DVD, or from an ISO image.

Linux Audit Quick Start


Gives a short overview how to enable and configure the auditing system and how to execute key tasks such as setting up audit rules, generating reports, and analyzing the log files.

AppArmor Quick Start

Helps you understand the main concepts behind AppArmor®.

Article “Virtualization with Linux Containers (LXC)”


Gives a short introduction to LXC (a lightweight “virtualization” method) and shows how to set up an LXC host and LXC containers.

Find HTML versions of most product manuals in your installed system under `/usr/share/doc/manual` or in the help centers of your desktop. Find the latest documentation updates at <http://www.suse.com/doc>  where you can download PDF or HTML versions of the manuals for your product.

2 Feedback


Several feedback channels are available:

Bugs and Enhancement Requests

For services and support options available for your product, refer to <http://www.suse.com/support/> .

To report bugs for a product component, log in to the Novell Customer Center from <http://www.suse.com/support/>  and select *My Support* > *Service Request*.

User Comments

We want to hear your comments about and suggestions for this manual and the other documentation included with this product. Use the User Comments feature at the bottom of each page in the online documentation or go to <http://www.suse.com/doc/feedback.html>  and enter your comments there.

Mail

For feedback on the documentation of this product, you can also send a mail to doc-team@suse.de. Make sure to include the document title, the product version, and the publication date of the documentation. To report errors or suggest enhancements, provide a concise description of the problem and refer to the respective section number and page (or URL).

3 Documentation Conventions

The following typographical conventions are used in this manual:

- /etc/passwd: directory names and filenames
- placeholder: replace placeholder with the actual value
- PATH: the environment variable PATH
- ls, --help: commands, options, and parameters
- user: users or groups
- **Alt** , **Alt** – **F1** : a key to press or a key combination; keys are shown in uppercase as on a keyboard
- *File*, *File* > *Save As*: menu items, buttons
- **IBM Z, ipseries** This paragraph is only relevant for the architectures System z and ipseries. The arrows mark the beginning and the end of the text block. ◀
- *Dancing Penguins* (Chapter *Penguins*, ↑Another Manual): This is a reference to a chapter in another manual.

I Requirements, Limitations, and Support Status

- 1 KVM Installation and Requirements 2
- 2 KVM Limitations 4
- 3 KVM Support Status 6
- 4 I/O Virtualization 10

1 KVM Installation and Requirements

KVM is a full virtualization solution supporting hardware virtualization. It consists of two main components: A set of Kernel modules providing the core virtualization infrastructure and processor specific drivers and a userspace program (`qemu-kvm`) that provides emulation for virtual devices and control mechanisms to manage VM Guests (virtual machines). The term KVM more properly refers to the Kernel level virtualization functionality, but is in practice more commonly used to reference the userspace component.

VM Guests (virtual machines), virtual storage and networks can be managed with `libvirt`-based and QEMU tools. `libvirt` is a library that provides an API to manage VM Guests based on different virtualization solutions, among them KVM and Xen. It offers a graphical user interface as well as a command line program. The QEMU tools are KVM/QEMU specific and are only available for the command line.

1.1 Supported Guest Operating Systems

KVM on SUSE Linux Enterprise for IBM System z only supports SLES 11 SP3 as a guest operating system.

1.2 The `kvm` package

The `kvm` package provides `qemu-kvm`, the program that performs the I/O emulation for the VM Guest. In addition to the `qemu-kvm` program, the `kvm` package also comes with a debug level monitoring utility (`kvm_stat`), firmware components, key-mapping files, and scripts.

Originally, the `kvm` package also provided the KVM Kernel modules. Now, these modules are included with the Kernel and only userspace components are included in the current `kvm` package.

Using the `libvirt`-based tools is the recommended way of managing VM Guests. Interoperability with other virtualization tools has been tested and is an essential part of SUSE's support stance. All tools are provided by packages carrying the tool's name.

- **libvirt**: A toolkit that provides management of VM Guests, virtual networks, and storage. **libvirt** provides an API, a daemon, and a shell (**virsh**).
- **virt-manager** (Virtual Machine Manager): A graphical management tool for VM Guests.
- **vm-install**: Define a VM Guest and install its operating system.
- **virt-viewer**: An X viewer client for VM Guests which supports TLS/SSL encryption of x509 certificate authentication and SASL authentication.

Support for creating and manipulating file-based virtual disk images is provided by **qemu-img**. **qemu-img** is provided by the package **virt-utils**.

1.3 Installing KVM

KVM is not installed by default. To install KVM and all virtualization tools, proceed as follows:

1. Start YaST and choose *Virtualization > Installing Hypervisor and Tools*.
2. Select *KVM* and confirm with *Accept*.
3. Confirm the list of packages that is to be installed with *Install*.
4. Agree to set up a network bridge by clicking *Yes*. It is recommended using a bridge on a VM Host Server (virtual machine host). If you prefer to manually configure a different network setup, you can safely skip this step by clicking *No*.
5. After the setup has been finished, reboot the machine as YaST suggests. Alternatively load the required kernel modules manually and start **libvirtd** to avoid a reboot:

```
modprobe kvm
rclibvirtd start
```


2 KVM Limitations

Although virtualized machines behave almost like physical machines, some limitations apply. These affect both, the VM Guest as well as the VM Host Server system.

2.1 General Limitations

The following general restrictions apply when using KVM:

Overcommits

KVM allows for both memory and disk space overcommit. It is up to the user to understand the implications of doing so. However, hard errors resulting from exceeding available resources will result in guest failures. CPU overcommit is also supported but carries performance implications.

MAC addresses

If no MAC address is specified for a NIC, a default MAC address will be assigned. This may result in network problems when more than one NIC receives the same MAC address. It is recommended to always assure a unique MAC address has been assigned for each NIC.

Live Migration

Live migration is not supported on IBM System z.

User Permissions

The management tools (Virtual Machine Manager, `virsh`, `vm-install`) need to authenticate with `libvirt`—see [Chapter 8, Connecting and Authorizing](#) for details. In order to invoke `qemu-kvm` from the command line, a user has to be a member of the group `kvm`.

Suspending/Hibernating the VM Host Server

Suspending or hibernating the VM Host Server system while guests are running is not supported.

2.2 Hardware Limitations

The following virtual hardware limits for guests have been tested. We ensure host and VMs install and work successfully, even when reaching the limits and there are no major performance regressions (CPU, memory, disk, network) since the last release.

<i>Max. Guest RAM Size</i>	4 TB
<i>Max. Virtual CPUs per Guest</i>	256
<i>Max. Virtual Network Devices per Guest</i>	8
<i>Max. Virtual Block Devices per Guest</i>	4 emulated (IDE), 20 para-virtual (using virtio-blk)
<i>Max. Number of VM Guests per VM Host Server</i>	Limit is defined as the total number of virtual CPUs in all guests being no greater than 8 times the number of CPU cores in the host.

The following hardware limits for the host server have been tested.

<i>Max. Physical CPUs</i>	4096
<i>Max. Physical Memory</i>	16 TB

3 KVM Support Status

The following list contains features and tools as supported by SUSE—this does not necessarily reflect the support status of the software itself. For a list of **qemu-kvm** command switches supported by SUSE, refer to [Section A.2, “QEMU Command Line Options”](#).

3.1 Supported Features and Tools

vm-install

Define and install VM Guests via **vm-install** including specifying the number of virtual processors, RAM, disk type and location, video type, keyboard mapping, NIC type, binding, MAC address, and boot method.

Restrictions: Currently only the **raw**, **qcow2** and **qed** disk formats are supported in read and write mode. The **vmdk**, **vpc** and **vhd / vhdx** formats are only supported in read-only mode. NIC creation is restricted to using virtio NICs. Sound cards are not supported.

Virtual Machine Manager

Manage guests via Virtual Machine Manager using the following functions: autostart, start, stop, restart, pause, unpause, save, restore, clone, migrate, special key sequence insertion, guest console viewers, performance monitoring, and CPU pinning. Furthermore, static modifications of CPU, RAM, boot method, disk, are supported.

virsh

Manage guests via the command line.

Most **virsh** subcommands are supported, including creation, modification, and destruction of guests and all life cycle operations. Any **virsh** subcommands which translate to unsupported qemu-kvm command-line or monitor syntax are also unsupported. Guest XML descriptions used by **virsh** can be created manually, using **vm-install**, the Virtual Machine Manager, or external tools and scripts.

qemu-kvm

Manage guests via the command line. Although managing via Virtual Machine Manager should be the preferred option, **qemu-kvm** may be used for greater flexibility. See [Section A.2.1, “Supported qemu-kvm Command Line Options”](#) for a list of supported options.

Restrictions: See [Section A.2.2, “Unsupported qemu-kvm Command Line Options”](#) for a list of not supported options.

Memory ballooning

Dynamically changing the amount of memory allocated to a guest is supported.

Sharing Folders between VM Host Server and VM Guest

Sharing folders between host and VM Guest is supported via VirtFs.

KVM Security

A `kvm` group is created by the KVM package, which permits a non-root user to access the KVM control device file (`/dev/kvm`). Where possible, guests should not be run as `root`. Steps have been taken to enable this for `libvirt` as well. A `setuid` bridge helper has been added so that a bridged network interface can be set up without needing `root` privileges.

Seccomp2 based sandboxing

The VM Guest can be run in a sandboxed environment where only predetermined system calls are permitted for added protection against malicious behavior.

APIC Virtualization

Hardware APIC Virtualization, allowing the processor to directly inject interrupts into the VM Guest to achieve better performance, is supported.

VirtFS (file system pass-through)

Directories in the host file system can be shared between the host and VM Guest or guests using `virtfs`. A `virtfs` proxy helper is provided to enable `virtfs` usage when KVM is used as non-root user.

Vhost-net kernel module support

The vhost-net kernel module allows for a more efficient network transport to the VM Guest. It is automatically used by `libvirt` if loaded, or when using the `qemu-kvm` command line, by adding `vhost=on` to the networking option.

AHCI guest storage interface

The AHCI interface for SATA storage has been recently added. It permits much higher block I/O performance than the IDE interface, and is particularly useful for use in recent Windows OS versions.

`qcow2` and `qed` storage formats

`qcow2` and `qed` storage formats can now be used with live migration.

Trim and Online Disk Resizing

Trim and online disk resizing support depends on the storage format used.

Virtio SCSI

Virtio SCSI allows for passing through host SCSI block or generic SCSI devices to the VM Guest, and provides additional storage options in a virtio SCSI interface within the guest.

Macvtap / vhost-net zero-copy transmits

Zero-copy packet transmits from the VM Guest are now possible using vhost-net and macvtap changes that have been added to the latest kernels.

Disk caching modes

The default caching mode for disk images is now *writeback* due to improvements in the handling of the image format. The `virtio-blk` back-end now automatically switches from 'writeback' to 'writethrough' if the VM Guest virtio driver does not support flushes.

Supported live migration scenarios

The following host operating system combinations are fully supported for live migrating guests from one host to another: SLES 11 SP3 to SLES 11 SP4, SLES 11 SP4 to SLES 11 SP4 and SLES 11 SP4 to SLES 12. When released, live migrating from SLES 11 SP4 to SLES 12 SP1 will be also supported.

Backwards migration is not supported: SLES 12 to SLES 11 SP4 and SLES 11 SP4 to SLES 11 SP3.

All supported guest systems can be migrated.

3.2 Unsupported Features and Tools

Power Management

Changing power states in the host while guests are running is not supported.

Spice

Spice interoperability is not supported.

Glusterfs

Glusterfs interoperability is not supported.

ISCSI

ISCSI integration is not supported. It is however possible for guests to access ISCSI targets available to the host via the blockio interfaces.

RBD (Rados Block Devices)

RBD integration is not supported.

CPU hotplugging

Dynamically changing the number of virtual CPUs assigned to the VM Guest is currently not supported.

KVM Kernel Module Parameters

Specifying parameters for the KVM Kernel modules is currently not supported unless done under the direction of SUSE support personnel.

Guest Agent

The guest agent (**qemu-ga**) allows programs on the VM Host Server to directly communicate with a VM Guest via an emulated or paravirtualized serial console. This feature is currently not supported.

Using QEMU without KVM (TCG (Tiny Code Generator) mode)

qemu-kvm can be invoked with the **-no-kvm** parameter. In this case VM Guest CPU instructions are emulated instead of being executed directly by the processor. This mode is not supported, but may be useful for problem resolution.

4 I/O Virtualization

VM Guests not only share CPU and memory resources of the host system, but also the I/O subsystem. Because software I/O virtualization techniques deliver less performance than bare metal, hardware solutions, that deliver almost "native" performance have been developed recently. SUSE Linux Enterprise Server supports the following I/O virtualization techniques:

Full Virtualization

Fully virtualized drivers emulate widely supported real devices, which can be used with an existing driver in the VM Guest. Since the physical device on the VM Host Server may differ from the emulated one, the hypervisor needs to process all I/O operations before handing them over to the physical device. Therefore all I/O operations need to traverse two software layers, a process that not only significantly impacts I/O performance, but also consumes CPU time.

Paravirtualization

Paravirtualization allows a direct communication between the hypervisor and the VM Guest. With less overhead involved, performance is much better than with full virtualization. However, paravirtualization requires either the guest operating system to be modified to support the paravirtualization API or paravirtualized drivers.

Direct Assignment via PCI-Passthrough

Directly assigning a PCI device to a VM Guest (PCI passthrough) avoids performance issues caused by avoiding any emulation in performance critical paths. With PCI passthrough, a VM Guest can directly access the real hardware using a native driver getting almost native performance. This method does not allow to share devices—each device can only be assigned to a single VM Guest. PCI-Passthrough needs to be supported by the VM Host Server CPU, chipset and the BIOS/EFI. The VM Guest needs to be equipped with drivers for the device. See [Section 10.2, “Adding a PCI Device with Virtual Machine Manager”](#) or [Section 10.3, “Adding a PCI Device with `virsh`”](#) for setup instructions.

Single Root I/O Virtualization (SR-IOV)

The latest I/O virtualization technique, SR-IOV combines the benefits of the aforementioned techniques— performance and the ability to share a device with several VM Guests. SR-IOV requires special I/O devices, which are capable of replicating resources, so they appear as multiple separate devices. Each such "pseudo" device can be directly used by a single guest. However, for network cards for example the number of concurrent queues that can be used is reduced, potentially reducing performance for the VM Guest compared

to paravirtualized drivers. On the VM Host Server SR-IOV must be supported by the I/O device, the CPU and chipset, the BIOS/EFI and the hypervisor. . See [Section 10.4, “Adding SR-IOV Devices”](#) for setup instructions.



Important: I/O Virtualization and Live Migration

Live migration is currently not supported when using devices with PCI passthrough or SR-IOV. In case live migration needs to be supported, you need to use software virtualization (paravirtualization or full Virtualization).

II Managing Virtual Machines with libvirt

- 5 Overview 13
- 6 Guest Installation 16
- 7 Basic VM Guest Management 24
- 8 Connecting and Authorizing 32
- 9 Managing Storage 54
- 10 Configuring Virtual Machines 70
- 11 Administrating VM Guests 83

5 Overview

libvirt is a library that provides a common API for managing popular virtualization solutions, among them KVM and Xen. The library provides a normalized management API for these virtualization solutions, allowing a stable, cross-hypervisor interface for higher-level management tools. The library also provides APIs for management of virtual networks and storage on the VM Host Server. The configuration of each VM Guest is stored in an XML file.

With libvirt you can also manage your VM Guests remotely. It supports TLS encryption and x509 certificates as well as authentication with SASL.

The communication between the virtualization solutions and libvirt is managed by the daemon libvirtd. It is also used by the management tools. libvirtd needs to run on the VM Host Server and on any remote machine on which the libvirt-based tools are started. Use the following commands to start, stop it or check its status:

```
~ # rclibvirtd start
Starting libvirtd                                done
~ # rclibvirtd status
Checking status of libvirtd                      running
~ # rclibvirtd stop
Shutting down libvirtd                          done
~ # rclibvirtd status
Checking status of libvirtd                      unused
```

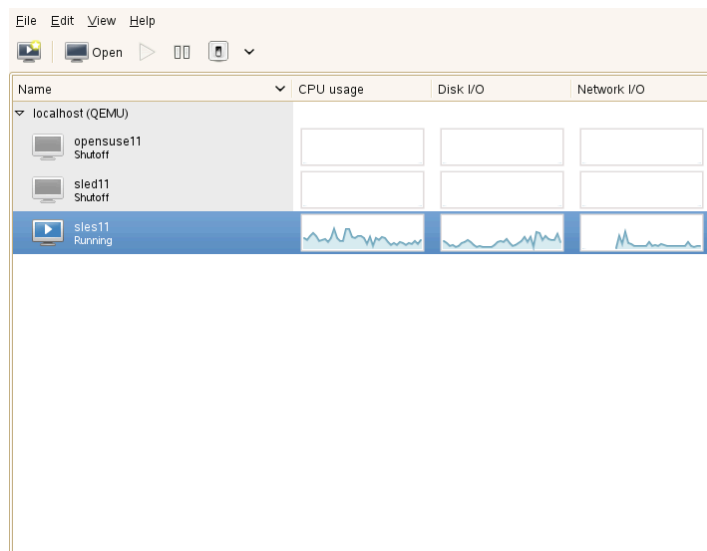
To automatically start libvirtd at boot time, either activate it using the YaST *System Services (Runlevel)* module or by entering the following command:

```
insserv libvirtd
```

The following libvirt-based tools are available on SUSE Linux Enterprise Server:

Virtual Machine Manager (virt-manager)

The Virtual Machine Manager is a desktop tool for managing VM Guests. It provides the ability to control the life cycle of existing machines (bootup/shutdown, pause/resume, suspend/restore). It lets you create new VM Guests and various types of storage, and manage virtual networks. Access the graphical console of VM Guests with the built-in VNC viewer, and view performance statistics, all done locally or remotely.

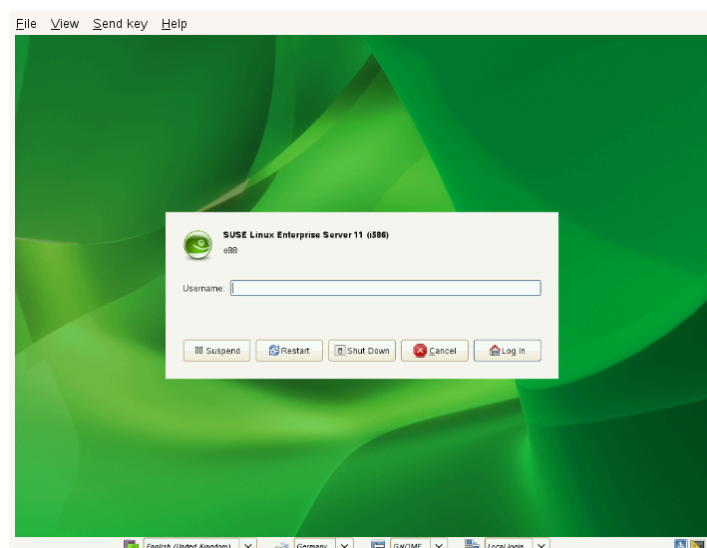


The Virtual Machine Manager does not need to run on the VM Host Server, it also lets you control VM Guests via remote connections. This enables you to manage VM Guests centrally from a single workstation without having to log in on the VM Host Server.

To start the Virtual Machine Manager, enter **virt-manager** at the command prompt.

virt-viewer

A viewer for the graphical console of a VM Guest. It uses the VNC protocol and supports TLS and x509 certificates. VM Guests can be accessed by name, ID, or UUID. If the guest is not already running, the viewer can be told to wait until the guest starts, before attempting to connect to the console.



vm-install

A tool to set up a VM Guest, configure its devices and start the operating system installation. Starts a GUI wizard when called from a graphical user interface. When invoked on a terminal, starts the wizard in command-line mode. vm-install is also started when creating a new virtual machine in the Virtual Machine Manager.

virsh

A command line tool to manage VM Guests with similar functionality as the Virtual Machine Manager. Allows you to change a VM Guest's status (start, stop, pause, etc.) to set up new guests and devices and to edit existing configurations. virsh is also useful to script VM Guest management operations.

virsh basically works like Subversion's svn command or zypper: it takes the first arguments as a command and further arguments as options to this command:

```
virsh [-c URI] commanddomain-id [OPTIONS]
```

Just like zypper, virsh can also be called without a command. In this case it starts a shell waiting for your commands. This mode is useful when having to run subsequent commands:

```
~> virsh -c qemu+ssh://wilber@mercury.example.com/system
Enter passphrase for key '/home/wilber/.ssh/id_rsa':
Welcome to virsh, the virtualization interactive terminal.

Type:  'help' for help with commands
       'quit' to quit

virsh # hostname
mercury.example.com
```

6 Guest Installation

A VM Guest is comprised of an image containing an operating system and data files and a configuration file describing the VM Guest's virtual hardware resources. VM Guests are hosted on and controlled by the VM Host Server.

6.1 Guest Installation with Virtual Machine Manager

Clicking *New* in the Virtual Machine Manager launches `vm-install`. It provides the graphical *Create Virtual Machine Wizard* that guides you through the guest installation. `vm-install` can also be run directly from the command-line or from YaST by choosing *Virtualization > Create Virtual Machine*.

1. Start the *Create Virtual Machine Wizard* as described above and click *Forward*.
2. Choose whether to install an operation system or an already existing image or disk.
3. Select the operating system you want to install from the list. Each entry provides reasonable defaults for the chosen operating system.
4. The *Summary* page shows the default configuration for the chosen operating system. Edit the configuration by clicking on a headline. When having chosen to install a system, you at least have to specify either an image or a CD/DVD device from which to boot or choose PXE boot. When accepting the configuration with *OK*, the guest system boots to start the installation.

6.1.1 Customizing the Default Settings

Change the proposed configuration by clicking on a headline in the Summary page of the Create Virtual Machine Wizard:

Summary

Click any headline to make changes. When the settings are correct, click **OK** to create the VM.

Name of Virtual Machine
Name: opensuse12
Description: openSUSE 12.1 RC 2

Hardware
Initial Memory: 768 MB
Maximum Memory: 768 MB
Virtual Processors: 1

Peripheral Devices
Graphics Adapter: Cirrus Logic GD5446 VGA
Keymap: de
Sound Card: None

Disks
1: 8.0 GB Hard Disk (qed:/virtual/opensuse_12.1_rc2)
2: 3.7 GB CD-ROM or DVD (file:/mounts/dist/install/openSUSE-12.1-RC2/iso/openSUSE-DVD-Build0025-x86_64.iso)

Network Adapters
1: QEMU Virtualized NIC Card; Randomly generated MAC address

Operating System Installation
Operating System: openSUSE 12
Installation Source: 3.7 GB CD-ROM or DVD (file:/mounts/dist/install/openSUSE-12.1-RC2/iso/openSUSE-DVD-Build0025-x86_64.iso)
Automated Installation:
Additional Arguments:

Cancel Back OK

6.1.1.1 Name of Virtual Machine

Specify a *Name* and an optional *Description* for the guest. The *Name* must contain only alphanumeric and `_ - . : +` characters. It must be unique among all VM Guests on the VM Host Server. It is used to create and name the guest's configuration file and you will be able to access the guest with this name from `virsh`.

6.1.1.2 Hardware

Change memory and CPU assignments in this screen. It is recommended not to specify values larger than the resources the VM Host Server can provide (overcommit), since it may result in errors or performance penalties.

This dialog also allows you to assign PCI devices (for example a network card) that can be directly used by the VM Guest (PCI passthrough). Click *Host Devices* › *Manage VM Devices* to get a list of available devices. Choose a device from the list and click *Add* to add it to the devices list

for the VM Guest. It is recommended to activate the *Managed* option for each device—it assures that `libvirt` automatically takes care of binding and unbinding drivers—see *Tip: 'managed' vs. 'unmanaged'* for more information.



Important: PCI devices cannot be shared

PCI devices cannot be shared between host and VM Guest or between VM Guests—each device can only be used by a single instance. Make sure to only add PCI devices not used elsewhere.

The *Advanced Settings* lets you activate or deactivate ACPI, APIC, and PAE. It is recommended not to change the default settings. You can also enable or disable paravirtualized I/O with `virtio` or choose to execute the kernel on boot (linux only) [here](#).



Important: Para-Virtualized I/O

If you enable paravirtualized I/O by activating `virtio`, all hard disks you create will be configured as `virtio` disks. If your operating system does not have appropriate drivers, the installation will fail. A Windows operating system installation even fails if providing a driver. By default, this feature is only activated for operating systems known to ship with `virtio` drivers.

6.1.1.3 *Peripheral Devices*

Configure the type of virtualized graphics hardware, the keymap and sound device in this dialog. If you disable the graphics card support, the machine is only accessible via network services (ssh) or serial port. Sound in VM Guests is currently not supported by SUSE, therefore *Sound* should be set to None.

6.1.1.4 *Disks*

Disks: Manage virtual hard disks and CD/DVD drives in this dialog. A VM Guest must have at least one virtual disk—either an existing one or a newly created disk. Virtual disks can be:

- a single file with a fixed size
- a single file that grows on demand (Sparse Image File)



Important: Sufficient Space for Sparse Image Files

When creating sparse image files, the partition on which you create them always needs sufficient free space. The VM Guest has no means to check the VM Host Server disk space. Having no space left on the host partition causes write errors and loss of data on the guest system.

- a block device, such as an entire disk, partition, or a network volume.

For best performance, create each virtual disk from an entire disk or a partition. For the next best performance, create an image file but do not create it as a sparse image file. A virtual disk based on a sparse image file delivers the most disk space flexibility but slows installation and disk access speeds.



Tip: Live Migration

If you need to be able to migrate your VM Guest to another host without shutting it down (live migration), all disks must reside on a network resource (network file system or iSCSI volume) that is accessible from both hosts.

By default, a single sparse raw disk image file is created in `/var/lib/kvm/images/VM_NAME/` where `VM_NAME` is the name of the virtual machine.



Note: Supported Disk format

Currently, only the disk formats `raw`, `qed` and `qcow2` are supported by SUSE with read and write support. The `vmdk`, `vpc` and `vhd/vhdx` formats are only supported in read-only mode.

PROCEDURE 6.1: CREATING A VIRTUAL DISK

1. Click *Harddisk*.
2. Enter a *Source*. If creating a file-backed disk, either enter the path directly or click *New*. When creating a disk from a device, enter the device node, for example `/dev/disk/by-path/path`. It is strongly recommended not to use the simple device paths such as `/dev/sdb` or `/dev/sda5`, since they may change (by adding a disk or by changing the disk order in the BIOS).

3. Specify the *Protocol*. For creating raw disks, choose either *file* for file-backed virtual disks or *phy* for device-backed disks. qcow2 or qed disks can be created by choosing the corresponding value.
4. Enter a *Size* in GB. This option is only available for file-backed disks.
5. Choose whether to create a *Sparse Image File*. This option is only available for file-backed disks. If you want to disable write-access to the disk, choose *Read-Only Access*.

If you want to install from DVD or CD-ROM, add the drive to the list of available hard disks. To learn about device nodes of the available optical drives, run:

```
hwinfo --cdrom | egrep "(Device File:|Model:)"
```

Instead of the real DVD or CD-ROM drive, you can also add the ISO image of an installation medium. Note that each CD-Rom drive or ISO image can only be used by one guest at the same time.

To add a CD/DVD-ROM device or an ISO image, proceed as follows:

1. Click *CD-ROM*.
2. Enter a *Source*. If adding a device, enter its node. If adding an ISO image, either enter the path directly or click *Browse* to open a file browser.
3. Specify the *Protocol*. Choose *file* for an ISO image and *phy* for a device.

The disks are listed in the order in which they have been created. This order also represents the boot order. Use the *Up* and *Down* arrows to change the disk order.

6.1.1.5 *Network Adapters*

By default, a single virtual network card is created for the virtual machine. It has a randomly generated MAC address that you can change to fit your desired configuration. If a bridge exists on the VM Host Server, the virtual network card will be attached to it, otherwise it will be attached to the libvirt's default virtual bridge (virbr1).

To add a new network adapter or edit an existing one, proceed as follows:

1. Click *New* to add a card or *Edit* to change the configuration of the selected card.
2. Choose a *Type* from the drop-down list.



Note: Supported Virtual Network Adapter Types

Currently, only *Fully Virtualized Realtek 8139*, *Fully Virtualized Intel e1000* or the paravirtualized *QEMU Virtualized NIC Card (virtio)* are supported by SUSE.

3. Choose a *Source* from the drop-down list.
4. Choose whether to assign a randomly generated MAC address or manually specify an address.



Note: MAC addresses need to be unique

When choosing to manually specify a MAC address, make sure it is not already used in your network. If so, it may result in network problems, especially when using DHCP. Therefore avoid specifying obvious MAC addresses such as 52:54:00:12:34:56 or 52:54:00:11:22:33, because they may already be in use. It is strongly recommended to always use a randomly generated MAC address for each adapter.

6.1.1.6 *Operating System Installation*

This dialog is only available when having chosen to install an operating system. The installation can be booted from a virtual disk, from a CD/DVD device, from an ISO image, a network resource or via PXE boot—use this dialog to configure the boot device.

Also use this dialog to configure the behavior of the VM Guest when the operating system is powered off, rebooted or if it crashes. The following options are available

destroy

normal cleanup

restart

a new VM Guest is started in place of the old one

preserve

no cleanup, do not delete temporary, configuration and image files

rename-restart

the VM Guest is not cleaned up but is renamed and a new domain started in its place

coredump-destroy

a crashed machine's core is dumped before a normal cleanup is performed

coredump-restart

a crashed machine's core is dumped before a normal restart is performed

6.2 Installing from the Command Line with **vm-install**

If `$DISPLAY` is not set (for example, when operating on a console or on an ssh shell with no X-forwarding), **vm-install** offers a command-line wizard to interactively set up a VM Guest for installation. Once the setup is completed, the newly created guest boots into the installation system which can be connected via VNC.



Important: Graphical User Interface needed for Installation

Once the VM Guest boots into the installation, you need to connect to the graphical console via VNC to attend the installation. Therefore, you need to start the viewer from a graphical user interface.

If you are working from a console with no access to a graphical user interface, you can set up the VM Guest configuration and start the installation at a later time. Refer to [Section 6.2.1, “Defining a VM Guest without Starting the Installation”](#) for instructions.

To start the wizard, just type **vm-install** to start. For a lot of parameters, the installation wizard already provides reasonable defaults which you can confirm by just pressing **Enter**. Here is a log of an interactive setup for a SUSE Linux Enterprise Server 11 installation:

EXAMPLE 6.1: **INTERACTIVE SETUP ON THE COMMAND LINE USING vm-install**

```
?????
```

You may also provide parameters on the command line. The wizard will then prompt you for any missing parameters. In the following all parameters from [Example 6.1, “Interactive Setup on the Command Line Using vm-install”](#) for which a command line switch exists, are specified. See **man 8 vm-install** for a full list of parameters.

EXAMPLE 6.2: **vm-install COMMAND LINE SWITCHES**

```
?????
```

6.2.1 Defining a VM Guest without Starting the Installation

`vm-install` provides the `--no-install` parameter. With this parameter the XML configuration file defining the VM Guest is created, but the guest is not booted automatically. You may use it regardless whether you start `vm-install` in wizard mode or whether you specify all other options in the command line. You can start the installation.



Warning: No Virtual Disk Creation

When using the `--no-install` parameter with `vm-install`, no virtual disks will be created. Therefore, you have to create the disks in advance using either `qemu-img` or `virsh`.

Once the VM Guest XML configuration file is successfully created, you need to “register” it so it is recognized by Virtual Machine Manager or `virsh`. Do so by running:

```
virsh -c qemu:///system define PATH_TO_XMLFILE
```

6.3 Advanced Guest Installation Scenarios

This section provides instructions for operations exceeding the scope of a normal installation, such as including add-on packages.

6.3.1 Including Add-On Products in the Installation

Some operating systems such as SUSE Linux Enterprise Server offer to include add-on products in the installation process. In case the add-on product installation source is provided via network, no special VM Guest configuration is needed. If it is provided via CD/DVD or ISO image, it is necessary to provide the VM Guest installation system with both, the standard installation images and the image for the add-on product.

First add the standard installation image, and second the physical CD/DVD-ROM or add-on image. The image or device added first is automatically chosen as the boot image. In case you install SUSE Linux Enterprise Server, it will be configured as `/dev/sr0`, while the add-on product source will be configured as `/dev/sr1`.

7 Basic VM Guest Management

Basic management tasks such as starting or stopping a VM Guest, can either be done using the graphical application Virtual Machine Manager or on the command line using **virsh**. Connecting to the graphical console via VNC is only possible from a graphical user interface.

7.1 Listing VM Guests

In order to be able to list VM Guests, you need to connect to a VM Host Server first. If you start the management tool on the VM Host Server itself, you are automatically connected. When operating from remote, refer to [Section 8.3, “Connecting to a VM Host Server”](#) for instructions.

7.1.1 Listing VM Guests with Virtual Machine Manager

The main Window of the Virtual Machine Manager shows a list of all VM Guests for each VM Host Server it is connected to. Each VM Guest entry contains the machine's name, its status (*Running*, *Paused*, or *Shutoff*) displayed as icon and literal, and a CPU usage bar.

7.1.2 Listing VM Guests with **virsh**

Use the command **virsh list** to get a list of VM Guests:

List running guests on localhost

```
virsh -c qemu:///system list
```

List running and inactive guests as user **wilber** on a remote host over a TLS connection

```
virsh -c qemu+tls://wilber@mercury.example.com/system list --all
```

List running and inactive guests as user **tux** on a remote host over an SSH tunnel

```
virsh -c qemu+ssh://tux@mercury.example.com/system list --inactive
```

7.2 Opening a Graphical Console

Opening a Graphical Console to a VM Guest lets you interact with the machine like a physical host via a VNC connection. If accessing the VNC server requires authentication, you are prompted to enter a username (if applicable) and a password.

Once you click into the VNC console, the cursor is “grabbed” and cannot be used outside the console anymore. To release it, press **Alt** – **Ctrl** .



Tip: Seamless (Absolute) Cursor Movement

In order to prevent the console from grabbing the cursor and to enable seamless cursor movement, add a tablet input device to the VM Guest. See [Section 10.1, “Enabling Seamless and Synchronized Cursor Movement”](#) for more information.

Certain key combinations such as **Ctrl** – **Alt** – **Del** are interpreted by the host system and are not passed to the VM Guest.

To pass such key combinations to a VM Guest, open the *Send Key* menu from the VNC window and choose the desired key combination entry. The *Send Key* menu is only available when using Virtual Machine Manager and **virt-viewer**



Note: Supported VNC Viewer

Principally all VNC viewers are able to connect to the console of a VM Guest. However, if you are using SASL authentication and/or TLS/SSL connection to access the guest, the options become limited. Common VNC viewers such as tightvnc or tigervnc support neither SASL authentication or TLS/SSL. The only supported alternative to Virtual Machine Manager and **virt-viewer** is vinagre.

7.2.1 Opening a Graphical Console with Virtual Machine Manager

1. In the Virtual Machine Manager, right-click a VM Guest entry.
2. Choose *Open* from the pop-up menu.

7.2.2 Opening a Graphical Console with **virt-viewer**

virt-viewer is a simple VNC viewer with added functionality for displaying VM Guest consoles. It can, for example, be started in “wait” mode, where it waits for a VM Guest to start before it connects. It also supports automatically reconnecting to a VM Guest that is rebooted.

virt-viewer addresses VM Guests by name, by ID or by UUID. Use **virsh list --all** to get this data.

To connect to a guest that is running or paused, either use the ID, UUID, or name. VM Guests that are shut off do not have an ID—you can only connect by UUID or name.

Local connect to guest with ID 8

```
virt-viewer -c qemu:///system 8
```

Local connect to the inactive guest sles11; will connect once the guest starts

```
virt-viewer -c qemu:///system --wait sles11
```

With the **--wait** option, the connection will be upheld even if the VM Guest is not running at the moment. Once the guest starts, the viewer will be launched.

Remote connect via ssh:

```
virt-viewer -c qemu+ssh://tux@mercury.example.com/system -w sles11
```

For more information, see **virt-viewer --help** or **man 1 virt-viewer**.

7.3 Changing a VM Guest's State: Start, Stop, Pause

Starting, stopping or pausing a VM Guest can either be done with Virtual Machine Manager or **virsh**. You can also configure a VM Guest to be automatically started when booting the VM Host Server.

When shutting down a VM Guest, you may either shut it down gracefully, or force the shutdown. The latter is equivalent to pulling the power plug on a physical host and is only recommended if there are no alternatives. Forcing a shutdown may cause file system corruption and loss of data on the VM Guest.



Tip: Graceful Shutdown

In order to be able to perform a graceful shutdown, the VM Guest must be configured to support ACPI. If you have created the guest with `vm-install` or with Virtual Machine Manager, ACPI should be available. Use the following procedure in Virtual Machine Manager to check:

Double-click the VM Guest entry in Virtual Machine Manager. Choose *View > Details* and then *Overview > Machine Settings*. *ACPI* should be checked.

Depending on the guest operating system, enabling ACPI may not be sufficient. It is strongly recommended to test shutting down and rebooting a guest before releasing it to production. openSUSE or SUSE Linux Enterprise Desktop, for example, may require PolicyKit authorization for shutdown and reboot. Make sure this policy is turned off on all VM Guests.

If ACPI was enabled during a Windows XP/Server 2003 guest installation, turning it on in the VM Guest configuration alone is not sufficient. See the following articles for more information:

<http://support.microsoft.com/kb/314088/EN-US/>

<http://support.microsoft.com/?kbid=309283>

A graceful shutdown is of course always possible from within the guest operating system, regardless of the VM Guest's configuration.

7.3.1 Changing a VM Guest's State with Virtual Machine Manager

Changing a VM Guest's state can either be done from Virtual Machine Manager's main window, or from a VNC window.

PROCEDURE 7.1: STATE CHANGE FROM THE VIRTUAL MACHINE MANAGER WINDOW

1. Right-click on a VM Guest entry.
2. Choose *Run*, *Pause*, or one of the *Shutdown options* from the pop-up menu.

PROCEDURE 7.2: STATE CHANGE FROM THE VNC WINDOW

1. Open a VNC Window as described in *Section 7.2.1, "Opening a Graphical Console with Virtual Machine Manager"*.

2. Choose *Run*, *Pause*, or one of the *Shut Down* options either from the toolbar or from the *Virtual Machine* menu.

7.3.1.1 Autostarting a VM Guest

Automatically starting a guest when the VM Host Server boots is not enabled by default. This feature needs to be turned on for each VM Guest individually. There is no way to activate it globally.

1. Double-click the VM Guest entry in Virtual Machine Manager to open its console.
2. Choose *View* > *Details* to open the VM Guest configuration window.
3. Choose *Boot Options* and check *Start virtual machine on host boot up*.
4. Save the new configuration with *Apply*.

7.3.2 Changing a VM Guest's State with **virsh**

In the following examples the state of a VM Guest named “sles11” is changed.

Start

```
virsh -c qemu:///system start sles11
```

Pause

```
virsh -c qemu:///system suspend sles11
```

Reboot

```
virsh -c qemu:///system reboot sles11
```

Graceful shutdown

```
virsh -c qemu:///system shutdown sles11
```

Force shutdown

```
virsh -c qemu:///system destroy sles11
```

Turn on autostart

```
virsh -c qemu:///system autostart sles11
```

Turn off autostart

```
virsh -c qemu:///system autostart --disable sles11
```

7.4 Saving and Restoring VM Guests

Saving a VM Guest preserves the exact state of the guest's memory. The operation is slightly similar to *hibernating* a computer. A saved VM Guest can be quickly restored to its previously saved running condition.

When saved, the VM Guest is paused, its current memory state is saved to disk, and then the guest is stopped. The operation does not make a copy of any portion of the VM Guest's virtual disk. The amount of time to save the virtual machine depends on the amount of memory allocated. When saved, a VM Guest's memory is returned to the pool of memory available on the VM Host Server. The restore operation loads a VM Guest's previously saved memory state file and starts it. The guest is not booted but rather resumes at the point where it was previously saved. The operation is slightly similar to coming out of hibernation.

The VM Guest is saved to a state file. Make sure there is enough space on the partition you are going to save to. Issue the following command on the guest to get a rough estimation of the file size in megabytes to be expected:

```
free -m | awk '/^Mem:/ {print $3}'
```



Warning

After using the save operation, do not boot, start, or run the saved VM Guest. Doing so would cause the machine's virtual disk and the saved memory state getting out of sync and can result in critical errors when restoring the guest.

7.4.1 Saving / Restoring with Virtual Machine Manager

PROCEDURE 7.3: SAVING A VM GUEST

1. Open a VNC connection window to a VM Guest. Make sure the guest is running.

2. Choose *Virtual Machine* > *Save*
3. Choose a location and a file name.
4. Click *Save*. Saving the guest's state may take some time. After the operation has finished, the VM Guest will automatically shut down.

PROCEDURE 7.4: RESTORING A VM GUEST

1. Start the Virtual Machine Manager.
2. Type **Alt + R** or choose *File* > *Restore Saved Machine*.
3. Choose the file you want to restore and proceed with *Open*. Once the file has been successfully loaded, the VM Guest is up and running.

7.4.2 Saving / Restoring with **virsh**

Save a running VM Guest with the command **virsh save** and specify the file to where it is saved.

Save the guest named `opensuse11`

```
virsh save opensuse11 /virtual/saves/opensuse11.vmsav
```

Save the guest with the ID 37

```
virsh save 37 /virtual/saves/opensuse11.vmsave
```

To restore it, use **virsh restore**:

```
virsh restore /virtual/saves/opensuse11.vmsave
```

7.5 Deleting a VM Guest

Deleting a VM Guest removes its XML configuration by default. Since the attached storage is not deleted by default, you will be able to use it with another VM Guest. With Virtual Machine Manager you may also delete a guest's storage files as well—this will completely erase the guest. In order to delete a VM Guest, it has to be shut down first (refer to [Section 7.3, “Changing a VM Guest's State: Start, Stop, Pause”](#) for instructions). It is not possible to delete a running guest.

7.5.1 Deleting a VM Guest with Virtual Machine Manager

1. In the Virtual Machine Manager, right-click a VM Guest entry.
2. Choose *Delete* from the pop-up menu.
3. A confirmation window opens. Clicking *Delete* will permanently erase the VM Guest. The deletion is not recoverable.
You may also choose to permanently delete the guest's virtual disk by ticking *Delete Associated Storage Files*. The deletion is not recoverable either.

7.5.2 Deleting a VM Guest with **virsh**

To delete a VM Guest with **virsh** run **virsh** `undefine VM_NAME`. There is no option to automatically delete the attached storage files.

8 Connecting and Authorizing

Having to manage several VM Host Servers, each hosting a couple of VM Guests, quickly becomes difficult to handle. One of the major benefits of `libvirt` is the ability to connect to several VM Host Servers at once, providing a single interface to manage all VM Guests and to connect to their graphical console.

In order to ensure only authorized users can connect, `libvirt` offers several connection types (via TLS, SSH, Unix sockets, and TCP) that can be combined with different authorization mechanisms (socket, PolicyKit, SASL and Kerberos).

8.1 Authentication

The power to manage VM Guests and to access their graphical console obviously is something that should be restricted to a well defined circle of persons. In order to achieve this goal, you can use the following authentication techniques on the VM Host Server:

- Access control for UNIX sockets with permissions and group ownership. This method is available for `libvirtd` connections only.
- Access control for UNIX sockets with PolicyKit. This method is available for local `libvirtd` connections only.
- Username and password authentication with SASL (Simple Authentication and Security Layer). This method is available for both, `libvirtd` and VNC connections. Using SASL does not require real user accounts on the server, since it uses its own database to store usernames and passwords. Connections authenticated with SASL are encrypted.
- Kerberos authentication. This method, available for `libvirtd` connections only, is not covered in this manual. Please refer to http://libvirt.org/auth.html#ACL_server_kerberos for details.
- Single password authentication. This method is available for VNC connections only.

Important: Authentication for `libvirtd` and VNC need to be configured separately

Access to the VM Guest management functions (via `libvirtd`) on the one hand and to their graphical console on the other hand, always needs to be configured separately. When restricting the access to the management tools, these restrictions do *not* automatically apply to VNC connections!

When accessing VM Guests from remote via TLS/SSL connections, access can be indirectly controlled on each client by restricting read permissions to the certificate's key file to a certain group. See [Section 8.2.2.5, "Restricting Access \(Security Considerations\)"](#) for details.

8.1.1 `libvirtd` Authentication

`libvirtd` authentication is configured in `/etc/libvirt/libvirtd.conf`. The configuration made here applies to all `libvirt` tools such as the Virtual Machine Manager or `virsh`.

`libvirt` offers two sockets: a read-only socket for monitoring purposes and a read-write socket to be used for management operations. Access to both sockets can be configured independently. By default, both sockets are owned by `root.root`. Default access permissions on the read-write socket are restricted to the user `root` (`0700`) and fully open on the read-only socket (`0777`).

In the following instructions you learn how to configure access permissions for the read-write socket. The same instructions also apply to the read-only socket. All configuration steps have to be carried out on the VM Host Server.

Note: Default Authentication Settings on SUSE Linux Enterprise Server

The default authentication method on SUSE Linux Enterprise Server is access control for UNIX sockets. Only the user `root` may authenticate. When accessing the `libvirt` tools as a non-root user directly on the VM Host Server, you need to provide the `root` password through PolicyKit once and are granted access for the current and for future sessions.

Alternatively you can configure `libvirt` to allow “system” access to non-privileged users. See [Section 8.3.1, ““system” Access for Non-Privileged Users”](#) for details.

RECOMMENDED AUTHORIZATION METHODS

Local Connections

Section 8.1.1.2, "Local Access Control for UNIX Sockets with PolicyKit"

Section 8.1.1.1, "Access Control for UNIX Sockets with Permissions and Group Ownership"

Remote Tunnel over SSH

Section 8.1.1.1, "Access Control for UNIX Sockets with Permissions and Group Ownership"

Remote TLS/SSL Connection

Section 8.1.1.3, "Username and Password Authentication with SASL"

none (access controlled on the client side by restricting access to the certificates)

8.1.1.1 Access Control for UNIX Sockets with Permissions and Group Ownership

In order to grant access for non- root accounts, configure the sockets to be owned and accessible by a certain group (libvirt in the following example). This authentication method can be used for local and remote SSH connections.

1. In case it does not exist, create the group which should own the socket:

```
groupadd libvirt
```



Important: Group Needs to Exist

The group must exist prior to restarting libvirtd. If not, the restart will fail.

2. Add the desired users to the group:

```
usermod -A libvirt tux
```

3. Change the configuration in /etc/libvirt/libvirtd.conf as follows:

```
unix_sock_group = "libvirt" ❶  
  unix_sock_rw_perms = "0770" ❷  
  auth_unix_rw = "none" ❸
```

- ❶ Group ownership will be set to group libvirt.

- ② Sets the access permissions for the socket (`srwxrwx---`).
- ③ Disables other authentication methods (PolicyKit or SASL). Access is solely controlled by the socket permissions.

4. Restart `libvirtd`:

```
rclibvirtd restart
```

8.1.1.2 Local Access Control for UNIX Sockets with PolicyKit

Access control for UNIX sockets with PolicyKit is the default authentication method on SUSE Linux Enterprise Server for non-remote connections. Therefore, no `libvirt` configuration changes are needed. With PolicyKit authorization enabled, permissions on both sockets default to `0777` and each application trying to access a socket needs to authenticate via PolicyKit. SUSE Linux Enterprise Server



Important: PolicyKit Authentication for Local Connections Only

Authentication with PolicyKit can only be used for local connections on the VM Host Server itself, since PolicyKit does not handle remote authentication.

Two policies for accessing `libvirt`'s sockets exist:

- `org.libvirt.unix.monitor`: accessing the read-only socket
- `org.libvirt.unix.manage`: accessing the read-write socket

By default, the policy for accessing the read-write socket is to authenticate with `root` password once and grant the privilege for the current and for future sessions (`auth_admin_keep_always`). In order to grant users access to the read-write socket without having to provide the `root` password, there are two possibilities:

1. Using the `polkit-auth` command, you can grant the privilege without any restrictions:

```
polkit-auth --user tux --grant org.libvirt.unix.manage # grant privilege
```



```
polkit-auth --user tux --revoke org.libvirt.unix.manage # revoke privilege
```

2. Editing `/etc/PolicyKit/PolicyKit.conf` offers more advanced options. Add the following XML snippet in between the existing `<config version="0.1">` and `</config>` tags:

```
<match action="org.libvirt.unix.manage">❶  
  <match user="tux">❷  
    <return result="yes"/>❸  
  </match>  
</match>
```

- ❶ The name of the policy; `org.libvirt.unix.manage` stands for accessing the read-write socket.
- ❷ The username(s) which to grant the privilege. Use the `|` symbol to separate entries (`user="tux|wilber"`).
- ❸ The privilege that is granted. The following options exist: `yes` (no restrictions), `no` (block access completely), `auth_self` or `auth_admin` (authenticate with own password/ `root` password every time the privilege is requested), `auth_self_keep_session` or `auth_admin_keep_session` (authenticate with own password/ `root` password once per session) and `auth_self_keep_always` or `auth_admin_keep_always` (authenticate only once with own password/ `root` password).

8.1.1.3 Username and Password Authentication with SASL

SASL provides username and password authentication as well as data encryption (digest-md5, by default). Since SASL maintains its own user database, the users do not need to exist on the VM Host Server. SASL is required by TCP connections and on top of TLS/SSL connections.



Important: Plain TCP and SASL with digest-md5 Encryption

Using digest-md5 encryption on an otherwise unencrypted TCP connection does not provide enough security for production environments. It is recommended to only use it in testing environments.



Tip: SASL Authentication on Top of TLS/SSL

Access from remote TLS/SSL connections can be indirectly controlled on the *client side* by restricting access to the certificate's key file. However, this might prove error-prone when dealing with a large number of clients. Utilizing SASL with TLS adds security by additionally controlling access on the server side.

To configure SASL authentication, proceed as follows:

1. Change the configuration in `/etc/libvirt/libvirtd.conf` as follows:

- a. To enable SASL for TCP connections:

```
auth_tcp = "sasl"
```

- b. To enable SASL for TLS/SSL connections:

```
auth_tls = "sasl"
```

2. Restart `libvirtd`:

```
rclibvirtd restart
```

3. The libvirt SASL configuration file is located at `/etc/sasl2/libvirtd.conf`. Normally, there is no need to change the defaults. However, if using SASL on top of TLS, you may turn off session encryption to avoid additional overhead— TLS connections are already encrypted— by commenting the `mech_list`. For TCP connections this parameter must be set to `digest-md5`:

```
mech_list: digest-md5    # mandatory for TCP connections
#mech_list: digest-md5    # apply default (username+password) TLS/SSL only!
```

4. By default, no SASL users are configured, so no logins are possible. Use the following commands to add, list, and delete users:

```
mercury:~ # saslpaswd2 -a libvirt tux          # add user tux
Password:
Again (for verification):
mercury:~ # sasldblistusers2 -f /etc/libvirt/passwd.db # list users
tux@mercury.example.com: userPassword
mercury:~ # saslpaswd2 -a libvirt -d tux        # delete user tux
```



Tip: **virsh** and SASL Authentication

When using SASL authentication you will be prompted for a username and password every time you issue a **virsh** command. Avoid this by using **virsh** in shell mode.

8.1.2 VNC Authentication

Since access to the graphical console of a VM Guest is not controlled by `libvirt`, but rather by `QEMU`, it is always necessary to additionally configure VNC authentication. The main configuration file is `/etc/libvirt/qemu.conf`.

Two authentication types are available: SASL and single password authentication. If you are using SASL for `libvirt` authentication, it is strongly recommended to use it for VNC authentication as well—it is possible to share the same database.

A third method to restrict access to the VM Guest is to enable the use of TLS encryption on the VNC server. This requires the VNC clients to have access to x509 client certificates. By restricting access to these certificates, access can indirectly be controlled on the client side. Refer to [Section 8.2.2.4.2, “VNC over TLS/SSL: Client Configuration”](#) for details.

8.1.2.1 Username and Password Authentication with SASL

SASL provides username and password authentication as well as data encryption. Since SASL maintains its own user database, the users do not need to exist on the VM Host Server. As with SASL authentication for `libvirt`, you may use SASL on top of TLS/SSL connections. Refer to [Section 8.2.2.4.2, “VNC over TLS/SSL: Client Configuration”](#) for details on configuring these connections.

To configure SASL authentication for VNC, proceed as follows:

1. Create a SASL configuration file. It is recommended to use the existing `libvirt` file. If you have already configured SASL for `libvirt` and are planning to use the same settings including the same username/password database, a simple link is suitable:

```
ln -s /etc/sasl2/libvirt.conf /etc/sasl2/qemu.conf
```

In case you are setting up SASL for VNC only or planning to use a different configuration than for `libvirt`, copy the existing file to use as a template and edit it according to your needs:

```
cp /etc/sasl2/libvirt.conf /etc/sasl2/qemu.conf
```

2. By default, no SASL users are configured, so no logins are possible. Use the following commands to add, list, and delete users:

```
mercury:~ # saslpasswd2 -a libvirt tux          # add user tux
Password:
Again (for verification):
mercury:~ # sasldblistusers2 -f /etc/libvirt/passwd.db # list users
tux@mercury.example.com: userPassword
mercury:~ # saslpasswd2 -a libvirt -d tux        # delete user tux
```

3. Change the configuration in `/etc/libvirt/qemu.conf` as follows:

```
vnc_listen = "0.0.0.0"
vnc_sasl = 1
```

The first parameter enables VNC to listen on all public interfaces (rather than to the local host only), and the second parameter enables SASL authentication.

4. Restart `libvirtd`:

```
rclibvirtd restart
```

5. Restart all VM Guests that have been running prior to changing the configuration. VM Guests that have not been restarted will not use SASL authentication for VNC connects.



Note: Supported VNC Viewers

Currently only the same VNC viewers that also support TLS/SSL connections, support SASL authentication, namely Virtual Machine Manager, **`virt-viewer`**, and **`vinagre`**.

8.1.2.2 Single Password Authentication

Access to the VNC server may also be controlled by setting a VNC password. You can either set a global password for all VM Guests or set individual passwords for each guest. The latter requires to edit the VM Guest's config files.



Note: Always Set a Global Password

If you are using the single password authentication, it is good practice to set a global password even if setting passwords for each VM Guest. This will always leave your virtual machines protected with a “fallback” password if you forget to set a per-machine password. The global password will only be used if no other password is set for the machine.

PROCEDURE 8.1: SETTING A GLOBAL VNC PASSWORD

1. Change the configuration in `/etc/libvirt/qemu.conf` as follows:

```
vnc_listen = "0.0.0.0"
vnc_password = "PASSWORD"
```

The first parameter enables VNC to listen on all public interfaces (rather than to the local host only), and the second parameter sets the password. The maximum length of the password is eight characters.

2. Restart `libvirtd`:

```
rclibvirtd restart
```

3. Restart all VM Guests that have been running prior to changing the configuration. VM Guests that have not been restarted will not use password authentication for VNC connects.

PROCEDURE 8.2: SETTING A VM GUEST SPECIFIC VNC PASSWORD

1. Change the configuration in `/etc/libvirt/qemu.conf` as follows to enable VNC to listen on all public interfaces (rather than to the local host only).

```
vnc_listen = "0.0.0.0"
```

2. Open the VM Guest's XML configuration file in an editor. Replace `VM NAME` in the following example with the name of the VM Guest. The editor that is used defaults to `$EDITOR`. If that variable is not set, `vi` is used.

```
virsh edit VM NAME
```

3. Search for the element `<graphics>` with the attribute `type='vnc'`, for example:

```
<graphics type='vnc' port='-1' autoport='yes' />
```

4. Add the `passwd=PASSWORD` attribute, save the file and leave the editor. The maximum length of the password is eight characters.

```
<graphics type='vnc' port='-1' autoport='yes' passwd='PASSWORD' />
```

5. Restart `libvirtd`:

```
rclibvirtd restart
```

6. Restart all VM Guests that have been running prior to changing the configuration. VM Guests that have not been restarted will not use password authentication for VNC connects.



Warning: Security

The VNC protocol is not considered to be safe. Although the password is sent encrypted, it might be vulnerable, when an attacker is able to sniff both, the encrypted password and the encryption key. Therefore, it is recommended to use VNC with TLS/SSL or tunneled over SSH. `virt-viewer`, as well as the Virtual Machine Manager and `vinagre` from version 2.30 on, support both methods.

8.2 Configuring Remote Connections

A major benefit of `libvirt` is the ability to manage VM Guests on different remote hosts from a central location. This section gives detailed instructions on how to configure server and client to allow remote connections.

8.2.1 Remote Tunnel over SSH (qemu+ssh)

Enabling a remote connection that is tunneled over SSH on the VM Host Server only requires the ability to accept SSH connections. Make sure the SSH daemon is started (`rcsshd status`) and that the ports for service `SSH` are opened in the firewall.

User authentication for SSH connections can be done using traditional file user/group ownership and permissions as described in [Section 8.1.1.1, "Access Control for UNIX Sockets with Permissions and Group Ownership"](#). Connecting as user `tux` (`qemu+ssh://tux@IVname;/system`) works out of the box and does not require additional configuration on the `libvirt` side.

When connecting via SSH `qemu+ssh://USER@SYSTEM` you need to provide the password for `USER`. This can be avoided by copying your public key to `~USER/.ssh/authorized_keys` on the VM Host Server as explained in *Book “Security Guide”, Chapter 14 “SSH: Secure Network Operations”, Section 14.5 “SSH Authentication Mechanisms”, Section 14.5.2 “Copying an SSH Key”*. Using an `ssh-agent` on the machine from which you are connecting adds even more convenience—see *Book “Security Guide”, Chapter 14 “SSH: Secure Network Operations”, Section 14.5 “SSH Authentication Mechanisms”, Section 14.5.3 “Using the **ssh-agent**”* for instructions.

8.2.2 Remote TLS/SSL Connection with x509 Certificate (qemu+tls)

Using TCP connections with TLS/SSL encryption and authentication via x509 certificates is much more complicated to set up than SSH, but it is a lot more scalable. Use this method if you have to manage several VM Host Servers with a varying number of administrators.

8.2.2.1 Basic concept

Basically, TLS (Transport Layer Security) encrypts the communication between two computers by using certificates. The computer starting the connection is always considered as the “client” using a “client certificate”, while the receiving computer is always considered as the “server”, using a “server certificate”. This scenario applies, for example, if you manage your VM Host Servers from a central desktop.

If connections are initiated from both computers, each needs to have a client *and* a server certificate. This is the case, for example, if you migrate a VM Guest from one host to another.

Each x509 certificate has a matching private key file. Only the combination of certificate and private key file is able to identify itself correctly. In order to assure that a certificate was issued by the assumed owner, it is signed and issued by a central certificate called certificate authority (CA). Both the client and the server certificates must be issued by the same CA.



Important: User Authentication

Using a remote TLS/SSL connection basically only ensures that two computers are allowed to communicate in a certain direction. Restricting access to certain users can indirectly be achieved on the client side by restricting access to the certificates. Refer to [Section 8.2.2.5](#),

“Restricting Access (Security Considerations)” for details. `libvirt` also supports user authentication on the server with SASL. Read more in [Section 8.2.2.6, “Central User Authentication with SASL for TLS Sockets”](#).

8.2.2.2 Configuring the VM Host Server

The VM Host Server is the machine receiving connections. Therefore, the *server* certificates have to be installed. The CA certificate needs to be installed, as well. Once the certificates are in place, TLS support can be turned on for `libvirt`.

1. Create the server certificate and export it together with the CA certificate as described in [Section A.1, “Generating x509 Client/Server Certificates”](#).
2. Create the following directories on the VM Host Server:

```
mkdir -p /etc/pki/CA/ /etc/pki/libvirt/private/
```

Install the certificates as follows:

```
/etc/pki/CA/cacert.pem  
/etc/pki/libvirt/servercert.pem  
/etc/pki/libvirt/private/serverkey.pem
```



Important: Restrict Access to Certificates

Make sure to restrict access to certificates as explained in [Section 8.2.2.5, “Restricting Access \(Security Considerations\)”](#).

3. Enable TLS support by editing `/etc/libvirt/libvirtd.conf` and setting `listen_tls = 1`. Restart `libvirtd`:

```
rclibvirtd restart
```

4. By default, `libvirt` uses the TCP port 16514 for accepting secure TLS connections. Open this port in the firewall.

! Important: Restarting `libvirtd` with TLS enabled

If you enable TLS for `libvirt`, the server certificates need to be in place, otherwise restarting `libvirtd` will fail. You also need to restart `libvirtd` in case you change the certificates.

8.2.2.3 Configuring the Client and Testing the Setup

The client is the machine initiating connections. Therefore the *client* certificates have to be installed. The CA certificate needs to be installed, as well.

1. Create the client certificate and export it together with the CA certificate as described in [Section A.1, "Generating x509 Client/Server Certificates"](#).
2. Create the following directories on the client:

```
mkdir -p /etc/pki/CA/ /etc/pki/libvirt/private/
```

Install the certificates as follows:

```
/etc/pki/CA/cacert.pem  
/etc/pki/libvirt/clientcert.pem  
/etc/pki/libvirt/private/clientkey.pem
```

! Important: Restrict Access to Certificates

Make sure to restrict access to certificates as explained in [Section 8.2.2.5, "Restricting Access \(Security Considerations\)"](#).

3. Test the client/server setup by issuing the following command. Replace `mercury.example.com` with the name of your VM Host Server. Specify the same full qualified hostname as used when creating the server certificate.

```
virsh -c qemu+tls://mercury.example.com/system list --all
```

If your setup is correct, you will see a list of all VM Guests registered with `libvirt` on the VM Host Server.

8.2.2.4 Enabling VNC for TLS/SSL connections

Currently, VNC communication over TLS is only supported by few tools. The widespread **tightvnc** or **tigervnc** viewer, for example, do not support TLS. Known to work are the Virtual Machine Manager (**virt-manager**), **virt-viewer** and the GNOME VNC viewer **vinagre**.

8.2.2.4.1 VNC over TLS/SSL: VM Host Server Configuration

In order to access the graphical console via VNC over TLS/SSL, you need to configure the VM Host Server as follows:

1. Open ports for the service **VNC** in your firewall.
2. Create a directory **/etc/pki/libvirt-vnc** and link the certificates into this directory as follows:

```
mkdir -p /etc/pki/libvirt-vnc && cd /etc/pki/libvirt-vnc
ln -s /etc/pki/CA/cacert.pem ca-cert.pem
ln -s /etc/pki/libvirt/servercert.pem server-cert.pem
ln -s /etc/pki/libvirt/private/serverkey.pem server-key.pem
```

3. Edit **/etc/libvirt/qemu.conf** and set the following parameters:

```
vnc_listen = "0.0.0.0"
vnc_tls = 1
vnc_tls_x509_verify = 1
```

4. Restart the **libvirtd**:

```
rclibvirtd restart
```



Important: VM Guests Need to be Restarted

The VNC TLS setting is only set when starting a VM Guest. Therefore, you need to restart all machines that have been running prior to making the configuration change.

8.2.2.4.2 VNC over TLS/SSL: Client Configuration

The only action needed on the client side is to place the x509 client certificates in a location recognized by the client of choice. Unfortunately, each supported client—Virtual Machine Manager, **virt-viewer**, and **vinagre**—expects the certificates in a different location. However, Virtual Machine Manager and **vinagre** can either read from a system-wide location applying to all users, or from a per user location.

Virtual Machine Manager (**virt-manager**)

In order to connect to the remote host, Virtual Machine Manager requires the setup explained in [Section 8.2.2.3, “Configuring the Client and Testing the Setup”](#). In order to be able to connect via VNC the client certificates also need to be placed in the following locations:

System wide location

/etc/pki/CA/cacert.pem
/etc/pki/libvirt-vnc/clientcert.pem
/etc/pki/libvirt-vnc/private/clientkey.pem

Per user location

/etc/pki/CA/cacert.pem
~/.pki/libvirt-vnc/clientcert.pem
~/.pki/libvirt-vnc/private/clientkey.pem

virt-viewer

virt-viewer only accepts certificates from a system wide location:

/etc/pki/CA/cacert.pem
/etc/pki/libvirt-vnc/clientcert.pem
/etc/pki/libvirt-vnc/private/clientkey.pem

vinagre

System wide location

/etc/pki/CA/cacert.pem
/etc/pki/vinagre/clientcert.pem
/etc/pki/vinagre/private/clientkey.pem

Per user location

\$HOME/.pki/CA/cacert.pem

```
~/.pki/vinagre/clientcert.pem  
~/.pki/vinagre/private/clientkey.pem
```



Important: Restrict Access to Certificates

Make sure to restrict access to certificates as explained in [Section 8.2.2.5, “Restricting Access \(Security Considerations\)”](#).

8.2.2.5 Restricting Access (Security Considerations)

Each x509 certificate consists of two pieces: the public certificate and a private key. A client can only authenticate using both pieces. Therefore, any user that has read access to the client certificate and its private key can access your VM Host Server. On the other hand, an arbitrary machine equipped with the full server certificate can pretend to be the VM Host Server. Since this is probably not desirable, access to at least the private key files needs to be restricted as much as possible. The easiest way to control access to a key file is to use access permissions.

Server Certificates

Server certificates need to be readable for QEMU processes. On SUSE Linux Enterprise Server QEMU processes started from `libvirt` tools are owned by `root`, so it is sufficient if `root` is able to read them certificates:

```
chmod 700 /etc/pki/libvirt/private/  
chmod 600 /etc/pki/libvirt/private/serverkey.pem
```

If you change the ownership for QEMU processes in `/etc/libvirt/qemu.conf`, you also need to adjust the ownership of the key file.

System Wide Client Certificates

To control access to a key file that is available system wide, restrict read access a certain group, so that only members of that group can read the key file. In the following example, a group `libvirt` is created and the group ownership of the `clientkey.pem` and its parent directory is set to `libvirt`. Afterwards, the access permissions are restricted to owner and group. Finally the user `tux` is added to the group `libvirt`, so he will be able to access the key file.

```
CERTPATH="/etc/pki/libvirt/"  
# create group libvirt
```

```
groupadd libvirt
# change ownership to user root and group libvirt
chown root.libvirt $CERTPATH/private $CERTPATH/clientkey.pem
# restrict permissions
chmod 750 $CERTPATH/private
chmod 640 $CERTPATH/private/clientkey.pem
# add user tux to group libvirt
usermod -A libvirt tux
```

Per User Certificates

User specific client certificates for accessing the graphical console of a VM Guest via VNC need to be placed in the users home directory in `~/.pki`. Contrary to, for example, the VNC viewer using these certificates do not check the access permissions of the private key file. Therefore, it is solely on the user's responsibility to make sure the key file is not readable by others.

8.2.2.5.1 Restricting Access from the Server Side

By default, every client that is equipped with appropriate client certificates may connect to a VM Host Server accepting TLS connections. Therefore, it is possible to use additional server side authentication with SASL as described in [Section 8.1.1.3, "Username and Password Authentication with SASL"](#).

It is also possible to restrict access with a whitelist of DN's (distinguished names), so only clients with a certificate matching a DN from the list can connect.

Add a list of allowed DN's to `tls_allowed_dn_list` in `/etc/libvirt/libvirtd.conf`. This list may contain wild cards. Do not specify an empty list, since that would result in refusing all connections.

```
tls_allowed_dn_list = [
    "C=US,L=Provo,O=SUSE Linux Products GmbH,OU=*,CN=venus.example.com,EMAIL=*",
    "C=DE,L=Nuremberg,O=SUSE Linux Products GmbH,OU=Documentation,CN=*" ]
```

Get the distinguished name of a certificate with the following command:

```
certtool -i --infile /etc/pki/libvirt/clientcert.pem | grep "Subject:"
```

Restart `libvirtd` after having changed the configuration:

```
rclibvirtd restart
```

8.2.2.6 Central User Authentication with SASL for TLS Sockets

A direct user authentication via TLS is not possible - this is handled indirectly on each client via the read permissions for the certificates as explained in [Section 8.2.2.5, “Restricting Access \(Security Considerations\)”](#). However, if a central, server based user authentication is needed `libvirt` also allows to use SASL (Simple Authentication and Security Layer) on top of TLS for direct user authentication. See [Section 8.1.1.3, “Username and Password Authentication with SASL”](#) for configuration details.

8.2.2.7 Troubleshooting

8.2.2.7.1 Virtual Machine Manager/`virsh` Cannot Connect to Server

Check the following in the given order:

Is it a firewall issue (TCP port 16514 needs to be open on the server)?

Is the client certificate (certificate and key) readable by the user that has started Virtual Machine Manager/`virsh`?

Has the same full qualified hostname as in the server certificate been specified with the connection?

Is TLS enabled on the server (`listen_tls = 1`)?

Has `libvirtd` been restarted on the server?

8.2.2.7.2 VNC Connection fails

Ensure that you can basically connect to the remote server using Virtual Machine Manager. If so, check whether the virtual machine on the server has been started with TLS support. The virtual machine's name in the following example is “sles11”.

```
ps ax | grep qemu | grep "\-name sles11" | awk -F" -vnc " '{ print FS $2 }'
```

If the output does not begin with a string similar to the following, the machine has not been started with TLS support and must be restarted.

```
-vnc 0.0.0.0:0,tls,x509verify=/etc/pki/libvirt
```

8.3 Connecting to a VM Host Server

In order to connect to a hypervisor with `libvirt`, you need to specify a uniform resource identifier (URI). This URI is needed with `virsh` and `virt-viewer` (except when working as `root` on the VM Host Server) and is optional for the Virtual Machine Manager. Although the latter can be called with a connection parameter (for example, `virt-manager -c qemu:///system`), it also offers a graphical interface to create connection URIs. See [Section 8.3.2, “Managing Connections with Virtual Machine Manager”](#) for details.

```
HYPERVISOR ❶ +PROTOCOL ❷ : //USER@REMOTE ❸ /CONNECTION_TYPE ❹
```

- ❶ Specify the hypervisor. SUSE Linux Enterprise Server currently supports the following hypervisors: `test` (dummy for testing), `qemu` (KVM), and `xen` (Xen). This parameter is mandatory.
- ❷ When connecting to a remote host, specify the protocol here. Can be one of: `ssh` (connection via SSH tunnel), `tcp` (TCP connection with SASL/Kerberos authentication), `tls` (TLS/SSL encrypted connection with authentication via x509 certificates).
- ❸ When connecting to a remote host, specify the user and the remote hostname. If no user is specified, the username that has called the command (`$USER`) is used. Please see below for more information. For TLS connections the hostname has to be specified exactly as in the x509 certificate.
- ❹ When connecting to `QEMU` hypervisor, two connections types are accepted: `system` for full access rights, or `session` for restricted access. Since `session` access is not supported on SUSE Linux Enterprise Server, this documentation focuses on `system` access.

EXAMPLE HYPERVISOR CONNECTION URIS

`test:///default`

Connect to the local dummy hypervisor. Useful for testing.

`qemu:///system`

Connect to the QEMU hypervisor on the local host having full access (type system).

`qemu+ssh://tux@mercury.example.com/system`

Connect to the QEMU hypervisor on the remote host `mercury.example.com`. The connection is established via an SSH tunnel.

`qemu+tls://saturn.example.com/system`

Connect to the QEMU hypervisor on the remote host `mercury.example.com`. The connection is established TLS/SSL.

For more details and examples, refer to the `libvirt` documentation at <http://libvirt.org/uri.html>.



Note: Usernames in URIs

A username needs to be specified when using Unix socket authentication (regardless whether using the username/password authentication scheme or PolicyKit). This applies to all SSH and local connections.

There is no need to specify a username when using SASL authentication (for TCP or TLS connections) or when doing no additional server side authentication for TLS connections. With SASL the username will not be evaluated—you will be prompted for a SASL user/password combination in any case.

8.3.1 “system” Access for Non-Privileged Users

As mentioned above, a connection to the QEMU hypervisor can be established using two different protocols: `session` and `system`. A “session” connection is spawned with the same privileges as the client program. Such a connection is intended for desktop virtualization, since it is restricted (for example no USB/PCI device assignments, no virtual network setup, limited remote access to `libvirtd`).

The “system” connection intended for server virtualization has no functional restrictions but is, by default, only accessible by `root`. However, with the addition of the DAC (Discretionary Access Control) driver to `libvirt` it is now possible to grant non-privileged users “system” access. To grant “system” access to the user `tux`, proceed as follows:

PROCEDURE 8.3: GRANTING “SYSTEM” ACCESS TO A REGULAR USER

1. Enable access via UNIX sockets as described in *Section 8.1.1.1, “Access Control for UNIX Sockets with Permissions and Group Ownership”*. In that example access to `libvirt` is granted to all members of the group `libvirt` and `tux` is made a member of this group. This ensures that `tux` can connect using `virsh` or Virtual Machine Manager.
2. Edit `/etc/libvirt/qemu.conf` and change the configuration as follows:

```
user = "tux"
group = "libvirt"
dynamic_ownership = 1
```


This ensures that the VM Guests are started by `tux` and that resources bound to the guest (for example virtual disks) can be accessed and modified by `tux`.

3. Make `tux` a member of the group `kvm`:

```
usermod -A kvm tux
```

This step is needed to grant access to `/dev/kvm` which is required to start VM Guests.

4. Restart `libvirtd`:

```
rclibvirtd restart
```

8.3.2 Managing Connections with Virtual Machine Manager

The Virtual Machine Manager uses a `Connection` for every VM Host Server it manages. Each connection contains all VM Guests on the respective host. By default, a connection to the local-host is already configured and connected.

All configured connections are displayed in the Virtual Machine Manager main window. Active connections are marked with a small triangle which you can click in order to fold or unfold the list of VM Guests for this connection.

Inactive connections are listed gray and are marked with `Not Connected`. Either double-click or right-click it and choose `Connect` from the context menu. You can also `Delete` an existing connection from this menu.



Note: Editing Existing Connections

It is not possible to edit an existing connection. In order to change a connection, create a new one with the desired parameters and delete the “old” one.

To add a new connection in the Virtual Machine Manager, proceed as follows:

1. Choose `File > Add Connection`
2. Choose the host's *Hypervisor* (*Xen* or *QEMU/KVM*)
3. Choose a *Connection* type—either *Local* for connecting to the host the Virtual Machine Manager was started on, or one of the remote connections (see [Section 8.2, “Configuring Remote Connections”](#) for more information).

4. In case of a remote connection, enter the *Hostname* of the remote machine as USER-NAME@REMOTE_HOST. Usernames must be specified for local connections as well as for SSH



Important: Specifying a Username

There is no need to specify a username for TCP and TLS connections; it will not be evaluated anyway. A username must be specified for local connections as well as for SSH connections—if not, the default user root will be used.

5. If you do not want the connection to be automatically activated when starting the Virtual Machine Manager, remove the tick from *Autoconnect*.
6. Finish the configuration by clicking *Connect*.

9 Managing Storage

When managing a VM Guest on the VM Host Server itself, it is possible to access the complete file system of the VM Host Server in order to attach or create virtual hard disks or to attach existing images to the VM Guest. However, this is not possible when managing VM Guests from a remote host. For this reason, `libvirt` supports so called “Storage Pools”, which can be accessed from remote machines.



Tip: CD/DVD ISO images

In order to be able to access CD/DVD ISO images on the VM Host Server from remote, they also need to be placed in a storage pool.

`libvirt` knows two different types of storage: volumes and pools.

Storage Volume

A storage volume is a storage device that can be assigned to a guest—a virtual disk or a CD/DVD/floppy image. Physically (on the VM Host Server) it can be a block device (a partition, a logical volume, etc.) or a file.

Storage Pool

A storage pool is a storage resource on the VM Host Server that can be used for storing volumes, similar to network storage for a desktop machine. Physically it can be one of the following types:

File System Directory (*dir*)

A directory for hosting image files. The files can be either one of the supported disk formats (raw, qcow2 and qed for read and write access, or mdk, vpc and vhd/vhdx in read-only mode), or ISO images.

Physical Disk Device (*disk*)

Use a complete physical disk as storage. A partition is created for each volume that is added to the pool.

Pre-Formatted Block Device (*fs*)

Specify a partition to be used in the same way as a file system directory pool (a directory for hosting image files). The only difference to using a file system directory is the fact that `libvirt` takes care of mounting the device.

iSCSI Target (iscsi)

Set up a pool on an iSCSI target. You need to have been logged into the volume once before, in order to use it with [libvirt](#) (use the YaST *iSCSI Initiator* to detect and log in to a volume, see *Book "Storage Administration Guide"* for details). Volume creation on iSCSI pools is not supported, instead each existing Logical Unit Number (LUN) represents a volume. Each volume/LUN also needs a valid (empty) partition table or disk label before you can use it. If missing, use [fdisk](#) to add it:

```
~ # fdisk -cu /dev/disk/by-path/ip-192.168.2.100:3260-iscsi-
iqn.2010-10.com.example:[...]-lun-2
Device contains neither a valid DOS partition table, nor Sun, SGI
or OSF disklabel
Building a new DOS disklabel with disk identifier 0xc15cdc4e.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

LVM Volume Group (logical)

Use an LVM volume group as a pool. You may either use a predefined volume group, or create a group by specifying the devices to use. Storage volumes are created as partitions on the volume.



Warning: Deleting the LVM-Based Pool

When the LVM-based pool is deleted in the Storage Manager, the volume group is deleted as well. This results in a non-recoverable loss of all data stored on the pool!

Multipath Devices (*mpath*)

At the moment, multipathing support is limited to assigning existing devices to the guests. Volume creation or configuring multipathing from within [libvirt](#) is not supported.

Network Exported Directory (*netfs*)

Specify a network directory to be used in the same way as a file system directory pool (a directory for hosting image files). The only difference to using a file system directory is the fact that `libvirt` takes care of mounting the directory. Supported protocols are NFS and glusterfs.

SCSI Host Adapter (*scsi*)

Use an SCSI host adapter in almost the same way as an iSCSI target. It is recommended to use a device name from `/dev/disk/by-*` rather than the simple `/dev/sdX`, since the latter may change (for example when adding or removing hard disks). Volume creation on iSCSI pools is not supported; instead, each existing LUN (Logical Unit Number) represents a volume.



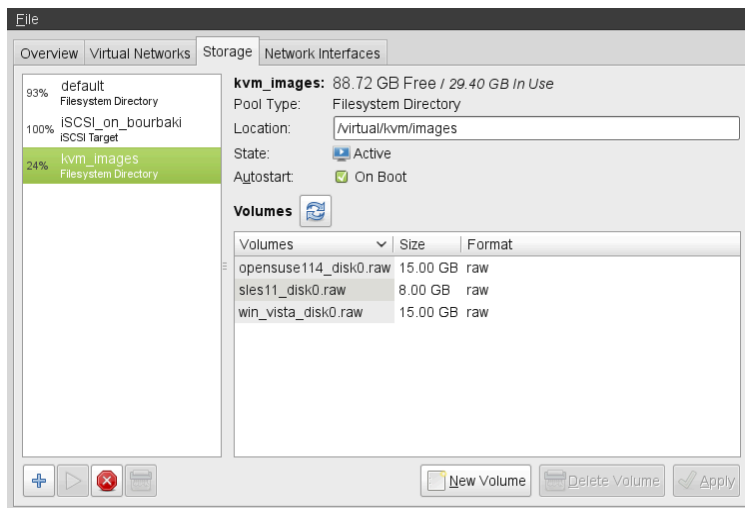
Warning: Security Considerations

In order to avoid data loss or data corruption, do not attempt to use resources such as LVM volume groups, iSCSI targets, etc. that are used to build storage pools on the VM Host Server, as well. There is no need to connect to these resources from the VM Host Server or to mount them on the VM Host Server—`libvirt` takes care of this.

Do not mount partitions on the VM Host Server by label. Under certain circumstances it is possible that a partition is labeled from within a VM Guest with a name already existing on the VM Host Server.

9.1 Managing Storage with Virtual Machine Manager

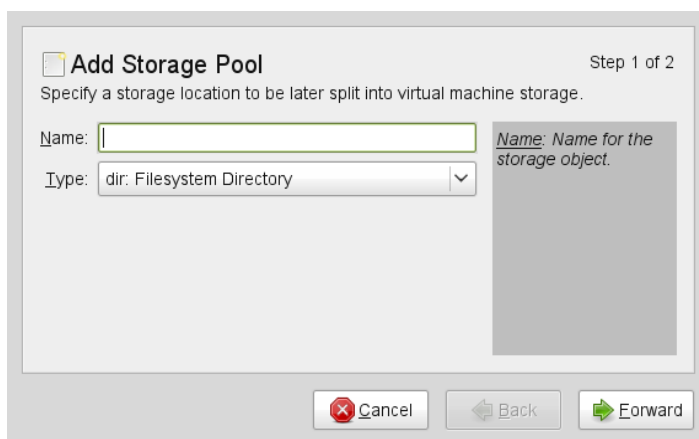
The Virtual Machine Manager provides a graphical interface—the Storage Manager—to manage storage volumes and pools. To access it, either right-click a connection and choose *Details*, or highlight a connection and choose *Edit > Connection Details*. Select the *Storage* tab.



9.1.1 Adding a Storage Pool

To add a storage pool, proceed as follows:

1. Click the plus symbol in the bottom left corner to open the *Add a New Storage Pool Window*.
2. Provide a *Name* for the pool (consisting of alphanumeric characters plus `_` and `.`) and select a *Type*. Proceed with *Forward*.



3. Specify the required details in the following window. The data that needs to be entered depends on the type of pool you are creating:

Typedir:

- *Target Path:* Specify an existing directory.

Typedisk:

- *Target Path:* The directory that hosts the devices. The default value `/dev` should fit in most cases.
- *Format:* Format of the device's partition table. Using `auto` should work in most cases. If not, get the required format by running the command `parted -l` on the VM Host Server.
- *Source Path:* Path to the device. It is recommended to use a device name from `/dev/disk/by-*` rather than the simple `/dev/sdX`, since the latter may change (for example when adding or removing hard disks). You need to specify the path that resembles the whole disk, not a partition on the disk (if existing).
- *Build Pool:* Activating this option formats the device. Use with care—all data on the device will be lost!

Typefs:

- *Target Path:* Mount point on the VM Host Server file system.
- *Format:* File system format of the device. The default value `auto` should work.
- *Source Path:* Path to the device file. It is recommended to use a device name from `/dev/disk/by-*` rather than the simple `/dev/sdX`, since the latter may change (for example when adding or removing hard disks).

Typeiscsi:

Get the necessary data by running the following command on the VM Host Server:

```
iscsiadm --mode node
```

It will return a list of iSCSI volumes with the following format. The elements highlighted with a bold font are the ones needed:

```
IP_ADDRESS:PORT,TPGT TARGET_NAME_(IQN)
```

- *Target Path*: The directory containing the device file. Use /dev/disk/by-path (default) or /dev/disk/by-id.
- *Host Name*: Host name or IP address of the iSCSI server.
- *Source Path*: The iSCSI target name (IQN).

Type*logical*:

- *Target Path*: In case you use an existing volume group, specify the existing device path. In case of building a new LVM volume group, specify a device name in the /dev directory that does not already exist.
- *Source Path*: Leave empty when using an existing volume group. When creating a new one, specify its devices here.
- *Build Pool*: Only activate when creating a new volume group.

Type*mpath*:

- *Target Path*: Support of multipathing is currently limited to making all multipath devices available. Therefore you may enter an arbitrary string here (required, otherwise the XML parser will fail); it will be ignored anyway.

Type*netfs*:

- *target Path*: Mount point on the VM Host Server file system.
- *Format*: Network file system protocol.
- *Host Name*: IP address or hostname of the server exporting the network file system.
- *Source Path*: Directory on the server that is being exported.

Typescsi:

- *Target Path*: The directory containing the device file. Use /dev/disk/by-path (default) or /dev/disk/by-id.
- *Source Path*: Name of the SCSI adapter.



Note: File Browsing

Using the file browser by clicking on *Browse* is not possible when operating from remote.

4. Click *Finish* to add the storage pool.

9.1.2 Managing Storage Pools

Virtual Machine Manager's Storage Manager lets you create or delete volumes in a pool. You may also temporarily deactivate or permanently delete existing storage pools. Changing the basic configuration of a pool is currently not supported by SUSE.

9.1.2.1 Starting, Stopping and Deleting Pools

The purpose of storage pools is to provide block devices located on the VM Host Server that can be added to a VM Guest when managing it from remote. In order to make a pool temporarily inaccessible from remote, you may *Stop* it by clicking on the stop symbol in the bottom left corner of the Storage Manager. Stopped pools are marked with *State: Inactive* and are grayed out in the list pane. By default, a newly created pool will be automatically started *On Boot* of the VM Host Server.

To *Start* an inactive pool and make it available from remote again click the play symbol in the bottom left corner of the Storage Manager.



Note: A Pool's State Does not Affect Attached Volumes

Volumes from a pool attached to VM Guests are always available, regardless of the pool's state (*Active* (stopped) or *Inactive* (started)). The state of the pool solely affects the ability to attach volumes to a VM Guest via remote management.

To permanently make a pool inaccessible, you can *Delete* it by clicking on the shredder symbol in the bottom left corner of the Storage Manager. You may only delete inactive pools. Deleting a pool does not physically erase its contents on VM Host Server—it only deletes the pool configuration. However, you need to be extra careful when deleting pools, especially when deleting LVM volume group-based tools:



Warning: Deleting Storage Pools

Deleting storage pools based on *local* file system directories, local partitions or disks has no effect on the availability of volumes from these pools currently attached to VM Guests.

Volumes located in pools of type iSCSI, SCSI, LVM group or Network Exported Directory will become inaccessible from the VM Guest if the pool is deleted. Although the volumes themselves will not be deleted, the VM Host Server will no longer have access to the resources.

Volumes on iSCSI/SCSI targets or Network Exported Directory will become accessible again when creating an adequate new pool or when mounting/accessing these resources directly from the host system.

When deleting an LVM group-based storage pool, the LVM group definition will be erased and the LVM group will no longer exist on the host system. The configuration is not recoverable and all volumes from this pool are lost.

9.1.2.2 Adding Volumes to a Storage Pool

Virtual Machine Manager lets you create volumes in all storage pools, except in pools of types Multipath, iSCSI, or SCSI. A volume in these pools is equivalent to a LUN and cannot be changed from within `libvirt`.

1. A new volume can either be created using the Storage Manager or while adding a new storage device to a VM Guest. In both cases, select a *Storage Pool* and then click *New Volume*.
2. Specify a *Name* for the image and choose an image format (note that SUSE currently only supports `raw`, `qcow2`, or `qed` images for read and write access). The latter option is not available on LVM group-based pools.
Specify a *Max Capacity* and the amount of space that should initially be allocated. If both values differ, a `sparse` image file, growing on demand, will be created.
3. Start the volume creation by clicking *Finish*.

9.1.2.3 Deleting Volumes From a Storage Pool

Deleting a volume can only be done from the Storage Manager, by selecting a volume and clicking *Delete Volume*. Confirm with *Yes*. Use this function with extreme care!



Warning: No Checks Upon Volume Deletion

A volume will be deleted in any case, regardless of whether it is currently used in an active or inactive VM Guest. There is no way to recover a deleted volume.

Whether a volume is used by a VM Guest is indicated in the *Used By* column in the Storage Manager.

9.2 Managing Storage with **virsh**

Managing storage from the command line is also possible by using **virsh**. However, creating storage pools is currently not supported by SUSE. Therefore this section is restricted to document functions like starting, stopping and deleting pools and volume management.

A list of all **virsh** subcommands for managing pools and volumes is available by running **virsh help pool** and **virsh help volume**, respectively.

9.2.1 Listing Pools and Volumes

List all pools currently active by executing the following command. To also list inactive pools, add the option **--all**:

```
virsh pool-list --details
```

Details about a specific pool can be obtained with the **pool-info** subcommand:

```
virsh pool-info POOL
```

Volumes can only be listed per pool by default. To list all volumes from a pool, enter the following command.

```
virsh vol-list --details POOL
```

At the moment **virsh** offers no tools to show whether a volume is used by a guest or not. The following procedure describes a way to list volumes from all pools that are currently used by a VM Guest.

PROCEDURE 9.1: LISTING ALL STORAGE VOLUMES CURRENTLY USED ON A VM HOST SERVER

1. Create an XSLT style sheet by saving the following content to a file, for example, `~/libvirt/guest_storage_list.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="text()"/>
  <xsl:strip-space elements="*" />
  <xsl:template match="disk">
    <xsl:text>  </xsl:text>
    <xsl:value-of select="(source/@file|source/@dev|source/@dir)[1]" />
    <xsl:text>&#10;</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

2. Run the following commands in a shell. It is assumed that the guest's XML definitions are all stored in the default location (`/etc/libvirt/qemu`). **xsltproc** is provided by the package `libxslt`.

```
SSHEET="$HOME/libvirt/guest_storage_list.xml"
cd /etc/libvirt/qemu
for FILE in *.xml; do
  basename $FILE .xml
  xsltproc $SSHEET $FILE
done
```

9.2.2 Starting, Stopping and Deleting Pools

Use the **virsh** pool subcommands to start, stop or delete a pool. Replace *P00L* with the pool's name or its UUID in the following examples:

Stopping a Pool

```
virsh pool-destroy P00L
```



Note: A Pool's State Does not Affect Attached Volumes

Volumes from a pool attached to VM Guests are always available, regardless of the pool's state (*Active* (stopped) or *Inactive* (started)). The state of the pool solely affects the ability to attach volumes to a VM Guest via remote management.

Deleting a Pool

```
virsh pool-delete POOL
```



Warning: Deleting Storage Pools

See [Warning: Deleting Storage Pools](#)

Starting a Pool

```
virsh pool-start POOL
```

Enable Autostarting a Pool

```
virsh pool-autostart POOL
```

Only pools that are marked to autostart will automatically be started if the VM Host Server reboots.

Disable Autostarting a Pool

```
virsh pool-autostart POOL --disable
```

9.2.3 Adding Volumes to a Storage Pool

virsh offers two ways to create storage pools: either from an XML definition with `vol-create` and `vol-create-from` or via command line arguments with `vol-create-as`. The first two methods are currently not supported by SUSE, therefore this section focuses on the subcommand `vol-create-as`.

To add a volume to an existing pool, enter the following command:

```
virsh vol-create-as POOL ❶NAME ❷ 12G --format ❸raw|qcow2|qed ❹ --allocation 4G ❺
```

- ❶ Name of the pool to which the volume should be added
- ❷ Name of the volume

- ③ Size of the image, in this example 12 gigabytes. Use the suffixes k, M, G, T for kilobyte, megabyte, gigabyte, and terabyte, respectively.
- ④ Format of the volume. SUSE currently supports `raw`, `qcow2`, and `qed` for read and write access.
- ⑤ Optional parameter. By default `virsh` creates a sparse image file that grows on demand. Specify the amount of space that should be allocated with this parameter (4 gigabytes in this example). Use the suffixes k, M, G, T for kilobyte, megabyte, gigabyte, and terabyte, respectively.

When not specifying this parameter, a sparse image file with no allocation will be generated. If you want to create a non-sparse volume, specify the whole image size with this parameter (would be `12G` in this example).

9.2.3.1 Cloning Existing Volumes

Another way to add volumes to a pool is to clone an existing volume. The new instance is always created in the same pool as the original.

```
virsh vol-clone NAME_EXISTING_VOLUME ① NAME_NEW_VOLUME ② --pool POOL ③
```

- ① Name of the existing volume that should be cloned
- ② Name of the new volume
- ③ Optional parameter. `libvirt` tries to locate the existing volume automatically. If that fails, specify this parameter.

9.2.4 Deleting Volumes from a Storage Pool

To permanently delete a volume from a pool, use the subcommand `vol-delete`:

```
virsh vol-delete NAME --pool POOL
```

`--pool` is optional. `libvirt` tries to locate the volume automatically. If that fails, specify this parameter.



Warning: No Checks Upon Volume Deletion

A volume will be deleted in any case, regardless of whether it is currently used in an active or inactive VM Guest. There is no way to recover a deleted volume.

Whether a volume is used by a VM Guest can only be detected by using by the method described in *Procedure 9.1, "Listing all Storage Volumes Currently Used on a VM Host Server"*.

9.3 Locking Disk Files and Block Devices with `virtlockd`

Locking block devices and disk files prevents concurrent writes to these resources from different VM Guests. It provides protection against starting the same VM Guest twice, or adding the same disk to two different virtual machines. This will reduce the risk of a virtual machine's disk image becoming corrupted as a result of a wrong configuration.

The locking is controlled by a daemon called `virtlockd`. Since it operates independently from the `libvirtd` daemon, locks will endure a crash or a restart of `libvirtd`. Locks will even persist in the case of an update of the `virtlockd` itself, since it has the ability to re-execute itself. This ensures that VM Guests do *not* have to be restarted upon a `virtlockd` update.



Note

`virtlockd` integration is only supported on a KVM VM Host Server.

9.3.1 Enable Locking

Locking virtual disks is not enabled by default on SUSE Linux Enterprise Server. To enable and automatically start it upon rebooting, perform the following steps:

1. Edit `/etc/libvirt/qemu.conf` and set

```
lock_manager = "lockd"
```

2. Start the `virtlockd` daemon with the following command:

```
rcvirtlockd start
```

3. Restart the `libvirtd` daemon with:

```
rclibvirtd restart
```

4. Make sure `virtlockd` is automatically started when booting the system:

```
insserv virtlockd
```

9.3.2 Configure Locking

By default `virtlockd` is configured to automatically lock all disks configured for your VM Guests. The default setting uses a "direct" lockspace, where the locks are acquired against the actual file paths associated with the VM Guest `<disk>` devices. For example, `flock(2)` will be called directly on `/var/lib/libvirt/images/my-server/disk0.raw` when the VM Guest contains the following `<disk>` device:

```
<disk type='file' device='disk'>
  <driver name='qemu' type='raw' />
  <source file='/var/lib/libvirt/images/my-server/disk0.raw' />
  <target dev='vda' bus='virtio' />
</disk>
```

The `virtlockd` configuration can be changed by editing the file `/etc/libvirt/qemu-lockd.conf`. It also contains detailed comments with further information. Make sure to activate configuration changes by reloading `virtlockd`:

```
rcvirtlockd reload
```



Note: Locking Currently Only Available for All Disks

As of SUSE Linux Enterprise Server 11 SP4 locking can only be activated globally, so that all virtual disks are locked. Support for locking selected disks is planned for future releases.

9.3.2.1 Enabling an Indirect Lockspace

`virtlockd`'s default configuration uses a "direct" lockspace, where the locks are acquired against the actual file paths associated with the `<disk>` devices. If the disk file paths are not accessible to all hosts, `virtlockd` can be configured to allow an "indirect" lockspace, where a hash of the disk file path is used to create a file in the indirect lockspace directory. The locks

are then held on these hash files instead of the actual disk file paths. Indirect lockspace is also useful if the file system containing the disk files does not support `fcntl()` locks. An indirect lockspace is specified with the `file_lockspace_dir` setting:

```
file_lockspace_dir = "/MY_LOCKSPACE_DIRECTORY"
```

9.3.2.2 Enable Locking on LVM or iSCSI Volumes

When wanting to lock virtual disks placed on LVM or iSCSI volumes shared by several hosts, locking needs to be done by UUID rather than by path (which is used by default). Furthermore, the lockspace directory needs to be placed on a shared file system accessible by all hosts sharing the volume. Set the following options for LVM and/or iSCSI:

```
lvm_lockspace_dir = "/MY_LOCKSPACE_DIRECTORY"
iscsi_lockspace_dir = "/MY_LOCKSPACE_DIRECTORY"
```

9.4 Online Resizing of Guest Block Devices

Sometimes you need to change—extend or shrink—the size of the block device used by your guest system. For example, when the disk space originally allocated is no longer enough, it is time to increase its size. If the guest disk resides on a *logical volume*, you can resize it while the guest system is running. This is a big advantage over an offline disk resizing package) as the service provided by the guest is not interrupted by the resizing process. To resize a VM Guest disk, follow these steps:

PROCEDURE 9.2: ONLINE RESIZING OF GUEST DISK

1. Inside the guest system, check the current size of the disk (for example `/dev/vda`).

```
root # fdisk -l /dev/vda
Disk /dev/sda: 160.0 GB, 160041885696 bytes, 312581808 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

2. On the host, resize the logical volume holding the `/dev/vda` disk of the guest to the required size, for example 200 GB.

```
root # lvresize -L 2048M /dev/mapper/vg00-home
```

```
Extending logical volume home to 2.00 GiB
Logical volume home successfully resized
```

3. On the host, resize the block device related to the disk `/dev/mapper/vg00-home` of the guest. Note that you can find the `domain_id` with **virsh list**.

```
root # virsh blockresize --path /dev/vg00/home --size 2048M domain_id
Block device '/dev/vg00/home' is resized
```

4. Check that the new disk size is accepted by the guest.

```
root # fdisk -l /dev/vda
Disk /dev/sda: 200.0 GB, 200052357120 bytes, 390727260 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

10 Configuring Virtual Machines

Virtual Machine Manager's *Details* view offers in-depth information about the VM Guest's complete configuration and hardware equipment. Using this view, you can also change the guest configuration or add and modify virtual hardware. To access this view, open the guests console in Virtual Machine Manager and either choose *View > Details* from the menu, or click the blue information icon in the toolbar.

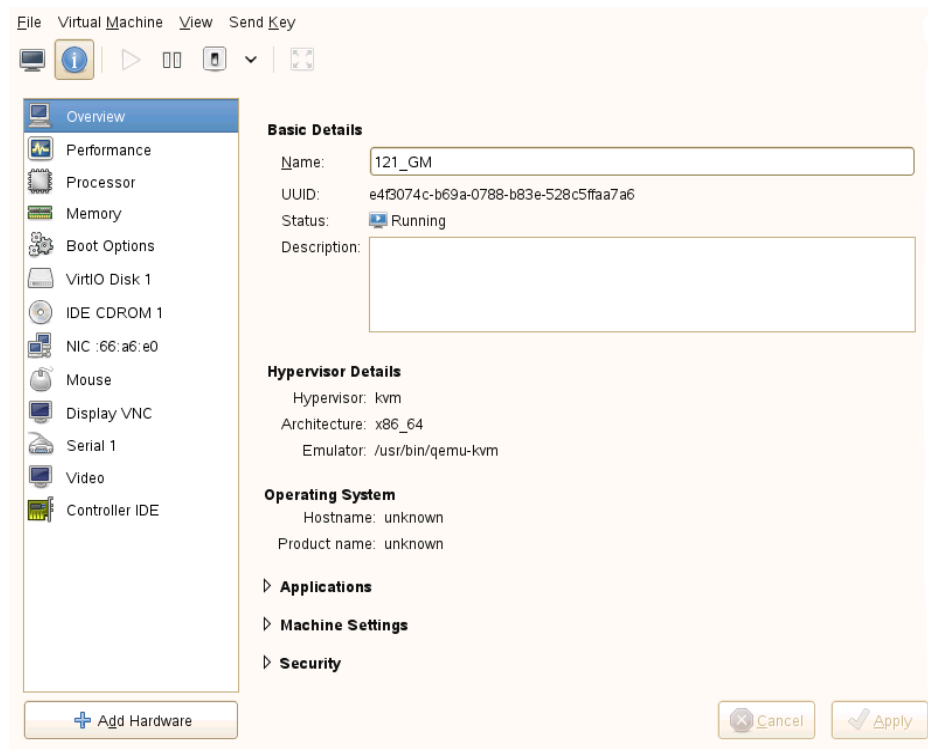


FIGURE 10.1: DETAILS VIEW OF A VM GUEST

10.1 Enabling Seamless and Synchronized Cursor Movement

When you click within a VM Guest's console with the mouse, the cursor is captured by the console window and cannot be used outside the console unless it is explicitly released (by pressing **Alt + Ctrl**). To prevent the console from grabbing the key and to enable seamless cursor movement between host and guest instead, add a tablet to the VM Guest.

Adding a tablet has the additional advantage of synchronizing the cursor movement between VM Host Server and VM Guest when using a graphical environment on the guest. With no tablet configured on the guest, you will often see two cursor symbols with one dragging behind the other.

1. Double-click a VM Guest entry in the Virtual Machine Manager to open its console and switch to the *Details* view with *View > Details*.
2. Click *Add Hardware* and choose *Input* and then *EvTouch USB Graphics Tablet* in the pop-up window. Proceed with *Finish*.
3. If you try to add the tablet while the guest is still running, you will be asked whether to enable the tablet after the next reboot. Confirm with *Yes*.
4. Once you (re)start the VM Guest, the tablet is available in the VM Guest.

10.2 Adding a PCI Device with Virtual Machine Manager

You can add PCI devices to guests using the graphical **virt-manager** tool. Once the PCI device is assigned to one VM Guest, it cannot be used by another one unless re-assigned. The following procedure adds a USB controller to a virtualized guest.

1. Double-click a VM Guest entry in the Virtual Machine Manager to open its console and switch to the *Details* view with *View > Details*.
2. Click *Add Hardware* and choose the *PCI Host Device* category in the left pane. A list of host PCI devices appears in the right part of the window.
3. From the list of available PCI devices, choose the USB controller to assign to the VM Guest. It may read for example USB2 Enhanced Host Controller. Confirm with *Finish*. *Finish*.



Tip

You cannot assign a PCI device live to a running VM Guest. If you are trying to add a PCI device to a running machine, the Virtual Machine Manager will ask if you want to assign the PCI device after the next VM Guest shutdown. Confirm with *Yes*, and after you reboot the VM Guest, the device assignment will be finished.

10.3 Adding a PCI Device with **virsh**

To dedicate and assign a PCI device to VM Guest with **virsh**, follow these steps:

1. Identify the PCI device.

Use the **virsh nodedev-list** or **lspci -n** commands to identify the PCI device designated for pass-through to VM Guest.

The following command lists available PCI devices only:

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

Note that PCI devices are identified by a string in the following format (**8086** is a variable that represents Intel architecture, and ******** stands for a four-digit hexadecimal code specific to each device):

```
pci_8086_****
```

Remember the PCI device number—you will need it in future steps.

2. Gather the information about the domain, bus, and function:

```
# virsh nodedev-dumpxml pci_8086_1d26
<device>
  <name>pci_8086_1d26</name>
  <parent>computer</parent>
  <driver>
    <name>ehci_hcd</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>0</bus>
    <slot>29</slot>
    <function>0</function>
    <product id="0x1d26">Patsburg USB2 Enhanced Host Controller #1</product>
    <vendor id="0x8086">Intel Corporation</vendor>
    <capability type='virt_functions'>
    </capability>
  </capability>
</device>
```

3. Detach the device from the host system prior to attaching it to VM Guest.

```
# virsh nodedev-detach pci_8086_1d26
Device pci_8086_1d26 detached
```

4. Convert the bus, slot, and function value from decimal to hexadecimal, and prepend '0x' to tell the system that the value is hexadecimal. In our example, bus = 0, slot = 29, and function = 0. Their hexadecimal values are:

```
# printf %x 0
0
# printf %x 29
1d
```

Bus and function hexadecimal numbers are '0x00', while slot number is '0x1d'.

5. Run **virsh edit** on your domain, and add the following device entry in the `<devices>` section.

```
<hostdev mode='subsystem' type='pci' managed='no'>
  <source>
    <address domain='0x0000' bus='0x00' slot='0x1d' function='0x00' />
  </source>
</hostdev>
```



Tip: 'managed' vs. 'unmanaged'

`libvirt` recognizes two modes for handling PCI devices: they can be either 'managed' or 'unmanaged'. In the managed case, `libvirt` will handle all the details of unbinding the device from the existing driver if needed, resetting the device, binding it to `pci-stub` before starting the domain, etc. When the domain is terminated or the device is removed from the domain, `libvirt` will unbind from `pci-stub` and rebind to the original driver in the case of a managed device. If the device is unmanaged, the user must take care to ensure all of these management aspects of the device are done before assigning it to a domain, and after the device is no longer used by the domain.

In our example, the `managed='no'` option means that the device is 'unmanaged', and we need to take care of the related driver with the **virsh nodedev-detach** and **virsh nodedev-attach** commands. To switch the device mode to 'managed', replace the snippet with `managed='yes'`, and skip the remaining steps (apart from starting the guest).

6. Once the VM Guest system is ready to use the PCI device, tell the host to stop using it. First check what driver the host system is using for the PCI device.

```
# readlink /sys/bus/pci/devices/0000\:00\:1d.0/driver
../../../../bus/pci/drivers/pci-stub
```

7. In our case, the pci-stub driver is loaded, so you can start the virtual machine. It will be able to use the PCI device automatically.

```
# virsh start sles11
```



Tip

When using a multi-function PCI device that does not support FLR (function level reset) or PM (power management) reset, you need to detach all its functions from the VM Host Server. The device must be reset for security reasons, and without FLR or PM reset, you must reset the whole device. libvirt will refuse to do this if a function of the device is still in use by the VM Host Server or another VM Guest.

You can safely detach a device function from the VM Guest with the virsh nodedev-detach command.



Tip

If your PCI device is not 'managed', and the driver controlling the PCI device is not pci-stub, you have to detach it from the device first:

```
virsh nodedev-detach pci_8086_1d26
```



Tip

If you are running SELinux on your host, you need to disable it for now with

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

and then start the virtual machine.

10.4 Adding SR-IOV Devices

Single Root I/O Virtualization (SR-IOV) capable PCIe devices are able to replicate their resources, so they appear to be multiple devices. Each of these "pseudo-devices" can be assigned to a VM Guest.

SR-IOV is an industry specification that was created by the Peripheral Component Interconnect Special Interest Group (PCI-SIG) consortium. It introduces physical functions (PF) and virtual functions (VF). PFs are full PCIe functions used to manage and configure the device. PFs also have the ability to move data. VFs lack the configuration and management part—they only have the ability to move data and a reduced set of configuration functions. Since VFs do not have all PCIe functions, the host operating system or the hypervisor must support SR-IOV in order to be able to access and initialize VFs. The theoretical maximum for VFs is 256 per device (consequently the maximum for a dual port Ethernet card would be 512). In practice this maximum is much lower, since each VF consumes resources.

10.4.1 Requirements

The following requirements must be met in order to be able to use SR-IOV:

- An SR-IOV-capable network card (as of SUSE Linux Enterprise Server 11 SP4 only network cards support SR-IOV)
- a x86_64 host supporting hardware virtualization (AMD-V or Intel VT-x)
- a chipset that supports device assignment (AMD-Vi or Intel VT-d)
- libvirt-0.9.10 or better
- SR-IOV drivers must be loaded and configured on the host system
- iommu enabled in the hypervisor (e.g. `intel_iommu=on` on the linux commandline of the host)
- a list of the PCI addresses of the VF(s) that will be assigned to VM Guests



Tip: Checking if a Device is SR-IOV Capable

The information whether a device is SR-IOV capable can be obtained from its PCI descriptor by running **lspci**. A device that supports SR-IOV reports a capability similar to the following:

```
Capabilities: [160 v1] Single Root I/O Virtualization (SR-IOV)
```

10.4.2 Load and configure the SR-IOV Host drivers

In order to be able to access and initialize VFs, an SR-IOV capable driver has to be loaded on the host system.

1. Before loading the driver, make sure the card is properly detected by running **lspci**. The following example shows the **lspci** output for the dual port Intel 82576NS network card:

```
~ > sudo /sbin/lspci | grep 82576
01:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
04:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
04:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
```

In case the card is not detected, it is likely that the hardware virtualization support in the BIOS/EFI has not been enabled.

2. Check whether the SR-IOV driver is already loaded by running **lsmod**. In the following example a check for the **igb** driver (for the Intel 82576NS network card) returns a result. That means the driver is already loaded. If the command returns nothing, the driver is not loaded.

```
~ > sudo /sbin/lsmod | egrep "^igb "
igb                185649  0
```

3. Skip this step if the driver is already loaded.

If the SR-IOV driver is not yet loaded, the non-SR-IOV driver needs to be removed first, before loading the new driver. Use `rmmod` to unload a driver. The following examples unload the non-SR-IOV driver for the Intel 82576NS network card:

```
sudo /sbin/rmmod igbvf
```

Load the SR-IOV driver subsequently using the `modprobe` command:

```
sudo /sbin/modprobe igb
```

4. Configure the driver by adding the number of VFs you would like to make available and—if necessary—by blacklisting the non-SR-IOV driver:

```
sudo echo -e "options igb max_vfs=8\nblacklist igbvf" >> /etc/modprobe.d/50-igb/
```

Make sure to replace the example values `igb`, `igbvf` and `50-igb` by values appropriate for your driver.

5. Now make sure the driver is loaded on boot. Edit `/etc/sysconfig/kernel` and add the driver to `MODULES_LOADED_ON_BOOT`:

```
MODULES_LOADED_ON_BOOT="igb"
```

Make sure to replace the example value `igb` by a value appropriate for your driver.

6. Reboot the machine and check if the SR-IOV driver is loaded by re-running the `lspci` command from the first step of this procedure. If the SR-IOV driver was loaded successfully you should see additional lines for the VFs:

```
01:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
01:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
01:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
01:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
[...]
04:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
04:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
(rev 01)
04:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
04:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
04:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
```

10.4.3 Adding a VF network device to an Existing VM Guest

Once the SR-IOV hardware is properly set up on the VM Host Server, you can add VFs to VM Guests. In order to do so, you need to collect some data first.

Note: The following procedure is using example data. Make sure to replace it by appropriate data from your setup.

1. Use the **virsh nodedev-list** command to get the PCI address of the VF you want to assign and its corresponding PF. Numerical values from the `lspci` output shown in [Section 10.4.2, "Load and configure the SR-IOV Host drivers"](#) (for example `01:00.0` or `04:00.1`) are transformed by adding the prefix "pci_0000_" and by replacing colons and dots with underscores. So a PCI ID listed as "04:00.0" by `lspci` is listed as "pci_0000_04_00_0" by `virsh`. The following example lists the PCI IDs for the second port of the Intel 82576NS network card:

```
~ > virsh nodedev-list | grep 0000_04_
pci_0000_04_00_0
pci_0000_04_00_1
pci_0000_04_10_0
pci_0000_04_10_1
pci_0000_04_10_2
pci_0000_04_10_3
pci_0000_04_10_4
pci_0000_04_10_5
pci_0000_04_10_6
pci_0000_04_10_7
pci_0000_04_11_0
pci_0000_04_11_1
pci_0000_04_11_2
pci_0000_04_11_3
pci_0000_04_11_4
pci_0000_04_11_5
```

The first two entries represent the PFs whereas the other entries represent the VFs.

2. Get more data that will be needed by running the command **virsh nodedev-dumpxml** on the PCI ID of the VF you want to add:

```
~ > virsh nodedev-dumpxml pci_0000_04_10_0
```

```

<device>
  <name>pci_0000_04_10_0</name>
  <parent>pci_0000_00_02_0</parent>
  <capability type='pci'>
    <domain>0</domain>
    <bus>4</bus>
    <slot>16</slot>
    <function>0</function>
    <product id="0x10ca">82576 Virtual Function</product>
    <vendor id="0x8086">Intel Corporation</vendor>
    <capability type='phys_function'>
      <address domain='0x0000' bus='0x04' slot='0x00' function='0x0' />
    </capability>
  </capability>
</device>

```

The following data is needed for the next step:

- <domain>0</domain>
- <bus>4</bus>
- <slot>16</slot>
- <function>0</function>

3. Create a temporary XML file (for example `/tmp/vf-interface.xml` containing the data necessary to add a VF network device to an existing VM Guest. The minimal content of the file needs to look like the following:

```

<interface type='hostdev'>❶
  <source>
    <address type='pci' domain='0' bus='11' slot='16' function='0'2/>❷
  </source>
</interface>

```

- ❶ VFs do not get a fixed MAC address, it changes everytime the host reboots. When adding network devices the “traditional” way with `<hostdev>`, it would require to reconfigure the VM Guest's network device after each reboot of the host, because of the MAC address change. To avoid this kind of problems, libvirt introduced the “`interface type='hostdev'`” directive, which sets up network specific data *before* assigning the device.
- ❷ Specify the data you acquired in the previous step here.

4. In case a device is already attached to the host, it cannot be attached to a guest. To make it available for guests, detach it from the host first:

```
virsh nodedev-detach pci_0000_04_10_0
```

5. Last, add the VF interface to an existing VM Guest:

```
virsh attach-device GUEST /tmp/vf-interface.xml --OPTION
```

GUEST needs to be replaced by the domain name, id or uuid of the VM Guest and --OPTION can be one of the following:

--persistent

This option will always add the device to the domain's persistent XML. In addition, if the domain is running, it will be hotplugged.

--config

This option will only affect the persistent XML, even if the domain is running. The device will only show up in the guest on next boot.

--live

This option will only affect a running domain. If the domain is inactive, the operation will fail. The device is not persisted in the XML and won't be available in the guest on next boot.

--current

This option affects the current state of the domain. If the domain is inactive, the device is added to the persistent XML and will be available on next boot. If the domain is active, the device is hotplugged but not added to the persistent XML.

To detach a VF interface, use the **virsh detach-device** command which also takes the options listed above.

10.5 Clock Settings

Keeping the correct time in a VM Guest is one of the more difficult aspects of virtualization. Keeping the correct time is especially important for network applications and is also a prerequisite to do a live migration of a VM Guest.



Tip: Time Keeping on the VM Host Server

It is strongly recommended to ensure the VM Host Server keeps the correct time as well, for example, by utilizing NTP (see *Book "Administration Guide", Chapter 24 "Time Synchronization with NTP"* for more information).

10.5.1 Using `kvm_clock`

KVM provides a paravirtualized clock which is currently supported by SUSE Linux Enterprise Server 10 SP3 and newer and RedHat Enterprise Linux 5.4 and newer via the `kvm_clock` driver. It is strongly recommended to use `kvm_clock` when available.

Use the following command inside a VM Guest running Linux to check whether the driver `kvm_clock` has been loaded:

```
~ # dmesg | grep kvm-clock
[ 0.000000] kvm-clock: cpu 0, msr 0:7d3a81, boot clock
[ 0.000000] kvm-clock: cpu 0, msr 0:1206a81, primary cpu clock
[ 0.012000] kvm-clock: cpu 1, msr 0:1306a81, secondary cpu clock
[ 0.160082] Switching to clocksource kvm-clock
```

To check which clock source is currently used, run the following command in the VM Guest. It should output `kvm-clock`:

```
cat /sys/devices/system/clocksource/clocksource0/current_clocksource
```



Important: `kvm-clock` and NTP

When using `kvm-clock`, it is recommended to use NTP in the VM Guest, as well. Using NTP on the VM Host Server is also recommended.

10.5.2 Other Time Keeping Methods

The paravirtualized `kvm-clock` is currently not available for SUSE Linux Enterprise Server 9 and Windows operating systems. For Windows, use the [Windows Time Service Tools](http://technet.microsoft.com/en-us/library/cc773263%28WS.10%29.aspx) for time synchronization (see <http://technet.microsoft.com/en-us/library/cc773263%28WS.10%29.aspx> for more information).

Correct time keeping in SUSE Linux Enterprise Server 9 SP4 can be achieved by using special boot parameters:

32-bit Kernel: clock=pmtmr

64-bit Kernel: ignore_lost_ticks

11 Administrating VM Guests

11.1 Migrating VM Guests

One of the major advantages of virtualization is the fact that VM Guests are portable. When a VM Host Server needs to go down for maintenance, or when the host gets overloaded, the guests can easily be moved to another VM Host Server. KVM and Xen even support “live” migrations during which the VM Guest is constantly available.

In order to successfully migrate a VM Guest to another VM Host Server, the following requirements need to be met:

- Host and target must have same processor manufacturer (Intel or AMD).
- Storage devices must be accessible from both machines (for example, via NFS or iSCSI) and must be configured as a storage pool on both machines (see [Chapter 9, Managing Storage](#) for more information).
- `libvirtd` needs to run on both VM Host Servers and you must be able to open a remote `libvirt` connection between the target and the source host (or vice versa). Refer to [Section 8.2, “Configuring Remote Connections”](#) for details.
- If a firewall is running on the target host ports need to be opened to allow the migration. If you do not specify a port during the migration process, `libvirt` chooses one from the range 49152:49215. Make sure that either this range (recommended) or a dedicated port of your choice is opened in the firewall on the *target host*.
- Host and target machine should be in the same subnet on the network, otherwise networking will not work after the migration.
- No running or paused VM Guest with the same name must exist on the target host. If a shut down machine with the same name exists, its configuration will be overwritten.
- Only the default CPU model (qemu64) should be used when migrating VM Guests.
- SATA disk device type is not migratable.
- File system pass-through feature is incompatible with migration.

11.1.1 Migrating with **virt-manager**

When using the Virtual Machine Manager to migrate VM Guests, it does not matter on which machine it is started. You can start Virtual Machine Manager on the source or the target host or even on a third host. In the latter case you need to be able to open remote connections to both the target and the source host.

1. Start Virtual Machine Manager and establish a connection to the target or the source host. If the Virtual Machine Manager was started neither on the target nor the source host, connections to both hosts need to be opened.
2. Right-click on the VM Guest that is to be migrated and choose *Migrate*. Make sure the guest is running or paused - it is not possible to migrate guests that are shut off.
3. Choose a *New Host* for the VM Guest. If the desired target host does not show up, make sure a connection to this host has been established.
By default, a “live” migration is performed. If you prefer an “offline” migration where the VM Guest is paused during the migration, tick *Migrate offline*.
4. Click *Migrate* to start a migration with the default port and bandwidth.
In order to change these defaults, make the advanced options available by clicking the triangle at *Advanced Options*. Here you can enter the target host's *Address* (IP address or hostname), a port and the bandwidth in megabit per second (Mbps). If you specify a *Port*, you must also specify an *Address*; the *Bandwidth* is optional.
5. Once the migration is complete, the *Migrate* window closes and the VM Guest is now listed on the new host in the Virtual Machine Manager Window. The original VM Guest will still be available on the target host (in state shut off).

11.1.2 Migrating with **virsh**

To migrate a VM Guest with **virsh migrate**, you need to have direct or remote shell access to the VM Host Server, because the command needs to be run on the host. Basically the migration command looks like this

```
virsh migrate [OPTIONS] VM_ID_or_NAMECONNECTION URI [--migrateuri tcp://REMOTE_HOST:PORT]
```

The most important options are listed below. See **virsh help migrate** for a full list.

--live

Does a live migration. If not specified, an offline migration where the VM Guest is paused during the migration, will be performed.

--suspend

Does an offline migration and does not restart the VM Guest on the target host.

--persistent

By default a migrated VM Guest will be migrated transient, so its configuration is automatically deleted on the target host if it is shut down. Use this switch to make the migration persistent.

--undefinesource

When specified, the VM Guest definition on the source host will be deleted after a successful migration (however, virtual disks attached to this guest will *not* be deleted).

The following examples use `mercury.example.com` as the source system and `jupiter.example.com` as the target system, the VM Guest's name is `opensuse11` with Id `37`.

Offline migration with default parameters

```
virsh migrate 37 qemu+ssh://tux@jupiter.example.com/system
```

Transient live migration with default parameters

```
virsh migrate --live opensuse11 qemu+ssh://tux@jupiter.example.com/system
```

Persistent live migration; delete VM definition on source

```
virsh migrate --live --persistent --undefinesource 37 \
qemu+tls://tux@jupiter.example.com/system
```

Offline migration using port 49152

```
virsh migrate opensuse11 qemu+ssh://tux@jupiter.example.com/system \
--migrateuri tcp://@jupiter.example.com:49152
```



Note: Transient vs. Persistent Migrations

By default **`virsh migrate`** creates a temporary (transient) copy of the VM Guest on the target host. A shut down version of the original guest description remains on the source host. A transient copy will be deleted from the server once it is shut down.

In order to create a permanent copy of a guest on the target host, use the switch `--persistent`. A shut down version of the original guest description remains on the source host, too. Use the option `--undefinesource` together with `--persistent` for a “real” move where a permanent copy is created on the target host and the version on the source host is deleted.

It is not recommended to use `--undefinesource` without the `--persistent` option, since this will result in the loss of both VM Guest definitions when the guest is shut down on the target host.

11.2 Monitoring

11.2.1 Monitoring with Virtual Machine Manager

After starting Virtual Machine Manager and connecting to the VM Host Server, a CPU usage graph of all the running guests is displayed.

It is also possible to get information about disk and network usage with this tool, however, you must first activate this in the *Preferences*:

1. Run `virt-manager`.
2. Select *Edit > Preferences*.
3. Change the tab from *General* to *Stats*.
4. Activate the check boxes for *Disk I/O* and *Network I/O*.
5. If desired, also change the update interval or the number of samples that are kept in the history.
6. Close the *Preferences* dialog.
7. Activate the graphs that should be displayed under *View > Graph*.

Afterwards, the disk and network statistics are also displayed in the main window of the Virtual Machine Manager.

More precise data is available from the VNC window. Open a VNC window as described in [Section 7.2, “Opening a Graphical Console”](#). Choose *Details* from the toolbar or the *View* menu. The statistics are displayed from the *Performance* entry of the left-hand tree menu.

11.2.2 Monitoring with **kvm_stat**

kvm_stat can be used to trace KVM performance events. It monitors `/sys/kernel/debug/kvm`, so it needs the debugfs to be mounted. On SUSE Linux Enterprise Server it should be mounted by default. In case it is not mounted, use the following command:


```
mount -t debugfs none /sys/kernel/debug
```

kvm_stat can be used in three different modes:

EXAMPLE 11.1: TYPICAL OUTPUT OF **kvm_stat**

```
kvm statistics

efer_reload          0          0
exits                11378946  218130
fpu_reload           62144     152
halt_exits           414866     100
halt_wakeup          260358      50
host_state_reload    539650     249
hypercalls            0          0
insn_emulation       6227331  173067
insn_emulation_fail   0          0
invlpg               227281      47
io_exits              113148      18
irq_exits             168474     127
irq_injections        482804     123
irq_window            51270      18
largepages            0          0
mmio_exits            6925         0
mmu_cache_miss        71820      19
mmu_flooded           35420        9
mmu_pde_zapped        64763      20
mmu_pte_updated        0          0
mmu_pte_write         213782      29
mmu_recycled           0          0
mmu_shadow_zapped     128690      17
mmu_unsync             46         -1
nmi_injections         0          0
nmi_window             0          0
pf_fixed             1553821     857
pf_guest              1018832     562
remote_tlb_flush      174007      37
request_irq            0          0
signal_exits           0          0
tlb_flush             394182     148
```

See <http://clalance.blogspot.com/2009/01/kvm-performance-tools.html>  for further information on how to interpret these values.

III Managing Virtual Machines with QEMU

- 12 QEMU Overview **90**
- 13 Guest Installation **91**
- 14 Running Virtual Machines with qemu-kvm **105**
- 15 KVM Disk Cache Modes **130**
- 16 Administrating Virtual Machines with QEMU Monitor **134**

12 QEMU Overview

QEMU is a fast, cross-platform Open Source machine emulator which can emulate a huge number of hardware architectures for you. QEMU lets you run a complete unmodified operating system (VM Guest) on top of your existing system (VM Host Server).

You can also use QEMU for debugging purposes - you can easily stop your running virtual machine, inspect its state and save and restore it later.

QEMU consists of the following parts:

- processor emulator (x86, s390x, PowerPC, Sparc ...)
- emulated devices (graphic card, network card, hard drives, mice ...)
- generic devices used to connect the emulated devices to the related host devices
- descriptions of the emulated machines (PC, Power Mac ...)
- debugger
- user interface used to interact with the emulator

As a virtualization solution, QEMU can be run together with the KVM kernel module. If the VM Guest hardware architecture is the same as VM Host Server's architecture, QEMU can take advantage of the KVM acceleration.

13 Guest Installation

A virtual machine is comprised of data and operating system files that define the virtual environment. Virtual machines are hosted and controlled by the VM Host Server. This chapter provides generalized instructions for installing virtual machines.

Before creating a virtual machine, consider the following:

- If you want to use an automated installation file (AutoYaST, NetWare® Response File, or RedHat Kickstart), you need to download it to a directory on the host machine server, or make it available through a network file system.
- When creating sparse image files, make sure the partition on which you create them always has sufficient free space. The guest system has no means to check the host's disk space. Having no space left on the host partition causes write errors and loss of data on the guest system.

For further prerequisites, consult the manuals of the respective operating system to install.

13.1 Basic Installation with **qemu-kvm**

The libvirt-based tools such as **virt-manager** or **vm-install** offer convenient interfaces to set up and manage virtual machines. They act as a kind of wrapper for **qemu-kvm**. However, it is also possible to use **qemu-kvm** directly without using libvirt-based tools at all.



Warning

Virtual machines created with **qemu-kvm** are not "visible" for the libvirt-based tools.

In the following example, a virtual machine with the same parameters as in *Example 6.1, "Interactive Setup on the Command Line Using **vm-install**"* will be set up using **qemu-kvm**. For detailed information on the commands, refer to the respective man pages.

If you do not already have an image of a system which you want to run in a virtualized environment, you need to create one from the installation media. In such case, you need to prepare a hard disk image, and obtain an image of the installation media or the media itself.

Create a hard disk with **qemu-img**.


```
qemu-img create① -f raw② /images/sles11/hda③ 8G④
```

- ① The subcommand `create` tells `qemu-img` to create a new image.
- ② Specify the disk's format with the `-f` parameter.
- ③ The full path to the image file.
- ④ The size of the image—8 GB in this case. The image is created as a sparse file that grows when the disk is filled with data. The specified size defines the maximum size to which the image file can grow.

After at least one hard disk image is created, you can set up a virtual machine with `qemu-kvm` that will boot into the installation system:

```
qemu-kvm -name "sles11"① -M pc-0.12② -m 768③ \
-smp 2④ -boot d⑤ \
-drive file=/images/sles11/hda,if=virtio,index=0,media=disk,format=raw⑥ \
-drive file=/isos/SLES-11-SP1-DVD-x86_64-GM-DVD1.iso,index=1,media=cdrom⑦ \
-net nic,model=virtio,macaddr=52:54:00:05:11:11⑧ \
-vga cirrus⑨ -balloon virtio⑩
```

- ① Name of the virtual machine that will be displayed in the window caption and also used for the VNC server. This name must be unique.
- ② Specifies the machine type (*Standard PC*, *ISA-only PC*, or *Intel-Mac*). Use `qemu-kvm -M ?` to display a list of valid parameters. `pc-0.12` is the default *Standard PC*.
- ③ Maximum amount of memory for the virtual machine.
- ④ Defines an SMP system with two processors.
- ⑤ Specifies the boot order. Valid values are `a`, `b` (floppy 1 and 2), `c` (first hard disk), `d` (first CD-ROM), or `n` to `p` (Ether-boot from network adapter 1-3). Defaults to `c`.
- ⑥ Defines the first (`index=0`) hard disk. It will be accessed as a paravirtualized (`if=virtio`) drive in `raw` format.
- ⑦ The second (`index=1`) image drive will act as a CD-ROM.
- ⑧ Defines a paravirtualized (`model=virtio`) network adapter with the MAC address `52:54:00:05:11:11`. Be sure to specify a unique MAC address, otherwise a network conflict may occur.
- ⑨ Specifies the graphic card. If you specify `none`, the graphic card will be disabled.
- ⑩ Defines the paravirtualized balloon device that allows to dynamically change the amount of memory (up to the maximum value specified with the parameter `-m`).

After the installation of the guest operating system finishes, you can easily start the related virtual machine without the need to specify the CD-ROM device:

```
qemu-kvm -name "sles11" -M pc-0.12 -m 768 \  
-smp 2 -boot c \  
-drive file=/images/sles11/hda,if=virtio,index=0,media=disk,format=raw \  
-net nic,model=virtio,macaddr=52:54:00:05:11:11 \  
-vga cirrus -balloon virtio
```

13.2 Managing Disk Images with **qemu-img**

In the previous section (see [Section 13.1, “Basic Installation with **qemu-kvm**”](#)), we used the **qemu-img** command to create an image of a hard disk. You can, however, use **qemu-img** for general disk image manipulation. This section introduces useful **qemu-img** subcommands to help manage the disk images flexibly.

13.2.1 General Information on **qemu-img** Invocation

qemu-img uses subcommands (like **zypper** does) to do specific tasks. Each subcommand understands a different set of options. Some of these options are general and used by more of these subcommands, while some of them are unique to the related subcommand. See the **qemu-img** manual page ([man 1 qemu-img](#)) for a complete list of all supported options. **qemu-img** uses the following general syntax:

```
qemu-img subcommand [options]
```

and supports the following subcommands:

create

Creates a new disk image on the file system.

check

Checks an existing disk image for errors.

convert

Converts an existing disk image to a new one in a different format.

info

Displays information about the relevant disk image.

snapshot

Manages snapshots of existing disk images.

commit

Applies changes made to an existing disk image.

rebase

Creates a new base image based on an existing image.

resize

Increases or decreases the size of an existing image.

13.2.2 Creating, Converting and Checking Disk Images

This section describes how to create disk images, check their condition, convert a disk image from one format to another, and get detailed information about a particular disk image.

13.2.2.1 `qemu-img create`

Use **`qemu-img create`** to create a new disk image for your VM Guest operating system. The command uses the following syntax:

```
qemu-img create -f fmt 1 -o options 2 fname 3 size 4
```

- 1** The format of the target image. To get a complete list of image formats supported by QEMU, run **`qemu-img -h`** and look at the last line of the output.
- 2** Some of the image formats support additional options to be passed on the command line. You can specify them here with the **`-o`** option. The **`raw`** image format supports only the **`size`** option, so it is possible to insert **`-o size=8G`** instead of adding the size option at the end of the command.
- 3** Path to the target disk image to be created.
- 4** Size of the target disk image (if not already specified with the **`-o size=<image_size>`** option. Optional suffixes for the image size are **`K`** (kilobyte), **`M`** (megabyte), **`G`** (gigabyte), or **`T`** (terabyte).

To create a new disk image **`sles11sp1.raw`** in the directory **`/images`** growing up to a maximum size of 4 GB, run the following command:

```
tux@venus:~> qemu-img create -f raw -o size=4G /images/sles11sp1.raw
```

```

Formatting '/images/sles11sp1.raw', fmt=raw size=4294967296

tux@venus:~> ls -l /images/sles11sp1.raw
-rw-r--r-- 1 tux users 4294967296 Nov 15 15:56 /images/sles11sp1.raw

tux@venus:~> qemu-img info /images/sles11sp1.raw
image: /images/sles11sp1.raw
file format: raw
virtual size: 4.0G (4294967296 bytes)
disk size: 0

```

As you can see, the *virtual* size of the newly created image is 4 GB, but the actual reported disk size is 0 as no data has been written to the image yet.

13.2.2.2 `qemu-img convert`

Use **`qemu-img convert`** to convert disk images to another format. To get a complete list of image formats supported by QEMU, run **`qemu-img -h`** and look at the last line of the output. The command uses the following syntax:

```
qemu-img convert -c ❶ -f fmt ❷ -O out_fmt ❸ -o options ❹ fname ❺ out_fname ❻
```

- ❶ Applies compression on the target disk image. Only `qcow` and `qcow2` formats support compression.
- ❷ The format of the source disk image. It is autodetected in most cases and can therefore be omitted.
- ❸ The format of the target disk image.
- ❹ Specify additional options relevant for the target image format. Use `-o ?` to view the list of options supported by the target image format:
- ❺ Path to the source disk image to be converted.
- ❻ Path to the converted target disk image.

```

tux@venus:~> qemu-img convert -O vmdk /images/sles11sp1.raw \
/images/sles11sp1.vmdk

tux@venus:~> ls -l /images/
-rw-r--r-- 1 tux users 4294967296 16. lis 10.50 sles11sp1.raw
-rw-r--r-- 1 tux users 2574450688 16. lis 14.18 sles11sp1.vmdk

```

To see a list of options relevant for the selected target image format, run the following command (replace `vmdk` with your image format):

```
tux@venus:~> qemu-img convert -O vmdk /images/sles11sp1.raw \
/images/sles11sp1.vmdk -o ?
Supported options:
size                Virtual disk size
backing_file        File name of a base image
compat6             VMDK version 6 image
subformat           VMDK flat extent format, can be one of {monolithicSparse \
                    (default) | monolithicFlat | twoGbMaxExtentSparse | twoGbMaxExtentFlat}
scsi                SCSI image
```

13.2.2.3 `qemu-img check`

Use **`qemu-img check`** to check the existing disk image for errors. Not all disk image formats support this feature. The command uses the following syntax:

```
qemu-img check -f fmt❶ fname❷
```

- ❶ The format of the source disk image. It is autodetected in most cases and can therefore be omitted.
- ❷ Path to the source disk image to be checked.

If no error is found, the command returns no output. Otherwise, the type and number of errors found is shown.

```
tux@venus:~> qemu-img check -f qcow2 /images/sles11sp1.qcow2
ERROR: invalid cluster offset=0x2af0000
[...]
ERROR: invalid cluster offset=0x34ab0000
378 errors were found on the image.
```

13.2.2.4 Increasing the Size of an Existing Disk Image

When creating a new image, you must specify its maximum size before the image is created (see [Section 13.2.2.1, “`qemu-img create`”](#)). After you install and run VM Guest for some time, the initial size of the image may no longer be sufficient and you need to add more space to it.

To increase the size of an existing disk image by 2 Gigabytes, use

```
qemu-img resize /images/sles11sp1.raw +2GB
```



Note

You can resize the raw disk images only. To resize other image formats, convert it to raw with qemu-img convert first.

The image now contains an empty space of 2 GB after the final partition. You can resize the existing partitions or add new ones.

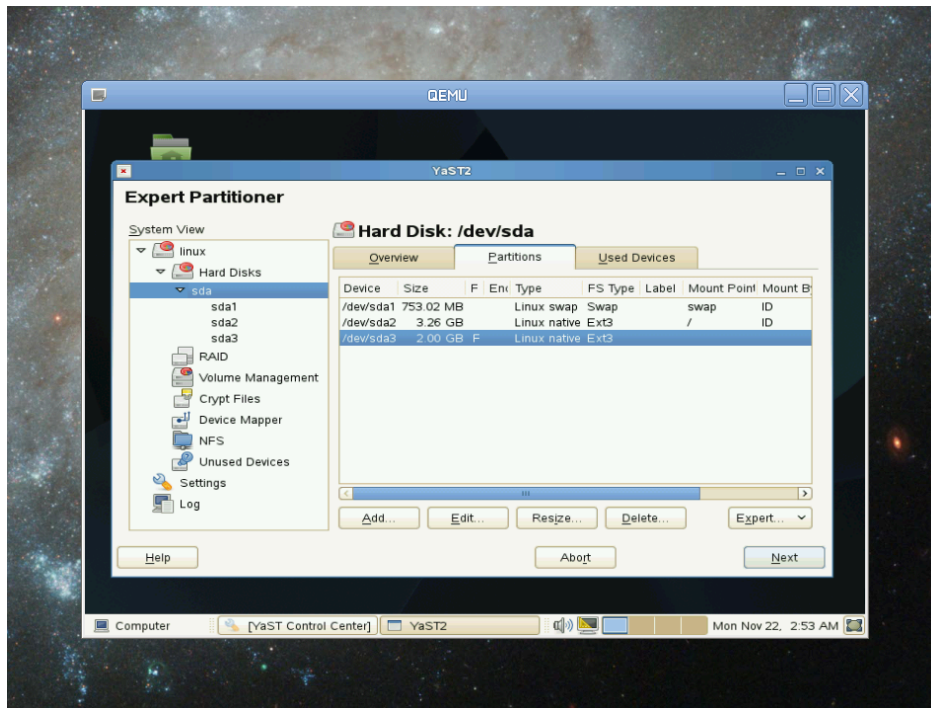


FIGURE 13.1: NEW 2GB PARTITION IN GUEST YAST PARTITIONER

13.2.3 Managing Snapshots of Virtual Machines with qemu-img

Virtual machine snapshots are snapshots of the complete environment in which a VM Guest is running. The snapshot includes the state of the processor (CPU), memory (RAM), devices, and all writable disks.

Snapshots are helpful when you need to save your virtual machine in a particular state. For example, after you configured network services on a virtualized server and want to quickly start the virtual machine in the same state you last saved it. Or you can create a snapshot after the virtual machine has been powered off to create a backup state before you try something experimental and possibly make VM Guest unstable. This section introduces the latter case, while the former is described in *Chapter 16, Adminstrating Virtual Machines with QEMU Monitor*.

To use snapshots, your VM Guest must contain at least one writable hard disk image in `qcow2` format. This device is usually the first virtual hard disk.

Virtual machine snapshots are created with the `savevm` command in the interactive QEMU monitor. You can assign a 'tag' to each snapshot which makes its identification easier. For more information on QEMU monitor, see [Chapter 16, Administrating Virtual Machines with QEMU Monitor](#).

Once your `qcow2` disk image contains saved snapshots, you can inspect them with the `qemu-img snapshot` command.



Warning

Do not create or delete virtual machine snapshots with the `qemu-img snapshot` command while the virtual machine is running. Otherwise, you can damage the disk image with the state of the virtual machine saved.

13.2.3.1 Listing Existing Snapshots

Use `qemu-img snapshot -l disk_image` to view a list of all existing snapshots saved in the `disk_image` image. You can get the list even while the VM Guest is running.

```
tux@venus:~> qemu-img snapshot -l /images/sles11sp1.qcow2
Snapshot list:
ID ①      TAG ②      VM SIZE ③      DATE ④      VM CLOCK ⑤
1      booting      4.4M 2010-11-22 10:51:10  00:00:20.476
2      booted      184M 2010-11-22 10:53:03  00:02:05.394
3      logged_in    273M 2010-11-22 11:00:25  00:04:34.843
4      ff_and_term_running 372M 2010-11-22 11:12:27  00:08:44.965
```

- ① Unique identification number of the snapshot. Usually auto-incremented.
- ② Unique description string of the snapshot. It is meant as a human readable version of the ID.
- ③ The disk space occupied by the snapshot. Note that the more memory is consumed by running applications, the bigger the snapshot is.
- ④ Time and date the snapshot was created.
- ⑤ The current state of the virtual machine's clock.

13.2.3.2 Creating Snapshots of a Powered-Off Virtual Machine

Use **qemu-img snapshot -c *snapshot_title* *disk_image*** to create a snapshot of the current state of a virtual machine which was previously powered off.

```
tux@venus:~> qemu-img snapshot -c backup_snapshot /images/sles11sp1.qcow2
```

```
tux@venus:~> qemu-img snapshot -l /images/sles11sp1.qcow2
```

Snapshot list:

ID	TAG	VM SIZE	DATE	VM CLOCK
1	booting	4.4M	2010-11-22 10:51:10	00:00:20.476
2	booted	184M	2010-11-22 10:53:03	00:02:05.394
3	logged_in	273M	2010-11-22 11:00:25	00:04:34.843
4	ff_and_term_running	372M	2010-11-22 11:12:27	00:08:44.965
5	backup_snapshot	0	2010-11-22 14:14:00	00:00:00.000

Once something breaks in your VM Guest and you need to restore the state of the saved snapshot (id 5 in our example), power off your VM Guest and do the following:

```
tux@venus:~> qemu-img snapshot -a 5 /images/sles11sp1.qcow2
```

Next time you run the virtual machine with **qemu-kvm**, it will be in the state of snapshot number 5.



Note

The **qemu-img snapshot -c** command is not related to the **savevm** command of QEMU monitor (see [Chapter 16, Administrating Virtual Machines with QEMU Monitor](#)). For example, you cannot apply a snapshot with **qemu-img snapshot -a** on a snapshot created with **savevm** in QEMU's monitor.

13.2.3.3 Deleting Snapshots

Use **qemu-img snapshot -d *snapshot_id* *disk_image*** to delete old or unneeded snapshots of a virtual machine. This saves some disk space inside the **qcow2** disk image as the space occupied by the snapshot data is restored:

```
tux@venus:~> qemu-img snapshot -d 2 /images/sles11sp1.qcow2
```


13.2.4 Manipulate Disk Images Effectively

Imagine the following real-life situation: you are a server administrator who runs and manages a number of virtualized operating systems. One group of these systems are based on one specific distribution, while another group (or groups) is based on different versions of the distribution or even on a different (and maybe non-Unix) platform. And to make the case even more complex, individual virtual guest systems based on the same distribution usually differ according to the department and deployment: a file server typically uses different setup and services than a Web server does, while both may still be based on SUSE® Linux Enterprise Server 11 SP1.

With QEMU it is possible to create “base” disk images. You can use them as template virtual machines. These base images will save you plenty of time because you will never need to install the same operating system more than once.

13.2.4.1 Base and Derived Images

First, build a disk image as usual and install the target system on it. For more information, see [Section 13.1, “Basic Installation with qemu-kvm”](#) and [Section 13.2.2, “Creating, Converting and Checking Disk Images”](#). Then build a new image while using the first one as a base image. The base image is also called a 'backing' file. After your new 'derived' image is built, never boot the base image again, but boot the derived image instead. Several derived images may depend on one base image at the same time. Therefore, changing the base image can damage the dependencies. While using your derived image, QEMU writes changes to it and uses the base image only for reading.

It is a good practice to create a base image from a freshly installed (and, if needed, registered) operating system with no patches applied and no additional applications installed or removed. Later on, you can create another base image with the latest patches applied and based on the original base image.

13.2.4.2 Creating Derived Images



Note

While you can use the `raw` format for base images, you cannot use it for derived images because the `raw` format does not support the `backing_file` option. Use for example the `qcow2` format for the derived images.

For example, `/images/sles11sp1_base.raw` is the base image holding a freshly installed system.

```
tux@venus:~> qemu-img info /images/sles11sp1_base.raw
image: /images/sles11sp1_base.raw
file format: raw
virtual size: 4.0G (4294967296 bytes)
disk size: 2.4G
```

The image's reserved size is 4 GB, the actual size is 2.4 GB, and its format is `raw`. Create an image derived from the `/images/sles11sp1_base.raw` base image with:

```
tux@venus:~> qemu-img create -f qcow2 /images/sles11sp1_derived.qcow2 \
-o backing_file=/images/sles11sp1_base.raw
Formatting '/images/sles11sp1_derived.qcow2', fmt=qcow2 size=4294967296 \
backing_file='/images/sles11sp1_base.raw' encryption=off cluster_size=0
```

Look at the derived image details:

```
tux@venus:~> qemu-img info /images/sles11sp1_derived.qcow2
image: /images/sles11sp1_derived.qcow2
file format: qcow2
virtual size: 4.0G (4294967296 bytes)
disk size: 140K
cluster_size: 65536
backing file: /images/sles11sp1_base.raw \
(actual path: /images/sles11sp1_base.raw)
```

Although the reserved size of the derived image is the same as the size of the base image (4 GB), the actual size is 140 KB only. The reason is that only changes made to the system inside the derived image are saved. Run the derived virtual machine, register it, if needed, and apply the latest patches. Do any other changes in the system such as removing unneeded or installing new software packages. Then shut the VM Guest down and examine its details once more:

```
tux@venus:~> qemu-img info /images/sles11sp1_derived.qcow2
image: /images/sles11sp1_derived.qcow2
file format: qcow2
virtual size: 4.0G (4294967296 bytes)
disk size: 1.1G
cluster_size: 65536
backing file: /images/sles11sp1_base.raw \
(actual path: /images/sles11sp1_base.raw)
```

The `disk size` value has grown to 1.1 GB, which is the disk space occupied by the changes on the file system compared to the base image.

13.2.4.3 Rebasing Derived Images

Once you modify the derived image (apply patches, install specific applications, or change the environment settings etc.) into a satisfactory shape, at some point you probably want to create a new base image 'merged' from the base image and the derived one. Your first base image (`/images/sles11sp1_base.raw`) holds a freshly installed system and can be a template for new modified base images, while the new one can contain the same system as the first one plus all security and update patches applied, for example. After you created this new base image, you can use it as a template for more specialized derived images as well. The new base image becomes independent of the original one. The process of creating base images from derived ones is called 'rebasing':

```
tux@venus:~> qemu-img convert /images/sles11sp1_derived.qcow2 \
-O raw /images/sles11sp1_base2.raw
```

This command created the new base image `/images/sles11sp1_base2.raw` using the `raw` format.

```
tux@venus:~> qemu-img info /images/sles11sp1_base2.raw
image: /images/sles11sp1_base2.raw
file format: raw
virtual size: 4.0G (4294967296 bytes)
disk size: 2.8G
```

The new image is 0.4 gigabytes bigger than the original base image. It uses no backing file, and you can easily create new derived images based upon it. This lets you create a sophisticated hierarchy of virtual disk images for your organization, saving a lot of time and work.

13.2.4.4 Mounting an Image on a VM Host Server

Sometimes it is useful to mount a virtual disk image under the host system. For example, if VM Host Server does not have a network support, this can be the only way to transfer files in and out of a VM Guest.

Linux systems can mount an internal partition of a `raw` disk image using a 'loopback' device. The first example procedure is more complex but more illustrative, while the second one is straightforward:

PROCEDURE 13.1: MOUNTING DISK IMAGE BY CALCULATING PARTITION OFFSET

1. Set a `loop` device on the disk image whose partition you want to mount.

```
tux@venus:~> losetup /dev/loop0 /images/sles11sp1_base.raw
```

2. Find the *sector size* and the starting *sector number* of the partition you want to mount.

```
tux@venus:~> fdisk -lu /dev/loop0
```

```
Disk /dev/loop0: 4294 MB, 4294967296 bytes
255 heads, 63 sectors/track, 522 cylinders, total 8388608 sectors
Units = sectors of 1 * 512 = 512 ❶ bytes
Disk identifier: 0x000ceca8
```

Device	Boot	Start	End	Blocks	Id	System
/dev/loop0p1		63	1542239	771088+	82	Linux swap
/dev/loop0p2	*	1542240 ❷	8385929	3421845	83	Linux

- ❶ The disk sector size.
- ❷ The starting sector of the partition.

3. Calculate the partition start offset:

```
sector_size * sector_start = 512 * 1542240 = 789626880
```

4. Delete the loop and mount the partition inside the disk image with the calculated offset on a prepared directory.

```
tux@venus:~> losetup -d /dev/loop0
tux@venus:~> mount -o loop,offset=789626880 \
/images/sles11sp1_base.raw /mnt/sles11sp1/
tux@venus:~> ls -l /mnt/sles11sp1/
total 112
drwxr-xr-x  2 root root  4096 Nov 16 10:02 bin
drwxr-xr-x  3 root root  4096 Nov 16 10:27 boot
drwxr-xr-x  5 root root  4096 Nov 16 09:11 dev
[...]
drwxrwxrwt 14 root root  4096 Nov 24 09:50 tmp
drwxr-xr-x 12 root root  4096 Nov 16 09:16 usr
drwxr-xr-x 15 root root  4096 Nov 16 09:22 var
```

5. Copy one or more files onto the mounted partition and unmount it when finished.

```
tux@venus:~> cp /etc/X11/xorg.conf /mnt/sles11sp1/root/tmp
tux@venus:~> ls -l /mnt/sles11sp1/root/tmp
tux@venus:~> umount /mnt/sles11sp1/
```

PROCEDURE 13.2: MOUNTING DISK IMAGE WHILE UTILIZING **kpartx**

1. Set a *loop* device on the disk image whose partition you want to mount.

```
tux@venus:~> losetup /dev/loop0 /images/sles11sp1_base.raw
```

2. Create a device map from the disk image's partitions.

```
tux@venus:~> kpartx -a /dev/loop0
```

3. Mount any partition of the disk image on a prepared mount point.

```
tux@venus:~> mount /dev/mapper/loop0p1 /mnt/p1
```

You can replace `loop0p1` with the number of the partition you want to mount, for example `loop0p3` to mount the third partition on the disk image.

4. Copy or move files or directories to and from the mounted partition as you like. Once you finish, unmount the partition and delete the loop.

```
tux@venus:~> umount /mnt/p1
```

```
tux@venus:~> losetup -d /dev/loop0
```



Warning

Never mount a partition of an image of a running virtual machine in a read-write mode. This could corrupt the partition and break the whole VM Guest.

14 Running Virtual Machines with qemu-kvm

Once you have a virtual disk image ready (for more information on disk images, see [Section 13.2](#), “*Managing Disk Images with `qemu-img`*”), it is time to start the related virtual machine. [Section 13.1](#), “*Basic Installation with `qemu-kvm`*” introduced simple commands to install and run a VM Guest. This chapter focuses on a more detailed explanation of `qemu-kvm` usage, and shows solutions of more specific tasks. For a complete list of `qemu-kvm`'s options, see its manual page (`man 1 qemu-kvm`).

14.1 Basic `qemu-kvm` Invocation

The `qemu-kvm` command uses the following syntax:

```
qemu-kvm options ❶ disk_img ❷
```

- ❶ `qemu-kvm` understands a large number of options. Most of them define parameters of the emulated hardware, while others affect more general emulator behavior. If you do not supply any options, default values are used, and you need to supply the path to a disk image to be run.
- ❷ Path to the disk image holding the guest system you want to virtualize. `qemu-kvm` supports a large number of image formats. Use `qemu-img --help` to list them. If you do not supply the path to a disk image as a separate argument, you have to use the `-drive file=` option.

14.2 General `qemu-kvm` Options

This section introduces general `qemu-kvm` options and options related to the basic emulated hardware, such as virtual machine's processor, memory, model type, or time processing methods.

`-name name_of_guest`

Specifies the name of the running guest system. The name is displayed in the window caption and also used for the VNC server.

`-boot options`

Specifies the order in which the defined drives will be booted. Drives are represented by letters, where 'a' and 'b' stands for the floppy drives 1 and 2, 'c' stands for the first hard disk, 'd' stands for the first CD-ROM drive, and 'n' to 'p' stand for Ether-boot network adapters.

For example, `qemu-kvm [...] -boot order=ndc` first tries to boot from network, then from the first CD-ROM drive, and finally from the first hard disk.

-pidfile *fname*

Stores the QEMU's process identification number (PID) in a file. This is useful if you run QEMU from a script.

-nodefaults

By default QEMU creates basic virtual devices even if you do not specify them on the command line. This option turns this feature off, and you must specify every single device manually, including graphical and network cards, parallel or serial ports, or virtual consoles. Even QEMU monitor is not attached by default.

-daemonize

'Daemonizes' the QEMU process after it is started. QEMU will detach from the standard input and standard output after it is ready to receive connections on any of its devices.

14.2.1 Basic Virtual Hardware

-M *machine_type*

Specifies the type of the emulated machine. Run `qemu-kvm -M help` to view a list of supported machine types.

```
tux@venus:~> qemu-kvm -M help
Supported machines are:
q35                Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-1.4)
pc-q35-1.4         Standard PC (Q35 + ICH9, 2009)
pc                 Standard PC (i440FX + PIIX, 1996)
pc-i440fx-1.4      Standard PC (i440FX + PIIX, 1996) (default)
pc-1.3             Standard PC
pc-1.2             Standard PC
pc-1.1             Standard PC
pc-1.0             Standard PC
pc-0.15            Standard PC
pc-0.14            Standard PC
pc-0.13            Standard PC
pc-0.12            Standard PC
pc-0.11            Standard PC, qemu 0.11
pc-0.10            Standard PC, qemu 0.10
isapc              ISA-only PC
none               empty machine
```

-m megabytes

Specifies how many megabytes are used for the virtual RAM size. Default is 512 MB.

-balloon virtio

Specifies a paravirtualized device to dynamically change the amount of virtual RAM memory assigned to VM Guest. The top limit is the amount of memory specified with -m.

-cpu cpu_model

Specifies the type of the processor (CPU) model. Run qemu-kvm -cpu ? to view a list of supported CPU models.

```
tux@venus:~> qemu-kvm -cpu help
x86      qemu64  QEMU Virtual CPU version 1.4.0
x86      phenom  AMD Phenom(tm) 9550 Quad-Core Processor
x86      core2duo Intel(R) Core(TM)2 Duo CPU    T7700  @ 2.40GHz
x86      kvm64   Common KVM processor
x86      qemu32  QEMU Virtual CPU version 1.4.0
x86      kvm32   Common 32-bit KVM processor
x86      coreduo Genuine Intel(R) CPU          T2600  @ 2.16GHz
x86      486
x86      pentium
x86      pentium2
x86      pentium3
x86      athlon  QEMU Virtual CPU version 1.4.0
x86      n270    Intel(R) Atom(TM) CPU N270    @ 1.60GHz
x86      Conroe  Intel Celeron_4x0 (Conroe/Merom Class Core 2)
x86      Penryn  Intel Core 2 Duo P9xxx (Penryn Class Core 2)
x86      Nehalem Intel Core i7 9xx (Nehalem Class Core i7)
x86      Westmere Westmere E56xx/L56xx/X56xx (Nehalem-C)
x86      SandyBridge Intel Xeon E312xx (Sandy Bridge)
x86      Haswell  Intel Core Processor (Haswell)
x86      Opteron_G1 AMD Opteron 240 (Gen 1 Class Opteron)
x86      Opteron_G2 AMD Opteron 22xx (Gen 2 Class Opteron)
x86      Opteron_G3 AMD Opteron 23xx (Gen 3 Class Opteron)
x86      Opteron_G4 AMD Opteron 62xx class CPU
x86      Opteron_G5 AMD Opteron 63xx class CPU
```

-smp number_of_cpus

Specifies how many CPUs will be emulated. QEMU supports up to 255 CPUs on the PC platform (up to 64 with KVM acceleration used). This option also takes other CPU-related parameters, such as number of *sockets*, number of *cores* per socket, or number of *threads* per core.

Following is an example of a working **qemu-kvm** command line:

```
qemu-kvm -name "SLES 11 SP1" -M pc-0.12 -m 512 -cpu kvm64 \
-smp 2 /images/sles11sp1.raw
```

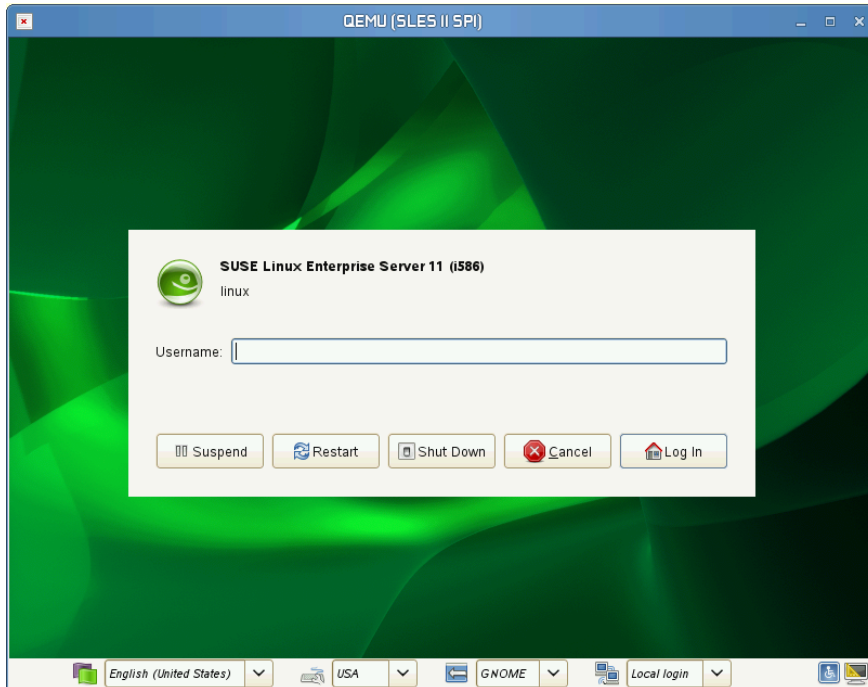


FIGURE 14.1: QEMU WINDOW WITH SLES 11 SP1 AS VM GUEST

-no-acpi

Disables ACPI support. Try to use it if VM Guest reports problems with ACPI interface.

-S

QEMU starts with CPU stopped. To start CPU, enter c in QEMU monitor. For more information, see *Chapter 16, Adminstrating Virtual Machines with QEMU Monitor*.

14.2.2 Storing and Reading Configuration of Virtual Devices

-readconfig *cfg_file*

Instead of entering the devices configuration options on the command line each time you want to run VM Guest, **qemu-kvm** can read it from a file which was either previously saved with -writeconfig or edited manually.

-writeconfig *cfg_file*

Dumps the current virtual machine devices configuration to a text file. It can be consequently re-used with the -readconfig option.

```
tux@venus:~> qemu-kvm -name "SLES 11 SP1" -M pc-0.12 -m 512 -cpu kvm64 \
-smp 2 /images/sles11sp1.raw -writeconfig /images/sles11sp1.cfg
(exited)
tux@venus:~> more /images/sles11sp1.cfg
# qemu config file

[drive]
  index = "0"
  media = "disk"
  file = "/images/sles11sp1_base.raw"
```

This way you can effectively manage the configuration of your virtual machines' devices in a well-arranged way.

14.2.3 Guest Real-time Clock

-rtc *options*

Specifies the way the RTC is handled inside a VM Guest. By default, the clock of the guest is derived from that of the host system. Therefore, it is recommended that the host system clock is synchronized with an accurate external clock (for example, via NTP service).

If you need to isolate the VM Guest clock from the host one, specify clock=vm instead of the default clock=host.

You can also specify a 'starting point' for VM Guest clock with the base option:

```
qemu-kvm [...] -rtc clock=vm,base=2010-12-03T01:02:00
```

Instead of a timestamp, you can specify utc or localtime. The former instructs VM Guest to start at the current UTC value (Coordinated Universal Time, see <http://en.wikipedia.org/wiki/UTC>), while the latter applies the local time setting.

14.3 Using Devices in QEMU

QEMU virtual machines emulate all devices needed to run a VM Guest. QEMU supports, for example, several types of network cards, block devices (hard and removable drives), character devices (serial and parallel ports). For satisfactory operation and performance of the virtual machine, some or all of these devices must be configured correctly. This section introduces options to configure various types of supported devices.



Tip

If your device, such as `-drive`, needs a special driver and driver properties to be set, specify them with the `-device` option, and identify with `drive=` suboption. For example

```
qemu [...] -drive if=none,id=drive0,format=raw \  
-device virtio-blk-pci,drive=drive0,scsi=off ...
```

To get help on available drivers and their properties, use `-device ?` and `-device driver, ?`.

14.3.1 Block Devices

Block devices are vital for virtual machines. In general, these are fixed or removable storage media usually referred to as 'drives'. One of the connected hard drives typically holds the guest operating system to be virtualized.

Virtual machine drives are defined with `-drive`. This option uses many suboptions, some of which are described in this section. For their complete list, see the manual page (`man 1 qemu-kvm`).

SUB-OPTIONS FOR THE `-drive` OPTION

`file=image_fname`

Specifies the path to the disk image which will be used with this drive. If not specified, an empty (removable) drive is assumed.

`if=drive_interface`

Specifies the type of interface to which the drive is connected. Currently only `floppy`, `ide`, or `virtio` are supported by SUSE. `virtio` defines a paravirtualized disk driver. Default is `ide`.

index=index_of_connector

Specifies the index number of a connector on the disk interface (see the if option) where the drive is connected. If not specified, the index is automatically incremented.

media=type

Specifies the type of the media. Can be disk for hard disks, or cdrom for removable CD-ROM drives.

format=img_fmt

Specifies the format of the connected disk image. If not specified, the format is autodetected. Currently, SUSE supports qcow2, qed and raw formats for read and write access. The vmrk, vpc and vhd/vhdv formats are only supported in read-only mode.

cache=method

Specifies the caching method for the drive. Possible values are unsafe, writethrough, writeback, directsync, or none. For the qcow2 image format, choose writeback if you care about performance. none disables the host page cache and, therefore, is the safest option. Default for image files is writeback.



Tip

To simplify defining of block devices, QEMU understands several shortcuts which you may find handy when entering the qemu-kvm command line.

You can use

```
qemu-kvm -cdrom /images/cdrom.iso
```

instead of

```
qemu-kvm -drive file=/images/cdrom.iso,index=2,media=cdrom
```

and

```
qemu-kvm -hda /images/image1.raw -hdb /images/image2.raw -hdc \  
/images/image3.raw -hdd /images/image4.raw
```

instead of

```
qemu-kvm -drive file=/images/image1.raw,index=0,media=disk \  
-drive file=/images/image2.raw,index=1,media=disk \  
-drive file=/images/image3.raw,index=2,media=disk \  

```

```
-drive file=/images/image4.raw,index=3,media=disk
```



Tip: Using Host Drives Instead of Images

Normally you will use disk images (see [Section 13.2, “Managing Disk Images with `qemu-img`”](#)) as disk drives of the virtual machine. However, you can also use existing VM Host Server disks, connect them as drives, and access them from VM Guest. Use the host disk device directly instead of disk image filenames.

To access the host CD-ROM drive, use

```
qemu-kvm [...] -drive file=/dev/cdrom,media=cdrom
```

To access the host hard disk, use

```
qemu-kvm [...] -drive file=/dev/hdb,media=disk
```

When accessing the host hard drive from VM Guest, always make sure the access is *read-only*. You can do so by modifying the host device permissions.

14.3.1.1 virtio-blk-data-plane

The *virtio-blk-data-plane* is a new performance feature for KVM. It enables a high-performance code path for I/O requests coming from VM Guests. More specifically, this feature introduces dedicated threads (one per virtual block device) to process I/O requests going through the *virtio-blk* driver. It makes use of Linux AIO (asynchronous I/O interface) support in the VM Host Server Kernel directly—without the need to go through the QEMU block layer. Therefore it can sustain very high I/O rates on storage setups.

The *virtio-blk-data-plane* feature can be enabled or disabled by the `x-data-plane=on|off` option on the `qemu` command line when starting the VM Guest:

```
qemu [...] -drive if=none,id=drive0,cache=none,aio=native,\  
format=raw,file=filename -device virtio-blk-pci,drive=drive0,scsi=off,\  
config-wce=off,x-data-plane=on [...]
```

As of now, the *virtio-blk-data-plane* has the following limitations:

- Only raw image format is supported.
- No support for live migration.

- Block jobs and hot unplug operations fail with -EBUSY.
- I/O throttling limits are ignored.
- Only Linux VM Host Servers are supported because of the Linux AIO usage, but non-Linux VM Guests are supported.



Important

The virtio-blk-data-plane feature is not yet supported in SUSE Linux Enterprise Server. It is released as a technical preview only.

14.3.2 Graphic Devices and Display Options

This section describes QEMU options affecting the type of the emulated video card and the way VM Guest graphical output is displayed.

14.3.2.1 Defining Video Cards

QEMU uses `-vga` to define a video card used to display VM Guest graphical output. The `-vga` option understands the following values:

none

Disables video cards on VM Guest (no video card is emulated). You can still access the running VM Guest via the QEMU monitor and the serial console.

std

Emulates a standard VESA 2.0 VBE video card. Use it if you intend to use high display resolution on VM Guest.

cirrus

Emulates Cirrus Logic GD5446 video card. Good choice if you insist on high compatibility of the emulated video hardware. Most operating systems (even Windows 95) recognize this type of card.



Tip

For best video performance with the `cirrus` type, use 16-bit color depth both on VM Guest and VM Host Server.

14.3.2.2 Display Options

The following options affect the way VM Guest graphical output is displayed.

-nographic

Disables QEMU's graphical output. The emulated serial port is redirected to the console. After starting the virtual machine with -nographic, press **Ctrl - A H** in the virtual console to view the list of other useful shortcuts, for example, to toggle between the console and the QEMU monitor.

```
tux@venus:~> qemu-kvm -hda /images/sles11sp1_base.raw -nographic

C-a h    print this help
C-a x    exit emulator
C-a s    save disk data back to file (if -snapshot)
C-a t    toggle console timestamps
C-a b    send break (magic sysrq)
C-a c    switch between console and monitor
C-a C-a  sends C-a
(pressed C-a c)

QEMU 0.12.5 monitor - type 'help' for more information
(qemu)
```

-no-frame

Disables decorations for the QEMU window. Convenient for dedicated desktop workspace.

-full-screen

Starts QEMU graphical output in full screen mode.

-no-quit

Disables the 'close' button of QEMU window and prevents it from being closed by force.

-alt-grab, -ctrl-grab

By default QEMU window releases the 'captured' mouse after **Ctrl - Alt** is pressed. You can change the key combination to either **Ctrl - Alt - Shift** (-alt-grab), or **Right Ctrl** (-ctrl-grab).

14.3.3 Character Devices

Use -chardev to create a new character device. The option uses the following general syntax:

```
qemu-kvm [...] -chardev backend_type,id=id_string
```

where backend_type can be one of null, socket, udp, msmouse, vc, file, pipe, console, serial, pty, stdio, braille, tty, or parport. All character devices must have a unique identification string up to 127 characters long. It is used to identify the device in other related directives. For the complete description of all back-end's suboptions, see the manual page ([man 1 qemu-kvm](#)). A brief description of the available back-ends follows:

null

Creates an empty device which outputs no data and drops any data it receives.

stdio

Connects to QEMU's process standard input and standard output.

socket

Creates a two-way stream socket. If path is specified, a Unix socket is created:

```
qemu-kvm [...] -chardev \  
socket,id=unix_socket1,path=/tmp/unix_socket1,server
```

The server suboption specifies that the socket is a listening socket.

If port is specified, a TCP socket is created:

```
qemu-kvm [...] -chardev \  
socket,id=tcp_socket1,host=localhost,port=7777,server,nowait
```

The command creates a local listening (server) TCP socket on port 7777. QEMU will not block waiting for a client to connect to the listening port (nowait).

udp

Sends all network traffic from VM Guest to a remote host over the UDP protocol.

```
qemu-kvm [...] -chardev udp,id=udp_fwd,host=mercury.example.com,port=7777
```

The command binds port 7777 on the remote host mercury.example.com and sends VM Guest network traffic there.

vc

Creates a new QEMU text console. You can optionally specify the dimensions of the virtual console:

```
qemu-kvm [...] -chardev vc,id=vc1,width=640,height=480 -mon chardev=vc1
```

The command creates a new virtual console called vc1 of the specified size, and connects the QEMU monitor to it.

file

Logs all traffic from VM Guest to a file on VM Host Server. The path is required and will be created if it does not exist.

```
qemu-kvm [...] -chardev file,id=qemu_log1,path=/var/log/qemu/guest1.log
```

By default QEMU creates a set of character devices for serial and parallel ports, and a special console for QEMU monitor. You can, however, create your own character devices and use them for just mentioned purposes. The following options will help you:

-serial *char_dev*

Redirects the VM Guest's virtual serial port to a character device *char_dev* on VM Host Server. By default, it is a virtual console (vc) in graphical mode, and stdio in non-graphical mode. The -serial understands many suboptions. See the manual page man 1 qemu-kvm for their complete list.

You can emulate up to 4 serial ports. Use -serial none to disable all serial ports.

-parallel *device*

Redirects the VM Guest's parallel port to a *device*. This option supports the same devices as -serial.



Tip

With SUSE Linux Enterprise Server as a VM Host Server, you can directly use the hardware parallel port devices /dev/parportN where N is the number of the port.

You can emulate up to 3 parallel ports. Use -parallel none to disable all parallel ports.

-monitor *char_dev*

Redirects the QEMU monitor to a character device *char_dev* on VM Host Server. This option supports the same devices as -serial. By default, it is a virtual console (vc) in a graphical mode, and stdio in non-graphical mode.

For a complete list of available character devices back-ends, see the man page (man 1 qemu-kvm).

14.4 Networking in QEMU

Use the `-net` option to define a network interface and a specific type of networking for your VM Guest. Currently, SUSE supports the following options: `none`, `nic`, `user`, `bridge`, and `tap`. For a complete list of `-net` suboptions, see the manual page (`man 1 qemu-kvm`).

SUPPORTED `-net` SUBOPTIONS

`none`

Disables a network card emulation on VM Guest. Only the loopback `lo` network interface is available.

`bridge`

Uses a specified network helper to configure the TAP interface and attach it to a specified bridge. For more information, see [Section 14.4.3, “Bridged Networking”](#).

`nic`

Creates a new Network Interface Card (NIC) and connects it to a specified Virtual Local Area Network (VLAN). For more information, see [Section 14.4.1, “Defining a Network Interface Card”](#).

`user`

Specifies a user-mode networking. For more information, see [Section 14.4.2, “User-mode Networking”](#).

`tap`

Specifies a bridged or routed networking. For more information, see [Section 14.4.3, “Bridged Networking”](#).

14.4.1 Defining a Network Interface Card

Use `-net nic` to add a new emulated network card:

```
qemu-kvm [...] -net nic,vlan=1❶,macaddr=00:16:35:AF:94:4B❷,\  
model=virtio❸,name=ncard1❹
```

- ❶ Connects the network interface to VLAN number 1. You can specify your own number, it is mainly useful for identification purpose. If you omit this suboption, QEMU uses the default 0.

- ❷ Specifies the Media Access Control (MAC) address for the network card. It is a unique identifier and you are advised to always specify it. If not, QEMU supplies its own default MAC address and creates a possible MAC address conflict within the related VLAN.
- ❸ Specifies the model of the network card. Use `-net nic,model=?` to get the list of all network card models supported by QEMU on your platform:
Currently, SUSE supports the models `rtl8139` and `virtio`.

14.4.2 User-mode Networking

The `-net user` option instructs QEMU to use a user-mode networking. This is the default if no networking mode is selected. Therefore, these command lines are equivalent:

```
qemu-kvm -hda /images/sles11sp1_base.raw
```

```
qemu-kvm -hda /images/sles11sp1_base.raw -net nic -net user
```

This mode is useful if you want to allow the VM Guest to access the external network resources, such as Internet. By default, no incoming traffic is permitted and therefore, the VM Guest is not visible to other machines on the network. No administrator privileges are required in this networking mode. The user-mode is also useful to do a 'network-booting' on your VM Guest from a local directory on VM Host Server.

The VM Guest allocates an IP address from a virtual DHCP server. VM Host Server (the DHCP server) is reachable at 10.0.2.2, while the IP address range for allocation starts from 10.0.2.15. You can use `ssh` to connect to VM Host Server at 10.0.2.2, and `scp` to copy files back and forth.

14.4.2.1 Command Line Examples

This section shows several examples on how to set up user-mode networking with QEMU.

EXAMPLE 14.1: RESTRICTED USER-MODE NETWORKING

```
qemu-kvm [...] -net user❶,vlan=1❷,name=user_net1❸,restrict=yes❹
```

- ❶ Specifies user-mode networking.
- ❷ Connect to VLAN number 1. If omitted, defaults to 0.
- ❸ Specifies a human readable name of the network stack. Useful when identifying it in the QEMU monitor.

- ④ Isolates VM Guest. It will not be able to communicate with VM Host Server and no network packets will be routed to the external network.

EXAMPLE 14.2: USER-MODE NETWORKING WITH CUSTOM IP RANGE

```
qemu-kvm [...] -net user,net=10.2.0.0/8①,host=10.2.0.6②,dhcpstart=10.2.0.20③,\
hostname=tux_kvm_guest④
```

- ① Specifies the IP address of the network that VM Guest sees and optionally the netmask. Default is 10.0.2.0/8.
- ② Specifies the VM Host Server IP address that VM Guest sees. Default is 10.0.2.2.
- ③ Specifies the first of the 16 IP addresses that the built-in DHCP server can assign to VM Guest. Default is 10.0.2.15.
- ④ Specifies the hostname that the built-in DHCP server will assign to VM Guest.

EXAMPLE 14.3: USER-MODE NETWORKING WITH NETWORK-BOOT AND TFTP

```
qemu-kvm [...] -net user,tftp=/images/tftp_dir①,bootfile=/images/boot/pxelinux.0②
```

- ① Activates a built-in TFTP (a file transfer protocol with the functionality of a very basic FTP) server. The files in the specified directory will be visible to a VM Guest as the root of a TFTP server.
- ② Broadcasts the specified file as a BOOTP (a network protocol which offers an IP address and a network location of a boot image, often used in diskless workstations) file. When used together with `tftp`, the VM Guest can boot from network from the local directory on the host.

EXAMPLE 14.4: USER-MODE NETWORKING WITH HOST PORT FORWARDING

```
qemu-kvm [...] -net user,hostfwd=tcp::2222-:22
```

Forwards incoming TCP connections to the port 2222 on the host to the port 22 (SSH) on VM Guest. If sshd is running on VM Guest, enter

```
ssh qemu_host -p 2222
```

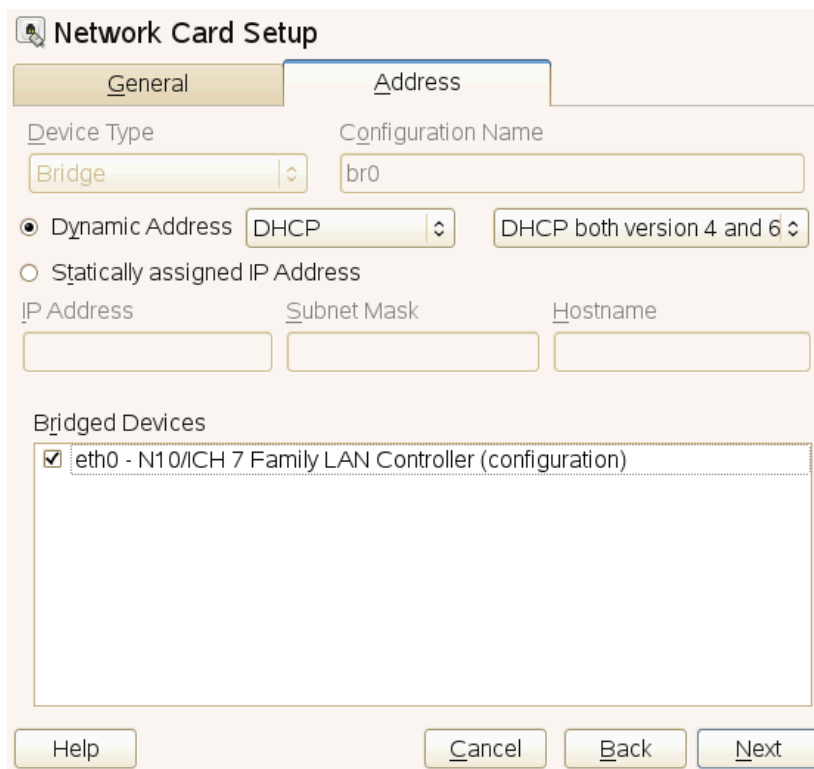
where qemu_host is the hostname or IP address of the host system, to get a SSH prompt from VM Guest.

14.4.3 Bridged Networking

With the `-net tap` option, QEMU creates a network bridge by connecting the host TAP network device to a specified VLAN of VM Guest. Its network interface is then visible to the rest of the network. This method does not work by default and has to be explicitly specified.

First, create a network bridge and add a VM Host Server physical network interface (usually `eth0`) to it:

1. Start YaST Control Center and select *Network Devices > Network Settings*.
2. Click *Add* and select *Bridge* from the *Device Type* drop-down list in the *Hardware Dialog* window. Click *Next*.
3. Choose whether you need a dynamically or statically assigned IP address, and fill the related network settings if applicable.
4. In the *Bridged Devices* pane, select the Ethernet device to add to the bridge.



The screenshot shows the 'Network Card Setup' dialog box with the 'Address' tab selected. The 'Device Type' is set to 'Bridge' and the 'Configuration Name' is 'br0'. Under 'Dynamic Address', 'DHCP' is selected, and 'DHCP both version 4 and 6' is also selected. The 'Statically assigned IP Address' option is unselected. The 'Bridged Devices' list contains one entry: 'eth0 - N10/ICH 7 Family LAN Controller (configuration)', which is checked. At the bottom, there are buttons for 'Help', 'Cancel', 'Back', and 'Next'.

FIGURE 14.2: CONFIGURING NETWORK BRIDGE WITH YAST

Click *Next*. When asked about adapting an already configured device, click *Continue*.

5. Click *OK* to apply the changes. Check if the bridge is created:

```
tux@venus:~> brctl show
bridge name bridge id          STP enabled  interfaces
br0          8000.001676d670e4   no          eth0
```

14.4.3.1 Connecting to a Bridge Manually

Use the following example script to connect VM Guest to the newly created bridge interface `br0`. Several commands in the script are run via the **`sudo`** mechanism because they require root privileges.



Note

Make sure the `tunctl` and `bridge-utils` packages are installed on the VM Host Server. If not, install them with **`zypper in tunctl bridge-utils`**.

```
#!/bin/bash
bridge=br0 ❶
tap=$(sudo tunctl -u $(whoami) -b) ❷
sudo ip link set $tap up ❸
sleep 1s ❹
sudo brctl addif $bridge $tap ❺
qemu-kvm -m 512 -hda /images/sles11sp1_base.raw \
-net nic,vlan=0,model=virtio,macaddr=00:16:35:AF:94:4B \
-net tap,vlan=0,ifname=$tap ❻,script=no ❼,downscript=no
sudo brctl delif $bridge $tap ❽
sudo ip link set $tap down ❾
sudo tunctl -d $tap ❿
```

- ❶ Name of the bridge device.
- ❷ Prepare a new TAP device and assign it to the user who runs the script. TAP devices are virtual network devices often used for virtualization and emulation setups.
- ❸ Bring up the newly created TAP network interface.
- ❹ Make a 1 second pause to make sure the new TAP network interface is really up.
- ❺ Add the new TAP device to the network bridge `br0`.
- ❻ The `ifname=` suboption specifies the name of the TAP network interface used for bridging.

- 7 Before `qemu-kvm` connects to a network bridge, it checks the `script` and `downscript` values. If it finds the specified scripts on the VM Host Server file system, it runs the `script` before it connects to the network bridge and `downscript` after it exits the network environment. You can use these scripts to first set up and bring up the bridged network devices, and then to deconfigure them. By default, `/etc/qemu-ifup` and `/etc/qemu-ifdown` are examined. If `script=no` and `downscript=no` are specified, the script execution is disabled and you have to take care manually.
- 8 Deletes the TAP interface from a network bridge `br0`.
- 9 Sets the state of the TAP device to 'down'.
- 10 Deconfigures the TAP device.

14.4.3.2 Connecting to a Bridge with `qemu-bridge-helper`

Another way to connect VM Guest to a network through a network bridge is by means of the `qemu-bridge-helper` helper program. It configures the TAP interface for you, and attaches it to the specified bridge. The default helper executable is `/usr/lib64/qemu-bridge-helper`. The helper executable is setuid root, which is only executable by the members of the virtualization group (`kvm`). Therefore the `qemu-kvm` command itself does not have to be run under `root` privileges.

You can call the helper the following way:

```
qemu-kvm [...] -net nic,vlan=0,model=virtio -net bridge,vlan=0,br=br0
```

You can specify your own custom helper script that will take care of the TAP device (de)configuration, with the `helper=/path/to/your/helper` option:

```
qemu-kvm [...] -net bridge,vlan=0,br=br1,helper=/path/to/bridge-helper
```



Tip

To define access privileges to `qemu-bridge-helper`, inspect the `/etc/qemu-kvm/bridge.conf` file. For example the following directive

```
allow br0
```

allows the `qemu-kvm` command to connect its VM Guest to the network bridge `br0`.

14.4.4 Accelerated Networking with vhost-net

The `vhost-net` module is used to accelerate KVM's paravirtualized network drivers. It provides better latency and greater throughput for network.

To make use of the module, verify that the host's running Kernel has `CONFIG_VHOST_NET` turned on or enabled as a module:

```
grep CONFIG_VHOST_NET /boot/config-`uname -r`
```

Also verify that the guest's running Kernel has `CONFIG_PCI_MSI` enabled:

```
grep CONFIG_PCI_MSI /boot/config-`uname -r`
```

If both conditions are met, use the `vhost-net` driver by starting the guest with the following example command line:

```
qemu-kvm [...] -netdev tap,id=guest0,vhost=on,script=no  
-net nic,model=virtio,netdev=guest0,macaddr=00:16:35:AF:94:4B
```

Note that `guest0` is an identification string of the vhost-driven device.

14.5 Viewing a VM Guest with VNC

QEMU normally uses an SDL (a cross-platform multimedia library) window to display the graphical output of a VM Guest. With the `-vnc` option specified, you can make QEMU listen on a specified VNC display and redirect its graphical output to the VNC session.



Tip

When working with QEMU's virtual machine via VNC session, it is useful to work with the `-usbdevice tablet` option.

Moreover, if you need to use another keyboard layout than the default `en-us`, specify it with the `-k` option.

The first suboption of `-vnc` must be a *display* value. The `-vnc` option understands the following display specifications:

`host:display`

Only connections from `host` on the display number `display` will be accepted. The TCP port on which the VNC session is then running is normally a `5900 + display number`. If you do not specify `host`, connections will be accepted from any host.

unix:path

The VNC server listens for connections on Unix domain sockets. The path option specifies the location of the related Unix socket.

none

The VNC server functionality is initialized, but the server itself is not started. You can start the VNC server later with the QEMU monitor. For more information, see [Chapter 16, *Administrating Virtual Machines with QEMU Monitor*](#).

```
tux@venus:~> qemu-kvm [...] -vnc :5
(on the client:)
wilber@jupiter:~> vinagre venus:5905 &
```

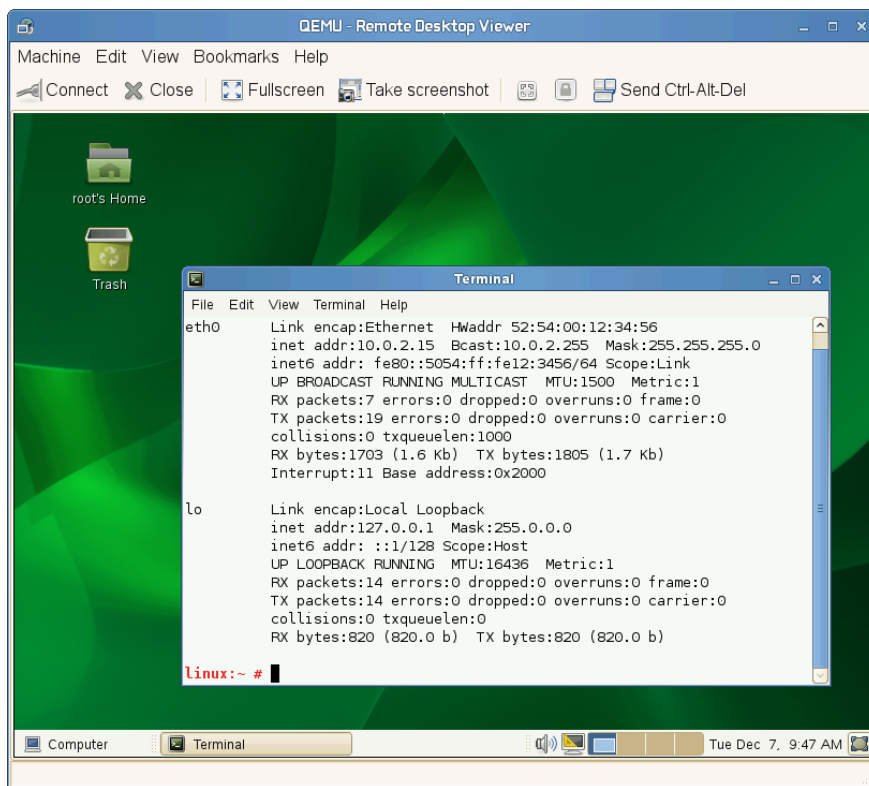


FIGURE 14.3: QEMU VNC SESSION

14.5.1 Secure VNC Connections

The default VNC server setup does not use any form of authentication. In the previous example, any user can connect and view the QEMU VNC Session from any host on the network.

There are several levels of security which you can apply to your VNC client/server connection. You can either protect your connection with a password, use x509 certificates, use SASL authentication, or even combine some of these authentication methods in one QEMU command.

See [Section A.1, “Generating x509 Client/Server Certificates”](#) for more information about the x509 certificates generation. For more information about configuring x509 certificates on a VM Host Server and the client, see [Section 8.2.2, “Remote TLS/SSL Connection with x509 Certificate \(qemu+tls\)”](#) and [Section 8.2.2.3, “Configuring the Client and Testing the Setup”](#).

The Vinagre VNC viewer supports advanced authentication mechanisms. Therefore, it will be used to view the graphical output of VM Guest in the following examples. For this example, let us assume that the server x509 certificates `ca-cert.pem`, `server-cert.pem`, and `server-key.pem` are located in the `/etc/pki/qemu` directory on the host, while the client's certificates are distributed in the following locations on the client:

```
/etc/pki/CA/cacert.pem
/etc/pki/libvirt-vnc/clientcert.pem
/etc/pki/libvirt-vnc/private/clientkey.pem
```

EXAMPLE 14.5: PASSWORD AUTHENTICATION

```
qemu-kvm [...] -vnc :5,password -monitor stdio
```

Starts the VM Guest graphical output on VNC display number 5 (usually port 5905). The `password` suboption initializes a simple password-based authentication method. There is no password set by default and you have to set one with the `change vnc password` command in QEMU monitor:

```
QEMU 0.12.5 monitor - type 'help' for more information
(qemu) change vnc password
Password: ****
```

You need the `-monitor stdio` option here, because you would not be able to manage the QEMU monitor without redirecting its input/output.

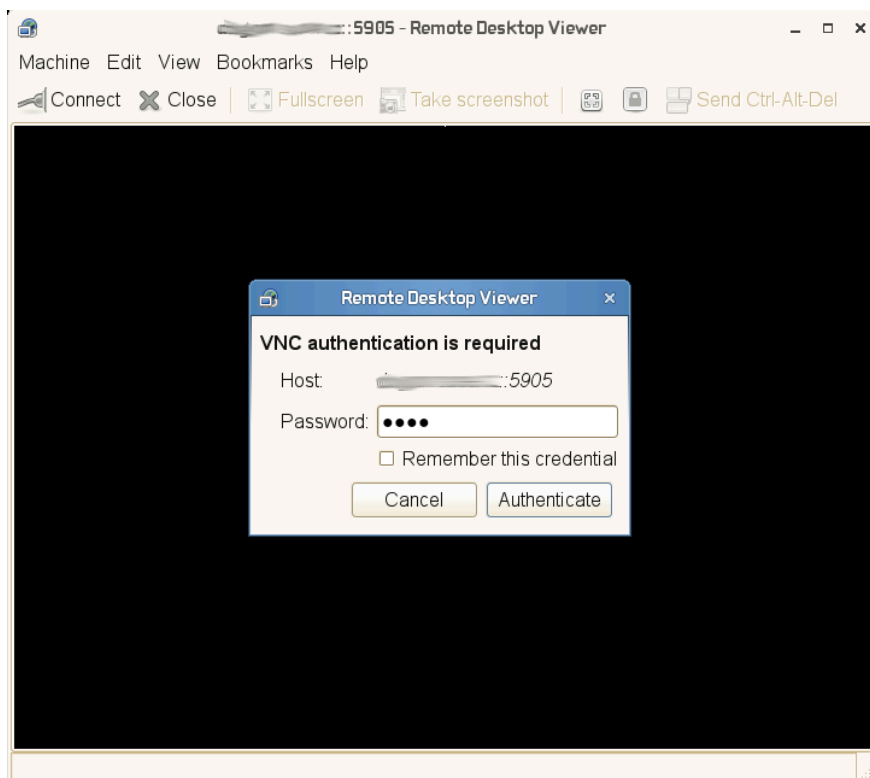


FIGURE 14.4: AUTHENTICATION DIALOG IN VINAGRE

EXAMPLE 14.6: X509 CERTIFICATE AUTHENTICATION

The QEMU VNC server can use TLS encryption for the session and x509 certificates for authentication. The server asks the client for a certificate and validates it against the CA certificate. Use this authentication type if your company provides an internal certificate authority.

```
qemu-kvm [...] -vnc :5,tls,x509verify=/etc/pki/qemu
```

EXAMPLE 14.7: X509 CERTIFICATE AND PASSWORD AUTHENTICATION

You can combine the password authentication with TLS encryption and x509 certificate authentication to create a two-layer authentication model for clients. Remember to set the password in the QEMU monitor after you run the following command:

```
qemu-kvm [...] -vnc :5,password,tls,x509verify=/etc/pki/qemu -monitor stdio
```

EXAMPLE 14.8: SASL AUTHENTICATION

Simple Authentication and Security Layer (SASL) is a framework for authentication and data security in Internet protocols. It integrates several authentication mechanisms, like PAM, Kerberos, LDAP and more. SASL keeps its own user database, so the connecting user accounts do not need to exist on VM Host Server.

For security reasons, you are advised to combine SASL authentication with TLS encryption and x509 certificates:

```
qemu-kvm [...] -vnc :5,tls,x509,sasl -monitor stdio
```

14.6 VirtFS: Sharing Folders between Host and Guests

VM Guests usually run in a separate computing space—they are provided their own memory range, dedicated CPUs, and file system space. Ability to share parts of VM Host Server's file system makes the virtualization environment more flexible by simplifying mutual data exchange. Network file systems, such as CIFS and NFS, have been the traditional way of sharing folders. But as they are not specifically designed for virtualization purposes, they suffer from major performance and feature issues.

KVM introduces a new and more optimized tool called *VirtFS* (sometimes referred to as a “file system pass-through”). VirtFS uses a paravirtual file system driver, which avoids converting the guest application file system operations into block device operations, and then again into host file system operations. VirtFS uses Plan-9 network protocol for communication between the guest and the host.

You can typically use VirtFS to

- access a shared folder from several guests, or to provide guest-to-guest file system access.
- replace the virtual disk as the root file system to which the guest's ramdisk connects to during the guest boot process
- provide storage services to different customers from a single host file system in a cloud environment

14.6.1 Implementation

In QEMU, the implementation of VirtFS is facilitated by defining two types of devices:

- `virtio-9p-pci` device which transports protocol messages and data between the host and the guest.
- `fsdev` device which defines the export file system properties, such as file system type and security model.

EXAMPLE 14.9: EXPORTING HOST'S FILESYSTEM WITH VIRTFS

```
qemu-kvm [...] -fsdev local,id=expl❶,path=/tmp/❷,security_model=mapped❸  
-device virtio-9p-pci,fsdev=expl❹,mount_tag=v_tmp❺
```

- ❶ Identification of the file system to be exported.
- ❷ Filesystem path on the host to be exported.
- ❸ Security model to be used—`mapped` keeps the guest file system modes and permissions isolated from the host, while `none` invokes a “pass-through” security model in which permission changes on the guest's files are reflected on the host as well.
- ❹ The exported file system ID defined before with `-fsdev id=`.
- ❺ Mount tag used later on the guest to mount the exported file system.

Such an exported file system can be mounted on the guest like this

```
mount -t 9p -o trans=virtio v_tmp /mnt
```

where `v_tmp` is the mount tag defined earlier with `-device mount_tag=` and `/mnt` is the mount point where you want to mount the exported file system.

14.7 KSM: Sharing Memory Pages between Guests

Kernel SamePage Merging (KSM) is a Linux Kernel feature which merges identical memory pages from multiple running processes into one memory region. Because KVM guests run as processes under Linux, KSM provides the memory overcommit feature to hypervisors for more efficient use of memory. Therefore, if you need to run multiple virtual machines on a host with limited memory, KSM is the best solution for you.

To make use of KSM, do the following.

1. Verify that KSM is enabled in your running Kernel:

```
grep KSM /boot/config-`uname -r`  
CONFIG_KSM=y
```

If KSM is enabled in the running Kernel, you will see the following files under the /sys/kernel/mm/ksm directory:

```
ls -l /sys/kernel/mm/ksm  
total 0  
drwxr-xr-x 2 root root 0 Nov 9 07:10 ./  
drwxr-xr-x 6 root root 0 Nov 9 07:10 ../  
-r--r--r-- 1 root root 4096 Nov 9 07:10 full_scans  
-r--r--r-- 1 root root 4096 Nov 9 07:10 pages_shared  
-r--r--r-- 1 root root 4096 Nov 9 07:10 pages_sharing  
-rw-r--r-- 1 root root 4096 Nov 9 07:10 pages_to_scan  
-r--r--r-- 1 root root 4096 Nov 9 07:10 pages_unshared  
-r--r--r-- 1 root root 4096 Nov 9 07:10 pages_volatile  
-rw-r--r-- 1 root root 4096 Nov 9 07:10 run  
-rw-r--r-- 1 root root 4096 Nov 9 07:10 sleep_millisecs
```

2. Check if KSM feature is turned on.

```
cat /sys/kernel/mm/ksm/run
```

If the command returns 0, turn KSM on with

```
echo 1 > /sys/kernel/mm/ksm/run
```

3. Now run several VM Guests under KVM and inspect the content of files pages_sharing and pages_shared, for example:

```
while [ 1 ]; do cat /sys/kernel/mm/ksm/pages_shared; sleep 1; done  
13522  
13523  
13519  
13518  
13520  
13520  
13528
```

For more information on the meaning of the /sys/kernel/mm/ksm/* files, see /usr/src/linux/Documentation/vm/ksm.txt (package kernel-source).

15 KVM Disk Cache Modes

15.1 Disk Interface Cache Modes

Qemu-kvm allows for various storage caching strategies to be specified when configuring a KVM guest. Each guest disk interface can have one of the following cache modes specified: *writethrough*, *writeback*, *none*, *directsync*, or *unsafe*. If no cache mode is specified, qemu-kvm uses an appropriate default cache mode. These cache modes influence how host based storage is accessed, as follows:

- read/write data may be cached in the host page cache
- the guest's storage controller is informed whether a write cache is present, allowing for the use of a flush command
- synchronous write mode, in which write requests are reported complete only when committed to the storage device, may be used
- flush commands (generated by the guest storage controller) may be ignored for performance reasons

In the event of an un-orderly disconnection between the guest and its storage, the cache mode in use will affect whether data loss occurs. The cache mode can also affect disk performance significantly. Additionally, some cache modes are incompatible with live migration, depending on a number of factors. There are no simple rules about what combination of cache mode, disk image format, image placement, or storage sub-system is best. The user should plan each guest's configuration carefully and experiment with various configurations to determine the optimal performance.

15.2 Description of Cache Modes

cache mode unspecified

In qemu-kvm versions older than v1.2 (eg SLES11 SP2), not specifying a cache mode meant that *writethrough* would be used as the default. Since that version, the various qemu-kvm guest storage interfaces have been fixed to handle *writeback* or *writethrough* semantics more correctly, allowing for the default caching mode to be switched to *writeback*. The guest

driver for each of ide, scsi, and virtio have within their power to disable the write back cache, causing the caching mode used to revert to *writethrough*. The typical guest's storage drivers will maintain the default caching mode as *writeback*, however.

cache = writethrough

This mode causes qemu-kvm to interact with the disk image file or block device with O_DSYNC semantics, where writes are reported as completed only when the data has been committed to the storage device. The host page cache is used in what can be termed a writethrough caching mode. The guest's virtual storage adapter is informed that there is no writeback cache, so the guest would not need to send down flush commands to manage data integrity. The storage behaves as if there is a writethrough cache.

cache = writeback

This mode causes qemu-kvm to interact with the disk image file or block device with neither O_DSYNC nor O_DIRECT semantics, so the host page cache is used and writes are reported to the guest as completed when placed in the host page cache, and the normal page cache management will handle commitment to the storage device. Additionally, the guest's virtual storage adapter is informed of the writeback cache, so the guest would be expected to send down flush commands as needed to manage data integrity. Analogous to a raid controller with RAM cache.

cache = none

This mode causes qemu-kvm to interact with the disk image file or block device with O_DIRECT semantics, so the host page cache is bypassed and I/O happens directly between the qemu-kvm userspace buffers and the storage device. Because the actual storage device may report a write as completed when placed in its write queue only, the guest's virtual storage adapter is informed that there is a writeback cache, so the guest would be expected to send down flush commands as needed to manage data integrity. Equivalent to direct access to your hosts' disk, performance wise.

cache = unsafe

This mode is similar to the cache=writeback mode discussed above. The key aspect of this “unsafe” mode, is that all flush commands from the guests are ignored. Using this mode implies that the user has accepted the trade-off of performance over risk of data loss in the event of a host failure. Useful, for example, during guest install, but not for production workloads.

`cache=directsync`

This mode causes qemu-kvm to interact with the disk image file or block device with both `O_DSYNC` and `O_DIRECT` semantics, where writes are reported as completed only when the data has been committed to the storage device, and when it is also desirable to bypass the host page cache. Like `cache = writethrough`, it is helpful to guests that do not send flushes when needed. It was the last cache mode added, completing the possible combinations of caching and direct access semantics.

15.3 Data Integrity Implications of Cache Modes

`cache = writethrough`, `cache = none`, `cache=directsync`

These are the safest modes, and considered equally safe, given that the guest operating system is “modern and well behaved”, which means that it uses flushes as needed. If you have a suspect guest, use *writethrough*, or *directsync*. Note that some file systems are not compatible with `cache=none` or `cache=directsync`, as they do not support `O_DIRECT`, which these cache modes relies on.

`cache = writeback`

This mode informs the guest of the presence of a write cache, and relies on the guest to send flush commands as needed to maintain data integrity within its disk image. This is a common storage design which is completely accounted for within modern filesystems. But it should be noted that because there is a window of time between the time a write is reported as completed, and that write being committed to the storage device, this mode exposes the guest to data loss in the unlikely event of a host failure.

`cache = unsafe`

This mode is similar to writeback caching except the guest flush commands are ignored, nullifying the data integrity control of these flush commands, and resulting in a higher risk of data loss due to host failure. The name “unsafe” should serve as a warning that there is a much higher potential for data loss due to a host failure than with the other modes. Note that as the guest terminates, the cached data is flushed at that time.

15.4 Performance Implications of Cache Modes

The choice to make full use of the page cache, or to write through it, or to bypass it altogether can have dramatic performance implications. Other factors which influence disk performance include the capabilities of the actual storage system, what disk image format is used, the potential size of the page cache and the IO scheduler used. Additionally, not flushing the write cache increases performance, but with risk, as noted above. As a general rule, high end systems typically perform best with `cache = none`, because of the reduced data copying that occurs. The potential benefit of having multiple guests share the common host page cache, the ratio of reads to writes, and the use of `aio = native` (see below) should also be considered.

15.5 Effect of Cache Modes on Live Migration

The caching of storage data and meta-data restricts the configurations which support live migration. Currently, only `raw`, `qcow2` and `qed` image formats can be used for live migration. If a clustered file system is used, all cache modes support live migration. Otherwise the only cache mode which supports live migration on read/write shared storage is `cache = none`.

The `libvirt` management layer includes checks for migration compatibility based on a number of factors. If the guest storage is hosted on a clustered filesystem, is readonly or is marked sharable, then the cache mode is ignored when determining if migration can be allowed. Otherwise `libvirt` will not allow migration unless the cache mode is set to `none`. However, this restriction can be overridden with the “unsafe” option to the migration APIs, which is also supported by `virsh`, as for example in

```
virsh migrate --live --unsafe
```



Tip

`cache = none` is required for the IO mode setting `aio = native`. If another cache mode is used, then the IO mode will silently be switched back to the default `aio = threads`. Qemu-kvm implements the guest flush within the host by using `fdatsync()`.

16 Administrating Virtual Machines with QEMU Monitor

When QEMU is running, a monitor console is provided for performing interaction with the user. Using the commands available in the monitor console, it is possible to inspect the running operating system, change removable media, take screen shots or audio grabs and control several other aspects of the virtual machine.

16.1 Accessing Monitor Console

To access the monitor console from QEMU, press `Ctrl - Alt - 2`. To return back to QEMU from the monitor console, press `Ctrl - Alt - 1`.

To get help while using the console, use `help` or `?`. To get help for a specific command, use `help command`.

16.2 Getting Information about the Guest System

To get information about the guest system, use the `info option` command. If used without any option, the list of possible options is printed. Options determine which part of the system will be analyzed:

info version

Shows the version of QEMU

info commands

Lists available QMP commands

info network

Shows the network state

info chardev

Shows the character devices

info block

Information about block devices, such as hard drives, floppy drives, or CD-ROMs

info blockstats

Read and write statistics on block devices

info registers

Shows the CPU registers

info cpus

Shows information about available CPUs

info history

Shows the command line history

info irq

Shows the interrupts statistics

info pic

Shows the i8259 (PIC) state

info pci

Shows the PCI information

info tlb

Shows virtual to physical memory mappings

info mem

Shows the active virtual memory mappings

info jit

Shows dynamic compiler information

info kvm

Shows the KVM information

info numa

Shows the NUMA information

info usb

Shows the guest USB devices

info usbhost

Shows the host USB devices

info profile

Shows the profiling information

info capture

Shows the capture (audio grab) information

info snapshots

Shows the currently saved virtual machine snapshots

info status

Shows the current virtual machine status

info pcmcia

Shows the guest PCMCIA status

info mice

Shows which guest mice is receiving events

info vnc

Shows the VNC server status

info name

Shows the current virtual machine name

info uuid

Shows the current virtual machine UUID

info usernet

Shows the user network stack connection states

info migrate

Shows the migration status

info balloon

Shows the balloon device information

info qtree

Shows the device tree

info qdm

Shows the qdev device model list

info roms

Shows the ROMs

info migrate_cache_sizes

Shows the current migration xbzrle (= Xor Based Zero Run Length Encoding) cache size.

info migrate_capabilities

Shows the status of the various migration capabilities, such as xbzrle compression.

info mtree

Shows the VM Guest memory hierarchy.

info trace-events

Shows available trace-events and their status.

16.3 Changing VNC Password

To change the VNC password, use the change vnc password command and enter the new password:

```
(qemu) change vnc password
Password: *****
(qemu)
```

16.4 Managing Devices

To release the device or file connected to the removable media device, use the eject device command. Use the optional -f to force ejection.

To change removable media (like CD-ROMs), use the change device command. The name of the removable media can be determined using the info block command:

```
(qemu) info block
ide1-cd0: type=cdrom removable=1 locked=0 file=/dev/sr0 ro=1 drv=host_device
(qemu) change ide1-cd0 /path/to/image
```

16.5 Controlling Keyboard and Mouse

It is possible to use the monitor console to emulate keyboard and mouse input if necessary. For example, if your graphical user interface intercepts some key combinations at low level (such as **Ctrl** – **Alt** – **F1** in X Window), you can still enter them using the sendkey keys:

```
sendkey ctrl-alt-f1
```

To list the key names used in the keys option, enter sendkey and press **Tab**.

To control the mouse, the following commands can be used:

mouse_move *dx dy [dz]*

Move the active mouse pointer to the specified coordinates *dx*, *dy* with the optional scroll axis *dz*.

mouse_button *val*

Change the state of the mouse buttons (1 = left, 2 = middle, 4 = right).

mouse_set *index*

Set which mouse device receives events. Device index numbers can be obtained with the **info mice** command.

16.6 Changing Available Memory

If the virtual machine was started with the **-balloon virtio** option and the paravirtualized balloon device that allows to dynamically change the amount of memory available is therefore enabled, it is possible to change the available memory dynamically. For more information about enabling the balloon device, see [Section 13.1, “Basic Installation with **qemu-kvm**”](#).

To get information about the balloon device in the monitor console and to determine whether the device is enabled, use the **info balloon** command:

```
(qemu) info balloon
```

If the balloon device is enabled, use the **balloon memory_in_MB** command to set the requested amount of memory:

```
(qemu) balloon 400
```

16.7 Dumping Virtual Machine Memory

To save the content of the virtual machine memory to a disk or console output, use the following commands:

memsave *addr size filename*

Saves virtual memory dump starting at *addr* of size *size* to file *filename*

pmemsave *addr size filename*

Saves physical memory dump starting at *addr* of size *size* to file *filename* -

`x / fmt addr`

Makes a virtual memory dump starting at address addr and formatted according to the fmt string. The fmt string consists of three parameters countformatsize:

The count parameter is the number of items to be dumped.

The format can be x (hex), d (signed decimal), u (unsigned decimal), o (octal), c (char) or i (assembly instruction).

The size parameter can be b (8 bits), h (16 bits), w (32 bits) or g (64 bits). On x86, h or w can be specified with the i format to respectively select 16 or 32-bit code instruction size. is the number of the items to be dumped.

`xp / fmt addr`

Makes a physical memory dump starting at address addr and formatted according to the fmt string. The fmt string consists of three parameters countformatsize:

The count parameter is the number of the items to be dumped.

The format can be x (hex), d (signed decimal), u (unsigned decimal), o (octal), c (char) or i (asm instruction).

The size parameter can be b (8 bits), h (16 bits), w (32 bits) or g (64 bits). On x86, h or w can be specified with the i format to respectively select 16 or 32-bit code instruction size. is the number of the items to be dumped.

16.8 Managing Virtual Machine Snapshots



Warning

Managing snapshots in QEMU monitor is not officially supported by SUSE yet. However, the information found in this section may be helpful in specific cases.

Virtual machine snapshots are snapshots of the complete virtual machine including the state of CPU, RAM, and the content of all writable disks. To use virtual machine snapshots, you must have at least one non-removable and writable block device using the qcow2 disk image format. Snapshots are helpful when you need to save your virtual machine in a particular state. For example, after you configured network services on a virtualized server and want to quickly start the virtual machine in the same state that has been saved last. You can also create a snapshot after the virtual machine has been powered off to create a backup state before you

try something experimental and possibly make VM Guest unstable. This section introduces the former case, while the latter is described in [Section 13.2.3, “Managing Snapshots of Virtual Machines with qemu-img”](#).

The following commands are available for managing snapshots in QEMU monitor:

savevm *name*

Creates a new virtual machine snapshot under the tag *name* or replaces an existing snapshot.

loadvm *name*

Loads a virtual machine snapshot tagged *name*.

delvm

Deletes a virtual machine snapshot.

info snapshots

Prints information about available snapshots.

```
(qemu) info snapshots
Snapshot list:
ID ①      TAG ②      VM SIZE ③      DATE ④      VM CLOCK ⑤
1        booting    4.4M 2010-11-22 10:51:10    00:00:20.476
2        booted    184M 2010-11-22 10:53:03    00:02:05.394
3        logged_in 273M 2010-11-22 11:00:25    00:04:34.843
4        ff_and_term_running 372M 2010-11-22 11:12:27    00:08:44.965
```

- ① Unique identification number of the snapshot. Usually auto-incremented.
- ② Unique description string of the snapshot. It is meant as a human readable version of the ID.
- ③ The disk space occupied by the snapshot. Note that the more memory is consumed by running applications, the bigger the snapshot is.
- ④ Time and date the snapshot was created.
- ⑤ The current state of the virtual machine's clock.

16.9 Suspending and Resuming Virtual Machine Execution

The following commands are available for suspending and resuming virtual machines:

stop

Suspends the execution of the virtual machine.

cont

Resumes the execution of the virtual machine.

system_powerdown

Sends an ACPI shutdown request to the machine. The effect is similar to the power button on a physical machine.

q or quit

Terminates QEMU immediately.

16.10 Live Migration

The live migration process allows to transmit any virtual machine from one host system to another host system without any interruption in availability. It is possible to change hosts permanently or just during a maintenance. It is recommended that the source and destination systems have the same architecture, however it is possible to migrate between hosts with AMD and Intel architectures.

The requirements for the live migration:

- Live migration is only possible between VM Host Servers with the same CPU features. The only supported CPU model for migration is -cpu qemu64 (default) with no additional features specified.
- No physical devices can be passed from host to guest.
- The VM Host Server and VM Guest need to have proper timekeeping installed.
- AHCI interface, virtfs feature, and the -mem-path command-line option are not compatible with migration.
- Migration from SP3 to SP2 or SP1 hosted guests is not supported.

- The virtual machine image must be accessible on both source and destination hosts. For example, it can be located on a shared NFS disk.
- The image directory should be located in the same path on both hosts.
- Both hosts must be located in the same subnet.
- The guest on the source and destination hosts must be started in the same way.

The live migration process has the following steps:

1. The virtual machine instance is running on the source host.
2. The virtual machine is started on the destination host in the frozen listening mode. The parameters used are the same as on the source host plus the `-incoming tcp:ip:port` parameter, where `ip` specifies the IP address and `port` specifies the port for listening to the incoming migration. If 0 is set as IP address, the virtual machine listens on all interfaces.
3. On the source host, switch to the monitor console and use the `migrate -d tcp: destination_ip:port` command to initiate the migration.
4. To determine the state of the migration, use the `info migrate` command in the monitor console on the source host.
5. To cancel the migration, use the `migrate_cancel` command in the monitor console on the source host.
6. To set the maximum tolerable downtime for migration in seconds, use the `migrate_set_downtime number_of_seconds` command.
7. To set the maximum speed for migration in bytes per second, use the `migrate_set_speed bytes_per_second` command.

A Appendix

A.1 Generating x509 Client/Server Certificates

In order to be able to create x509 client and server certificates you need to issue them by a Certificate Authority (CA). It is recommended to set up an independent CA that only issues certificates for `libvirt`.

1. Set up a CA as described in *Book "Security Guide", Chapter 17 "Managing X.509 Certification", Section 17.2 "YaST Modules for CA Management", Section 17.2.1 "Creating a Root CA"*.
2. Create a server and a client certificate as described in *Book "Security Guide", Chapter 17 "Managing X.509 Certification", Section 17.2 "YaST Modules for CA Management", Section 17.2.4 "Creating or Revoking User Certificates"*. The Common Name (CN) for the server certificate must be the full qualified hostname, the Common Name for the client certificate can be freely chosen. For all other fields stick with the defaults suggested by YaST.
Export the client and server certificates to a temporary location (for example, `/tmp/x509/`) by performing the following steps:

- a. Select the certificate on the *certificates* tab.
- b. Choose *Export > Export to File > Certificate and the Key Unencrypted in PEM Format*, provide the *Certificate Password* and the full path and the filename under *File Name*, for example, `/tmp/x509/server.pem` or `/tmp/x509/client.pem`.
- c. Open a terminal and change to the directory where you have saved the certificate and issue the following commands to split it into certificate and key (this example splits the server key):

```
csplit -z -f s_ server.pem '/-----BEGIN/' '{1}'  
mv s_00 servercert.pem  
mv s_01 serverkey.pem
```

- d. Repeat the procedure for each client and server certificate you would like to export.
3. Finally export the CA certificate by performing the following steps:
 - a. Switch to the *Description* tab.

- b. Choose *Advanced* > *Export to File* > *Only the Certificate in PEM Format* and enter the full path and the filename under *File Name*, for example, /tmp/x509/cacert.pem.

A.2 QEMU Command Line Options

A.2.1 Supported **qemu-kvm** Command Line Options

-alt-grab
-append ...
-audio-help
-balloon ...
-boot ...
-cdrom ...
-chardev ...
-clock
-cpu ...
-ctrl-grab
-d ...
-daemonize
-debugcon ...
-device [isa-serial|isa-parallel|isa-fdc|ide-drive| ide-hd|ide-cd|pci-assign|kvm-pci-assign|VGA| cirrus-vga|rtl8139|virtio-net-pci|virtio-blk-pci|virtio-balloon-pci|virtio-9p-pci|usb-hub|usb-ehci| usb-tablet|usb-storage|usb-mouse|usb-kbd|virtserialport| virtconsole|virtio-serial-pci|virtio-serial|sga|i 82559er|e1000|virtio-scsi-pci|scsi-cd|scsi-hd|scsi-generic| scsi-disk|scsi-block|pc-sysfw|pci-serial|pci-serial-2x| pci-serial-4x|ich9-ahci|piix-usb-uhci|usb-host|usb-serial |usb-wacom-tablet|usb_braille|usb-net|pci-ohci| piix4-usb-uhci|virtio-rng-pci]
-display ...
-drive if=[ide|floppy|virtio] format=[raw|qcow2|qed] snapshot=off ...
-echr ...
-enable-kvm
-fda/-fdb ...
-fsdev ...

-full-screen
-gdb ...
-global ...
-h
-hda/-hdb/-hdc/-hdd ...
-help
-incoming ...
-initrd ...
-kernel ...
-loadvm ...
-m ...
-machine [help|?|none|pc|pc-0.12|pc-0.14|pc-0.15|pc-i440fx-1.4]
-mem-path ...
-mem-prealloc
-mon ...
-monitor ...
-M [help|?|none|pc|pc-0.12|pc-0.14|pc-0.15|pc-i440fx-1.4]
-name ...
-netdev ...
-net [nic|user|tap|bridge|none] mode=[rtl8139|e1000|virtio]
-no-acpi
-nodefaults
-nodefconfig
-no-frame
-nographic
-no-hpet
-no-quit
-no-reboot
-no-shutdown
-no-user-config
-object
-parallel ...
-pidfile ...
-qmp ...
-readconfig ...
-rtc ...

- runas ...
- s
- S
- sandbox ...
- sdl
- serial ...
- smbios ...
- smp ...
- tdf
- usb
- usbdevice [disk|host|serial|braille|net|tablet|mouse]
- uuid ..
- version
- vga [std|cirrus|none]
- virtfs ...
- vnc ...
- watchdog ...
- watchdog-action ...
- writeconfig ...

A.2.1.1 Deprecated features

The use of boot=on for virtio disks is no longer needed since the BIOS used supports the virtio block interface directly. In fact, its usage may cause problems, and is now considered deprecated.

The use of ? as a parameter to -cpu, -soundhw, -device, -M, -machine, -d, and -clock is now considered deprecated. Use help instead.

The -tdf command line option is now considered deprecated. The unsupported -no-kvm-pit command line option is now considered deprecated.

The unsupported -no-kvm-pit-reinjection command line option is now considered deprecated.

The -pcidevice qemu-kvm command line option is no longer recognized. Use -device pci-assign instead.

The unsupported -device testdev command line option is no longer recognized. Use -device pc-testdev instead.

The unsupported `-kvm-shadow-memory` command line option is no longer recognized. Its function is now accessible via the `kvm_shadow_mem=` parameter to the `-machine` command line option.

The unsupported `-no-kvm-irqchip` command line option is now considered deprecated. Its function is now accessible via the `kernel_irqchip=` parameter to the `-machine` command line option.

The unsupported `-osk qemu-kvm` command line option is no longer recognized.

The unsupported `-M mac qemu-kvm` command line option is no longer recognized.

The unsupported `-enable-nesting` command line option is no longer recognized.

The unsupported `-old-param` command line option is no longer recognized.

The unsupported `-semihosting` command line option is no longer recognized.

The unsupported `-nvrAm` command line option is no longer recognized.

The unsupported `cpu_set` monitor command is no longer recognized.

The deprecated Windows drivers (`win-virtio-drivers.iso`) are no longer provided. The Virtual Machine Driver Pack is the supported way to get virtio drivers for Windows guests.

A.2.2 Unsupported **qemu-kvm** Command Line Options

The following **qemu-kvm** command line options are *not* supported by SUSE:

`-acpitable ...`
`-add-fd ...`
`-bios ...`
`-bt ...`
`-chroot ...`
`-curses`
`-device [ipoctal232|sysbus-ohci|i82562|ccid-card-passthru| smbus-eeeprom|nec-usb-xhci|hda-duplex|hda-output|cfi.pflash01 |ivshmem|usb-bot|lsi53c895a|ich9-usb-uhci2|ich9-usb-uhci6| q35-pcihost|ich9-usb-uhci5|ich9-usb-uhci3|i6300esb|isa-debug-exit|ne2k_pci|vfio-pci|usb-uas|ich9-usb-uhci4| ioh3420|isa-ide|esp|usb-ccid|ich9-usb-ehci2|pcnet| ich9-intel-hda|dc390|ich9-usb-ehci1|sysbus-ah-ci|hda-micro| pci-bridge|x3130-upstream|isa-cirrus-vga|ich9-usb-uhci1| pc-testdev|ne2k_isa|isa-vga|cs4231a|sysbus-fdc|gus|| vmware-svga||i82801b11-bridge|i82557a|i82557c|i82557b| i82801|AC97|am53c974|intel-hda||i82558a|`

i82558b|usb-audio| i82550|isa-debugcon|ib700|sb16|megasas|i82551| xio3130-
downstream|vt82c686b-usb-uhci|tpci200|i82559a| i82559b|i82559c|xlnx,ps7-usb|
SUNW,fdtwo|isa-applesmc| exynos4210-ehci-usb|mch|usb-bt-dongle]
-drive if=[scsi|mtid|pflash], snapshot=on, format=[anything besides from raw,
qcow2, qed]
-dtb
-g ...
-icount ...
-iscsi ...
-L ...
-machine [q35|pc-q35-1.4|pc-1.3|pc-1.2|pc-1.1|pc-1.0|pc-0.13|pc-0.11|pc-0.10|
isapc]
-M [q35|pc-q35-1.4|pc-1.3|pc-1.2|pc-1.1|pc-1.0|pc-0.13|pc-0.11|pc-0.10|isapc]
-mtdblock ...
-net [socket|dump] ...
-no-fd-bootchk
-no-kvm
-no-kvm-irqchip
-no-kvm-pit
-no-kvm-pit-reinjection
-numa ...
-option-rom ...
-pflash ...
-portrait
-prom-env ...
-qtest ...
-qtest-log ...
-rotate
-sd ...
-set ...
-show-cursor
-singlestep
-snapshot
-soundhw ...
-spice ...
-tb-size ...

```
-trace ...  
-vga [vmware|qxl|xenfb]  
-virtioconsole ...  
-win2k-hack
```

A.2.3 Supported **qemu-kvm** monitor Command Line Options

The following **qemu-kvm** `monitor` command line options are supported by SUSE:

```
?  
balloon target ...  
block_resize ...  
boot_set ...  
[c|cont]  
change device ...  
cpu ...  
delvm ...  
device_add ...  
device_del ...  
drive_add ...  
drive_del ...  
dump_guest_memory ...  
eject ...  
gdbserver ...  
help  
info ...  
loadvm ...  
logfile ...  
logitem ...  
mce ...  
memsave ...  
migrate ...  
migrate_set_cache_size ...  
migrate_set_capability ...  
migrate_set_downtime ...  
migrate_set_speed ...
```

mouse_button ...
mouse_move ...
mouse_set ...
nmi ...
pci_add ...
pci_del ...
pmemsave ...
[p|print] ...
q
savevm ...
sendkey ...
stop
system_powerdown
system_reset
system_wakeup
usb_add ...
usb_del ...
watchdog_action ...
x ...
xp ...

A.2.4 Unsupported **qemu-kvm** monitor Command Line Options

The following qemu-kvm monitor command line options are *not* supported by SUSE:

acl_add ...
acl_policy ...
acl_remove ...
acl_reset ...
acl_show ...
block_job_cancel ...
block_job_complete ...
block_job_pause ...
block_job_resume ...
block_job_set_speed ...
block_passwd ...

client_migrate_info ...
close_fd ...
commit ...
drive_mirror ...
expire_password ...
hostfwd_add ...
hostfwd_remove ...
host_net_add ...
host_net_remove ...
i ...
migrate_cancel
nbd_server_add ...
nbd_server_start ...
nbd_server_stop ...
netdev_add
netdev_del ...
o ...
pcie_aer_inject_error ...
ringbuf_read ...
ringbuf_write ...
screendump ...
set_link ...
set_password ...
singlestep ...
snapshot_blkdev ...
stopcapture ...
sum ...
trace_event ...
wavcapture ...

In addition to the listed human monitor commands above, a JSON-based monitor interface called QMP (Qemu Monitor Protocol) is provided which allows for a more programmatic and control-oriented interaction with the monitor. See </usr/share/doc/packages/kvm/qmp-commands.txt> for details on executing QMP commands. Below is the list of QMP commands:

add_client
add-fd

balloon
block_passwd
block_resize
block_set_io_throttle
block-snapshot-sync
change
chardev-add
chardev-remove
client_migrate_info
closefd
cont
cpu
device_add
device_del
drive-mirror
eject
expire_password
getfd
human-monitor-command
inject-nmi
memsave
migrate
migrate_cancel
migrate-set-cache-size
migrate-set-capabilities
migrate_set_downtime
migrate_set_speed
netdev_add
netdev_del
pmemsave
qmp_capabilities
query-balloon
query-block
query-blockstats
query-chardev
query-commands

query-cpus
query-events
query-fdsets
query-kvm
query-mice
query-migrate
query-migrate-cache-size
query-name
query-pci
query-spice
query-status
query-uuid
query-version
query-vnc
quit
remove-fd
ringbuf-read
ringbuf-write
screendump
send-key
set_link
set_password
stop
system_powerdown
system_reset
system_wakeup
transaction
xen-save-devices-state
xen-set-global-dirty-flag

B GNU Licenses

This appendix contains the GNU Free Documentation License version 1.2.

B.1 GNU Free Documentation License

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with gener-

ic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near

the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named sub-unit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further

copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version

to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the sec-

tion all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents,

make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sec-

tions. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail. If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the

present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy,
distribute and/or modify this document
under the terms of the GNU Free
Documentation License, Version 1.2
or any later version published by the
Free Software Foundation;
with no Invariant Sections, no Front-
Cover Texts, and no Back-Cover Texts.
A copy of the license is included in
the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST
THEIR TITLES, with the
Front-Cover Texts being LIST, and with
the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.