Red Hat Enterprise Linux 6

Virtualization Administration Guide

Managing your virtual environment

Last Updated: 2022-06-24

Red Hat Enterprise Linux 6 Virtualization Administration Guide

Managing your virtual environment

Jiri Herrmann
Red Hat Customer Content Services
jherrman@redhat.com

Yehuda Zimmerman
Red Hat Customer Content Services

Laura Novich
Red Hat Customer Content Services

Scott Radvan
Red Hat Customer Content Services

Dayle Parker
Red Hat Customer Content Services

Legal Notice

Abstract

The Virtualization Administration Guide covers administration of host physical machines, networking, storage, device and guest virtual machine management, and troubleshooting. To expand your expertise, you might also be interested in the Red Hat Virtualization (RH318) training course.

Table of Contents

CHAPTER 1. SERVER BEST PRACTICES 13

CHAPTER 2. SVIRT 14
2.1. SECURITY AND VIRTUALIZATION 15
2.2. SVIRT LABELING 16

CHAPTER 3. CLONING VIRTUAL MACHINES 17
3.1. PREPARING VIRTUAL MACHINES FOR CLONING 17
3.2. CLONING A VIRTUAL MACHINE 20
3.2.1. Cloning Guests with virt-clone 20
3.2.2. Cloning Guests with virt-manager 21

CHAPTER 4. KVM LIVE MIGRATION 24
4.1. LIVE MIGRATION REQUIREMENTS 24
4.2. LIVE MIGRATION AND RED HAT ENTERPRISE LINUX VERSION COMPATIBILITY 26
4.3. SHARED STORAGE EXAMPLE: NFS FOR A SIMPLE MIGRATION 27
4.4. LIVE KVM MIGRATION WITH VIRSH 28
4.4.1. Additional Tips for Migration with virsh 30
4.4.2. Additional Options for the virsh migrate Command 31
4.5. MIGRATING WITH VIRT-MANAGER 32

CHAPTER 5. REMOTE MANAGEMENT OF GUESTS 39
5.1. REMOTE MANAGEMENT WITH SSH 39
5.2. REMOTE MANAGEMENT OVER TLS AND SSL 41
5.3. TRANSPORT MODES 44

CHAPTER 6. OVERCOMMITTING WITH KVM 48
6.1. OVERCOMMITTING MEMORY 48
6.2. OVERCOMMITTING VIRTUALIZED CPUS 48

CHAPTER 7. KSM 50

CHAPTER 8. ADVANCED GUEST VIRTUAL MACHINE ADMINISTRATION 54
8.1. CONTROL GROUPS (CGROUPS) 54
8.2. HUGE PAGE SUPPORT 54
8.3. RUNNING RED HAT ENTERPRISE LINUX AS A GUEST VIRTUAL MACHINE ON A HYPER-V HYPERVISOR 55
8.4. GUEST VIRTUAL MACHINE MEMORY ALLOCATION 55
8.5. AUTOMATICALLY STARTING GUEST VIRTUAL MACHINES 56
8.6. DISABLE SMART DISK MONITORING FOR GUEST VIRTUAL MACHINES 57
8.7. CONFIGURING A VNC SERVER 57
8.8. GENERATING A NEW UNIQUE MAC ADDRESS 57
8.8.1. Another Method to Generate a New MAC for Your Guest Virtual Machine 58
8.9. IMPROVING GUEST VIRTUAL MACHINE RESPONSE TIME 58
8.10. VIRTUAL MACHINE TIMER MANAGEMENT WITH LIBVIRT 59
8.10.1. timer Child Element for clock 60
8.10.2. track 61
8.10.3. tickpolicy 61
8.10.4. frequency, mode, and present 61
8.10.5. Examples Using Clock Synchronization 62
8.11. USING PMU TO MONITOR GUEST VIRTUAL MACHINE PERFORMANCE 63
8.12. GUEST VIRTUAL MACHINE POWER MANAGEMENT 63

CHAPTER 9. GUEST VIRTUAL MACHINE DEVICE CONFIGURATION 65

9.1. PCI DEVICES 66
9.1.1. Assigning a PCI Device with virsh 67
9.1.2. Assigning a PCI Device with virt-manager 70
9.1.3. PCI Device Assignment with virt-install 73
9.1.4. Detaching an Assigned PCI Device 75
9.1.5. Creating PCI Bridges 77
9.1.6. PCI Passthrough 77
9.1.7. Configuring PCI Assignment (Passthrough) with SR-IOV Devices 78
9.1.8. Setting PCI Device Assignment from a Pool of SR-IOV Virtual Functions 80
9.2. USB DEVICES 82
9.2.1. Assigning USB Devices to Guest Virtual Machines 82
9.2.2. Setting a Limit on USB Device Redirection 83
9.3. CONFIGURING DEVICE CONTROLLERS 84
9.4. SETTING ADDRESSES FOR DEVICES 87
9.5. MANAGING STORAGE CONTROLLERS IN A GUEST VIRTUAL MACHINE 89
9.6. RANDOM NUMBER GENERATOR (RNG) DEVICE 90

CHAPTER 10. QEMU-IMG AND QEMU GUEST AGENT 92
10.1. USING QEMU-IMG 92
10.2. QEMU GUEST AGENT 97
10.2.1. Install and Enable the Guest Agent 97
10.2.2. Setting up Communication between Guest Agent and Host 97
10.2.3. Using the QEMU Guest Agent 98
10.2.4. Using the QEMU Guest Agent with libvirt 98
10.2.5. Creating a Guest Virtual Machine Disk Backup 99
10.3. RUNNING THE QEMU GUEST AGENT ON A WINDOWS GUEST 100
10.3.1. Using libvirt Commands with the QEMU Guest Agent on Windows Guests 102
10.4. SETTING A LIMIT ON DEVICE REDIRECTION 103
10.5. DYNAMICALLY CHANGING A HOST PHYSICAL MACHINE OR A NETWORK BRIDGE THAT IS ATTACHED TO A VIRTUAL NIC 104

CHAPTER 11. STORAGE CONCEPTS 105
11.1. STORAGE POOLS 105
11.2. VOLUMES 106

CHAPTER 12. STORAGE POOLS 108
12.1. DISK-BASED STORAGE POOLS 108
12.1.1. Creating a Disk-based Storage Pool Using virsh 109
12.1.2. Deleting a Storage Pool Using virsh 111
12.2. PARTITION-BASED STORAGE POOLS 112
12.2.1. Creating a Partition-based Storage Pool Using virt-manager 112
12.2.2. Deleting a Storage Pool Using virt-manager 115
12.2.3. Creating a Partition-based Storage Pool Using virsh 116
12.2.4. Deleting a Storage Pool Using virsh 118
12.3. DIRECTORY-BASED STORAGE POOLS 118
12.3.1. Creating a Directory-based Storage Pool with virt-manager 119
12.3.2. Deleting a Storage Pool Using virt-manager 122
12.3.3. Creating a Directory-based Storage Pool with virsh 123
12.3.4. Deleting a Storage Pool Using virsh 125
12.4. LVM-BASED STORAGE POOLS 125
12.4.1. Creating an LVM-based Storage Pool with virt-manager 126
12.4.2. Deleting a Storage Pool Using virt-manager 130
12.4.3. Creating an LVM-based Storage Pool with virsh 131
12.4.4. Deleting a Storage Pool Using virsh 133

12.5. ISCSI-BASED STORAGE POOLS 133
12.5.1. Configuring a Software iSCSI Target 133
12.5.2. Adding an iSCSI Target to virt-manager 137
12.5.3. Deleting a Storage Pool Using virt-manager 139
12.5.4. Creating an iSCSI-based Storage Pool with virsh 140
12.5.5. Deleting a Storage Pool Using virsh 142
12.6. NFS-BASED STORAGE POOLS 142
12.6.1. Creating an NFS-based Storage Pool with virt-manager 142
12.6.2. Deleting a Storage Pool Using virt-manager 144
12.7. GLUSTERFS STORAGE POOLS 145
12.8. USING AN NPIV VIRTUAL ADAPTER (VHBA) WITH SCSI DEVICES 145
12.8.1. Creating a vHBA 146
12.8.2. Creating a Storage Pool Using the vHBA 147
12.8.3. Configuring the Virtual Machine to Use a vHBA LUN 149
12.8.4. Destroying the vHBA Storage Pool 149

CHAPTER 13. VOLUMES 151
13.1. CREATING VOLUMES 151
13.2. CLONING VOLUMES 151
13.3. ADDING STORAGE DEVICES TO GUESTS 152
13.3.1. Adding File-based Storage to a Guest 152
13.3.2. Adding Hard Drives and Other Block Devices to a Guest 155
13.4. DELETING AND REMOVING VOLUMES 156

CHAPTER 14. MANAGING GUEST VIRTUAL MACHINES WITH VIRSH 157
14.1. GENERIC COMMANDS 157
14.1.1. help 157
14.1.2. quit and exit 158
14.1.3. version 158
14.1.4. Argument Display 158
14.1.5. connect 158
14.1.6. Displaying Basic Information 159
14.1.7. Injecting NMI 159
14.2. ATTACHING AND UPDATING A DEVICE WITH VIRSH 159
14.3. ATTACHING INTERFACE DEVICES 160
14.4. CHANGING THE MEDIA OF A CDROM 161
14.5. DOMAIN COMMANDS 161
14.5.1. Configuring a Domain to be Started Automatically at Boot 161
14.5.2. Connecting the Serial Console for the Guest Virtual Machine 161
14.5.3. Defining a Domain with an XML File 162
14.5.4. Editing and Displaying a Description and Title of a Domain 162
14.5.5. Displaying Device Block Statistics 162
14.5.6. Retrieving Network Statistics 163
14.5.7. Modifying the Link State of a Domain's Virtual Interface 163
14.5.8. Listing the Link State of a Domain's Virtual Interface 163
14.5.9. Setting Network Interface Bandwidth Parameters 163
14.5.10. Retrieving Memory Statistics for a Running Domain 164
14.5.11. Displaying Errors on Block Devices 164
14.5.12. Displaying the Block Device Size 164
14.5.13. Displaying the Block Devices Associated with a Domain 164
14.5.14. Displaying Virtual Interfaces Associated with a Domain 165
14.5.15. Using blockcommit to Shorten a Backing Chain 165
14.5.16. Using blockpull to Shorten a Backing Chain 165

14.5.17. Using blockresize to Change the Size of a Domain Path 167
14.5.18. Disk Image Management with Live Block Copy 167
14.5.19. Displaying a URI for Connection to a Graphical Display 169
14.5.20. Domain Retrieval Commands 169
14.5.21. Converting QEMU Arguments to Domain XML 170
14.5.22. Creating a Dump File of a Domain's Core 171
14.5.23. Creating a Virtual Machine XML Dump (Configuration File) 171
14.5.24. Creating a Guest Virtual Machine from a Configuration File 173
14.6. EDITING A GUEST VIRTUAL MACHINE'S CONFIGURATION FILE 173
14.6.1. Adding Multifunction PCI Devices to KVM Guest Virtual Machines 173
14.6.2. Stopping a Running Domain to Restart It Later 174
14.6.3. Displaying CPU Statistics for a Specified Domain 174
14.6.4. Saving a Screenshot 174
14.6.5. Sending a Keystroke Combination to a Specified Domain 175
14.6.6. Sending Process Signal Names to Virtual Processes 175
14.6.7. Displaying the IP Address and Port Number for the VNC Display 176
14.7. NUMA NODE MANAGEMENT 176
14.7.1. Displaying Node Information 176
14.7.2. Setting NUMA Parameters 176
14.7.3. Displaying the Amount of Free Memory in a NUMA Cell 177
14.7.4. Displaying a CPU List 177
14.7.5. Displaying CPU Statistics 177
14.7.6. Suspending the Host Physical Machine 177
14.7.7. Setting and Displaying the Node Memory Parameters 178
14.7.8. Creating Devices on Host Nodes 178
14.7.9. Detaching a Node Device 178
14.7.10. Retrieving a Device's Configuration Settings 178
14.7.11. Listing Devices on a Node 178
14.7.12. Triggering a Reset for a Node 179
14.8. STARTING, SUSPENDING, RESUMING, SAVING, AND RESTORING A GUEST VIRTUAL MACHINE 179
14.8.1. Starting a Defined Domain 179
14.8.2. Suspending a Guest Virtual Machine 180
14.8.3. Suspending a Running Domain 180
14.8.4. Waking Up a Domain from a pmsuspend State 180
14.8.5. Undefining a Domain 180
14.8.6. Resuming a Guest Virtual Machine 181
14.8.7. Save a Guest Virtual Machine 181
14.8.8. Updating the Domain XML File that will be Used for Restoring the Guest 182
14.8.9. Extracting the Domain XML File 182
14.8.10. Edit Domain XML Configuration Files 182
14.8.11. Restore a Guest Virtual Machine 182
14.9. SHUTTING DOWN, REBOOTING, AND FORCING SHUTDOWN OF A GUEST VIRTUAL MACHINE 182
14.9.1. Shutting Down a Guest Virtual Machine 183
14.9.2. Shutting Down Red Hat Enterprise Linux 6 Guests on a Red Hat Enterprise Linux 7 Host 183
14.9.3. Manipulating the libvirt-guests Configuration Settings 185
14.9.4. Rebooting a Guest Virtual Machine 187
14.9.5. Forcing a Guest Virtual Machine to Stop 187
14.9.6. Resetting a Virtual Machine 188
14.10. RETRIEVING GUEST VIRTUAL MACHINE INFORMATION 188
14.10.1. Getting the Domain ID of a Guest Virtual Machine 188
14.10.2. Getting the Domain Name of a Guest Virtual Machine 188
14.10.3. Getting the UUID of a Guest Virtual Machine 188
14.10.4. Displaying Guest Virtual Machine Information 188

14.11. STORAGE POOL COMMANDS 189
14.11.1. Searching for a Storage Pool XML 189
14.11.2. Creating, Defining, and Starting Storage Pools 190
14.11.2.1. Building a storage pool 190
14.11.2.2. Creating and defining a storage pool from an XML file 190
14.11.2.3. Creating and starting a storage pool from raw parameters 190
14.11.2.4. Auto-starting a storage pool 190
14.11.3. Stopping and Deleting Storage Pools 191
14.11.4. Creating an XML Dump File for a Storage Pool 191
14.11.5. Editing the Storage Pool's Configuration File 191
14.11.6. Converting Storage Pools 191
14.12. STORAGE VOLUME COMMANDS 191
14.12.1. Creating Storage Volumes 191
14.12.1.1. Creating a storage volume from an XML file 192
14.12.1.2. Cloning a storage volume 192
14.12.2. Deleting Storage Volumes 192
14.12.3. Dumping Storage Volume Information to an XML File 193
14.12.4. Listing Volume Information 193
14.12.5. Retrieving Storage Volume Information 193
14.12.6. Uploading and Downloading Storage Volumes 194
14.12.6.1. Uploading contents to a storage volume 194
14.12.6.2. Downloading the contents from a storage volume 194
14.12.7. Re-sizing Storage Volumes 194
14.13. DISPLAYING PER-GUEST VIRTUAL MACHINE INFORMATION 194
14.13.1. Displaying the Guest Virtual Machines 194
14.13.2. Displaying Virtual CPU Information 196
14.13.3. Configuring Virtual CPU Affinity 196
14.13.4. Displaying Information about the Virtual CPU Counts of a Domain 197
14.13.5. Configuring Virtual CPU Affinity 197
14.13.6. Configuring Virtual CPU Count 198
14.13.7. Configuring Memory Allocation 199
14.13.8. Changing the Memory Allocation for the Domain 200
14.13.9. Displaying Guest Virtual Machine Block Device Information 200
14.13.10. Displaying Guest Virtual Machine Network Device Information 200
14.14. MANAGING VIRTUAL NETWORKS 200
14.15. MIGRATING GUEST VIRTUAL MACHINES WITH VIRSH 201
14.15.1. Interface Commands 201
14.15.1.1. Defining and starting a host physical machine interface via an XML file 202
14.15.1.2. Editing the XML configuration file for the host interface 202
14.15.1.3. Listing active host interfaces 202
14.15.1.4. Converting a MAC address into an interface name 202
14.15.1.5. Stopping a specific host physical machine interface 202
14.15.1.6. Displaying the host configuration file 203
14.15.1.7. Creating bridge devices 203
14.15.1.8. Tearing down a bridge device 203
14.15.1.9. Manipulating interface snapshots 203
14.15.2. Managing Snapshots 203
14.15.2.1. Creating Snapshots 203
14.15.2.2. Creating a snapshot for the current domain 204
14.15.2.3. Taking a snapshot of the current domain 205
14.15.2.4. snapshot-edit-domain 205
14.15.2.5. snapshot-info-domain 205
14.15.2.6. snapshot-list-domain 206

14.15.2.7. snapshot-dumpxml domain snapshot 206
14.15.2.8. snapshot-parent domain 207
14.15.2.9. snapshot-revert domain 207
14.15.2.10. snapshot-delete domain 207
14.16. GUEST VIRTUAL MACHINE CPU MODEL CONFIGURATION 208
14.16.1. Introduction 208
14.16.2. Learning about the Host Physical Machine CPU Model 208
14.16.3. Determining a Compatible CPU Model to Suit a Pool of Host Physical Machines 209
14.17. CONFIGURING THE GUEST VIRTUAL MACHINE CPU MODEL 211
14.18. MANAGING RESOURCES FOR GUEST VIRTUAL MACHINES 212
14.19. SETTING SCHEDULE PARAMETERS 212
14.20. DISPLAY OR SET BLOCK I/O PARAMETERS 213
14.21. CONFIGURING MEMORY TUNING 214
14.22. VIRTUAL NETWORKING COMMANDS 214
14.22.1. Autostarting a Virtual Network 214
14.22.2. Creating a Virtual Network from an XML File 214
14.22.3. Defining a Virtual Network from an XML File 214
14.22.4. Stopping a Virtual Network 214
14.22.5. Creating a Dump File 214
14.22.6. Editing a Virtual Network's XML Configuration File 215
14.22.7. Getting Information about a Virtual Network 215
14.22.8. Listing Information about a Virtual Network 215
14.22.9. Converting a Network UUID to Network Name 215
14.22.10. Starting a (Previously Defined) Inactive Network 215
14.22.11. Undefining the Configuration for an Inactive Network 215
14.22.12. Converting a Network Name to Network UUID 215
14.22.13. Updating an Existing Network Definition File 216

CHAPTER 15. MANAGING GUESTS WITH THE VIRTUAL MACHINE MANAGER (VIRT-MANAGER) 217
15.1. STARTING VIRT-MANAGER 217
15.2. THE VIRTUAL MACHINE MANAGER MAIN WINDOW 218
15.3. THE VIRTUAL HARDWARE DETAILS WINDOW 218
15.3.1. Attaching USB Devices to a Guest Virtual Machine 220
15.4. VIRTUAL MACHINE GRAPHICAL CONSOLE 222
15.5. ADDING A REMOTE CONNECTION 224
15.6. DISPLAYING GUEST DETAILS 225
15.7. PERFORMANCE MONITORING 232
15.8. DISPLAYING CPU USAGE FOR GUESTS 233
15.9. DISPLAYING CPU USAGE FOR HOSTS 234
15.10. DISPLAYING DISK I/O 235
15.11. DISPLAYING NETWORK I/O 236

CHAPTER 16. GUEST VIRTUAL MACHINE DISK ACCESS WITH OFFLINE TOOLS 240
16.1. INTRODUCTION 240
16.2. TERMINOLOGY 241
16.3. INSTALLATION 242
16.4. THE GUESTFISH SHELL 242
16.4.1. Viewing File Systems with guestfish 243
16.4.1.1. Manual Listing and Viewing 243
16.4.1.2. Using guestfish inspection 244
16.4.1.3. Accessing a guest virtual machine by name 245
16.4.2. Modifying Files with guestfish 245
16.4.3. Other Actions with guestfish 245

16.4.4. Shell Scripting with guestfish 246
16.4.5. Augeas and libguestfs Scripting 246
16.5. OTHER COMMANDS 247
16.6. VIRT-RESCUE: THE RESCUE SHELL 247
16.6.1. Introduction 248
16.6.2. Running virt-rescue 248
16.7. VIRT-DF: MONITORING DISK USAGE 249
16.7.1. Introduction 249
16.7.2. Running virt-df 249
16.8. VIRT-RESIZE: RESIZING GUEST VIRTUAL MACHINES OFFLINE 250
16.8.1. Introduction 250
16.8.2. Expanding a Disk Image 250
16.9. VIRT-INSPECTOR: INSPECTING GUEST VIRTUAL MACHINES 252
16.9.1. Introduction 252
16.9.2. Installation 252
16.9.3. Running virt-inspector 252
16.10. VIRT-WIN-REG: READING AND EDITING THE WINDOWS REGISTRY 254
16.10.1. Introduction 254
16.10.2. Installation 254
16.10.3. Using virt-win-reg 254
16.11. USING THE API FROM PROGRAMMING LANGUAGES 255
16.11.1. Interaction with the API through a C Program 256
16.12. VIRT-SYSPREP: RESETTING VIRTUAL MACHINE SETTINGS 259
16.13. TROUBLESHOOTING 262
16.14. WHERE TO FIND FURTHER DOCUMENTATION 263

CHAPTER 17. GRAPHICAL USER INTERFACE TOOLS FOR GUEST VIRTUAL MACHINE MANAGEMENT 264
17.1. VIRT-VIEWER 264
Syntax 264
Connecting to a guest virtual machine 264
Interface 264
Setting hotkeys 265
Kiosk mode 266
17.2. REMOTE-VIEWER 266
Syntax 266
Connecting to a guest virtual machine 266
Interface 267

CHAPTER 18. VIRTUAL NETWORKING 268
18.1. VIRTUAL NETWORK SWITCHES 268
18.2. BRIDGED MODE 268
18.3. NETWORK ADDRESS TRANSLATION MODE 269
18.3.1. DNS and DHCP 270
18.4. ROUTED MODE 271
18.5. ISOLATED MODE 271
18.6. THE DEFAULT CONFIGURATION 272
18.7. EXAMPLES OF COMMON SCENARIOS 273
18.7.1. Bridged Mode 273
18.7.2. Routed Mode 273
18.7.3. NAT Mode 274
18.7.4. Isolated Mode 274
18.8. MANAGING A VIRTUAL NETWORK 275
18.9. CREATING A VIRTUAL NETWORK 276

18.10. ATTACHING A VIRTUAL NETWORK TO A GUEST 282
18.11. ATTACHING A VIRTUAL NIC DIRECTLY TO A PHYSICAL INTERFACE 285
18.12. APPLYING NETWORK FILTERING 289
18.12.1. Introduction 289
18.12.2. Filtering Chains 290
18.12.3. Filtering Chain Priorities 291
18.12.4. Usage of Variables in Filters 292
18.12.5. Automatic IP Address Detection and DHCP Snooping 294
18.12.5.1. Introduction 294
18.12.5.2. DHCP Snooping 294
18.12.6. Reserved Variables 295
18.12.7. Element and Attribute Overview 296
18.12.8. References to Other Filters 296
18.12.9. Filter Rules 296
18.12.10. Supported Protocols 298
18.12.10.1. MAC (Ethernet) 298
18.12.10.2. VLAN (802.1Q) 299
18.12.10.3. STP (Spanning Tree Protocol) 300
18.12.10.4. ARP/RARP 301
18.12.10.5. IPv4 302
18.12.10.6. IPv6 303
18.12.10.7. TCP/UDP/SCTP 304
18.12.10.8. ICMP 305
18.12.10.9. IGMP, ESP, AH, UDPLITE, 'ALL' 306
18.12.10.10. TCP/UDP/SCTP over IPV6 307
18.12.10.11. ICMPv6 308
18.12.10.12. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv6 309
18.12.11. Advanced Filter Configuration Topics 310
18.12.11.1. Connection tracking 310
18.12.11.2. Limiting Number of Connections 311
18.12.11.3. Command line tools 312
18.12.11.4. Pre-existing network filters 312
18.12.11.5. Writing your own filters 313
18.12.11.6. Sample custom filter 315
18.12.12. Limitations 318
18.13. CREATING TUNNELS 319
18.13.1. Creating Multicast Tunnels 319
18.13.2. Creating TCP Tunnels 319
18.14. SETTING VLAN TAGS 320
18.15. APPLYING QOS TO YOUR VIRTUAL NETWORK 321

CHAPTER 19. QEMU-KVM COMMANDS, FLAGS, AND ARGUMENTS 322
19.1. INTRODUCTION 322
Whitelist Format 322
19.2. BASIC OPTIONS 322
Emulated Machine 322
Processor Type 322
Processor Topology 323
NUMA System 323
Memory Size 323
Keyboard Layout 323
Guest Name 323
Guest UUID 323

19.3. DISK OPTIONS 323
Generic Drive 323
Boot Option 324
Snapshot Mode 324
19.4. DISPLAY OPTIONS 324
Disable Graphics 325
VGA Card Emulation 325
VNC Display 325
Spice Desktop 325
19.5. NETWORK OPTIONS 326
TAP network 326
19.6. DEVICE OPTIONS 327
General Device 327
Global Device Setting 334
Character Device 335
Enable USB 335
19.7. LINUX/MULTIBOOT BOOT 335
Kernel File 335
Ram Disk 335
Command Line Parameter 335
19.8. EXPERT OPTIONS 335
KVM Virtualization 335
Disable Kernel Mode PIT Reinjection 335
No Shutdown 335
No Reboot 335
Serial Port, Monitor, QMP 336
Monitor Redirect 336
Manual CPU Start 336
RTC 336
Watchdog 336
Watchdog Reaction 336
Guest Memory Backing 336
SMBIOS Entry 336
19.9. HELP AND INFORMATION OPTIONS 336
Help 336
Version 337
Audio Help 337
19.10. MISCELLANEOUS OPTIONS 337
Migration 337
No Default Configuration 337
Device Configuration File 337
Loaded Saved State 337

CHAPTER 20. MANIPULATING THE DOMAIN XML **338**
20.1. GENERAL INFORMATION AND METADATA 338
20.2. OPERATING SYSTEM BOOTING 339
20.2.1. BIOS Boot loader 339
20.2.2. Host Physical Machine Boot Loader 341
20.2.3. Direct kernel boot 341
20.3. SMBIOS SYSTEM INFORMATION 342
20.4. CPU ALLOCATION 343
20.5. CPU TUNING 344
20.6. MEMORY BACKING 346

20.7. MEMORY TUNING 346
20.8. NUMA NODE TUNING 347
20.9. BLOCK I/O TUNING 348
20.10. RESOURCE PARTITIONING 349
20.11. CPU MODEL AND TOPOLOGY 349
20.11.1. Guest virtual machine NUMA topology 354
20.12. EVENTS CONFIGURATION 354
20.13. POWER MANAGEMENT 356
20.14. HYPERVISOR FEATURES 357
20.15. TIMEKEEPING 358
20.16. DEVICES 360
20.16.1. Hard Drives, Floppy Disks, CDROMs 361
20.16.1.1. Disk element 362
20.16.1.2. Source element 363
20.16.1.3. Mirror element 363
20.16.1.4. Target element 363
20.16.1.5. iotune 363
20.16.1.6. driver 364
20.16.1.7. Additional device elements 365
20.16.2. Filesystems 366
20.16.3. Device Addresses 368
20.16.4. Controllers 369
20.16.5. Device Leases 370
20.16.6. Host Physical Machine Device Assignment 371
20.16.6.1. USB / PCI Devices 371
20.16.6.2. Block / character devices 374
20.16.7. Redirected Devices 375
20.16.8. Smartcard Devices 376
20.16.9. Network Interfaces 378
20.16.9.1. Virtual networks 378
20.16.9.2. Bridge to LAN 380
20.16.9.3. Setting a port masquerading range 381
20.16.9.4. User-space SLIRP stack 381
20.16.9.5. Generic Ethernet connection 382
20.16.9.6. Direct attachment to physical interfaces 382
20.16.9.7. PCI passthrough 385
20.16.9.8. Multicast tunnel 386
20.16.9.9. TCP tunnel 386
20.16.9.10. Setting NIC driver-specific options 387
20.16.9.11. Overriding the target element 388
20.16.9.12. Specifying boot order 389
20.16.9.13. Interface ROM BIOS configuration 389
20.16.9.14. Quality of service 390
20.16.9.15. Setting VLAN tag (on supported network types only) 391
20.16.9.16. Modifying virtual link state 391
20.16.10. Input Devices 392
20.16.11. Hub Devices 392
20.16.12. Graphical framebuffers 393
20.16.13. Video Devices 396
20.16.14. Consoles, Serial, Parallel, and Channel Devices 397
20.16.15. Guest Virtual Machine Interfaces 398
20.16.16. Channel 400
20.16.17. Host Physical Machine Interface 401

20.17. SOUND DEVICES 405
20.18. WATCHDOG DEVICE 406
20.19. MEMORY BALLOON DEVICE 407
20.20. SECURITY LABEL 407
20.21. EXAMPLE DOMAIN XML CONFIGURATION 409

CHAPTER 21. TROUBLESHOOTING **411**
21.1. DEBUGGING AND TROUBLESHOOTING TOOLS 411
21.2. PREPARING FOR DISASTER RECOVERY 412
21.3. CREATING VIRSH DUMP FILES 413
21.4. KVM_STAT 414
21.5. GUEST VIRTUAL MACHINE FAILS TO SHUTDOWN 417
21.6. TROUBLESHOOTING WITH SERIAL CONSOLES 418
21.7. VIRTUALIZATION LOG FILES 419
21.8. LOOP DEVICE ERRORS 419
21.9. LIVE MIGRATION ERRORS 419
21.10. ENABLING INTEL VT-X AND AMD-V VIRTUALIZATION HARDWARE EXTENSIONS IN BIOS 419
21.11. KVM NETWORKING PERFORMANCE 420
21.12. WORKAROUND FOR CREATING EXTERNAL SNAPSHOTS WITH LIBVIRT 422
21.13. MISSING CHARACTERS ON GUEST CONSOLE WITH JAPANESE KEYBOARD 422
21.14. VERIFYING VIRTUALIZATION EXTENSIONS 423

APPENDIX A. THE VIRTUAL HOST METRICS DAEMON (VHOSTMD) **424**

APPENDIX B. ADDITIONAL RESOURCES **425**
B.1. ONLINE RESOURCES 425
B.2. INSTALLED DOCUMENTATION 425

APPENDIX C. REVISION HISTORY **426**

CHAPTER 1. SERVER BEST PRACTICES

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

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

```
# setenforce 1
```

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

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

- Avoid running any unessential applications on your host. Running applications on the host may impact virtual machine performance and can affect server stability. Any application which may crash the server will also cause all virtual machines on the server to go down.

- Use a central location for virtual machine installations and images. Virtual machine images should be stored under **/var/lib/libvirt/images/**. If you are using a different directory for your virtual machine images make sure you add the directory to your SELinux policy and relabel it before starting the installation. Use of shareable, network storage in a central location is highly recommended.

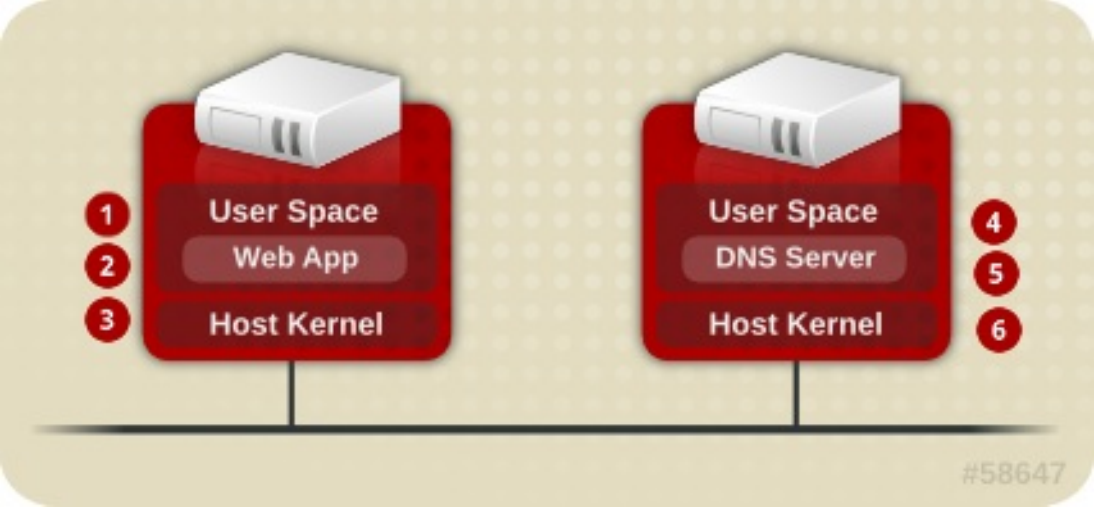
CHAPTER 2. SVIRT

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

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

Non-virtualized Environments

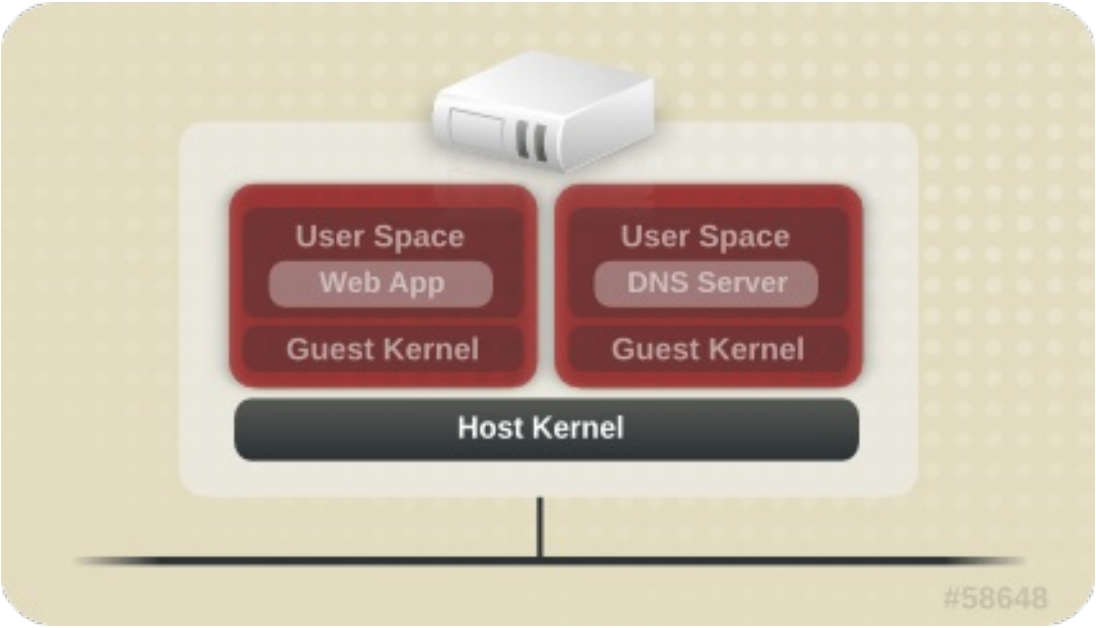
In a non-virtualized environment, host physical machines are separated from each other physically and each host physical machine has a self-contained environment, consisting of services such as a web server, or a DNS server. These services communicate directly to their own user space, host physical machine's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



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

Virtualized Environments

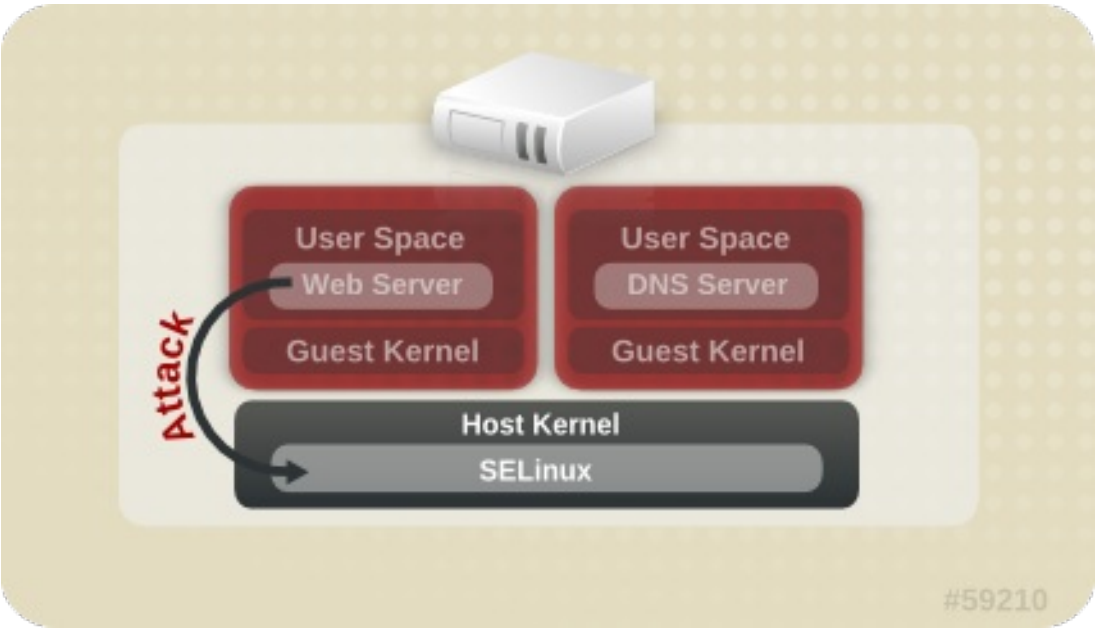
In a virtualized environment, several virtual operating systems can run on a single kernel residing on a host physical machine. The following image represents a virtualized environment:



2.1. SECURITY AND VIRTUALIZATION

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

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



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

2.2. SVIRT LABELING

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

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

```
# ps -eZ | grep qemu

system_u:system_r:svirt_t:s0:c87,c520 27950 ?  00:00:17 qemu-kvm
```

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

```
# ls -lZ /var/lib/libvirt/images/*

  system_u:object_r:svirt_image_t:s0:c87,c520   image1
```

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

Table 2.1. sVirt context labels

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

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

CHAPTER 3. CLONING VIRTUAL MACHINES

There are two types of guest virtual machine instances used in creating guest copies:

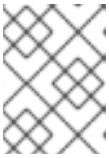
- *Clones* are instances of a single virtual machine. Clones can be used to set up a network of identical virtual machines, and they can also be distributed to other destinations.
- *Templates* are instances of a virtual machine that are designed to be used as a source for cloning. You can create multiple clones from a template and make minor modifications to each clone. This is useful in seeing the effects of these changes on the system.

Both clones and templates are virtual machine instances. The difference between them is in how they are used.

For the created clone to work properly, information and configurations unique to the virtual machine that is being cloned usually has to be removed before cloning. The information that needs to be removed differs, based on how the clones will be used.

The information and configurations to be removed may be on any of the following levels:

- *Platform level* information and configurations include anything assigned to the virtual machine by the virtualization solution. Examples include the number of Network Interface Cards (NICs) and their MAC addresses.
- *Guest operating system level* information and configurations include anything configured within the virtual machine. Examples include SSH keys.
- *Application level* information and configurations include anything configured by an application installed on the virtual machine. Examples include activation codes and registration information.



NOTE

This chapter does not include information about removing the application level, because the information and approach is specific to each application.

As a result, some of the information and configurations must be removed from within the virtual machine, while other information and configurations must be removed from the virtual machine using the virtualization environment (for example, Virtual Machine Manager or VMware).

3.1. PREPARING VIRTUAL MACHINES FOR CLONING

Before cloning a virtual machine, it must be prepared by running the virt-sysprep utility on its disk image, or by using the following steps:

Procedure 3.1. Preparing a virtual machine for cloning

1. **Setup the virtual machine**
   a. Build the virtual machine that is to be used for the clone or template.
      - Install any software needed on the clone.
      - Configure any non-unique settings for the operating system.
      - Configure any non-unique application settings.

2. **Remove the network configuration**

   a. Remove any persistent udev rules using the following command:

   ```
   # rm -f /etc/udev/rules.d/70-persistent-net.rules
   ```

   NOTE

   If udev rules are not removed, the name of the first NIC may be eth1 instead of eth0.

   b. Remove unique network details from ifcfg scripts by making the following edits to **/etc/sysconfig/network-scripts/ifcfg-eth[x]**:

      i. Remove the HWADDR and Static lines

      NOTE

      If the HWADDR does not match the new guest's MAC address, the ifcfg will be ignored. Therefore, it is important to remove the HWADDR from the file.

      ```
      DEVICE=eth[x]
      BOOTPROTO=none
      ONBOOT=yes
      #NETWORK=10.0.1.0       <- REMOVE
      #NETMASK=255.255.255.0  <- REMOVE
      #IPADDR=10.0.1.20       <- REMOVE
      #HWADDR=xx:xx:xx:xx:xx  <- REMOVE
      #USERCTL=no             <- REMOVE
      # Remove any other *unique* or non-desired settings, such as UUID.
      ```

      ii. Ensure that a DHCP configuration remains that does not include a HWADDR or any unique information.

      ```
      DEVICE=eth[x]
      BOOTPROTO=dhcp
      ONBOOT=yes
      ```

      iii. Ensure that the file includes the following lines:

      ```
      DEVICE=eth[x]
      ONBOOT=yes
      ```

   c. If the following files exist, ensure that they contain the same content:

      - **/etc/sysconfig/networking/devices/ifcfg-eth[x]**
      - **/etc/sysconfig/networking/profiles/default/ifcfg-eth[x]**

NOTE

If NetworkManager or any special settings were used with the virtual machine, ensure that any additional unique information is removed from the ifcfg scripts.

3. **Remove registration details**

   a. Remove registration details using one of the following:

      - For Red Hat Network (RHN) registered guest virtual machines, run the following command:

        ```
        # rm /etc/sysconfig/rhn/systemid
        ```

      - For Red Hat Subscription Manager (RHSM) registered guest virtual machines:

        - If the original virtual machine will not be used, run the following commands:

          ```
          # subscription-manager unsubscribe --all
          # subscription-manager unregister
          # subscription-manager clean
          ```

        - If the original virtual machine will be used, run only the following command:

          ```
          # subscription-manager clean
          ```

NOTE

The original RHSM profile remains in the portal.

4. **Removing other unique details**

   a. Remove any sshd public/private key pairs using the following command:

      ```
      # rm -rf /etc/ssh/ssh_host_*
      ```

NOTE

Removing ssh keys prevents problems with ssh clients not trusting these hosts.

   b. Remove any other application-specific identifiers or configurations that may cause conflicts if running on multiple machines.

5. **Configure the virtual machine to run configuration wizards on the next boot**

   a. Configure the virtual machine to run the relevant configuration wizards the next time it is booted by doing one of the following:

      - For Red Hat Enterprise Linux 6 and below, create an empty file on the root file system called .unconfigured using the following command:

```
# touch /.unconfigured
```

- For Red Hat Enterprise Linux 7, enable the first boot and initial-setup wizards by running the following commands:

```
# sed -ie 's/RUN_FIRSTBOOT=NO/RUN_FIRSTBOOT=YES/'
/etc/sysconfig/firstboot
# systemctl enable firstboot-graphical
# systemctl enable initial-setup-graphical
```

NOTE

The wizards that run on the next boot depend on the configurations that have been removed from the virtual machine. In addition, on the first boot of the clone, it is recommended that you change the host name.

3.2. CLONING A VIRTUAL MACHINE

Before proceeding with cloning, shut down the virtual machine. You can clone the virtual machine using **virt-clone** or **virt-manager**.

3.2.1. Cloning Guests with virt-clone

You can use **virt-clone** to clone virtual machines from the command line.

Note that you need root privileges for **virt-clone** to complete successfully.

The **virt-clone** command provides a number of options that can be passed on the command line. These include general options, storage configuration options, networking configuration options, and miscellaneous options. Only the **--original** is required. To see a complete list of options, enter the following command:

```
# virt-clone --help
```

The **virt-clone** man page also documents each command option, important variables, and examples.

The following example shows how to clone a guest virtual machine called "demo" on the default connection, automatically generating a new name and disk clone path.

Example 3.1. Using virt-clone to clone a guest

```
# virt-clone --original demo --auto-clone
```

The following example shows how to clone a QEMU guest virtual machine called "demo" with multiple disks.

Example 3.2. Using virt-clone to clone a guest

```
# virt-clone --connect qemu:///system --original demo --name newdemo --file
/var/lib/xen/images/newdemo.img --file /var/lib/xen/images/newdata.img
```

■

3.2.2. Cloning Guests with virt-manager

This procedure describes cloning a guest virtual machine using the **virt-manager** utility.

Procedure 3.2. Cloning a Virtual Machine with virt-manager

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

   Select the guest virtual machine you want to clone from the list of guest virtual machines in **Virtual Machine Manager**.

   Right-click the guest virtual machine you want to clone and select **Clone**. The Clone Virtual Machine window opens.

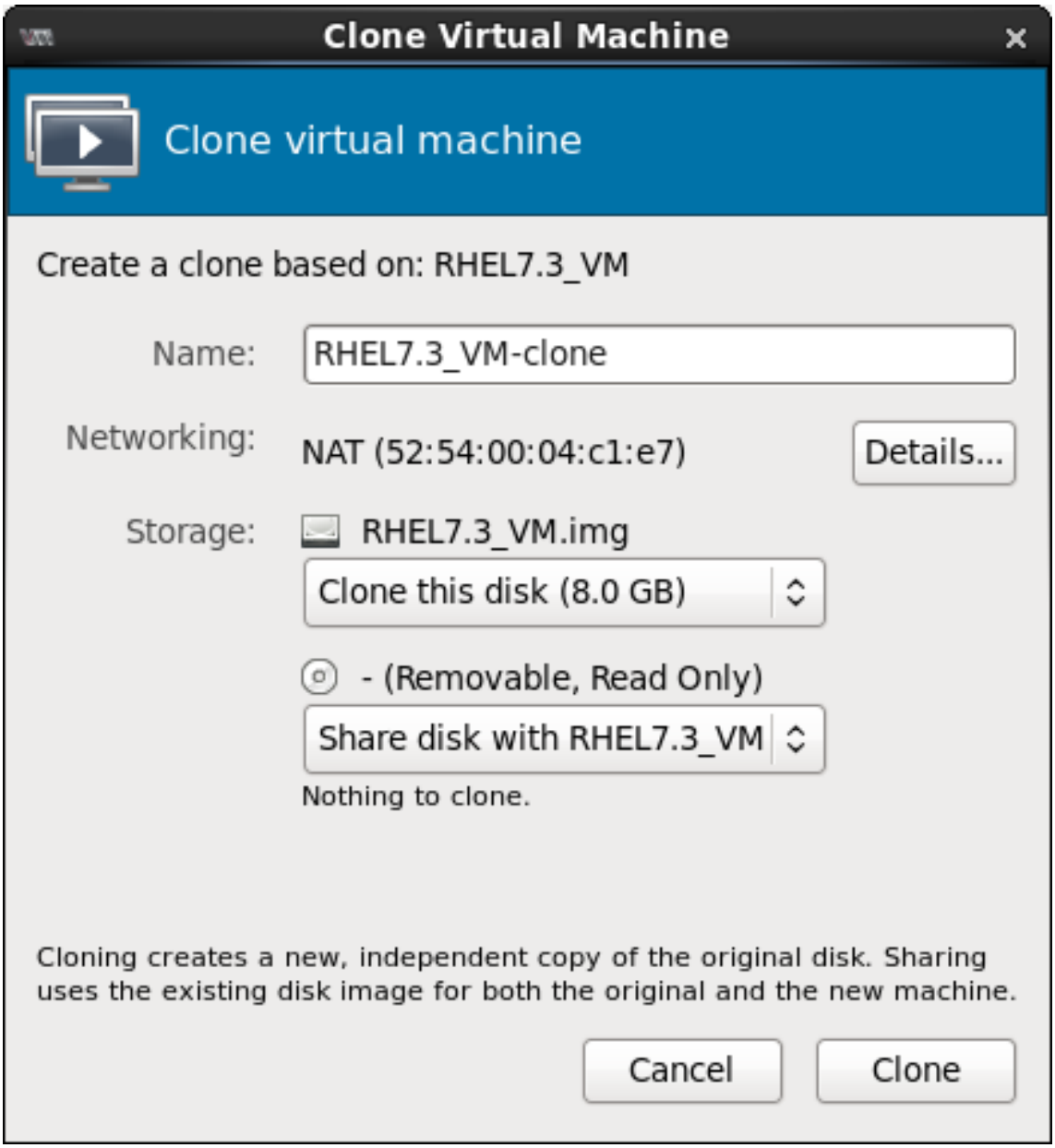


   Figure 3.1. Clone Virtual Machine window

2. **Configure the clone**

- To change the name of the clone, enter a new name for the clone.
- To change the networking configuration, click **Details**.

  Enter a new MAC address for the clone.

  Click **OK**.

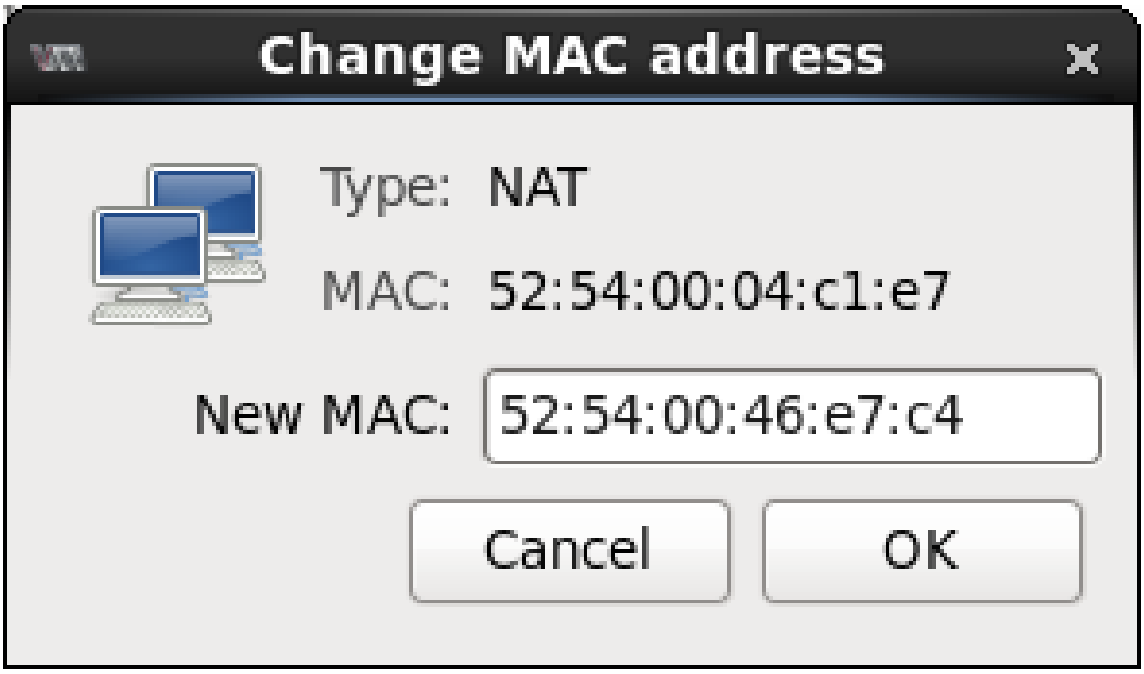


Figure 3.2. Change MAC Address window

- For each disk in the cloned guest virtual machine, select one of the following options:
  - **Clone this disk** - The disk will be cloned for the cloned guest virtual machine
  - **Share disk with *guest virtual machine name*** - The disk will be shared by the guest virtual machine that will be cloned and its clone
  - **Details** - Opens the Change storage path window, which enables selecting a new path for the disk

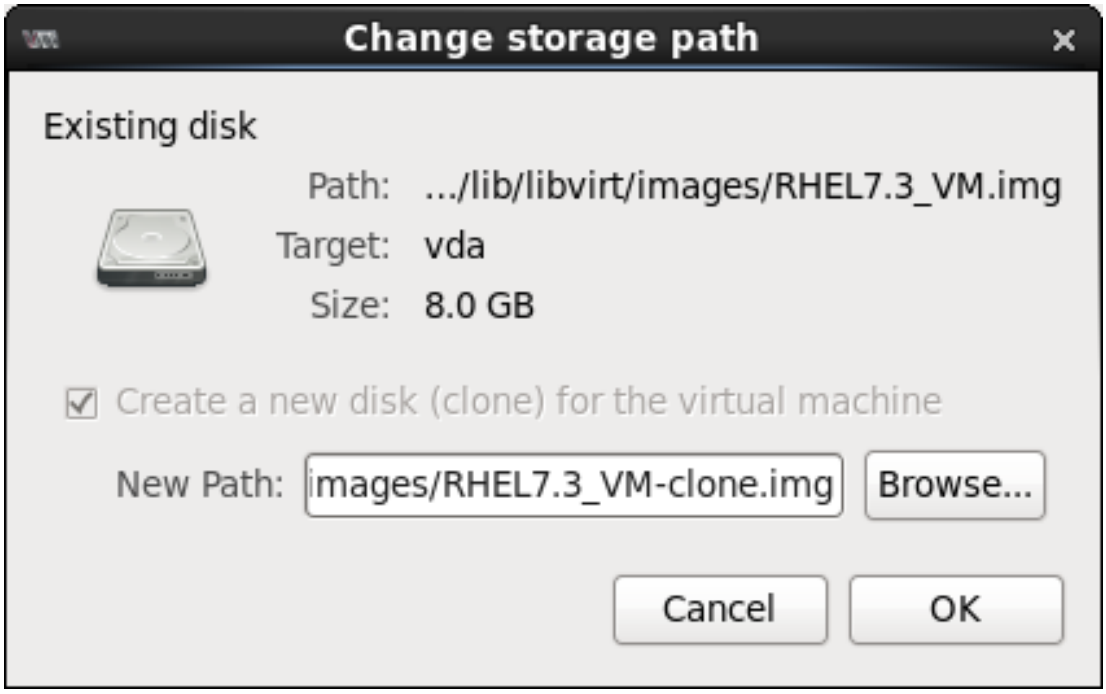


Figure 3.3. Change **storage path** window

3. **Clone the guest virtual machine**
   Click **Clone**.

CHAPTER 4. KVM LIVE MIGRATION

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

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

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

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

Migrations can be performed live or not.

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

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

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

4.1. LIVE MIGRATION REQUIREMENTS

Migrating guest virtual machines requires the following:

Migration requirements

- A guest virtual machine installed on shared storage using one of the following protocols:
  - Fibre Channel-based LUNs
  - iSCSI
  - FCoE
  - NFS
  - GFS2
  - SCSI RDMA protocols (SCSI RCP): the block export protocol used in Infiniband and 10GbE iWARP adapters
- The migration platforms and versions should be checked against table Table 4.1, "Live Migration Compatibility". It should also be noted that Red Hat Enterprise Linux 6 supports live migration of guest virtual machines using raw and qcow2 images on shared storage.
- Both systems must have the appropriate TCP/IP ports open. In cases where a firewall is used, refer to the *Red Hat Enterprise Linux Virtualization Security Guide* which can be found at https://access.redhat.com/site/documentation/ for detailed port information.
- A separate system exporting the shared storage medium. Storage should not reside on either of the two host physical machines being used for migration.
- Shared storage must mount at the same location on source and destination systems. The mounted directory names must be identical. Although it is possible to keep the images using different paths, it is not recommended. Note that, if you are intending to use virt-manager to perform the migration, the path names must be identical. If however you intend to use virsh to perform the migration, different network configurations and mount directories can be used with the help of **--xml** option or pre-hooks when doing migrations. Even without shared storage, migration can still succeed with the option **--copy-storage-all** (deprecated). For more information on **prehooks**, refer to libvirt.org, and for more information on the XML option, refer to Chapter 20, *Manipulating the Domain XML*.
- When migration is attempted on an existing guest virtual machine in a public bridge+tap network, the source and destination host physical machines must be located in the same network. Otherwise, the guest virtual machine network will not operate after migration.
- In Red Hat Enterprise Linux 5 and 6, the default cache mode of KVM guest virtual machines is set to ***none***, which prevents inconsistent disk states. Setting the cache option to ***none*** (using **virsh attach-disk cache none**, for example), causes all of the guest virtual machine's files to be opened using the ***O_DIRECT*** flag (when calling the **open** syscall), thus bypassing the host physical machine's cache, and only providing caching on the guest virtual machine. Setting the cache mode to ***none*** prevents any potential inconsistency problems, and when used makes it possible to live-migrate virtual machines. For information on setting cache to ***none***, refer to Section 13.3, "Adding Storage Devices to Guests".

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

Procedure 4.1. Configuring libvirtd.conf

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

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

2. Change the parameters as needed and save the file.

3. Restart the **libvirtd** service:

```
# service libvirtd restart
```

4.2. LIVE MIGRATION AND RED HAT ENTERPRISE LINUX VERSION COMPATIBILITY

Live Migration is supported as shown in table Table 4.1, "Live Migration Compatibility":

Table 4.1. Live Migration Compatibility

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

Troubleshooting problems with migration

- **Issues with SPICE** – It has been found that SPICE has an incompatible change when migrating from Red Hat Enterprise Linux 6.0 → 6.1. In such cases, the client may disconnect and then reconnect, causing a temporary loss of audio and video. This is only temporary and all services will resume.

- **Issues with USB** – Red Hat Enterprise Linux 6.2 added USB functionality which included migration support, but not without certain caveats which reset USB devices and caused any

application running over the device to abort. This problem was fixed in Red Hat Enterprise Linux 6.4, and should not occur in future versions. To prevent this from happening in a version prior to 6.4, abstain from migrating while USB devices are in use.

- **Issues with the migration protocol** – If backward migration ends with "unknown section error", repeating the migration process can repair the issue as it may be a transient error. If not, please report the problem.

Configuring Network Storage

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

Alternatively, use the NFS example in Section 4.3, "Shared Storage Example: NFS for a Simple Migration"

4.3. SHARED STORAGE EXAMPLE: NFS FOR A SIMPLE MIGRATION

IMPORTANT

This example uses NFS to share guest virtual machine images with other KVM host physical machines. Although not practical for large installations, it is presented to demonstrate migration techniques only. Do not use this example for migrating or running more than a few guest virtual machines. In addition, it is required that the ***sync*** parameter is enabled. This is required for proper export of the NFS storage. In addition, it is strongly recommended that the NFS is mounted on source host physical machine, and the guest virtual machine's image needs to be created on the NFS mounted directory located on source host physical machine. It should also be noted that NFS file locking must **not** be used as it is not supported in KVM.

iSCSI storage is a better choice for large deployments. Refer to Section 12.5, "iSCSI-based Storage Pools" for configuration details.

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

1. **Create a directory for the disk images**
   This shared directory will contain the disk images for the guest virtual machines. To do this create a directory in a location different from **/var/lib/libvirt/images**. For example:

   ```
   # mkdir /var/lib/libvirt-img/images
   ```

2. **Add the new directory path to the NFS configuration file**
   The NFS configuration file is a text file located in **/etc/exports**. Open the file and edit it adding the path to the new file you created in step 1.

   ```
   # echo "/var/lib/libvirt-img/images" >> /etc/exports/[NFS-Config-FILENAME.txt]
   ```

3. **Start NFS**

   a. Make sure that the ports for NFS in **iptables** (2049, for example) are opened and add NFS to the **/etc/hosts.allow** file.

   b. Start the NFS service:

```
# service nfs start
```

4. **Mount the shared storage on both the source and the destination**
   Mount the **/var/lib/libvirt/images** directory on both the source and destination system, running the following command twice. Once on the source system and again on the destination system.

```
# mount source_host:/var/lib/libvirt-img/images /var/lib/libvirt/images
```

WARNING

Make sure that the directories you create in this procedure is compliant with the requirements as outlined in Section 4.1, "Live Migration Requirements". In addition, the directory may need to be labeled with the correct SELinux label. For more information consult the NFS chapter in the Red Hat Enterprise Linux Storage Administration Guide.

4.4. LIVE KVM MIGRATION WITH VIRSH

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

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

Note that the **--live** option may be eliminated when live migration is not desired. Additional options are listed in Section 4.4.2, "Additional Options for the virsh migrate Command".

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

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

NOTE

The ***DestinationURL*** parameter for normal migration and peer-to-peer migration has different semantics:

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

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

IMPORTANT

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

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

Example: Live Migration with virsh

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

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

1. **Verify the guest virtual machine is running**
   From the source system, **host1.example.com**, verify **guest1-rhel6-64** is running:

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

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

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

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

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

   NOTE

   During the migration, the completion percentage indicator number is likely to decrease multiple times before the process finishes. This is caused by a recalculation of the overall progress, as source memory pages that are changed after the migration starts need to be be copied again. Therefore, this behavior is expected and does not indicate any problems with the migration.

4. **Verify the guest virtual machine has arrived at the destination host**
   From the destination system, **host2.example.com**, verify **guest1-rhel6-64** is running:

   ```
   [root@host2 ~]# virsh list
   Id Name                 State
   ```

```
----------------------------------
 10 guest1-rhel6-64     running
```

The live migration is now complete.

NOTE

libvirt supports a variety of networking methods including TLS/SSL, UNIX sockets, SSH, and unencrypted TCP. Refer to Chapter 5, *Remote Management of Guests* for more information on using other methods.

NOTE

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

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

4.4.1. Additional Tips for Migration with virsh

It is possible to perform multiple, concurrent live migrations where each migration runs in a separate command shell. However, this should be done with caution and should involve careful calculations as each migration instance uses one MAX_CLIENT from each side (source and target). As the default setting is 20, there is enough to run 10 instances without changing the settings. Should you need to change the settings, refer to the procedure Procedure 4.1, “Configuring libvirtd.conf”.

1. Open the libvirtd.conf file as described in Procedure 4.1, “Configuring libvirtd.conf”.

2. Look for the Processing controls section.

```
#################################################################
#
# Processing controls
#

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


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


# The number of priority workers. If all workers from above
# pool will stuck, some calls marked as high priority
```

```
# (notably domainDestroy) can be executed in this pool.
#prio_workers = 5

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

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

#################################################################
```

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

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

4. Save the file and restart the service.

> **NOTE**
>
> There may be cases where a migration connection drops because there are too many ssh sessions that have been started, but not yet authenticated. By default, **sshd** allows only 10 sessions to be in a "pre-authenticated state" at any time. This setting is controlled by the **MaxStartups** parameter in the sshd configuration file (located here: **/etc/ssh/sshd_config**), which may require some adjustment. Adjusting this parameter should be done with caution as the limitation is put in place to prevent DoS attacks (and over-use of resources in general). Setting this value too high will negate its purpose. To change this parameter, edit the file **/etc/ssh/sshd_config**, remove the **#** from the beginning of the **MaxStartups** line, and change the **10** (default value) to a higher number. Remember to save the file and restart the **sshd** service. For more information, refer to the **sshd_config** man page.

4.4.2. Additional Options for the virsh migrate Command

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

- **--direct** - used for direct migration
- **--p2p** - used for peer-to-peer migration
- **--tunnelled** - used for tunneled migration
- **--persistent** - leaves the domain in a persistent state on the destination host physical machine
- **--undefinesource** - removes the guest virtual machine on the source host physical machine
- **--suspend** - leaves the domain in a paused state on the destination host physical machine
- **--change-protection** - enforces that no incompatible configuration changes will be made to the domain while the migration is underway; this option is implicitly enabled when supported by the hypervisor, but can be explicitly used to reject the migration if the hypervisor lacks change protection support.
- **--unsafe** - forces the migration to occur, ignoring all safety procedures.
- **--verbose** - displays the progress of migration as it is occurring
- **--abort-on-error** - cancels the migration if a soft error (such as an I/O error) happens during the migration process.
- **--migrateuri** - the migration URI which is usually omitted.
- **--domain** *[string]*- domain name, id or uuid
- **--desturi** *[string]*- connection URI of the destination host physical machine as seen from the client(normal migration) or source(p2p migration)
- **--migrateuri** - migration URI, usually can be omitted
- **--timeout** *[seconds]*- forces a guest virtual machine to suspend when the live migration counter exceeds N seconds. It can only be used with a live migration. Once the timeout is initiated, the migration continues on the suspended guest virtual machine.
- **--dname** *[string]* - changes the name of the guest virtual machine to a new name during migration (if supported)
- **--xml** - the filename indicated can be used to supply an alternative XML file for use on the destination to supply a larger set of changes to any host-specific portions of the domain XML, such as accounting for naming differences between source and destination in accessing underlying storage. This option is usually omitted.

Refer to the virsh man page for more information.

4.5. MIGRATING WITH VIRT-MANAGER

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

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

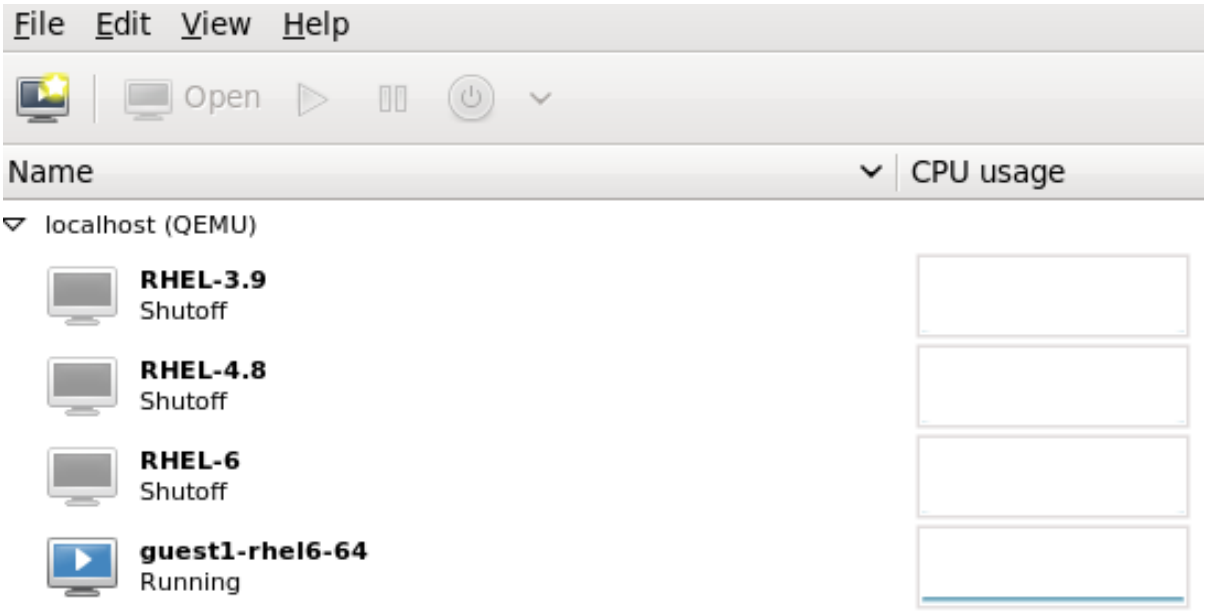


Figure 4.1. Virt-Manager main menu

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

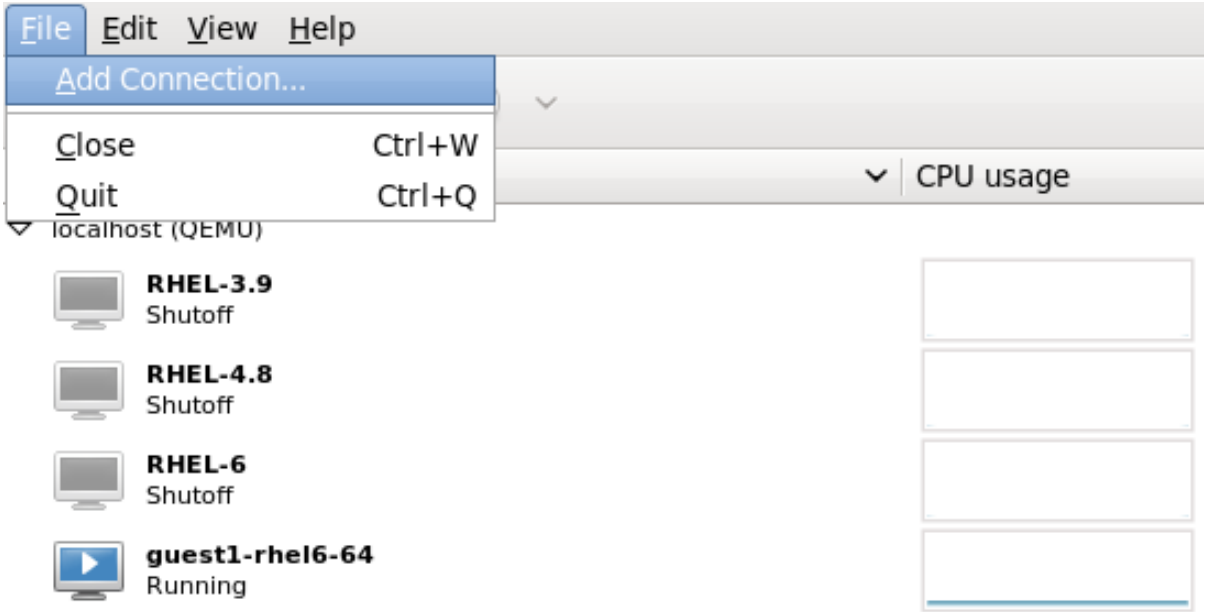


Figure 4.2. Open Add Connection window

3. **Add connection**
   The **Add Connection** window appears.

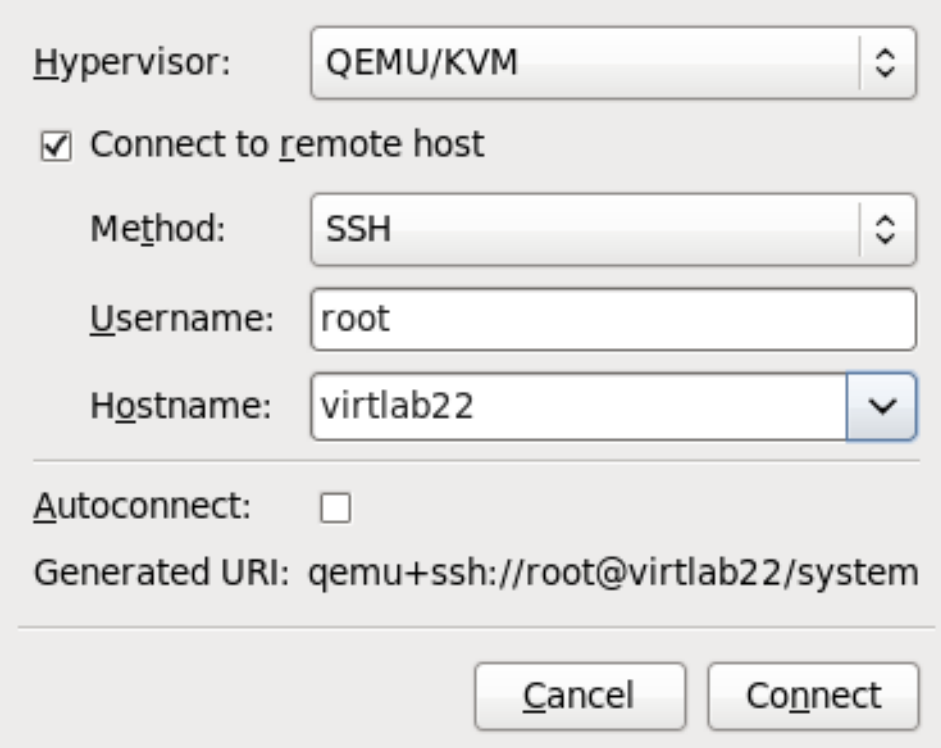


Figure 4.3. Adding a connection to the target host physical machine

Enter the following details:

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

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

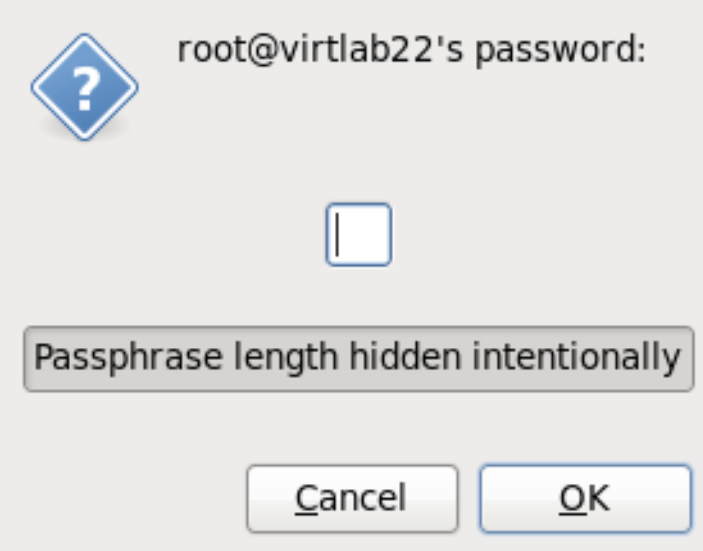


Figure 4.4. Enter password

4. **Migrate guest virtual machines**

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

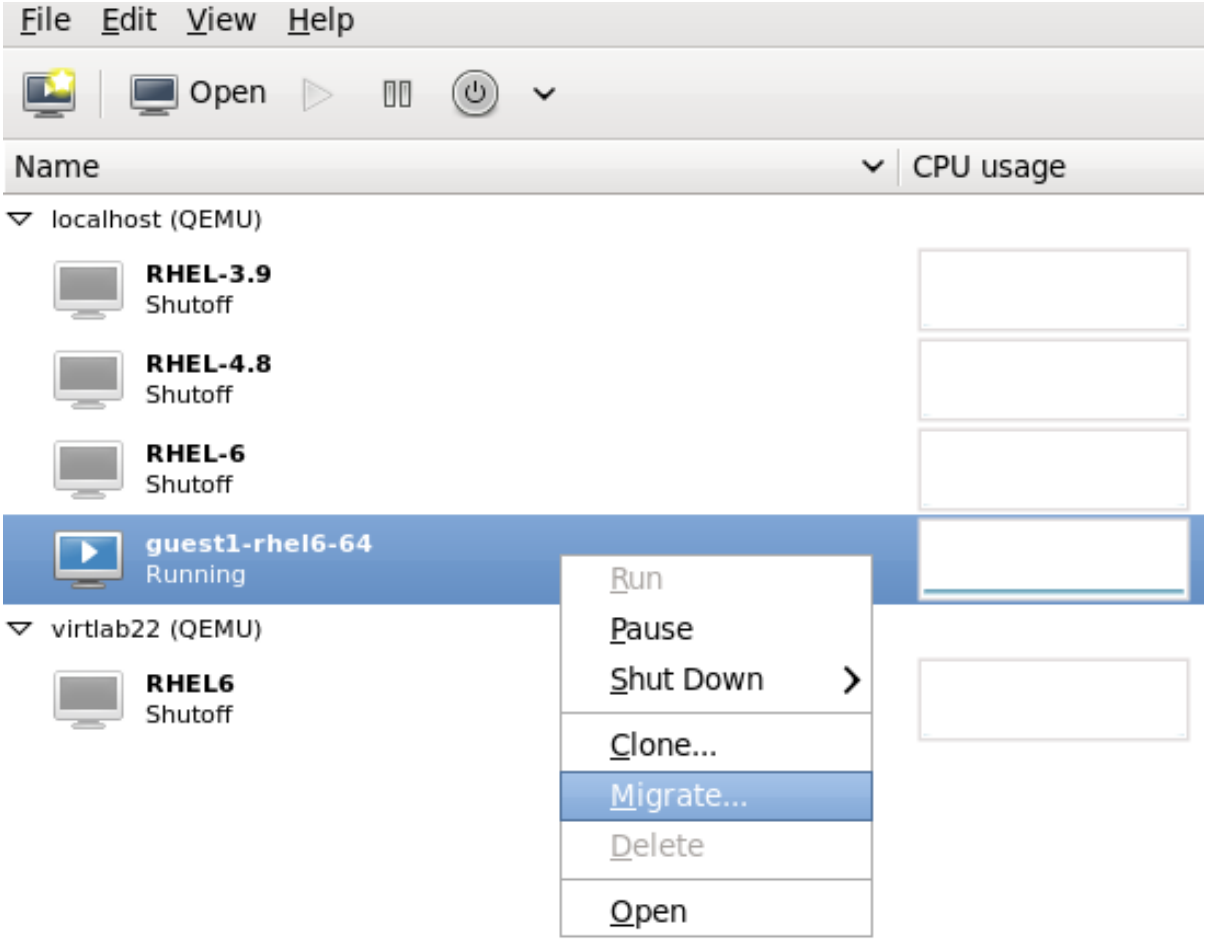


Figure 4.5. Choosing the guest to be migrated

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

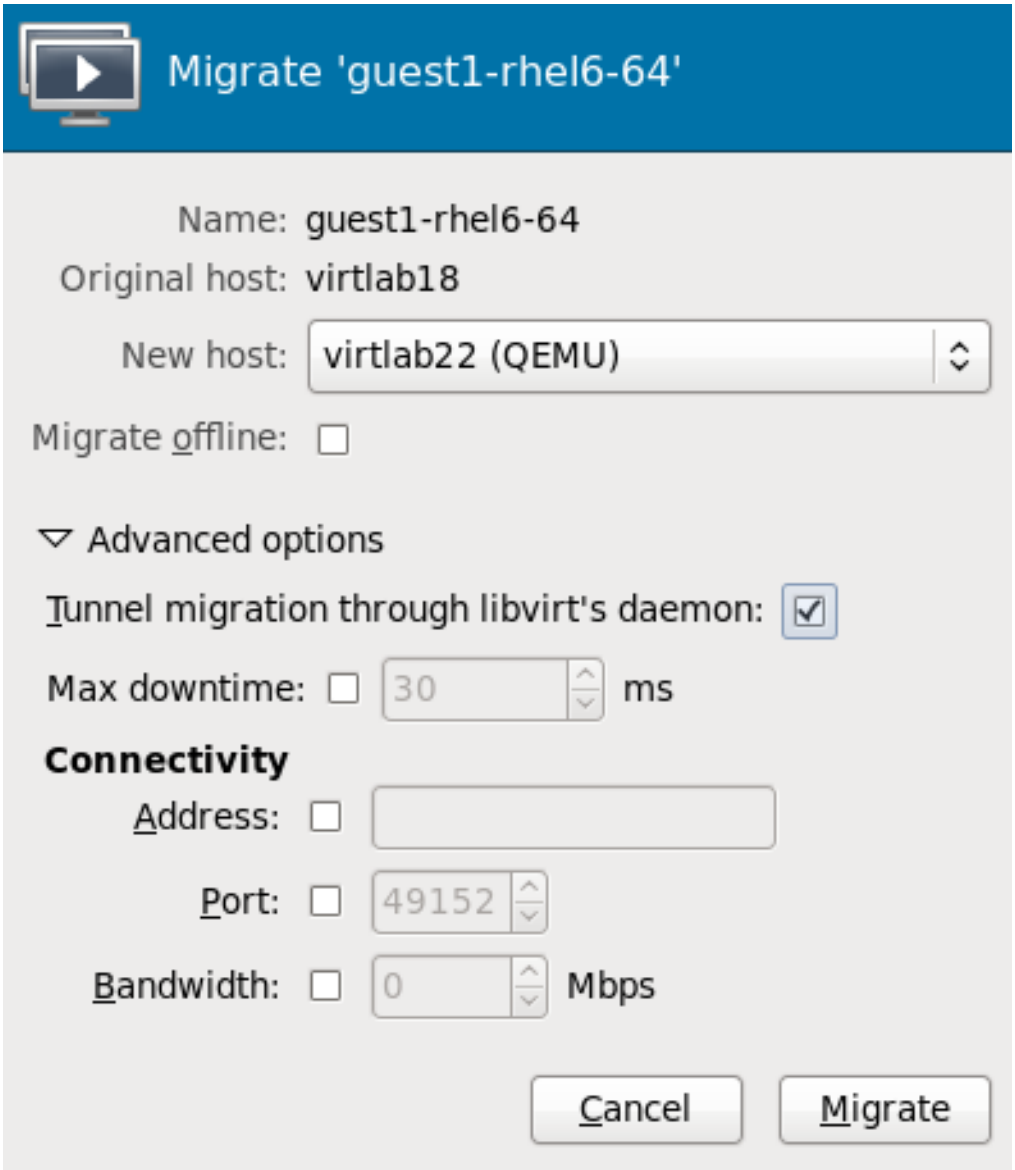


Figure 4.6. Choosing the destination host physical machine and starting the migration process

A progress window will appear.

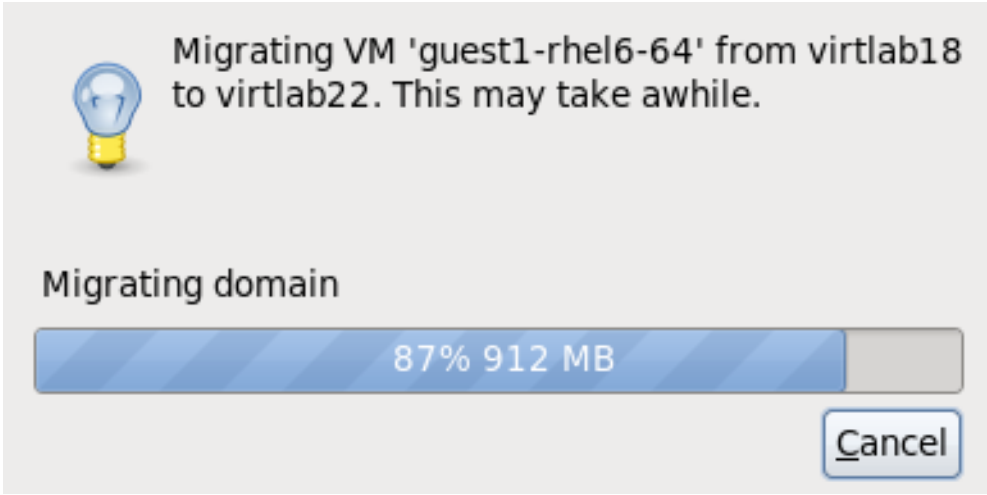


Figure 4.7. Progress window

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

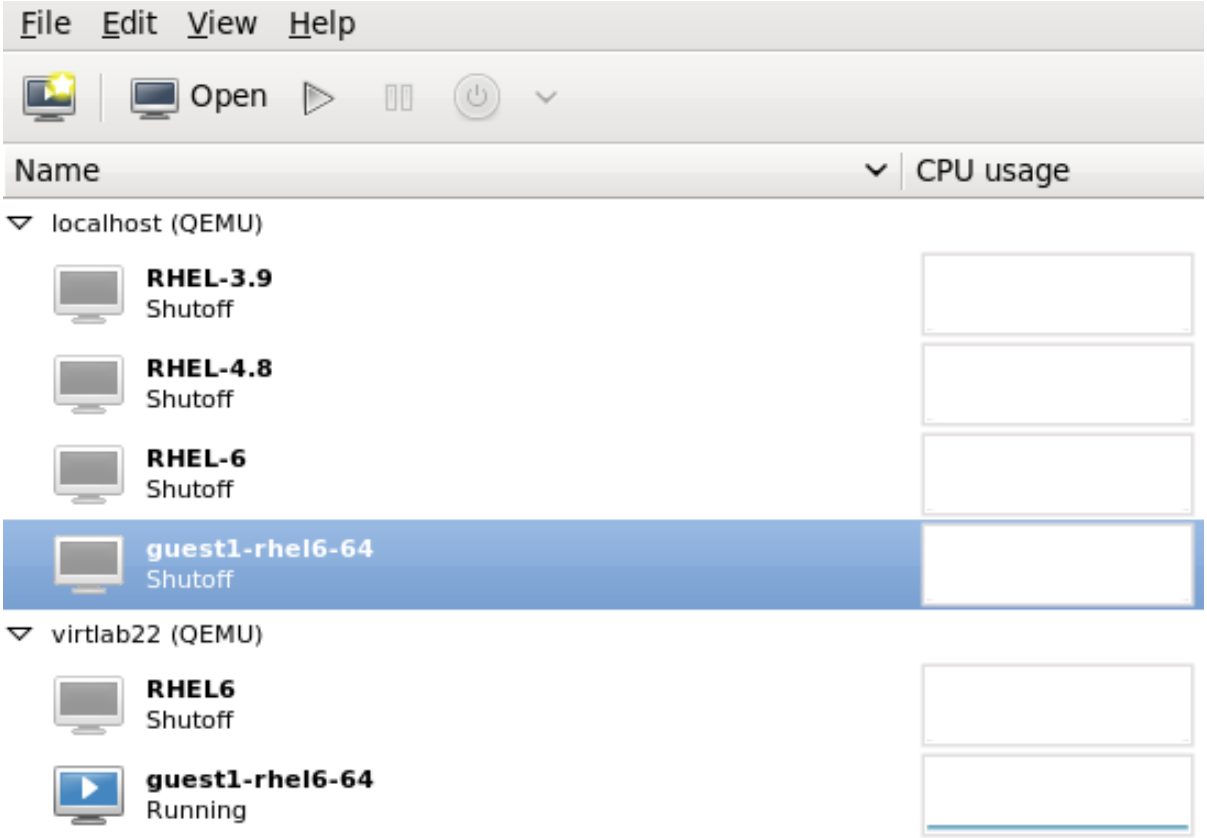


Figure 4.8. Migrated guest virtual machine running in the destination host physical machine

5. **Optional - View the storage details for the host physical machine**
   In the **Edit** menu, click **Connection Details**, the Connection Details window appears.

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

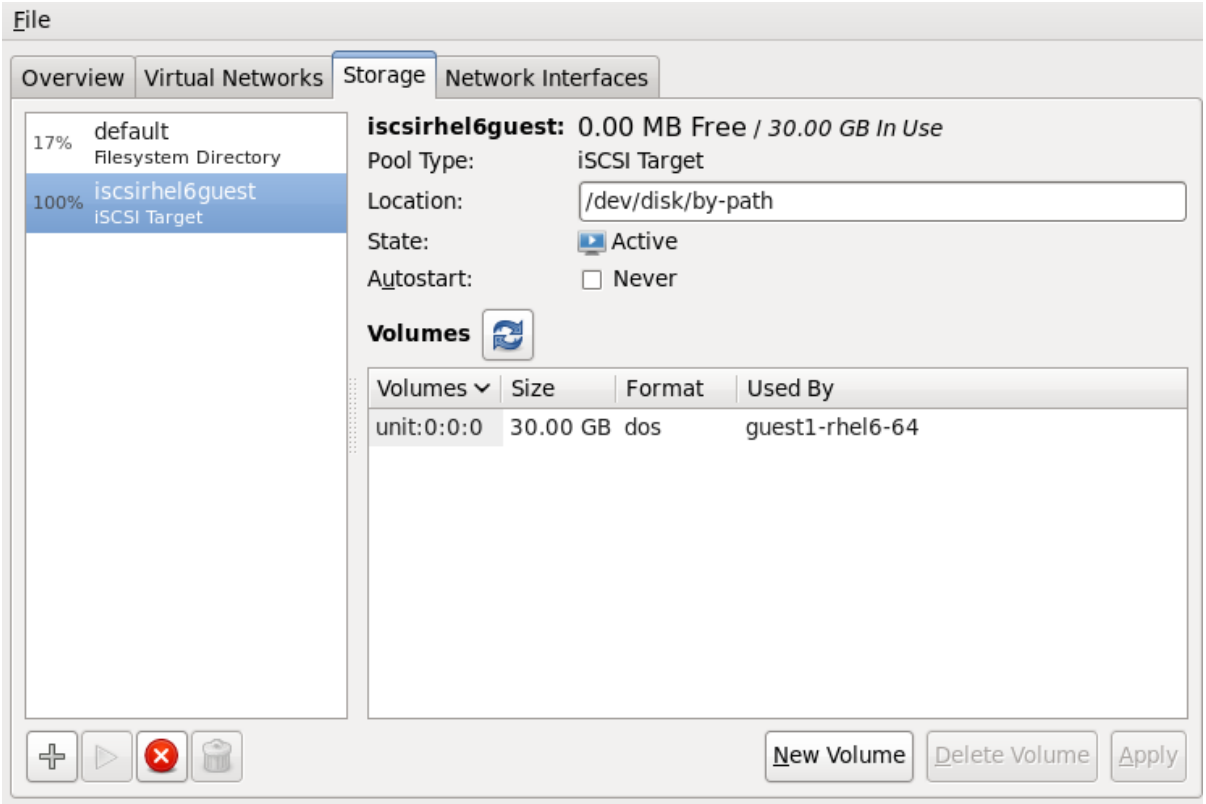


Figure 4.9. Storage details

This host was defined by the following XML configuration:

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

Figure 4.10. XML configuration for the destination host physical machine

CHAPTER 5. REMOTE MANAGEMENT OF GUESTS

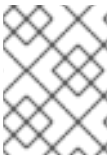
This section explains how to remotely manage your guests using **ssh** or TLS and SSL. More information on SSH can be found in the *Red Hat Enterprise Linux Deployment Guide* .

5.1. REMOTE MANAGEMENT WITH SSH

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

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

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



NOTE

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

The following packages are required for ssh access:

- openssh
- openssh-askpass
- openssh-clients
- openssh-server

Configuring Password-less or Password-managed SSH Access for virt-manager

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

IMPORTANT

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

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

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

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

   ```
   $ su -
   ```

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

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

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

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

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

   Repeat for other systems, as required.

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

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

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

   The SSH key is added to the remote system.

The libvirt Daemon (libvirtd)

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

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

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

Accessing Remote Hosts with virt-manager

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

1. Start virt-manager.
2. Open the **File->Add Connection** menu.

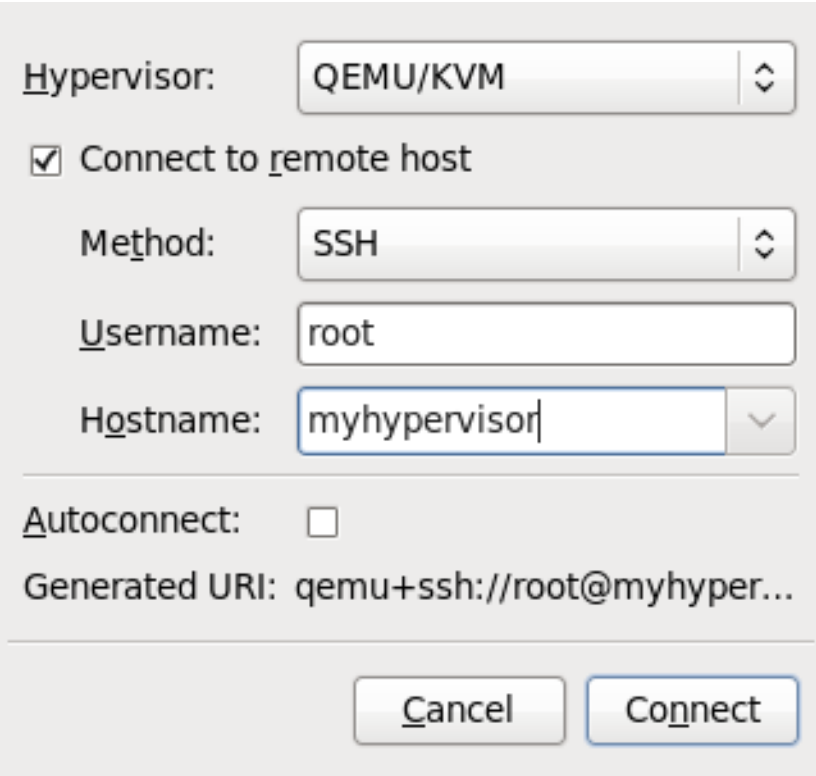


Figure 5.1. Add connection menu

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

5.2. REMOTE MANAGEMENT OVER TLS AND SSL

You can manage virtual machines using TLS and SSL. TLS and SSL provides greater scalability but is more complicated than ssh (refer to Section 5.1, "Remote Management with SSH"). TLS and SSL is the same technology used by web browsers for secure connections. The **libvirt** management connection opens a TCP port for incoming connections, which is securely encrypted and authenticated based on x509 certificates. The procedures that follow provide instructions on creating and deploying authentication certificates for TLS and SSL management.

Procedure 5.1. Creating a certificate authority (CA) key for TLS management

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

```
# yum install gnutls-utils
```

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

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

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

```
# vim ca.info
```

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

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

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

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

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

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

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

Procedure 5.2. Issuing a server certificate

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

1. Create a private key for the server.

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

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

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

3. Create the certificate with the following command:

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

4. This results in two files being generated:

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

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

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

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

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

Procedure 5.3. Issuing a client certificate

1. For every client (ie. any program linked with libvirt, such as virt-manager), you need to issue a certificate with the X.509 Distinguished Name (DN) set to a suitable name. This needs to be decided on a corporate level.

   For example purposes the following information will be used:

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

   This process is quite similar to Procedure 5.2, "Issuing a server certificate" , with the following exceptions noted.

2. Make a private key with the following command:

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

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

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

4. Sign the certificate with the following command:

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

5. Install the certificates on the client machine:

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

5.3. TRANSPORT MODES

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

Transport Layer Security (TLS)

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

UNIX Sockets

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

SSH

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

ext

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

TCP

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

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

Remote URIs

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

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

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

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

The following are examples of valid remote URIs:

- qemu://hostname/

- xen://hostname/
- xen+ssh://hostname/

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

Examples of remote management parameters

- Connect to a remote KVM host named **host2**, using SSH transport and the SSH user name **virtuser**.The connect command for each is **connect [<name>] [--readonly]**, where **<name>** is a valid URI as explained here. For more information about the **virsh connect** command refer to Section 14.1.5, "connect"

```
qemu+ssh://virtuser@hot2/
```

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

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

Testing examples

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

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

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

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

Extra URI Parameters

Extra parameters can be appended to remote URIs. The table below Table 5.1, "Extra URI parameters" covers the recognized parameters. All other parameters are ignored. Note that parameter values must be URI-escaped (that is, a question mark (?) is appended before the parameter and special characters are converted into the URI format).

Table 5.1. Extra URI parameters

Name	Transport mode	Description	Example usage

Name	Transport mode	Description	Example usage
name	all modes	The name passed to the remote virConnectOpen function. The name is normally formed by removing transport, host name, port number, user name and extra parameters from the remote URI, but in certain very complex cases it may be better to supply the name explicitly.	name=qemu:///system
command	ssh and ext	The external command. For ext transport this is required. For ssh the default is ssh. The PATH is searched for the command.	command=/opt/openssh/bin/ssh
socket	unix and ssh	The path to the UNIX domain socket, which overrides the default. For ssh transport, this is passed to the remote netcat command (see netcat).	socket=/opt/libvirt/run/libvirt/libvirt-sock
netcat	ssh	The **netcat** command can be used to connect to remote systems. The default netcat parameter uses the **nc** command. For SSH transport, libvirt constructs an SSH command using the form below: ***command*** -p ***port*** [-l ***username***] ***hostname*** ***netcat*** -U socket The ***port***, ***username*** and ***hostname*** parameters can be specified as part of the remote URI. The ***command***, ***netcat*** and ***socket*** come from other extra parameters.	netcat=/opt/netcat/bin/nc

Name	Transport mode	Description	Example usage
no_verify	tls	If set to a non-zero value, this disables client checks of the server's certificate. Note that to disable server checks of the client's certificate or IP address you must change the libvirtd configuration.	no_verify=1
no_tty	ssh	If set to a non-zero value, this stops ssh from asking for a password if it cannot log in to the remote machine automatically . Use this when you do not have access to a terminal .	no_tty=1

CHAPTER 6. OVERCOMMITTING WITH KVM

The KVM hypervisor automatically overcommits CPUs and memory. This means that more virtualized CPUs and memory can be allocated to virtual machines than there are physical resources on the system. This is possible because most processes do not access 100% of their allocated resources all the time.

As a result, under-utilized virtualized servers or desktops can run on fewer hosts, which saves a number of system resources, with the net effect of less power, cooling, and investment in server hardware.

6.1. OVERCOMMITTING MEMORY

Guest virtual machines running on a KVM hypervisor do not have dedicated blocks of physical RAM assigned to them. Instead, each guest virtual machine functions as a Linux process where the host physical machine's Linux kernel allocates memory only when requested. In addition the host's memory manager can move the guest virtual machine's memory between its own physical memory and swap space.

Overcommitting requires allotting sufficient swap space on the host physical machine to accommodate all guest virtual machines as well as enough memory for the host physical machine's processes. As a basic rule, the host physical machine's operating system requires a maximum of 4GB of memory along with a minimum of 4GB of swap space. For advanced instructions on determining an appropriate size for the swap partition, see the Red Hat KnowledgeBase .

IMPORTANT

Overcommitting is not an ideal solution for general memory issues. The recommended methods to deal with memory shortage are to allocate less memory per guest, add more physical memory to the host, or utilize swap space.

A virtual machine will run slower if it is swapped frequently. In addition, overcommitting can cause the system to run out of memory (OOM), which may lead to the Linux kernel shutting down important system processes. If you decide to overcommit memory, ensure sufficient testing is performed. Contact Red Hat support for assistance with overcommitting.

Overcommitting does not work with all guest virtual machines, but has been found to work in a desktop virtualization setup with minimal intensive usage or running several identical guest virtual machines with kernel same-page merging (KSM).

For more information on KSM and overcommitting, refer to Chapter 7, *KSM*.

IMPORTANT

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

6.2. OVERCOMMITTING VIRTUALIZED CPUS

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

Virtualized CPUs (vCPUs) are overcommitted best when a single host physical machine has multiple guest virtual machines that do not share the same vCPU. KVM should safely support guest virtual machines with loads under 100% at a ratio of five VCPUs (on 5 virtual machines) to one physical CPU on one single host physical machine. KVM will switch between all of the machines, making sure that the load is balanced.

Do not overcommit guest virtual machines on more than the physical number of processing cores. For example a guest virtual machine with four vCPUs should not be run on a host physical machine with a dual core processor, but on a quad core host. In addition, it is not recommended to have more than 10 total allocated vCPUs per physical processor core.

IMPORTANT

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

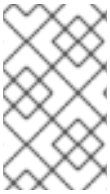
For more information on how to get the best performance out of your virtual machine, refer to the Red Hat Enterprise Linux 6 Virtualization Tuning and Optimization Guide.

CHAPTER 7. KSM

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

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

This is useful for virtualization with KVM. When a guest virtual machine is started, it inherits only the memory from the parent **qemu-kvm** process. Once the guest virtual machine is running, the contents of the guest virtual machine operating system image can be shared when guests are running the same operating system or applications.



NOTE

The page deduplication technology (used also by the KSM implementation) may introduce side channels that could potentially be used to leak information across multiple guests. In case this is a concern, KSM can be disabled on a per-guest basis.

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

NOTE

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

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

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

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

The KSM Service

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

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

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

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

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

```
# chkconfig ksm on
```

The KSM Tuning Service

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

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

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

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

- **thres** - Activation threshold, in kbytes. A KSM cycle is triggered when the ***thres*** value added to the sum of all ***qemu-kvm*** processes RSZ exceeds total system memory. This parameter is the equivalent in kbytes of the percentage defined in ***KSM_THRES_COEF***.

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

```
# Configuration file for ksmtuned.

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

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

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

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

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

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

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

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

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

KSM Variables and Monitoring

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

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

The /sys/kernel/mm/ksm/ files

full_scans

Full scans run.

pages_shared

Total pages shared.

pages_sharing

Pages presently shared.

pages_to_scan

Pages not scanned.

pages_unshared

Pages no longer shared.

pages_volatile

Number of volatile pages.

run

Whether the KSM process is running.

sleep_millisecs

Sleep milliseconds.

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

Deactivating KSM

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

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

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

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

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

IMPORTANT

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

CHAPTER 8. ADVANCED GUEST VIRTUAL MACHINE ADMINISTRATION

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

8.1. CONTROL GROUPS (CGROUPS)

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

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

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

8.2. HUGE PAGE SUPPORT

This section provides information about huge page support.

Introduction

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

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

Transparent Huge Pages

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

To use transparent huge pages, no special configuration in the **qemu.conf** file is required. Huge pages are used by default if **/sys/kernel/mm/redhat_transparent_hugepage/enabled** is set to **always**.

Transparent huge pages do not prevent the use of the **hugetlbfs** feature. However, when hugetlbfs is not used, KVM will use transparent huge pages instead of the regular 4kB page size.

NOTE

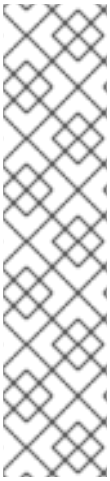
See the Red Hat Enterprise Linux 7 Virtualization Tuning and Optimization Guide for instructions on tuning memory performance with huge pages.

8.3. RUNNING RED HAT ENTERPRISE LINUX AS A GUEST VIRTUAL MACHINE ON A HYPER-V HYPERVISOR

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

- Upgraded VMBUS protocols - VMBUS protocols have been upgraded to Windows 8 level. As part of this work, now VMBUS interrupts can be processed on all available virtual CPUs in the guest. Furthermore, the signaling protocol between the Red Hat Enterprise Linux guest virtual machine and the Windows host physical machine has been optimized.
- Synthetic frame buffer driver - Provides enhanced graphics performance and superior resolution for Red Hat Enterprise Linux desktop users.
- Live Virtual Machine Backup support - Provisions uninterrupted backup support for live Red Hat Enterprise Linux guest virtual machines.
- Dynamic expansion of fixed size Linux VHDs - Allows expansion of live mounted fixed size Red Hat Enterprise Linux VHDs.

For more information, refer to the following article: Enabling Linux Support on Windows Server 2012 R2 Hyper-V.



NOTE

The Hyper-V hypervisor supports shrinking a GPT-partitioned disk on a Red Hat Enterprise Linux guest if there is free space after the last partition, by allowing the user to drop the unused last part of the disk. However, this operation will silently delete the secondary GPT header on the disk, which may produce error messages when the guest examines the partition table (for example, when printing the partition table with **parted**). This is a known limit of Hyper-V. As a workaround, it is possible to manually restore the secondary GPT header after shrinking the GPT disk by using the expert menu in **gdisk** and the **e** command. Furthermore, using the "expand" option in the Hyper-V manager also places the GPT secondary header in a location other than at the end of disk, but this can be moved with **parted**. See the **gdisk** and **parted** man pages for more information on these commands.

8.4. GUEST VIRTUAL MACHINE MEMORY ALLOCATION

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

Valid memory units include:

- ***b*** or ***bytes*** for bytes
- ***KB*** for kilobytes (10^3 or blocks of 1,000 bytes)
- ***k*** or ***KiB*** for kibibytes (2^{10} or blocks of 1024 bytes)
- ***MB*** for megabytes (10^6 or blocks of 1,000,000 bytes)

- ***M*** or ***MiB*** for mebibytes (2^{20} or blocks of 1,048,576 bytes)
- ***GB*** for gigabytes (10^{9} or blocks of 1,000,000,000 bytes)
- ***G*** or ***GiB*** for gibibytes (2^{30} or blocks of 1,073,741,824 bytes)
- ***TB*** for terabytes (10^{12} or blocks of 1,000,000,000,000 bytes)
- ***T*** or ***TiB*** for tebibytes (2^{40} or blocks of 1,099,511,627,776 bytes)

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

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

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

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

```
<domain>

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

8.5. AUTOMATICALLY STARTING GUEST VIRTUAL MACHINES

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

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

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

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

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

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

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

8.6. DISABLE SMART DISK MONITORING FOR GUEST VIRTUAL MACHINES

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

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

8.7. CONFIGURING A VNC SERVER

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

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

If needed, Create and then Edit the **~/.vnc/xstartup** file to start a GNOME session whenever **vncserver** is started. The first time you run the **vncserver** script it will ask you for a password you want to use for your VNC session. For more information on vnc server files refer to the *Red Hat Enterprise Linux Installation Guide*.

8.8. GENERATING A NEW UNIQUE MAC ADDRESS

In some cases you will need to generate a new and unique MAC address for a guest virtual machine. There is no command line tool available to generate a new MAC address at the time of writing. The script provided below can generate a new MAC address for your guest virtual machines. Save the script on your guest virtual machine as **macgen.py**. Now from that directory you can run the script using **./macgen.py** and it will generate a new MAC address. A sample output would look like the following:

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

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

8.8.1. Another Method to Generate a New MAC for Your Guest Virtual Machine

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

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

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

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

8.9. IMPROVING GUEST VIRTUAL MACHINE RESPONSE TIME

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

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

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

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

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

For more information on overcommitting with KVM, refer to Chapter 6, *Overcommitting with KVM*.

WARNING

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

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

8.10. VIRTUAL MACHINE TIMER MANAGEMENT WITH LIBVIRT

Accurate time keeping on guest virtual machines is a key challenge for virtualization platforms. Different hypervisors attempt to handle the problem of time keeping in a variety of ways. libvirt provides hypervisor independent configuration settings for time management, using the <clock> and <timer> elements in the domain XML. The domain XML can be edited using the **virsh edit** command. See Section 14.6, "Editing a Guest Virtual Machine's configuration file" for details.

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

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

  Table 8.1. Offset attribute values

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

NOTE

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

- The **timezone** attribute determines which timezone is used for the guest virtual machine clock.
- The **adjustment** attribute provides the delta for guest virtual machine clock synchronization. In seconds, relative to UTC.

Example 8.1. Always synchronize to UTC

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

Example 8.2. Always synchronize to the host physical machine timezone

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

Example 8.3. Synchronize to an arbitrary timezone

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

Example 8.4. Synchronize to UTC + arbitrary offset

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

8.10.1. timer Child Element for clock

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

The **name** attribute dictates the type of the time source to use, and can be one of the following:

Table 8.2. name attribute values

Value	Description
pit	Programmable Interval Timer - a timer with periodic interrupts.
rtc	Real Time Clock - a continuously running timer with periodic interrupts.

Value	Description
tsc	Time Stamp Counter - counts the number of ticks since reset, no interrupts.
kvmclock	KVM clock - recommended clock source for KVM guest virtual machines. KVM pvclock, or kvm-clock lets guest virtual machines read the host physical machine's wall clock time.

8.10.2. track

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

Table 8.3. track attribute values

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

8.10.3. tickpolicy

The **tickpolicy** attribute assigns the policy used to pass ticks on to the guest virtual machine. The following values are accepted:

Table 8.4. tickpolicy attribute values

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

8.10.4. frequency, mode, and present

The **frequency** attribute is used to set a fixed frequency, and is measured in Hz. This attribute is only relevant when the **name** element has a value of **tsc**. All other timers operate at a fixed frequency (**pit**, **rtc**).

mode determines how the time source is exposed to the guest virtual machine. This attribute is only relevant for a name value of **tsc**. All other timers are always emulated. Command is as follows: **<timer name='tsc' frequency='NNN' mode='auto|native|emulate|smpsafe'/>**. Mode definitions are given in the table.

Table 8.5. mode attribute values

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

present is used to override the default set of timers visible to the guest virtual machine..

Table 8.6. present attribute values

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

8.10.5. Examples Using Clock Synchronization

Example 8.5. Clock synchronizing to local time with RTC and PIT timers

In this example. the clock is synchronized to local time with RTC and PIT timers

```
<clock offset="localtime">
 <timer name="rtc" tickpolicy="catchup" track="guest virtual machine" />
 <timer name="pit" tickpolicy="delay" />

</clock>
```

NOTE

The PIT clocksource can be used with a 32-bit guest running under a 64-bit host (which cannot use PIT), with the following conditions:

- Guest virtual machine may have only one CPU
- APIC timer must be disabled (use the "noapictimer" command line option)
- NoHZ mode must be disabled in the guest (use the "nohz=off" command line option)
- High resolution timer mode must be disabled in the guest (use the "highres=off" command line option)
- The PIT clocksource is not compatible with either high resolution timer mode, or NoHz mode.

8.11. USING PMU TO MONITOR GUEST VIRTUAL MACHINE PERFORMANCE

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

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

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

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

```
# yum install perf.
```

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

8.12. GUEST VIRTUAL MACHINE POWER MANAGEMENT

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

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

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

CHAPTER 9. GUEST VIRTUAL MACHINE DEVICE CONFIGURATION

Red Hat Enterprise Linux 6 supports three classes of devices for guest virtual machines:

- *Emulated devices* are purely virtual devices that mimic real hardware, allowing unmodified guest operating systems to work with them using their standard in-box drivers. Red Hat Enterprise Linux 6 supports up to 216 virtio devices.
- *Virtio devices* are purely virtual devices designed to work optimally in a virtual machine. Virtio devices are similar to emulated devices, however, non-Linux virtual machines do not include the drivers they require by default. Virtualization management software like the Virtual Machine Manager (**virt-manager**) and the Red Hat Virtualization Hypervisor (RHV-H) install these drivers automatically for supported non-Linux guest operating systems. Red Hat Enterprise Linux 6 supports up to 700 scsi disks.
- *Assigned devices* are physical devices that are exposed to the virtual machine. This method is also known as 'passthrough'. Device assignment allows virtual machines exclusive access to PCI devices for a range of tasks, and allows PCI devices to appear and behave as if they were physically attached to the guest operating system. Red Hat Enterprise Linux 6 supports up to 32 assigned devices per virtual machine.

Device assignment is supported on PCIe devices, including select graphics devices. Nvidia K-series Quadro, GRID, and Tesla graphics card GPU functions are now supported with device assignment in Red Hat Enterprise Linux 6. Parallel PCI devices may be supported as assigned devices, but they have severe limitations due to security and system configuration conflicts.

NOTE

The number of devices that can be attached to a virtual machine depends on several factors. One factor is the number of files open by the QEMU process (configured in **/etc/security/limits.conf**, which can be overridden by **/etc/libvirt/qemu.conf**). Other limitation factors include the number of slots available on the virtual bus, as well as the system-wide limit on open files set by sysctl.

For more information on specific devices and for limitations refer to Section 20.16, “Devices”.

Red Hat Enterprise Linux 6 supports PCI hot plug of devices exposed as single function slots to the virtual machine. Single function host devices and individual functions of multi-function host devices may be configured to enable this. Configurations exposing devices as multi-function PCI slots to the virtual machine are recommended only for non-hotplug applications.

NOTE

Platform support for interrupt remapping is required to fully isolate a guest with assigned devices from the host. Without such support, the host may be vulnerable to interrupt injection attacks from a malicious guest. In an environment where guests are trusted, the admin may opt-in to still allow PCI device assignment using the **allow_unsafe_interrupts** option to the ***vfio_iommu_type1*** module. This may either be done persistently by adding a .conf file (for example **local.conf**) to **/etc/modprobe.d** containing the following:

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

or dynamically using the sysfs entry to do the same:

```
# echo 1 > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
```

9.1. PCI DEVICES

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

Procedure 9.1. Preparing an Intel system for PCI device assignment

1. **Enable the Intel VT-d specifications**
   The Intel VT-d specifications provide hardware support for directly assigning a physical device to a virtual machine. These specifications are required to use PCI device assignment with Red Hat Enterprise Linux.

   The Intel VT-d specifications must be enabled in the BIOS. Some system manufacturers disable these specifications by default. The terms used to refer to these specifications can differ between manufacturers; consult your system manufacturer's documentation for the appropriate terms.

2. **Activate Intel VT-d in the kernel**
   Activate Intel VT-d in the kernel by adding the ***intel_iommu=on*** parameter to the end of the GRUB_CMDLINX_LINUX line, within the quotes, in the **/etc/sysconfig/grub** file.

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

   ```
   GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
   vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
   vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ] && /usr/sbin/
   rhcrashkernel-param || :) rhgb quiet intel_iommu=on"
   ```

3. **Regenerate config file**
   Regenerate /etc/grub2.cfg by running:

   ```
   grub2-mkconfig -o /etc/grub2.cfg
   ```

   Note that if you are using a UEFI-based host, the target file should be **/etc/grub2-efi.cfg**.

4. **Ready to use**

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

Procedure 9.2. Preparing an AMD system for PCI device assignment

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

2. **Enable IOMMU kernel support**
   Append ***amd_iommu=on*** to the end of the GRUB_CMDLINX_LINUX line, within the quotes, in **/etc/sysconfig/grub** so that AMD IOMMU specifications are enabled at boot.

3. **Regenerate config file**
   Regenerate /etc/grub2.cfg by running:

   ```
   grub2-mkconfig -o /etc/grub2.cfg
   ```

   Note that if you are using a UEFI-based host, the target file should be **/etc/grub2-efi.cfg**.

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

9.1.1. Assigning a PCI Device with virsh

These steps cover assigning a PCI device to a virtual machine on a KVM hypervisor.

This example uses a PCIe network controller with the PCI identifier code, **pci_0000_01_00_0**, and a fully virtualized guest machine named *guest1-rhel6-64*.

Procedure 9.3. Assigning a PCI device to a guest virtual machine with virsh

1. **Identify the device**
   First, identify the PCI device designated for device assignment to the virtual machine. Use the **lspci** command to list the available PCI devices. You can refine the output of **lspci** with **grep**.

   This example uses the Ethernet controller highlighted in the following output:

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

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

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

   This example highlights the string that maps to the Ethernet controller with the short identifier **00:19.0**. In this example, the **:** and **.** characters are replaced with underscores in the full identifier.

```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

Record the PCI device number that maps to the device you want to use; this is required in other steps.

2. **Review device information**
   Information on the domain, bus, and function are available from output of the **virsh nodedev-dumpxml** command:

```
virsh nodedev-dumpxml pci_0000_00_19_0
<device>
  <name>pci_0000_00_19_0</name>
  <parent>computer</parent>
  <driver>
    <name>e1000e</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>0</bus>
    <slot>25</slot>
    <function>0</function>
    <product id='0x1502'>82579LM Gigabit Network Connection</product>
```

```
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='7'>
      <address domain='0x0000' bus='0x00' slot='0x19' function='0x0'/>
    </iommuGroup>
  </capability>
</device>
```

NOTE

An IOMMU group is determined based on the visibility and isolation of devices from the perspective of the IOMMU. Each IOMMU group may contain one or more devices. When multiple devices are present, all endpoints within the IOMMU group must be claimed for any device within the group to be assigned to a guest. This can be accomplished either by also assigning the extra endpoints to the guest or by detaching them from the host driver using **virsh nodedev-detach**. Devices contained within a single group may not be split between multiple guests or split between host and guest. Non-endpoint devices such as PCIe root ports, switch ports, and bridges should not be detached from the host drivers and will not interfere with assignment of endpoints.

Devices within an IOMMU group can be determined using the iommuGroup section of the **virsh nodedev-dumpxml** output. Each member of the group is provided via a separate "address" field. This information may also be found in sysfs using the following:

```
$ ls /sys/bus/pci/devices/0000:01:00.0/iommu_group/devices/
```

An example of the output from this would be:

```
0000:01:00.0  0000:01:00.1
```

To assign only 0000.01.00.0 to the guest, the unused endpoint should be detached from the host before starting the guest:

```
$ virsh nodedev-detach pci_0000_01_00_1
```

3. **Determine required configuration details**
   Refer to the output from the **virsh nodedev-dumpxml pci_0000_00_19_0** command for the values required for the configuration file.

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

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

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

```
# virsh edit guest1-rhel6-64
<hostdev mode='subsystem' type='pci' managed='yes'>
```

```
  <source>
    <address domain='0' bus='0' slot='25' function='0'/>
  </source>
</hostdev>
```

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

```
virsh attach-device guest1-rhel6-64 file.xml
```

5. **Start the virtual machine**

```
# virsh start guest1-rhel6-64
```

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

9.1.2. Assigning a PCI Device with virt-manager

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

Procedure 9.4. Assigning a PCI device to a guest virtual machine using virt-manager

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

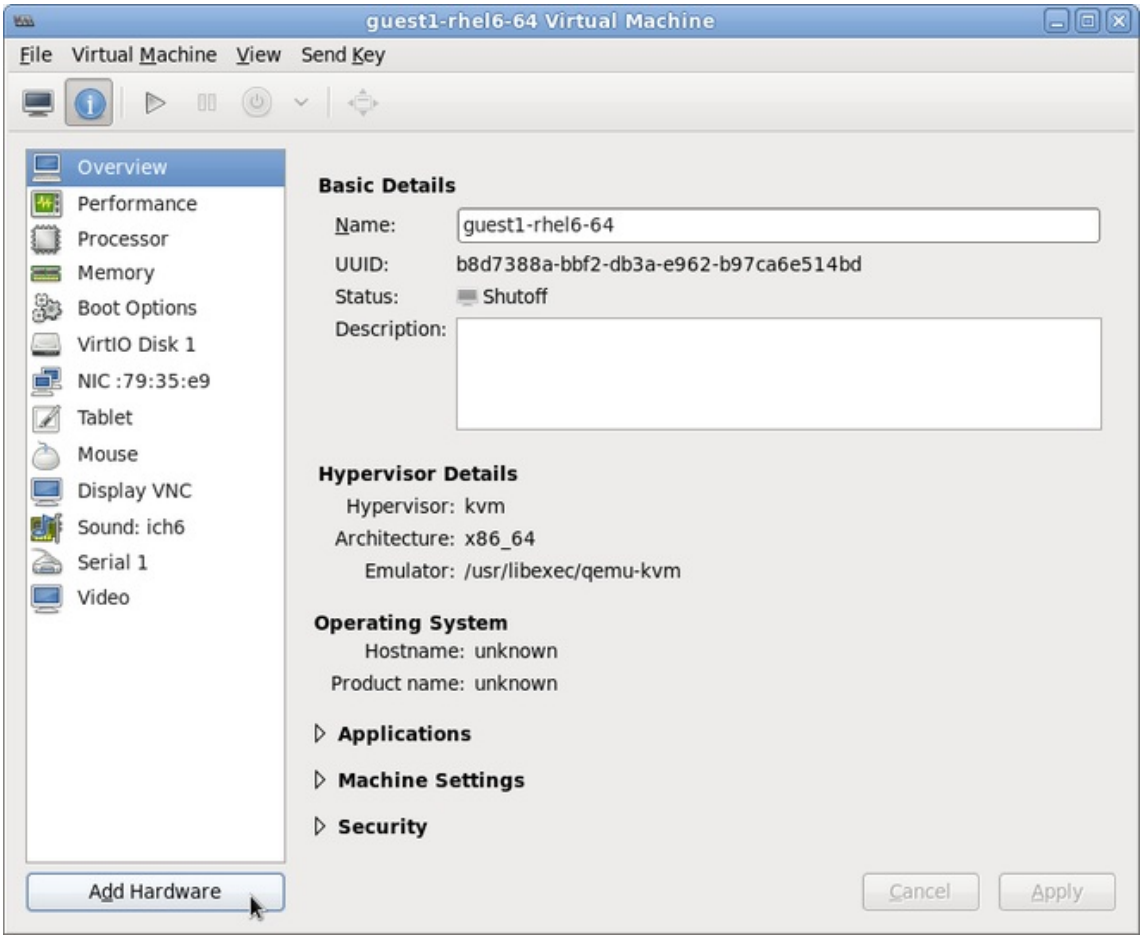


Figure 9.1. The virtual machine hardware information window

2. **Select a PCI device**
   Select **PCI Host Device** from the **Hardware** list on the left.

   Select an unused PCI device. If you select a PCI device that is in use by another guest an error may result. In this example, a spare 82576 network device is used. Click **Finish** to complete setup.

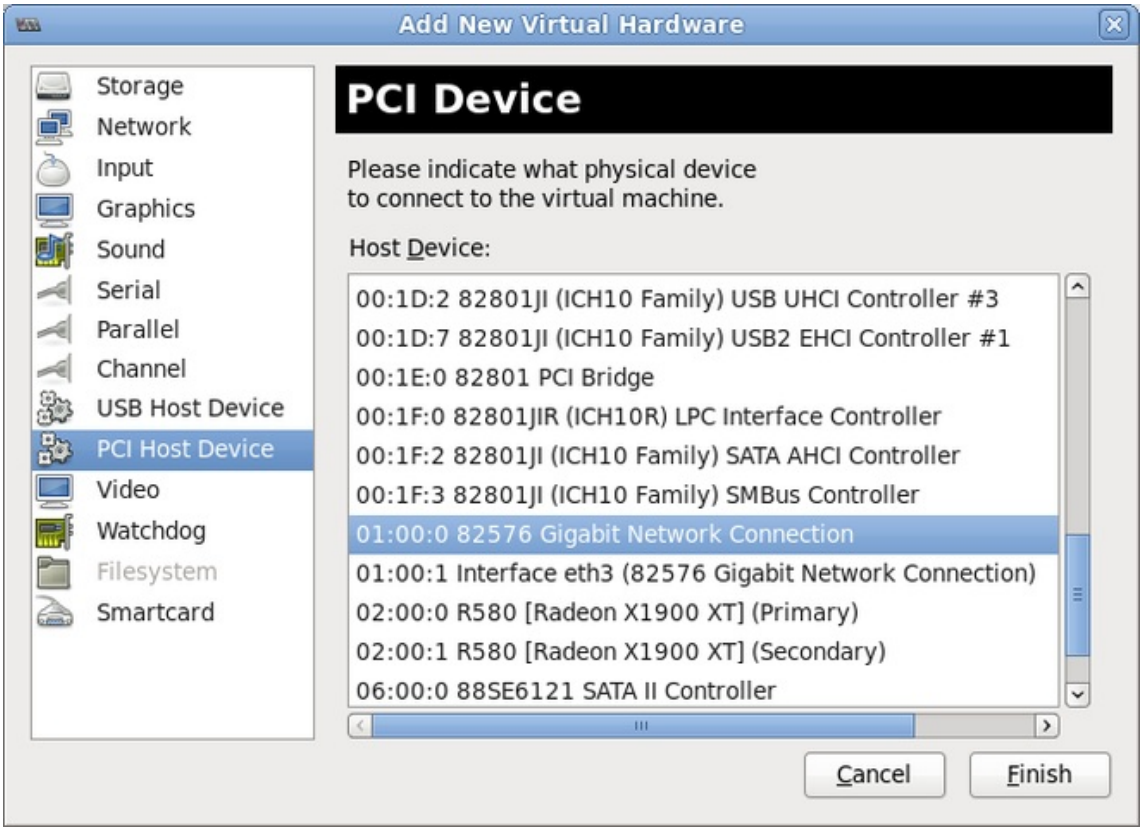


Figure 9.2. The Add new virtual hardware wizard

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

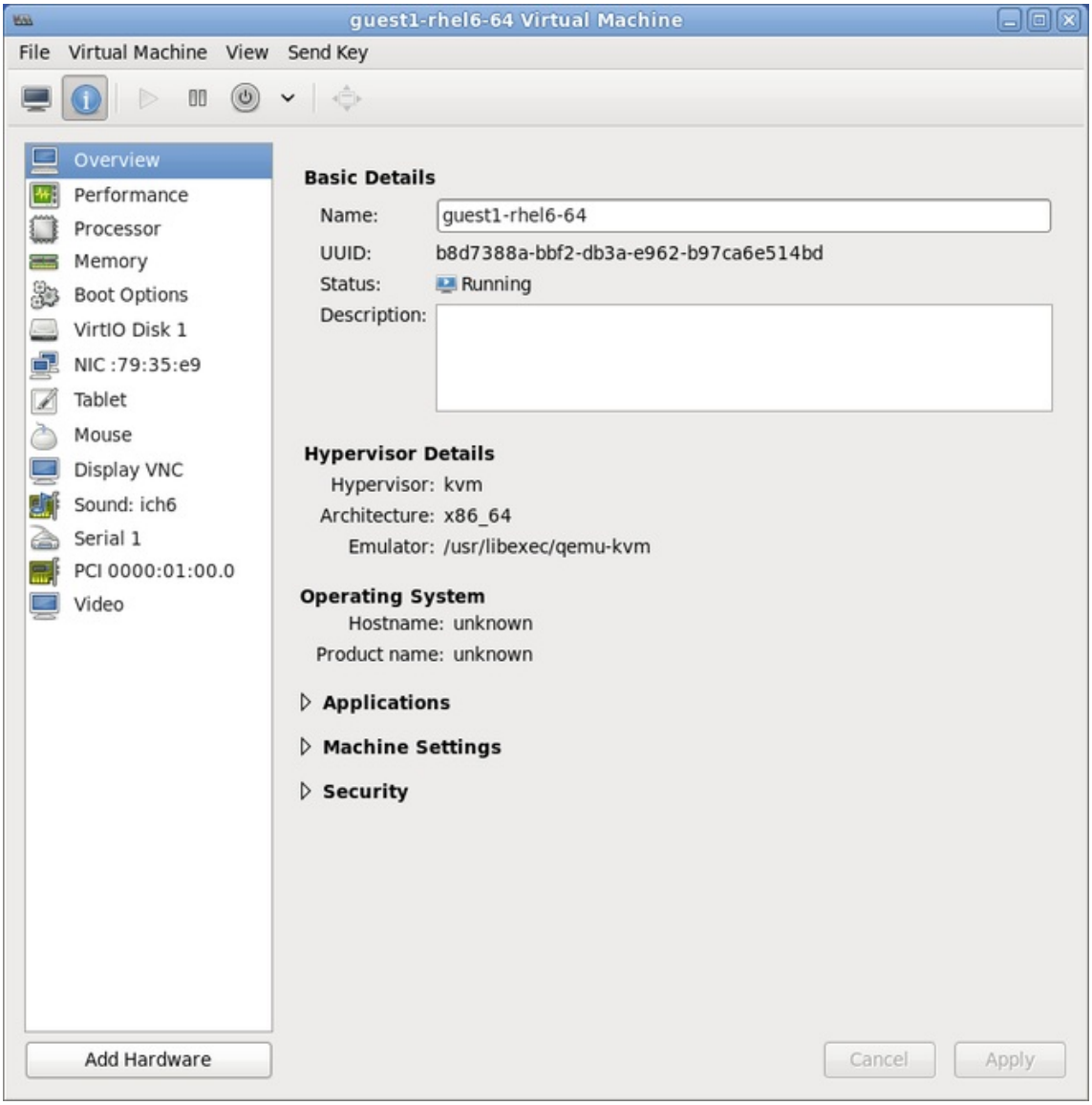
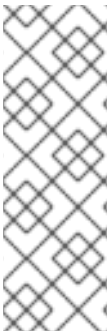


Figure 9.3. The virtual machine hardware information window



NOTE

If device assignment fails, there may be other endpoints in the same IOMMU group that are still attached to the host. There is no way to retrieve group information using virt-manager, but virsh commands can be used to analyze the bounds of the IOMMU group and if necessary sequester devices.

Refer to the Note in Section 9.1.1, "Assigning a PCI Device with virsh" for more information on IOMMU groups and how to detach endpoint devices using virsh.

9.1.3. PCI Device Assignment with virt-install

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

Procedure 9.5. Assigning a PCI device to a virtual machine with virt-install

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

```
# lspci | grep Ethernet
```

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

The **virsh nodedev-list** command lists all devices attached to the system, and identifies each PCI device with a string. To limit output to only PCI devices, run the following command:

```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

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

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

```
# virsh nodedev-dumpxml pci_0000_01_00_0
<device>
  <name>pci_0000_01_00_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
```

```
  <capability type='pci'>
    <domain>0</domain>
    <bus>1</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='7'>
      <address domain='0x0000' bus='0x00' slot='0x19' function='0x0'/>
    </iommuGroup>
  </capability>
</device>
```

NOTE

If there are multiple endpoints in the IOMMU group and not all of them are assigned to the guest, you will need to manually detach the other endpoint(s) from the host by running the following command before you start the guest:

```
$ virsh nodedev-detach pci_0000_00_19_1
```

Refer to the Note in Section 9.1.1, "Assigning a PCI Device with virsh" for more information on IOMMU groups.

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

```
virt-install \
--name=guest1-rhel6-64 \
--disk path=/var/lib/libvirt/images/guest1-rhel6-64.img,size=8 \
--nonsparse --graphics spice \
--vcpus=2 --ram=2048 \
--location=http://example1.com/installation_tree/RHEL6.0-Server-x86_64/os \
--nonetworks \
--os-type=linux \
--os-variant=rhel6
--host-device=pci_0000_01_00_0
```

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

9.1.4. Detaching an Assigned PCI Device

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

Procedure 9.6. Detaching a PCI device from a guest with virsh

1. **Detach the device**
   Use the following command to detach the PCI device from the guest by removing it in the guest's XML file:

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

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

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

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

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

   ```
   virsh nodedev-reattach pci_0000_01_00_0
   ```

   The device is now available for host use.

Procedure 9.7. Detaching a PCI Device from a guest with virt-manager

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

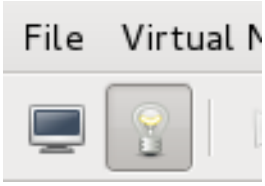
   

   Figure 9.4. The virtual hardware details button

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

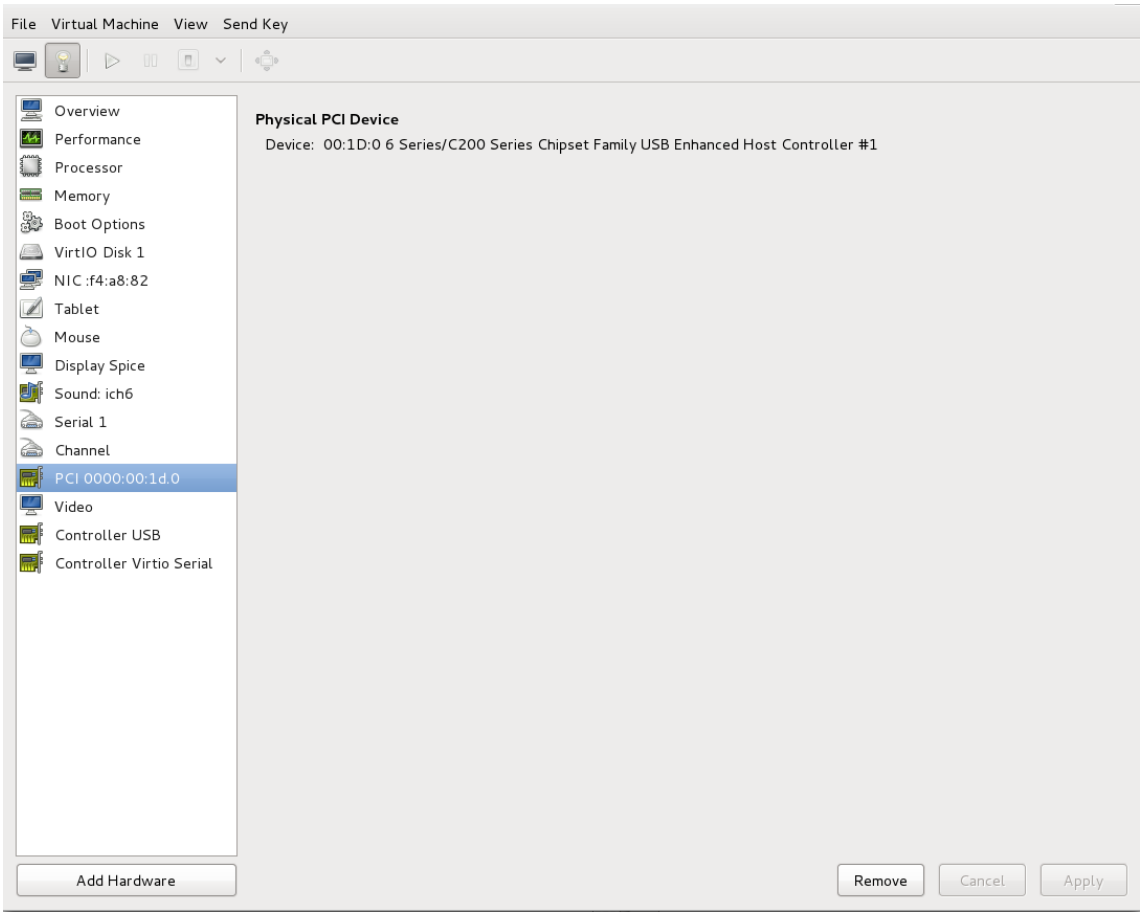


Figure 9.5. Selecting the PCI device to be detached

Click the **Remove** button to confirm. The device is now available for host use.

9.1.5. Creating PCI Bridges

Peripheral Component Interconnects (PCI) bridges are used to attach to devices such as network cards, modems and sound cards. Just like their physical counterparts, virtual devices can also be attached to a PCI Bridge. In the past, only 31 PCI devices could be added to any guest virtual machine. Now, when a 31st PCI device is added, a PCI bridge is automatically placed in the 31st slot moving the additional PCI device to the PCI bridge. Each PCI bridge has 31 slots for 31 additional devices, all of which can be bridges. In this manner, over 900 devices can be available for guest virtual machines.

NOTE

This action cannot be performed when the guest virtual machine is running. You must add the PCI device on a guest virtual machine that is shutdown.

9.1.6. PCI Passthrough

A PCI network device (specified by the **<source>** element) is directly assigned to the guest using generic device *passthrough*, after first optionally setting the device's MAC address to the configured value, and associating the device with an 802.1Qbh capable switch using an optionally specified **<virtualport>** element (see the examples of virtualport given above for type='direct' network devices). Due to limitations in standard single-port PCI ethernet card driver design - only SR-IOV (Single Root I/O Virtualization) virtual function (VF) devices can be assigned in this manner; to assign a standard single-port PCI or PCIe Ethernet card to a guest, use the traditional **<hostdev>** device definition.

To use VFIO device assignment rather than traditional/legacy KVM device assignment (VFIO is a new

method of device assignment that is compatible with UEFI Secure Boot), a **<type='hostdev'>** interface can have an optional driver sub-element with a name attribute set to "vfio". To use legacy KVM device assignment you can set name to "kvm" (or simply omit the **<driver>** element, since **<driver='kvm'>** is currently the default).

NOTE

Intelligent passthrough of network devices is very similar to the functionality of a standard **<hostdev>** device, the difference being that this method allows specifying a MAC address and **<virtualport>** for the passed-through device. If these capabilities are not required, if you have a standard single-port PCI, PCIe, or USB network card that does not support SR-IOV (and hence would anyway lose the configured MAC address during reset after being assigned to the guest domain), or if you are using a version of libvirt older than 0.9.11, you should use standard **<hostdev>** to assign the device to the guest instead of **<interface type='hostdev'/>**.

```
 <devices>
<interface type='hostdev'>
  <driver name='vfio'/>
  <source>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
  </source>
  <mac address='52:54:00:6d:90:02'>
  <virtualport type='802.1Qbh'>
    <parameters profileid='finance'/>
  </virtualport>
</interface>
</devices>
```

Figure 9.6. XML example for PCI device assignment

9.1.7. Configuring PCI Assignment (Passthrough) with SR-IOV Devices

This section is for SR-IOV devices only. SR-IOV network cards provide multiple *Virtual Functions* (VFs) that can each be individually assigned to a guest virtual machines using PCI device assignment. Once assigned, each will behave as a full physical network device. This permits many guest virtual machines to gain the performance advantage of direct PCI device assignment, while only using a single slot on the host physical machine.

These VFs can be assigned to guest virtual machines in the traditional manner using the element **<hostdev>**, but as SR-IOV VF network devices do not have permanent unique MAC addresses, it causes issues where the guest virtual machine's network settings would have to be re-configured each time the host physical machine is rebooted. To remedy this, you would need to set the MAC address prior to assigning the VF to the host physical machine and you would need to set this each and every time the guest virtual machine boots. In order to assign this MAC address as well as other options, refer to the procedure described in Procedure 9.8, “Configuring MAC addresses, vLAN, and virtual ports for assigning PCI devices on SR-IOV”.

Procedure 9.8. Configuring MAC addresses, vLAN, and virtual ports for assigning PCI devices on SR-IOV

It is important to note that the **<hostdev>** element cannot be used for function-specific items like MAC address assignment, vLAN tag ID assignment, or virtual port assignment because the **<mac>**, **<vlan>**,

and **<virtualport>** elements are not valid children for **<hostdev>**. As they are valid for **<interface>**, support for a new interface type was added (**<interface type='hostdev'>**). This new interface device type behaves as a hybrid of an **<interface>** and **<hostdev>**. Thus, before assigning the PCI device to the guest virtual machine, libvirt initializes the network-specific hardware/switch that is indicated (such as setting the MAC address, setting a vLAN tag, or associating with an 802.1Qbh switch) in the guest virtual machine's XML configuration file. For information on setting the vLAN tag, refer to Section 18.14, "Setting vLAN Tags".

1. **Shutdown the guest virtual machine**
   Using **virsh shutdown** command (refer to Section 14.9.1, "Shutting Down a Guest Virtual Machine"), shutdown the guest virtual machine named *guestVM*.

   ```
   # virsh shutdown guestVM
   ```

2. **Gather information**
   In order to use **<interface type='hostdev'>**, you must have an SR-IOV-capable network card, host physical machine hardware that supports either the Intel VT-d or AMD IOMMU extensions, and you must know the PCI address of the VF that you wish to assign.

3. **Open the XML file for editing**
   Run the # **virsh save-image-edit** command to open the XML file for editing (refer to Section 14.8.10, "Edit Domain XML Configuration Files" for more information). As you would want to restore the guest virtual machine to its former running state, the **--running** would be used in this case. The name of the configuration file in this example is *guestVM.xml*, as the name of the guest virtual machine is *guestVM*.

   ```
   # virsh save-image-edit guestVM.xml --running
   ```

   The *guestVM.xml* opens in your default editor.

4. **Edit the XML file**
   Update the configuration file (*guestVM.xml*) to have a **<devices>** entry similar to the following:

```
<devices>
  ...
  <interface type='hostdev' managed='yes'>
    <source>
      <address type='pci' domain='0x0' bus='0x00' slot='0x07' function='0x0'/> <!--these values
can be decimal as well-->
    </source>
    <mac address='52:54:00:6d:90:02'/>                    <!--sets the mac address-->
    <virtualport type='802.1Qbh'>                         <!--sets the virtual port for the
802.1Qbh switch-->
      <parameters profileid='finance'/>
    </virtualport>
    <vlan>                                                <!--sets the vlan tag-->
     <tag id='42'/>
    </vlan>
  </interface>
  ...
</devices>
```

Figure 9.7. Sample domain XML for hostdev interface type

Note that if you do not provide a MAC address, one will be automatically generated, just as with any other type of interface device. Also, the **<virtualport>** element is only used if you are connecting to an 802.11Qgh hardware switch (802.11Qbg (a.k.a. "VEPA") switches are currently not supported.

5. **Re-start the guest virtual machine**
   Run the **virsh start** command to restart the guest virtual machine you shutdown in the first step (example uses guestVM as the guest virtual machine's domain name). Refer to Section 14.8.1, "Starting a Defined Domain" for more information.

   ```
   # virsh start guestVM
   ```

   When the guest virtual machine starts, it sees the network device provided to it by the physical host machine's adapter, with the configured MAC address. This MAC address will remain unchanged across guest virtual machine and host physical machine reboots.

9.1.8. Setting PCI Device Assignment from a Pool of SR-IOV Virtual Functions

Hard coding the PCI addresses of a particular *Virtual Functions* (VFs) into a guest's configuration has two serious limitations:

- The specified VF must be available any time the guest virtual machine is started, implying that the administrator must permanently assign each VF to a single guest virtual machine (or modify the configuration file for every guest virtual machine to specify a currently unused VF's PCI address each time every guest virtual machine is started).
- If the guest virtual machine is moved to another host physical machine, that host physical machine must have exactly the same hardware in the same location on the PCI bus (or, again, the guest virtual machine configuration must be modified prior to start).

It is possible to avoid both of these problems by creating a libvirt network with a device pool containing all the VFs of an SR-IOV device. Once that is done you would configure the guest virtual machine to

reference this network. Each time the guest is started, a single VF will be allocated from the pool and assigned to the guest virtual machine. When the guest virtual machine is stopped, the VF will be returned to the pool for use by another guest virtual machine.

Procedure 9.9. Creating a device pool

1. **Shutdown the guest virtual machine**
   Using **virsh shutdown** command (refer to Section 14.9, "Shutting Down, Rebooting, and Forcing Shutdown of a Guest Virtual Machine"), shutdown the guest virtual machine named *guestVM*.

   ```
   # virsh shutdown guestVM
   ```

2. **Create a configuration file**
   Using your editor of choice create an XML file (named *passthrough.xml*, for example) in the **/tmp** directory. Make sure to replace **pf dev='eth3'** with the netdev name of your own SR-IOV device's PF

   The following is an example network definition that will make available a pool of all VFs for the SR-IOV adapter with its *physical function (PF)* at "eth3' on the host physical machine:

   ```
   <network>
     <name>passthrough</name>                    <!--This is the name of the file
   you created-->
     <forward mode='hostdev' managed='yes'>
       <pf dev='myNetDevName'/>                  <!--Use the netdev name of your
   SR-IOV devices PF here-->
     </forward>
   </network>
   ```

   Figure 9.8. Sample network definition domain XML

3. **Load the new XML file**
   Run the following command, replacing */tmp/passthrough.xml*, with the name and location of your XML file you created in the previous step:

   ```
   # virsh net-define /tmp/passthrough.xml
   ```

4. **Restarting the guest**
   Run the following replacing *passthrough.xml*, with the name of your XML file you created in the previous step:

   ```
   # virsh net-autostart passthrough # virsh net-start passthrough
   ```

5. **Re-start the guest virtual machine**
   Run the **virsh start** command to restart the guest virtual machine you shutdown in the first step (example uses guestVM as the guest virtual machine's domain name). Refer to Section 14.8.1, "Starting a Defined Domain" for more information.

   ```
   # virsh start guestVM
   ```

6. **Initiating passthrough for devices**
   Although only a single device is shown, libvirt will automatically derive the list of all VFs associated with that PF the first time a guest virtual machine is started with an interface definition in its domain XML like the following:

```
<interface type='network'>
  <source network='passthrough'>
</interface>
```

Figure 9.9. Sample domain XML for interface network definition

7. **Verification**
   You can verify this by running **virsh net-dumpxml *passthrough*** command after starting the first guest that uses the network; you will get output similar to the following:

```
<network connections='1'>
  <name>passthrough</name>
  <uuid>a6b49429-d353-d7ad-3185-4451cc786437</uuid>
  <forward mode='hostdev' managed='yes'>
    <pf dev='eth3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x1'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x5'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x7'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x1'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x5'/>
  </forward>
</network>
```

Figure 9.10. XML dump file*passthrough* contents

9.2. USB DEVICES

This section gives the commands required for handling USB devices.

9.2.1. Assigning USB Devices to Guest Virtual Machines

Most devices such as web cameras, card readers, keyboards, or mice are connected to a computer using a USB port and cable. There are two ways to pass such devices to a guest virtual machine:

- Using USB passthrough - this requires the device to be physically connected to the host physical machine that is hosting the guest virtual machine. SPICE is not needed in this case. USB devices on the host can be passed to the guest using the command line or **virt-manager**. Refer to Section 15.3.1, “Attaching USB Devices to a Guest Virtual Machine” for **virt manager** directions.

NOTE

virt-manager should not be used for hot plugging or hot unplugging devices. If you want to hot plug/or hot unplug a USB device, refer to Procedure 14.1, "Hot plugging USB devices for use by the guest virtual machine".

- Using USB re-direction - USB re-direction is best used in cases where there is a host physical machine that is running in a data center. The user connects to his/her guest virtual machine from a local machine or thin client. On this local machine there is a SPICE client. The user can attach any USB device to the thin client and the SPICE client will redirect the device to the host physical machine on the data center so it can be used by the guest virtual machine that is running on the thin client. For instructions on USB re-direction using the virt-manager, refer to Section 15.3.1, "Attaching USB Devices to a Guest Virtual Machine" It should be noted that USB redirection is not possible using the TCP protocol (Refer to BZ#1085318).

9.2.2. Setting a Limit on USB Device Redirection

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

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

Use the value ***-1*** to designate it to accept any value for a particular field. You may use multiple rules on the same command line using | as a separator.

IMPORTANT

If a device matches none of the rule filters, redirecting it will not be allowed!

Example 9.1. An example of limiting redirection with a windows guest virtual machine

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

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

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

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

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

4. Plug a USB device into a host physical machine, and use **virt-manager** to connect to the guest virtual machine.

5. Click **Redirect USB Service** in the menu, which will produce the following message: "Some USB devices are blocked by host policy". Click **OK** to confirm and continue.

   The filter takes effect.

6. To make sure that the filter captures properly check the USB device vendor and product, then make the following changes in the guest virtual machine's domain XML to allow for USB redirection.

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

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

9.3. CONFIGURING DEVICE CONTROLLERS

Depending on the guest virtual machine architecture, some device buses can appear more than once, with a group of virtual devices tied to a virtual controller. Normally, libvirt can automatically infer such controllers without requiring explicit XML markup, but in some cases it is better to explicitly set a virtual controller element.

```
...
<devices>
  <controller type='ide' index='0'/>
  <controller type='virtio-serial' index='0' ports='16' vectors='4'/>
  <controller type='virtio-serial' index='1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0'/>
  </controller>
  ...
</devices>
...
```

Figure 9.11. Domain XML example for virtual controllers

Each controller has a mandatory attribute **<controller type>**, which must be one of:

- ide
- fdc
- scsi
- sata
- usb
- ccid

- virtio-serial
- pci

The **<controller>** element has a mandatory attribute **<controller index>** which is the decimal integer describing in which order the bus controller is encountered (for use in controller attributes of **<address>** elements). When **<controller type ='virtio-serial'>** there are two additional optional attributes (named **ports** and **vectors**), which control how many devices can be connected through the controller. Note that Red Hat Enterprise Linux 6 does not support the use of more than 32 vectors per device. Using more vectors will cause failures in migrating the guest virtual machine.

When **<controller type ='scsi'>**, there is an optional attribute **model** model, which can have the following values:

- auto
- buslogic
- ibmvscsi
- lsilogic
- lsisas1068
- lsisas1078
- virtio-scsi
- vmpvscsi

When **<controller type ='usb'>**, there is an optional attribute **model** model, which can have the following values:

- piix3-uhci
- piix4-uhci
- ehci
- ich9-ehci1
- ich9-uhci1
- ich9-uhci2
- ich9-uhci3
- vt82c686b-uhci
- pci-ohci
- nec-xhci

NOTE

If the USB bus needs to be explicitly disabled for the guest virtual machine, **<model='none'>** may be used. .

For controllers that are themselves devices on a PCI or USB bus, an optional sub-element **<address>** can specify the exact relationship of the controller to its master bus, with semantics as shown in Section 9.4, "Setting Addresses for Devices".

An optional sub-element **<driver>** can specify the driver specific options. Currently it only supports attribute queues, which specifies the number of queues for the controller. For best performance, it is recommended to specify a value matching the number of vCPUs.

USB companion controllers have an optional sub-element **<master>** to specify the exact relationship of the companion to its master controller. A companion controller is on the same bus as its master, so the companion **index** value should be equal.

An example XML which can be used is as follows:

```
  ...
<devices>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0' bus='0' slot='4' function='7'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0'/>
    <address type='pci' domain='0' bus='0' slot='4' function='0' multifunction='on'/>
  </controller>
  ...
</devices>
...
```

Figure 9.12. Domain XML example for USB controllers

PCI controllers have an optional **model** attribute with the following possible values:

- pci-root
- pcie-root
- pci-bridge
- dmi-to-pci-bridge

The root controllers (**pci-root** and **pcie-root**) have an optional **pcihole64** element specifying how big (in kilobytes, or in the unit specified by **pcihole64**'s **unit** attribute) the 64-bit PCI hole should be. Some guest virtual machines (such as Windows Server 2003) may cause a crash, unless **unit** is disabled (set to 0 **unit='0'**).

For machine types which provide an implicit PCI bus, the pci-root controller with **index='0'** is auto-added and required to use PCI devices. pci-root has no address. PCI bridges are auto-added if there are too many devices to fit on the one bus provided by **model='pci-root'**, or a PCI bus number greater than zero was specified. PCI bridges can also be specified manually, but their addresses should only refer to PCI buses provided by already specified PCI controllers. Leaving gaps in the PCI controller indexes might lead to an invalid configuration. The following XML example can be added to the **<devices>** section:

```
...
<devices>
  <controller type='pci' index='0' model='pci-root'/>
  <controller type='pci' index='1' model='pci-bridge'>
    <address type='pci' domain='0' bus='0' slot='5' function='0' multifunction='off'/>
  </controller>
</devices>
...
```

Figure 9.13. Domain XML example for PCI bridge

For machine types which provide an implicit PCI Express (PCIe) bus (for example, the machine types based on the Q35 chipset), the pcie-root controller with **index='0'** is auto-added to the domain's configuration. pcie-root has also no address, but provides 31 slots (numbered 1-31) and can only be used to attach PCIe devices. In order to connect standard PCI devices on a system which has a pcie-root controller, a pci controller with **model='dmi-to-pci-bridge'** is automatically added. A dmi-to-pci-bridge controller plugs into a PCIe slot (as provided by pcie-root), and itself provides 31 standard PCI slots (which are not hot-pluggable). In order to have hot-pluggable PCI slots in the guest system, a pci-bridge controller will also be automatically created and connected to one of the slots of the auto-created dmi-to-pci-bridge controller; all guest devices with PCI addresses that are auto-determined by libvirt will be placed on this pci-bridge device.

```
  ...
<devices>
  <controller type='pci' index='0' model='pcie-root'/>
  <controller type='pci' index='1' model='dmi-to-pci-bridge'>
    <address type='pci' domain='0' bus='0' slot='0xe' function='0'/>
  </controller>
  <controller type='pci' index='2' model='pci-bridge'>
    <address type='pci' domain='0' bus='1' slot='1' function='0'/>
  </controller>
</devices>
...
```

Figure 9.14. Domain XML example for PCIe (PCI express)

9.4. SETTING ADDRESSES FOR DEVICES

Many devices have an optional **<address>** sub-element which is used to describe where the device is placed on the virtual bus presented to the guest virtual machine. If an address (or any optional attribute within an address) is omitted on input, libvirt will generate an appropriate address; but an explicit address is required if more control over layout is required. See Figure 9.6, "XML example for PCI device assignment" for domain XML device examples including an **<address>** element.

Every address has a mandatory attribute **type** that describes which bus the device is on. The choice of which address to use for a given device is constrained in part by the device and the architecture of the guest virtual machine. For example, a **<disk>** device uses **type='drive'**, while a **<console>** device would use **type='pci'** on i686 or x86_64 guest virtual machine architectures. Each address type has further optional attributes that control where on the bus the device will be placed as described in the table:

Table 9.1. Supported device address types

Address type	Description
type='pci'	PCI addresses have the following additional attributes: ● domain (a 2-byte hex integer, not currently used by qemu) ● bus (a hex value between 0 and 0xff, inclusive) ● slot (a hex value between 0x0 and 0x1f, inclusive) ● function (a value between 0 and 7, inclusive) ● multifunction controls turning on the multifunction bit for a particular slot/function in the PCI control register By default it is set to 'off', but should be set to 'on' for function 0 of a slot that will have multiple functions used.
type='drive'	Drive addresses have the following additional attributes: ● controller (a 2-digit controller number) ● bus (a 2-digit bus number ● target (a 2-digit bus number) ● unit (a 2-digit unit number on the bus)
type='virtio-serial'	Each virtio-serial address has the following additional attributes: ● controller (a 2-digit controller number) ● bus (a 2-digit bus number) ● slot (a 2-digit slot within the bus)
type='ccid'	A CCID address, for smart-cards, has the following additional attributes: ● bus (a 2-digit bus number) ● slot attribute (a 2-digit slot within the bus)

Address type	Description
type='usb'	USB addresses have the following additional attributes: • bus (a hex value between 0 and 0xfff, inclusive) • port (a dotted notation of up to four octets, such as 1.2 or 2.1.3.1)
type='isa'	ISA addresses have the following additional attributes: • iobase • irq

9.5. MANAGING STORAGE CONTROLLERS IN A GUEST VIRTUAL MACHINE

Starting from Red Hat Enterprise Linux 6.4, it is supported to add SCSI and virtio-SCSI devices to guest virtual machines that are running Red Hat Enterprise Linux 6.4 or later. Unlike virtio disks, SCSI devices require the presence of a controller in the guest virtual machine. Virtio-SCSI provides the ability to connect directly to SCSI LUNs and significantly improves scalability compared to virtio-blk. The advantage of virtio-SCSI is that it is capable of handling hundreds of devices compared to virtio-blk which can only handle 28 devices and exhausts PCI slots. Virtio-SCSI is now capable of inheriting the feature set of the target device with the ability to:

- attach a virtual hard drive or CD through the virtio-scsi controller,
- pass-through a physical SCSI device from the host to the guest via the QEMU scsi-block device,
- and allow the usage of hundreds of devices per guest; an improvement from the 28-device limit of virtio-blk.

This section details the necessary steps to create a virtual SCSI controller (also known as "Host Bus Adapter", or HBA) and to add SCSI storage to the guest virtual machine.

Procedure 9.10. Creating a virtual SCSI controller

1. Display the configuration of the guest virtual machine (**Guest1**) and look for a pre-existing SCSI controller:

   ```
   # virsh dumpxml Guest1 | grep controller.*scsi
   ```

   If a device controller is present, the command will output one or more lines similar to the following:

   ```
   <controller type='scsi' model='virtio-scsi' index='0'/>
   ```

2. If the previous step did not show a device controller, create the description for one in a new file and add it to the virtual machine, using the following steps:

   a. Create the device controller by writing a **<controller>** element in a new file and save this file with an XML extension. **virtio-scsi-controller.xml**, for example.

   ```
   <controller type='scsi' model='virtio-scsi'/>
   ```

   b. Associate the device controller you just created in **virtio-scsi-controller.xml** with your guest virtual machine (Guest1, for example):

   ```
   # virsh attach-device --config Guest1 ~/virtio-scsi-controller.xml
   ```

   In this example the **--config** option behaves the same as it does for disks. Refer to Procedure 13.2, "Adding physical block devices to guests" for more information.

3. Add a new SCSI disk or CD-ROM. The new disk can be added using the methods in sections Section 13.3.1, "Adding File-based Storage to a Guest" and Section 13.3.2, "Adding Hard Drives and Other Block Devices to a Guest". In order to create a SCSI disk, specify a target device name that starts with *sd*.

   ```
   # virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img sdb --cache none
   ```

   Depending on the version of the driver in the guest virtual machine, the new disk may not be detected immediately by a running guest virtual machine. Follow the steps in the *Red Hat Enterprise Linux Storage Administration Guide*.

9.6. RANDOM NUMBER GENERATOR (RNG) DEVICE

virtio-rng is a virtual RNG (*random number generator*) device that feeds RNG data to the guest virtual machine's operating system, thereby providing fresh entropy for guest virtual machines on request.

Using an RNG is particularly useful when a device such as a keyboard, mouse and other inputs are not enough to generate sufficient entropy on the guest virtual machine. The virtio-rng device is available for both Red Hat Enterprise Linux and Windows guest virtual machines. Refer to the Note for instructions on installing the Windows requirements. Unless noted, the following descriptions are for both Red Hat Enterprise Linux and Windows guest virtual machines.

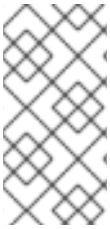
When **virtio-rng** is enabled on a Linux guest virtual machine, a chardev is created in the guest virtual machine at the location **/dev/hwrng/**. This chardev can then be opened and read to fetch entropy from the host physical machine. In order for guest virtual machines' applications to benefit from using randomness from the virtio-rng device transparently, the input from **/dev/hwrng/** must be relayed to the kernel entropy pool in the guest virtual machine. This can be accomplished if the information in this location is coupled with the rgnd daemon (contained within the rng-tools).

This coupling results in the entropy to be routed to the guest virtual machine's **/dev/random** file. The process is done manually in Red Hat Enterprise Linux 6 guest virtual machines.

Red Hat Enterprise Linux 6 guest virtual machines are coupled by running the following command:

```
# rngd -b -r /dev/hwrng/ -o /dev/random/
```

For more assistance, run the **man rngd** command for an explanation of the command options shown here. For further examples, refer to Procedure 9.11, "Implementing virtio-rng with the command line tools" for configuring the virtio-rng device.



NOTE

Windows guest virtual machines require the driver **viorng** to be installed. Once installed, the virtual RNG device will work using the CNG (crypto next generation) API provided by Microsoft. Once the driver is installed, the **virtrng** device appears in the list of RNG providers.

Procedure 9.11. Implementing virtio-rng with the command line tools

1. Shut down the guest virtual machine.
2. In a terminal window, using the **virsh edit *domain-name*** command, open the XML file for the desired guest virtual machine.
3. Edit the **<devices>** element to include the following:

```
...
<devices>
  <rng model='virtio'>
    <rate period="2000" bytes="1234"/>
    <backend model='random'>/dev/random</backend>
        <source mode='bind' service='1234'>
        <source mode='connect' host='192.0.2.1' service='1234'>
    </backend>
  </rng>
</devices>
...
```

CHAPTER 10. QEMU-IMG AND QEMU GUEST AGENT

This chapter contain useful hints and tips for using the qemu-img package with guest virtual machines. If you are looking for information on QEMU trace events and arguments, refer to the README file located here: **/usr/share/doc/qemu-*/README.systemtap**.

10.1. USING QEMU-IMG

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

Check

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

```
# qemu-img check -f qcow2 --output=qcow2 -r all filename-img.qcow2
```

NOTE

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

Using the ***-r*** tries to repair any inconsistencies that are found during the check, but when used with ***-r leaks*** cluster leaks are repaired and when used with ***-r all*** all kinds of errors are fixed. Note that this has a risk of choosing the wrong fix or hiding corruption issues that may have already occurred.

Commit

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

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

Convert

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

Command format:

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

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

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

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

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

Create

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

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

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

Preallocation is an option that may only be used with creating qcow2 images. Accepted values include **-o preallocation=*off*|*meta*|*full*|*falloc***. Images with preallocated metadata are larger than images without. However in cases where the image size increases, performance will improve as the image grows.

It should be noted that using **full** allocation can take a long time with large images. In cases where you want full allocation and time is of the essence, using **falloc** will save you time.

Info

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

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

This command is often used to discover the size reserved on disk which can be different from the displayed size. If snapshots are stored in the disk image, they are displayed also. This command will show for example, how much space is being taken by a **qcow2** image on a block device. This is done by running the **qemu-img**. You can check that the image in use is the one that matches the output of the **qemu-img info** command with the **qemu-img check** command. Refer to Section 10.1, “Using qemu-img”.

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

Map

The **# qemu-img map [-f *format*] [--output=*output_format*] filename** command dumps the metadata of the image filename and its backing file chain. Specifically, this command dumps the allocation state of every sector of a specified file, together with the topmost file that allocates it in the backing file chain. For example, if you have a chain such as c.qcow2 → b.qcow2 → a.qcow2, a.qcow is the original file, b.qcow2 is the changes made to a.qcow2 and c.qcow2 is the delta file from b.qcow2. When this chain is created the image files stores the normal image data, plus information about what is in which file and where it is located within the file. This information is referred to as the image's metadata. The **-f** format option is the format of the specified image file. Formats such as raw, qcow2, vhdx and vmdk may be used. There are two output options possible: **human** and **json**.

- **human** is the default setting. It is designed to be more readable to the human eye, and as such, this format should not be parsed. For clarity and simplicity, the default **human** format only dumps known-nonzero areas of the file. Known-zero parts of the file are omitted altogether, and likewise for parts that are not allocated throughout the chain. When the command is executed, qemu-img output will identify a file from where the data can be read, and the offset in the file. The output is displayed as a table with four columns; the first three of which are hexadecimal numbers.

```
# qemu-img map -f qcow2 --output=human /tmp/test.qcow2
Offset          Length          Mapped to       File
0               0x20000         0x50000         /tmp/test.qcow2
0x100000        0x80000         0x70000         /tmp/test.qcow2
0x200000        0x1f0000        0xf0000         /tmp/test.qcow2
0x3c00000       0x20000         0x2e0000        /tmp/test.qcow2
0x3fd0000       0x10000         0x300000        /tmp/test.qcow2
```

- **json**, or *JSON* (JavaScript Object Notation), is readable by humans, but as it is a programming language, it is also designed to be parsed. For example, if you want to parse the output of "qemu-img map" in a parser then you should use the option **--output=json**.

```
# qemu-img map -f qcow2 --output=json /tmp/test.qcow2
[{ "start": 0, "length": 131072, "depth": 0, "zero": false, "data": true, "offset": 327680},
{ "start": 131072, "length": 917504, "depth": 0, "zero": true, "data": false},
```

For more information on the JSON format, refer to the qemu-img(1) man page.

Rebase

Changes the backing file of an image.

```
# qemu-img rebase [-f format] [-t cache] [-p] [-u] -b backing_file [-F backing_format] filename
```

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

NOTE

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

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

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

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

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

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

Resize

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

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

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

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

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

WARNING

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

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

Snapshot

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

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

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

Supported Formats

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

raw

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

qcow2

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

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

bochs

Bochs disk image format.

cloop

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

cow

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

dmg

Mac disk image format.

nbd

Network block device.

parallels

Parallels virtualization disk image format.

qcow

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

vdi

Oracle VM VirtualBox hard disk image format.

vmdk

VMware compatible image format (read-write support for versions 1 and 2, and read-only support for version 3).

vpc

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

vvfat

Virtual VFAT disk image format.

10.2. QEMU GUEST AGENT

The QEMU guest agent runs inside the guest and allows the host machine to issue commands to the guest operating system using libvirt. The guest operating system then responds to those commands asynchronously. This chapter covers the libvirt commands and options available to the guest agent.

IMPORTANT

Note that it is only safe to rely on the guest agent when run by trusted guests. An untrusted guest may maliciously ignore or abuse the guest agent protocol, and although built-in safeguards exist to prevent a denial of service attack on the host, the host requires guest co-operation for operations to run as expected.

Note that QEMU guest agent can be used to enable and disable virtual CPUs (vCPUs) while the guest is running, thus adjusting the number of vCPUs without using the hot plug and hot unplug features. Refer to Section 14.13.6, "Configuring Virtual CPU Count" for more information.

10.2.1. Install and Enable the Guest Agent

Install qemu-guest-agent on the guest virtual machine with the **yum install qemu-guest-agent** command and make it run automatically at every boot as a service (qemu-guest-agent.service).

10.2.2. Setting up Communication between Guest Agent and Host

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

NOTE

For instructions on how to set up the QEMU guest agent on Windows guests, refer to the instructions found here.

Procedure 10.1. Setting up communication between guest agent and host

1. **Open the guest XML**
   Open the guest XML with the QEMU guest agent configuration. You will need the guest name to open the file. Use the command **# virsh list** on the host machine to list the guests that it can recognize. In this example, the guest's name is *rhel6*:

   ```
   # virsh edit rhel6
   ```

2. **Edit the guest XML file**
   Add the following elements to the XML file and save the changes.

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/rhel6.agent'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

Figure 10.1. Editing the guest XML to configure the QEMU guest agent

3. **Start the QEMU guest agent in the guest**
   Download and install the guest agent in the guest virtual machine using **yum install qemu-guest-agent** if you have not done so already. Once installed, start the service as follows:

   ```
   # service start qemu-guest-agent
   ```

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

10.2.3. Using the QEMU Guest Agent

The QEMU guest agent protocol (QEMU GA) package, qemu-guest-agent, is fully supported in Red Hat Enterprise Linux 6.5 and newer. However, there are the following limitations with regards to isa-serial/virtio-serial transport:

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

10.2.4. Using the QEMU Guest Agent with libvirt

Installing the QEMU guest agent allows various other libvirt commands to become more powerful. The guest agent enhances the following **virsh** commands:

- **virsh shutdown --mode=agent** - This shutdown method is more reliable than **virsh shutdown --mode=acpi**, as **virsh shutdown** used with the QEMU guest agent is guaranteed to shut down a cooperative guest in a clean state. If the agent is not present, libvirt has to instead rely on injecting an ACPI shutdown event, but some guests ignore that event and thus will not shut down.

  Can be used with the same syntax for **virsh reboot**.

- **virsh snapshot-create --quiesce** - Allows the guest to flush its I/O into a stable state before the snapshot is created, which allows use of the snapshot without having to perform a fsck or losing partial database transactions. The guest agent allows a high level of disk contents stability by providing guest co-operation.
- **virsh setvcpus --guest** - Instructs the guest to take CPUs offline.

- **virsh dompmsuspend** - Suspends a running guest gracefully using the guest operating system's power management functions.

10.2.5. Creating a Guest Virtual Machine Disk Backup

libvirt can communicate with qemu-ga to assure that snapshots of guest virtual machine file systems are consistent internally and ready for use on an as needed basis. Improvements in Red Hat Enterprise Linux 6 have been made to make sure that both file and application level synchronization (flushing) is done. Guest system administrators can write and install application-specific freeze/thaw hook scripts. Before freezing the filesystems, the qemu-ga invokes the main hook script (included in the qemu-ga package). The freezing process temporarily deactivates all guest virtual machine applications.

Just before filesystems are frozen, the following actions occur:

- File system applications / databases flush working buffers to the virtual disk and stop accepting client connections
- Applications bring their data files into a consistent state
- Main hook script returns
- qemu-ga freezes the filesystems and management stack takes a snapshot
- Snapshot is confirmed
- Filesystem function resumes

Thawing happens in reverse order.

Use the **snapshot-create-as** command to create a snapshot of the guest disk. See Section 14.15.2.2, "Creating a snapshot for the current domain" for more details on this command.

NOTE

An application-specific hook script might need various SELinux permissions in order to run correctly, as is done when the script needs to connect to a socket in order to talk to a database. In general, local SELinux policies should be developed and installed for such purposes. Accessing file system nodes should work out of the box, after issuing the **restorecon -FvvR** command listed in Table 10.1, "QEMU guest agent package contents" in the table row labeled **/etc/qemu-ga/fsfreeze-hook.d/**.

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

Table 10.1. QEMU guest agent package contents

File name	Description
/etc/rc.d/init.d/qemu-ga	Service control script (start/stop) for the QEMU guest agent.
/etc/sysconfig/qemu-ga	Configuration file for the QEMU guest agent, as it is read by the **/etc/rc.d/init.d/qemu-ga** control script. The settings are documented in the file with shell script comments.

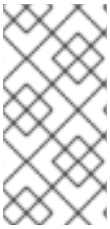
File name	Description
/usr/bin/qemu-ga	QEMU guest agent binary file.
/usr/libexec/qemu-ga/	Root directory for hook scripts.
/usr/libexec/qemu-ga/fsfreeze-hook	Main hook script. No modifications are needed here.
/usr/libexec/qemu-ga/fsfreeze-hook.d/	Directory for individual, application-specific hook scripts. The guest system administrator should copy hook scripts manually into this directory, ensure proper file mode bits for them, and then run **restorecon -FvvR** on this directory.
/usr/share/qemu-kvm/qemu-ga/	Directory with sample scripts (for example purposes only). The scripts contained here are not executed.

The main hook script, **/usr/libexec/qemu-ga/fsfreeze-hook** logs its own messages, as well as the application-specific script's standard output and error messages, in the following log file: **/var/log/qemu-ga/fsfreeze-hook.log**. For more information, refer to the qemu-guest-agent wiki page at wiki.qemu.org or libvirt.org.

10.3. RUNNING THE QEMU GUEST AGENT ON A WINDOWS GUEST

A Red Hat Enterprise Linux host machine can issue commands to Windows guests by running the QEMU guest agent in the guest. This is supported in hosts running Red Hat Enterprise Linux 6.5 and newer, and in the following Windows guest operating systems:

- Windows XP Service Pack 3 (VSS is not supported)
- Windows Server 2003 R2 - x86 and AMD64 (VSS is not supported)
- Windows Server 2008
- Windows Server 2008 R2
- Windows 7 - x86 and AMD64
- Windows Server 2012
- Windows Server 2012 R2
- Windows 8 - x86 and AMD64
- Windows 8.1 - x86 and AMD64



NOTE

Windows guest virtual machines require the QEMU guest agent package for Windows, qemu-guest-agent-win. This agent is required for VSS (Volume Shadow Copy Service) support for Windows guest virtual machines running on Red Hat Enterprise Linux. More information can be found here.

Procedure 10.2. Configuring the QEMU guest agent on a Windows guest

Follow these steps for Windows guests running on a Red Hat Enterprise Linux host machine.

1. **Prepare the Red Hat Enterprise Linux host machine**
   Make sure the following package is installed on the Red Hat Enterprise Linux host physical machine:

   - virtio-win, located in **/usr/share/virtio-win/**

   To copy the drivers in the Windows guest, make an ***.iso** file for the **qxl** driver using the following command:

   ```
   # mkisofs -o /var/lib/libvirt/images/virtiowin.iso /usr/share/virtio-win/drivers
   ```

2. **Prepare the Windows guest**
   Install the virtio-serial driver in guest by mounting the ***.iso** to the Windows guest in order to update the driver. Start the guest, then attach the driver .iso file to the guest as shown (using a disk named *hdb*):

   ```
   # virsh attach-disk guest /var/lib/libvirt/images/virtiowin.iso hdb
   ```

   To install the drivers using the Windows **Control Panel**, navigate to the following menus:

   - To install the virtio-win driver - Select **Hardware and Sound** > **Device manager** > **virtio-serial driver**.

3. **Update the Windows guest XML configuration file**
   The guest XML file for the Windows guest is located on the Red Hat Enterprise Linux host machine. To gain access to this file, you need the Windows guest name. Use the **# virsh list** command on the host machine to list the guests that it can recognize. In this example, the guest's name is win7x86.

   Add the following elements to the XML file using the **# virsh edit *win7x86*** command and save the changes. Note that the source socket name must be unique in the host, named *win7x86.agent* in this example:

   ```
   ...
   <channel type='unix'>
     <source mode='bind' path='/var/lib/libvirt/qemu/win7x86.agent'/>
     <target type='virtio' name='org.qemu.guest_agent.0'/>
     <address type='virtio-serial' controller='0' bus='0' port='1'/>
   </channel>
   <channel type='spicevmc'>
     <target type='virtio' name='com.redhat.spice.0'/>
     <address type='virtio-serial' controller='0' bus='0' port='2'/>
   </channel>
   ...
   ```

   Figure 10.2. Editing the Windows guest XML to configure the QEMU guest agent

4. **Reboot the Windows guest**
   Reboot the Windows guest to apply the changes:

```
# virsh reboot win7x86
```

5. **Prepare the QEMU guest agent in the Windows guest**
   To prepare the guest agent in a Windows guest:

   a. **Install the latestvirtio-win package**
   Run the following command on the Red Hat Enterprise Linux host physical machine terminal window to locate the file to install. Note that the file shown below may not be exactly the same as the one your system finds, but it should be latest official version.

   ```
   # rpm -qa|grep virtio-win
   virtio-win-1.6.8-5.el6.noarch

   # rpm -iv virtio-win-1.6.8-5.el6.noarch
   ```

   b. **Confirm the installation completed**
   After the virtio-win package finishes installing, check the **/usr/share/virtio-win/guest-agent/** folder and you will find an file named qemu-ga-x64.msi or the qemu-ga-x86.msi as shown:

   ```
   # ls -l /usr/share/virtio-win/guest-agent/

   total 1544

   -rw-r--r--. 1 root root 856064 Oct 23 04:58 qemu-ga-x64.msi

   -rw-r--r--. 1 root root 724992 Oct 23 04:58 qemu-ga-x86.msi
   ```

   c. **Install the .msi file**
   From the Windows guest (win7x86, for example) install the qemu-ga-x64.msi or the qemu-ga-x86.msi by double clicking on the file. Once installed, it will be shown as a qemu-ga service in the Windows guest within the System Manager. This same manager can be used to monitor the status of the service.

10.3.1. Using libvirt Commands with the QEMU Guest Agent on Windows Guests

The QEMU guest agent can use the following **virsh** commands with Windows guests:

- **virsh shutdown --mode=agent** - This shutdown method is more reliable than **virsh shutdown --mode=acpi**, as **virsh shutdown** used with the QEMU guest agent is guaranteed to shut down a cooperative guest in a clean state. If the agent is not present, libvirt has to instead rely on injecting an ACPI shutdown event, but some guests ignore that event and thus will not shut down.

  Can be used with the same syntax for **virsh reboot**.

- **virsh snapshot-create --quiesce** - Allows the guest to flush its I/O into a stable state before the snapshot is created, which allows use of the snapshot without having to perform a fsck or losing partial database transactions. The guest agent allows a high level of disk contents stability by providing guest co-operation.

- **virsh dompmsuspend** - Suspends a running guest gracefully using the guest operating system's power management functions.

10.4. SETTING A LIMIT ON DEVICE REDIRECTION

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

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

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

Example 10.1. Limiting redirection with a Windows guest virtual machine

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

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

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

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

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

4. Plug a USB device into a host physical machine, and use **virt-viewer** to connect to the guest virtual machine.
5. Click **USB device selection** in the menu, which will produce the following message: "Some USB devices are blocked by host policy". Click **OK** to confirm and continue.

   The filter takes effect.
6. To make sure that the filter captures properly check the USB device vendor and product, then make the following changes in the host physical machine's domain XML to allow for USB redirection.

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

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

10.5. DYNAMICALLY CHANGING A HOST PHYSICAL MACHINE OR A NETWORK BRIDGE THAT IS ATTACHED TO A VIRTUAL NIC

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

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

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

2. Prepare an XML file for interface update:

```
# cat br1.xml
```

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

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

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

vnet0
virbr1          8000.525400682996       yes             virbr1-nic
```

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

```
# virsh update-device test1 br1.xml

Device updated successfully
```

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

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

CHAPTER 11. STORAGE CONCEPTS

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

11.1. STORAGE POOLS

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

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

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

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

NOTE

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

11.2. VOLUMES

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

Referencing Volumes

To reference a specific volume, three approaches are possible:

The name of the volume and the storage pool

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

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

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

virsh #
```

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

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

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

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

The unique volume key

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

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

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

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

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

For example, a volume with the volume key of *Wlvnf7-a4a3-Tlje-lJDa-9eak-PZBv-LoZuUr*:

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

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

vol-name

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

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

vol-path

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

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

The vol-key command

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

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

CHAPTER 12. STORAGE POOLS

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

Example 12.1. NFS storage pool

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

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

NOTE

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

WARNING

When creating storage pools on a guest, make sure to follow security considerations. This information is discussed in more detail in the *Red Hat Enterprise Linux Virtualization Security Guide* which can be found at https://access.redhat.com/site/documentation/.

12.1. DISK-BASED STORAGE POOLS

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

> **WARNING**
>
> Guests should not be given write access to whole disks or block devices (for example, **/dev/sdb**). Use partitions (for example, **/dev/sdb1**) or LVM volumes.
>
> If you pass an entire block device to the guest, the guest will likely partition it or create its own LVM groups on it. This can cause the host physical machine to detect these partitions or LVM groups and cause errors.

12.1.1. Creating a Disk-based Storage Pool Using virsh

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

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

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

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

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

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

   <name>guest_images_disk</name>

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

   <device path='*/dev/sdb*'/>

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

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

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

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

<format type='*gpt*'/>

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

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

Example 12.2. Disk based storage device storage pool

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

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

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

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

```
# virsh pool-start guest_images_disk
Pool guest_images_disk started
# virsh pool-list --all
Name                 State      Autostart
```

```
------------------------------------------
default              active     yes
guest_images_disk    active     no
```

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

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

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

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

7. **Optional: Remove the temporary configuration file**
   Remove the temporary storage pool XML configuration file if it is not needed.

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

A disk based storage pool is now available.

12.1.2. Deleting a Storage Pool Using virsh

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

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

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

2. Remove the storage pool's definition

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

12.2. PARTITION-BASED STORAGE POOLS

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

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

12.2.1. Creating a Partition-based Storage Pool Using virt-manager

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

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

1. **Open the storage pool settings**

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

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

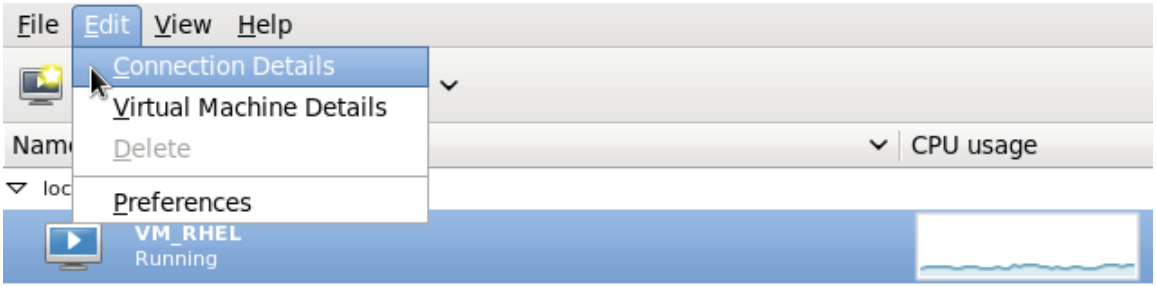


   Figure 12.1. Connection Details

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

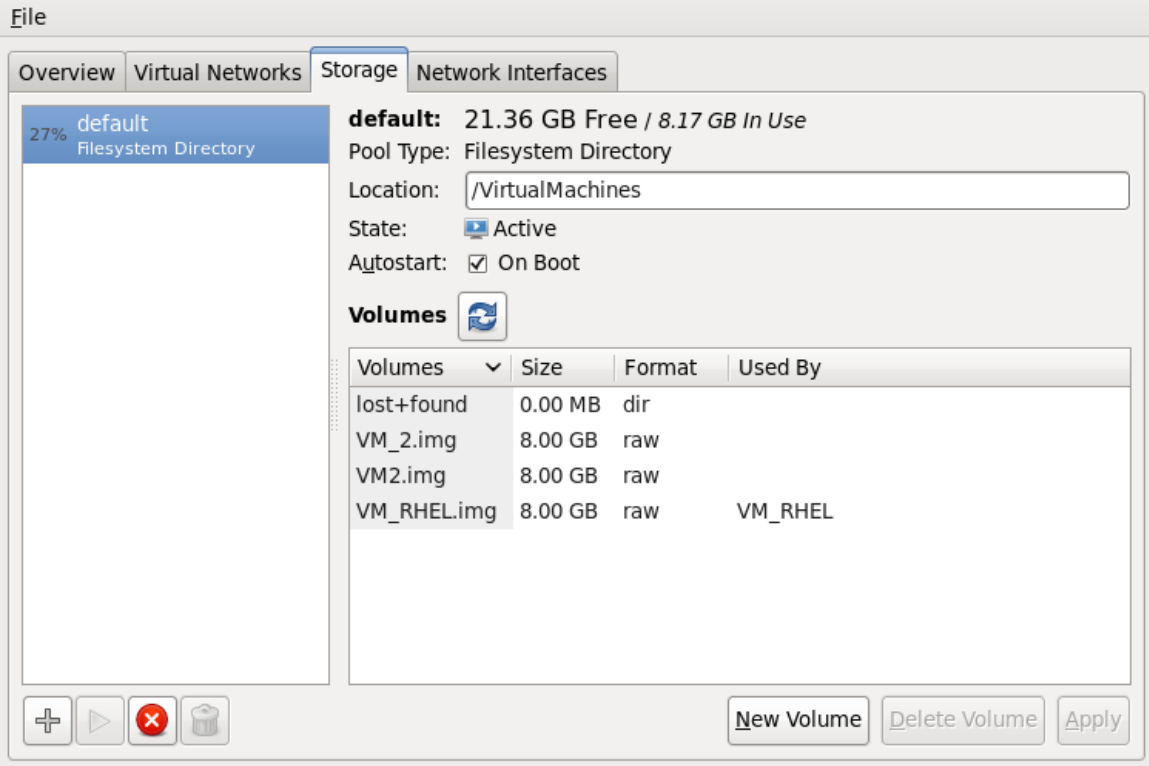


Figure 12.2. Storage tab

2. **Create the new storage pool**

   a. **Add a new pool (part 1)**
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

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

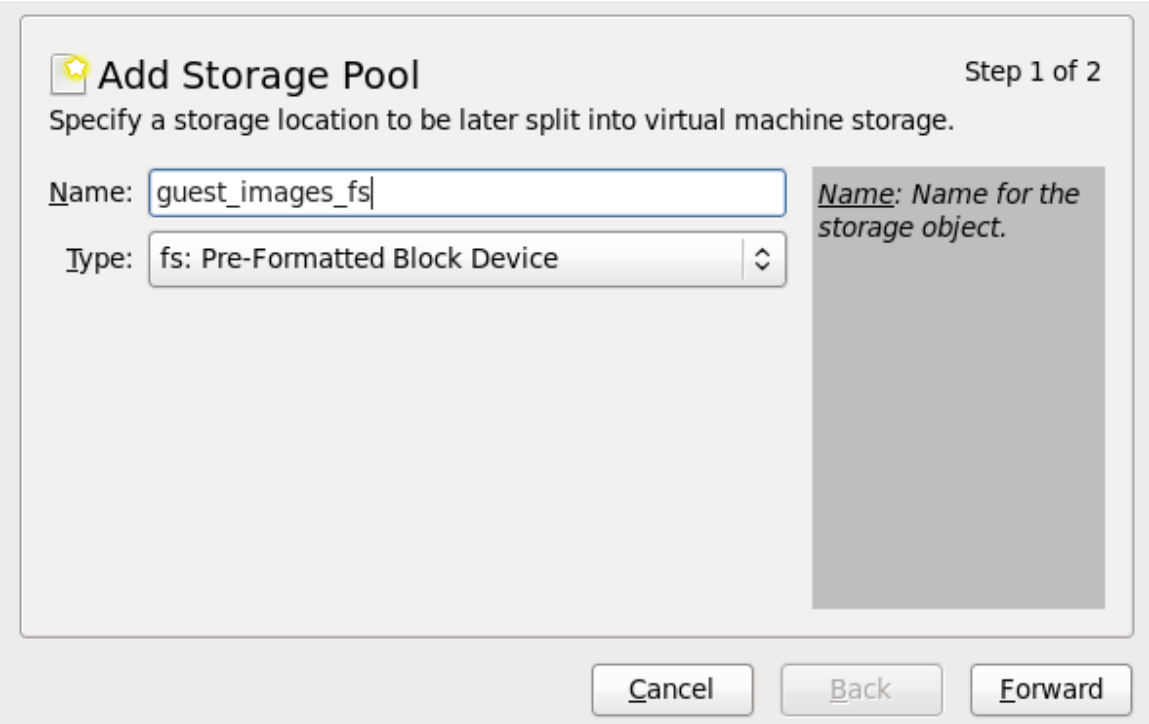


Figure 12.3. Storage pool name and type

Press the **Forward** button to continue.

b. **Add a new pool (part 2)**
Change the **Target Path**, **Format**, and **Source Path** fields.

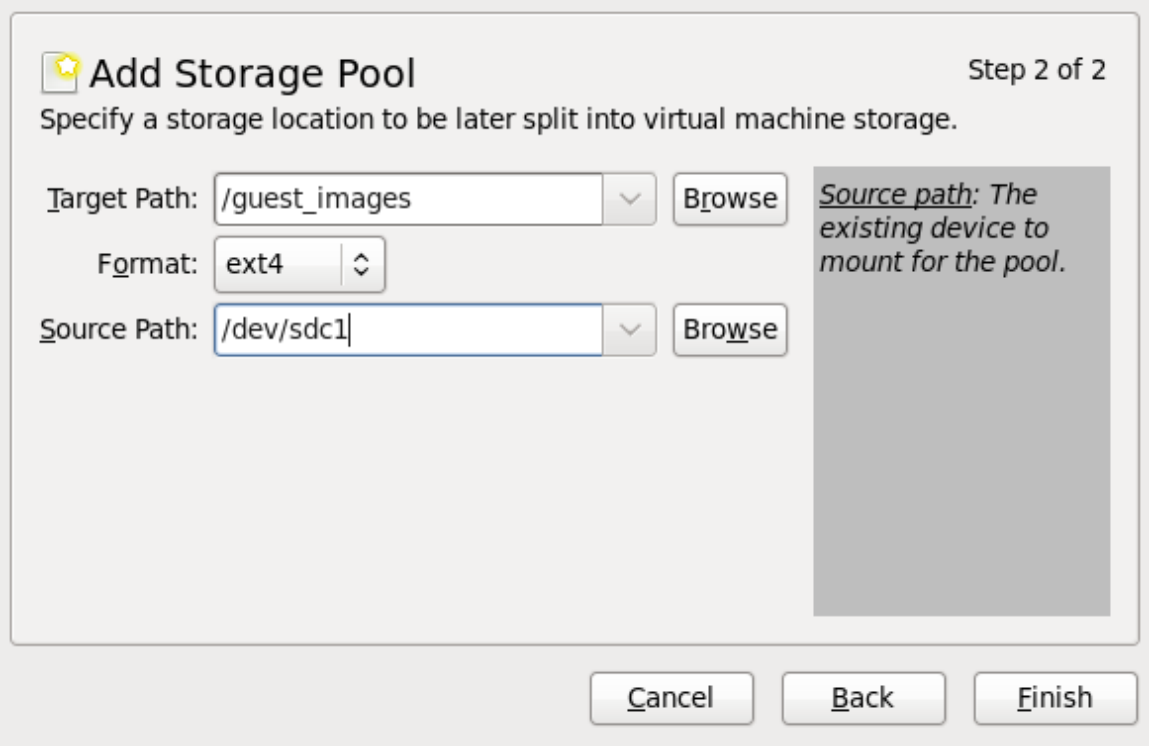


Figure 12.4. Storage pool path and format

Target Path

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

Format

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

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

Source Path

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

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

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

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

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

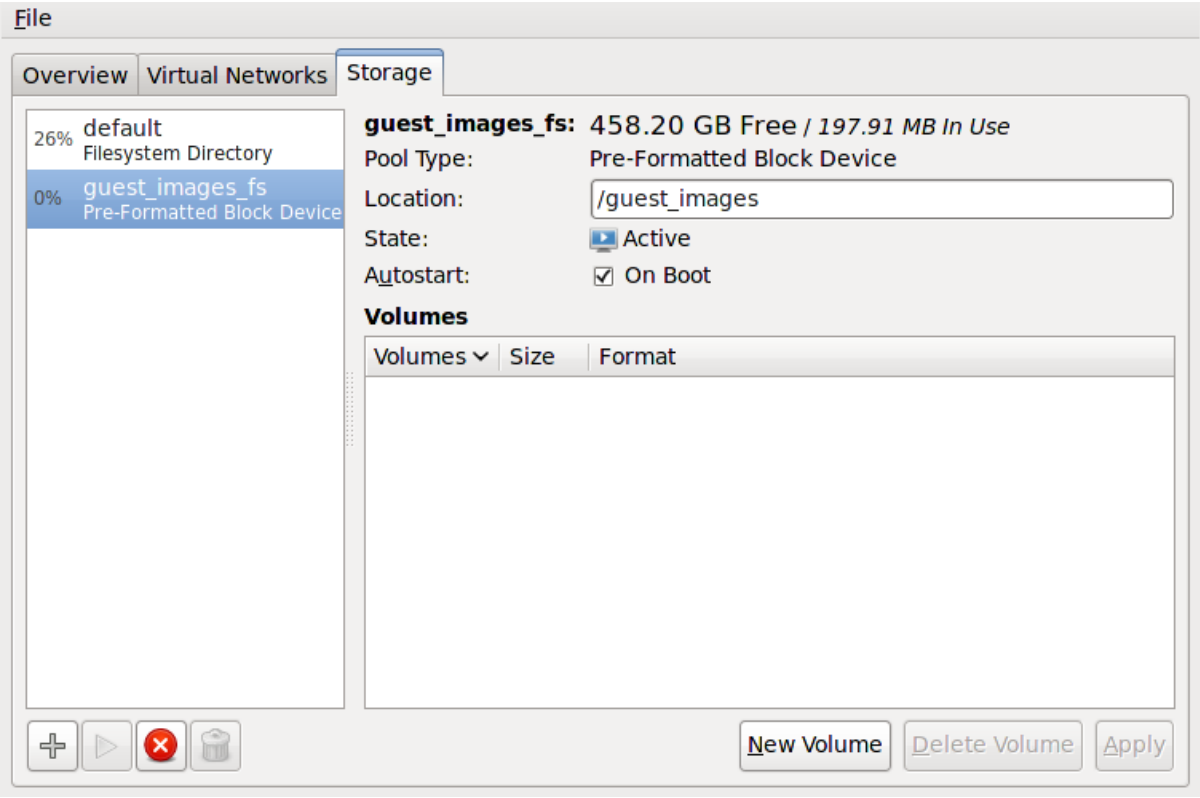


Figure 12.5. Storage list confirmation

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

12.2.2. Deleting a Storage Pool Using virt-manager

This procedure demonstrates how to delete a storage pool.

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

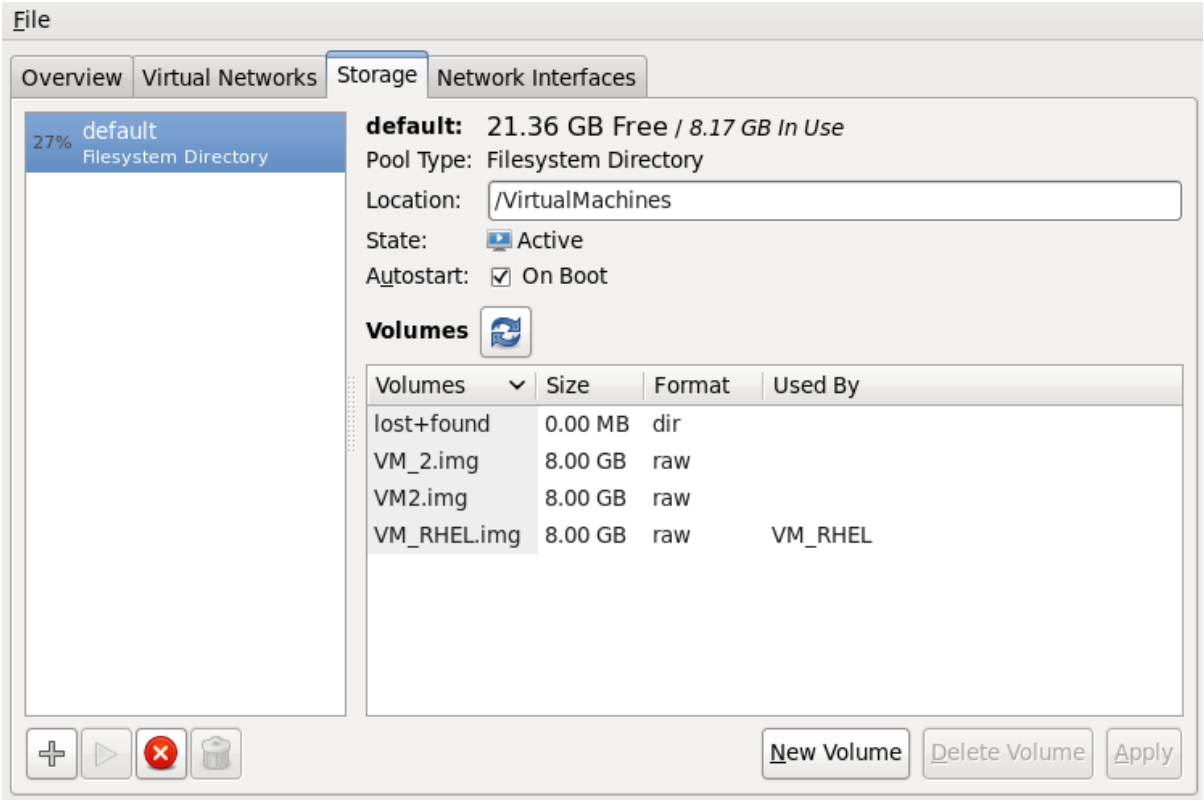


Figure 12.6. Stop Icon

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

12.2.3. Creating a Partition-based Storage Pool Using virsh

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

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

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

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

   Partition name

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

   device

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

mountpoint

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

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

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

The new pool and mount points are now created.

2. **Verify the new pool**
   List the present storage pools.

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

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

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

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

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

5. **Turn on autostart**

By default, a storage pool defined with **virsh**, is not set to automatically start each time **libvirtd** starts. To remedy this, enable the automatic start with the **virsh pool-autostart** command. The storage pool is now automatically started each time **libvirtd** starts.

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

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

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

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

12.2.4. Deleting a Storage Pool Using virsh

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

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

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

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

3. Remove the storage pool's definition

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

12.3. DIRECTORY-BASED STORAGE POOLS

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

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

12.3.1. Creating a Directory-based Storage Pool with virt-manager

1. **Create the local directory**

   a. **Optional: Create a new directory for the storage pool**
   Create the directory on the host physical machine for the storage pool. This example uses a directory named */guest virtual machine_images* .

   ```
   # mkdir /guest_images
   ```

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

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

   c. **Set directory permissions**
   Change the file permissions of the directory.

   ```
   # chmod 700 /guest_images
   ```

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

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

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

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

3. **Open the storage pool settings**

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

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

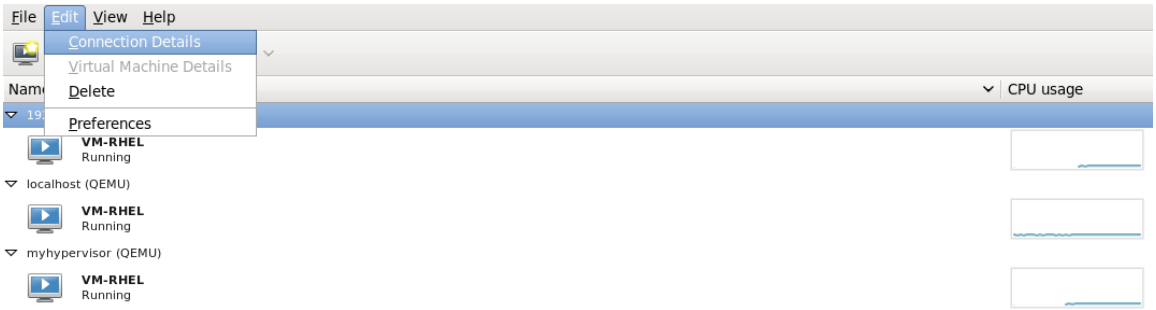


Figure 12.7. Connection details window

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

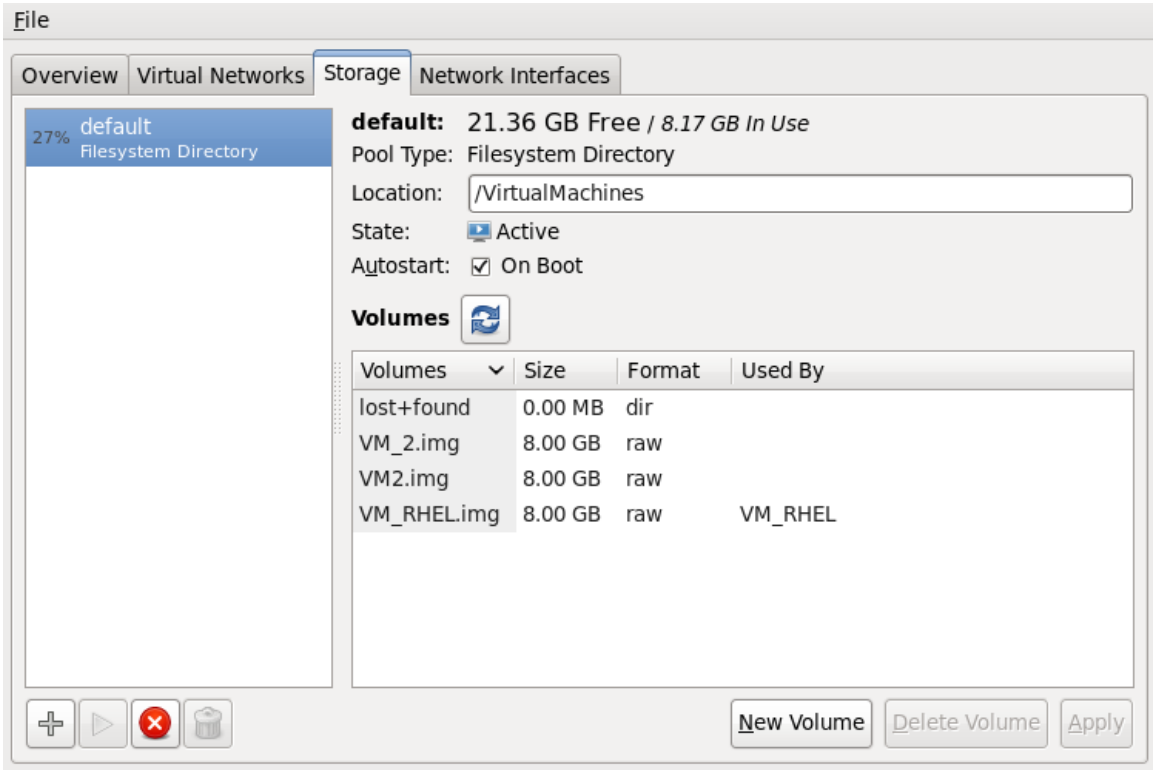


Figure 12.8. Storage tab

4. **Create the new storage pool**

   a. **Add a new pool (part 1)**
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

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

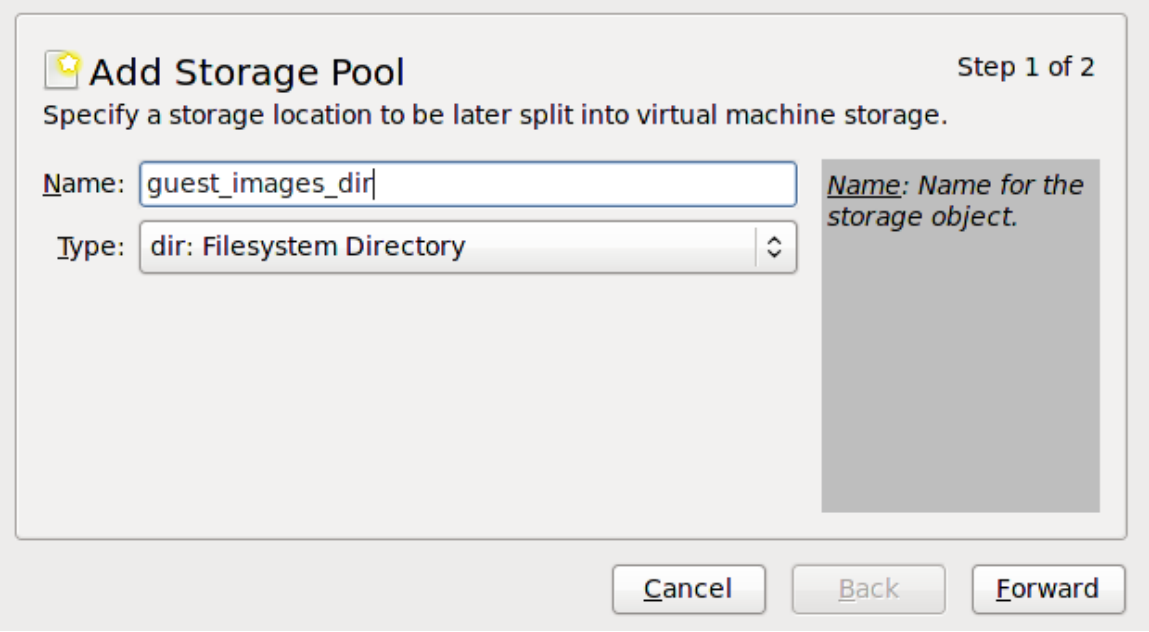


Figure 12.9. Name the storage pool

Press the **Forward** button to continue.

b. **Add a new pool (part 2)**
Change the **Target Path** field. For example, */guest_images*.

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

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

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

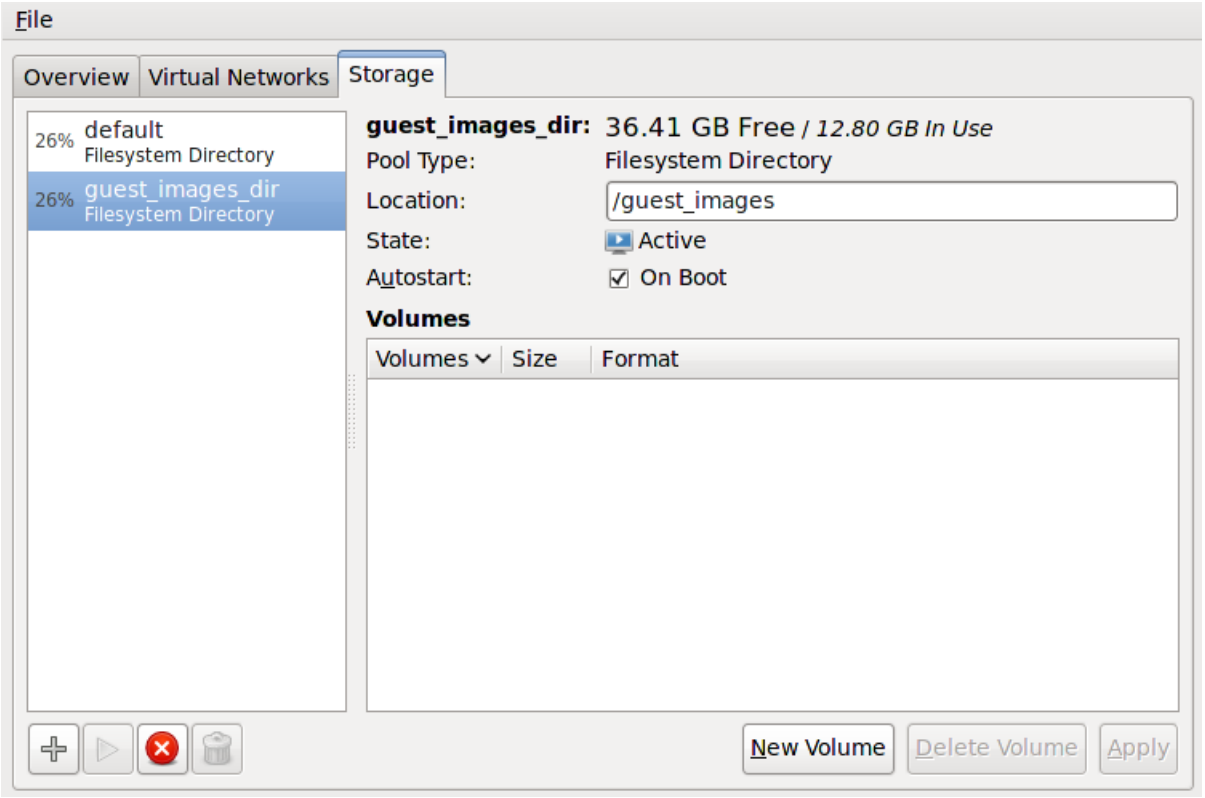


Figure 12.10. Verify the storage pool information

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

12.3.2. Deleting a Storage Pool Using virt-manager

This procedure demonstrates how to delete a storage pool.

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

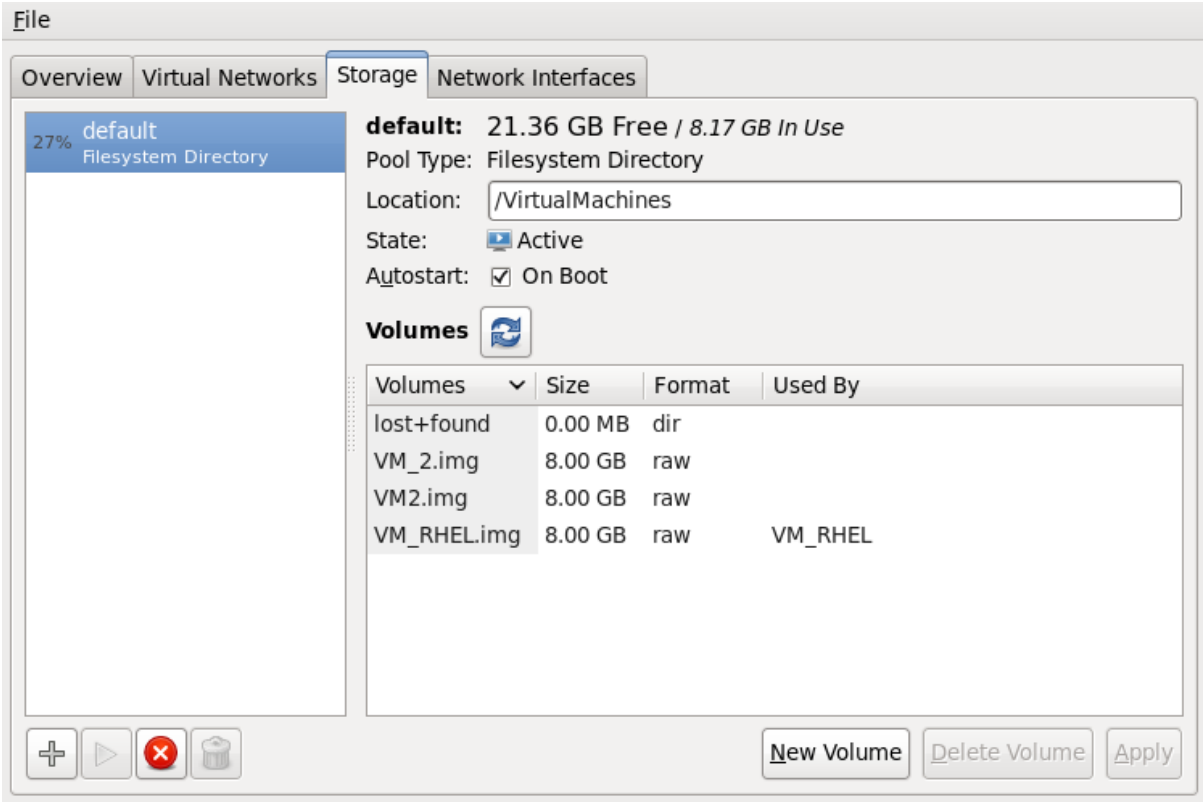


Figure 12.11. Stop Icon

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

12.3.3. Creating a Directory-based Storage Pool with virsh

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

   - The **name** of the storage pool.

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

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

     This example uses the */guest_images* directory.

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

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

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

–

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

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

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

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

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

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

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

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

```
Allocation:     12.80 GB
Available:      36.41 GB

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

A directory-based storage pool is now available.

12.3.4. Deleting a Storage Pool Using virsh

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

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

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

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

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

3. Remove the storage pool's definition

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

12.4. LVM-BASED STORAGE POOLS

This chapter covers using LVM volume groups as storage pools.

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

NOTE

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

NOTE

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

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

12.4.1. Creating an LVM-based Storage Pool with virt-manager

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

1. **Optional: Create new partition for LVM volumes**
   These steps describe how to create a new partition and LVM volume group on a new hard disk drive.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

i. write changes to disk and quit.

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

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

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

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

2. **Open the storage pool settings**

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

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

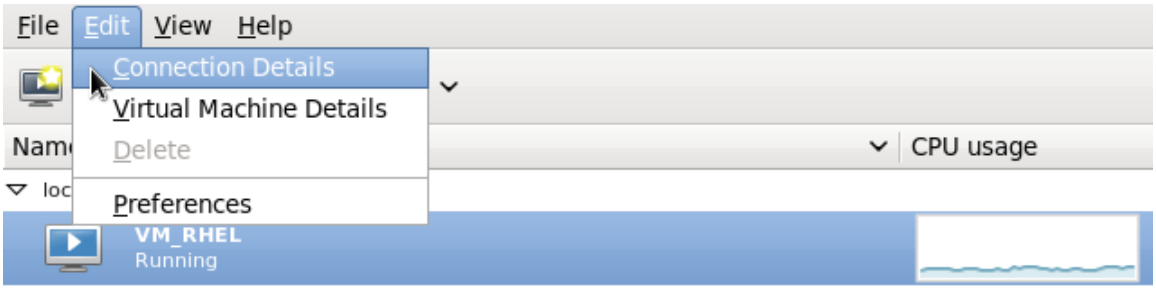


Figure 12.12. Connection details

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

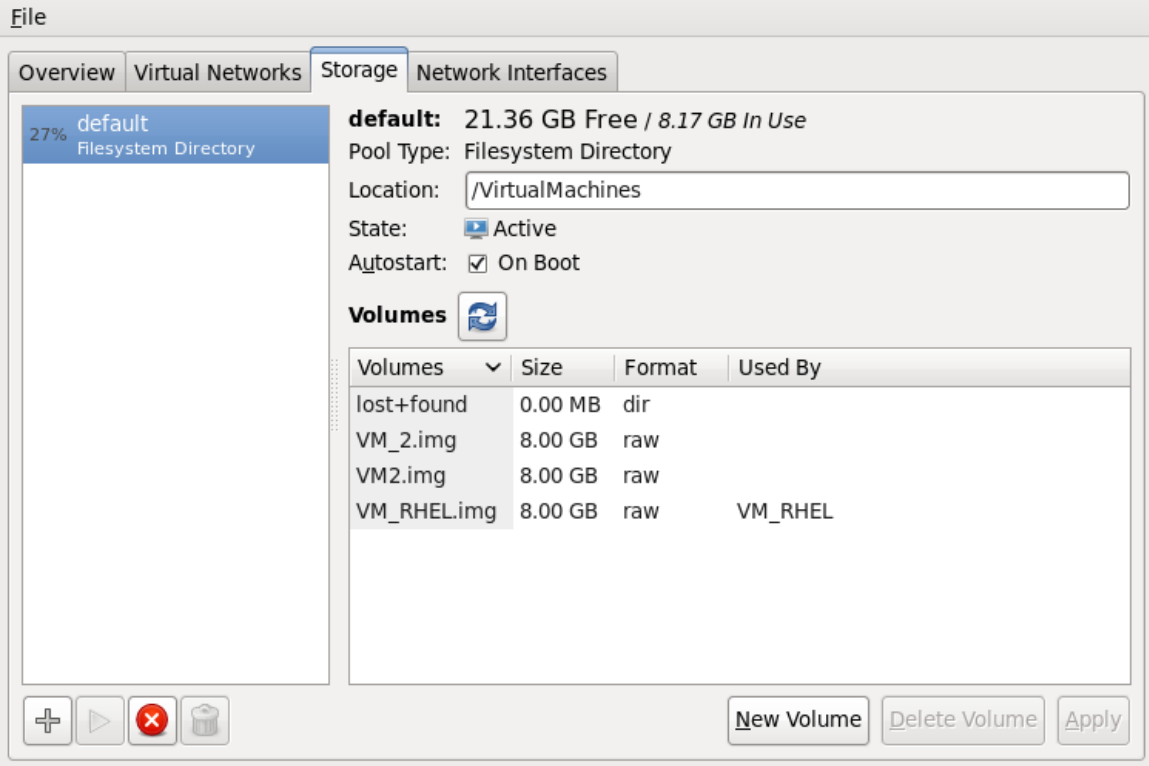


Figure 12.13. Storage tab

3. **Create the new storage pool**

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

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

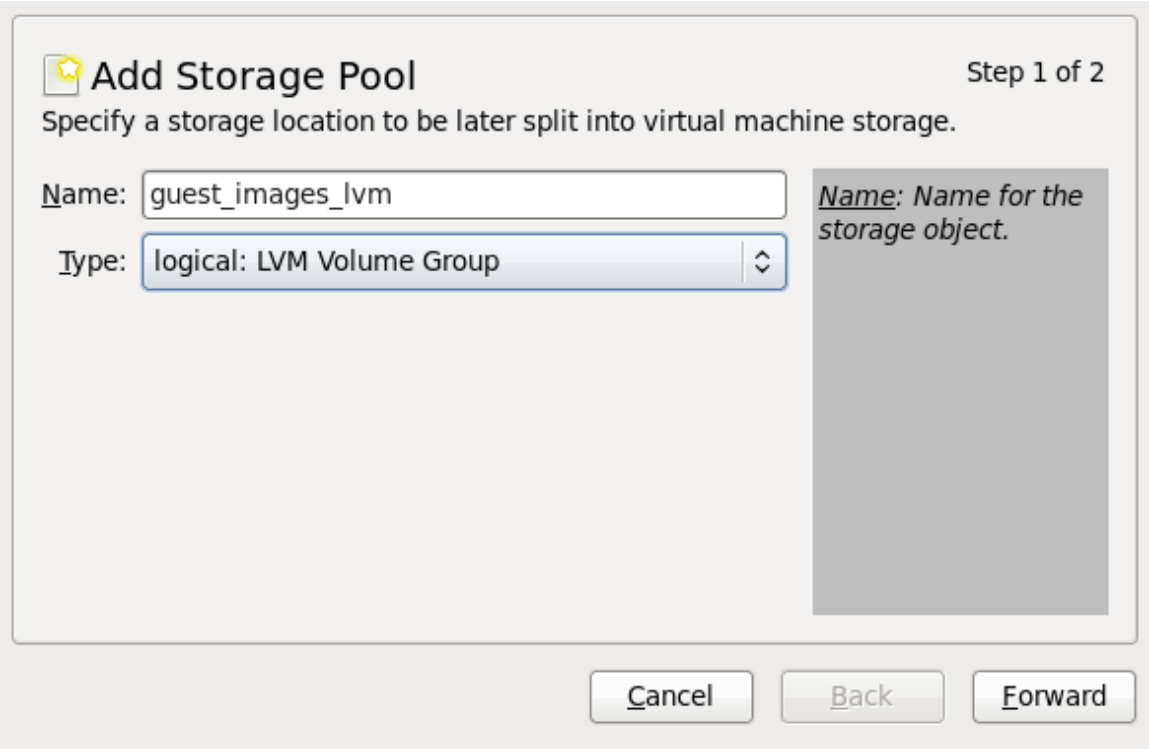


Figure 12.14. Add LVM storage pool

Press the **Forward** button to continue.

b. **Add a new pool (part 2)**
Change the **Target Path** field. This example uses */guest_images*.

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

- Use the **Target Path** field to *either* select an existing LVM volume group or as the name for a new volume group. The default format is /**dev**/*storage_pool_name*.

  This example uses a new volume group named */dev/guest_images_lvm*.

- The **Source Path** field is optional if an existing LVM volume group is used in the **Target Path**.

  For new LVM volume groups, input the location of a storage device in the **Source Path** field. This example uses a blank partition */dev/sdc*.

- The **Build Pool** check box instructs **virt-manager** to create a new LVM volume group. If you are using an existing volume group you should not select the **Build Pool** check box.

  This example is using a blank partition to create a new volume group so the **Build Pool** check box must be selected.

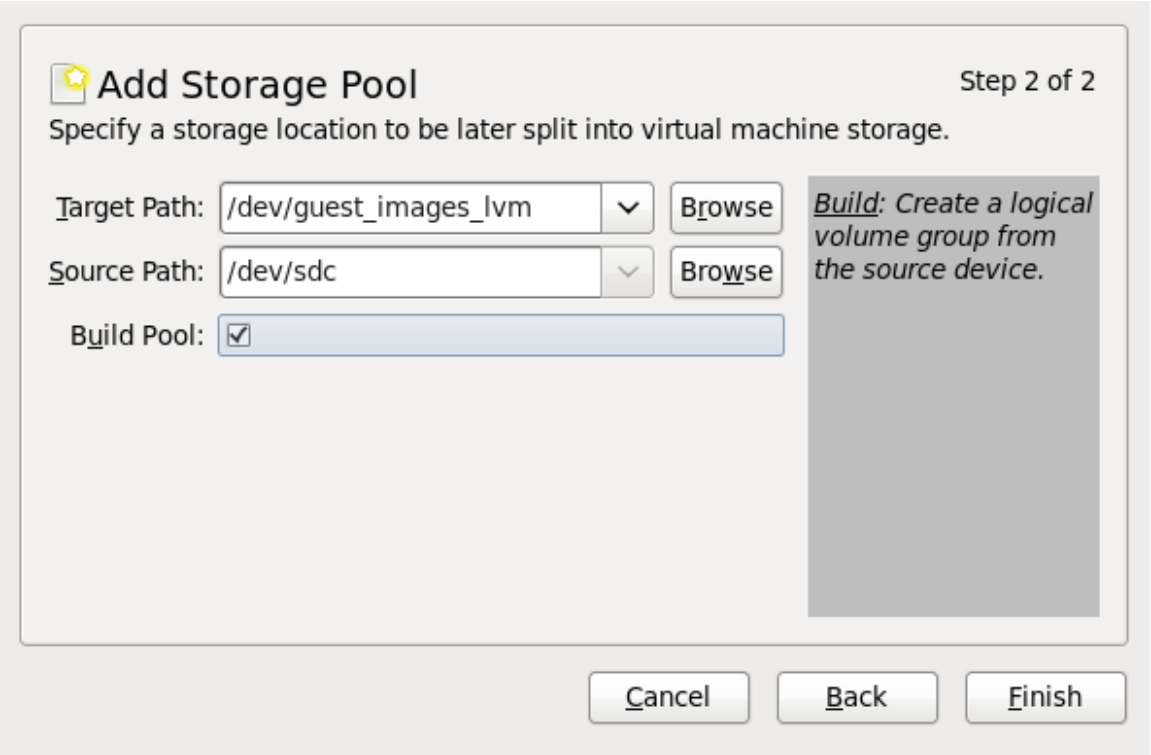


Figure 12.15. Add target and source

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

c. **Confirm the device to be formatted**
A warning message appears.

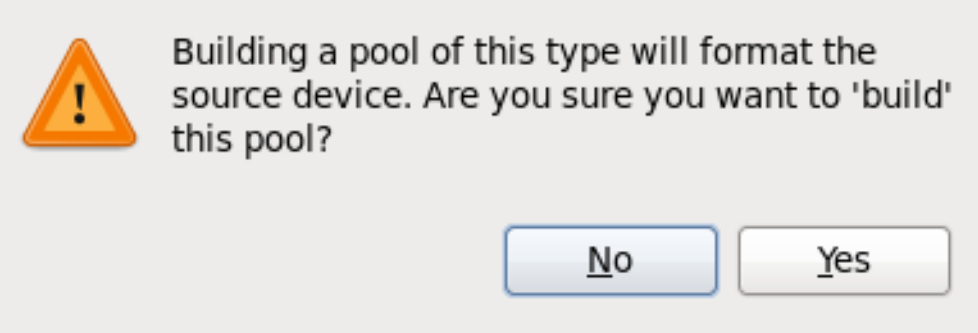


Figure 12.16. Warning message

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

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

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

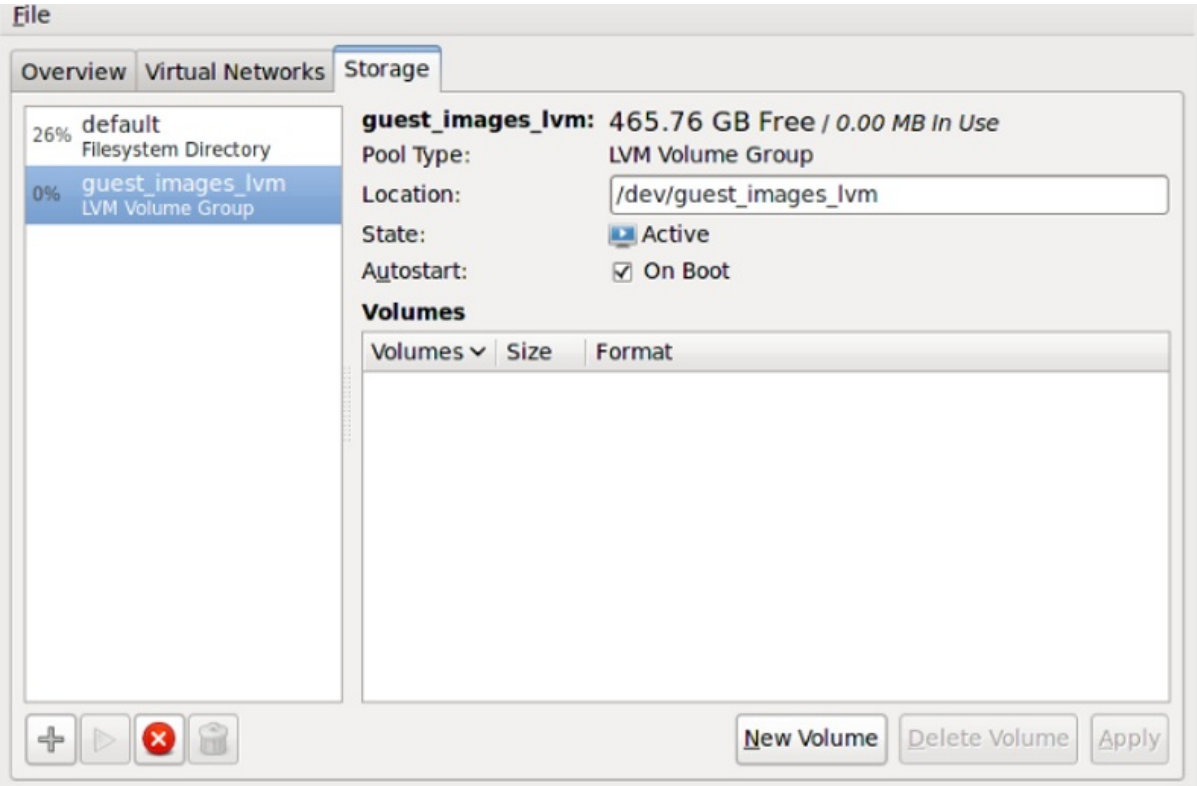


Figure 12.17. Confirm LVM storage pool details

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

12.4.2. Deleting a Storage Pool Using virt-manager

This procedure demonstrates how to delete a storage pool.

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

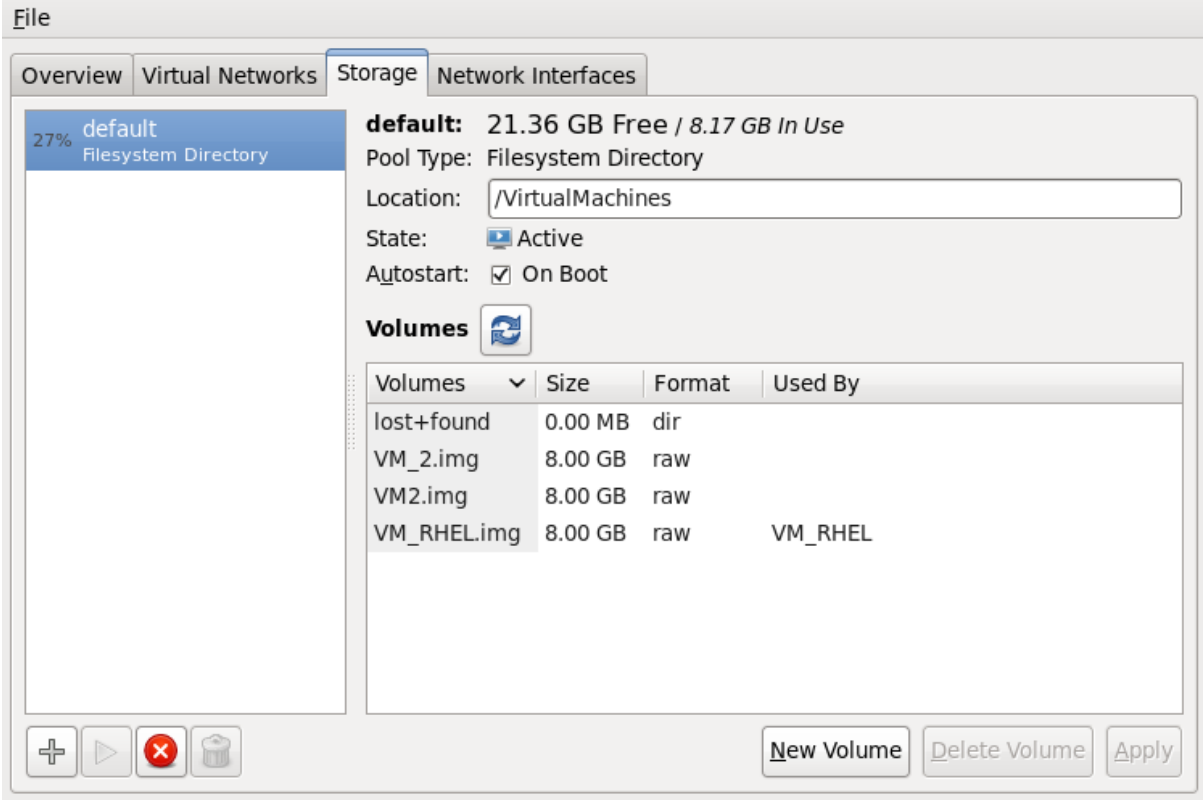


Figure 12.18. Stop Icon

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

12.4.3. Creating an LVM-based Storage Pool with virsh

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

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

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

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

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

```
# virsh pool-build guest_images_lvm

Pool guest_images_lvm built
```

3. Initialize the new pool.

```
# virsh pool-start guest_images_lvm

Pool guest_images_lvm started
```

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

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

5. Set the pool to start automatically.

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

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

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

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

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

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

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

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

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

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

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

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

12.4.4. Deleting a Storage Pool Using virsh

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

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

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

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

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

3. Remove the storage pool's definition

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

12.5. ISCSI-BASED STORAGE POOLS

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

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

12.5.1. Configuring a Software iSCSI Target

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

Procedure 12.4. Creating an iSCSI target

1. **Install the required packages**
   Install the scsi-target-utils package and all dependencies

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

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

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

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

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

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

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

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

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

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

4. **Optional: Create file-based images**
   File-based storage is sufficient for testing but is not recommended for production environments or any significant I/O activity. This optional procedure creates a file based imaged named *virtimage2.img* for an iSCSI target.

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

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

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

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

   c. **Configure SELinux file contexts**
      Configure the correct SELinux context for the new image and directory.

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

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

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

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

   Where:

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

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

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

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

IMPORTANT

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

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

```
# service tgtd restart
```

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

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

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

```
# tgt-admin --show
Target 1: iqn.2010-05.com.example.server1:iscsirhel6guest
System information:
Driver: iscsi
State: ready
I_T nexus information:
LUN information:
LUN: 0
    Type: controller
    SCSI ID: IET     00010000
    SCSI SN: beaf10
    Size: 0 MB
```

```
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: None
LUN: 1
    Type: disk
    SCSI ID: IET     00010001
    SCSI SN: beaf11
    Size: 20000 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: /dev/virtstore/virtimage1
LUN: 2
    Type: disk
    SCSI ID: IET     00010002
    SCSI SN: beaf12
    Size: 10000 MB
    Online: Yes
    Removable media: No
    Backing store type: rdwr
    Backing store path: /var/lib/tgtd/virtualization/virtimage2.img
Account information:
ACL information:
ALL
```

WARNING

The ACL list is set to all. This allows all systems on the local network to access this device. It is recommended to set host physical machine access ACLs for production environments.

9. **Optional: Test discovery**
   Test whether the new iSCSI device is discoverable.

```
# iscsiadm --mode discovery --type sendtargets --portal server1.example.com
127.0.0.1:3260,1 iqn.2010-05.com.example.server1:iscsirhel6guest
```

10. **Optional: Test attaching the device**
    Attach the new device (*iqn.2010-05.com.example.server1:iscsirhel6guest*) to determine whether the device can be attached.

```
# iscsiadm -d2 -m node --login
scsiadm: Max file limits 1024 1024

Logging in to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260]
Login to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260] successful.
```

Detach the device.

```
# iscsiadm -d2 -m node --logout
scsiadm: Max file limits 1024 1024

Logging out of session [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel6guest,
portal: 10.0.0.1,3260
Logout of [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel6guest, portal:
10.0.0.1,3260] successful.
```

An iSCSI device is now ready to use for virtualization.

12.5.2. Adding an iSCSI Target to virt-manager

This procedure covers creating a storage pool with an iSCSI target in **virt-manager**.

Procedure 12.5. Adding an iSCSI device to virt-manager

1. **Open the host physical machine's storage tab**
   Open the **Storage** tab in the **Connection Details** window.

   a. Open **virt-manager**.

   b. Select a host physical machine from the main **virt-manager** window. Click **Edit menu** and select **Connection Details**.

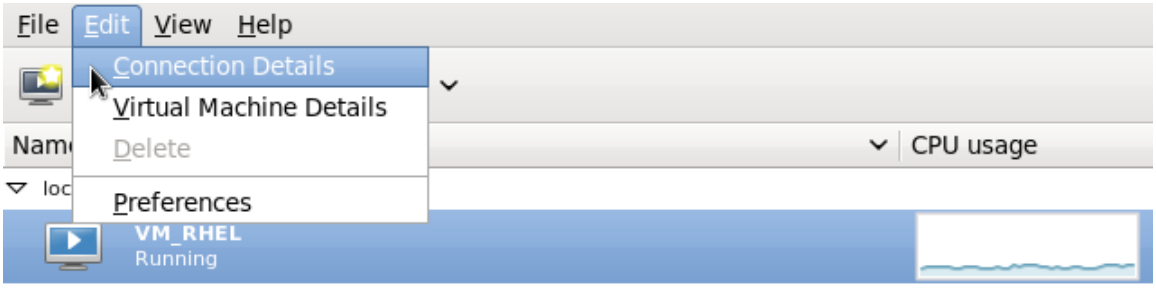


   Figure 12.19. Connection details

   c. Click on the **Storage** tab.

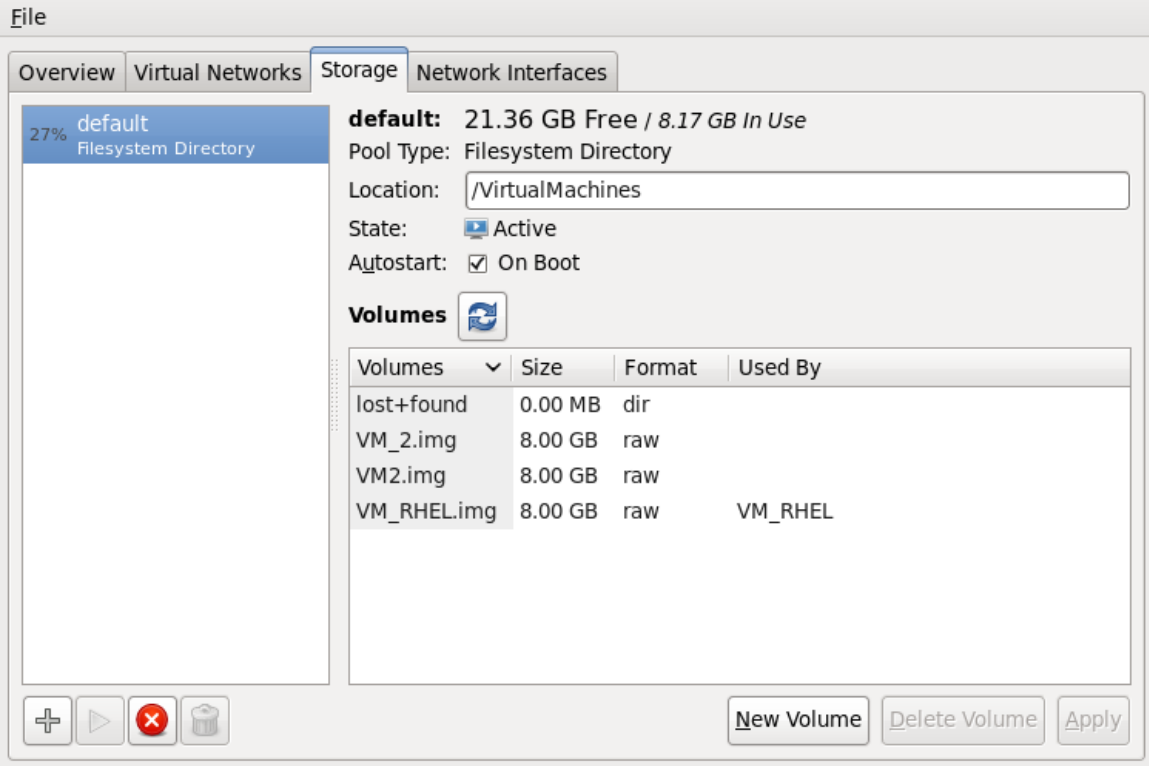


Figure 12.20. Storage menu

2. **Add a new pool (part 1)**
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

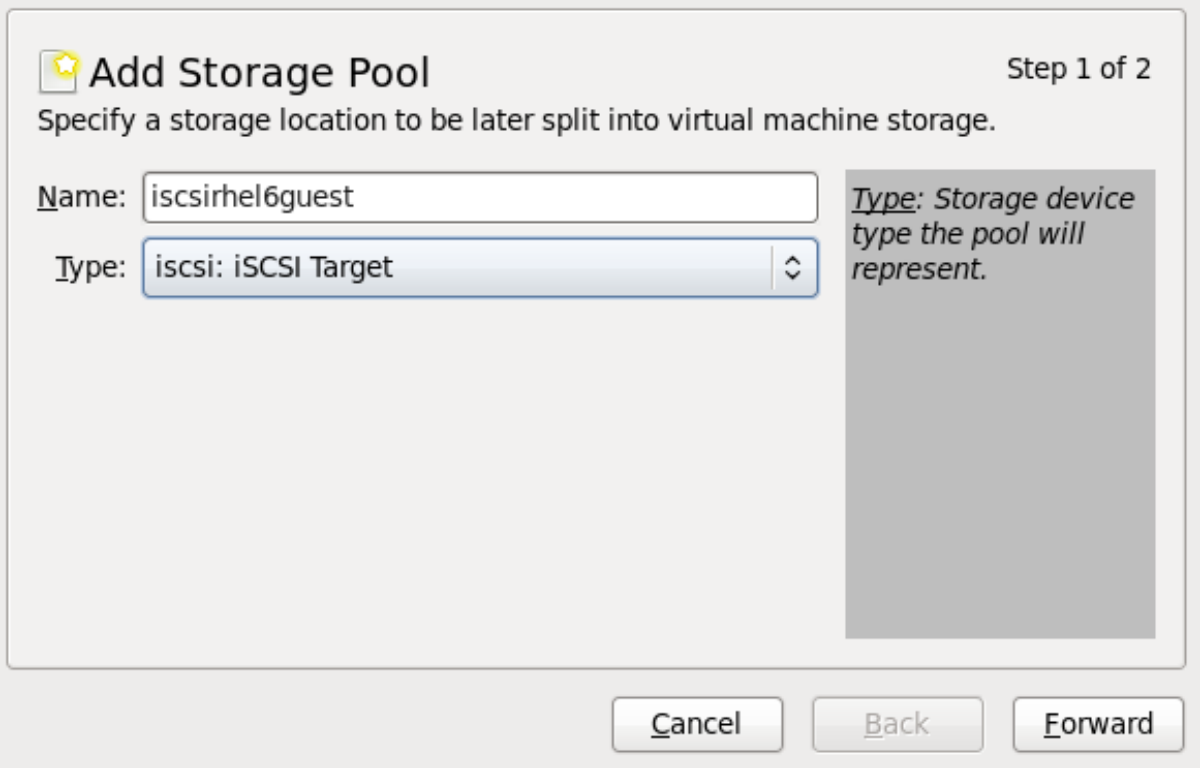


Figure 12.21. Add an iscsi storage pool name and type

Choose a name for the storage pool, change the Type to iscsi, and press **Forward** to continue.

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

You will need the information you used in Section 12.5, "iSCSI-based Storage Pools" and Procedure 12.4, "Creating an iSCSI target" to complete the fields in this menu.

a. Enter the iSCSI source and target. The **Format** option is not available as formatting is handled by the guest virtual machines. It is not advised to edit the **Target Path**. The default target path value, **/dev/disk/by-path/**, adds the drive path to that directory. The target path should be the same on all host physical machines for migration.

b. Enter the host name or IP address of the iSCSI target. This example uses **host1.example.com**.

c. In the **Source Path**field, enter the iSCSI target IQN. If you look at Procedure 12.4, "Creating an iSCSI target" in Section 12.5, "iSCSI-based Storage Pools" , this is the information you added in the **/etc/tgt/targets.conf file**. This example uses **iqn.2010-05.com.example.server1:iscsirhel6guest**.

d. Check the **IQN** check box to enter the IQN for the initiator. This example uses **iqn.2010-05.com.example.host1:iscsirhel6**.

e. Click **Finish** to create the new storage pool.

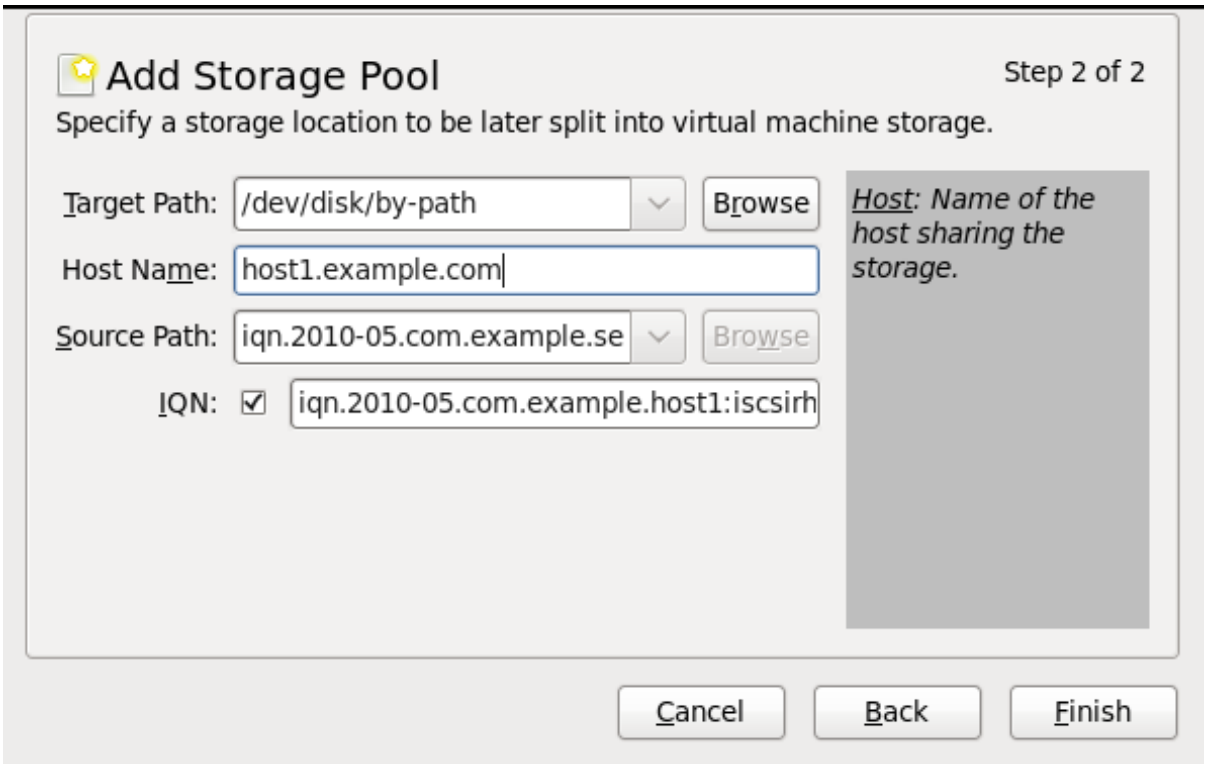


Figure 12.22. Create an iscsi storage pool

12.5.3. Deleting a Storage Pool Using virt-manager

This procedure demonstrates how to delete a storage pool.

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

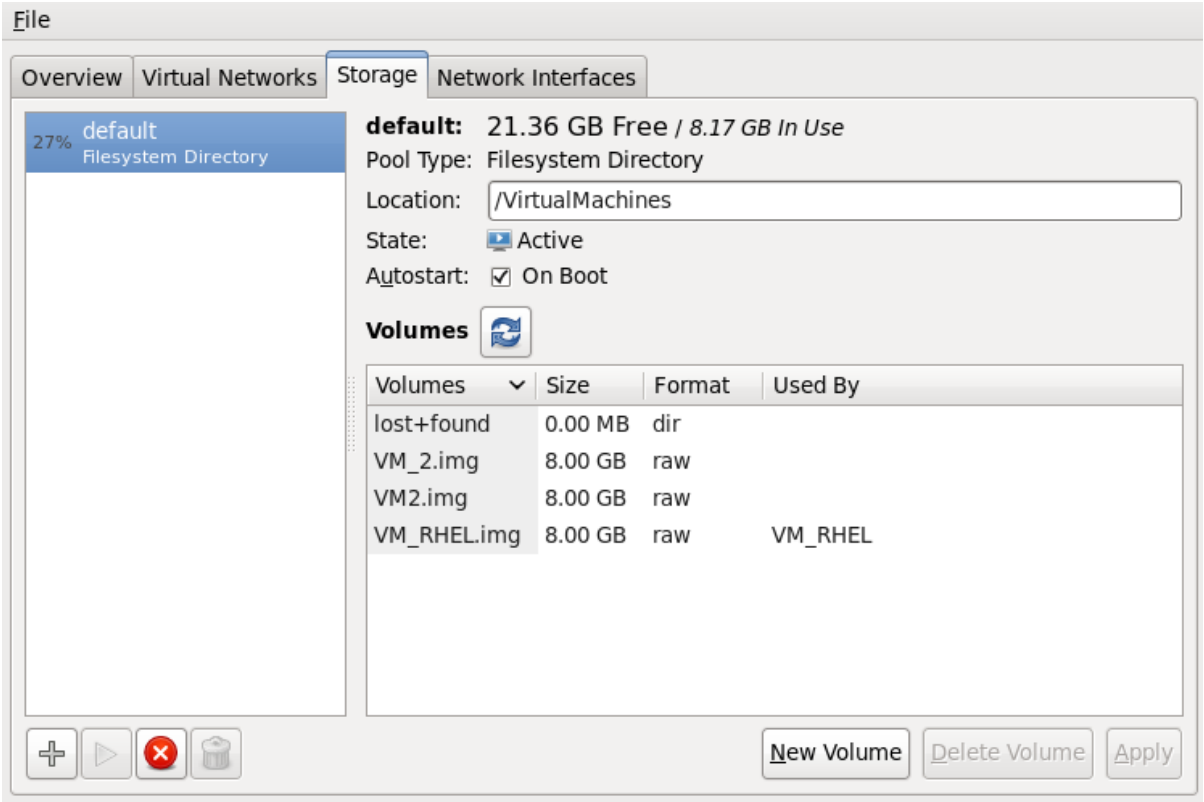


Figure 12.23. Stop Icon

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

12.5.4. Creating an iSCSI-based Storage Pool with virsh

1. **Use pool-define-as to define the pool from the command line**
   Storage pool definitions can be created with the **virsh** command line tool. Creating storage pools with **virsh** is useful for systems administrators using scripts to create multiple storage pools.

   The **virsh pool-define-as** command has several parameters which are accepted in the following format:

   > virsh pool-define-as *name type source-host source-path source-dev source-name target*

   The parameters are explained as follows:

   type

   defines this pool as a particular type, iscsi for example

   name

   must be unique and sets the name for the storage pool

   source-host and source-path

   the host name and iSCSI IQN respectively

   source-dev and source-name

these parameters are not required for iSCSI-based pools, use a **-** character to leave the field blank.

target

defines the location for mounting the iSCSI device on the host physical machine

The example below creates the same iSCSI-based storage pool as the previous step.

```
# virsh pool-define-as --name scsirhel6guest --type iscsi \
  --source-host server1.example.com \
  --source-dev iqn.2010-05.com.example.server1:iscsirhel6guest
  --target /dev/disk/by-path
Pool iscsirhel6guest defined
```

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

```
# virsh pool-list --all
Name                State      Autostart
-----------------------------------------
default             active     yes
iscsirhel6guest     inactive   no
```

3. **Start the storage pool**
   Use the virsh command **pool-start** for this. **pool-start** enables a directory storage pool, allowing it to be used for volumes and guest virtual machines.

```
# virsh pool-start guest_images_disk
Pool guest_images_disk started
# virsh pool-list --all
Name                State      Autostart
-----------------------------------------
default             active     yes
iscsirhel6guest     active     no
```

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

```
# virsh pool-autostart iscsirhel6guest
Pool iscsirhel6guest marked as autostarted
```

   Verify that the *iscsirhel6guest* pool has autostart set:

```
# virsh pool-list --all
Name                State      Autostart
-----------------------------------------
default             active     yes
iscsirhel6guest     active     yes
```

5. **Verify the storage pool configuration**

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

```
# virsh pool-info iscsirhel6guest
Name:          iscsirhel6guest
UUID:          afcc5367-6770-e151-bcb3-847bc36c5e28
State:         running
Persistent:    unknown
Autostart:     yes
Capacity:      100.31 GB
Allocation:    0.00
Available:     100.31 GB
```

An iSCSI-based storage pool is now available.

12.5.5. Deleting a Storage Pool Using virsh

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

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

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

2. Remove the storage pool's definition

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

12.6. NFS-BASED STORAGE POOLS

This procedure covers creating a storage pool with a NFS mount point in **virt-manager**.

12.6.1. Creating an NFS-based Storage Pool with virt-manager

1. **Open the host physical machine's storage tab**
   Open the **Storage** tab in the **Host Details** window.

   a. Open **virt-manager**.

   b. Select a host physical machine from the main **virt-manager** window. Click **Edit menu** and select **Connection Details**.

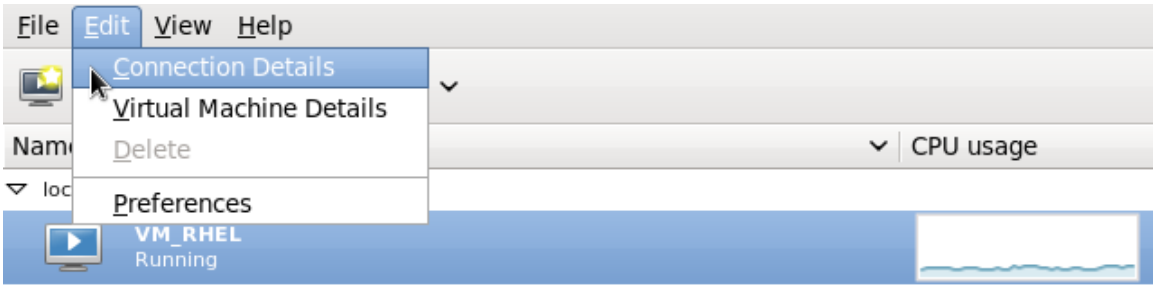


Figure 12.24. Connection details

c. Click on the Storage tab.

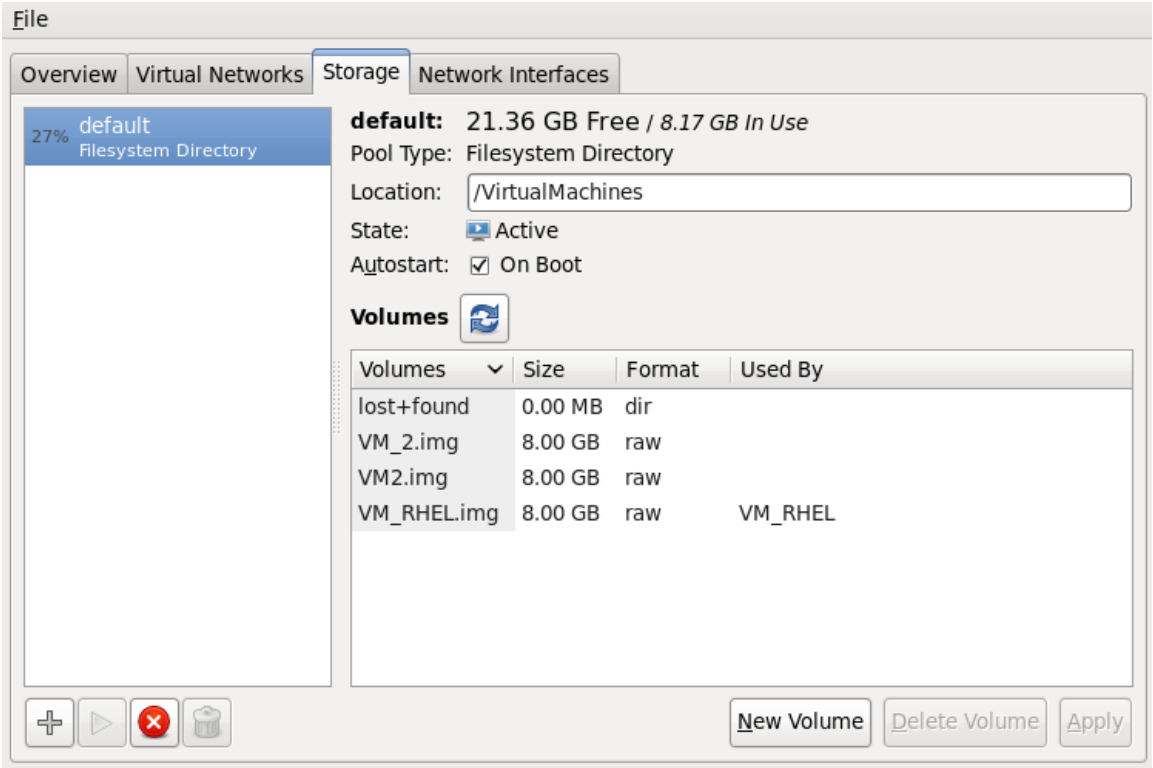


Figure 12.25. Storage tab

2. **Create a new pool (part 1)**
   Press the **+** button (the add pool button). The **Add a New Storage Pool** wizard appears.

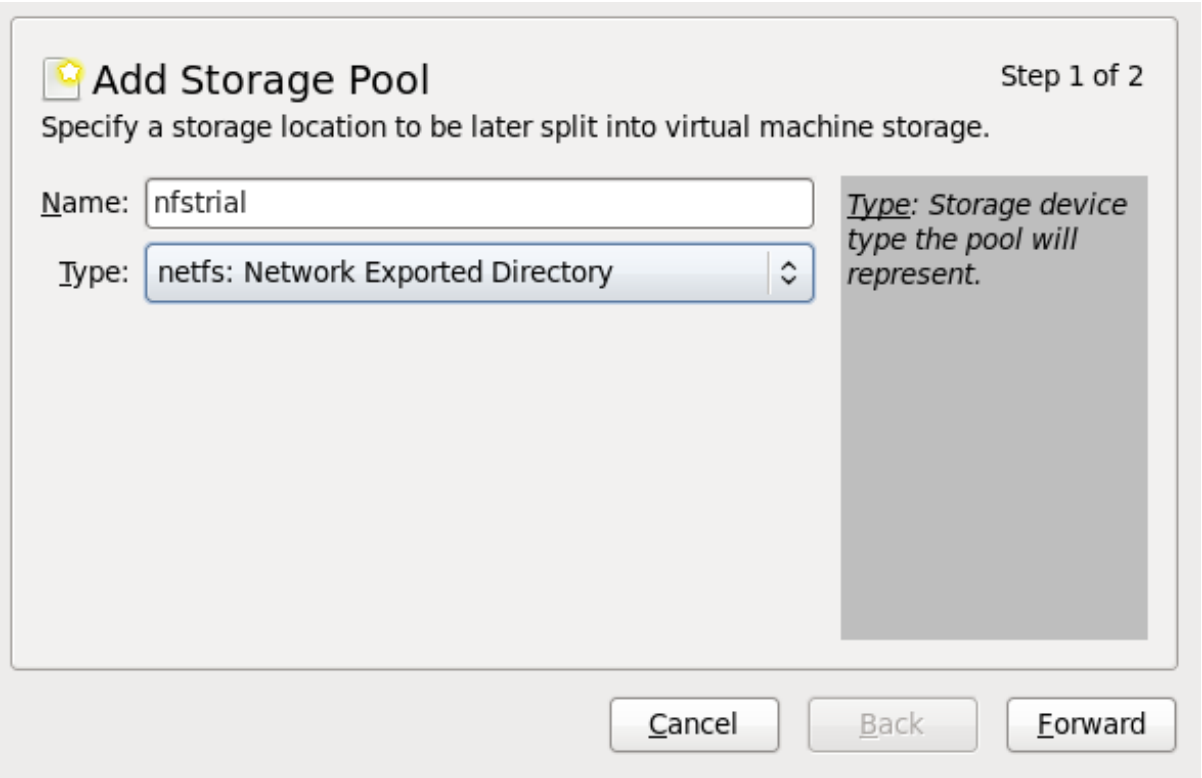


Figure 12.26. Add an NFS name and type

Choose a name for the storage pool and press **Forward** to continue.

3. **Create a new pool (part 2)**
   Enter the target path for the device, the host name and the NFS share path. Set the **Format** option to **NFS** or **auto** (to detect the type). The target path must be identical on all host physical machines for migration.

   Enter the host name or IP address of the NFS server. This example uses **server1.example.com**.

   Enter the NFS path. This example uses **/nfstrial**.

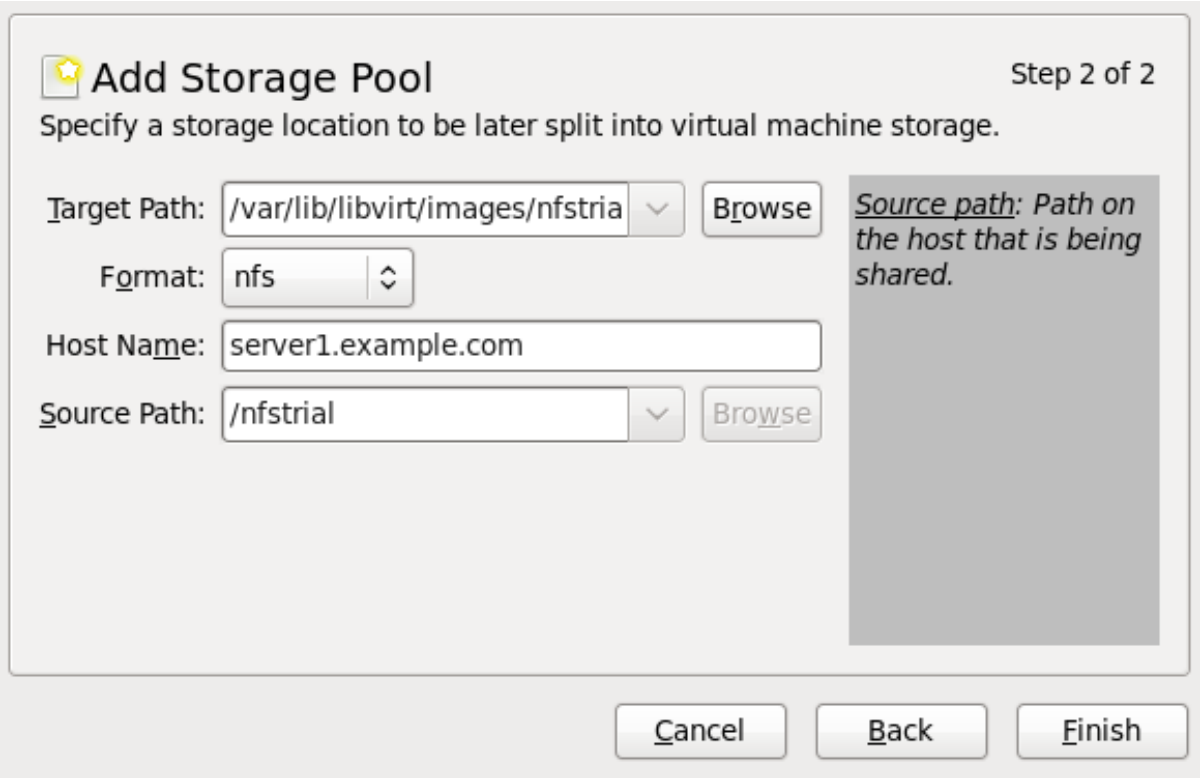


Figure 12.27. Create an NFS storage pool

   Press **Finish** to create the new storage pool.

12.6.2. Deleting a Storage Pool Using virt-manager

This procedure demonstrates how to delete a storage pool.

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

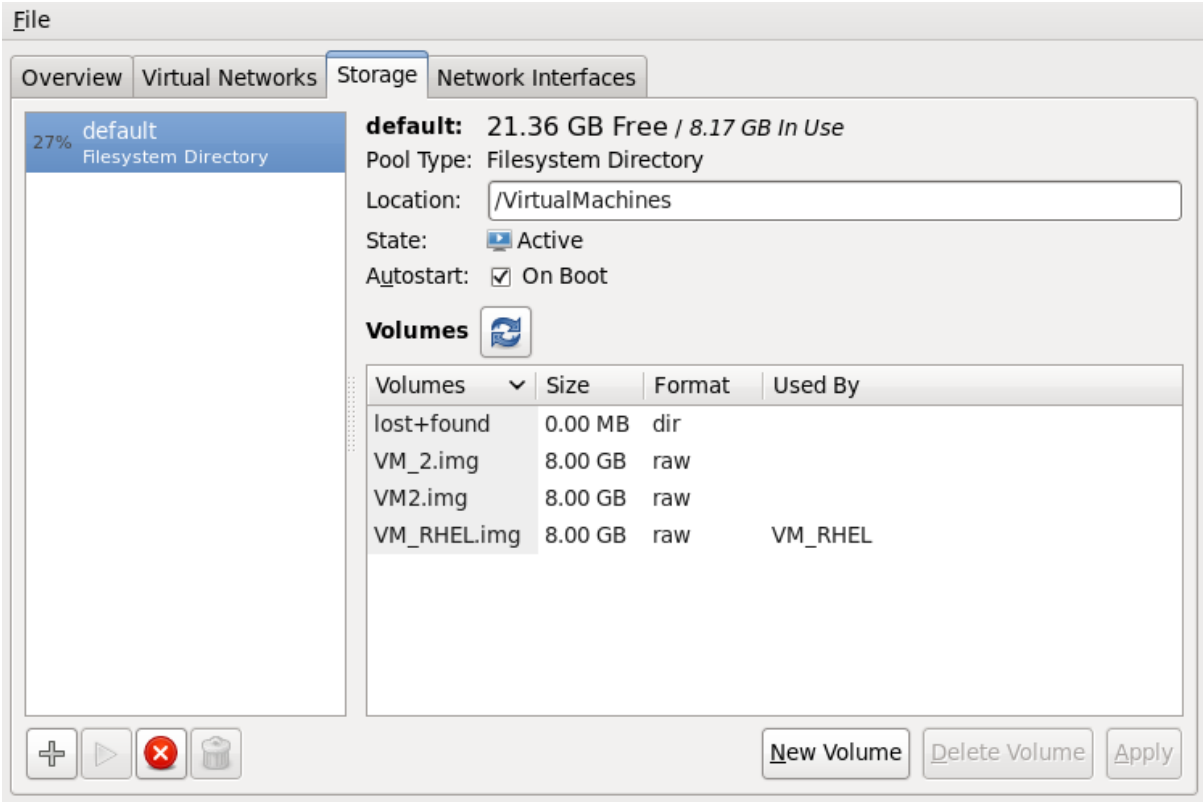


Figure 12.28. Stop Icon

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

12.7. GLUSTERFS STORAGE POOLS

GlusterFS is a userspace file system that uses FUSE. When enabled in a guest virtual machine it enables a KVM host physical machine to boot guest virtual machine images from one or more GlusterFS storage volumes, and to use images from a GlusterFS storage volume as data disks for guest virtual machines.

IMPORTANT

Red Hat Red Hat Enterprise Linux 6 does not support the use of GlusterFS with storage pools. However, Red Hat Enterprise Linux 6.5 and later includes native support for creating virtual machines with GlusterFS using the libgfapi library.

12.8. USING AN NPIV VIRTUAL ADAPTER (VHBA) WITH SCSI DEVICES

NPIV (N_Port ID Virtualization) is a software technology that allows sharing of a single physical Fibre Channel host bus adapter (HBA).

This allows multiple guests to see the same storage from multiple physical hosts, and thus allows for easier migration paths for the storage. As a result, there is no need for the migration to create or copy storage, as long as the correct storage path is specified.

In virtualization, the *virtual host bus adapter*, or *vHBA*, controls the LUNs for virtual machines. Each vHBA is identified by its own WWNN (World Wide Node Name) and WWPN (World Wide Port Name). The path to the storage is determined by the WWNN and WWPN values.

This section provides instructions for configuring a vHBA on a virtual machine. Note that Red Hat Enterprise Linux 6 does not support persistent vHBA configuration across host reboots; verify any vHBA-related settings following a host reboot.

12.8.1. Creating a vHBA

Procedure 12.6. Creating a vHBA

1. **Locate HBAs on the host system**
   To locate the HBAs on your host system, examine the SCSI devices on the host system to locate a **scsi_host** with **vport** capability.

   Run the following command to retrieve a **scsi_host** list:

   ```
   # virsh nodedev-list --cap scsi_host
   scsi_host0
   scsi_host1
   scsi_host2
   scsi_host3
   scsi_host4
   ```

   For each **scsi_host**, run the following command to examine the device XML for the line **<capability type='vport_ops'>**, which indicates a **scsi_host** with **vport** capability.

   ```
   # virsh nodedev-dumpxml scsi_hostN
   ```

2. **Check the HBA's details**
   Use the **virsh nodedev-dumpxml *HBA_device*** command to see the HBA's details.

   The XML output from the **virsh nodedev-dumpxml** command will list the fields **<name>**, **<wwnn>**, and **<wwpn>**, which are used to create a vHBA. The **<max_vports>** value shows the maximum number of supported vHBAs.

   ```
   # virsh nodedev-dumpxml scsi_host3
   <device>
     <name>scsi_host3</name>
     <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
     <parent>pci_0000_10_00_0</parent>
     <capability type='scsi_host'>
       <host>3</host>
       <capability type='fc_host'>
         <wwnn>20000000c9848140</wwnn>
         <wwpn>10000000c9848140</wwpn>
         <fabric_wwn>2002000573de9a81</fabric_wwn>
       </capability>
       <capability type='vport_ops'>
         <max_vports>127</max_vports>
         <vports>0</vports>
       </capability>
     </capability>
   </device>
   ```

In this example, the **<max_vports>** value shows there are a total 127 virtual ports available for use in the HBA configuration. The **<vports>** value shows the number of virtual ports currently being used. These values update after creating a vHBA.

3. **Create a vHBA host device**
   Create an XML file similar to the following (in this example, named *vhba_host3.xml*) for the vHBA host.

```
# cat vhba_host3.xml
  <device>
    <parent>scsi_host3</parent>
    <capability type='scsi_host'>
      <capability type='fc_host'>
      </capability>
    </capability>
  </device>
```

   The **<parent>** field specifies the HBA device to associate with this vHBA device. The details in the **<device>** tag are used in the next step to create a new vHBA device for the host. See http://libvirt.org/formatnode.html for more information on the **nodedev** XML format.

4. **Create a new vHBA on the vHBA host device**
   To create a vHBA on *vhba_host3*, use the **virsh nodedev-create** command:

```
# virsh nodedev-create vhba_host3.xml
Node device scsi_host5 created from vhba_host3.xml
```

5. **Verify the vHBA**
   Verify the new vHBA's details (**scsi_host5**) with the **virsh nodedev-dumpxml** command:

```
# virsh nodedev-dumpxml scsi_host5
<device>
  <name>scsi_host5</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3/vport-3:0-0/host5</path>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <host>5</host>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
  </capability>
</device>
```

12.8.2. Creating a Storage Pool Using the vHBA

It is recommended to define a libvirt storage pool based on the vHBA in order to preserve the vHBA configuration.

Using a storage pool has two primary advantages:

- the libvirt code can easily find the LUN's path using the **virsh** command output, and

- virtual machine migration requires only defining and starting a storage pool with the same vHBA name on the target machine. To do this, the vHBA LUN, libvirt storage pool and volume name must be specified in the virtual machine's XML configuration. Refer to Section 12.8.3, "Configuring the Virtual Machine to Use a vHBA LUN" for an example.

1. **Create a SCSI storage pool**
   To create a vHBA configuration, first create a libvirt **'scsi'** storage pool XML file based on the vHBA using the format below.

   NOTE

   Ensure you use the vHBA created in Procedure 12.6, "Creating a vHBA" as the host name, modifying the vHBA name *scsi_hostN* to *hostN* for the storage pool configuration. In this example, the vHBA is named *scsi_host5*, which is specified as **<adapter name='host5'/>** in a Red Hat Enterprise Linux 6 libvirt storage pool.

   It is recommended to use a stable location for the **<path>** value, such as one of the **/dev/disk/by-{path|id|uuid|label}** locations on your system. More information on **<path>** and the elements within **<target>** can be found at http://libvirt.org/formatstorage.html.

   In this example, the **'scsi'** storage pool is named *vhbapool_host3.xml*:

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <uuid>e9392370-2917-565e-692b-d057f46512d6</uuid>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <adapter name='host5'/>
  </source>
   <target>
     <path>/dev/disk/by-path</path>
     <permissions>
       <mode>0700</mode>
       <owner>0</owner>
       <group>0</group>
     </permissions>
   </target>
 </pool>
```

2. **Define the pool**
   To define the storage pool (named *vhbapool_host3* in this example), use the **virsh pool-define** command:

```
# virsh pool-define vhbapool_host3.xml
Pool vhbapool_host3 defined from vhbapool_host3.xml
```

3. **Start the pool**
   Start the storage pool with the following command:

```
# virsh pool-start vhbapool_host3
Pool vhbapool_host3 started
```

4. **Enable autostart**
   Finally, to ensure that subsequent host reboots will automatically define vHBAs for use in virtual machines, set the storage pool autostart feature (in this example, for a pool named *vhbapool_host3*):

```
# virsh pool-autostart vhbapool_host3
```

12.8.3. Configuring the Virtual Machine to Use a vHBA LUN

After a storage pool is created for a vHBA, add the vHBA LUN to the virtual machine configuration.

1. **Find available LUNs**
   First, use the **virsh vol-list** command in order to generate a list of available LUNs on the vHBA. For example:

```
# virsh vol-list vhbapool_host3
 Name                 Path
------------------------------------------------------------------------------
 unit:0:4:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016844602198-lun-0
 unit:0:5:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0
```

   The list of LUN names displayed will be available for use as disk volumes in virtual machine configurations.

2. **Add the vHBA LUN to the virtual machine**
   Add the vHBA LUN to the virtual machine by specifying in the virtual machine's XML:

   - the device type as **lun** or **disk** in the **<disk>** parameter, and
   - the source device in the **<source>** parameter. Note this can be entered as **/dev/sdaN**, or as a symbolic link generated by udev in **/dev/disk/by-path|by-id|by-uuid|by-label**, which can be found by running the **virsh vol-list *pool*** command.

   For example:

```
<disk type='block' device='lun'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/disk/by-path/pci-0000\:04\:00.1-fc-0x203400a0b85ad1d7-lun-0'/>
  <target dev='sda' bus='scsi'/>
</disk>
```

12.8.4. Destroying the vHBA Storage Pool

A vHBA storage pool can be destroyed by the **virsh pool-destroy** command:

```
# virsh pool-destroy vhbapool_host3
```

Delete the vHBA with the following command

```
# virsh nodedev-destroy scsi_host5
```

To verify the pool and vHBA have been destroyed, run:

```
# virsh nodedev-list --cap scsi_host
```

scsi_host5 will no longer appear in the list of results.

CHAPTER 13. VOLUMES

13.1. CREATING VOLUMES

This section shows how to create disk volumes inside a block based storage pool. In the example below, the **virsh vol-create-as** command will create a storage volume with a specific size in GB within the *guest_images_disk* storage pool. As this command is repeated per volume needed, three volumes are created as shown in the example.

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

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

# virsh vol-create-as guest_images_disk volume3 8G
Vol volume3 created

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
volume1              /dev/sdb1
volume2              /dev/sdb2
volume3              /dev/sdb3

# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start   End     Size    File system  Name     Flags
2       17.4kB  8590MB  8590MB               primary
3       8590MB  17.2GB  8590MB               primary
1       21.5GB  30.1GB  8590MB               primary
```

13.2. CLONING VOLUMES

The new volume will be allocated from storage in the same storage pool as the volume being cloned. The **virsh vol-clone** must have the **--pool** argument which dictates the name of the storage pool that contains the volume to be cloned. The rest of the command names the volume to be cloned (volume3) and the name of the new volume that was cloned (clone1). The **virsh vol-list** command lists the volumes that are present in the storage pool (guest_images_disk).

```
# virsh vol-clone --pool guest_images_disk volume3 clone1
Vol clone1 cloned from volume3

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
volume1              /dev/sdb1
volume2              /dev/sdb2
```

```
volume3            /dev/sdb3
clone1            /dev/sdb4


# parted -s /dev/sdb print
Model: ATA ST3500418AS (scsi)
Disk /dev/sdb: 500GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    File system  Name     Flags
1       4211MB  12.8GB  8595MB  primary
2       12.8GB  21.4GB  8595MB  primary
3       21.4GB  30.0GB  8595MB  primary
4       30.0GB  38.6GB  8595MB  primary
```

13.3. ADDING STORAGE DEVICES TO GUESTS

This section covers adding storage devices to a guest. Additional storage can only be added as needed.

13.3.1. Adding File-based Storage to a Guest

File-based storage is a collection of files that are stored on the host physical machines file system that act as virtualized hard drives for guests. To add file-based storage, perform the following steps:

Procedure 13.1. Adding file-based storage

1. Create a storage file or use an existing file (such as an IMG file). Note that both of the following commands create a 4GB file which can be used as additional storage for a guest:

   - Pre-allocated files are recommended for file-based storage images. Create a pre-allocated file using the following **dd** command as shown:

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

   - Alternatively, create a sparse file instead of a pre-allocated file. Sparse files are created much faster and can be used for testing, but are not recommended for production environments due to data integrity and performance issues.

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

2. Create the additional storage by writing a *<disk>* element in a new file. In this example, this file will be known as **NewStorage.xml**.

   A **<disk>** element describes the source of the disk, and a device name for the virtual block device. The device name should be unique across all devices in the guest, and identifies the bus on which the guest will find the virtual block device. The following example defines a virtio block device whose source is a file-based storage container named **FileName.img**:

   ```
   <disk type='file' device='disk'>
     <driver name='qemu' type='raw' cache='none'/>
     <source file='/var/lib/libvirt/images/FileName.img'/>
   ```

```
  <target dev='vdb'/>
</disk>
```

Device names can also start with "hd" or "sd", identifying respectively an IDE and a SCSI disk. The configuration file can also contain an **<address>** sub-element that specifies the position on the bus for the new device. In the case of virtio block devices, this should be a PCI address. Omitting the **<address>** sub-element lets libvirt locate and assign the next available PCI slot.

3. Attach the CD-ROM as follows:

```
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw' cache='none'/>
  <source file='/var/lib/libvirt/images/FileName.img'/>
  <readonly/>
  <target dev='hdc'/>
</disk >
```

4. Add the device defined in **NewStorage.xml** with your guest (**Guest1**):

```
# virsh attach-device --config Guest1 ~/NewStorage.xml
```

NOTE

This change will only apply after the guest has been destroyed and restarted. In addition, persistent devices can only be added to a persistent domain, that is a domain whose configuration has been saved with **virsh define** command.

If the guest is running, and you want the new device to be added temporarily until the guest is destroyed, omit the **--config** option:

```
# virsh attach-device Guest1 ~/NewStorage.xml
```

NOTE

The **virsh** command allows for an **attach-disk** command that can set a limited number of parameters with a simpler syntax and without the need to create an XML file. The **attach-disk** command is used in a similar manner to the **attach-device** command mentioned previously, as shown:

```
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img vdb --cache
none --driver qemu --subdriver raw
```

Note that the **virsh attach-disk** command also accepts the **--config** option.

5. Start the guest machine (if it is currently not running):

```
# virsh start Guest1
```

NOTE

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

6. **Partitioning the disk drive**
   The guest now has a hard disk device called **/dev/vdb**. If required, partition this disk drive and format the partitions. If you do not see the device that you added, then it indicates that there is an issue with the disk hotplug in your guest's operating system.

   a. Start **fdisk** for the new device:

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

   b. Type ***n*** for a new partition.

   c. The following appears:

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

   Type ***p*** for a primary partition.

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

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

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

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

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

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

   g. Enter **t** to configure the partition type.

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

   h. Select the partition you created in the previous steps. In this example, the partition number is **1** as there was only one partition created and fdisk automatically selected partition 1.

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

   i. Enter **83** for a Linux partition.

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

j. Enter **w** to write changes and quit.

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

k. Format the new partition with the **ext3** file system.

```
# mke2fs -j /dev/vdb1
```

7. Create a mount directory, and mount the disk on the guest. In this example, the directory is located in *myfiles*.

```
# mkdir /myfiles
# mount /dev/vdb1 /myfiles
```

The guest now has an additional virtualized file-based storage device. Note however, that this storage will not mount persistently across reboot unless defined in the guest's **/etc/fstab** file:

```
/dev/vdb1    /myfiles    ext3    defaults    0 0
```

13.3.2. Adding Hard Drives and Other Block Devices to a Guest

System administrators have the option to use additional hard drives to provide increased storage space for a guest, or to separate system data from user data.

Procedure 13.2. Adding physical block devices to guests

1. This procedure describes how to add a hard drive on the host physical machine to a guest. It applies to all physical block devices, including CD-ROM, DVD and floppy devices.

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

2. Do one of the following:

   a. Create the additional storage by writing a **disk** element in a new file. In this example, this file will be known as **NewStorage.xml**. The following example is a configuration file section which contains an additional device-based storage container for the host physical machine partition **/dev/sr0:**

```
<disk type='block' device='disk'>
    <driver name='qemu' type='raw' cache='none'/>
    <source dev='/dev/sr0'/>
    <target dev='vdc' bus='virtio'/>
</disk>
```

   b. Follow the instruction in the previous section to attach the device to the guest virtual machine. Alternatively, you can use the ***virsh attach-disk*** command, as shown:

```
# virsh attach-disk Guest1 /dev/sr0 vdc
```

   Note that the following options are available:

- The **virsh attach-disk** command also accepts the ***--config***, ***--type***, and ***--mode*** options, as shown:

```
# virsh attach-disk Guest1 /dev/sr0 vdc --config --type cdrom --mode readonly
```

- Additionally, ***--type*** also accepts ***--type disk*** in cases where the device is a hard drive.

3. The guest virtual machine now has a new hard disk device called **/dev/vdc** on Linux (or something similar, depending on what the guest virtual machine OS chooses) or **D: drive** (for example) on Windows. You can now initialize the disk from the guest virtual machine, following the standard procedures for the guest virtual machine's operating system. Refer to Procedure 13.1, "Adding file-based storage" for an example.

WARNING

When adding block devices to a guest, make sure to follow security considerations. This information is discussed in more detail in the *Red Hat Enterprise Linux Virtualization Security Guide* which can be found at https://access.redhat.com/site/documentation/.

IMPORTANT

Guest virtual machines should not be given write access to whole disks or block devices (for example, **/dev/sdb**). Guest virtual machines with access to whole block devices may be able to modify volume labels, which can be used to compromise the host physical machine system. Use partitions (for example, **/dev/sdb1**) or LVM volumes to prevent this issue.

13.4. DELETING AND REMOVING VOLUMES

This section shows how to delete a disk volume from a block based storage pool using the **virsh vol-delete** command. In this example, the volume is *volume 1* and the storage pool is *guest_images*.

```
# virsh vol-delete --pool guest_images volume1
Vol volume1 deleted
```

CHAPTER 14. MANAGING GUEST VIRTUAL MACHINES WITH VIRSH

virsh is a command line interface tool for managing guest virtual machines and the hypervisor. The **virsh** command-line tool is built on the **libvirt** management API and operates as an alternative to the **qemu-kvm** command and the graphical **virt-manager** application. The **virsh** command can be used in read-only mode by unprivileged users or, with root access, full administration functionality. The **virsh** command is ideal for scripting virtualization administration.

14.1. GENERIC COMMANDS

The commands in this section are generic because they are not specific to any domain.

14.1.1. help

$ virsh help [command|group] The help command can be used with or without options. When used without options, all commands are listed, one per line. When used with an option, it is grouped into categories, displaying the keyword for each group.

To display the commands that are only for a specific option, you need to give the keyword for that group as an option. For example:

```
$ virsh help pool
 Storage Pool (help keyword 'pool'):
   find-storage-pool-sources-as   find potential storage pool sources
   find-storage-pool-sources      discover potential storage pool sources
   pool-autostart                 autostart a pool
   pool-build                     build a pool
   pool-create-as                 create a pool from a set of args
   pool-create                    create a pool from an XML file
   pool-define-as                 define a pool from a set of args
   pool-define                    define (but don't start) a pool from an XML file
   pool-delete                    delete a pool
   pool-destroy                   destroy (stop) a pool
   pool-dumpxml                   pool information in XML
   pool-edit                      edit XML configuration for a storage pool
   pool-info                      storage pool information
   pool-list                      list pools
   pool-name                      convert a pool UUID to pool name
   pool-refresh                   refresh a pool
   pool-start                     start a (previously defined) inactive pool
   pool-undefine                  undefine an inactive pool
   pool-uuid                      convert a pool name to pool UUID
```

Using the same command with a command option, gives the help information on that one specific command. For example:

```
$ virsh help vol-path
  NAME
    vol-path - returns the volume path for a given volume name or key

  SYNOPSIS
    vol-path <vol> [--pool <string>]
```

```
OPTIONS
  [--vol] <string>  volume name or key
  --pool <string>  pool name or uuid
```

14.1.2. quit and exit

The quit command and the exit command will close the terminal. For example:

```
$ virsh exit
```

```
$ virsh quit
```

14.1.3. version

The version command displays the current libvirt version and displays information about where the build is from. For example:

```
$ virsh version
Compiled against library: libvirt 1.1.1
Using library: libvirt 1.1.1
Using API: QEMU 1.1.1
Running hypervisor: QEMU 1.5.3
```

14.1.4. Argument Display

The **virsh echo [--shell][--xml][arg]** command echos or displays the specified argument. Each argument echoed will be separated by a space. by using the **--shell** option, the output will be single quoted where needed so that it is suitable for reusing in a shell command. If the **--xml** option is used the output will be made suitable for use in an XML file. For example, the command **virsh echo --shell "hello world"** will send the output **'hello world'**.

14.1.5. connect

Connects to a hypervisor session. When the shell is first started this command runs automatically when the URI parameter is requested by the **-c** command. The URI specifies how to connect to the hypervisor. The most commonly used URIs are:

- **xen:**/// - connects to the local Xen hypervisor.
- **qemu:///system** - connects locally as root to the daemon supervising QEMU and KVM domains.
- **xen:///session** - connects locally as a user to the user's set of QEMU and KVM domains.
- **lxc:**/// - connects to a local Linux container.

Additional values are available on libvirt's website http://libvirt.org/uri.html.

The command can be run as follows:

```
$ virsh connect {name|URI}
```

Where ***{name}*** is the machine name (host name) or URL (the output of the **virsh uri** command) of the hypervisor. To initiate a read-only connection, append the above command with **--readonly**. For more

information on URIs refer to Remote URIs. If you are unsure of the URI, the **virsh uri** command will display it:

```
$ virsh uri
qemu:///session
```

14.1.6. Displaying Basic Information

The following commands may be used to display basic information:

- $ **hostname** - displays the hypervisor's host name
- $ **sysinfo** - displays the XML representation of the hypervisor's system information, if available

14.1.7. Injecting NMI

The $ **virsh inject-nmi [domain]** injects NMI (non-maskable interrupt) message to the guest virtual machine. This is used when response time is critical, such as non-recoverable hardware errors. To run this command:

```
$ virsh inject-nmi guest-1
```

14.2. ATTACHING AND UPDATING A DEVICE WITH VIRSH

For information on attaching storage devices refer to Section 13.3.1, "Adding File-based Storage to a Guest"

Procedure 14.1. Hot plugging USB devices for use by the guest virtual machine

The following procedure demonstrates how to attach USB devices to the guest virtual machine. This can be done while the guest virtual machine is running as a hotplug procedure or it can be done while the guest is shutoff. The device you want to emulate needs to be attached to the host physical machine.

1. Locate the USB device you want to attach with the following command:

```
# lsusb -v

idVendor          0x17ef Lenovo
idProduct         0x480f Integrated Webcam [R5U877]
```

2. Create an XML file and give it a logical name (**usb_device.xml**, for example). Make sure you copy the vendor and product IDs exactly as was displayed in your search.

```
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x17ef'/>
    <product id='0x480f'/>
  </source>
</hostdev>
...
```

Figure 14.1. USB Devices XML Snippet

3. Attach the device with the following command:

```
# virsh attach-device rhel6 --file usb_device.xml --config
```

In this example [rhel6] is the name of your guest virtual machine and [usb_device.xml] is the file you created in the previous step. If you want to have the change take effect in the next reboot, use the **--config** option. If you want this change to be persistent, use the **--persistent** option. If you want the change to take effect on the current domain, use the **--current** option. See the Virsh man page for additional information.

4. If you want to detach the device (hot unplug), perform the following command:

```
# virsh detach-device rhel6 --file usb_device.xml
```

In this example [rhel6] is the name of your guest virtual machine and [usb_device.xml] is the file you attached in the previous step

14.3. ATTACHING INTERFACE DEVICES

The **virsh attach-interface*domain type source*** command can take the following options:

- **--live** - get value from running domain
- **--config** - get value to be used on next boot
- **--current** - get value according to current domain state
- **--persistent** - behaves like **--config** for an offline domain, and like **--live** for a running domain.
- **--target** - indicates the target device in the guest virtual machine.
- **--mac** - use this to specify the MAC address of the network interface
- **--script** - use this to specify a path to a script file handling a bridge instead of the default one.
- **--model** - use this to specify the model type.
- **--inbound** - controls the inbound bandwidth of the interface. Acceptable values are **average**, **peak**, and **burst**.
- **--outbound** - controls the outbound bandwidth of the interface. Acceptable values are **average**, **peak**, and **burst**.

The *type* can be either **network** to indicate a physical network device, or **bridge** to indicate a bridge to a device. *source* is the source of the device. To remove the attached device, use the **virsh detach-device**.

14.4. CHANGING THE MEDIA OF A CDROM

Changing the media of a CDROM to another source or format

```
# change-media domain path source --eject --insert --update --current --live --config --force
```

- **--path** - A string containing a fully-qualified path or target of disk device
- **--source** - A string containing the source of the media
- **--eject** - Eject the media
- **--insert** - Insert the media
- **--update** - Update the media
- **--current** - can be either or both of **--live** and **--config**, depends on implementation of hypervisor driver
- **--live** - alter live configuration of running domain
- **--config** - alter persistent configuration, effect observed on next boot
- **--force** - force media changing

14.5. DOMAIN COMMANDS

A domain name is required for most of these commands as they manipulate the specified domain directly. The domain may be given as a short integer (0,1,2...), a name, or a full UUID.

14.5.1. Configuring a Domain to be Started Automatically at Boot

$ virsh autostart [--disable] domain will automatically start the specified domain at boot. Using the **--disable** option disables autostart.

```
# virsh autostart rhel6
```

In the example above, the rhel6 guest virtual machine will automatically start when the host physical machine boots

```
# virsh autostart rhel6 --disable
```

In the example above, the autostart function is disabled and the guest virtual machine will no longer start automatically when the host physical machine boots.

14.5.2. Connecting the Serial Console for the Guest Virtual Machine

The **$ virsh console <domain> [--devname <string>] [--force] [--safe]** command connects the virtual serial console for the guest virtual machine. The optional --devname <string> parameter refers to the device alias of an alternate console, serial, or parallel device configured for the guest virtual machine. If

this parameter is omitted, the primary console will be opened. The **--force** option will force the console connection or when used with disconnect, will disconnect connections. Using the **--safe** option will only allow the guest to connect if safe console handling is supported.

```
$ virsh console virtual_machine --safe
```

14.5.3. Defining a Domain with an XML File

The **define <FILE>** command defines a domain from an XML file. The domain definition in this case is registered but not started. If the domain is already running, the changes will take effect on the next boot.

14.5.4. Editing and Displaying a Description and Title of a Domain

The following command is used to show or modify the description and title of a domain, but does not configure it:

```
# virsh desc [domain-name] [[--live] [--config] | [--current]] [--title] [--edit] [--new-desc New description
or title message]
```

These values are user fields that allow storage of arbitrary textual data to allow easy identification of domains. Ideally, the title should be short, although this is not enforced by libvirt.

The options **--live** or **--config** select whether this command works on live or persistent definitions of the domain. If both **--live** and **--config** are specified, the **--config** option will be implemented first, where the description entered in the command becomes the new configuration setting which is applied to both the live configuration and persistent configuration setting. The **--current** option will modify or get the current state configuration and will not be persistent. The **--current** option will be used if neither **--live** nor **--config**, nor **--current** are specified. The **--edit** option specifies that an editor with the contents of current description or title should be opened and the contents saved back afterwards. Using the **--title** option will show or modify the domain's title field only and not include its description. In addition, if neither **--edit** nor **--new-desc** are used in the command, then only the description is displayed and cannot be modified.

For example, the following command changes the guest virtual machine's title from *testvm* to *TestVM-4F* and will change the description to *Guest VM on fourth floor*:

```
$ virsh desc testvm --current --title TestVM-4F --new-desc Guest VM on fourth floor
```

14.5.5. Displaying Device Block Statistics

This command will display the block statistics for a running domain. You need to have both the domain name and the device name (use the **virsh domblklist** to list the devices.)In this case a block device is the unique target name (<target dev='name'/>) or a source file (< source file ='name'/>). Note that not every hypervisor can display every field. To make sure that the output is presented in its most legible form use the **--human** option, as shown:

```
# virsh domblklist rhel6
Target     Source
------------------------------------------------
vda        /VirtualMachines/rhel6.img
hdc        -

# virsh domblkstat --human rhel6 vda
```

```
Device: vda
 number of read operations:      174670
 number of bytes read:          3219440128
 number of write operations:     23897
 number of bytes written:        164849664
 number of flush operations:     11577
 total duration of reads (ns):   1005410244506
 total duration of writes (ns):  1085306686457
 total duration of flushes (ns): 340645193294
```

14.5.6. Retrieving Network Statistics

The **domnetstat [domain][interface-device]** command displays the network interface statistics for the specified device running on a given domain.

```
# domifstat rhel6 eth0
```

14.5.7. Modifying the Link State of a Domain's Virtual Interface

The following command can either configure a specified interface as up or down:

```
# domif-setlink [domain][interface-device][state]{--config}
```

Using this modifies the status of the specified interface for the specified domain. Note that if you only want the persistent configuration of the domain to be modified, you need to use the **--config**option. It should also be noted that for compatibility reasons, **--persistent** is an alias of **--config**. The "interface device" can be the interface's target name or the MAC address.

```
# domif-setlink rhel6 eth0 up
```

14.5.8. Listing the Link State of a Domain's Virtual Interface

This command can be used to query the state of a specified interface on a given domain. Note that if you only want the persistent configuration of the domain to be modified, you need to use the **--config**option. It should also be noted that for compatibility reasons, **--persistent** is an alias of **--config**. The "interface device" can be the interface's target name or the MAC address.

```
# domif-getlink rhel6 eth0 up
```

14.5.9. Setting Network Interface Bandwidth Parameters

domiftune sets the guest virtual machine's network interface bandwidth parameters. The following format should be used:

```
#virsh domiftune domain interface-device [[--config] [--live] | [--current]] [--inbound
average,peak,burst] [--outbound average,peak,burst]
```

The only required parameter is the domain name and interface device of the guest virtual machine, the **--config**, **--live**, and **--current** functions the same as in Section 14.19, “Setting Schedule Parameters”. If no limit is specified, it will query current network interface setting. Otherwise, alter the limits with the following options:

- <interface-device> This is mandatory and it will set or query the domain's network interface's bandwidth parameters. **interface-device** can be the interface's target name (<target dev='name'/>), or the MAC address.

- If no **--inbound** or **--outbound** is specified, this command will query and show the bandwidth settings. Otherwise, it will set the inbound or outbound bandwidth. average,peak,burst is the same as in **attach-interface** command. Refer to Section 14.3, "Attaching Interface Devices"

14.5.10. Retrieving Memory Statistics for a Running Domain

This command may return varied results depending on the hypervisor you are using.

The **dommemstat [domain] [--period (sec)][[--config][--live]|[--current]]** displays the memory statistics for a running domain. Using the **--period** option requires a time period in seconds. Setting this option to a value larger than 0 will allow the balloon driver to return additional statistics which will be displayed by subsequent **domemstat** commands. Setting the **--period** option to 0, will stop the balloon driver collection but does not clear the statistics in the balloon driver. You cannot use the **--live**, **--config**, or **--current** options without also setting the **--period** option in order to also set the collection period for the balloon driver. If the **--live** option is specified, only the running guest's collection period is affected. If the **--config** option is used, it will affect the next boot of a persistent guest. If **--current** option is used, it will affect the current guest state

Both the **--live** and **--config** options may be used but **--current** is exclusive. If no option is specified, the behavior will be different depending on the guest's state.

```
#virsh domemstat rhel6 --current
```

14.5.11. Displaying Errors on Block Devices

This command is best used following a **domstate** that reports that a domain is paused due to an I/O error. The **domblkerror *domain*** command shows all block devices that are in error state on a given domain and it displays the error message that the device is reporting.

```
# virsh domblkerror rhel6
```

14.5.12. Displaying the Block Device Size

In this case a block device is the unique target name (<target dev='name'/>) or a source file (< source file ='name'/>). To retrieve a list you can run **domblklist**. This **domblkinfo** requires a *domain* name.

```
# virsh domblkinfo rhel6
```

14.5.13. Displaying the Block Devices Associated with a Domain

The **domblklist *domain* --inactive --details** displays a table of all block devices that are associated with the specified domain.

If **--inactive** is specified, the result will show the devices that are to be used at the next boot and will not show those that are currently running in use by the running domain. If **--details** is specified, the disk type and device value will be included in the table. The information displayed in this table can be used with the **domblkinfo** and **snapshot-create**.

```
#domblklist rhel6 --details
```

–

14.5.14. Displaying Virtual Interfaces Associated with a Domain

Running the **domiflist** command results in a table that displays information of all the virtual interfaces that are associated with a specified domain. The **domiflist** requires a *domain* name and optionally can take the **--inactive** option.

If **--inactive** is specified, the result will show the devices that are to be used at the next boot and will not show those that are currently running in use by the running domain.

Commands that require a MAC address of a virtual interface (such as **detach-interface** or **domif-setlink**) will accept the output displayed by this command.

14.5.15. Using blockcommit to Shorten a Backing Chain

This section demonstrates how to use **virsh blockcommit** to shorten a backing chain. For more background on backing chains, see Section 14.5.18, "Disk Image Management with Live Block Copy" .

blockcommit copies data from one part of the chain down into a backing file, allowing you to pivot the rest of the chain in order to bypass the committed portions. For example, suppose this is the current state:

```
base ← snap1 ← snap2 ← active.
```

Using **blockcommit** moves the contents of snap2 into snap1, allowing you to delete snap2 from the chain, making backups much quicker.

Procedure 14.2. virsh blockcommit

- Run the following command:

  ```
  # virsh blockcommit $dom $disk -base snap1 -top snap2 -wait -verbose
  ```

  The contents of snap2 are moved into snap1, resulting in:

  base ← **snap1** ← **active**. Snap2 is no longer valid and can be deleted

> **WARNING**
>
> **blockcommit** will corrupt any file that depends on the ***-base*** option (other than files that depend on the ***-top*** option, as those files now point to the base). To prevent this, do not commit changes into files shared by more than one guest. The ***-verbose*** option allows the progress to be printed on the screen.

14.5.16. Using blockpull to Shorten a Backing Chain

blockpull can be used in in the following applications:

- Flattens an image by populating it with data from its backing image chain. This makes the image file self-contained so that it no longer depends on backing images and looks like this:
  - Before: base.img ← Active
  - After: base.img is no longer used by the guest and Active contains all of the data.
- Flattens part of the backing image chain. This can be used to flatten snapshots into the top-level image and looks like this:
  - Before: base ← sn1 ←sn2 ← active
  - After: base.img ← active. Note that active now contains all data from sn1 and sn2 and neither sn1 nor sn2 are used by the guest.
- Moves the disk image to a new file system on the host. This is allows image files to be moved while the guest is running and looks like this:
  - Before (The original image file): **/fs1/base.vm.img**
  - After: **/fs2/active.vm.qcow2** is now the new file system and **/fs1/base.vm.img** is no longer used.
- Useful in live migration with post-copy storage migration. The disk image is copied from the source host to the destination host after live migration completes.

  In short this is what happens: Before:**/source-host/base.vm.img** After:**/destination-host/active.vm.qcow2**./**source-host/base.vm.img** is no longer used.

Procedure 14.3. Using blockpull to Shorten a Backing Chain

1. It may be helpful to run this command prior to running **blockpull**:

   ```
   # virsh snapshot-create-as $dom $name - disk-only
   ```

2. If the chain looks like this: **base** ← **snap1** ← **snap2** ← **active** run the following:

   ```
   # virsh blockpull $dom $disk snap1
   ```

   This command makes 'snap1' the backing file of active, by pulling data from snap2 into active resulting in: base ← snap1 ← active.

3. Once the **blockpull** is complete, the **libvirt** tracking of the snapshot that created the extra image in the chain is no longer useful. Delete the tracking on the outdated snapshot with this command:

   ```
   # virsh snapshot-delete $dom $name - metadata
   ```

Additional applications of **blockpull** can be done as follows:

- To flatten a single image and populate it with data from its backing image chain:**# virsh blockpull *example-domain* vda - wait**
- To flatten part of the backing image chain:**# virsh blockpull *example-domain* vda - base /path/to/base.img - wait**

- To move the disk image to a new file system on the host:**# virsh snapshot-create *example-domain* - xmlfile /path/to/new.xml - disk-only** followed by **# virsh blockpull *example-domain* vda - wait**

- To use live migration with post-copy storage migration:

  - On the destination run:

    ```
    # qemu-img create -f qcow2 -o backing_file=/source-host/vm.img /destination-host/vm.qcow2
    ```

  - On the source run:

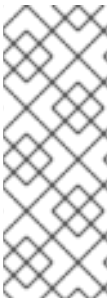
    ```
    # virsh migrate example-domain
    ```

  - On the destination run:

    ```
    # virsh blockpull example-domain vda - wait
    ```

14.5.17. Using blockresize to Change the Size of a Domain Path

blockresize can be used to re-size a block device of a domain while the domain is running, using the absolute path of the block device which also corresponds to a unique target name (**<target dev="name"/>**) or source file (**<source file="name"/>**). This can be applied to one of the disk devices attached to domain (you can use the command **domblklist** to print a table showing the brief information of all block devices associated with a given domain).



NOTE

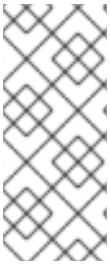
Live image re-sizing will always re-size the image, but may not immediately be picked up by guests. With recent guest kernels, the size of virtio-blk devices is automatically updated (older kernels require a guest reboot). With SCSI devices, it is required to manually trigger a re-scan in the guest with the command, **echo > /sys/class/scsi_device/0:0:0:0/device/rescan**. In addition, with IDE it is required to reboot the guest before it picks up the new size.

- Run the following command: **blockresize [domain] [path size]** where:

  - Domain is the unique target name or source file of the domain whose size you want to change

  - Path size is a scaled integer which defaults to KiB (blocks of 1024 bytes) if there is no suffix. You must use a suffix of "B" to for bytes.

14.5.18. Disk Image Management with Live Block Copy



NOTE

Live block copy is a feature that is not supported with the version of KVM that is supplied with Red Hat Enterprise Linux. Live block copy is available with the version of KVM that is supplied with Red Hat Virtualization. This version of KVM must be running on your physical host machine in order for the feature to be supported. Contact your representative at Red Hat for more details.

Live block copy allows you to copy an in use guest disk image to a destination image and switches the guest disk image to the destination guest image while the guest is running. Whilst live migration moves the memory and registry state of the host, the guest is kept in shared storage. Live block copy allows you to move the entire guest contents to another host on the fly while the guest is running. Live block copy may also be used for live migration without requiring permanent share storage. In this method the disk image is copied to the destination host after migration, but while the guest is running.

Live block copy is especially useful for the following applications:

- moving the guest image from local storage to a central location
- when maintenance is required, guests can be transferred to another location, with no loss of performance
- allows for management of guest images for speed and efficiency
- image format conversions can be done without having to shut down the guest

Example 14.1. Example using live block copy

This example shows what happens when live block copy is performed. The example has a backing file (base) that is shared between a source and destination. It also has two overlays (sn1 and sn2) that are only present on the source and must be copied.

1. The backing file chain at the beginning looks like this:

   base ← sn1 ← sn2

   The components are as follows:

   - base - the original disk image
   - sn1 - the first snapshot that was taken of the base disk image
   - sn2 - the most current snapshot
   - active - the copy of the disk

2. When a copy of the image is created as a new image on top of sn2 the result is this:

   base ← sn1 ← sn2 ← active

3. At this point the read permissions are all in the correct order and are set automatically. To make sure write permissions are set properly, a mirror mechanism redirects all writes to both sn2 and active, so that sn2 and active read the same at any time (and this mirror mechanism is the essential difference between live block copy and image streaming).

4. A background task that loops over all disk clusters is executed. For each cluster, there are the following possible cases and actions:

- The cluster is already allocated in active and there is nothing to do.
- Use **bdrv_is_allocated()** to follow the backing file chain. If the cluster is read from base (which is shared) there is nothing to do.
- If **bdrv_is_allocated()** variant is not feasible, rebase the image and compare the read data with write data in base in order to decide if a copy is needed.
- In all other cases, copy the cluster into **active**

5. When the copy has completed, the backing file of active is switched to base (similar to rebase)

To reduce the length of a backing chain after a series of snapshots, the following commands are helpful: **blockcommit** and **blockpull**. See Section 14.5.15, "Using blockcommit to Shorten a Backing Chain" for more information.

14.5.19. Displaying a URI for Connection to a Graphical Display

Running the **virsh domdisplay** command will output a URI which can then be used to connect to the graphical display of the domain via VNC, SPICE, or RDP. If the **--include-password** option is used, the SPICE channel password will be included in the URI.

14.5.20. Domain Retrieval Commands

The following commands will display different information about a given domain

- **virsh domhostname *domain*** displays the host name of the specified domain provided the hypervisor can publish it.
- **virsh dominfo *domain*** displays basic information about a specified domain.
- **virsh domuid *domain*|*ID*** converts a given domain name or ID into a UUID.
- **virsh domid *domain*|*ID*** converts a given domain name or UUID into an ID.
- **virsh domjobabort *domain*** aborts the currently running job on the specified domain.
- **virsh domjobinfo *domain*** displays information about jobs running on the specified domain, including migration statistics
- **virsh domname *domain ID*|*UUID*** converts a given domain ID or UUID into a domain name.
- **virsh domstate *domain*** displays the state of the given domain. Using the **--reason** option will also display the reason for the displayed state.
- **virsh domcontrol *domain*** displays the state of an interface to VMM that were used to control a domain. For states that are not OK or Error, it will also print the number of seconds that have elapsed since the control interface entered the displayed state.

Example 14.2. Example of statistical feedback

In order to get information about the domain, run the following command:

```
# virsh domjobinfo rhel6
```

```
Job type:         Unbounded
Time elapsed:     1603        ms
Data processed:   47.004 MiB
Data remaining:   658.633 MiB
Data total:       1.125 GiB
Memory processed: 47.004 MiB
Memory remaining: 658.633 MiB
Memory total:     1.125 GiB
Constant pages:   114382
Normal pages:     12005
Normal data:      46.895 MiB
Expected downtime: 0          ms
Compression cache: 64.000 MiB
Compressed data:  0.000 B
Compressed pages: 0
Compression cache misses: 12005
Compression overflows: 0
```

14.5.21. Converting QEMU Arguments to Domain XML

The **virsh domxml-from-native** provides a way to convert an existing set of QEMU arguments into a guest description using libvirt Domain XML that can then be used by libvirt. Note that this command is intended to be used only to convert existing qemu guests previously started from the command line in order to allow them to be managed through libvirt. The method described here should not be used to create new guests from scratch. New guests should be created using either virsh or virt-manager. Additional information can be found here.

Suppose you have a QEMU guest with the following args file:

```
 $ cat demo.args
LC_ALL=C
PATH=/bin
HOME=/home/test
USER=test
LOGNAME=test /usr/bin/qemu -S -M pc -m 214 -smp 1 -nographic -monitor pty -no-acpi -boot c -hda
/dev/HostVG/QEMUGuest1 -net none -serial none -parallel none -usb
```

To convert this to a domain XML file so that the guest can be managed by libvirt, run:

```
$ virsh domxml-from-native qemu-argv demo.args
```

This command turns the args file above, into this domain XML file:

```
<domain type='qemu'>
  <uuid>00000000-0000-0000-0000-000000000000</uuid>
  <memory>219136</memory>
  <currentMemory>219136</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='hd'/>
  </os>
```

```
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/bin/qemu</emulator>
    <disk type='block' device='disk'>
      <source dev='/dev/HostVG/QEMUGuest1'/>
      <target dev='hda' bus='ide'/>
    </disk>
  </devices>
</domain>
```

14.5.22. Creating a Dump File of a Domain's Core

Sometimes it is necessary (especially in the cases of troubleshooting), to create a dump file containing the core of the domain so that it can be analyzed. In this case, running **virsh dump *domain corefilepath* --bypass-cache --live |--crash |--reset --verbose --memory-only** dumps the domain core to a file specified by the *corefilepath* Note that some hypervisors may gave restrictions on this action and may require the user to manually ensure proper permissions on the file and path specified in the *corefilepath* parameter. This command is supported with SR-IOV devices as well as other passthrough devices. The following options are supported and have the following effect:

- **--bypass-cache** the file saved will not contain the file system cache. Note that selecting this option may slow down dump operation.
- **--live** will save the file as the domain continues to run and will not pause or stop the domain.
- **--crash** puts the domain in a crashed status rather than leaving it in a paused state while the dump file is saved.
- **--reset** once the dump file is successfully saved, the domain will reset.
- **--verbose** displays the progress of the dump process
- **--memory-only** the only information that will be saved in the dump file will be the domain's memory and CPU common register file.

Note that the entire process can be monitored using the **domjobinfo** command and can be canceled using the **domjobabort** command.

14.5.23. Creating a Virtual Machine XML Dump (Configuration File)

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

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

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

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

This file **guest.xml** can recreate the guest virtual machine (refer to Section 14.6, "Editing a Guest Virtual Machine's configuration file". You can edit this XML configuration file to configure additional devices or to deploy additional guest virtual machines.

An example of **virsh dumpxml** output:

```
# virsh dumpxml guest1-rhel6-64
<domain type='kvm'>
  <name>guest1-rhel6-64</name>
  <uuid>b8d7388a-bbf2-db3a-e962-b97ca6e514bd</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='threads'/>
      <source file='/home/guest-images/guest1-rhel6-64.img'/>
      <target dev='vda' bus='virtio'/>
      <shareable/<
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </disk>
    <interface type='bridge'>
      <mac address='52:54:00:b9:35:a9'/>
      <source bridge='br0'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <input type='tablet' bus='usb'/>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes'/>
    <sound model='ich6'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
```

```
    </video>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

Note that the <shareable/> flag is set. This indicates the device is expected to be shared between domains (assuming the hypervisor and OS support this), which means that caching should be deactivated for that device.

14.5.24. Creating a Guest Virtual Machine from a Configuration File

Guest virtual machines can be created from XML configuration files. You can copy existing XML from previously created guest virtual machines or use the **dumpxml** option (refer to Section 14.5.23, "Creating a Virtual Machine XML Dump (Configuration File)"). To create a guest virtual machine with **virsh** from an XML file:

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

14.6. EDITING A GUEST VIRTUAL MACHINE'S CONFIGURATION FILE

Instead of using the **dumpxml** option (refer to Section 14.5.23, "Creating a Virtual Machine XML Dump (Configuration File)"), guest virtual machines can be edited either while they are running or while they are offline. The **virsh edit** command provides this functionality. For example, to edit the guest virtual machine named ***rhel6***:

```
# virsh edit rhel6
```

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

14.6.1. Adding Multifunction PCI Devices to KVM Guest Virtual Machines

This section will demonstrate how to add multi-function PCI devices to KVM guest virtual machines.

1. Run the **virsh edit *[guestname]*** command to edit the XML configuration file for the guest virtual machine.

2. In the address type tag, add a **multifunction='on'** entry for **function='0x0'**.

   This enables the guest virtual machine to use the multifunction PCI devices.

   ```
   <disk type='file' device='disk'>
   <driver name='qemu' type='raw' cache='none'/>
   <source file='/var/lib/libvirt/images/rhel62-1.img'/>
   <target dev='vda' bus='virtio'/>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' multifunction='on'/
   </disk>
   ```

   For a PCI device with two functions, amend the XML configuration file to include a second device with the same slot number as the first device and a different function number, such as **function='0x1'**.

   For Example:

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/var/lib/libvirt/images/rhel62-1.img'/>
<target dev='vda' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on'/>
</disk>
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/var/lib/libvirt/images/rhel62-2.img'/>
<target dev='vdb' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1'/>
</disk>
```

3. **lspci** output from the KVM guest virtual machine shows:

```
$ lspci

00:05.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:05.1 SCSI storage controller: Red Hat, Inc Virtio block device
```

14.6.2. Stopping a Running Domain to Restart It Later

virsh managedsave *domain* --bypass-cache --running | --paused | --verbose saves and destroys (stops) a running domain so that it can be restarted from the same state at a later time. When used with a **virsh start** command it is automatically started from this save point. If it is used with the **--bypass-cache** option the save will avoid the filesystem cache. Note that this option may slow down the save process speed.

--verbose displays the progress of the dump process

Under normal conditions, the managed save will decide between using the running or paused state as determined by the state the domain is in when the save is done. However, this can be overridden by using the **--running** option to indicate that it must be left in a running state or by using **--paused** option which indicates it is to be left in a paused state.

To remove the managed save state, use the **virsh managedsave-remove** command which will force the domain to do a full boot the next time it is started.

Note that the entire managed save process can be monitored using the **domjobinfo** command and can also be canceled using the **domjobabort** command.

14.6.3. Displaying CPU Statistics for a Specified Domain

The **virsh cpu-stats *domain* --total start count** command provides the CPU statistical information on the specified domain. By default it shows the statistics for all CPUs, as well as a total. The **--total** option will only display the total statistics.

14.6.4. Saving a Screenshot

The **virsh screenshot** command takes a screenshot of a current domain console and stores it into a file. If however the hypervisor supports more displays for a domain, using the **--screen** and giving a screen ID will specify which screen to capture. In the case where there are multiple graphics cards, where the heads are numerated before their devices, screen ID 5 addresses the second head on the second card.

14.6.5. Sending a Keystroke Combination to a Specified Domain

Using the **virsh send-key *domain* --codeset --holdtime *keycode*** command you can send a sequence as a *keycode* to a specific domain.

Each *keycode* can either be a numeric value or a symbolic name from the corresponding *codeset*. If multiple *keycodes* are specified, thay are all sent simultaneously to the guest virtual machine and as such may be received in random order. If you need distinct keycodes, you must send the **send-key** command multiple times.

```
# virsh send-key rhel6 --holdtime 1000 0xf
```

If a **--holdtime** is given, each keystroke will be held for the specified amount in milliseconds. The **--codeset** allows you to specify a code set, the default being Linux, but the following options are permitted:

- **linux** - choosing this option causes the symbolic names to match the corresponding Linux key constant macro names and the numeric values are those offered by the Linux generic input event subsystems.
- **xt**- this will send a value that is defined by the XT keyboard controller. No symbolic names are provided.
- **atset1** - the numeric values are those that are defined by the AT keyboard controller, set1 (XT compatible set). Extended keycodes from the atset1 may differ from extended keycodes in the XT codeset. No symbolic names are provided.
- **atset2** - The numeric values are those defined by the AT keyboard controller, set 2. No symbolic names are provided.
- **atset3** - The numeric values are those defined by the AT keyboard controller, set 3 (PS/2 compatible). No symbolic names are provided.
- **os_x** - The numeric values are those defined by the OS-X keyboard input subsystem. The symbolic names match the corresponding OS-X key constant macro names.
- **xt_kbd** - The numeric values are those defined by the Linux KBD device. These are a variant on the original XT codeset, but often with different encoding for extended keycodes. No symbolic names are provided.
- **win32** - The numeric values are those defined by the Win32 keyboard input subsystem. The symbolic names match the corresponding Win32 key constant macro names.
- **usb** - The numeric values are those defined by the USB HID specification for keyboard input. No symbolic names are provided.
- **rfb** - The numeric values are those defined by the RFB extension for sending raw keycodes. These are a variant on the XT codeset, but extended keycodes have the low bit of the second bite set, instead of the high bit of the first byte. No symbolic names are provided.

14.6.6. Sending Process Signal Names to Virtual Processes

Using the **virsh send-process-signal *domain-ID PID signame*** command sends the specified signal (identified by its *signame*) to a process running in a virtual domain (specified by the *domain ID*) and identified by its process ID (*PID*).

Either an integer signal constant number or a symbolic signal name can be sent this way. Thus, for example, both of the following commands send the *kill* signal to process ID *187* on the *rhel6* domain:

```
# virsh send-process-signal rhel6 187 kill
# virsh send-process-signal rhel6 187 9
```

For the full list of available signals and their uses, see the virsh(1) and signal(7) manual pages.

14.6.7. Displaying the IP Address and Port Number for the VNC Display

The **virsh vncdisplay** will print the IP address and port number of the VNC display for the specified domain. If the information is unavailable the exit code 1 will be displayed.

```
# virsh vncdisplay rhel6
127.0.0.1:0
```

14.7. NUMA NODE MANAGEMENT

This section contains the commands needed for NUMA node management.

14.7.1. Displaying Node Information

The **nodeinfo** command displays basic information about the node, including the model number, number of CPUs, type of CPU, and size of the physical memory. The output corresponds to **virNodeInfo** structure. Specifically, the "CPU socket(s)" field indicates the number of CPU sockets per NUMA cell.

```
$ virsh nodeinfo
CPU model:           x86_64
CPU(s):              4
CPU frequency:       1199 MHz
CPU socket(s):       1
Core(s) per socket:  2
Thread(s) per core:  2
NUMA cell(s):        1
Memory size:         3715908 KiB
```

14.7.2. Setting NUMA Parameters

The **virsh numatune** can either set or retrieve the NUMA parameters for a specified domain. Within the Domain XML file these parameters are nested within the **<numatune>** element. Without using options, only the current settings are displayed. The **numatune** ***domain*** command requires a specified domain and can take the following options:

- **--mode** – The mode can be set to either **strict**, **interleave**, or **preferred**. Running domains cannot have their mode changed while live unless the domain was started within **strict** mode.
- **--nodeset** contains a list of NUMA nodes that are used by the host physical machine for running the domain. The list contains nodes, each separated by a comma, with a dash **-** used for node ranges and a caret **^** used for excluding a node.
- Only one of the following three options can be used per instance:
  - **--config** will take effect on the next boot of a persistent guest virtual machine.

- **--live** will set the scheduler information of a running guest virtual machine.
- **--current** will affect the current state of the guest virtual machine.

14.7.3. Displaying the Amount of Free Memory in a NUMA Cell

The **virsh freecell** displays the available amount of memory on the machine within a specified NUMA cell. This command can provide one of three different displays of available memory on the machine depending on the options specified. If no options are used, the total free memory on the machine is displayed. Using the **--all** option, it displays the free memory in each cell and the total free memory on the machine. By using a numeric argument or with **--cellno** option along with a cell number it will display the free memory for the specified cell.

14.7.4. Displaying a CPU List

The **nodecpumap** command displays the number of CPUs that are available to the node, whether they are online or not and it also lists the number that are currently online.

```
$ virsh nodecpumap
  CPUs present: 4
  CPUs online: 1
  CPU map: y
```

14.7.5. Displaying CPU Statistics

The **nodecpustats** command displays statistical information about the specified CPU, if the CPU is given. If not, it will display the CPU status of the node. If a percent is specified, it will display the percentage of each type of CPU statistics that were recorded over an one (1) second interval.

This example shows no CPU specified:

```
$ virsh nodecpustats
user:            1056442260000000
system:           401675280000000
idle:            7549613380000000
iowait:            94593570000000
```

This example shows the statistical percentages for CPU number 2:

```
$ virsh nodecpustats 2 --percent
usage:          2.0%
user:           1.0%
system:         1.0%
idle:          98.0%
iowait:         0.0%
```

You can control the behavior of the rebooting guest virtual machine by modifying the **on_reboot** element in the guest virtual machine's configuration file.

14.7.6. Suspending the Host Physical Machine

The **nodesuspend** command puts the host physical machine into a system-wide sleep state similar to that of Suspend-to-RAM (s3), Suspend-to-Disk (s4), or Hybrid-Suspend and sets up a Real-Time-

Clock to wake up the node after the duration that is set has past. The **--target** option can be set to either ***mem***,***disk***, or ***hybrid***. These options indicate to set the memory, disk, or combination of the two to suspend. Setting the **--duration** instructs the host physical machine to wake up after the set duration time has run out. It is set in seconds. It is recommended that the duration time be longer than 60 seconds.

```
$ virsh nodesuspend disk 60
```

14.7.7. Setting and Displaying the Node Memory Parameters

The **node-memory-tune [shm-pages-to-scan] [shm-sleep-milisecs] [shm-merge-across-nodes]** command displays and allows you to set the node memory parameters. There are three parameters that may be set with this command:

- **shm-pages-to-scan** - sets the number of pages to scan before the shared memory service goes to sleep.
- **shm-sleep-milisecs** - sets the number of milliseconds that the shared memory service will sleep before the next scan
- **shm-merge-across-nodes** - specifies if pages from different NUMA nodes can be merged. Values allowed are ***0*** and ***1***. When set to ***0***, the only pages that can be merged are those that are physically residing in the memory area of the same NUMA node. When set to ***1***, pages from all of the NUMA nodes can be merged. The default setting is ***1***.

14.7.8. Creating Devices on Host Nodes

The **virsh nodedev-create *file*** command allows you to create a device on a host node and then assign it to a guest virtual machine. **libvirt** normally detects which host nodes are available for use automatically, but this command allows for the registration of host hardware that **libvirt** did not detect. The *file* should contain the XML for the top level **<device>** description of the node device.

To stop this device, use the **nodedev-destroy *device*** command.

14.7.9. Detaching a Node Device

The **virsh nodedev-detach** detaches the nodedev from the host so it can be safely used by guests via **<hostdev>** passthrough. This action can be reversed with the **nodedev-reattach** command but it is done automatically for managed services. This command also accepts **nodedev-dettach**.

Note that different drivers expect the device to be bound to different dummy devices. Using the **--driver** option allows you to specify the desired back-end driver.

14.7.10. Retrieving a Device's Configuration Settings

The **virsh nodedev-dumpxml *[device]*** command dumps the XML configuration file for the given node **<device>**. The XML configuration includes information such as: the device name, which bus owns for example the device, the vendor, and product ID. The argument *device* can either be a device name or a WWN pair in WWNN | WWPN format (HBA only).

14.7.11. Listing Devices on a Node

The **virsh nodedev-list** ***cap*** **--tree** command lists all the devices available on the node that are known by **libvirt**. *cap* is used to filter the list by capability types, each separated by a comma and cannot be used with **--tree**. Using the **--tree** option, puts the output into a tree structure as shown:

```
 #  virsh nodedev-list --tree
 computer
|
+- net_lo_00_00_00_00_00_00
+- net_macvtap0_52_54_00_12_fe_50
+- net_tun0
+- net_virbr0_nic_52_54_00_03_7d_cb
+- pci_0000_00_00_0
+- pci_0000_00_02_0
+- pci_0000_00_16_0
+- pci_0000_00_19_0
|   |
|   +- net_eth0_f0_de_f1_3a_35_4f

(this is a partial screen)
```

14.7.12. Triggering a Reset for a Node

The **nodedev-reset** ***nodedev*** command triggers a device reset for the specified *nodedev*. Running this command is useful prior to transferring a node device between guest virtual machine passthrough and the host physical machine. libvirt will do this action implicitly when required, but this command allows an explicit reset when needed.

14.8. STARTING, SUSPENDING, RESUMING, SAVING, AND RESTORING A GUEST VIRTUAL MACHINE

This section provides information on starting, suspending, resuming, saving, and restoring guest virtual machines.

14.8.1. Starting a Defined Domain

The **virsh start** ***domain*** **--console --paused --autodestroy --bypass-cache --force-boot --pass-fds** command starts a inactive domain that was already defined but whose state is inactive since its last managed save state or a fresh boot. The command can take the following options:

- **--console** - will boot the domain attaching to the console
- **--paused** - If this is supported by the driver it will boot the domain and then put it into a paused state
- **--autodestroy** - the guest virtual machine is automatically destroyed when the virsh session closes or the connection to libvirt closes, or it otherwise exits
- **--bypass-cache** - used if the domain is in the managedsave state. If this is used, it will restore the guest virtual machine, avoiding the system cache. Note this will slow down the restore process.
- **--force-boot** - discards any managedsave options and causes a fresh boot to occur

- **--pass-fds** - is a list of additional options separated by commas, which are passed onto the guest virtual machine.

14.8.2. Suspending a Guest Virtual Machine

Suspend a guest virtual machine with **virsh**:

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

When a guest virtual machine is in a suspended state, it consumes system RAM but not processor resources. Disk and network I/O does not occur while the guest virtual machine is suspended. This operation is immediate and the guest virtual machine can be restarted with the **resume** (Section 14.8.6, "Resuming a Guest Virtual Machine") option.

14.8.3. Suspending a Running Domain

The **virsh dompmsuspend *domain* --duration --target** command will take a running domain and suspended it so it can be placed into one of three possible states (S3, S4, or a hybrid of the two).

```
# virsh dompmsuspend rhel6 --duration 100 --target mem
```

This command can take the following options:

- **--duration** - sets the duration for the state change in seconds
- **--target** - can be either **mem (suspend to RAM (S3))disk (suspend to disk (S4))**, or **hybrid (hybrid suspend)**

14.8.4. Waking Up a Domain from a pmsuspend State

This command will inject a wake-up alert to a guest that is in a pmsuspend state, rather than waiting for the duration time set to expire. This operation will not fail if the domain is running.

```
# dompmwakeup rhel6
```

This command requires the name of the domain, *rhel6* for example as shown.

14.8.5. Undefining a Domain

```
# virsh undefine domain --managed-save --snapshots-metadata --storage --remove-all-storage --
wipe-storage
```

This command will undefine a domain. Although it can work on a running domain, it will convert the running domain into a transient domain without stopping it. If the domain is inactive, the domain configuration is removed.

The command can take the following options:

- **--managed-save** - this option guarantees that any managed save image is also cleaned up. Without using this option, attempts to undefine a domain with a managed save image will fail.
- **--snapshots-metadata** - this option guarantees that any snapshots (as shown with **snapshot-list**) are also cleaned up when undefining an inactive domain. Note that any attempts to

undefine an inactive domain whose configuration file contains snapshot metadata will fail. If this option is used and the domain is active, it is ignored.

- **--storage** - using this option requires a comma separated list of volume target names or source paths of storage volumes to be removed along with the undefined domain. This action will undefine the storage volume before it is removed. Note that this can only be done with inactive domains. Note too that this will only work with storage volumes that are managed by libvirt.
- **--remove-all-storage** - in addition to undefining the domain, all associated storage volumes are deleted.
- **--wipe-storage** - in addition to deleting the storage volume, the contents are wiped.

14.8.6. Resuming a Guest Virtual Machine

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

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

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

14.8.7. Save a Guest Virtual Machine

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

```
# virsh save {domain-name|domain-id|domain-uuid} state-file --bypass-cache --xml --running --
paused --verbose
```

This stops the guest virtual machine you specify and saves the data to a file, which may take some time given the amount of memory in use by your guest virtual machine. You can restore the state of the guest virtual machine with the **restore** (Section 14.8.11, “Restore a Guest Virtual Machine”) option. Save is similar to pause, instead of just pausing a guest virtual machine the present state of the guest virtual machine is saved.

The **virsh save** command can take the following options:

- **--bypass-cache** - causes the restore to avoid the file system cache but note that using this option may slow down the restore operation.
- **--xml** - this option must be used with an XML file name. Although this option is usually omitted, it can be used to supply an alternative XML file for use on a restored guest virtual machine with changes only in the host-specific portions of the domain XML. For example, it can be used to account for the file naming differences in underlying storage due to disk snapshots taken after the guest was saved.
- **--running** - overrides the state recorded in the save image to start the domain as running.
- **--paused**- overrides the state recorded in the save image to start the domain as paused.
- **--verbose** - displays the progress of the save.

If you want to restore the guest virtual machine directly from the XML file, the **virsh restore** command will do just that. You can monitor the process with the **domjobinfo** and cancel it with the **domjobabort**.

14.8.8. Updating the Domain XML File that will be Used for Restoring the Guest

The **virsh save-image-define *file xml* --running|--paused** command will update the domain XML file that will be used when the specified file is later used during the **virsh restore** command. The *xml* argument must be an XML file name containing the alternative XML with changes only in the host physical machine specific portions of the domain XML. For example, it can be used to account for the file naming differences resulting from creating disk snapshots of underlying storage after the guest was saved. The save image records if the domain should be restored to a running or paused state. Using the options **--running** or **--paused** dictates the state that is to be used.

14.8.9. Extracting the Domain XML File

save-image-dumpxml *file* --security-info command will extract the domain XML file that was in effect at the time the saved state file (used in the **virsh save** command) was referenced. Using the **--security-info** option includes security sensitive information in the file.

14.8.10. Edit Domain XML Configuration Files

save-image-edit *file* --running --paused command edits the XML configuration file that is associated with a saved *file* that was created by the **virsh save** command.

Note that the save image records whether the domain should be restored to a **--running** or **--paused** state. Without using these options the state is determined by the file itself. By selecting **--running** or **--paused** you can overwrite the state that **virsh restore** should use.

14.8.11. Restore a Guest Virtual Machine

Restore a guest virtual machine previously saved with the **virsh save** command (Section 14.8.7, "Save a Guest Virtual Machine") using **virsh**:

```
# virsh restore state-file
```

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

The **virsh restore *state-file*** command can take the following options:

- **--bypass-cache** - causes the restore to avoid the file system cache but note that using this option may slow down the restore operation.
- **--xml** - this option must be used with an XML file name. Although this option is usually omitted, it can be used to supply an alternative XML file for use on a restored guest virtual machine with changes only in the host-specific portions of the domain XML. For example, it can be used to account for the file naming differences in underlying storage due to disk snapshots taken after the guest was saved.
- **--running** - overrides the state recorded in the save image to start the domain as running.
- **--paused**- overrides the state recorded in the save image to start the domain as paused.

14.9. SHUTTING DOWN, REBOOTING, AND FORCING SHUTDOWN OF A GUEST VIRTUAL MACHINE

This section provides information about shutting down, rebooting, and forcing shutdown of a guest virtual machine.

14.9.1. Shutting Down a Guest Virtual Machine

Shut down a guest virtual machine using the **virsh shutdown** command:

```
# virsh shutdown {domain-id, domain-name or domain-uuid} [--mode method]
```

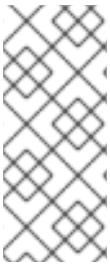
You can control the behavior of the rebooting guest virtual machine by modifying the **on_shutdown** parameter in the guest virtual machine's configuration file.

14.9.2. Shutting Down Red Hat Enterprise Linux 6 Guests on a Red Hat Enterprise Linux 7 Host

Installing Red Hat Enterprise Linux 6 guest virtual machines with the **Minimal installation** option does not install the acpid package. Red Hat Enterprise Linux 7 no longer requires this package, as it has been taken over by **systemd**. However, Red Hat Enterprise Linux 6 guest virtual machines running on a Red Hat Enterprise Linux 7 host still require it.

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

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



NOTE

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

Procedure 14.4. Workaround for Red Hat Enterprise Linux 6 guests

1. **Install the acpid package**
   The **acpid** service listen and processes ACPI requests.

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

   ```
   # yum install acpid
   ```

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

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

3. **Prepare guest domain xml**

Edit the domain XML file to include the following element. Replace the virtio serial port with **org.qemu.guest_agent.0** and use your guest's name instead of *$guestname*

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/{$guestname}.agent'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

Figure 14.2. Guest XML replacement

4. **Install the QEMU guest agent**
   Install the QEMU guest agent (QEMU-GA) and start the service as directed in Chapter 10, *QEMU-img and QEMU Guest Agent*. If you are running a Windows guest there are instructions in this chapter for that as well.

5. **Shutdown the guest**

   a. Run the following commands

   ```
   # virsh list --all  - this command lists all of the known domains
     Id Name              State
   ----------------------------------
     rhel6               running
   ```

   b. Shut down the guest virtual machine

   ```
   # virsh shutdown rhel6

   Domain rhel6 is being shutdown
   ```

   c. Wait a few seconds for the guest virtual machine to shut down.

   ```
   # virsh list --all
    Id Name                State
   ----------------------------------
     . rhel6               shut off
   ```

   d. Start the domain named *rhel6*, with the XML file you edited.

   ```
   # virsh start rhel6
   ```

   e. Shut down the acpi in the *rhel6* guest virtual machine.

   ```
   # virsh shutdown --mode acpi rhel6
   ```

   f. List all the domains again, *rhel6* should still be on the list, and it should indicate it is shut off.

   ```
   # virsh list --all
     Id Name                State
   ----------------------------------
   ```

```
  rhel6                shut off
```

g. Start the domain named *rhel6*, with the XML file you edited.

```
# virsh start rhel6
```

h. Shut down the *rhel6* guest virtual machine guest agent.

```
# virsh shutdown --mode agent rhel6
```

i. List the domains. *rhel6* should still be on the list, and it should indicate it is shut off

```
# virsh list --all
  Id Name                State
----------------------------------
  rhel6                shut off
```

The guest virtual machine will shut down using the **virsh shutdown** command for the consecutive shutdowns, without using the workaround described above.

In addition to the method described above, a guest can be automatically shutdown, by stopping the **libvirt-guest** service. Refer to Section 14.9.3, "Manipulating the libvirt-guests Configuration Settings" for more information on this method.

14.9.3. Manipulating the libvirt-guests Configuration Settings

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

Procedure 14.5. Changing the libvirt-guests service parameters to allow for the graceful shutdown of guests

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

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

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

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

# action taken on host boot
# - start   all guests which were running on shutdown are started on boot
```

```
#           regardless on their autostart settings

# - ignore  libvirt-guests init script won't start any guest on boot, however,

#           guests marked as autostart will still be automatically started by

#           libvirtd

#ON_BOOT=start

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

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

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

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

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

1	***URIS*** - checks the specified connections for a running guest. The ***Default*** setting functions in the same manner as **virsh** does when no explicit URI is set In addition, one can explicitly set the URI from **/etc/libvirt/libvirt.conf**. It should be noted that when using the libvirt configuration file default setting, no probing will be used.
2	***ON_BOOT*** - specifies the action to be done to / on the guests when the host boots. The ***start*** option starts all guests that were running prior to shutdown regardless on their autostart settings. The ***ignore*** option will not start the formally running guest on boot, however, any guest marked as autostart will still be automatically started by libvirtd.

3	The ***START_DELAY*** - sets a delay interval in between starting up the guests. This time period is set in seconds. Use the 0 time setting to make sure there is no delay and that all guests are started simultaneously.
4	***ON_SHUTDOWN*** - specifies the action taken when a host shuts down. Options that can be set include: ***suspend*** which suspends all running guests using **virsh managedsave** and ***shutdown*** which shuts down all running guests. It is best to be careful with using the **shutdown** option as there is no way to distinguish between a guest which is stuck or ignores shutdown requests and a guest that just needs a longer time to shutdown. When setting the ***ON_SHUTDOWN=shutdown***, you must also set ***SHUTDOWN_TIMEOUT*** to a value suitable for the guests.
5	***PARALLEL_SHUTDOWN*** Dictates that the number of guests on shutdown at any time will not exceed number set in this variable and the guests will be suspended concurrently. If set to **0**, then guests are not shutdown concurrently.
6	Number of seconds to wait for a guest to shut down. If ***SHUTDOWN_TIMEOUT*** is enabled, this timeout applies as a timeout for shutting down all guests on a single URI defined in the variable URIS. If ***SHUTDOWN_TIMEOUT*** is set to ***0***, then there is no time out (use with caution, as guests might not respond to a shutdown request). The default value is 300 seconds (5 minutes).
7	***BYPASS_CACHE*** can have 2 values, 0 to disable and 1 to enable. If enabled it will by-pass the file system cache when guests are restored. Note that setting this may effect performance and may cause slower operation for some file systems.

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

14.9.4. Rebooting a Guest Virtual Machine

Use the **virsh reboot** command to reboot a guest virtual machine. The prompt will return once the reboot has executed. Note that there may be a time lapse until the guest virtual machine returns.

```
#virsh reboot {domain-id, domain-name or domain-uuid} [--mode method]
```

You can control the behavior of the rebooting guest virtual machine by modifying the **<on_reboot>** element in the guest virtual machine's configuration file. Refer to Section 20.12, "Events Configuration" for more information.

By default, the hypervisor will try to pick a suitable shutdown method. To specify an alternative method, the **--mode** option can specify a comma separated list which includes **initctl**, **acpi**, **agent**, and **signal**. The order in which drivers will try each mode is not related to the order specified in the command. For strict control over ordering, use a single mode at a time and repeat the command.

14.9.5. Forcing a Guest Virtual Machine to Stop

Force a guest virtual machine to stop with the **virsh destroy** command:

```
# virsh destroy {domain-id, domain-name or domain-uuid} [--graceful]
```

This command does an immediate ungraceful shutdown and stops the specified guest virtual machine. Using **virsh destroy** can corrupt guest virtual machine file systems. Use the **destroy** option only when

the guest virtual machine is unresponsive. If you want to initiate a graceful shutdown, use the **virsh destroy --graceful** command.

14.9.6. Resetting a Virtual Machine

virsh reset ***domain*** resets the domain immediately without any guest shutdown. A reset emulates the power reset button on a machine, where all guest hardware sees the RST line and re-initializes the internal state. Note that without any guest virtual machine OS shutdown, there are risks for data loss.

14.10. RETRIEVING GUEST VIRTUAL MACHINE INFORMATION

This section provides information on retrieving guest virtual machine information.

14.10.1. Getting the Domain ID of a Guest Virtual Machine

To get the domain ID of a guest virtual machine:

```
# virsh domid {domain-name or domain-uuid}
```

14.10.2. Getting the Domain Name of a Guest Virtual Machine

To get the domain name of a guest virtual machine:

```
# virsh domname {domain-id or domain-uuid}
```

14.10.3. Getting the UUID of a Guest Virtual Machine

To get the Universally Unique Identifier (UUID) for a guest virtual machine:

```
# virsh domuuid {domain-id or domain-name}
```

An example of **virsh domuuid** output:

```
# virsh domuuid r5b2-mySQL01
4a4c59a7-ee3f-c781-96e4-288f2862f011
```

14.10.4. Displaying Guest Virtual Machine Information

Using **virsh** with the guest virtual machine's domain ID, domain name or UUID you can display information on the specified guest virtual machine:

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

This is an example of **virsh dominfo** output:

```
# virsh dominfo vr-rhel6u1-x86_64-kvm
Id:          9
Name:        vr-rhel6u1-x86_64-kvm
UUID:        a03093a1-5da6-a2a2-3baf-a845db2f10b9
OS Type:     hvm
State:       running
```

```
CPU(s):         1
CPU time:       21.6s
Max memory:     2097152 kB
Used memory:    1025000 kB
Persistent:     yes
Autostart:      disable
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c612,c921 (permissive)
```

14.11. STORAGE POOL COMMANDS

The following commands manipulate storage pools. Using libvirt you can manage various storage solutions, including files, raw partitions, and domain-specific formats, used to provide the storage volumes visible as devices within virtual machines. For more detailed information about this feature, see more information at libvirt.org. Many of the commands for storage pools are similar to the ones used for domains.

14.11.1. Searching for a Storage Pool XML

The **find-storage-pool-sources *type srcSpec*** command displays the XML describing all storage pools of a given *type* that could be found. If *srcSpec* is provided, it is a file that contains XML to further restrict the query for pools.

The **find-storage-pool-sources-as *type host port initiator*** displays the XML describing all storage pools of a given *type* that could be found. If *host*, *port*, or *initiator* are provided, they control where the query is performed.

The **pool-info *pool-or-uuid*** command will list the basic information about the specified storage pool object. This command requires the name or UUID of the storage pool. To retrieve this information, use the following coomand:

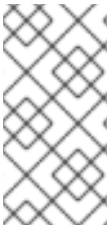
```
pool-list [--inactive] [--all] [--persistent] [--transient] [--autostart] [--no-autostart] [--details] type
```

This lists all storage pool objects known to libvirt. By default, only active pools are listed; but using the **--inactive** option lists just the inactive pools, and using the **--all** option lists all of the storage pools.

In addition to those options there are several sets of filtering options that can be used to filter the content of the list. **--persistent** restricts the list to persistent pools, **--transient** restricts the list to transient pools, **--autostart** restricts the list to autostarting pools and finally **--no-autostart** restricts the list to the storage pools that have autostarting disabled.

For all storage pool commands which require a *type*, the pool types must be separated by comma. The valid pool types include: **dir**, **fs**, **netfs**, **logical**, **disk**, **iscsi**, **scsi**, **mpath**, **rbd**, and **sheepdog**.

The **--details** option instructs **virsh** to additionally display pool persistence and capacity related information where available.



NOTE

When this command is used with older servers, it is forced to use a series of API calls with an inherent race, where a pool might not be listed or might appear more than once if it changed its state between calls while the list was being collected. Newer servers however, do not have this problem.

The **pool-refresh** ***pool-or-uuid*** refreshes the list of volumes contained in pool.

14.11.2. Creating, Defining, and Starting Storage Pools

This section provides information about creating, defining, and starting storage pools.

14.11.2.1. Building a storage pool

The **pool-build** ***pool-or-uuid*** **--overwrite --no-overwrite** command builds a pool with a specified *pool name or UUID*. The options **--overwrite** and **--no-overwrite** can only be used for a pool whose type is file system. If neither option is specified, and the pool is a file system type pool, then the resulting build will only make the directory.

If **--no-overwrite** is specified, it probes to determine if a file system already exists on the target device, returning an error if it exists, or using **mkfs** to format the target device if it does not. If **--overwrite** is specified, then the **mkfs** command is executed and any existing data on the target device is overwritten.

14.11.2.2. Creating and defining a storage pool from an XML file

The **pool-create** ***file*** creates and starts a storage pool from its associated XML file.

The **pool-define** ***file*** creates, but does not start, a storage pool object from the XML file.

14.11.2.3. Creating and starting a storage pool from raw parameters

```
# pool-create-as name --print-xml type source-host source-path source-dev source-name <target> --
source-format format
```

This command creates and starts a pool object name from the raw parameters given.

If **--print-xml** is specified, then it prints the XML of the storage pool object without creating the pool. Otherwise, the pool requires a type in order to be built. For all storage pool commands which require a *type*, the pool types must be separated by comma. The valid pool types include: **dir**, **fs**, **netfs**, **logical**, **disk**, **iscsi**, **scsi**, **mpath**, **rbd**, and **sheepdog**.

In contrast, the following command creates, but does not start, a pool object name from the raw parameters given:

```
# pool-define-as name --print-xml type source-host source-path source-dev source-name <target> --
source-format format
```

If **--print-xml** is specified, then it prints the XML of the pool object without defining the pool. Otherwise, the pool has to have a specified type. For all storage pool commands which require a *type*, the pool types must be separated by comma. The valid pool types include: **dir**, **fs**, **netfs**, **logical**, **disk**, **iscsi**, **scsi**, **mpath**, **rbd**, and **sheepdog**.

The **pool-start** ***pool-or-uuid*** starts the specified storage pool, which was previously defined but inactive.

14.11.2.4. Auto-starting a storage pool

The **pool-autostart** ***pool-or-uuid*** **--disable** command enables or disables a storage pool to automatically start at boot. This command requires the pool name or UUID. To disable the **pool-autostart** command use the **--disable** option.

14.11.3. Stopping and Deleting Storage Pools

The **pool-destroy** ***pool-or-uuid*** stops a storage pool. Once stopped, libvirt will no longer manage the pool but the raw data contained in the pool is not changed, and can be later recovered with the **pool-create** command.

The **pool-delete** ***pool-or-uuid*** destroys the resources used by the specified storage pool. It is important to note that this operation is non-recoverable and non-reversible. However, the pool structure will still exist after this command, ready to accept the creation of new storage volumes.

The **pool-undefine** ***pool-or-uuid*** command undefines the configuration for an inactive pool.

14.11.4. Creating an XML Dump File for a Storage Pool

The **pool-dumpxml --inactive** ***pool-or-uuid*** command returns the XML information about the specified storage pool object. Using **--inactive** dumps the configuration that will be used on next start of the pool as opposed to the current pool configuration.

14.11.5. Editing the Storage Pool's Configuration File

The **pool-edit** ***pool-or-uuid*** opens the specified storage pool's XML configuration file for editing.

This method is the only method that should be used to edit an XML configuration file as it does error checking before applying.

14.11.6. Converting Storage Pools

The **pool-name** ***uuid*** command converts the specified UUID to a pool name.

The **pool-uuid** ***pool*** command returns the UUID of the specified pool.

14.12. STORAGE VOLUME COMMANDS

This section covers all commands for creating, deleting, and managing storage volumes. It is best to do this once you have created a storage pool as the storage pool name or UUID will be required. For information on storage pools refer to Chapter 12, *Storage Pools*. For information on storage volumes refer to, Chapter 13, *Volumes* .

14.12.1. Creating Storage Volumes

The **vol-create-from** ***pool-or-uuid file*** **--inputpool** ***pool-or-uuid vol-name-or-key-or-path*** command creates a storage volume, using another storage volume as a template for its contents. This command requires a *pool-or-uuid* which is the name or UUID of the storage pool to create the volume in.

The *file* argument specifies the XML file and path containing the volume definition. The **--inputpool** ***pool-or-uuid*** option specifies the name or uuid of the storage pool the source volume is in. The *vol-name-or-key-or-path* argument specifies the name or key or path of the source volume. For some examples, refer to Section 13.1, “Creating Volumes”.

The **vol-create-as** command creates a volume from a set of arguments. The *pool-or-uuid* argument contains the name or UUID of the storage pool to create the volume in.

```
vol-create-as pool-or-uuid name capacity --allocation <size> --format <string> --backing-vol <vol-name-or-key-or-path> --backing-vol-format <string>
```

name is the name of the new volume. *capacity* is the size of the volume to be created, as a scaled integer, defaulting to bytes if there is no suffix. **--allocation <size>** is the initial size to be allocated in the volume, also as a scaled integer defaulting to bytes. **--format <string>** is used in file based storage pools to specify the volume file format which is a string of acceptable formats separated by a comma. Acceptable formats include **raw**, **bochs**, **qcow**, **qcow2**, **vmdk**, **--backing-vol *vol-name-or-key-or-path*** is the source backing volume to be used if taking a snapshot of an existing volume. **--backing-vol-format *string*** is the format of the snapshot backing volume which is a string of formats separated by a comma. Accepted values include: **raw**, **bochs**, **qcow**, **qcow2**, , **vmdk**, and **host_device**. These are, however, only meant for file based storage pools.

14.12.1.1. Creating a storage volume from an XML file

The **vol-create *pool-or-uuid file*** creates a storage volume from a saved XML file. This command also requires the *pool-or-uuid*, which is the name or UUID of the storage pool in which the volume will be created. The *file* argument contains the path with the volume definition's XML file. An easy way to create the XML file is to use the **vol-dumpxml** command to obtain the definition of a pre-existing volume, modify it and then save it and then run the **vol-create**.

```
virsh vol-dumpxml --pool storagepool1 appvolume1 > newvolume.xml
virsh edit newvolume.xml
virsh vol-create differentstoragepool newvolume.xml
```

Other options available include:

- The **--inactive** option lists the inactive guest virtual machines (that is, guest virtual machines that have been defined but are not currently active).
- The **--all** option lists all guest virtual machines.

14.12.1.2. Cloning a storage volume

The **vol-clone --pool *pool-or-uuid vol-name-or-key-or-path name*** command clones an existing storage volume. Although the **vol-create-from** may also be used, it is not the recommended way to clone a storage volume. The **--pool *pool-or-uuid*** option is the name or UUID of the storage pool to create the volume in. The *vol-name-or-key-or-path* argument is the name or key or path of the source volume. Using a *name* argument refers to the name of the new volume.

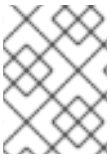
14.12.2. Deleting Storage Volumes

The **vol-delete --pool *pool-or-uuid vol-name-or-key-or-path*** command deletes a given volume. The command requires a specific **--pool *pool-or-uuid*** which is the name or UUID of the storage pool the volume is in. The ***vol-name-or-key-or-path*** option specifies the name or key or path of the volume to delete.

The **vol-wipe --pool *pool-or-uuid* --algorithm *algorithm vol-name-or-key-or-path*** command wipes a volume, to ensure data previously on the volume is not accessible to future reads. The command requires a **--pool *pool-or-uuid***, which is the name or UUID of the storage pool the volume is in. The *vol-name-or-key-or-path* contains the name or key or path of the volume to wipe. Note it is possible to choose different wiping algorithms instead of the default (where every sector of the storage volume is written with value "0"). To specify a wiping algorithm, use the **--algorithm** option with one of the following supported *algorithm* types:

- **zero** - 1-pass all zeroes

- **nnsa** - 4-pass NNSA Policy Letter NAP-14.1-C (XVI-8) for sanitizing removable and non-removable hard disks: random x2, 0x00, verify.
- **dod** - 4-pass DoD 5220.22-M section 8-306 procedure for sanitizing removable and non-removable rigid disks: random, 0x00, 0xff, verify.
- **bsi** - 9-pass method recommended by the German Center of Security in Information Technologies (http://www.bsi.bund.de): 0xff, 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f.
- **gutmann** - The canonical 35-pass sequence described in Gutmann's paper.
- **schneier** - 7-pass method described by Bruce Schneier in "Applied Cryptography" (1996): 0x00, 0xff, random x5.
- **pfitzner7** - Roy Pfitzner's 7-random-pass method: random x7
- **pfitzner33** - Roy Pfitzner's 33-random-pass method: random x33.
- **random** - 1-pass pattern: random.



NOTE

The version of the scrub binary installed on the host will limit the algorithms that are available.

14.12.3. Dumping Storage Volume Information to an XML File

vol-dumpxml --pool *pool-or-uuid vol-name-or-key-or-path* command takes the volume information as an XML dump to a specified file.

This command requires a **--pool *pool-or-uuid***, which is the name or UUID of the storage pool the volume is in. *vol-name-or-key-or-path* is the name or key or path of the volume to place the resulting XML file.

14.12.4. Listing Volume Information

The **vol-info --pool *pool-or-uuid vol-name-or-key-or-path*** command lists basic information about the given storage volume **--pool**, where *pool-or-uuid* is the name or UUID of the storage pool the volume is in. *vol-name-or-key-or-path* is the name or key or path of the volume to return information for.

The **vol-list --pool *pool-or-uuid* --details** lists all of volumes in the specified storage pool. This command requires **--pool *pool-or-uuid*** which is the name or UUID of the storage pool. The **--details** option instructs virsh to additionally display volume type and capacity related information where available.

14.12.5. Retrieving Storage Volume Information

The **vol-pool --uuid *vol-key-or-path*** command returns the pool name or UUID for a given volume. By default, the pool name is returned. If the **--uuid** option is given, the pool UUID is returned instead. The command requires the *vol-key-or-path* which is the key or path of the volume for which to return the requested information.

The **vol-path --pool *pool-or-uuid vol-name-or-key*** command returns the path for a given volume. The command requires **--pool *pool-or-uuid***, which is the name or UUID of the storage pool the volume is in. It also requires *vol-name-or-key* which is the name or key of the volume for which the path has been requested.

The **vol-name *vol-key-or-path*** command returns the name for a given volume, where *vol-key-or-path* is the key or path of the volume to return the name for.

The **vol-key --pool *pool-or-uuid vol-name-or-path*** command returns the volume key for a given volume where **--pool *pool-or-uuid*** is the name or UUID of the storage pool the volume is in and *vol-name-or-path* is the name or path of the volume to return the volume key for.

14.12.6. Uploading and Downloading Storage Volumes

This section will instruct how to upload and download information to and from storage volumes.

14.12.6.1. Uploading contents to a storage volume

The **vol-upload --pool *pool-or-uuid* --offset bytes --length bytes *vol-name-or-key-or-path local-file*** command uploads the contents of specified *local-file* to a storage volume. The command requires **--pool *pool-or-uuid*** which is the name or UUID of the storage pool the volume is in. It also requires *vol-name-or-key-or-path* which is the name or key or path of the volume to wipe. The **--offset** option is the position in the storage volume at which to start writing the data. **--length *length*** dictates an upper limit for the amount of data to be uploaded. An error will occur if the *local-file* is greater than the specified **--length**.

14.12.6.2. Downloading the contents from a storage volume

```
# vol-download --pool pool-or-uuid --offset bytes --length bytes vol-name-or-key-or-path local-file
```

This command command downloads the contents of local-file from a storage volume. It requires a **--pool *pool-or-uuid*** which is the name or UUID of the storage pool that the volume is in. It also requires *vol-name-or-key-or-path* which is the name or key or path of the volume to wipe. Using the option **--offset** dictates the position in the storage volume at which to start reading the data. **--length *length*** dictates an upper limit for the amount of data to be downloaded.

14.12.7. Re-sizing Storage Volumes

```
# vol-resize --pool pool-or-uuid vol-name-or-path pool-or-uuid capacity --allocate --delta --shrink
```

This command command re-sizes the capacity of the given volume, in bytes. The command requires **--pool *pool-or-uuid*** which is the name or UUID of the storage pool the volume is in. This command also requires *vol-name-or-key-or-path* is the name or key or path of the volume to re-size.

The new capacity may create a *sparse file* unless the **--allocate** option is specified. Normally, capacity is the new size, but if **--delta** is present, then it is added to the existing size. Attempts to shrink the volume will fail unless the **--shrink** option is present.

Note that capacity cannot be negative unless the **--shrink** option is provided and a negative sign is not necessary. *capacity* is a scaled integer which defaults to bytes if there is no suffix. Note too that this command is only safe for storage volumes not in use by an active guest. Refer to Section 14.5.17, "Using blockresize to Change the Size of a Domain Path" for live re-sizing.

14.13. DISPLAYING PER-GUEST VIRTUAL MACHINE INFORMATION

This section provides information about displaying virtual machine information for each guest.

14.13.1. Displaying the Guest Virtual Machines

To display the guest virtual machine list and their current states with **virsh**:

```
# virsh list
```

Other options available include:

- **--inactive** option lists the inactive guest virtual machines (that is, guest virtual machines that have been defined but are not currently active)
- **--all** option lists all guest virtual machines. For example:

  ```
  # virsh list --all
   Id Name                 State
  ----------------------------------
    0 Domain-0             running
    1 Domain202            paused
    2 Domain010            inactive
    3 Domain9600           crashed
  ```

  There are seven states that can be visible using this command:

  - Running - The **running** state refers to guest virtual machines which are currently active on a CPU.
  - Idle - The **idle** state indicates that the domain is idle, and may not be running or able to run. This can be caused because the domain is waiting on IO (a traditional wait state) or has gone to sleep because there was nothing else for it to do.
  - Paused - The **paused** state lists domains that are paused. This occurs if an administrator uses the **paused** button in **virt-manager** or **virsh suspend**. When a guest virtual machine is paused it consumes memory and other resources but it is ineligible for scheduling and CPU resources from the hypervisor.
  - Shutdown - The **shutdown** state is for guest virtual machines in the process of shutting down. The guest virtual machine is sent a shutdown signal and should be in the process of stopping its operations gracefully. This may not work with all guest virtual machine operating systems; some operating systems do not respond to these signals.
  - Shut off - The **shut off** state indicates that the domain is not running. This can be caused when a domain completely shuts down or has not been started.
  - Crashed - The **crashed** state indicates that the domain has crashed and can only occur if the guest virtual machine has been configured not to restart on crash.
  - Dying - Domains in the **dying** state are in is in process of dying, which is a state where the domain has not completely shut-down or crashed.

- **--managed-save** Although this option alone does not filter the domains, it will list the domains that have managed save state enabled. In order to actually list the domains separately you will need to use the **--inactive** option as well.
- **--name** is specified domain names are printed in a list. If **--uuid** is specified the domain's UUID is printed instead. Using the option **--table** specifies that a table style output should be used. All three commands are mutually exclusive

- **--title** This command must be used with **--table** output. **--title**will cause an extra column to be created in the table with the short domain description (title).
- **--persistent**includes persistent domains in a list. Use the **--transient** option.
- **--with-managed-save** lists the domains that have been configured with managed save. To list the commands without it, use the command **--without-managed-save**
- **--state-running** filters out for the domains that are running, **--state-paused** for paused domains, **--state-shutoff** for domains that are turned off, and **--state-other** lists all states as a fallback.
- **--autostart** this option will cause the auto-starting domains to be listed. To list domains with this feature disabled, use the option **--no-autostart**.
- **--with-snapshot** will list the domains whose snapshot images can be listed. To filter for the domains without a snapshot, use the option **--without-snapshot**

```
$ virsh list --title --name

 Id    Name                           State    Title
 0     Domain-0                        running  Mailserver1
 2     rhelvm                         paused
```

For an example of **virsh vcpuinfo** output, refer to Section 14.13.2, "Displaying Virtual CPU Information"

14.13.2. Displaying Virtual CPU Information

To display virtual CPU information from a guest virtual machine with **virsh**:

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

An example of **virsh vcpuinfo** output:

```
# virsh vcpuinfo rhel6
VCPU:          0
CPU:           2
State:         running
CPU time:      7152.4s
CPU Affinity:  yyyy

VCPU:          1
CPU:           2
State:         running
CPU time:      10889.1s
CPU Affinity:  yyyy
```

14.13.3. Configuring Virtual CPU Affinity

To configure the affinity of virtual CPUs with physical CPUs, refer to Example 14.3, "Pinning vCPU to a host physical machine's CPU".

Example 14.3. Pinning vCPU to a host physical machine's CPU

The **virsh vcpupin** assigns a virtual CPU to a physical one.

```
# virsh vcpupin rhel6
VCPU: CPU Affinity
----------------------------------
   0: 0-3
   1: 0-3
```

The **vcpupin** can take the following options:

- **--vcpu** requires the vcpu number
- **[--cpulist] >string<** lists the host physical machine's CPU number(s) to set, or omit an optional query
- **--config** affects next boot
- **--live** affects the running domain
- **--current** affects the current domain

14.13.4. Displaying Information about the Virtual CPU Counts of a Domain

virsh vcpucount requires a *domain* name or a domain ID. For example:

```
# virsh vcpucount rhel6
maximum      config         2
maximum      live           2
current      config         2
current      live           2
```

The **vcpucount** can take the following options:

- **--maximum** displays the maximum number of vCPUs available
- **--active** displays the number of currently active vCPUs
- **--live** displays the value from the running domain
- **--config** displays the value to be configured on guest virtual machine's next boot
- **--current** displays the value according to current domain state
- **--guest** displays the count that is returned is from the perspective of the guest

14.13.5. Configuring Virtual CPU Affinity

To configure the affinity of virtual CPUs with physical CPUs:

```
# virsh vcpupin domain-id vcpu cpulist
```

The **domain-id** parameter is the guest virtual machine's ID number or name.

The **vcpu** parameter denotes the number of virtualized CPUs allocated to the guest virtual machine.The **vcpu** parameter must be provided.

The **cpulist** parameter is a list of physical CPU identifier numbers separated by commas. The **cpulist** parameter determines which physical CPUs the VCPUs can run on.

Additional parameters such as **--config** affect the next boot, whereas **--live** affects the running domain, and **--current** affects the current domain.

14.13.6. Configuring Virtual CPU Count

To modify the number of CPUs assigned to a guest virtual machine, use the **virsh setvcpus** command:

> **# virsh setvcpus *{domain-name, domain-id or domain-uuid} count [[--config] [--live] | [--current] [--guest]***

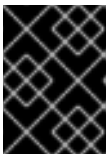
The following parameters may be set for the **virsh setvcpus** command:

- ***{domain-name, domain-id or domain-uuid}*** - Specifies the virtual machine.
- ***count*** - Specifies the number of virtual CPUs to set.

NOTE

The ***count*** value cannot exceed the number of CPUs that were assigned to the guest virtual machine when it was created. It may also be limited by the host or the hypervisor. For Xen, you can only adjust the virtual CPUs of a running domain if the domain is paravirtualized.

- ***--live*** - The default option, used if none are specified. The configuration change takes effect on the running guest virtual machine. This is referred to as a *hot plug* if the number of vCPUs is increased, and *hot unplug* if it is reduced.



IMPORTANT

The vCPU hot unplug feature is a Technology Preview. Therefore, it is not supported and not recommended for use in high-value deployments.

- ***--config*** - The configuration change takes effect on the next reboot of the guest. Both the **--config** and **--live** options may be specified together if supported by the hypervisor.
- ***--current*** - Configuration change takes effect on the current state of the guest virtual machine. If used on a running guest, it acts as ***--live***, if used on a shut-down guest, it acts as ***--config***.
- ***--maximum*** - Sets a maximum vCPU limit that can be hot-plugged on the next reboot of the guest. As such, it must only be used with the **--config** option, and not with the **--live** option.
- ***--guest*** - Instead of a hot plug or a hot unplug, the QEMU guest agent modifies the vCPU count directly in the running guest by enabling or disabling vCPUs. This option cannot be used with ***count*** value higher than the current number of vCPUs in the gueet, and configurations set with ***--guest*** are reset when a guest is rebooted.

> **Example 14.4. vCPU hot plug and hot unplug**

To hot-plug a vCPU, run the following command on a guest with a single vCPU:

```
virsh setvcpus guestVM1 2 --live
```

This increases the number of vCPUs for guestVM1 to two. The change is performed while guestVM1 is running, as indicated by the ***--live*** option.

To hot-unplug one vCPU from the same running guest, run the following:

```
virsh setvcpus guestVM1 1 --live
```

Be aware, however, that currently, using vCPU hot unplug can lead to problems with further modifications of the vCPU count.

14.13.7. Configuring Memory Allocation

To modify a guest virtual machine's memory allocation with **virsh**:

```
# virsh setmem {domain-id or domain-name} count
```

```
# virsh setmem vr-rhel6u1-x86_64-kvm --kilobytes 1025000
```

You must specify the **count** in kilobytes. The new count value cannot exceed the amount you specified when you created the guest virtual machine. Values lower than 64 MB are unlikely to work with most guest virtual machine operating systems. A higher maximum memory value does not affect active guest virtual machines. If the new value is lower than the available memory, it will shrink possibly causing the guest virtual machine to crash.

This command has the following options:

- [--domain] <string> domain name, id or uuid
- [--size] <number> new memory size, as scaled integer (default KiB)

  Valid memory units include:

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

- ***T*** or ***TiB*** for tebibytes (2^{40} or blocks of 1,099,511,627,776 bytes)

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

- --config takes affect next boot
- --live controls the memory of the running domain
- --current controls the memory on the current domain

14.13.8. Changing the Memory Allocation for the Domain

The **virsh setmaxmem *domain size* --config --live --current** allows the setting of the maximum memory allocation for a guest virtual machine as shown:

```
virsh setmaxmem rhel6 1024 --current
```

The size that can be given for the maximum memory is a scaled integer that by default is expressed in kibibytes, unless a supported suffix is provided. The following options can be used with this command:

- **--config** - takes affect next boot
- **--live** - controls the memory of the running domain, providing the hypervisor supports this action as not all hypervisors allow live changes of the maximum memory limit.
- **--current** - controls the memory on the current domain

14.13.9. Displaying Guest Virtual Machine Block Device Information

Use **virsh domblkstat** to display block device statistics for a running guest virtual machine.

```
# virsh domblkstat GuestName block-device
```

14.13.10. Displaying Guest Virtual Machine Network Device Information

Use **virsh domifstat** to display network interface statistics for a running guest virtual machine.

```
# virsh domifstat GuestName interface-device
```

14.14. MANAGING VIRTUAL NETWORKS

This section covers managing virtual networks with the **virsh** command. To list virtual networks:

```
# virsh net-list
```

This command generates output similar to:

```
# virsh net-list
```

```
Name              State      Autostart
-----------------------------------------
default            active     yes
vnet1             active     yes
vnet2             active     yes
```

To view network information for a specific virtual network:

```
# virsh net-dumpxml NetworkName
```

This displays information about a specified virtual network in XML format:

```
# virsh net-dumpxml vnet1
<network>
  <name>vnet1</name>
  <uuid>98361b46-1581-acb7-1643-85a412626e70</uuid>
  <forward dev='eth0'/>
  <bridge name='vnet0' stp='on' forwardDelay='0' />
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254' />
    </dhcp>
  </ip>
</network>
```

Other **virsh** commands used in managing virtual networks are:

- **virsh net-autostart *network-name*** – Autostart a network specified as *network-name*.
- **virsh net-create *XMLfile*** – generates and starts a new network using an existing XML file.
- **virsh net-define *XMLfile*** – generates a new network device from an existing XML file without starting it.
- **virsh net-destroy *network-name*** – destroy a network specified as *network-name*.
- **virsh net-name *networkUUID*** – convert a specified *networkUUID* to a network name.
- **virsh net-uuid *network-name*** – convert a specified *network-name* to a network UUID.
- **virsh net-start *nameOfInactiveNetwork*** – starts an inactive network.
- **virsh net-undefine *nameOfInactiveNetwork*** – removes the definition of an inactive network.

14.15. MIGRATING GUEST VIRTUAL MACHINES WITH VIRSH

Information on migration using virsh is located in the section entitled Live KVM Migration with virsh Refer to Section 4.4, “Live KVM Migration with virsh”

14.15.1. Interface Commands

The following commands manipulate host interfaces and as such should not be run from the guest virtual machine. These commands should be run from a terminal on the host physical machine.

> **WARNING**
>
> The commands in this section are only supported if the machine has the NetworkManager service disabled, and is using the **network** service instead.

Often, these host interfaces can then be used by name within domain **<interface>** elements (such as a system-created bridge interface), but there is no requirement that host interfaces be tied to any particular guest configuration XML at all. Many of the commands for host interfaces are similar to the ones used for domains, and the way to name an interface is either by its name or its MAC address. However, using a MAC address for an **iface** option only works when that address is unique (if an interface and a bridge share the same MAC address, which is often the case, then using that MAC address results in an error due to ambiguity, and you must resort to a name instead).

14.15.1.1. Defining and starting a host physical machine interface via an XML file

The **virsh iface-define *file*** command define a host interface from an XML file. This command will only define the interface and will not start it.

```
virsh iface-define iface.xml
```

To start an interface which has already been defined, run **iface-start *interface***, where *interface* is the interface name.

14.15.1.2. Editing the XML configuration file for the host interface

The command **iface-edit *interface*** edits the XML configuration file for a host interface. This is the **only** recommended way to edit the XML configuration file. (Refer to Chapter 20, *Manipulating the Domain XML* for more information about these files.)

14.15.1.3. Listing active host interfaces

The **iface-list --inactive --all** displays a list of active host interfaces. If **--all** is specified, this list will also include interfaces that are defined but are inactive. If **--inactive** is specified only the inactive interfaces will be listed.

14.15.1.4. Converting a MAC address into an interface name

The **iface-name *interface*** command converts a host interface MAC to an interface name, provided the MAC address is unique among the host's interfaces. This command requires *interface* which is the interface's MAC address.

The **iface-mac *interface*** command will convert a host's interface name to MAC address where in this case *interface*, is the interface name.

14.15.1.5. Stopping a specific host physical machine interface

The **virsh iface-destroy *interface*** command destroys (stops) a given host interface, which is the same as running **if-down** on the host. This command will disable that interface from active use and takes effect immediately.

To undefine the interface, use the **iface-undefine** ***interface*** command along with the interface name.

14.15.1.6. Displaying the host configuration file

virsh iface-dumpxml ***interface*** **--inactive** displays the host interface information as an XML dump to stdout. If the **--inactive** option is specified, then the output reflects the persistent state of the interface that will be used the next time it is started.

14.15.1.7. Creating bridge devices

The **iface-bridge** creates a bridge device named bridge, and attaches the existing network device interface to the new bridge, which starts working immediately, with STP enabled and a delay of 0.

```
# virsh iface-bridge interface bridge --no-stp delay --no-start
```

Note that these settings can be altered with --no-stp, --no-start, and an integer number of seconds for delay. All IP address configuration of interface will be moved to the new bridge device. Refer to Section 14.15.1.8, "Tearing down a bridge device" for information on tearing down the bridge.

14.15.1.8. Tearing down a bridge device

The **iface-unbridge** ***bridge*** **--no-start** command tears down a specified bridge device named *bridge*, releases its underlying interface back to normal usage, and moves all IP address configuration from the bridge device to the underlying device. The underlying interface is restarted unless **--no-start** option is used, but keep in mind not restarting is generally not recommended. Refer to Section 14.15.1.7, "Creating bridge devices" for the command to use to create a bridge.

14.15.1.9. Manipulating interface snapshots

The **iface-begin** command creates a snapshot of current host interface settings, which can later be committed (with **iface-commit**) or restored (**iface-rollback**). If a snapshot already exists, then this command will fail until the previous snapshot has been committed or restored. Undefined behavior will result if any external changes are made to host interfaces outside of the libvirt API between the time of the creation of a snapshot and its eventual commit or rollback.

Use the **iface-commit** command to declare all changes made since the last **iface-begin** as working, and then delete the rollback point. If no interface snapshot has already been started via **iface-begin**, then this command will fail.

Use the **iface-rollback** to revert all host interface settings back to the state that recorded the last time the **iface-begin** command was executed. If **iface-begin** command had not been previously executed, then **iface-rollback** will fail. Note that rebooting the host physical machine also serves as an implicit rollback point.

14.15.2. Managing Snapshots

The sections that follow describe actions that can be done in order to manipulate domain snapshots. *Snapshots* take the disk, memory, and device state of a domain at a specified point-in-time, and save it for future use. Snapshots have many uses, from saving a "clean" copy of an OS image to saving a domain's state before what may be a potentially destructive operation. Snapshots are identified with a unique name. See the libvirt website for documentation of the XML format used to represent properties of snapshots.

14.15.2.1. Creating Snapshots

The **virsh snapshot-create** command creates a snapshot for domain with the properties specified in the domain XML file (such as <name> and <description> elements, as well as <disks>).

To create a snapshot, run:

```
# snapshot-create <domain> <xmlfile> [--redefine] [--current] [--no-metadata] [--reuse-external]
```

The domain name, ID, or UID may be used as the domain requirement. The XML requirement is a string must contain the <name>, <description> and <disks> elements.

NOTE

Live snapshots are not supported in Red Hat Enterprise Linux. There are additional options available with the **virsh snapshot-create** command for use with live snapshots which are visible in libvirt, but not supported in Red Hat Enterprise Linux 6.

The options available in Red Hat Enterprise Linux include:

- **--redefine** specifies that if all XML elements produced by **snapshot-dumpxml** are valid; it can be used to migrate snapshot hierarchy from one machine to another, to recreate hierarchy for the case of a transient domain that goes away and is later recreated with the same name and UUID, or to make slight alterations in the snapshot metadata (such as host-specific aspects of the domain XML embedded in the snapshot). When this option is supplied, the **xmlfile** argument is mandatory, and the domain's current snapshot will not be altered unless the **--current** option is also given.

- **--no-metadata** creates the snapshot, but any metadata is immediately discarded (that is, libvirt does not treat the snapshot as current, and cannot revert to the snapshot unless **--redefine** is later used to teach libvirt about the metadata again).

- **--reuse-external**, if used, this option specifies the location of an existing external XML snapshot to use. If an existing external snapshot does not already exist, the command will fail to take a snapshot to avoid losing contents of the existing files.

14.15.2.2. Creating a snapshot for the current domain

The **virsh snapshot-create-as domain** command creates a snapshot for the domain with the properties specified in the domain XML file (such as <name> and <description> elements). If these values are not included in the XML string, libvirt will choose a value. To create a snapshot run:

```
# virsh snapshot-create-as domain {[--print-xml] | [--no-metadata] [--reuse-external]} [name]
[description] [--diskspec] diskspec]
```

The remaining options are as follows:

- **--print-xml** creates appropriate XML for **snapshot-create** as output, rather than actually creating a snapshot.

- **--diskspec** option can be used to control how **--disk-only** and external checkpoints create external files. This option can occur multiple times, according to the number of <disk> elements in the domain XML. Each <diskspec> is in the form disk**[,snapshot=type][,driver=type][,file=name]**. To include a literal comma in disk or in **file=name**, escape it with a second comma.

A literal **--diskspec** must precede each diskspec unless all three of <domain>, <name>, and <description> are also present. For example, a diskspec of **vda,snapshot=external,file=/path/to,,new** results in the following XML:

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new'/>
</disk>
```

- **--reuse-external** creates an external snapshot reusing an existing file as the destination (meaning this file is overwritten). If this destination does not exist, the snapshot request will be refused to avoid losing contents of the existing files.
- **--no-metadata** creates snapshot data but any metadata is immediately discarded (that is, libvirt does not treat the snapshot as current, and cannot revert to the snapshot unless snapshot-create is later used to teach libvirt about the metadata again). This option is incompatible with **--print-xml**.

14.15.2.3. Taking a snapshot of the current domain

This command is used to query which snapshot is currently in use. To use, run:

```
# virsh snapshot-current domain {[--name] | [--security-info] | [snapshotname]}
```

If **snapshotname** is not used, snapshot XML for the domain's current snapshot (if there is one) will be displayed as output. If **--name** is specified, just the current snapshot name instead of the full XML will be sent as output. If **--security-info** is supplied, security sensitive information will be included in the XML. Using **snapshotname**, libvirt generates a request to make the existing named snapshot become the current snapshot, without reverting it to the domain.

14.15.2.4. snapshot-edit-domain

This command is used to edit the snapshot that is currently in use. To use, run:

```
#virsh snapshot-edit domain [snapshotname] [--current] {[--rename] [--clone]}
```

If both **snapshotname** and **--current** are specified, it forces the edited snapshot to become the current snapshot. If **snapshotname** is omitted, then **--current** must be supplied, in order to edit the current snapshot.

This is equivalent to the following command sequence below, but it also includes some error checking:

```
# virsh snapshot-dumpxml dom name > snapshot.xml
# vi snapshot.xml [note - this can be any editor]
# virsh snapshot-create dom snapshot.xml --redefine [--current]
```

If **--rename** is specified, then the resulting edited file gets saved in a different file name. If **--clone** is specified, then changing the snapshot name will create a clone of the snapshot metadata. If neither is specified, then the edits will not change the snapshot name. Note that changing a snapshot name must be done with care, since the contents of some snapshots, such as internal snapshots within a single qcow2 file, are accessible only from the original snapshot filename.

14.15.2.5. snapshot-info-domain

snapshot-info-domain displays information about the snapshots. To use, run:

```
# snapshot-info domain {snapshot | --current}
```

Outputs basic information about a specified **snapshot** , or the current snapshot with **--current**.

14.15.2.6. snapshot-list-domain

List all of the available snapshots for the given domain, defaulting to show columns for the snapshot name, creation time, and domain state. To use, run:

```
#virsh snapshot-list domain [{--parent | --roots | --tree}] [{[--from] snapshot | --current} [--descendants]] [--metadata] [--no-metadata] [--leaves] [--no-leaves] [--inactive] [--active] [--internal] [--external]
```

The remaining optional options are as follows:

- **--parent** adds a column to the output table giving the name of the parent of each snapshot. This option may not be used with **--roots** or **--tree**.
- **--roots** filters the list to show only the snapshots that have no parents. This option may not be used with **--parent** or **--tree**.
- **--tree** displays output in a tree format, listing just snapshot names. These three options are mutually exclusive. This option may not be used with **--roots** or **--parent**.
- **--from** filters the list to snapshots which are children of the given snapshot; or if **--current** is provided, will cause the list to start at the current snapshot. When used in isolation or with **--parent**, the list is limited to direct children unless **--descendants** is also present. When used with **--tree**, the use of **--descendants** is implied. This option is not compatible with **--roots**. Note that the starting point of **--from** or **--current** is not included in the list unless the **--tree** option is also present.
- **--leaves** is specified, the list will be filtered to just snapshots that have no children. Likewise, if **--no-leaves** is specified, the list will be filtered to just snapshots with children. (Note that omitting both options does no filtering, while providing both options will either produce the same list or error out depending on whether the server recognizes the options) Filtering options are not compatible with **--tree**..
- **--metadata** is specified, the list will be filtered to just snapshots that involve libvirt metadata, and thus would prevent the undefining of a persistent domain, or be lost on destroy of a transient domain. Likewise, if **--no-metadata** is specified, the list will be filtered to just snapshots that exist without the need for libvirt metadata.
- **--inactive** is specified, the list will be filtered to snapshots that were taken when the domain was shut off. If **--active** is specified, the list will be filtered to snapshots that were taken when the domain was running, and where the snapshot includes the memory state to revert to that running state. If **--disk-only** is specified, the list will be filtered to snapshots that were taken when the domain was running, but where the snapshot includes only disk state.
- **--internal** is specified, the list will be filtered to snapshots that use internal storage of existing disk images. If --external is specified, the list will be filtered to snapshots that use external files for disk images or memory state.

14.15.2.7. snapshot-dumpxml domain snapshot

virsh snapshot-dumpxml domain snapshot outputs the snapshot XML for the domain's snapshot named snapshot. To use, run:

```
# virsh snapshot-dumpxml domain snapshot [--security-info]
```

The **--security-info** option will also include security sensitive information. Use **snapshot-current** to easily access the XML of the current snapshot.

14.15.2.8. snapshot-parent domain

Outputs the name of the parent snapshot, if any, for the given snapshot, or for the current snapshot with **--current**. To use, run:

```
#virsh snapshot-parent domain {snapshot | --current}
```

14.15.2.9. snapshot-revert domain

Reverts the given domain to the snapshot specified by **snapshot**, or to the current snapshot with **--current**.

> **WARNING**
>
> Be aware that this is a destructive action; any changes in the domain since the last snapshot was taken will be lost. Also note that the state of the domain after **snapshot-revert** is complete will be the state of the domain at the time the original snapshot was taken.

To revert the snapshot, run

```
# snapshot-revert domain {snapshot | --current} [{--running | --paused}] [--force]
```

Normally, reverting to a snapshot leaves the domain in the state it was at the time the snapshot was created, except that a disk snapshot with no guest virtual machine state leaves the domain in an inactive state. Passing either the **--running** or **--paused** option will perform additional state changes (such as booting an inactive domain, or pausing a running domain). Since transient domains cannot be inactive, it is required to use one of these options when reverting to a disk snapshot of a transient domain.

There are two cases where a **snapshot revert** involves extra risk, which requires the use of **--force** to proceed. One is the case of a snapshot that lacks full domain information for reverting configuration; since libvirt cannot prove that the current configuration matches what was in use at the time of the snapshot, supplying **--force** assures libvirt that the snapshot is compatible with the current configuration (and if it is not, the domain will likely fail to run). The other is the case of reverting from a running domain to an active state where a new hypervisor has to be created rather than reusing the existing hypervisor, because it implies drawbacks such as breaking any existing VNC or Spice connections; this condition happens with an active snapshot that uses a provably incompatible configuration, as well as with an inactive snapshot that is combined with the **--start** or **--pause** option.

14.15.2.10. snapshot-delete domain

snapshot-delete domain deletes the snapshot for the specified domain. To do this, run:

```
# virsh snapshot-delete domain {snapshot | --current} [--metadata] [{--children | --children-only}]
```

This command Deletes the snapshot for the domain named **snapshot**, or the current snapshot with **--current**. If this snapshot has child snapshots, changes from this snapshot will be merged into the children. If the option **--children** is used, then it will delete this snapshot and any children of this snapshot. If **--children-only** is used, then it will delete any children of this snapshot, but leave this snapshot intact. These two options are mutually exclusive.

The **--metadata** is used it will delete the snapshot's metadata maintained by libvirt, while leaving the snapshot contents intact for access by external tools; otherwise deleting a snapshot also removes its data contents from that point in time.

14.16. GUEST VIRTUAL MACHINE CPU MODEL CONFIGURATION

This section provides information about guest virtual machine CPU model configuration.

14.16.1. Introduction

Every hypervisor has its own policy for what a guest virtual machine will see for its CPUs by default. Whereas some hypervisors decide which CPU host physical machine features will be available for the guest virtual machine, QEMU/KVM presents the guest virtual machine with a generic model named **qemu32** or **qemu64**. These hypervisors perform more advanced filtering, classifying all physical CPUs into a handful of groups and have one baseline CPU model for each group that is presented to the guest virtual machine. Such behavior enables the safe migration of guest virtual machines between host physical machines, provided they all have physical CPUs that classify into the same group. libvirt does not typically enforce policy itself, rather it provides the mechanism on which the higher layers define their own desired policy. Understanding how to obtain CPU model information and define a suitable guest virtual machine CPU model is critical to ensure guest virtual machine migration is successful between host physical machines. Note that a hypervisor can only emulate features that it is aware of and features that were created after the hypervisor was released may not be emulated.

14.16.2. Learning about the Host Physical Machine CPU Model

The **virsh capabilities** command displays an XML document describing the capabilities of the hypervisor connection and host physical machine. The XML schema displayed has been extended to provide information about the host physical machine CPU model. One of the big challenges in describing a CPU model is that every architecture has a different approach to exposing their capabilities. On x86, the capabilities of a modern CPU are exposed via the CPUID instruction. Essentially this comes down to a set of 32-bit integers with each bit given a specific meaning. Fortunately AMD and Intel agree on common semantics for these bits. Other hypervisors expose the notion of CPUID masks directly in their guest virtual machine configuration format. However, QEMU/KVM supports far more than just the x86 architecture, so CPUID is clearly not suitable as the canonical configuration format. QEMU ended up using a scheme which combines a CPU model name string, with a set of named options. On x86, the CPU model maps to a baseline CPUID mask, and the options can be used to then toggle bits in the mask on or off. libvirt decided to follow this lead and uses a combination of a model name and options.

It is not practical to have a database listing all known CPU models, so libvirt has a small list of baseline CPU model names. It chooses the one that shares the greatest number of CPUID bits with the actual host physical machine CPU and then lists the remaining bits as named features. Notice that libvirt does not display which features the baseline CPU contains. This might seem like a flaw at first, but as will be explained in this section, it is not actually necessary to know this information.

14.16.3. Determining a Compatible CPU Model to Suit a Pool of Host Physical Machines

Now that it is possible to find out what CPU capabilities a single host physical machine has, the next step is to determine what CPU capabilities are best to expose to the guest virtual machine. If it is known that the guest virtual machine will never need to be migrated to another host physical machine, the host physical machine CPU model can be passed straight through unmodified. A virtualized data center may have a set of configurations that can guarantee all servers will have 100% identical CPUs. Again the host physical machine CPU model can be passed straight through unmodified. The more common case, though, is where there is variation in CPUs between host physical machines. In this mixed CPU environment, the lowest common denominator CPU must be determined. This is not entirely straightforward, so libvirt provides an API for exactly this task. If libvirt is provided a list of XML documents, each describing a CPU model for a host physical machine, libvirt will internally convert these to CPUID masks, calculate their intersection, and convert the CPUID mask result back into an XML CPU description.

Here is an example of what libvirt reports as the capabilities on a basic workstation, when the **virsh capabilities**is executed:

```
<capabilities>
  <host>
    <cpu>
      <arch>i686</arch>
      <model>pentium3</model>
      <topology sockets='1' cores='2' threads='1'/>
      <feature name='lahf_lm'/>
      <feature name='lm'/>
      <feature name='xtpr'/>
      <feature name='cx16'/>
      <feature name='ssse3'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='monitor'/>
      <feature name='pni'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='sse2'/>
      <feature name='acpi'/>
      <feature name='ds'/>
      <feature name='clflush'/>
      <feature name='apic'/>
    </cpu>
  </host>
</capabilities>
```

Figure 14.3. Pulling host physical machine's CPU model information

Now compare that to any random server, with the same **virsh capabilities** command:

```
<capabilities>
  <host>
    <cpu>
      <arch>x86_64</arch>
      <model>phenom</model>
      <topology sockets='2' cores='4' threads='1'/>
      <feature name='osvw'/>
      <feature name='3dnowprefetch'/>
      <feature name='misalignsse'/>
      <feature name='sse4a'/>
      <feature name='abm'/>
      <feature name='cr8legacy'/>
      <feature name='extapic'/>
      <feature name='cmp_legacy'/>
      <feature name='lahf_lm'/>
      <feature name='rdtscp'/>
      <feature name='pdpe1gb'/>
      <feature name='popcnt'/>
      <feature name='cx16'/>
      <feature name='ht'/>
      <feature name='vme'/>
    </cpu>
    ...snip...
```

Figure 14.4. Generate CPU description from a random server

To see if this CPU description is compatible with the previous workstation CPU description, use the **virsh cpu-compare** command.

The reduced content was stored in a file named **virsh-caps-workstation-cpu-only.xml** and the **virsh cpu-compare** command can be executed on this file:

```
# virsh cpu-compare virsh-caps-workstation-cpu-only.xml
Host physical machine CPU is a superset of CPU described in virsh-caps-workstation-cpu-only.xml
```

As seen in this output, libvirt is correctly reporting that the CPUs are not strictly compatible. This is because there are several features in the server CPU that are missing in the client CPU. To be able to migrate between the client and the server, it will be necessary to open the XML file and comment out some features. To determine which features need to be removed, run the **virsh cpu-baseline** command, on the **both-cpus.xml** which contains the CPU information for both machines. Running **# virsh cpu-baseline both-cpus.xml**, results in:

```
<cpu match='exact'>
  <model>pentium3</model>
  <feature policy='require' name='lahf_lm'/>
  <feature policy='require' name='lm'/>
  <feature policy='require' name='cx16'/>
  <feature policy='require' name='monitor'/>
  <feature policy='require' name='pni'/>
  <feature policy='require' name='ht'/>
  <feature policy='require' name='sse2'/>
  <feature policy='require' name='clflush'/>
  <feature policy='require' name='apic'/>
</cpu>
```

Figure 14.5. Composite CPU baseline

This composite file shows which elements are in common. Everything that is not in common should be commented out.

14.17. CONFIGURING THE GUEST VIRTUAL MACHINE CPU MODEL

For simple defaults, the guest virtual machine CPU configuration accepts the same basic XML representation as the host physical machine capabilities XML exposes. In other words, the XML from the **cpu-baseline** virsh command can now be copied directly into the guest virtual machine XML at the top level under the <domain> element. In the previous XML snippet, there are a few extra attributes available when describing a CPU in the guest virtual machine XML. These can mostly be ignored, but for the curious here is a quick description of what they do. The top level <cpu> element has an attribute called **match** with possible values of:

- match='minimum' - the host physical machine CPU must have at least the CPU features described in the guest virtual machine XML. If the host physical machine has additional features beyond the guest virtual machine configuration, these will also be exposed to the guest virtual machine.
- match='exact' - the host physical machine CPU must have at least the CPU features described in the guest virtual machine XML. If the host physical machine has additional features beyond the guest virtual machine configuration, these will be masked out from the guest virtual machine.
- match='strict' - the host physical machine CPU must have exactly the same CPU features described in the guest virtual machine XML.

The next enhancement is that the <feature> elements can each have an extra 'policy' attribute with possible values of:

- policy='force' - expose the feature to the guest virtual machine even if the host physical machine does not have it. This is usually only useful in the case of software emulation.
- policy='require' - expose the feature to the guest virtual machine and fail if the host physical machine does not have it. This is the sensible default.
- policy='optional' - expose the feature to the guest virtual machine if it happens to support it.
- policy='disable' - if the host physical machine has this feature, then hide it from the guest virtual machine.

- policy='forbid' - if the host physical machine has this feature, then fail and refuse to start the guest virtual machine.

The 'forbid' policy is for a niche scenario where an incorrectly functioning application will try to use a feature even if it is not in the CPUID mask, and you wish to prevent accidentally running the guest virtual machine on a host physical machine with that feature. The 'optional' policy has special behavior with respect to migration. When the guest virtual machine is initially started the parameter is optional, but when the guest virtual machine is live migrated, this policy turns into 'require', since you cannot have features disappearing across migration.

14.18. MANAGING RESOURCES FOR GUEST VIRTUAL MACHINES

virsh allows the grouping and allocation of resources on a per guest virtual machine basis. This is managed by the libvirt daemon, which creates *cgroups* and manages them on behalf of the guest virtual machine. The only thing that is left for the system administrator to do is to either query or set tunables against specified guest virtual machines. The following tunables may used:

- **memory** - The memory controller allows for setting limits on RAM and swap usage and querying cumulative usage of all processes in the group
- **cpuset** - The CPU set controller binds processes within a group to a set of CPUs and controls migration between CPUs.
- **cpuacct** - The CPU accounting controller provides information about CPU usage for a group of processes.
- **cpu** -The CPU scheduler controller controls the prioritization of processes in the group. This is similar to granting **nice** level privileges.
- **devices** - The devices controller grants access control lists on character and block devices.
- **freezer** - The freezer controller pauses and resumes execution of processes in the group. This is similar to **SIGSTOP** for the whole group.
- **net_cls** - The network class controller manages network utilization by associating processes with a **tc** network class.

In creating a group hierarchy cgroup will leave mount point and directory setup entirely to the administrators' discretion and is more complex than just adding some mount points to **/etc/fstab**. It is necessary to setup the directory hierarchy and decide how processes get placed within it. This can be done with the following virsh commands:

- **schedinfo** - described in Section 14.19, "Setting Schedule Parameters"
- **blkiotune**- described in Section 14.20, "Display or Set Block I/O Parameters"
- **domiftune**- described in Section 14.5.9, "Setting Network Interface Bandwidth Parameters"
- **memtune** - described in Section 14.21, "Configuring Memory Tuning"

14.19. SETTING SCHEDULE PARAMETERS

schedinfo allows scheduler parameters to be passed to guest virtual machines. The following command format should be used:

```
#virsh schedinfo domain --set --weight --cap --current --config --live
```

Each parameter is explained below:

- **domain** - this is the guest virtual machine domain
- **--set** - the string placed here is the controller or action that is to be called. Additional parameters or values if required should be added as well.
- **--current** - when used with **--set**, will use the specified **set** string as the current scheduler information. When used without will display the current scheduler information.
- **--config** - - when used with **--set**, will use the specified **set** string on the next reboot. When used without will display the scheduler information that is saved in the configuration file.
- **--live** - when used with **--set**, will use the specified **set** string on a guest virtual machine that is currently running. When used without will display the configuration setting currently used by the running virtual machine

The scheduler can be set with any of the following parameters: ***cpu_shares***, ***vcpu_period*** and ***vcpu_quota***.

Example 14.5. schedinfo show

This example shows the shell guest virtual machine's schedule information

```
# virsh schedinfo shell
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
```

Example 14.6. schedinfo set

In this example, the cpu_shares is changed to 2046. This effects the current state and not the configuration file.

```
# virsh schedinfo --set cpu_shares=2046 shell
Scheduler      : posix
cpu_shares     : 2046
vcpu_period    : 100000
vcpu_quota     : -1
```

14.20. DISPLAY OR SET BLOCK I/O PARAMETERS

blkiotune sets and or displays the I/O parameters for a specified guest virtual machine. The following format should be used:

```
# virsh blkiotune domain [--weight weight] [--device-weights device-weights] [[--config] [--live] | [--current]]
```

More information on this command can be found in the *Virtualization Tuning and Optimization Guide*

14.21. CONFIGURING MEMORY TUNING

The **virsh memtune virtual_machine --parameter size** is covered in the *Virtualization Tuning and Optimization Guide*.

14.22. VIRTUAL NETWORKING COMMANDS

The following commands manipulate virtual networks. libvirt has the capability to define virtual networks which can then be used by domains and linked to actual network devices. For more detailed information about this feature see the documentation at libvirt's website . Many of the commands for virtual networks are similar to the ones used for domains, but the way to name a virtual network is either by its name or UUID.

14.22.1. Autostarting a Virtual Network

This command will configure a virtual network to be started automatically when the guest virtual machine boots. To run this command:

```
# virsh net-autostart network [--disable]
```

This command accepts the **--disable** option which disables the autostart command.

14.22.2. Creating a Virtual Network from an XML File

This command creates a virtual network from an XML file. Refer to libvirt's website to get a description of the XML network format used by libvirt. In this command *file* is the path to the XML file. To create the virtual network from an XML file, run:

```
# virsh net-create file
```

14.22.3. Defining a Virtual Network from an XML File

This command defines a virtual network from an XML file, the network is just defined but not instantiated. To define the virtual network, run:

```
# net-define file
```

14.22.4. Stopping a Virtual Network

This command destroys (stops) a given virtual network specified by its name or UUID. This takes effect immediately. To stop the specified network *network* is required.

```
# net-destroy network
```

14.22.5. Creating a Dump File

This command outputs the virtual network information as an XML dump to stdout for the specified virtual network. If **--inactive** is specified, then physical functions are not expanded into their associated virtual functions. To create the dump file, run:

```
# virsh net-dumpxml network [--inactive]
```

14.22.6. Editing a Virtual Network's XML Configuration File

The following command edits the XML configuration file for a network:

```
# virsh net-edit network
```

The editor used for editing the XML file can be supplied by the $VISUAL or $EDITOR environment variables, and defaults to **vi**.

14.22.7. Getting Information about a Virtual Network

This command returns basic information about the *network* object. To get the network information, run:

```
# virsh net-info network
```

14.22.8. Listing Information about a Virtual Network

Returns the list of active networks, if **--all** is specified this will also include defined but inactive networks, if **--inactive** is specified only the inactive ones will be listed. You may also want to filter the returned networks by **--persistent** to list the persitent ones, **--transient** to list the transient ones, **--autostart** to list the ones with autostart enabled, and **--no-autostart** to list the ones with autostart disabled.

Note: When talking to older servers, this command is forced to use a series of API calls with an inherent race, where a pool might not be listed or might appear more than once if it changed state between calls while the list was being collected. Newer servers do not have this problem.

To list the virtual networks, run:

```
# net-list [--inactive | --all] [--persistent] [<--transient>] [--autostart] [<--no-autostart>]
```

14.22.9. Converting a Network UUID to Network Name

This command converts a network UUID to network name. To do this run:

```
# virsh net-name network-UUID
```

14.22.10. Starting a (Previously Defined) Inactive Network

This command starts a (previously defined) inactive network. To do this, run:

```
# virsh net-start network
```

14.22.11. Undefining the Configuration for an Inactive Network

This command undefines the configuration for an inactive network. To do this, run:

```
# net-undefine network
```

14.22.12. Converting a Network Name to Network UUID

This command converts a network name to network UUID. To do this, run:

```
# virsh net-uuid network-name
```

14.22.13. Updating an Existing Network Definition File

This command updates the given section of an existing network definition, taking effect immediately, without needing to destroy and re-start the network. This command is one of "add-first", "add-last", "add" (a synonym for add-last), "delete", or "modify". section is one of ""bridge", "domain", "ip", "ip-dhcp-host", "ip-dhcp-range", "forward", "forward-interface", "forward-pf", "portgroup", "dns-host", "dns-txt", or "dns-srv", each section being named by a concatenation of the xml element hierarchy leading to the element being changed. For example, "ip-dhcp-host" will change a **<host>** element that is contained inside a **<dhcp>** element inside an **<ip>** element of the network. xml is either the text of a complete xml element of the type being changed (for e "<host mac="00:11:22:33:44:55' ip='192.0.2.1'/>", or the name of a file that contains a complete xml element. Disambiguation is done by looking at the first character of the provided text - if the first character is "<", it is xml text, if the first character is not ">", it is the name of a file that contains the xml text to be used. The --parent-index option is used to specify which of several parent elements the requested element is in (0-based). For example, a dhcp **<host>** element could be in any one of multiple **<ip>** elements in the network; if a parent-index isn't provided, the "most appropriate" **<ip>** element will be selected (usually the only one that already has a **<dhcp>** element), but if ***--parent***-index is given, that particular instance of **<ip>** will get the modification. If ***--live*** is specified, affect a running network. If ***--config*** is specified, affect the next startup of a persistent network. If ***-- current*** is specified, affect the current network state. Both ***--live*** and ***--config*** options may be given, but --current is exclusive. Not specifying any option is the same as specifying ***--current***.

To update the configuration file, run:

```
# virsh net-update network command section xml [--parent-index index] [[--live] [--config] | [--current]]
```

CHAPTER 15. MANAGING GUESTS WITH THE VIRTUAL MACHINE MANAGER (VIRT-MANAGER)

This section describes the Virtual Machine Manager (**virt-manager**) windows, dialog boxes, and various GUI controls.

virt-manager provides a graphical view of hypervisors and guests on your host system and on remote host systems. **virt-manager** can perform virtualization management tasks, including:

- defining and creating guests,
- assigning memory,
- assigning virtual CPUs,
- monitoring operational performance,
- saving and restoring, pausing and resuming, and shutting down and starting guests,
- links to the textual and graphical consoles, and
- live and offline migrations.

15.1. STARTING VIRT-MANAGER

To start **virt-manager** session open the **Applications** menu, then the **System Tools** menu and select **Virtual Machine Manager** (**virt-manager**).

The **virt-manager** main window appears.

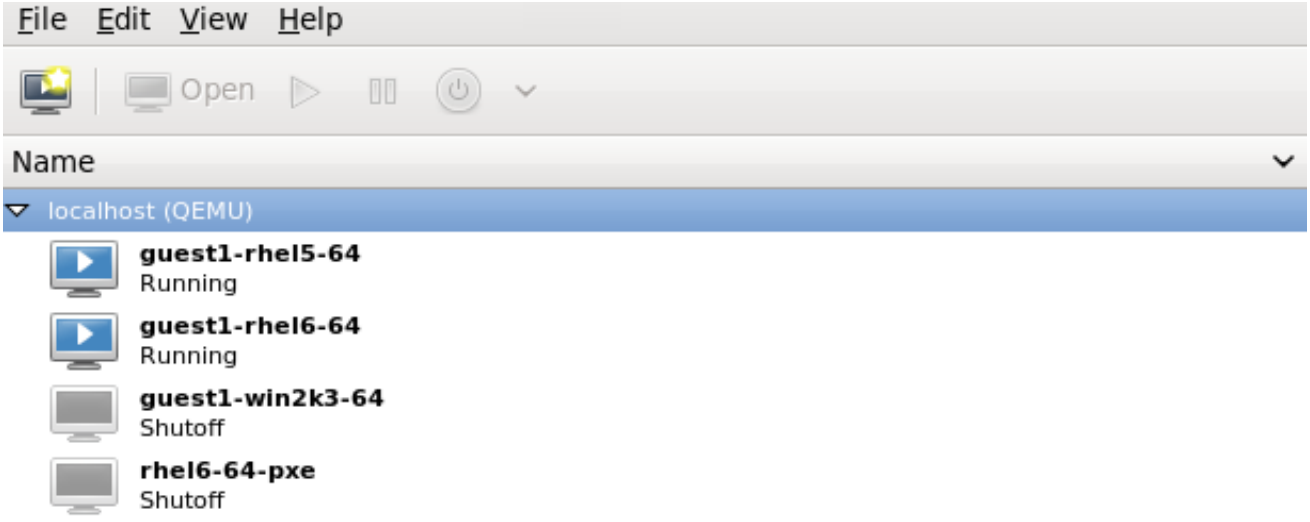


Figure 15.1. Starting virt-manager

Alternatively, **virt-manager** can be started remotely using ssh as demonstrated in the following command:

```
ssh -X host's address
[remotehost]# virt-manager
```

Using **ssh** to manage virtual machines and hosts is discussed further in Section 5.1, “Remote Management with SSH”.

15.2. THE VIRTUAL MACHINE MANAGER MAIN WINDOW

This main window displays all the running guests and resources used by guests. Select a guest by double clicking the guest's name.

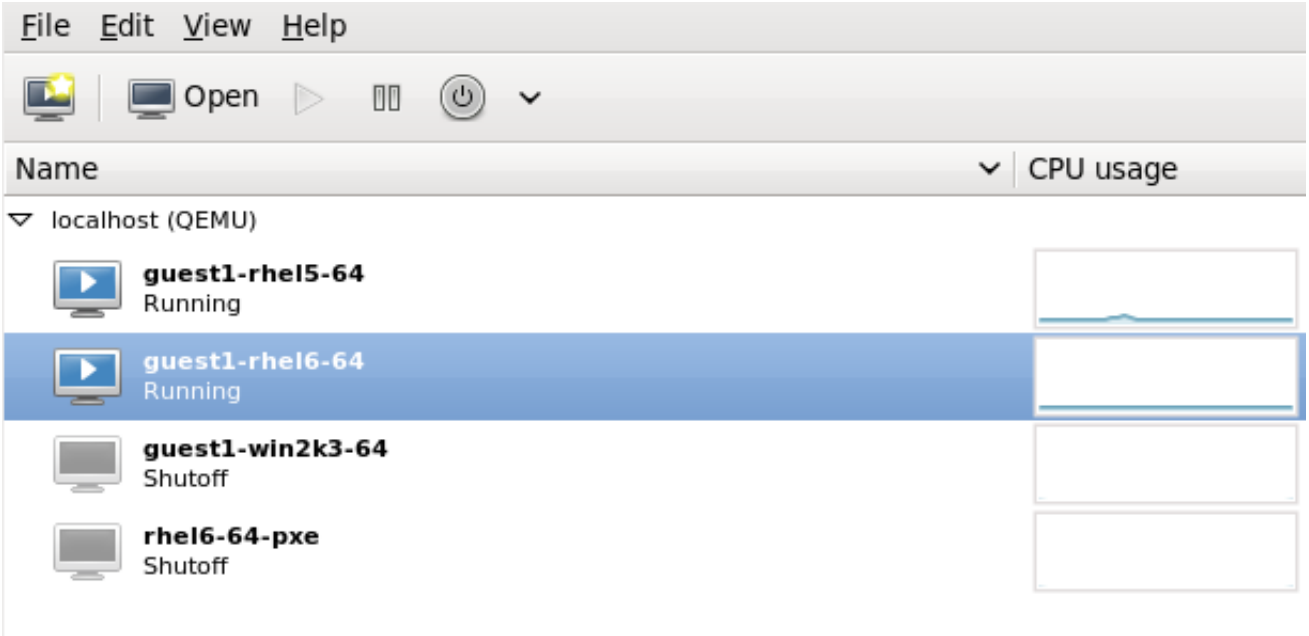


Figure 15.2. Virtual Machine Manager main window

15.3. THE VIRTUAL HARDWARE DETAILS WINDOW

The virtual hardware details window displays information about the virtual hardware configured for the guest. Virtual hardware resources can be added, removed and modified in this window. To access the virtual hardware details window, click on the icon in the toolbar.

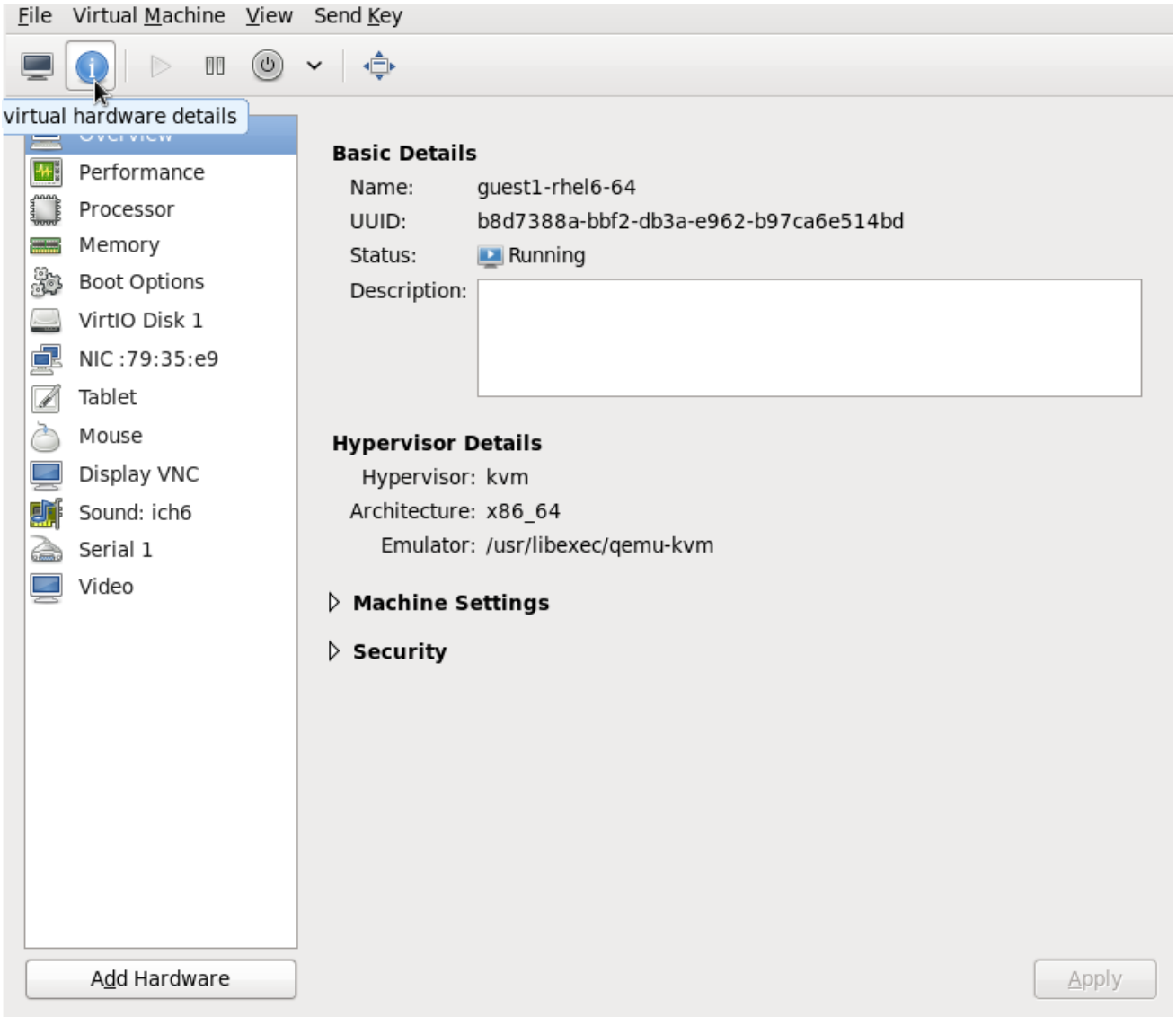


Figure 15.3. The virtual hardware details icon

Clicking the icon displays the virtual hardware details window.

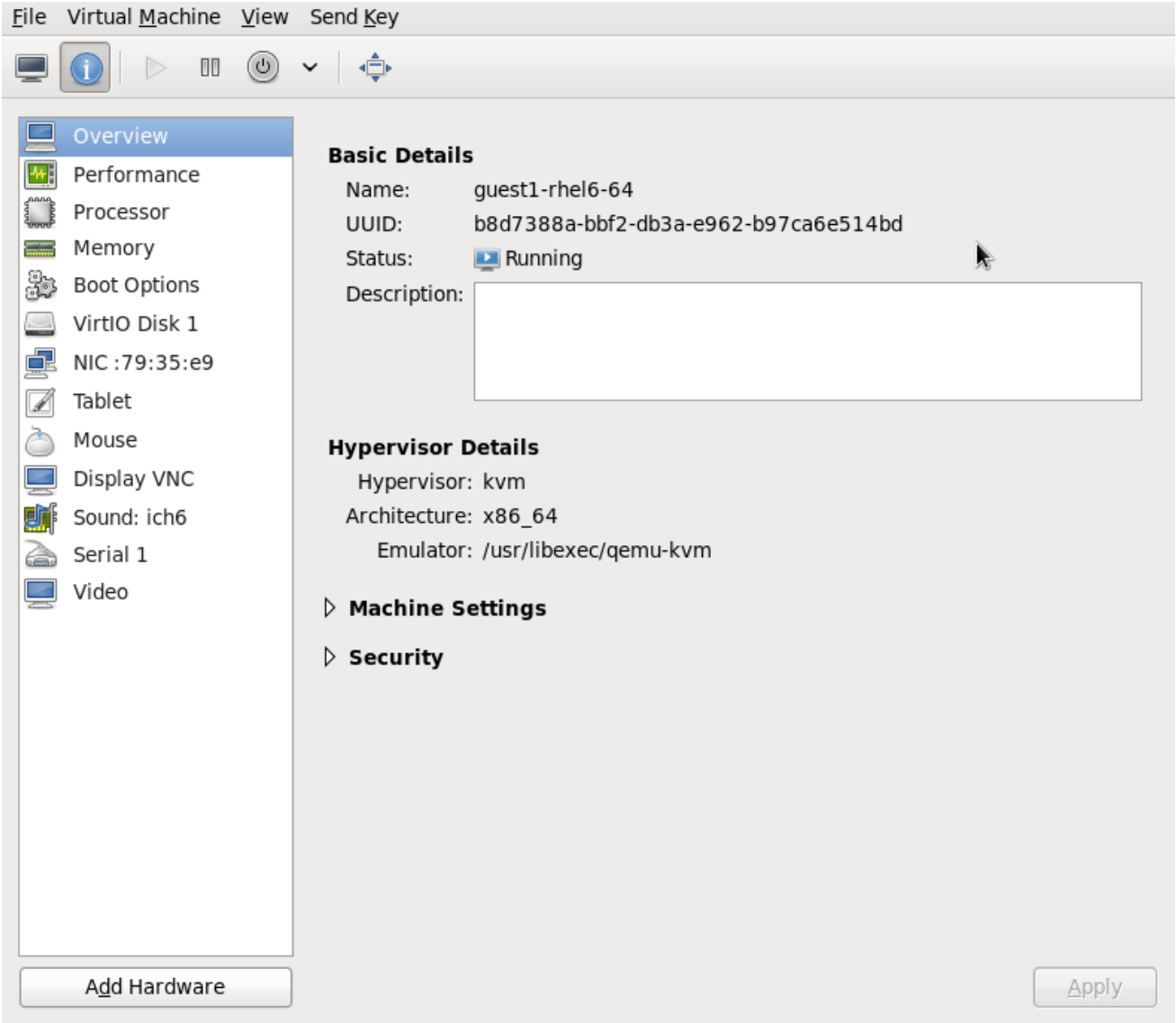


Figure 15.4. The virtual hardware details window

15.3.1. Attaching USB Devices to a Guest Virtual Machine

NOTE

In order to attach the USB device to the guest virtual machine, you first must attach it to the host physical machine and confirm that the device is working. If the guest is running, you need to shut it down before proceeding.

Procedure 15.1. Attaching USB Devices using Virt-Manager

1. Open the guest virtual machine's Virtual Machine Details screen.
2. Click **Add Hardware**

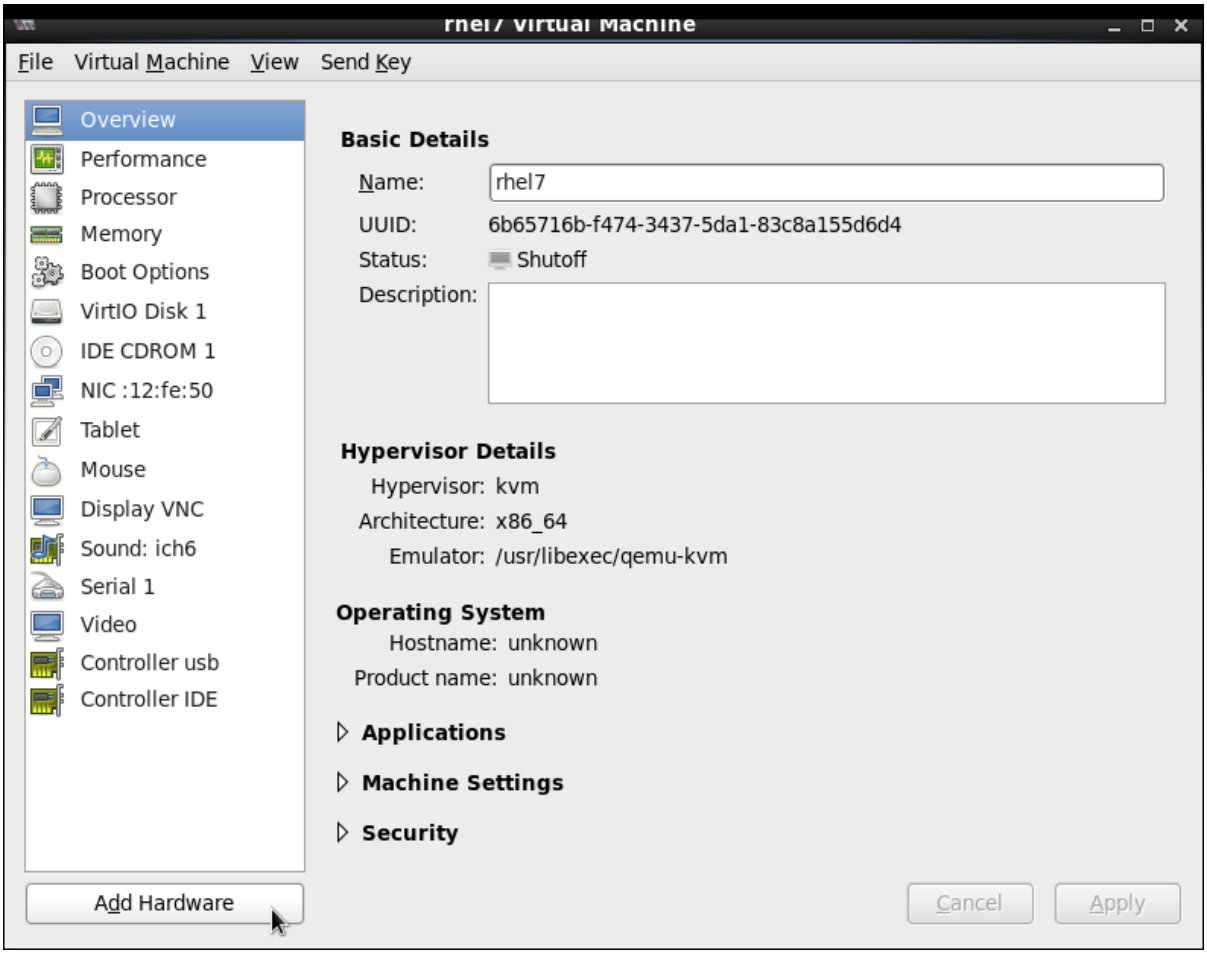


Figure 15.5. Add Hardware Button

3. In the **Add New Virtual Hardware** popup, select **USB Host Device**, select the device you want to attach from the list and Click **Finish**.

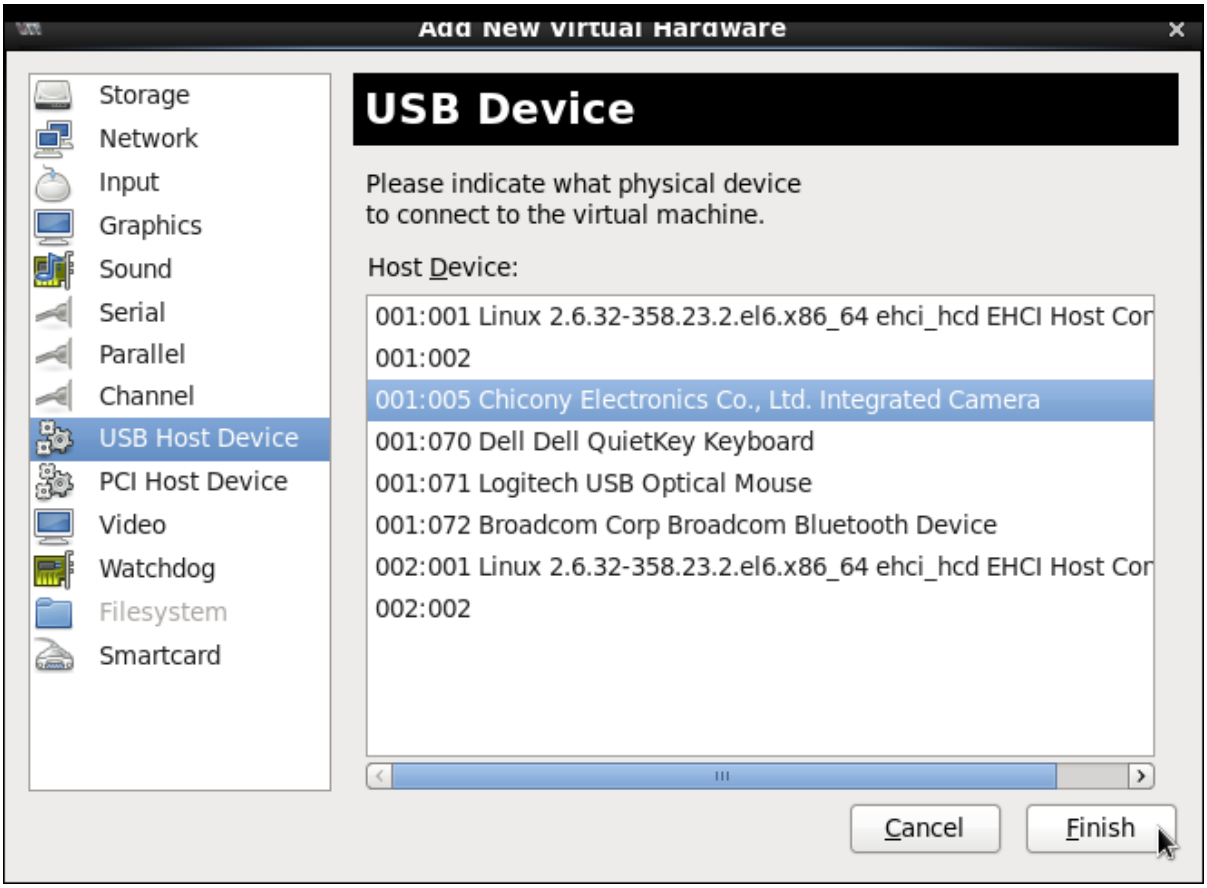


Figure 15.6. Add USB Device

4. To use the USB device in the guest virtual machine, start the guest virtual machine.

15.4. VIRTUAL MACHINE GRAPHICAL CONSOLE

This window displays a guest's graphical console. Guests can use several different protocols to export their graphical framebuffers: **virt-manager** supports **VNC** and **SPICE**. If your virtual machine is set to require authentication, the Virtual Machine graphical console prompts you for a password before the display appears.

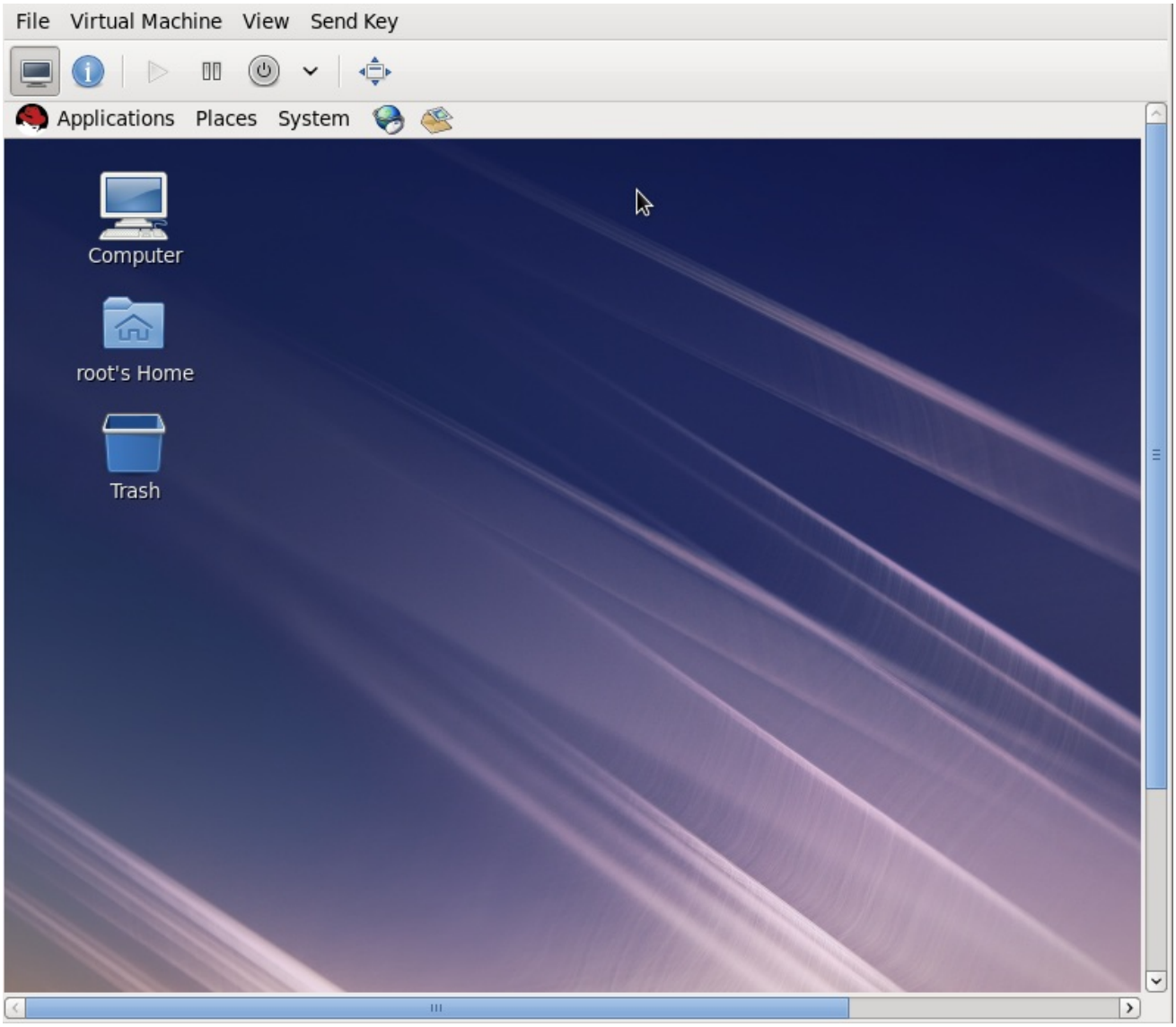


Figure 15.7. Graphical console window

NOTE

VNC is considered insecure by many security experts, however, several changes have been made to enable the secure usage of VNC for virtualization on Red Hat enterprise Linux. The guest machines only listen to the local host's loopback address (**127.0.0.1**). This ensures only those with shell privileges on the host can access virt-manager and the virtual machine through VNC. Although virt-manager is configured to listen to other public network interfaces and alternative methods can be configured, it is not recommended.

Remote administration can be performed by tunneling over SSH which encrypts the traffic. Although VNC can be configured to access remotely without tunneling over SSH, for security reasons, it is not recommended. To remotely administer the guest follow the instructions in: Chapter 5, *Remote Management of Guests*. TLS can provide enterprise level security for managing guest and host systems.

Your local desktop can intercept key combinations (for example, Ctrl+Alt+F1) to prevent them from being sent to the guest machine. You can use the **Send key** menu option to send these sequences. From the guest machine window, click the **Send key** menu and select the key sequence to send. In addition, from this menu you can also capture the screen output.

SPICE is an alternative to VNC available for Red Hat Enterprise Linux.

15.5. ADDING A REMOTE CONNECTION

This procedure covers how to set up a connection to a remote system using **virt-manager**.

1. To create a new connection open the **File** menu and select the **Add Connection...** menu item.

2. The **Add Connection** wizard appears. Select the hypervisor. For Red Hat Enterprise Linux 6 systems select **QEMU/KVM**. Select Local for the local system or one of the remote connection options and click **Connect**. This example uses Remote tunnel over SSH which works on default installations. For more information on configuring remote connections refer to Chapter 5, *Remote Management of Guests*

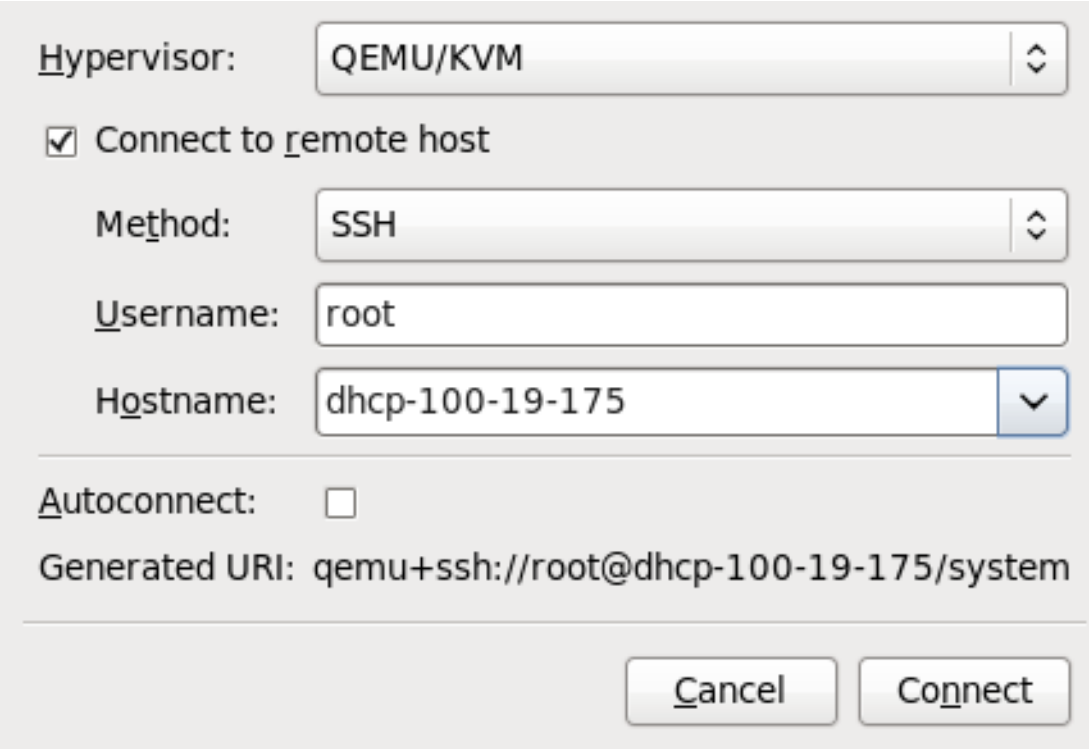


Figure 15.8. Add Connection

3. Enter the root password for the selected host when prompted.

A remote host is now connected and appears in the main **virt-manager** window.

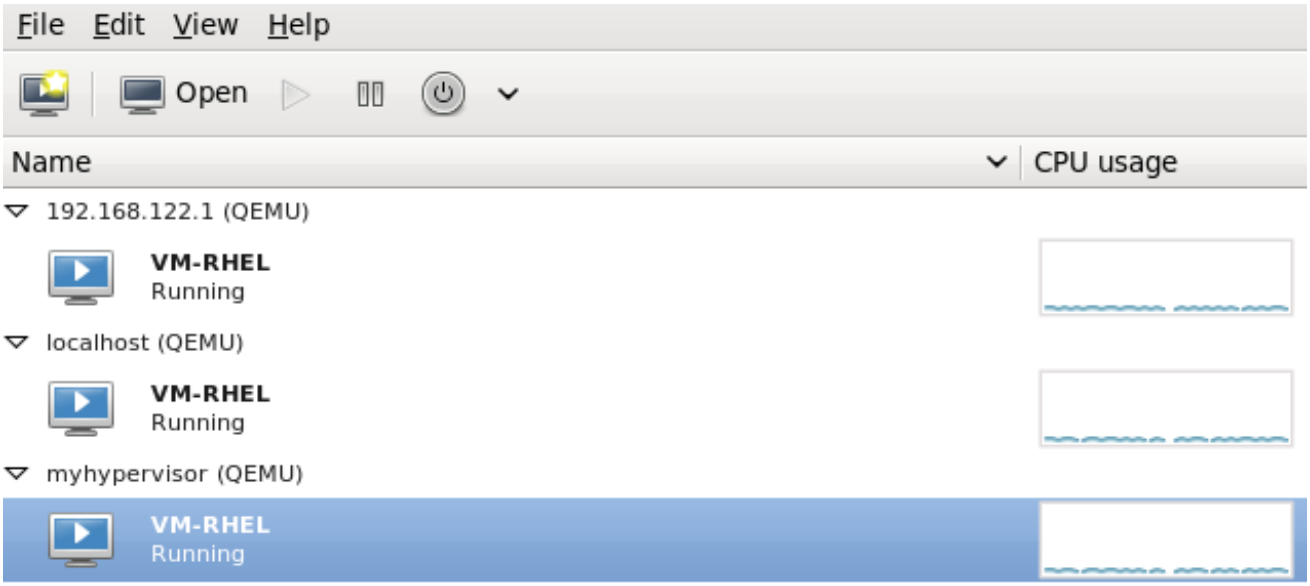


Figure 15.9. Remote host in the main virt-manager window

15.6. DISPLAYING GUEST DETAILS

You can use the Virtual Machine Monitor to view activity information for any virtual machines on your system.

To view a virtual system's details:

1. In the Virtual Machine Manager main window, highlight the virtual machine that you want to view.

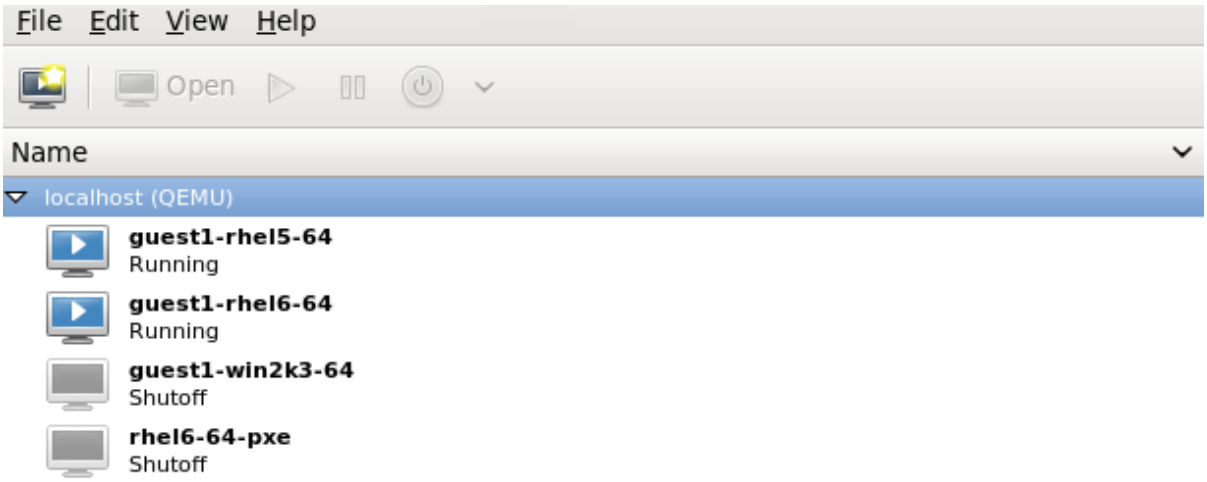


Figure 15.10. Selecting a virtual machine to display

2. From the Virtual Machine Manager **Edit** menu, select **Virtual Machine Details**.

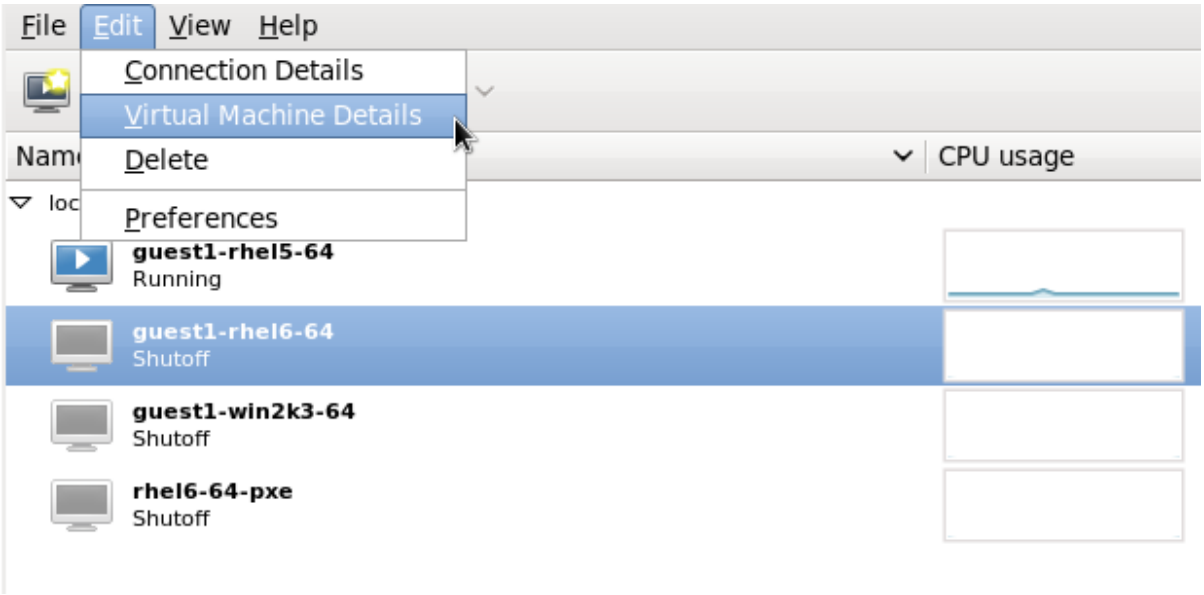


Figure 15.11. Displaying the virtual machine details

When the Virtual Machine details window opens, there may be a console displayed. Should this happen, click **View** and then select **Details**. The Overview window opens first by default. To go back to this window, select **Overview** from the navigation pane on the left hand side.

The **Overview** view shows a summary of configuration details for the guest.

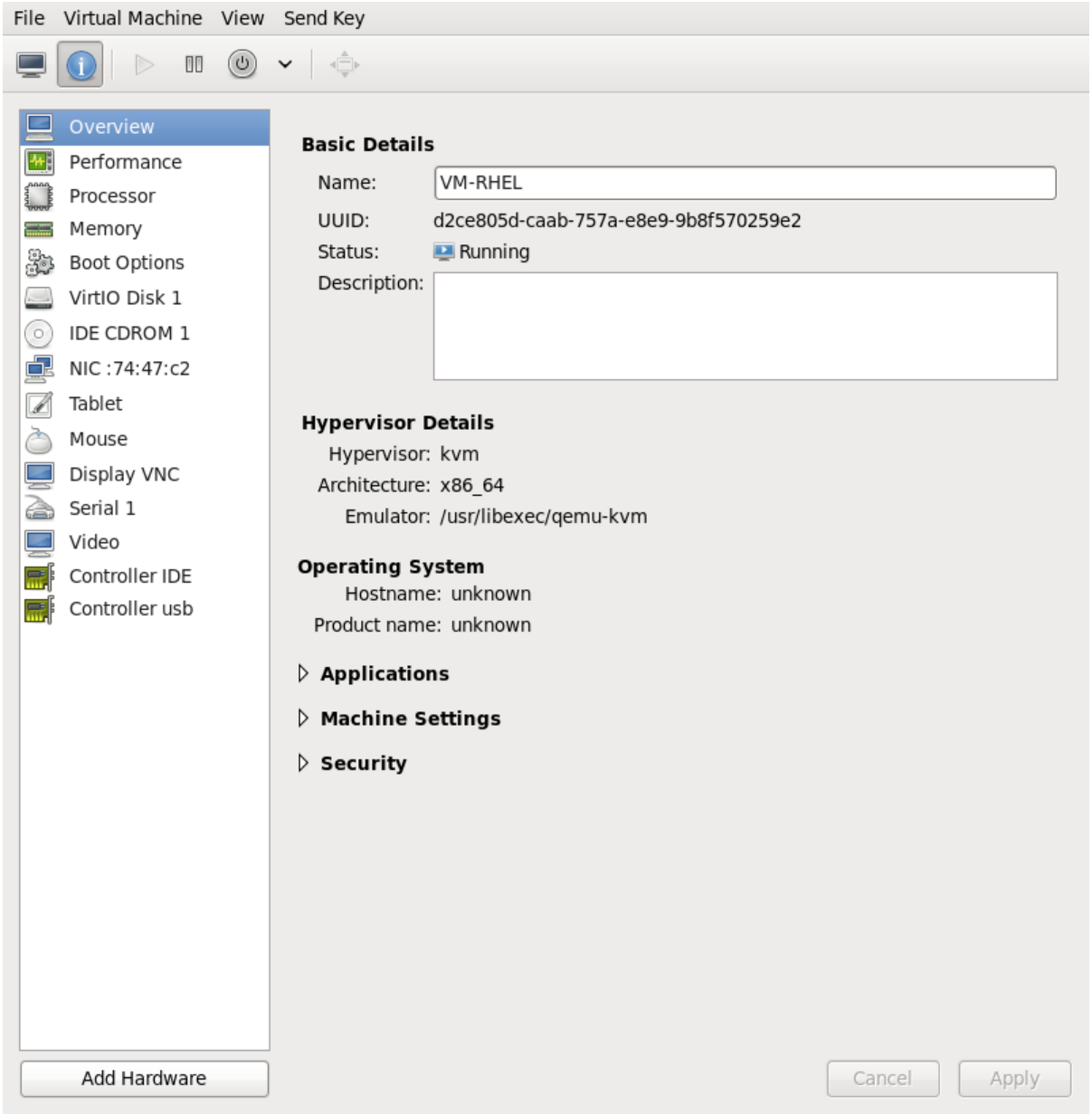


Figure 15.12. Displaying guest details overview

3. Select **Performance** from the navigation pane on the left hand side.

   The **Performance** view shows a summary of guest performance, including CPU and Memory usage.

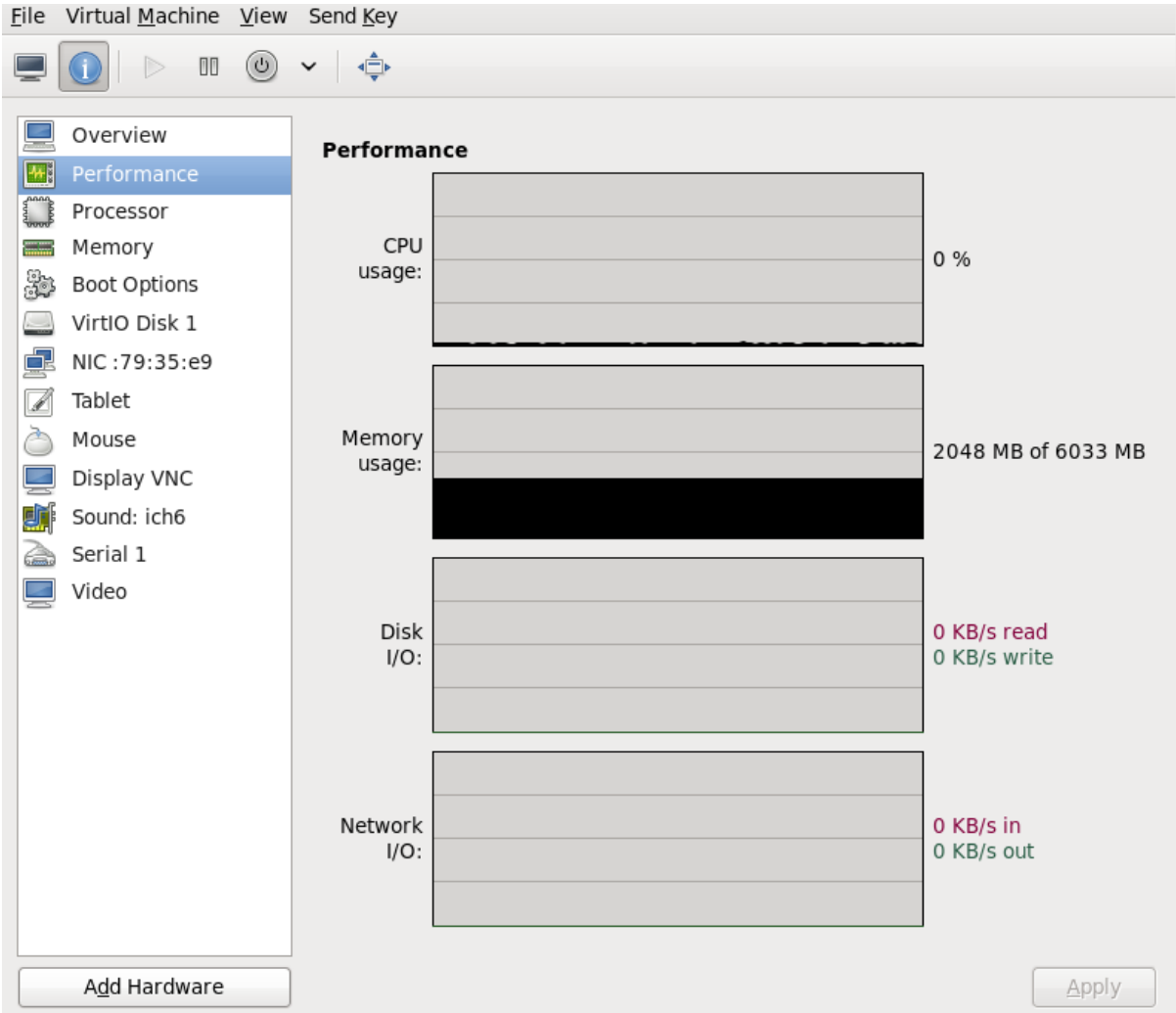


Figure 15.13. Displaying guest performance details

4. Select **Processor** from the navigation pane on the left hand side. The **Processor** view allows you to view the current processor allocation, as well as to change it.

   It is also possible to change the number of virtual CPUs (vCPUs) while the virtual machine is running, which is referred to as *hot plugging* and *hot unplugging*.

   IMPORTANT

   The hot unplugging feature is only available as a Technology Preview. Therefore, it is not supported and not recommended for use in high-value deployments.

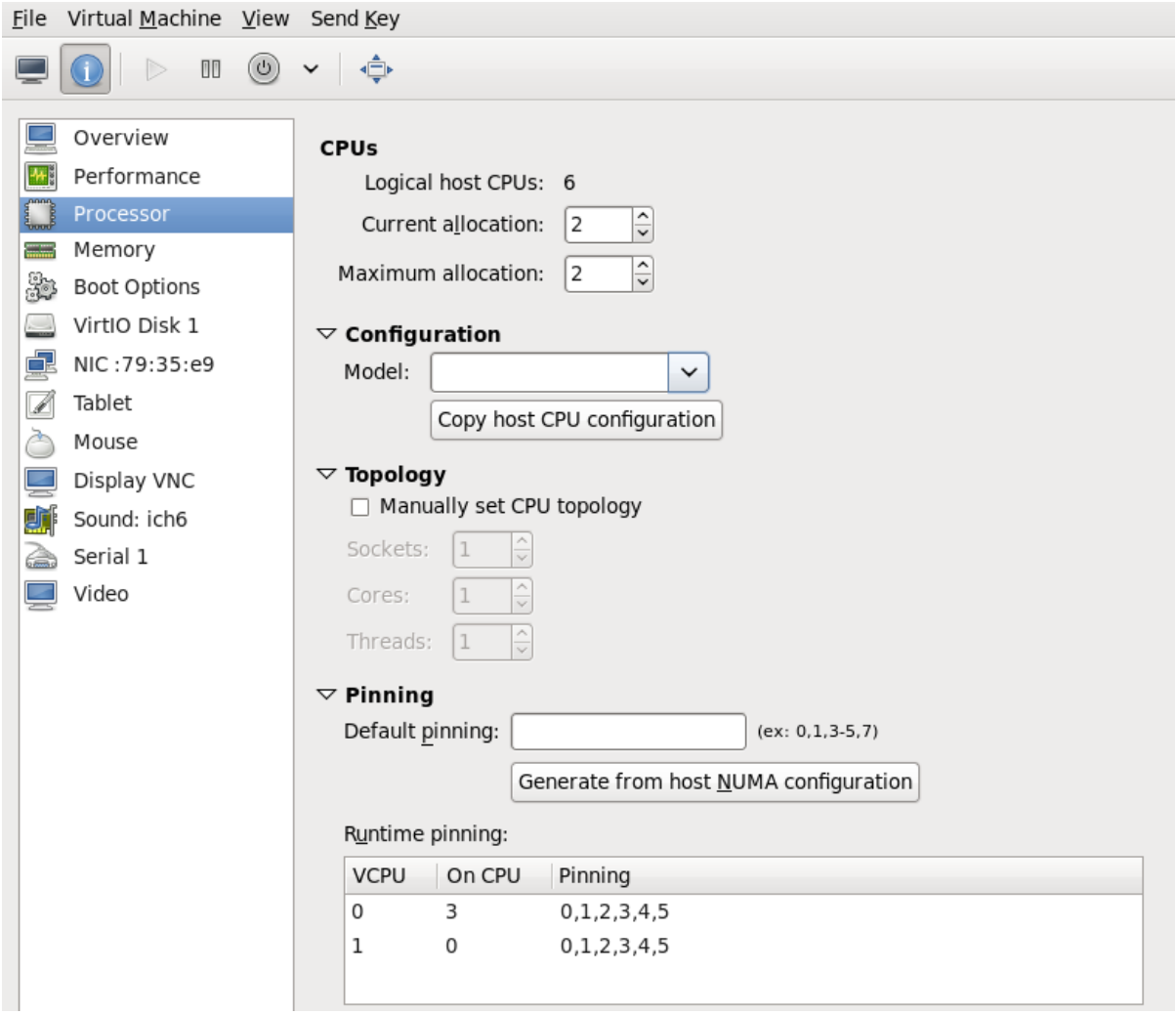


Figure 15.14. Processor allocation panel

5. Select **Memory** from the navigation pane on the left hand side. The **Memory** view allows you to view or change the current memory allocation.

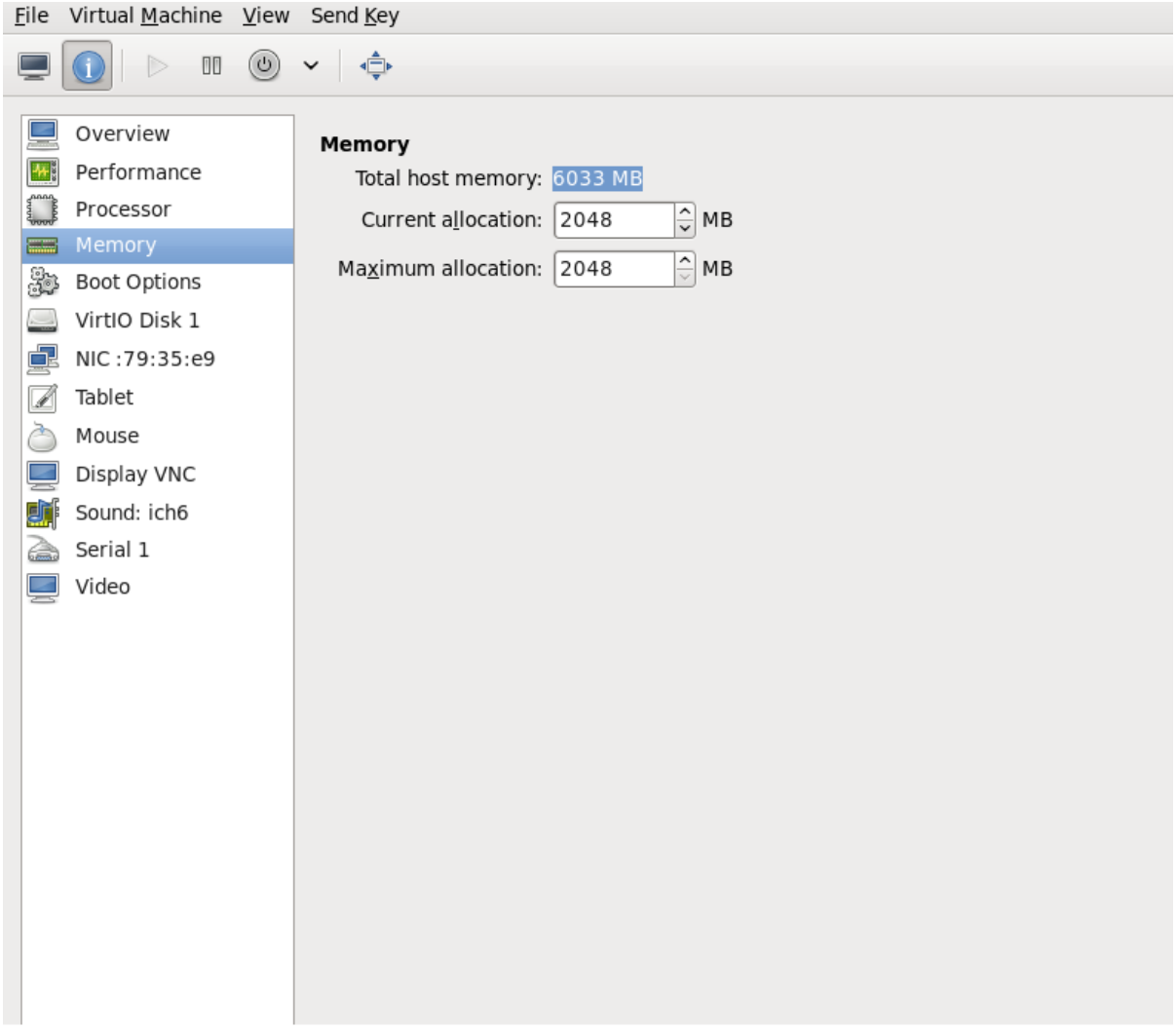


Figure 15.15. Displaying memory allocation

6. Each virtual disk attached to the virtual machine is displayed in the navigation pane. Click on a virtual disk to modify or remove it.

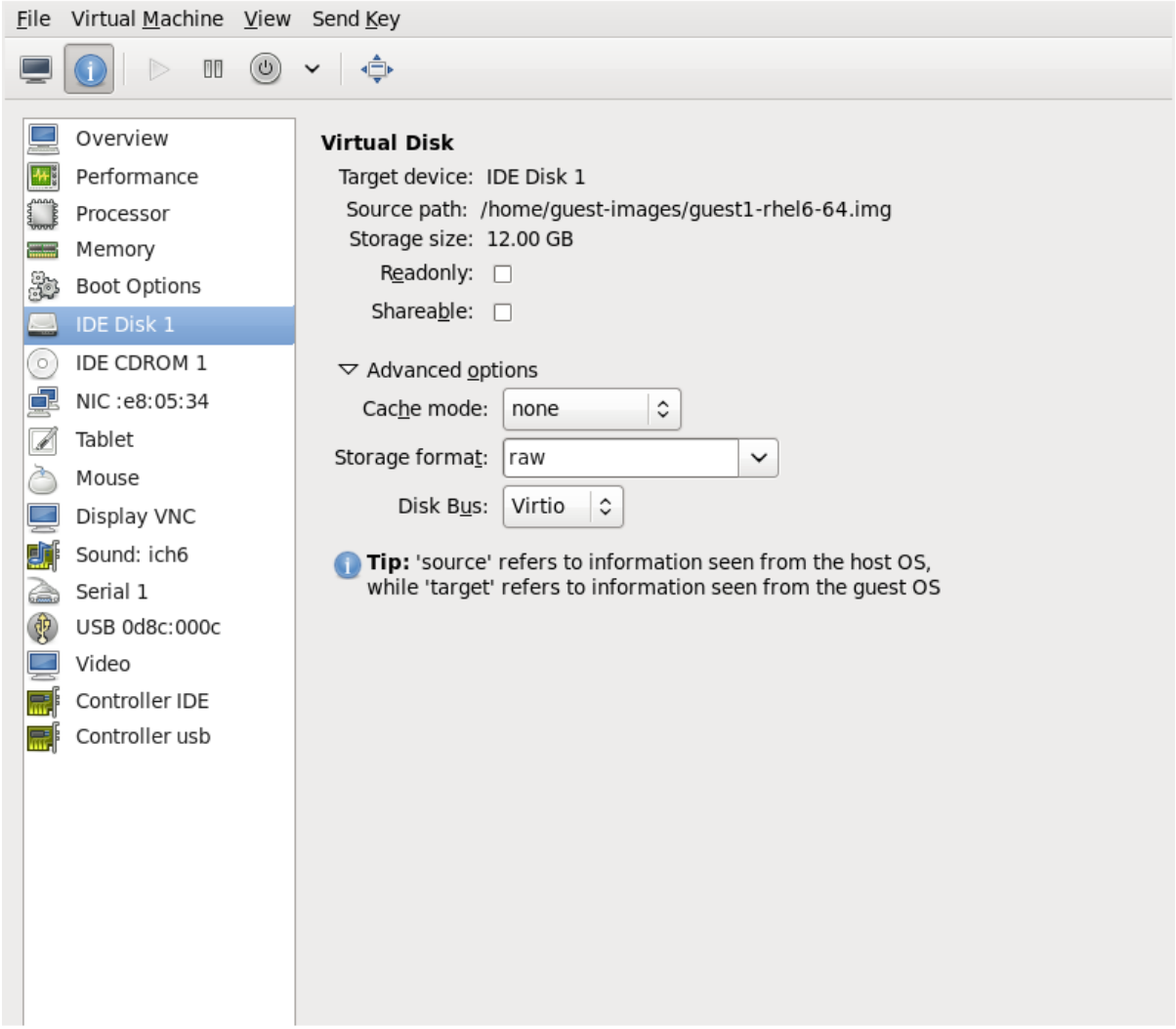


Figure 15.16. Displaying disk configuration

7. Each virtual network interface attached to the virtual machine is displayed in the navigation pane. Click on a virtual network interface to modify or remove it.

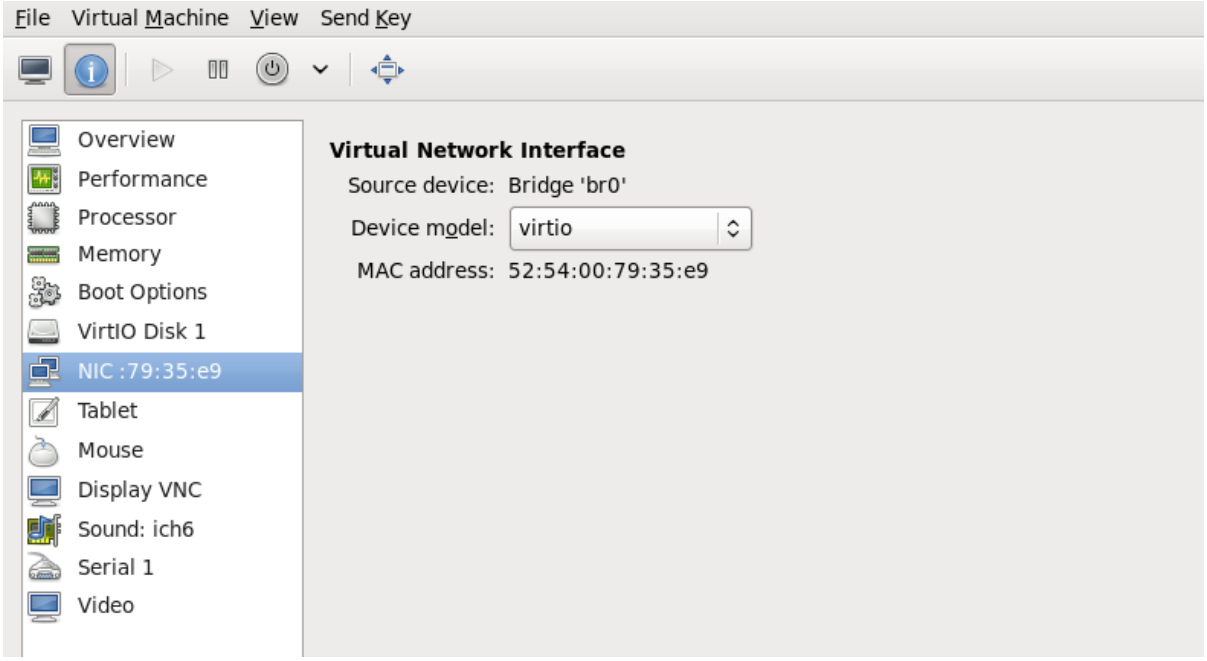


Figure 15.17. Displaying network configuration

15.7. PERFORMANCE MONITORING

Performance monitoring preferences can be modified with **virt-manager**'s preferences window.

To configure performance monitoring:

1. From the **Edit** menu, select **Preferences**.

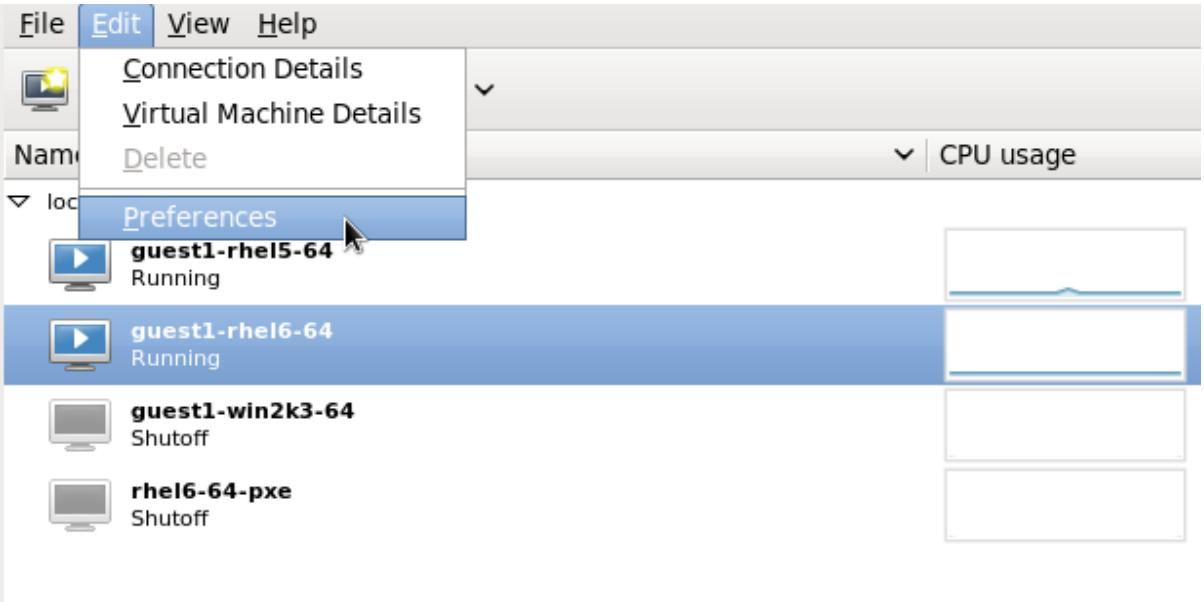


Figure 15.18. Modifying guest preferences

The **Preferences** window appears.

2. From the **Stats** tab specify the time in seconds or stats polling options.

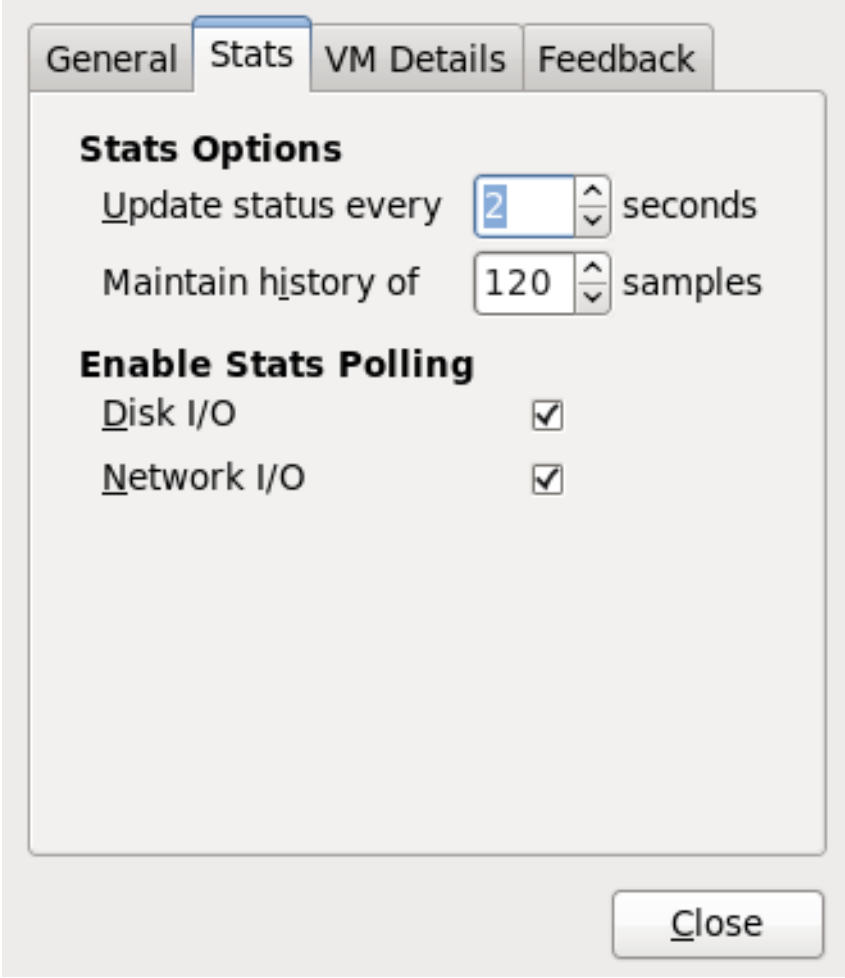


Figure 15.19. Configuring performance monitoring

15.8. DISPLAYING CPU USAGE FOR GUESTS

To view the CPU usage for all guests on your system:

1. From the **View** menu, select **Graph**, then the **Guest CPU Usage** check box.

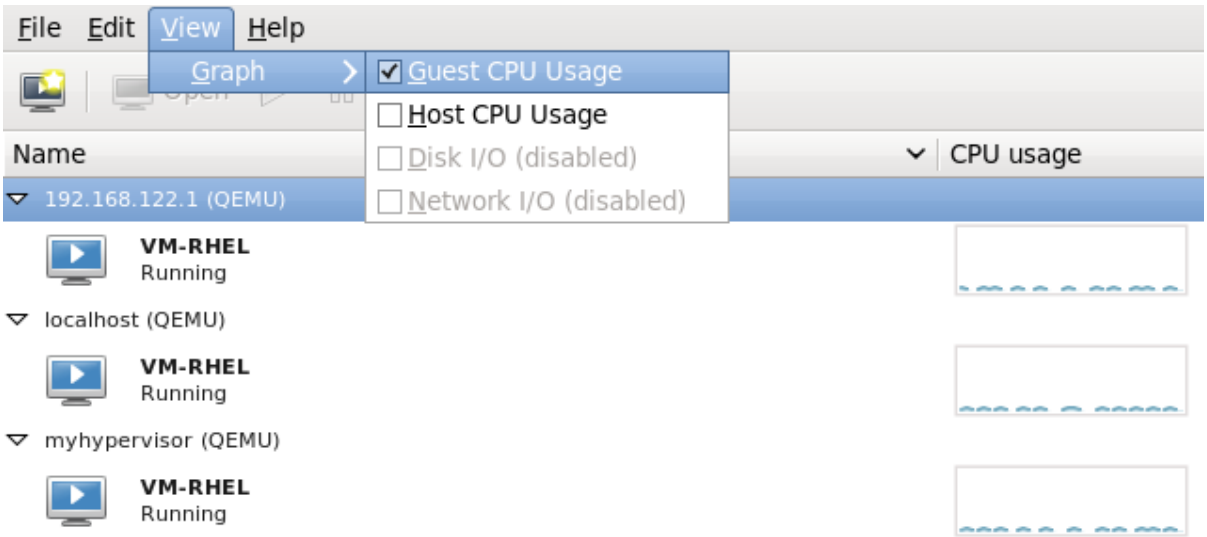


Figure 15.20. Enabling guest CPU usage statistics graphing

2. The Virtual Machine Manager shows a graph of CPU usage for all virtual machines on your system.

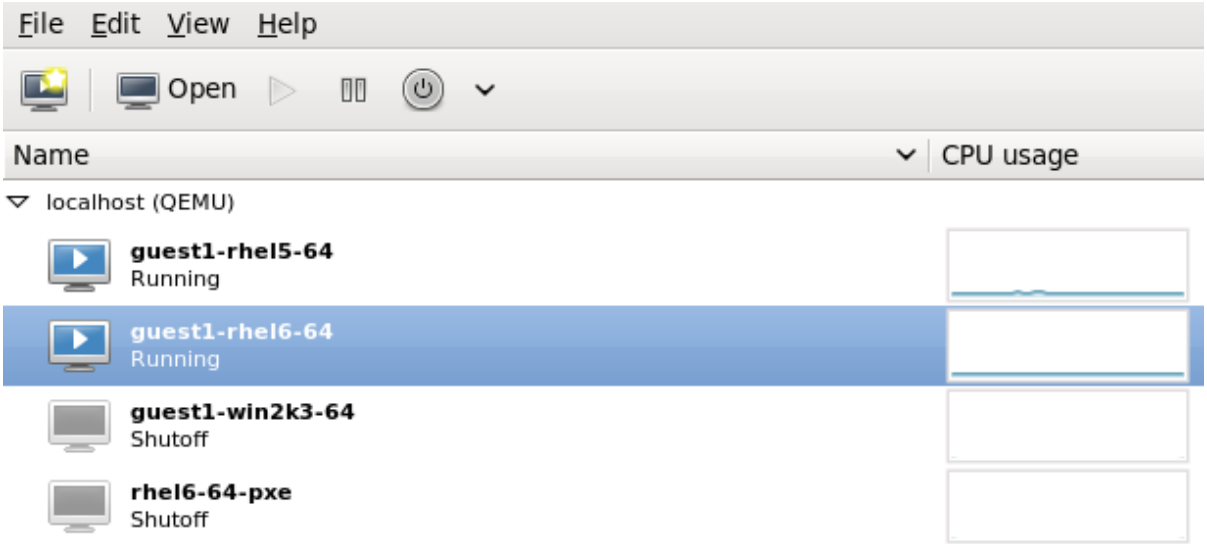


Figure 15.21. Guest CPU usage graph

15.9. DISPLAYING CPU USAGE FOR HOSTS

To view the CPU usage for all hosts on your system:

1. From the **View** menu, select **Graph**, then the **Host CPU Usage** check box.

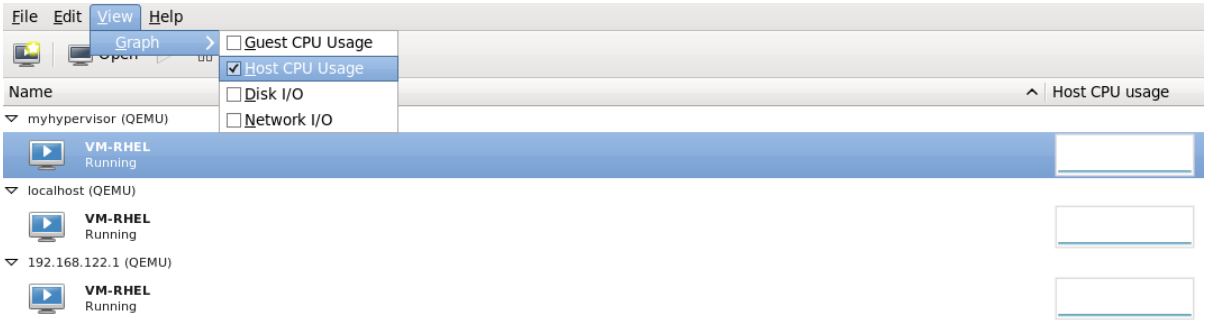


Figure 15.22. Enabling host CPU usage statistics graphing

2. The Virtual Machine Manager shows a graph of host CPU usage on your system.

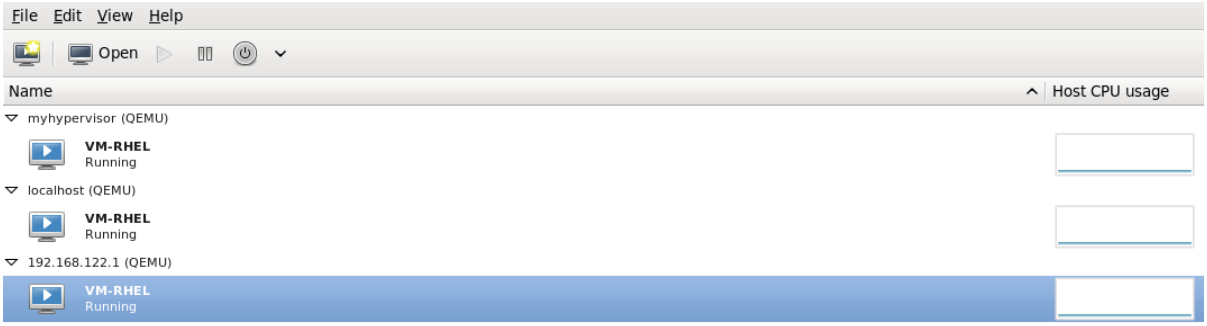


Figure 15.23. Host CPU usage graph

15.10. DISPLAYING DISK I/O

To view the disk I/O for all virtual machines on your system:

1. Make sure that the Disk I/O statistics collection is enabled. To do this, from the **Edit** menu, select **Preferences** and click the **Stats**tab.

2. Select the **Disk I/O** check box.

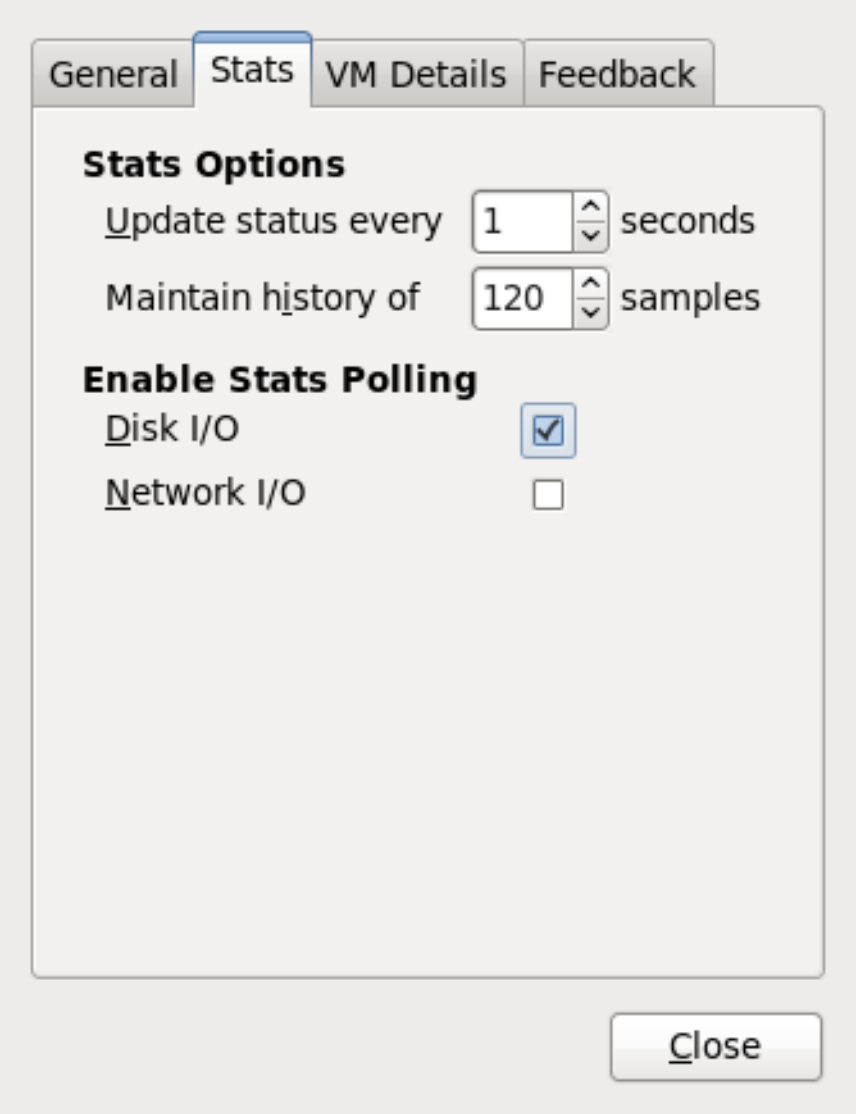


Figure 15.24. Enabling Disk I/O

3. To enable the Disk I.O display, from the **View** menu, select **Graph**, then the **Disk I/O** check box.

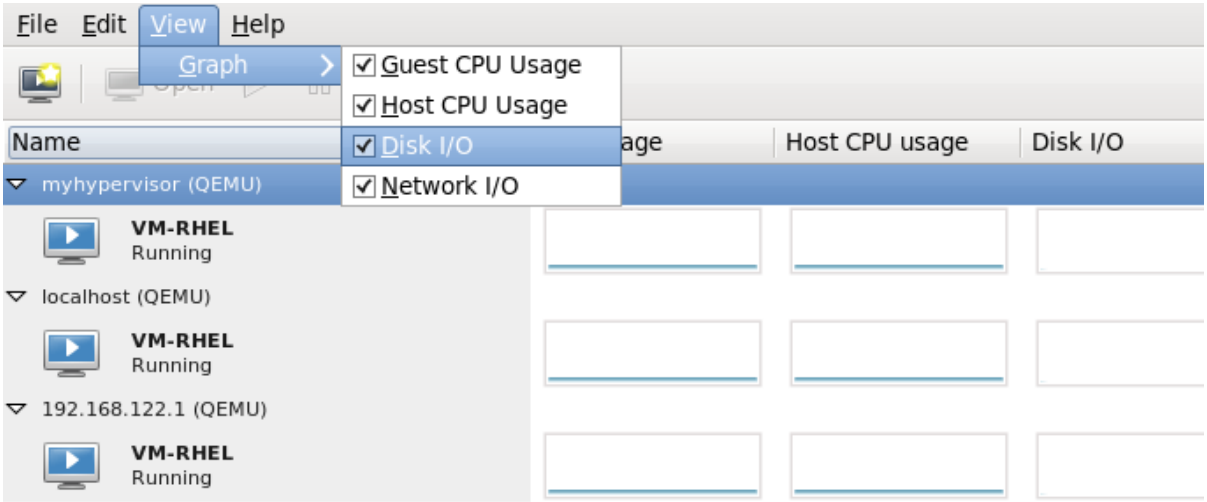


Figure 15.25. Selecting Disk I/O

4. The Virtual Machine Manager shows a graph of Disk I/O for all virtual machines on your system.

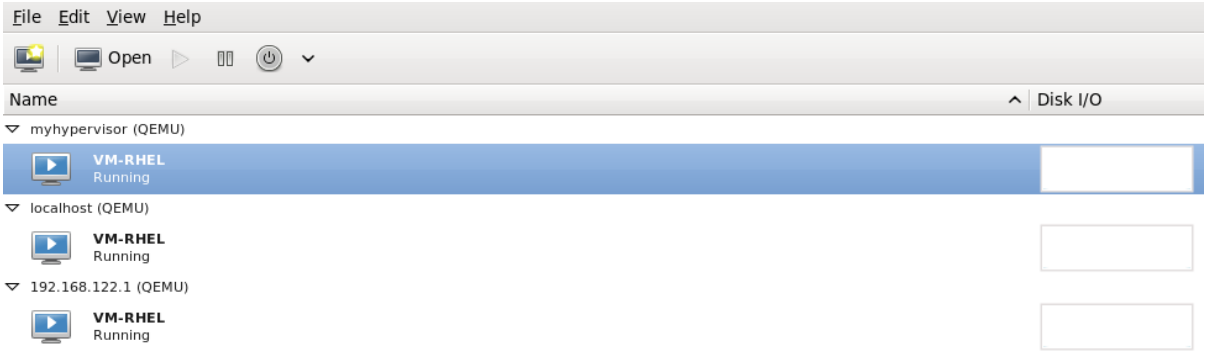


Figure 15.26. Displaying Disk I/O

15.11. DISPLAYING NETWORK I/O

To view the network I/O for all virtual machines on your system:

1. Make sure that the Network I/O statistics collection is enabled. To do this, from the **Edit** menu, select **Preferences** and click the **Stats**tab.

2. Select the **Network I/O** check box.

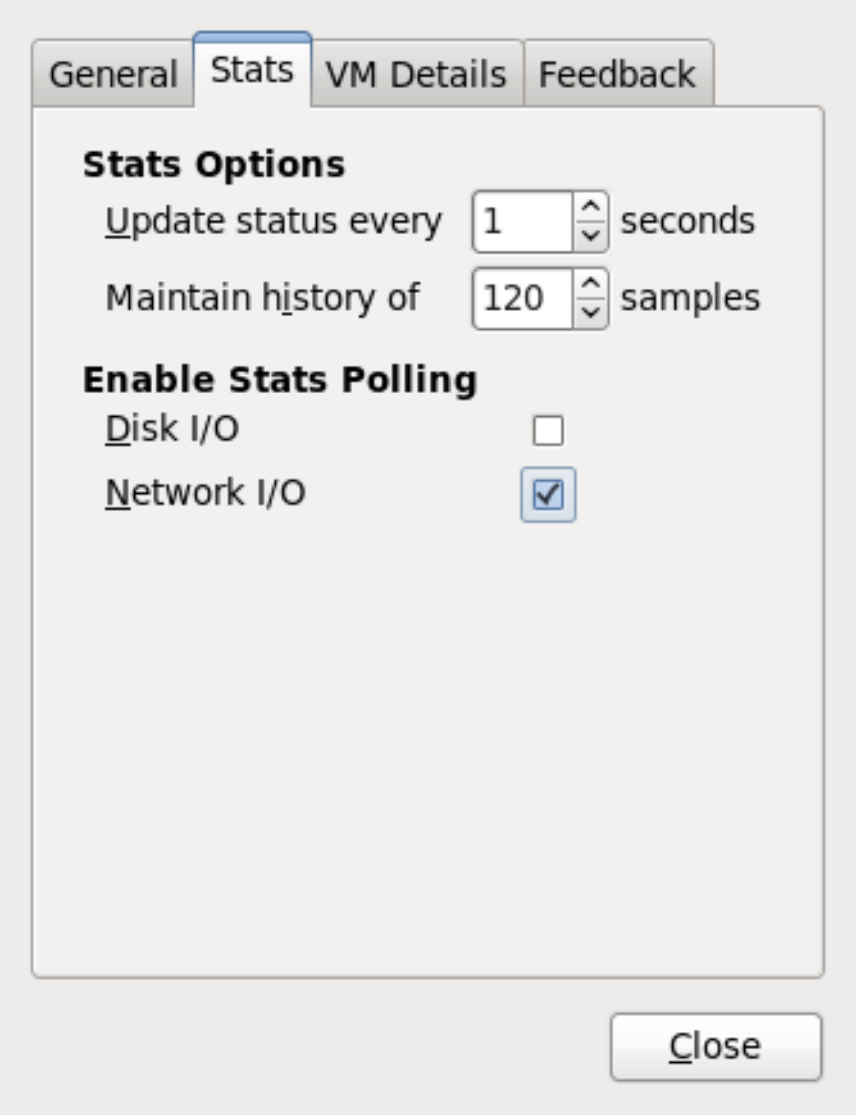


Figure 15.27. Enabling Network I/O

3. To display the Network I/O statistics, from the **View** menu, select **Graph**, then the **Network I/O** check box.

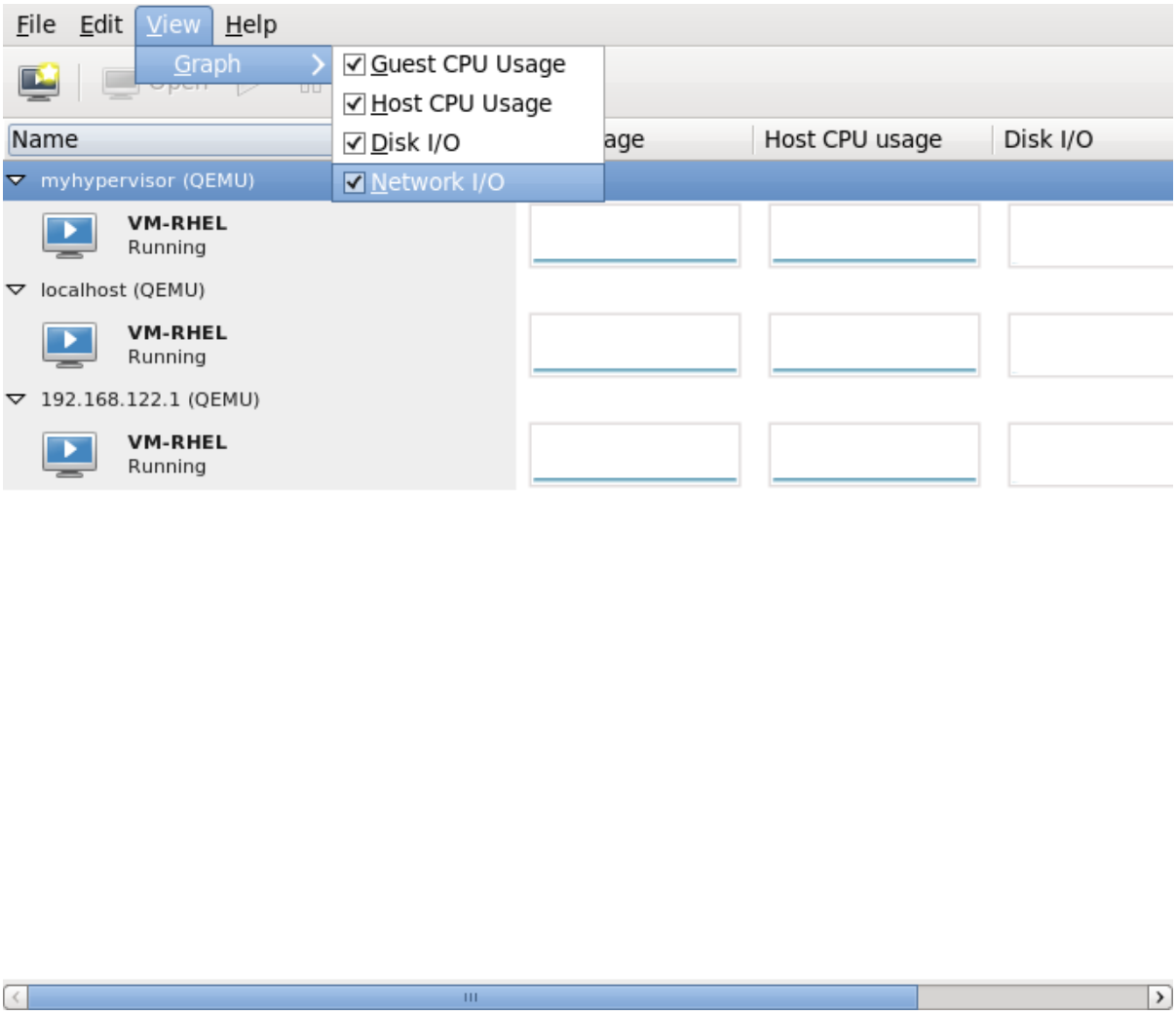


Figure 15.28. Selecting Network I/O

4. The Virtual Machine Manager shows a graph of Network I/O for all virtual machines on your system.

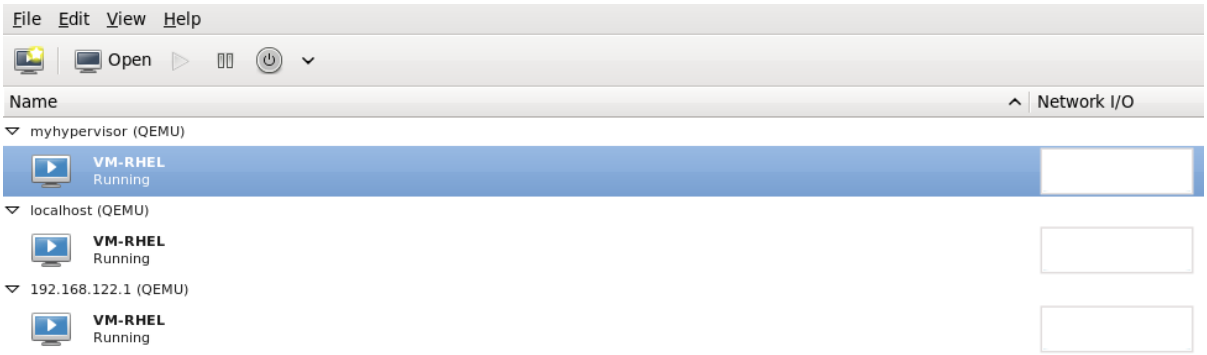


Figure 15.29. Displaying Network I/O

CHAPTER 16. GUEST VIRTUAL MACHINE DISK ACCESS WITH OFFLINE TOOLS

16.1. INTRODUCTION

Red Hat Enterprise Linux 6 comes with tools to access, edit and create host physical machine disks or other disk images. There are several uses for these tools, including:

- Viewing or downloading files located on a host physical machine disk.
- Editing or uploading files onto a host physical machine disk.
- Reading or writing host physical machine configuration.
- Reading or writing the Windows Registry in Windows host physical machines.
- Preparing new disk images containing files, directories, file systems, partitions, logical volumes and other options.
- Rescuing and repairing host physical machines that fail to boot or those that need boot configuration changes.
- Monitoring disk usage of host physical machines.
- Auditing compliance of host physical machines, for example to organizational security standards.
- Deploying host physical machines by cloning and modifying templates.
- Reading CD and DVD ISO and floppy disk images.

WARNING

You must **never** use these tools to write to a host physical machine or disk image which is attached to a running virtual machine, not even to open such a disk image in write mode. Doing so will result in disk corruption of the guest virtual machine. The tools try to prevent you from doing this, however do not catch all cases. If there is any suspicion that a guest virtual machine might be running, it is strongly recommended that the tools not be used, or at least **always** use the tools in read-only mode.

NOTE

Some virtualization commands in Red Hat Enterprise Linux 6 allow you to specify a remote libvirt connection. For example:

```
virt-df -c qemu://remote/system -d Guest
```

However, libguestfs in Red Hat Enterprise Linux 6 cannot access remote guests, and commands using remote URLs like this do not work as expected. This affects the following Red Hat Enterprise Linux 6 commands:

- guestfish
- guestmount
- virt-alignment-scan
- virt-cat
- virt-copy-in
- virt-copy-out
- virt-df
- virt-edit
- virt-filesystems
- virt-inspector
- virt-inspector2
- virt-list-filesystems
- virt-list-partitions
- virt-ls
- virt-rescue
- virt-sysprep
- virt-tar
- virt-tar-in
- virt-tar-out
- virt-win-reg

16.2. TERMINOLOGY

This section explains the terms used throughout this chapter.

- **libguestfs (Guest file system library)**- the underlying C library that provides the basic functionality for opening disk images, reading and writing files and so on. You can write C programs directly to this API, but it is quite low level.
- **guestfish (Guest file system interactive shell)**is an interactive shell that you can use from the command line or from shell scripts. It exposes all of the functionality of the libguestfs API.
- Various virt tools are built on top of libguestfs, and these provide a way to perform specific single tasks from the command line. Tools include **virt-df**, **virt-rescue**, **virt-resize** and **virt-edit**.
- **hivex** and **Augeas** are libraries for editing the Windows Registry and Linux configuration files respectively. Although these are separate from libguestfs, much of the value of libguestfs comes from the combination of these tools.
- **guestmount** is an interface between libguestfs and FUSE. It is primarily used to mount file systems from disk images on your host physical machine. This functionality is not necessary, but can be useful.

16.3. INSTALLATION

To install libguestfs, guestfish, the libguestfs tools, guestmount and support for Windows guest virtual machines, subscribe to the Red Hat Enterprise Linux V2WIN channel, go to the Red Hat Website and run the following command:

```
# yum install libguestfs guestfish libguestfs-tools libguestfs-winsupport
```

To install every libguestfs-related package including the language bindings, run the following command:

```
# yum install '*guestf*'
```

16.4. THE GUESTFISH SHELL

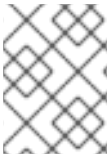
guestfish is an interactive shell that you can use from the command line or from shell scripts to access guest virtual machine file systems. All of the functionality of the libguestfs API is available from the shell.

To begin viewing or editing a virtual machine disk image, run the following command, substituting the path to your desired disk image:

```
guestfish --ro -a /path/to/disk/image
```

--ro means that the disk image is opened read-only. This mode is always safe but does not allow write access. Only omit this option when you are **certain** that the guest virtual machine is not running, or the disk image is not attached to a live guest virtual machine. It is not possible to use **libguestfs** to edit a live guest virtual machine, and attempting to will result in irreversible disk corruption.

/path/to/disk/image is the path to the disk. This can be a file, a host physical machine logical volume (such as /dev/VG/LV), a host physical machine device (/dev/cdrom) or a SAN LUN (/dev/sdf3).



NOTE

libguestfs and guestfish do not require root privileges. You only need to run them as root if the disk image being accessed needs root to read or write or both.

When you start guestfish interactively, it will display this prompt:

```
 guestfish --ro -a /path/to/disk/image

Welcome to guestfish, the libguestfs filesystem interactive shell for editing virtual machine
filesystems.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

><fs>
```

At the prompt, type **run** to initiate the library and attach the disk image. This can take up to 30 seconds the first time it is done. Subsequent starts will complete much faster.

NOTE

libguestfs will use hardware virtualization acceleration such as KVM (if available) to speed up this process.

Once the **run** command has been entered, other commands can be used, as the following section demonstrates.

16.4.1. Viewing File Systems with guestfish

This section provides information about viewing files with guestfish.

16.4.1.1. Manual Listing and Viewing

The **list-filesystems** command will list file systems found by libguestfs. This output shows a Red Hat Enterprise Linux 4 disk image:

```
><fs> run
><fs> list-filesystems
/dev/vda1: ext3
/dev/VolGroup00/LogVol00: ext3
/dev/VolGroup00/LogVol01: swap
```

This output shows a Windows disk image:

```
><fs> run
><fs> list-filesystems
/dev/vda1: ntfs
/dev/vda2: ntfs
```

Other useful commands are **list-devices**, **list-partitions**, **lvs**, **pvs**, **vfs-type** and **file**. You can get more information and help on any command by typing **help** *command*, as shown in the following output:

```
><fs> help vfs-type
 NAME
   vfs-type - get the Linux VFS type corresponding to a mounted device
```

```
SYNOPSIS
    vfs-type device

DESCRIPTION
   This command gets the file system type corresponding to the file system on
   "device".

   For most file systems, the result is the name of the Linux VFS module
   which would be used to mount this file system if you mounted it without
   specifying the file system type. For example a string such as "ext3" or
   "ntfs".
```

To view the actual contents of a file system, it must first be mounted. This example uses one of the Windows partitions shown in the previous output (**/dev/vda2**), which in this case is known to correspond to the **C:** drive:

```
><fs> mount-ro /dev/vda2 /
><fs> ll /
total 1834753
 drwxrwxrwx  1 root root      4096 Nov  1 11:40 .
 drwxr-xr-x 21 root root      4096 Nov 16 21:45 ..
 lrwxrwxrwx  2 root root        60 Jul 14  2009 Documents and Settings
 drwxrwxrwx  1 root root      4096 Nov 15 18:00 Program Files
 drwxrwxrwx  1 root root      4096 Sep 19 10:34 Users
 drwxrwxrwx  1 root root     16384 Sep 19 10:34 Windows
```

You can use guestfish commands such as **ls**, **ll**, **cat**, **more**, **download** and **tar-out** to view and download files and directories.

NOTE

There is no concept of a current working directory in this shell. Unlike ordinary shells, you cannot for example use the **cd** command to change directories. All paths must be fully qualified starting at the top with a forward slash (**/**) character. Use the *Tab* key to complete paths.

To exit from the guestfish shell, type **exit** or enter **Ctrl+d**.

16.4.1.2. Using guestfish inspection

Instead of listing and mounting file systems by hand, it is possible to let guestfish itself inspect the image and mount the file systems as they would be in the guest virtual machine. To do this, add the **-i** option on the command line:

```
guestfish --ro -a /path/to/disk/image -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

 Operating system: Red Hat Enterprise Linux AS release 4 (Nahant Update 8)
```

```
/dev/VolGroup00/LogVol00 mounted on /
/dev/vda1 mounted on /boot

><fs> ll /
total 210
drwxr-xr-x. 24 root root  4096 Oct 28 09:09 .
drwxr-xr-x  21 root root  4096 Nov 17 15:10 ..
drwxr-xr-x.  2 root root  4096 Oct 27 22:37 bin
drwxr-xr-x.  4 root root  1024 Oct 27 21:52 boot
drwxr-xr-x.  4 root root  4096 Oct 27 21:21 dev
drwxr-xr-x. 86 root root 12288 Oct 28 09:09 etc
[etc]
```

Because guestfish needs to start up the libguestfs back end in order to perform the inspection and mounting, the **run** command is not necessary when using the **-i** option. The **-i** option works for many common Linux and Windows guest virtual machines.

16.4.1.3. Accessing a guest virtual machine by name

A guest virtual machine can be accessed from the command line when you specify its name as known to libvirt (in other words, as it appears in **virsh list --all**). Use the **-d** option to access a guest virtual machine by its name, with or without the **-i** option:

```
guestfish --ro -d GuestName -i
```

16.4.2. Modifying Files with guestfish

To modify files, create directories or make other changes to a guest virtual machine, first heed the warning at the beginning of this section: your guest virtual machine must be shut down . Editing or changing a running disk with guestfish **will** result in disk corruption. This section gives an example of editing the **/boot/grub/grub.conf** file. When you are sure the guest virtual machine is shut down you can omit the **--ro** option in order to get write access via a command such as:

```
guestfish -d RHEL3 -i

Welcome to guestfish, the libguestfs filesystem interactive shell for
 editing virtual machine filesystems.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

 Operating system: Red Hat Enterprise Linux AS release 3 (Taroon Update 9)
 /dev/vda2 mounted on /
 /dev/vda1 mounted on /boot

><fs> edit /boot/grub/grub.conf
```

Commands to edit files include **edit**, **vi** and **emacs**. Many commands also exist for creating files and directories, such as **write**, **mkdir**, **upload** and **tar-in**.

16.4.3. Other Actions with guestfish

You can also format file systems, create partitions, create and resize LVM logical volumes and much more, with commands such as **mkfs**, **part-add**, **lvresize**, **lvcreate**, **vgcreate** and **pvcreate**.

16.4.4. Shell Scripting with guestfish

Once you are familiar with using guestfish interactively, according to your needs, writing shell scripts with it may be useful. The following is a simple shell script to add a new MOTD (message of the day) to a guest:

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$guestname" -i <<'EOF'
   write /etc/motd "Welcome to Acme Incorporated."
   chmod 0644 /etc/motd
 EOF
```

16.4.5. Augeas and libguestfs Scripting

Combining libguestfs with Augeas can help when writing scripts to manipulate Linux guest virtual machine configuration. For example, the following script uses Augeas to parse the keyboard configuration of a guest virtual machine, and to print out the layout. Note that this example only works with guest virtual machines running Red Hat Enterprise Linux:

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$1" -i --ro <<'EOF'
   aug-init / 0
   aug-get /files/etc/sysconfig/keyboard/LAYOUT
 EOF
```

Augeas can also be used to modify configuration files. You can modify the above script to **change** the keyboard layout:

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$1" -i <<'EOF'
   aug-init / 0
   aug-set /files/etc/sysconfig/keyboard/LAYOUT '"gb"'
   aug-save
 EOF
```

Note the three changes between the two scripts:

1. The **--ro** option has been removed in the second example, giving the ability to write to the guest virtual machine.

2. The **aug-get** command has been changed to **aug-set** to modify the value instead of fetching it. The new value will be **"gb"** (including the quotes).

3. The **aug-save** command is used here so Augeas will write the changes out to disk.

> **NOTE**
>
> More information about Augeas can be found on the website http://augeas.net.

guestfish can do much more than we can cover in this introductory document. For example, creating disk images from scratch:

```
guestfish -N fs
```

Or copying out whole directories from a disk image:

```
><fs> copy-out /home /tmp/home
```

For more information see the man page guestfish(1).

16.5. OTHER COMMANDS

This section describes tools that are simpler equivalents to using guestfish to view and edit guest virtual machine disk images.

- **virt-cat** is similar to the guestfish **download** command. It downloads and displays a single file to the guest virtual machine. For example:

  ```
  # virt-cat RHEL3 /etc/ntp.conf | grep ^server
   server     127.127.1.0       # local clock
  ```

- **virt-edit** is similar to the guestfish **edit** command. It can be used to interactively edit a single file within a guest virtual machine. For example, you may need to edit the **grub.conf** file in a Linux-based guest virtual machine that will not boot:

  ```
  # virt-edit LinuxGuest /boot/grub/grub.conf
  ```

  virt-edit has another mode where it can be used to make simple non-interactive changes to a single file. For this, the **-e** option is used. This command, for example, changes the root password in a Linux guest virtual machine to having no password:

  ```
  # virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/'
  ```

- **virt-ls** is similar to the guestfish **ls**, **ll** and **find** commands. It is used to list a directory or directories (recursively). For example, the following command would recursively list files and directories under /home in a Linux guest virtual machine:

  ```
  # virt-ls -R LinuxGuest /home/ | less
  ```

16.6. VIRT-RESCUE: THE RESCUE SHELL

This section provides information about using the rescue shell.

16.6.1. Introduction

This section describes **virt-rescue**, which can be considered analogous to a rescue CD for virtual machines. It boots a guest virtual machine into a rescue shell so that maintenance can be performed to correct errors and the guest virtual machine can be repaired.

There is some overlap between virt-rescue and guestfish. It is important to distinguish their differing uses. virt-rescue is for making interactive, ad-hoc changes using ordinary Linux file system tools. It is particularly suited to rescuing a guest virtual machine that has failed . virt-rescue cannot be scripted.

In contrast, guestfish is particularly useful for making scripted, structured changes through a formal set of commands (the libguestfs API), although it can also be used interactively.

16.6.2. Running virt-rescue

Before you use **virt-rescue** on a guest virtual machine, make sure the guest virtual machine is not running, otherwise disk corruption will occur. When you are sure the guest virtual machine is not live, enter:

```
virt-rescue GuestName
```

(where GuestName is the guest name as known to libvirt), or:

```
virt-rescue /path/to/disk/image
```

(where the path can be any file, any logical volume, LUN, or so on) containing a guest virtual machine disk.

You will first see output scroll past, as virt-rescue boots the rescue VM. In the end you will see:

```
Welcome to virt-rescue, the libguestfs rescue shell.

 Note: The contents of / are the rescue appliance.
 You have to mount the guest virtual machine's partitions under /sysroot
 before you can examine them.

 bash: cannot set terminal process group (-1): Inappropriate ioctl for device
 bash: no job control in this shell
 ><rescue>
```

The shell prompt here is an ordinary bash shell, and a reduced set of ordinary Red Hat Enterprise Linux commands is available. For example, you can enter:

```
><rescue> fdisk -l /dev/vda
```

The previous command will list disk partitions. To mount a file system, it is suggested that you mount it under **/sysroot**, which is an empty directory in the rescue machine for the user to mount anything you like. Note that the files under / are files from the rescue VM itself:

```
><rescue> mount /dev/vda1 /sysroot/
EXT4-fs (vda1): mounted filesystem with ordered data mode. Opts: (null)
><rescue> ls -l /sysroot/grub/
 total 324
 -rw-r--r--. 1 root root     63 Sep 16 18:14 device.map
```

```
-rw-r--r--. 1 root root  13200 Sep 16 18:14 e2fs_stage1_5
-rw-r--r--. 1 root root  12512 Sep 16 18:14 fat_stage1_5
-rw-r--r--. 1 root root  11744 Sep 16 18:14 ffs_stage1_5
-rw-------. 1 root root   1503 Oct 15 11:19 grub.conf
[...]
```

When you are finished rescuing the guest virtual machine, exit the shell by entering **exit** or **Ctrl+d**.

virt-rescue has many command line options. The options most often used are:

- **--ro**: Operate in read-only mode on the guest virtual machine. No changes will be saved. You can use this to experiment with the guest virtual machine. As soon as you exit from the shell, all of your changes are discarded.
- **--network**: Enable network access from the rescue shell. Use this if you need to, for example, download RPM or other files into the guest virtual machine.

16.7. VIRT-DF: MONITORING DISK USAGE

This section provides information about monitoring disk usage using **virt-df**.

16.7.1. Introduction

This section describes **virt-df**, which displays file system usage from a disk image or a guest virtual machine. It is similar to the Linux **df** command, but for virtual machines.

16.7.2. Running virt-df

To display file system usage for all file systems found in a disk image, enter the following:

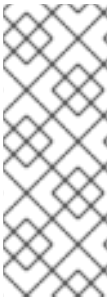
```
# virt-df /dev/vg_guests/RHEL6
Filesystem                1K-blocks       Used  Available  Use%
RHEL6:/dev/sda1                101086      10233      85634   11%
RHEL6:/dev/VolGroup00/LogVol00 7127864    2272744    4493036   32%
```

(Where **/dev/vg_guests/RHEL6** is a Red Hat Enterprise Linux 6 guest virtual machine disk image. The path in this case is the host physical machine logical volume where this disk image is located.)

You can also use **virt-df** on its own to list information about all of your guest virtual machines (ie. those known to libvirt). The **virt-df** command recognizes some of the same options as the standard **df** such as **-h** (human-readable) and **-i** (show inodes instead of blocks).

virt-df also works on Windows guest virtual machines:

```
# virt-df -h
Filesystem                     Size       Used  Available  Use%
F14x64:/dev/sda1               484.2M      66.3M     392.9M   14%
F14x64:/dev/vg_f14x64/lv_root    7.4G       3.0G       4.4G   41%
RHEL6brewx64:/dev/sda1         484.2M      52.6M     406.6M   11%
RHEL6brewx64:/dev/vg_rhel6brewx64/lv_root
                              13.3G       3.4G       9.2G   26%
Win7x32:/dev/sda1              100.0M      24.1M      75.9M   25%
Win7x32:/dev/sda2      19.9G      7.4G      12.5G   38%
```



NOTE

You can use **virt-df** safely on live guest virtual machines, since it only needs read-only access. However, you should not expect the numbers to be precisely the same as those from a **df** command running inside the guest virtual machine. This is because what is on disk will be slightly out of synch with the state of the live guest virtual machine. Nevertheless it should be a good enough approximation for analysis and monitoring purposes.

virt-df is designed to allow you to integrate the statistics into monitoring tools, databases and so on. This allows system administrators to generate reports on trends in disk usage, and alerts if a guest virtual machine is about to run out of disk space. To do this you should use the **--csv** option to generate machine-readable Comma-Separated-Values (CSV) output. CSV output is readable by most databases, spreadsheet software and a variety of other tools and programming languages. The raw CSV looks like the following:

```
# virt-df --csv WindowsGuest
 Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
 Win7x32,/dev/sda1,102396,24712,77684,24.1%
 Win7x32,/dev/sda2,20866940,7786652,13080288,37.3%
```

For resources and ideas on how to process this output to produce trends and alerts, refer to the following URL: http://libguestfs.org/virt-df.1.html.

16.8. VIRT-RESIZE: RESIZING GUEST VIRTUAL MACHINES OFFLINE

This section provides information about resizing offline guest virtual machines.

16.8.1. Introduction

This section describes **virt-resize**, a tool for expanding or shrinking guest virtual machines. It only works for guest virtual machines which are offline (shut down). It works by copying the guest virtual machine image and leaving the original disk image untouched. This is ideal because you can use the original image as a backup, however there is a trade-off as you need twice the amount of disk space.

16.8.2. Expanding a Disk Image

This section demonstrates a simple case of expanding a disk image:

1. Locate the disk image to be resized. You can use the command **virsh dumpxml GuestName** for a libvirt guest virtual machine.

2. Decide on how you wish to expand the guest virtual machine. Run **virt-df -h** and **virt-list-partitions -lh** on the guest virtual machine disk, as shown in the following output:

```
# virt-df -h /dev/vg_guests/RHEL6
Filesystem                    Size       Used  Available  Use%
RHEL6:/dev/sda1               98.7M      10.0M      83.6M   11%
RHEL6:/dev/VolGroup00/LogVol00  6.8G       2.2G       4.3G   32%

# virt-list-partitions -lh /dev/vg_guests/RHEL6
/dev/sda1 ext3 101.9M
/dev/sda2 pv 7.9G
```

This example will demonstrate how to:

- Increase the size of the first (boot) partition, from approximately 100MB to 500MB.
- Increase the total disk size from 8GB to 16GB.
- Expand the second partition to fill the remaining space.
- Expand **/dev/VolGroup00/LogVol00** to fill the new space in the second partition.

1. Make sure the guest virtual machine is shut down.
2. Rename the original disk as the backup. How you do this depends on the host physical machine storage environment for the original disk. If it is stored as a file, use the **mv** command. For logical volumes (as demonstrated in this example), use **lvrename**:

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. Create the new disk. The requirements in this example are to expand the total disk size up to 16GB. Since logical volumes are used here, the following command is used:

```
# lvcreate -L 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. The requirements from step 2 are expressed by this command:

```
# virt-resize \
      /dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 \
      --resize /dev/sda1=500M \
      --expand /dev/sda2 \
      --LV-expand /dev/VolGroup00/LogVol00
```

The first two arguments are the input disk and output disk. **--resize /dev/sda1=500M** resizes the first partition up to 500MB. **--expand /dev/sda2** expands the second partition to fill all remaining space. **--LV-expand /dev/VolGroup00/LogVol00** expands the guest virtual machine logical volume to fill the extra space in the second partition.

virt-resize describes what it is doing in the output:

```
Summary of changes:
  /dev/sda1: partition will be resized from 101.9M to 500.0M
  /dev/sda1: content will be expanded using the 'resize2fs' method
  /dev/sda2: partition will be resized from 7.9G to 15.5G
  /dev/sda2: content will be expanded using the 'pvresize' method
  /dev/VolGroup00/LogVol00: LV will be expanded to maximum size
  /dev/VolGroup00/LogVol00: content will be expanded using the 'resize2fs' method
  Copying /dev/sda1 ...
  [#####################################################]
  Copying /dev/sda2 ...
  [#####################################################]
  Expanding /dev/sda1 using the 'resize2fs' method
  Expanding /dev/sda2 using the 'pvresize' method
  Expanding /dev/VolGroup00/LogVol00 using the 'resize2fs' method
```

5. Try to boot the virtual machine. If it works (and after testing it thoroughly) you can delete the backup disk. If it fails, shut down the virtual machine, delete the new disk, and rename the backup disk back to its original name.

6. Use **virt-df** or **virt-list-partitions** to show the new size:

```
# virt-df -h /dev/vg_pin/RHEL6
  Filesystem                    Size      Used  Available  Use%
  RHEL6:/dev/sda1               484.4M     10.8M     448.6M    3%
  RHEL6:/dev/VolGroup00/LogVol00 14.3G      2.2G      11.4G   16%
```

Resizing guest virtual machines is not an exact science. If **virt-resize** fails, there are a number of tips that you can review and attempt in the virt-resize(1) man page. For some older Red Hat Enterprise Linux guest virtual machines, you may need to pay particular attention to the tip regarding GRUB.

16.9. VIRT-INSPECTOR: INSPECTING GUEST VIRTUAL MACHINES

This section provides information about inspecting guest virtual machines using **virt-inspector**.

16.9.1. Introduction

virt-inspector is a tool for inspecting a disk image to find out what operating system it contains.

NOTE

Red Hat Enterprise Linux 6.2 provides two variations of this program: **virt-inspector** is the original program as found in Red Hat Enterprise Linux 6.0 and is now deprecated upstream. **virt-inspector2** is the same as the new upstream **virt-inspector** program.

16.9.2. Installation

To install virt-inspector and the documentation, enter the following command:

```
# yum install libguestfs-tools libguestfs-devel
```

To process Windows guest virtual machines you must also install libguestfs-winsupport. Refer to Section 16.10.2, “Installation” for details. The documentation, including example XML output and a Relax-NG schema for the output, will be installed in **/usr/share/doc/libguestfs-devel-***/ where "*" is replaced by the version number of libguestfs.

16.9.3. Running virt-inspector

You can run **virt-inspector** against any disk image or libvirt guest virtual machine as shown in the following example:

```
virt-inspector --xml disk.img > report.xml
```

Or as shown here:

```
virt-inspector --xml GuestName > report.xml
```

The result will be an XML report (**report.xml**). The main components of the XML file are a top-level <operatingsystems> element containing usually a single <operatingsystem> element, similar to the following:

```
<operatingsystems>
  <operatingsystem>

    <!-- the type of operating system and Linux distribution -->
    <name>linux</name>
    <distro>rhel</distro>
    <!-- the name, version and architecture -->
    <product_name>Red Hat Enterprise Linux Server release 6.4 </product_name>
    <major_version>6</major_version>
    <minor_version>4</minor_version>
    <package_format>rpm</package_format>
    <package_management>yum</package_management>
    <root>/dev/VolGroup/lv_root</root>
    <!-- how the filesystems would be mounted when live -->
    <mountpoints>
      <mountpoint dev="/dev/VolGroup/lv_root">/</mountpoint>
      <mountpoint dev="/dev/sda1">/boot</mountpoint>
      <mountpoint dev="/dev/VolGroup/lv_swap">swap</mountpoint>
    </mountpoints>

   < !-- filesystems-->
     <filesystem dev="/dev/VolGroup/lv_root">
       <label></label>
       <uuid>b24d9161-5613-4ab8-8649-f27a8a8068d3</uuid>
       <type>ext4</type>
       <content>linux-root</content>
       <spec>/dev/mapper/VolGroup-lv_root</spec>
     </filesystem>
     <filesystem dev="/dev/VolGroup/lv_swap">
       <type>swap</type>
       <spec>/dev/mapper/VolGroup-lv_swap</spec>
     </filesystem>
    <!-- packages installed -->
    <applications>
      <application>
        <name>firefox</name>
        <version>3.5.5</version>
        <release>1.fc12</release>
      </application>
    </applications>

  </operatingsystem>
</operatingsystems>
```

Processing these reports is best done using W3C standard XPath queries. Red Hat Enterprise Linux 6 comes with a command line program (**xpath**) which can be used for simple instances; however, for long-term and advanced usage, you should consider using an XPath library along with your favorite programming language.

As an example, you can list out all file system devices using the following XPath query:

```
virt-inspector --xml GuestName | xpath //filesystem/@dev
```

```
Found 3 nodes:
-- NODE --
dev="/dev/sda1"
-- NODE --
dev="/dev/vg_f12x64/lv_root"
-- NODE --
dev="/dev/vg_f12x64/lv_swap"
```

Or list the names of all applications installed by entering:

```
virt-inspector --xml GuestName | xpath //application/name
[...long list...]
```

16.10. VIRT-WIN-REG: READING AND EDITING THE WINDOWS REGISTRY

16.10.1. Introduction

virt-win-reg is a tool that manipulates the Registry in Windows guest virtual machines. It can be used to read out registry keys. You can also use it to make changes to the Registry, but you must **never** try to do this for live/running guest virtual machines, as it will result in disk corruption.

16.10.2. Installation

To use **virt-win-reg** you must run the following:

```
# yum install /usr/bin/virt-win-reg
```

16.10.3. Using virt-win-reg

To read out Registry keys, specify the name of the guest virtual machine (or its disk image) and the name of the Registry key. You must use single quotes to surround the name of the desired key:

```
# virt-win-reg WindowsGuest \
    'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall' \
    | less
```

The output is in the standard text-based format used by **.REG** files on Windows.

NOTE

Hex-quoting is used for strings because the format does not properly define a portable encoding method for strings. This is the only way to ensure fidelity when transporting **.REG** files from one machine to another.

You can make hex-quoted strings printable by piping the output of **virt-win-reg** through this simple Perl script:

```
perl -MEncode -pe's?hex\((\d+)\):(\S+)?
$t=$1;$_=$2;s,\,,,g;"str($t):\"".decode(utf16le=>pack("H*",$_)).."\""?eg'
```

To merge changes into the Windows Registry of an offline guest virtual machine, you must first prepare a **.REG** file. There is a great deal of documentation about doing this available here. When you have prepared a **.REG** file, enter the following:

```
# virt-win-reg --merge WindowsGuest input.reg
```

This will update the registry in the guest virtual machine.

16.11. USING THE API FROM PROGRAMMING LANGUAGES

The libguestfs API can be used directly from the following languages in Red Hat Enterprise Linux 6.2: C, C++, Perl, Python, Java, Ruby and OCaml.

- To install C and C++ bindings, enter the following command:

  ```
  # yum install libguestfs-devel
  ```

- To install Perl bindings:

  ```
  # yum install 'perl(Sys::Guestfs)'
  ```

- To install Python bindings:

  ```
  # yum install python-libguestfs
  ```

- To install Java bindings:

  ```
  # yum install libguestfs-java libguestfs-java-devel libguestfs-javadoc
  ```

- To install Ruby bindings:

  ```
  # yum install ruby-libguestfs
  ```

- To install OCaml bindings:

  ```
  # yum install ocaml-libguestfs ocaml-libguestfs-devel
  ```

The binding for each language is essentially the same, but with minor syntactic changes. A C statement:

```
guestfs_launch (g);
```

Would appear like the following in Perl:

```
$g->launch ()
```

Or like the following in OCaml:

```
g#launch ()
```

Only the API from C is detailed in this section.

In the C and C++ bindings, you must manually check for errors. In the other bindings, errors are converted into exceptions; the additional error checks shown in the examples below are not necessary for other languages, but conversely you may wish to add code to catch exceptions. Refer to the following list for some points of interest regarding the architecture of the libguestfs API:

- The libguestfs API is synchronous. Each call blocks until it has completed. If you want to make calls asynchronously, you have to create a thread.
- The libguestfs API is not thread safe: each handle should be used only from a single thread, or if you want to share a handle between threads you should implement your own mutex to ensure that two threads cannot execute commands on one handle at the same time.
- You should not open multiple handles on the same disk image. It is permissible if all the handles are read-only, but still not recommended.
- You should not add a disk image for writing if anything else could be using that disk image (eg. a live VM). Doing this will cause disk corruption.
- Opening a read-only handle on a disk image which is currently in use (eg. by a live VM) is possible; however, the results may be unpredictable or inconsistent particularly if the disk image is being heavily written to at the time you are reading it.

16.11.1. Interaction with the API through a C Program

Your C program should start by including the <guestfs.h> header file, and creating a handle:

```
#include <stdio.h>
#include <stdlib.h>
#include <guestfs.h>

int
main (int argc, char *argv[])
{
  guestfs_h *g;

  g = guestfs_create ();
  if (g == NULL) {
    perror ("failed to create libguestfs handle");
    exit (EXIT_FAILURE);
  }

  /* ... */

  guestfs_close (g);

  exit (EXIT_SUCCESS);
}
```

Save this program to a file (**test.c**). Compile this program and run it with the following two commands:

```
gcc -Wall test.c -o test -lguestfs
./test
```

At this stage it should print no output. The rest of this section demonstrates an example showing how to extend this program to create a new disk image, partition it, format it with an ext4 file system, and create

some files in the file system. The disk image will be called **disk.img** and be created in the current directory.

The outline of the program is:

- Create the handle.
- Add disk(s) to the handle.
- Launch the libguestfs back end.
- Create the partition, file system and files.
- Close the handle and exit.

Here is the modified program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <guestfs.h>

 int
 main (int argc, char *argv[])
 {
   guestfs_h *g;
   size_t i;

   g = guestfs_create ();
   if (g == NULL) {
     perror ("failed to create libguestfs handle");
     exit (EXIT_FAILURE);
  }

   /* Create a raw-format sparse disk image, 512 MB in size. */
   int fd = open ("disk.img", O_CREAT|O_WRONLY|O_TRUNC|O_NOCTTY, 0666);
   if (fd == -1) {
     perror ("disk.img");
     exit (EXIT_FAILURE);
   }
   if (ftruncate (fd, 512 * 1024 * 1024) == -1) {
     perror ("disk.img: truncate");
     exit (EXIT_FAILURE);
   }
   if (close (fd) == -1) {
     perror ("disk.img: close");
     exit (EXIT_FAILURE);
   }

   /* Set the trace flag so that we can see each libguestfs call. */
   guestfs_set_trace (g, 1);

   /* Set the autosync flag so that the disk will be synchronized
    * automatically when the libguestfs handle is closed.
```

```
 */
guestfs_set_autosync (g, 1);

/* Add the disk image to libguestfs. */
if (guestfs_add_drive_opts (g, "disk.img",
      GUESTFS_ADD_DRIVE_OPTS_FORMAT, "raw", /* raw format */
      GUESTFS_ADD_DRIVE_OPTS_READONLY, 0,   /* for write */
      -1 /* this marks end of optional arguments */ )
    == -1)
  exit (EXIT_FAILURE);

/* Run the libguestfs back-end. */
if (guestfs_launch (g) == -1)
  exit (EXIT_FAILURE);

/* Get the list of devices.  Because we only added one drive
 * above, we expect that this list should contain a single
 * element.
 */
char **devices = guestfs_list_devices (g);
if (devices == NULL)
  exit (EXIT_FAILURE);
if (devices[0] == NULL || devices[1] != NULL) {
  fprintf (stderr,
           "error: expected a single device from list-devices\n");
  exit (EXIT_FAILURE);
}

/* Partition the disk as one single MBR partition. */
if (guestfs_part_disk (g, devices[0], "mbr") == -1)
  exit (EXIT_FAILURE);

/* Get the list of partitions.  We expect a single element, which
 * is the partition we have just created.
 */
char **partitions = guestfs_list_partitions (g);
if (partitions == NULL)
  exit (EXIT_FAILURE);
if (partitions[0] == NULL || partitions[1] != NULL) {
  fprintf (stderr,
           "error: expected a single partition from list-partitions\n");
  exit (EXIT_FAILURE);
}

/* Create an ext4 filesystem on the partition. */
if (guestfs_mkfs (g, "ext4", partitions[0]) == -1)
  exit (EXIT_FAILURE);

/* Now mount the filesystem so that we can add files. */
if (guestfs_mount_options (g, "", partitions[0], "/") == -1)
  exit (EXIT_FAILURE);

/* Create some files and directories. */
if (guestfs_touch (g, "/empty") == -1)
  exit (EXIT_FAILURE);
```

```
  const char *message = "Hello, world\n";
  if (guestfs_write (g, "/hello", message, strlen (message)) == -1)
    exit (EXIT_FAILURE);

  if (guestfs_mkdir (g, "/foo") == -1)
    exit (EXIT_FAILURE);

  /* This uploads the local file /etc/resolv.conf into the disk image. */
  if (guestfs_upload (g, "/etc/resolv.conf", "/foo/resolv.conf") == -1)
    exit (EXIT_FAILURE);

  /* Because 'autosync' was set (above) we can just close the handle
   * and the disk contents will be synchronized.  You can also do
   * this manually by calling guestfs_umount_all and guestfs_sync.
   */
  guestfs_close (g);

  /* Free up the lists. */
  for (i = 0; devices[i] != NULL; ++i)
    free (devices[i]);
  free (devices);
  for (i = 0; partitions[i] != NULL; ++i)
    free (partitions[i]);
  free (partitions);

  exit (EXIT_SUCCESS);
}
```

Compile and run this program with the following two commands:

```
gcc -Wall test.c -o test -lguestfs
./test
```

If the program runs to completion successfully then you should be left with a disk image called **disk.img**, which you can examine with guestfish:

```
guestfish --ro -a disk.img -m /dev/sda1
><fs> ll /
><fs> cat /foo/resolv.conf
```

By default (for C and C++ bindings only), libguestfs prints errors to stderr. You can change this behavior by setting an error handler. The guestfs(3) man page discusses this in detail.

16.12. VIRT-SYSPREP: RESETTING VIRTUAL MACHINE SETTINGS

The **virt-sysprep** command line tool can be used to reset or unconfigure a guest virtual machine so that clones can be made from it. This process involves removing SSH host keys, persistent network MAC configuration, and user accounts. **virt-sysprep** can also customize a virtual machine, for instance by adding SSH keys, users or logos. Each step can be enabled or disabled as required.

The term "sysprep" is derived from the System Preparation tool (sysprep.exe) which is used with the Microsoft Windows systems. Despite this, the tool does not currently work on Windows guests.

NOTE

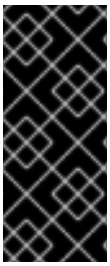
libguestfs and guestfish do not require root privileges. You only need to run them as root if the disk image being accessed needs root access to read or write or both.

The **virt-sysprep** tool is part of the libguestfs-tools-c package, which is installed with the following command:

```
$ yum install libguestfs-tools-c
```

Alternatively, just the **virt-sysprep** tool can be installed with the following command:

```
$ yum install /usr/bin/virt-sysprep
```



IMPORTANT

virt-sysprep modifies the guest or disk image in place. To use **virt-sysprep**, the guest virtual machine must be offline, so you must shut it down before running the commands. To preserve the existing contents of the guest virtual machine, you must snapshot, copy or clone the disk first. Refer to libguestfs.org for more information on copying and cloning disks.

The following commands are available to use with **virt-sysprep**:

Table 16.1. virt-sysprep commands

Command	Description	Example
--help	Displays a brief help entry about a particular command or about the whole package. For additional help, see the virt-sysprep man page.	$ **virt-sysprep --help**
-a [*file*] or --add [*file*]	Adds the specified *file*, which should be a disk image from a guest virtual machine. The format of the disk image is auto-detected. To override this and force a particular format, use the **--format** option.	$ **virt-sysprep --add /dev/vms/disk.img**
-c [*URI*] or --connect [*URI*]	Connects to the given URI, if using libvirt. If omitted, it will connect via the KVM hypervisor. If you specify guest block devices directly (**virt-sysprep -a**), then libvirt is not used at all.	$ **virt-sysprep -c qemu:///system**

Command	Description	Example
-d [*guest*] or --domain [*guest*]	Adds all the disks from the specified guest virtual machine. Domain UUIDs can be used instead of domain names.	$ **virt-sysprep --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-n or --dry-run or --dryrun	Performs a read-only "dry run" sysprep operation on the guest virtual machine. This runs the sysprep operation, but throws away any changes to the disk at the end.	$ **virt-sysprep -n**
--enable [*operations*]	Enables the specified *operations*. To list the possible operations, use the --list command.	$ **virt-sysprep --enable ssh-hotkeys,udev-persistent-net**
--format [**raw**|**qcow2**|**auto**]	The default for the -a option is to auto-detect the format of the disk image. Using this forces the disk format for -a options which follow on the command line. Using --format auto switches back to auto-detection for subsequent -a options (see the -a command above).	$ **virt-sysprep --format raw -a *disk.img*** forces raw format (no auto-detection) for disk.img, but **virt-sysprep --format raw -a *disk.img* --format auto -a another.img** forces raw format (no auto-detection) for **disk.img** and reverts to auto-detection for **another.img**. If you have untrusted raw-format guest disk images, you should use this option to specify the disk format. This avoids a possible security problem with malicious guests.
--list-operations	Lists the operations supported by the virt-sysprep program. These are listed one per line, with one or more single-space-separated fields. The first field in the output is the operation name, which can be supplied to the **--enable** flag. The second field is a * character if the operation is enabled by default, or is blank if not. Additional fields on the same line include a description of the operation.	$ **virt-sysprep --list-operations**

Command	Description	Example
--mount-options	Sets the mount options for each mount point in the guest virtual machine. Use a semicolon-separated list of mountpoint:options pairs. You may need to place quotes around this list to protect it from the shell.	$ **virt-sysprep --mount-options "/:notime"** will mount the root directory with the **notime** operation.
--selinux-relabel and --no-selinux-relabel	virt-sysprep does not always schedule a SELinux relabelling at the first boot of the guest. In some cases, a relabel is performed (for example, when virt-sysprep has modified files). However, when all operations only remove files (for example, when using **--enable delete --delete /some/file**) no relabelling is scheduled. Using the **--selinux-relabel** option always forces SELinux relabelling, while with **--no-selinux-relabel** set, relabelling is never scheduled. It is recommended to use **--selinux-relabel** to ensure that files have the correct SELinux labels.	$ **virt-sysprep --selinux-relabel**
-q or --quiet	Prevents the printing of log messages.	$ **virt-sysprep -q**
-v or --verbose	Enables verbose messages for debugging purposes.	$ **virt-sysprep -v**
-V or --version	Displays the virt-sysprep version number and exits.	$ **virt-sysprep -V**
--root-password	Sets the root password. Can either be used to specify the new password explicitly, or to use the string from the first line of a selected file, which is more secure.	$ **virt-sysprep --root-password password:***123456* **-a** *guest.img* or $ **virt-sysprep --root-password file:***SOURCE_FILE_PATH* **-a** *guest.img*

For more information, refer to the libguestfs documentation.

16.13. TROUBLESHOOTING

A test tool is available to check that libguestfs is working. Run the following command after installing libguestfs (root access not required) to test for normal operation:

```
$ libguestfs-test-tool
```

This tool prints a large amount of text to test the operation of libguestfs. If the test is successful, the following text will appear near the end of the output:

```
===== TEST FINISHED OK =====
```

16.14. WHERE TO FIND FURTHER DOCUMENTATION

The primary source for documentation for libguestfs and the tools are the Unix man pages. The API is documented in guestfs(3). guestfish is documented in guestfish(1). The virt tools are documented in their own man pages (eg. virt-df(1)).

CHAPTER 17. GRAPHICAL USER INTERFACE TOOLS FOR GUEST VIRTUAL MACHINE MANAGEMENT

In addition to **virt-manager**, Red Hat Enterprise Linux 6 provides the following tools that enable you to access your guest virtual machine's console.

17.1. VIRT-VIEWER

virt-viewer is a minimalistic command-line utility for displaying the graphical console of a guest virtual machine. The console is accessed using the VNC or SPICE protocol. The guest can be referred to by its name, ID, or UUID. If the guest is not already running, the viewer can be set to wait until is starts before attempting to connect to the console. The viewer can connect to remote hosts to get the console information and then also connect to the remote console using the same network transport.

In comparison with *virt-manager*, *virt-viewer* offers a smaller set of features, but is less resource-demanding. In addition, unlike *virt-manager*, *virt-viewer* in most cases does not require read-write permissions to libvirt. Therefore, it can be used by non-privileged users who should be able to connect to and display guests, but not to configure them.

To install the virt-viewer utility, run:

```
# sudo yum install virt-viewer
```

Syntax

The basic *virt-viewer* command-line syntax is as follows:

```
# virt-viewer [OPTIONS] {guest-name|id|uuid}
```

The basic *virt-viewer* command-line syntax is as follows:

Connecting to a guest virtual machine

If used without any options, *virt-viewer* lists guests that it can connect to on the default hypervisor of the local system.

To connect to a guest virtual machine that uses the default hypervisor:

```
# virt-viewer guest-name-or-UUID
```

To connect to a guest virtual machine that uses the KVM-QEMU hypervisor:

```
# virt-viewer --connect qemu:///system guest-name-or-UUID
```

To connect to a remote console using TLS:

```
# virt-viewer --connect xen://example.org/ guest-name-or-UUID
```

To connect to a console on a remote host by using SSH, look up the guest configuration and then make a direct non-tunneled connection to the console:

```
# virt-viewer --direct --connect xen+ssh://root@example.org/ guest-name-or-UUID
```

Interface

By default, the *virt-viewer* interface provides only the basic tools for interacting with the guest:

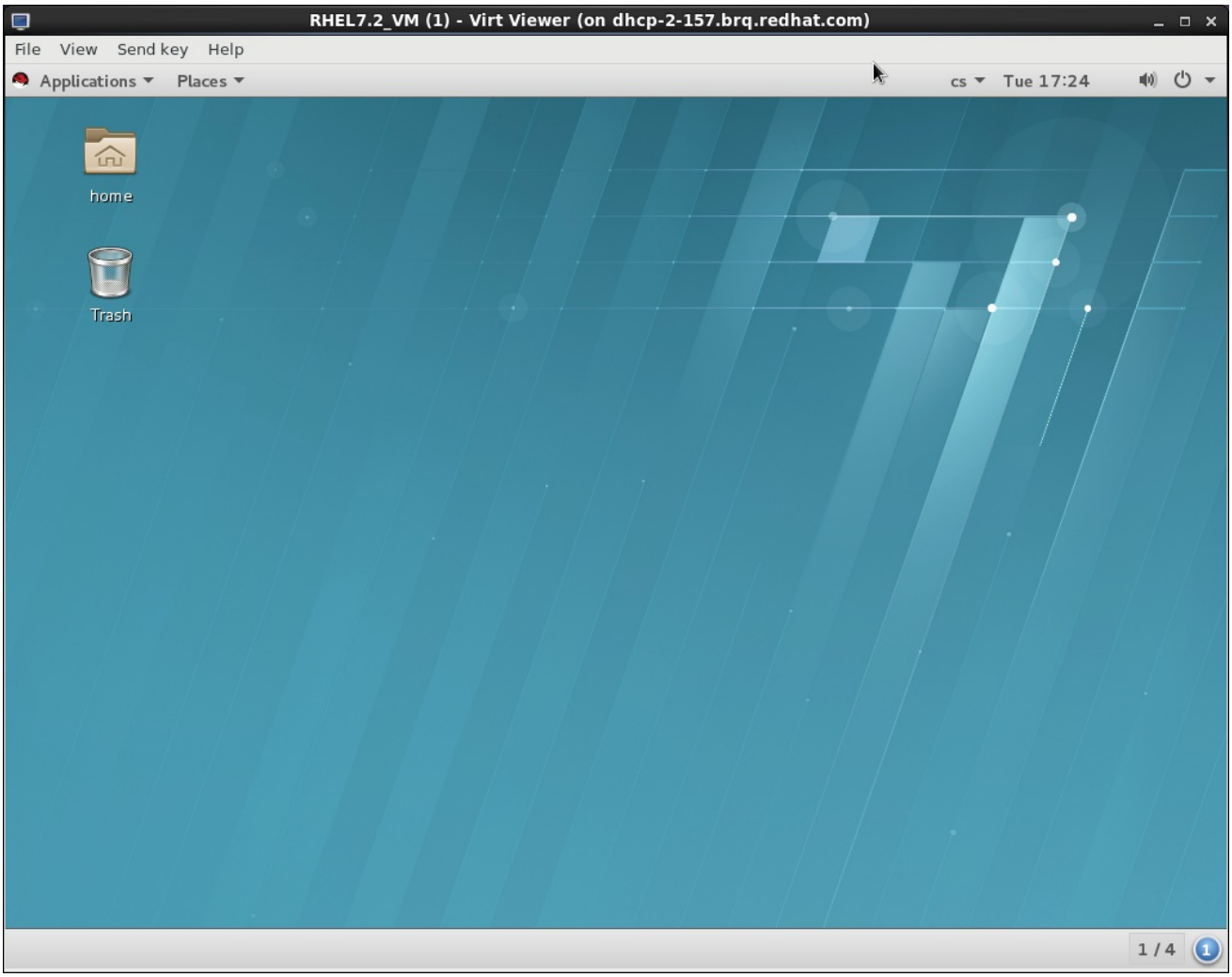


Figure 17.1. Sample *virt-viewer* interface

Setting hotkeys

To create a customized keyboard shortcut (also referred to as a hotkey) for the *virt-viewer* session, use the **--hotkeys** option:

```
# virt-viewer --hotkeys=action1=key-combination1[,action2=key-combination2] guest-name-or-UUID
```

The following actions can be assigned to a hotkey:

- toggle-fullscreen
- release-cursor
- smartcard-insert
- smartcard-remove

Key-name combination hotkeys are not case-sensitive. Note that the hotkey setting does not carry over to future *virt-viewer* sessions.

Example 17.1. Setting a *virt-viewer* hotkey

> To add a hotkey to change to full screen mode when connecting to a KVM-QEMU guest called testguest:
>
> ```
> # virt-viewer --hotkeys=toggle-fullscreen=shift+f11 qemu:///system testguest
> ```

Kiosk mode

In kiosk mode, *virt-viewer* only allows the user to interact with the connected desktop, and does not provide any options to interact with the guest settings or the host system unless the guest is shut down. This can be useful for example when an administrator wants to restrict a user's range of actions to a specified guest.

To use kiosk mode, connect to a guest with the **-k** or **--kiosk** option.

> **Example 17.2. Using *virt-viewer* in kiosk mode**
>
> To connect to a KVM-QEMU virtual machine in kiosk mode that terminates after the machine is shut down, use the following command:
>
> ```
> # virt-viewer --connect qemu:///system guest-name-or-UUID --kiosk --kiosk-quit on-disconnect
> ```

Note, however, that kiosk mode alone cannot ensure that the user does not interact with the host system or the guest settings after the guest is shut down. This would require further security measures, such as disabling the window manager on the host.

17.2. REMOTE-VIEWER

The **remote-viewer** is a simple remote desktop display client that supports SPICE and VNC. It shares most of the features and limitations with **virt-viewer**.

However, unlike virt-viewer, *remote-viewer* does not require libvirt to connect to the remote guest display. As such, *remote-viewer* can be used for example to connect to a virtual machine on a remote host that does not provide permissions to interact with libvirt or to use SSH connections.

To install the **remote-viewer** utility, run:

```
# sudo yum install virt-viewer
```

Syntax

The basic *remote-viewer* command-line syntax is as follows:

```
# remote-viewer [OPTIONS] {guest-name|id|uuid}
```

To see the full list of options available for use with remote-viewer, use **man remote-viewer**.

Connecting to a guest virtual machine

If used without any options, *remote-viewer* lists guests that it can connect to on the default URI of the local system.

To connect to a specific guest using remote-viewer, use the VNC/SPICE URI. For information about obtaining the URI, see Section 14.5.19, “Displaying a URI for Connection to a Graphical Display” .

Example 17.3. Connecting to a guest display using SPICE

Use the following to connect to a SPICE server on a machine called "testguest" that uses port 5900 for SPICE communication:

```
# remote-viewer spice://testguest:5900
```

Example 17.4. Connecting to a guest display using VNC

Use the following to connect to a VNC server on a machine called **testguest2** that uses port 5900 for VNC communication:

```
# remote-viewer vnc://testguest2:5900
```

Interface

By default, the *remote-viewer* interface provides only the basic tools for interacting with the guest:

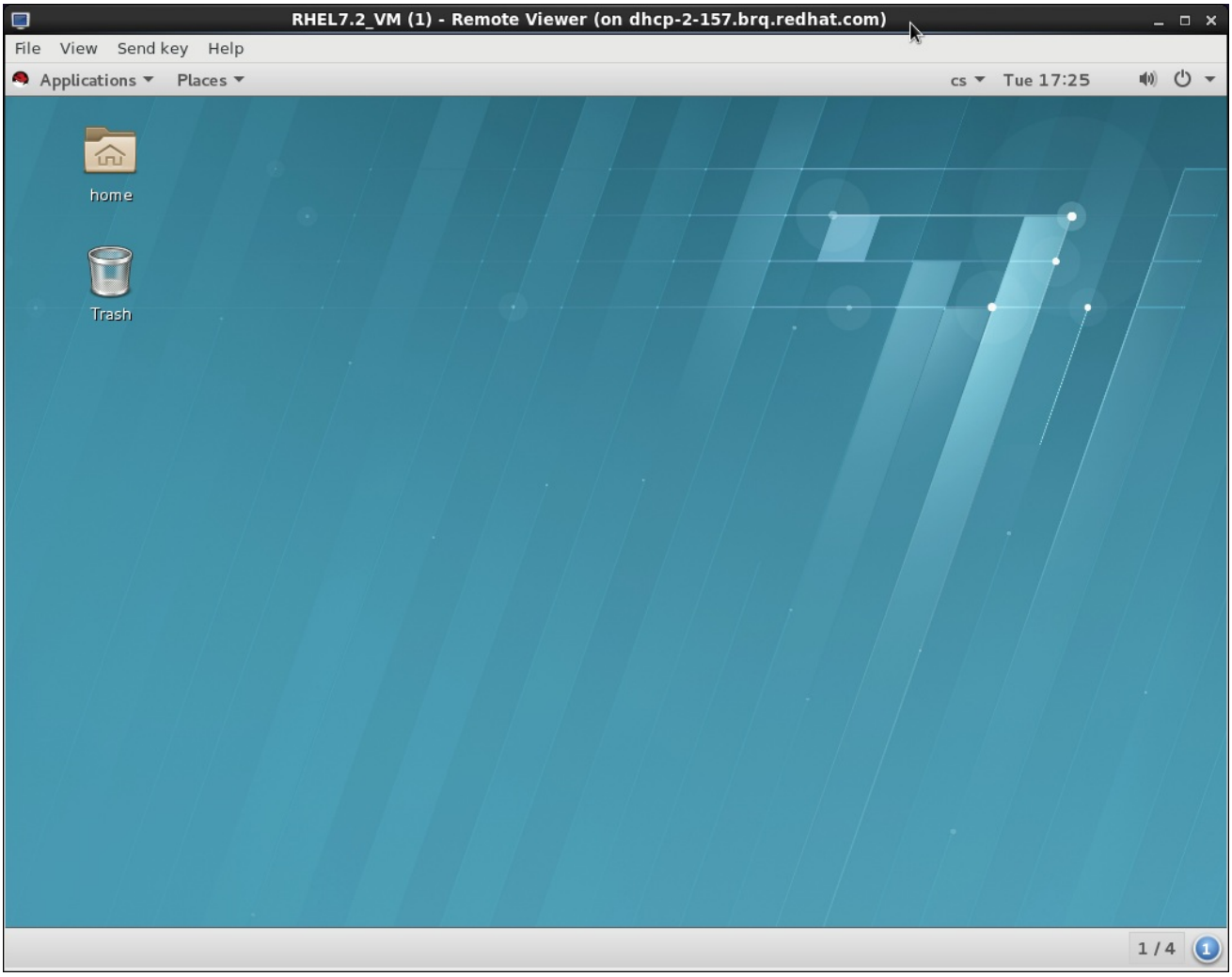


Figure 17.2. Sample *remote-viewer* interface

CHAPTER 18. VIRTUAL NETWORKING

This chapter introduces the concepts needed to create, start, stop, remove, and modify virtual networks with libvirt.

Additional information can be found in the libvirt reference chapter

18.1. VIRTUAL NETWORK SWITCHES

Libvirt virtual networking uses the concept of a *virtual network switch*. A virtual network switch is a software construct that operates on a host physical machine server, to which virtual machines (guests) connect. The network traffic for a guest is directed through this switch:

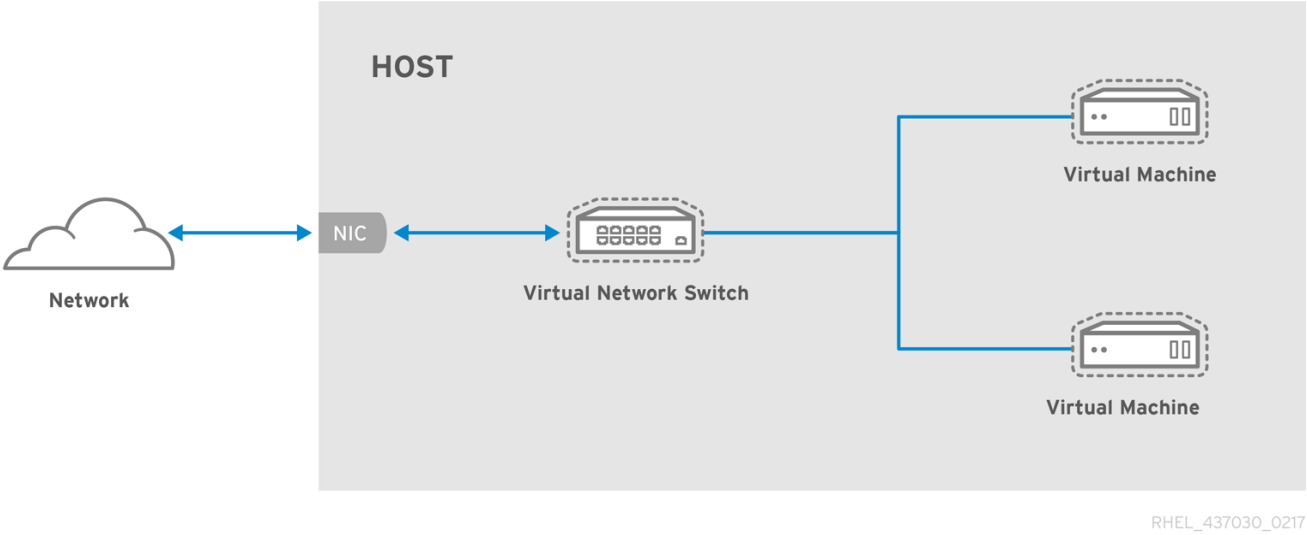


Figure 18.1. Virtual network switch with two guests

Linux host physical machine servers represent a virtual network switch as a network interface. When the libvirtd daemon (**libvirtd**) is first installed and started, the default network interface representing the virtual network switch is **virbr0**.

This **virbr0** interface can be viewed with the **ip** command like any other interface:

```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

18.2. BRIDGED MODE

When using *Bridged mode*, all of the guest virtual machines appear within the same subnet as the host physical machine. All other physical machines on the same physical network are aware of the virtual machines, and can access the virtual machines. Bridging operates on Layer 2 of the OSI networking model.

It is possible to use multiple physical interfaces on the hypervisor by joining them together with a bond. The bond is then added to a bridge and then guest virtual machines are added onto the bridge as well. However, the bonding driver has several modes of operation, and only a few of these modes work with a bridge where virtual guest machines are in use.

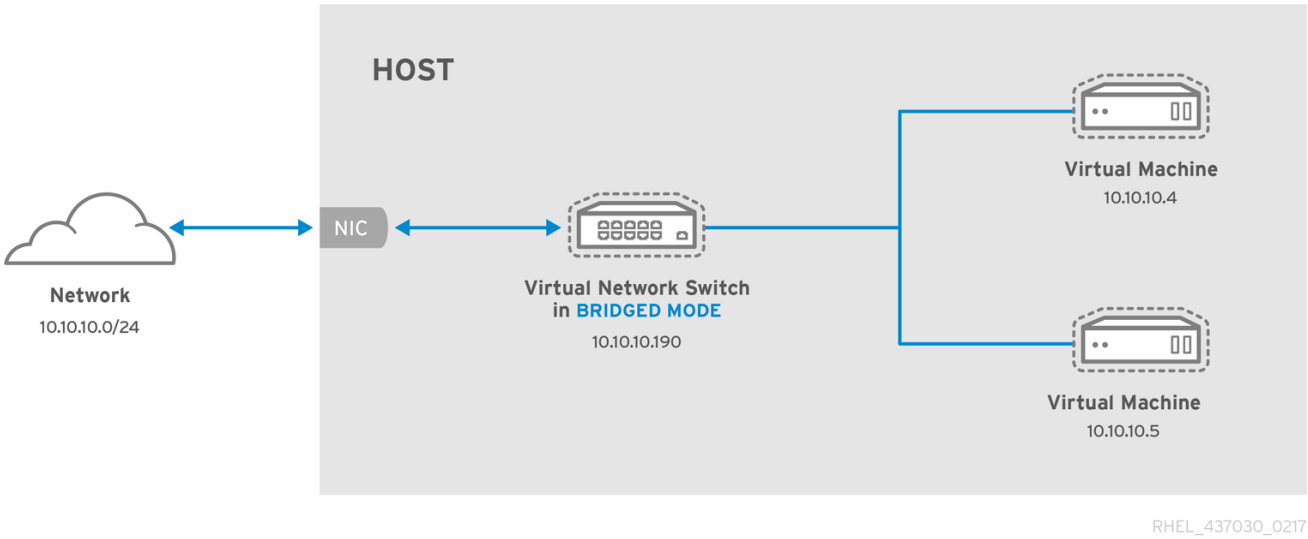


Figure 18.2. Virtual network switch in bridged mode

> **WARNING**
>
> The only bonding modes that should be used with a guest virtual machine are Mode 1, Mode 2, and Mode 4. Under no circumstances should Modes 0, 3, 5, or 6 be used. It should also be noted that mii-monitoring should be used to monitor bonding modes as arp-monitoring does not work.

For more information on bonding modes, refer to the knowledgebase article on bonding modes, or The Red Hat Enterprise Linux 6 Deployment Guide.

For a detailed explanation of bridge_opts parameters, see the *Red Hat Virtualization Administration Guide*.

18.3. NETWORK ADDRESS TRANSLATION MODE

By default, virtual network switches operate in NAT mode. They use IP masquerading rather than SNAT (Source-NAT) or DNAT (Destination-NAT). IP masquerading enables connected guests to use the host physical machine IP address for communication to any external network. By default, computers that are placed externally to the host physical machine cannot communicate to the guests inside when the virtual network switch is operating in NAT mode, as shown in the following diagram:

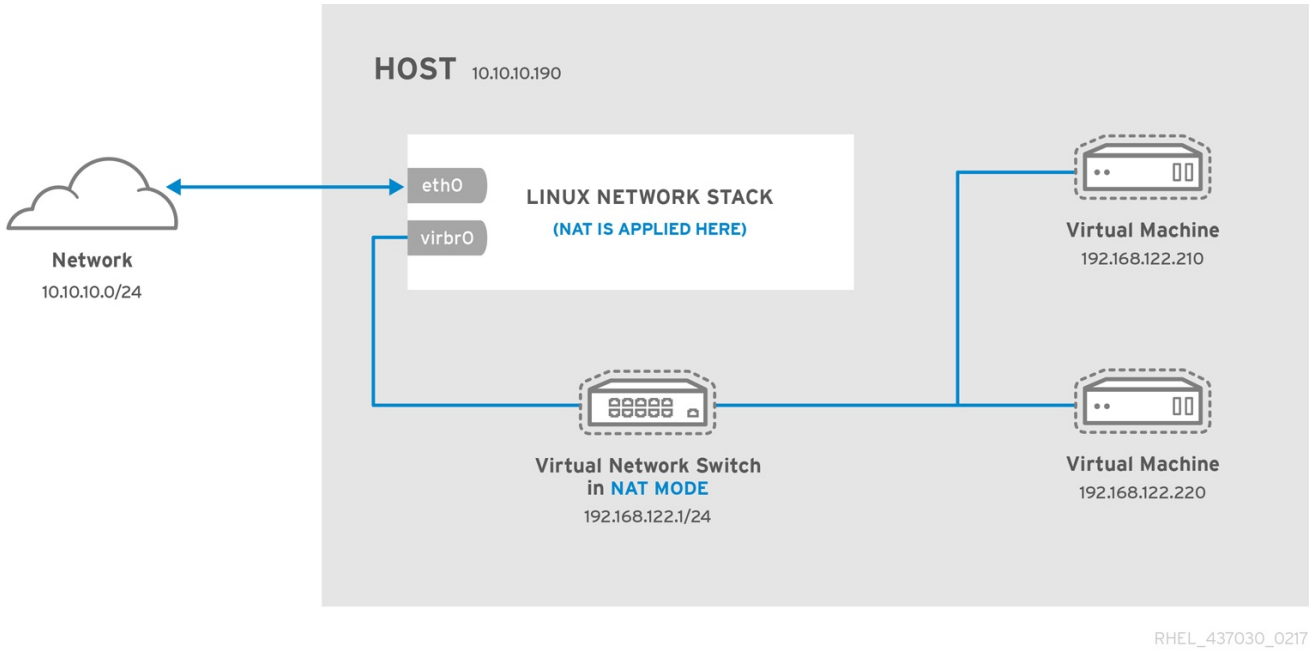


Figure 18.3. Virtual network switch using NAT with two guests

> **WARNING**
>
> Virtual network switches use NAT configured by iptables rules. Editing these rules while the switch is running is not recommended, as incorrect rules may result in the switch being unable to communicate.

If the switch is not running, you can set th public IP range for forward mode NAT in order to create a port masquerading range by running:

```
# iptables -j SNAT --to-source [start]-[end]
```

18.3.1. DNS and DHCP

IP information can be assigned to guests via DHCP. A pool of addresses can be assigned to a virtual network switch for this purpose. Libvirt uses the **dnsmasq** program for this. An instance of dnsmasq is automatically configured and started by libvirt for each virtual network switch that needs it.

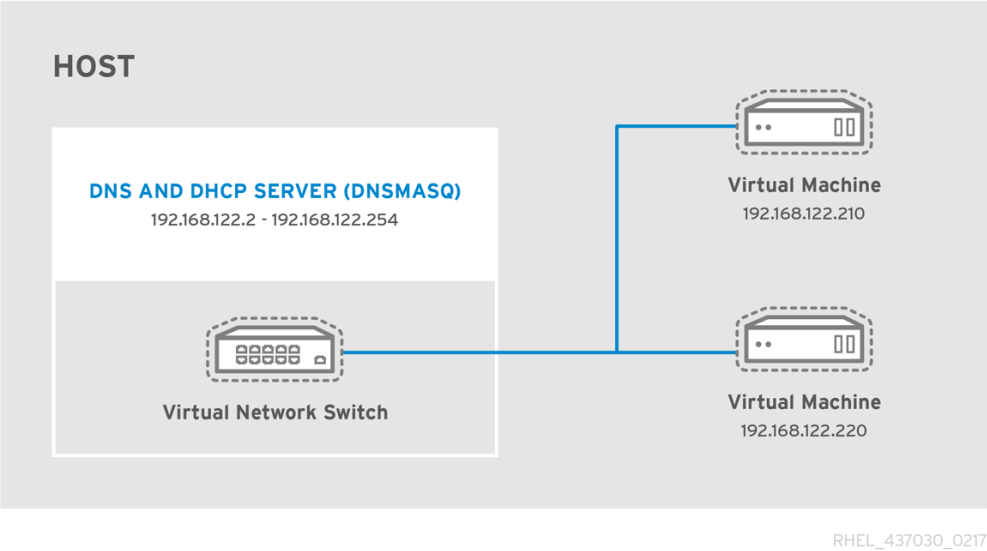


Figure 18.4. Virtual network switch running dnsmasq

18.4. ROUTED MODE

When using *Routed mode*, the virtual switch connects to the physical LAN connected to the host physical machine, passing traffic back and forth without the use of NAT. The virtual switch can examine all traffic and use the information contained within the network packets to make routing decisions. When using this mode, all of the virtual machines are in their own subnet, routed through a virtual switch. This situation is not always ideal as no other host physical machines on the physical network are aware of the virtual machines without manual physical router configuration, and cannot access the virtual machines. Routed mode operates at Layer 3 of the OSI networking model.

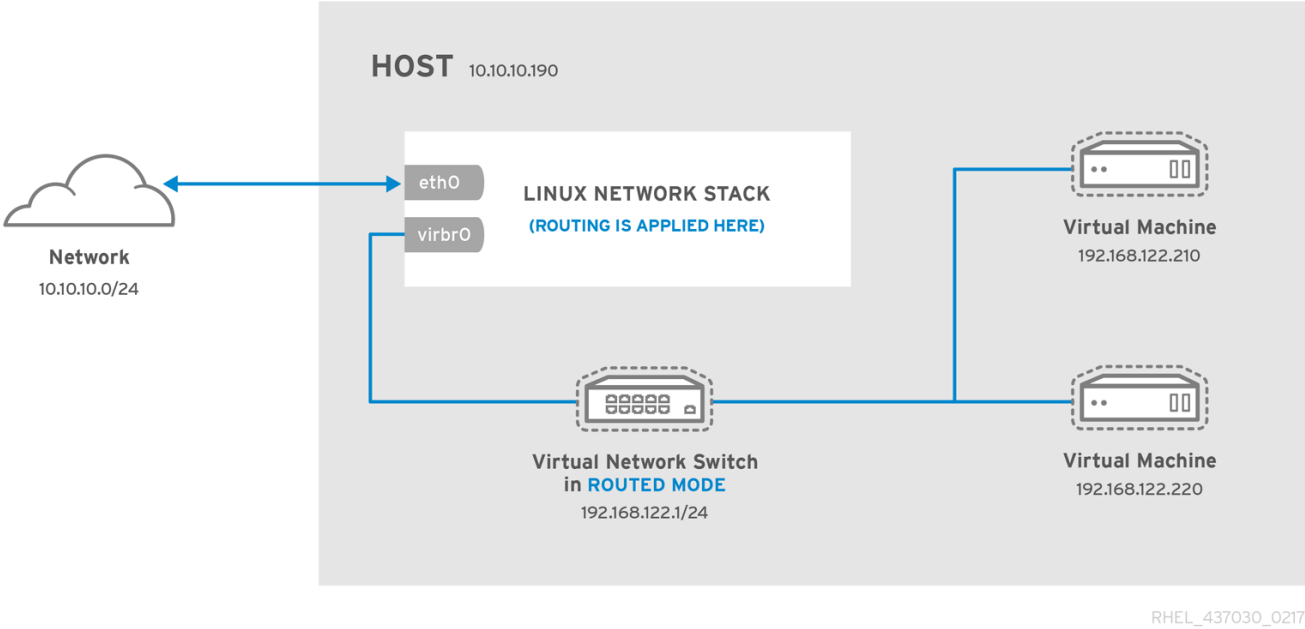


Figure 18.5. Virtual network switch in routed mode

18.5. ISOLATED MODE

When using *Isolated mode*, guests connected to the virtual switch can communicate with each other, and with the host physical machine, but their traffic will not pass outside of the host physical machine, nor can they receive traffic from outside the host physical machine. Using dnsmasq in this mode is required

for basic functionality such as DHCP. However, even if this network is isolated from any physical network, DNS names are still resolved. Therefore a situation can arise when DNS names resolve but ICMP echo request (ping) commands fail.

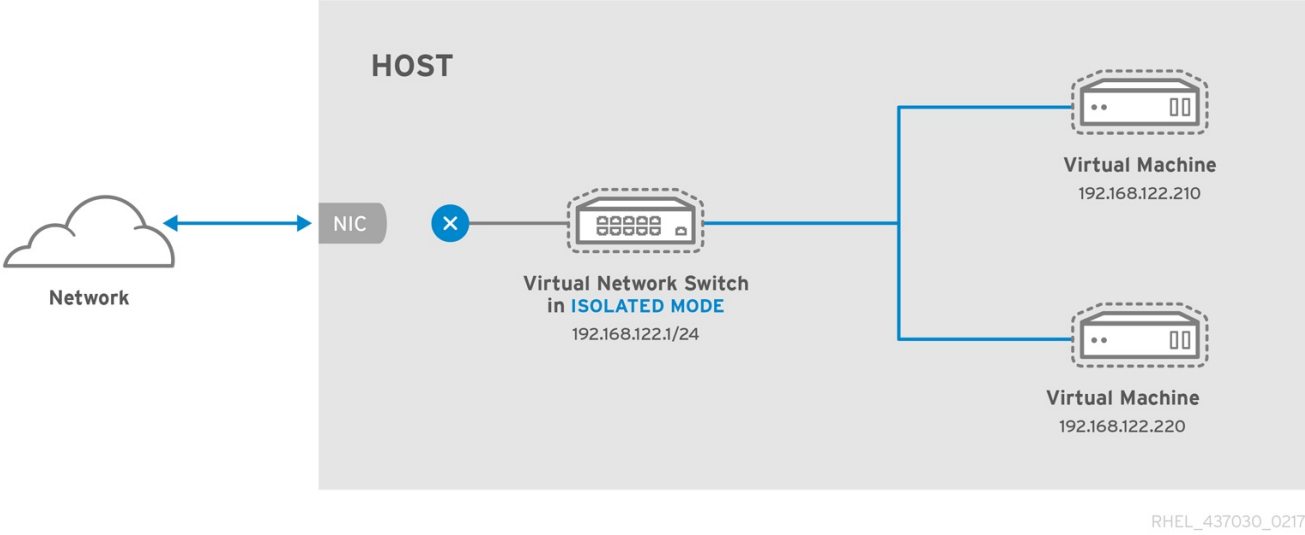


Figure 18.6. Virtual network switch in isolated mode

18.6. THE DEFAULT CONFIGURATION

When the libvirtd daemon (**libvirtd**) is first installed, it contains an initial virtual network switch configuration in NAT mode. This configuration is used so that installed guests can communicate to the external network, through the host physical machine. The following image demonstrates this default configuration for **libvirtd**:

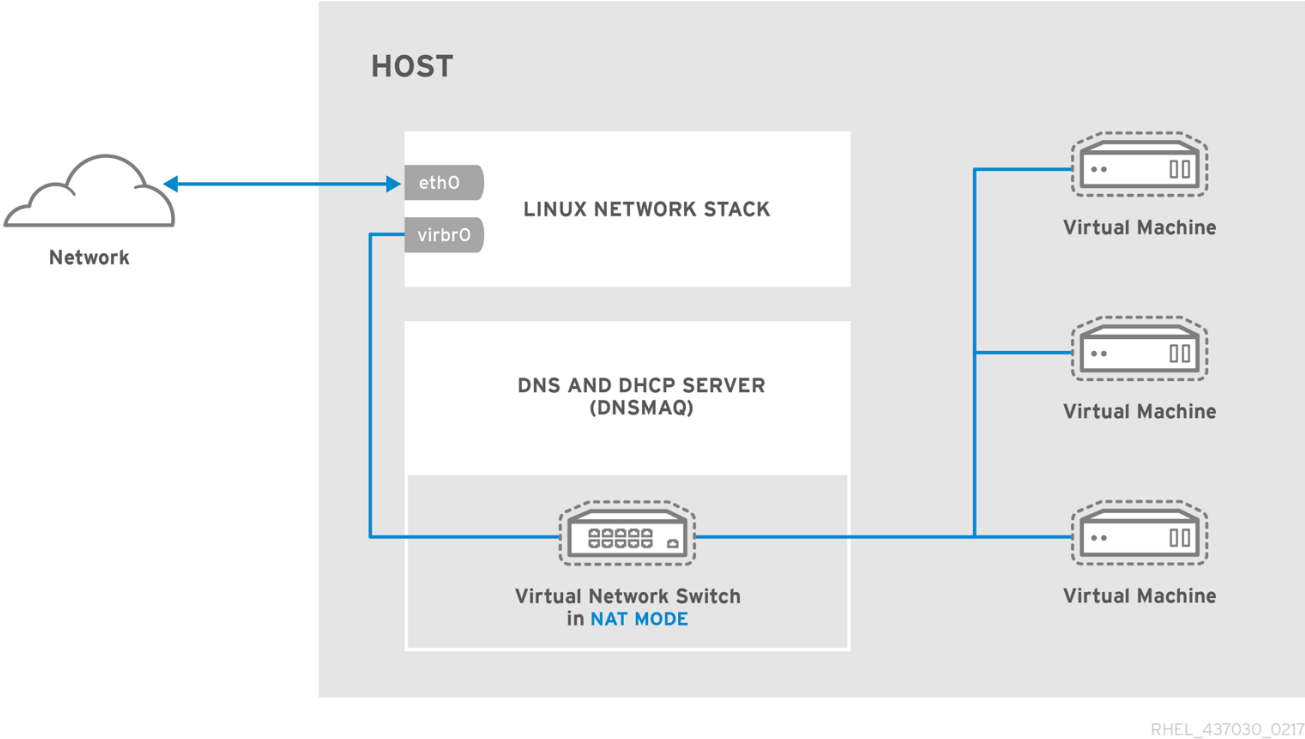


Figure 18.7. Default libvirt network configuration

NOTE

A virtual network can be restricted to a specific physical interface. This may be useful on a physical system that has several interfaces (for example, **eth0**, **eth1** and **eth2**). This is only useful in routed and NAT modes, and can be defined in the **dev=<interface>** option, or in **virt-manager** when creating a new virtual network.

18.7. EXAMPLES OF COMMON SCENARIOS

This section demonstrates different virtual networking modes and provides some example scenarios.

18.7.1. Bridged Mode

Bridged mode operates on Layer 2 of the OSI model. When used, all of the guest virtual machines will appear on the same subnet as the host physical machine. The most common use cases for bridged mode include:

- Deploying guest virtual machines in an existing network alongside host physical machines making the difference between virtual and physical machines transparent to the end user.
- Deploying guest virtual machines without making any changes to existing physical network configuration settings.
- Deploying guest virtual machines which must be easily accessible to an existing physical network. Placing guest virtual machines on a physical network where they must access services within an existing broadcast domain, such as DHCP.
- Connecting guest virtual machines to an existing network where VLANs are used.

18.7.2. Routed Mode

This section provides information about routed mode.

DMZ

Consider a network where one or more nodes are placed in a controlled subnetwork for security reasons. The deployment of a special subnetwork such as this is a common practice, and the subnetwork is known as a DMZ. Refer to the following diagram for more details on this layout:

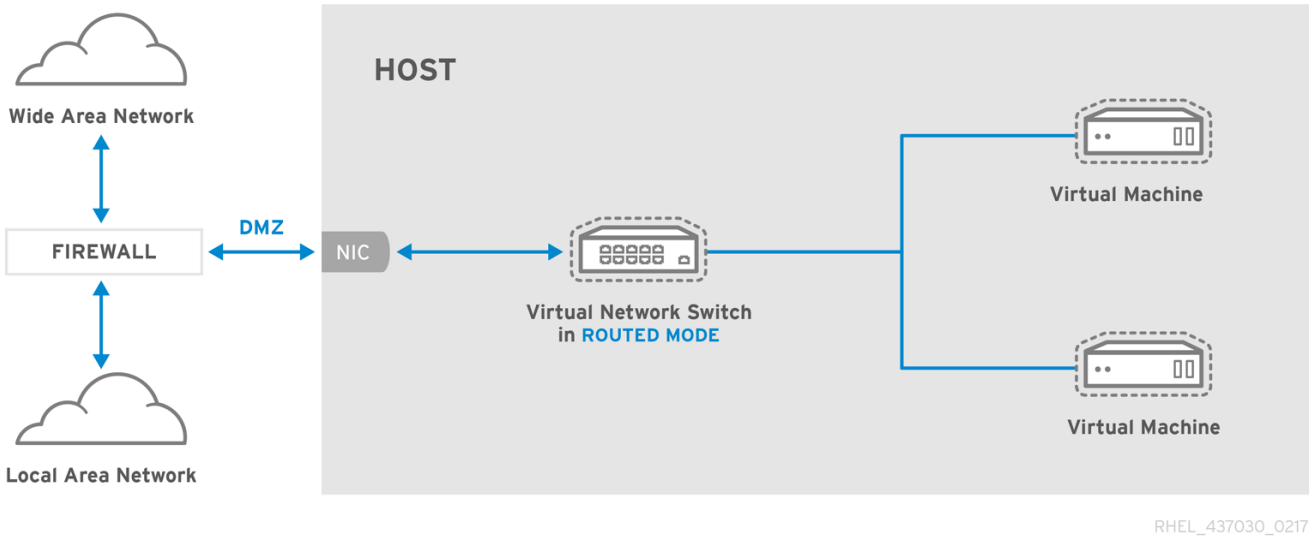


Figure 18.8. Sample DMZ configuration

Host physical machines in a DMZ typically provide services to WAN (external) host physical machines as well as LAN (internal) host physical machines. As this requires them to be accessible from multiple locations, and considering that these locations are controlled and operated in different ways based on their security and trust level, routed mode is the best configuration for this environment.

Virtual Server Hosting

Consider a virtual server hosting company that has several host physical machines, each with two physical network connections. One interface is used for management and accounting, the other is for the virtual machines to connect through. Each guest has its own public IP address, but the host physical machines use private IP address as management of the guests can only be performed by internal administrators. Refer to the following diagram to understand this scenario:

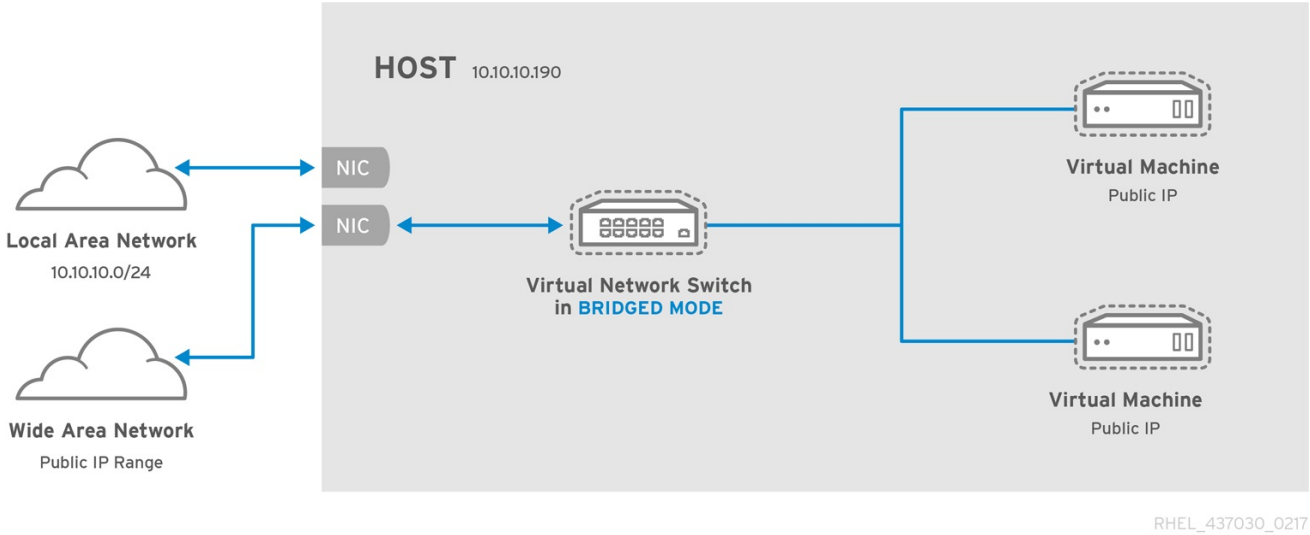


Figure 18.9. Virtual server hosting sample configuration

18.7.3. NAT Mode

NAT (Network Address Translation) mode is the default mode. It can be used for testing when there is no need for direct network visibility.

18.7.4. Isolated Mode

Isolated mode allows virtual machines to communicate with each other only. They are unable to interact with the physical network.

18.8. MANAGING A VIRTUAL NETWORK

To configure a virtual network on your system:

1. From the **Edit** menu, select **Connection Details**.

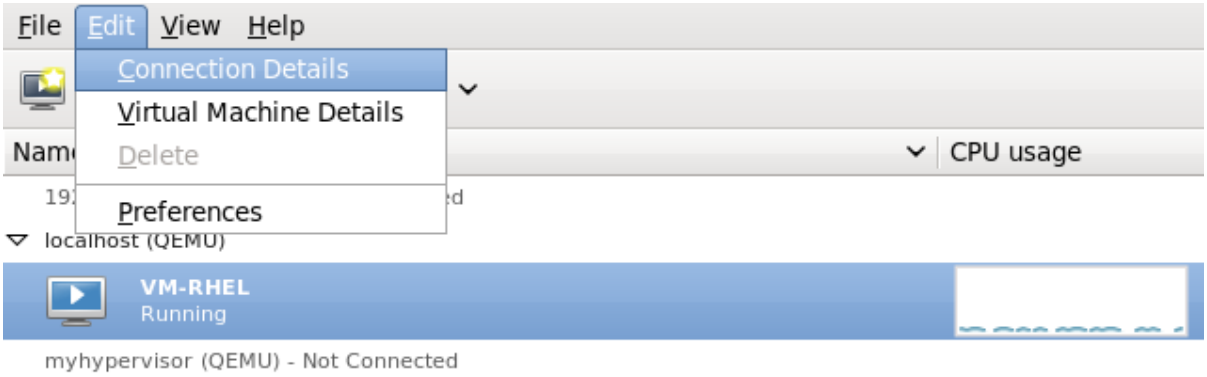


Figure 18.10. Selecting a host physical machine's details

2. This will open the **Connection Details** menu. Click the **Virtual Networks** tab.

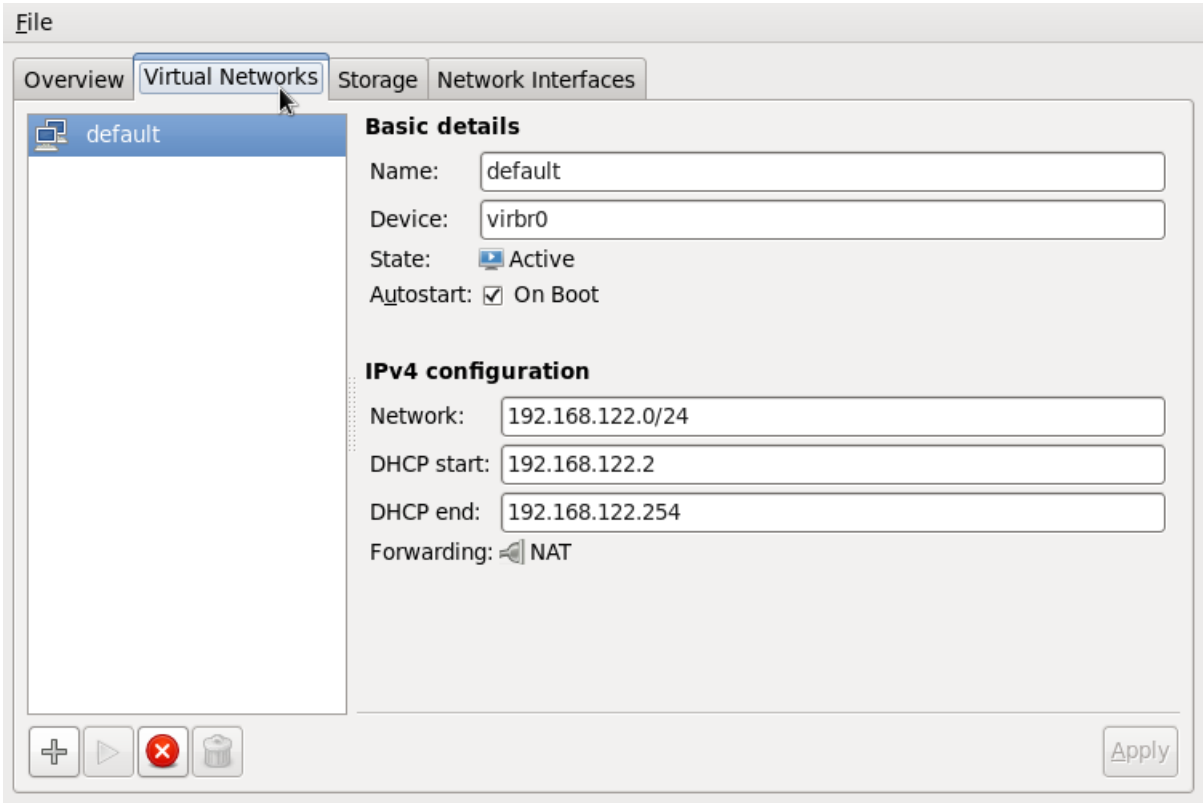


Figure 18.11. Virtual network configuration

3. All available virtual networks are listed on the left-hand box of the menu. You can edit the configuration of a virtual network by selecting it from this box and editing as you see fit.

18.9. CREATING A VIRTUAL NETWORK

To create a virtual network on your system:

1. Open the **Virtual Networks** tab from within the **Connection Details** menu. Click the **Add Network** button, identified by a plus sign (+) icon. For more information, refer to Section 18.8, "Managing a Virtual Network".

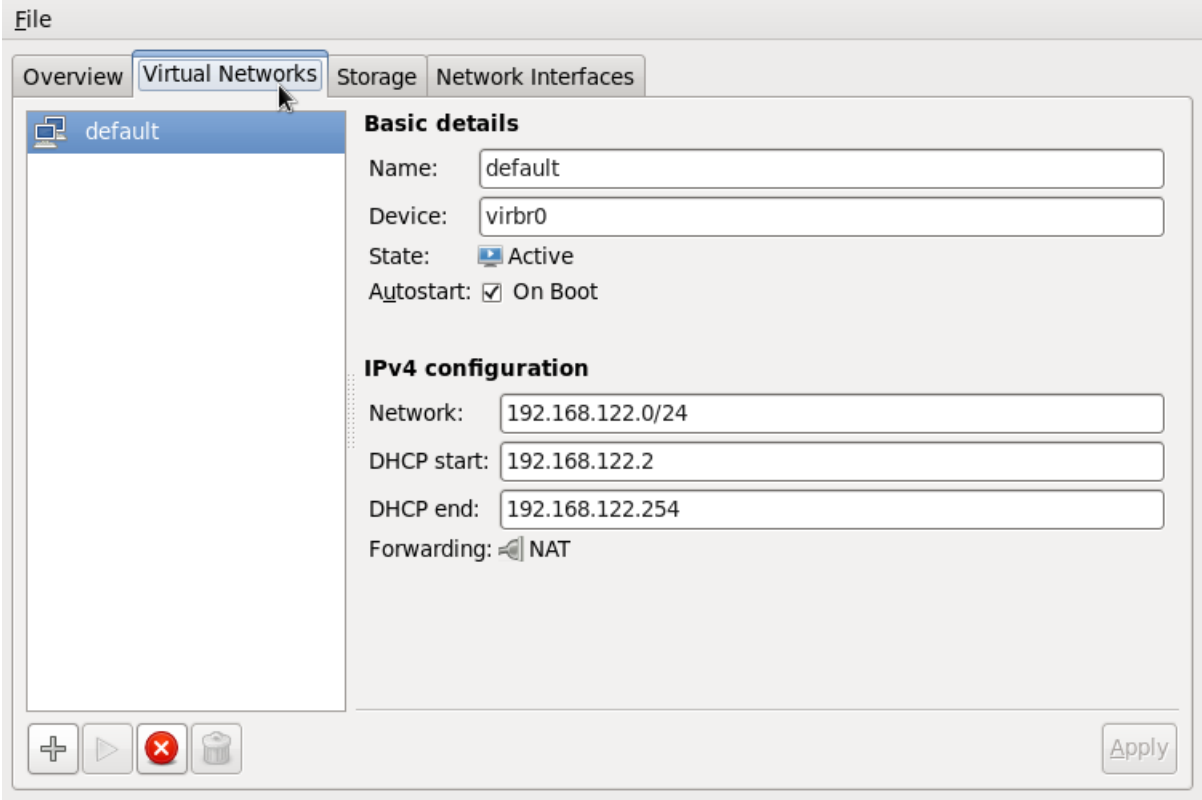


Figure 18.12. Virtual network configuration

This will open the **Create a new virtual network** window. Click **Forward** to continue.

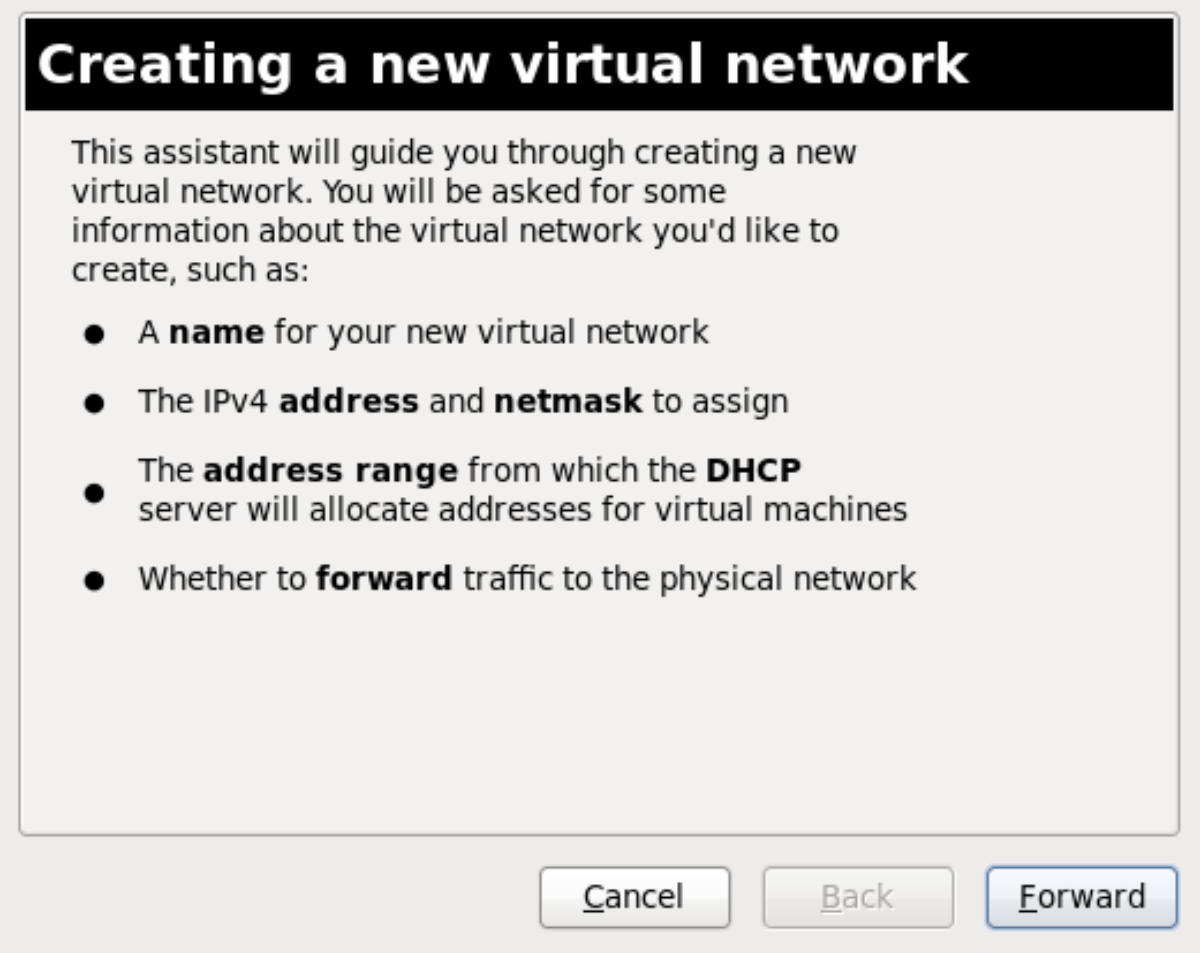


Figure 18.13. Creating a new virtual network

2. Enter an appropriate name for your virtual network and click **Forward**.

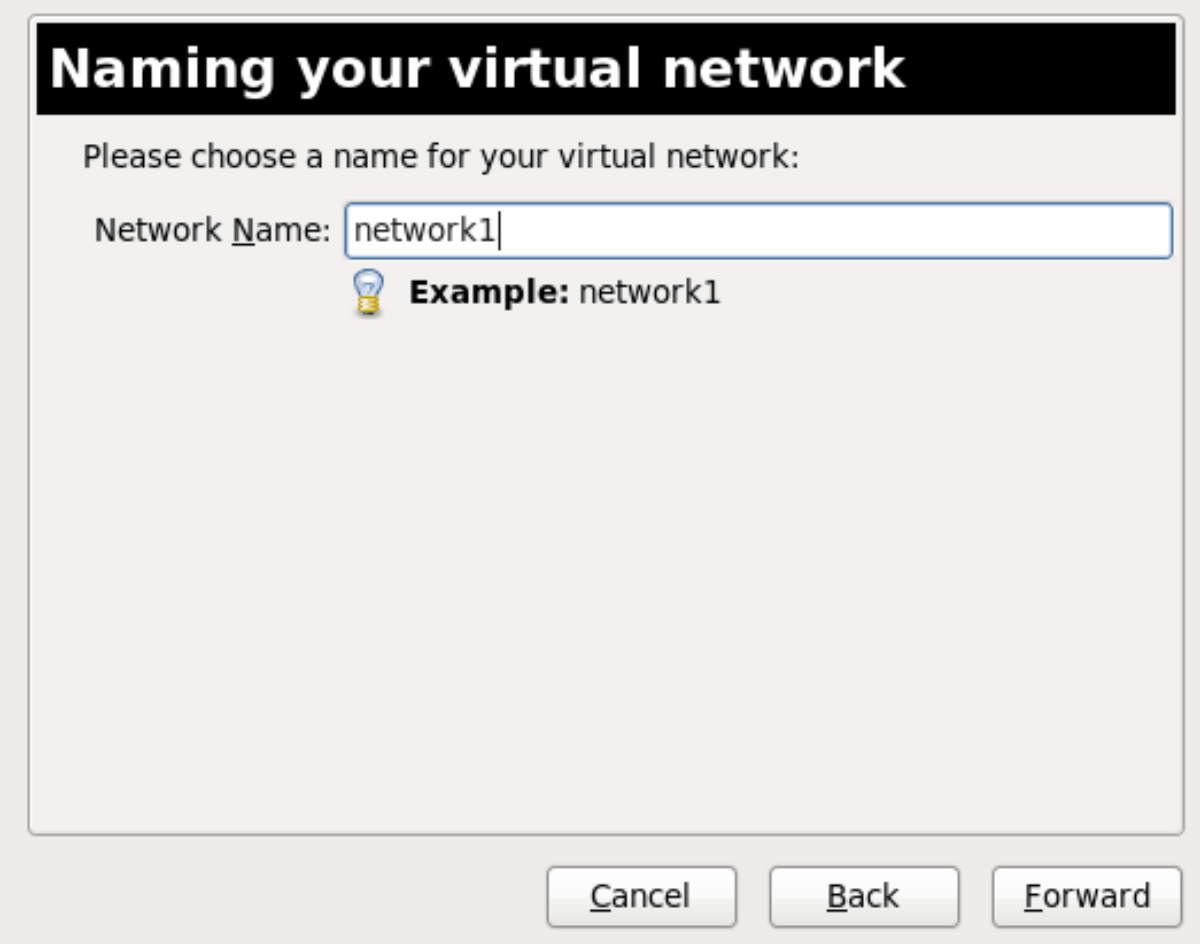


Figure 18.14. Naming your virtual network

3. Enter an IPv4 address space for your virtual network and click **Forward**.

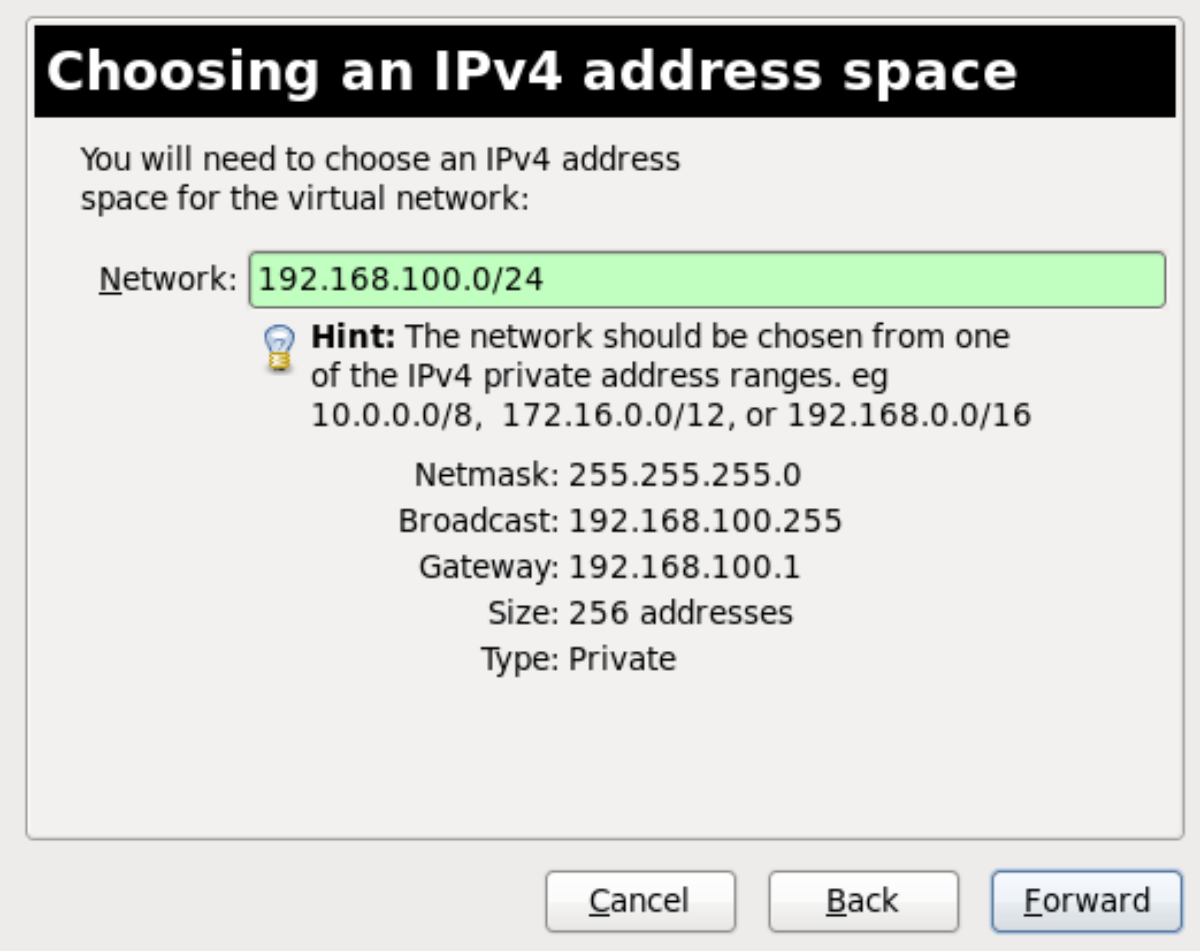


Figure 18.15. Choosing an IPv4 address space

4. Define the DHCP range for your virtual network by specifying a **Start** and **End** range of IP addresses. Click **Forward** to continue.

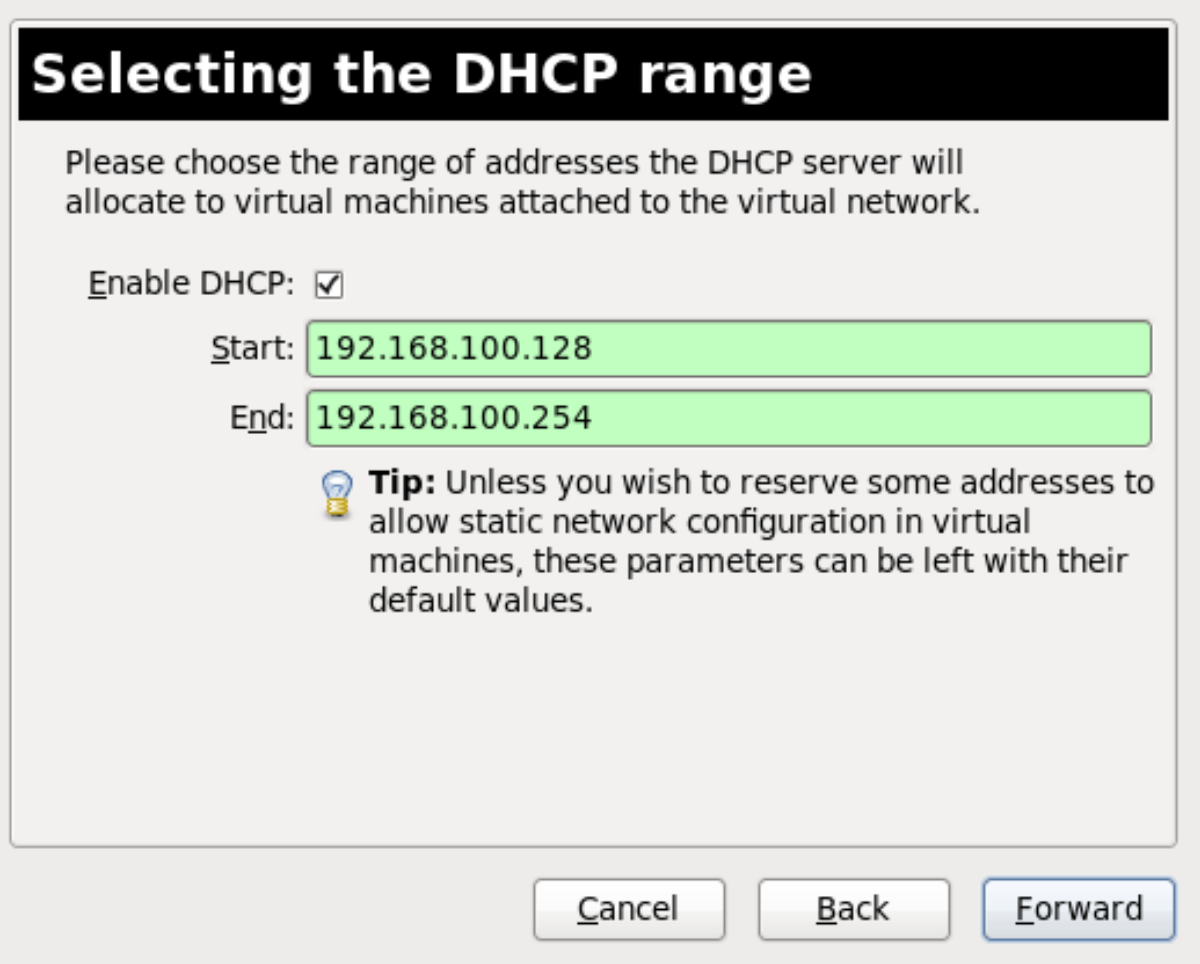


Figure 18.16. Selecting the DHCP range

5. Select how the virtual network should connect to the physical network.

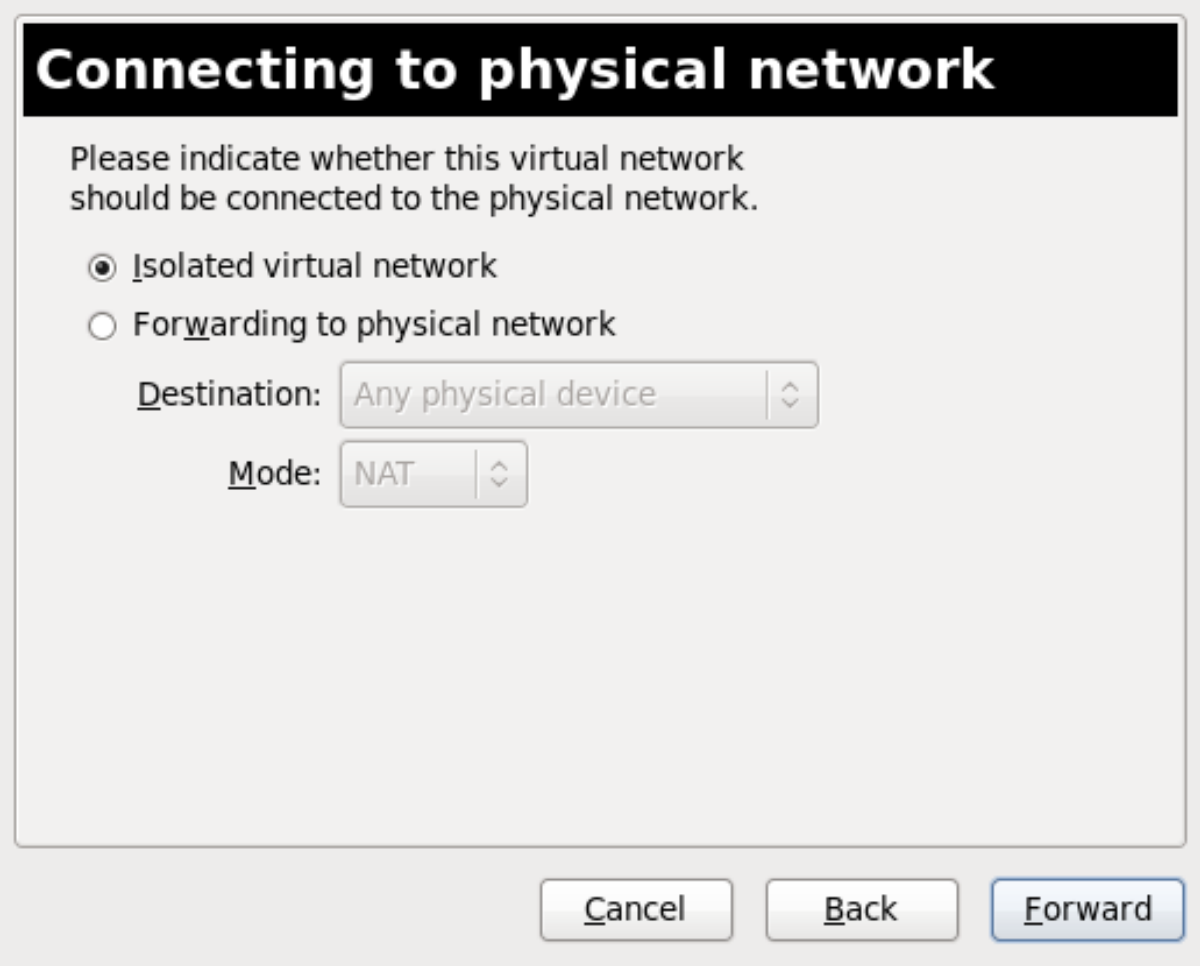


Figure 18.17. Connecting to physical network

If you select **Forwarding to physical network**, choose whether the **Destination** should be **Any physical device** or a specific physical device. Also select whether the **Mode** should be **NAT** or **Routed**.

Click **Forward** to continue.

6. You are now ready to create the network. Check the configuration of your network and click **Finish**.

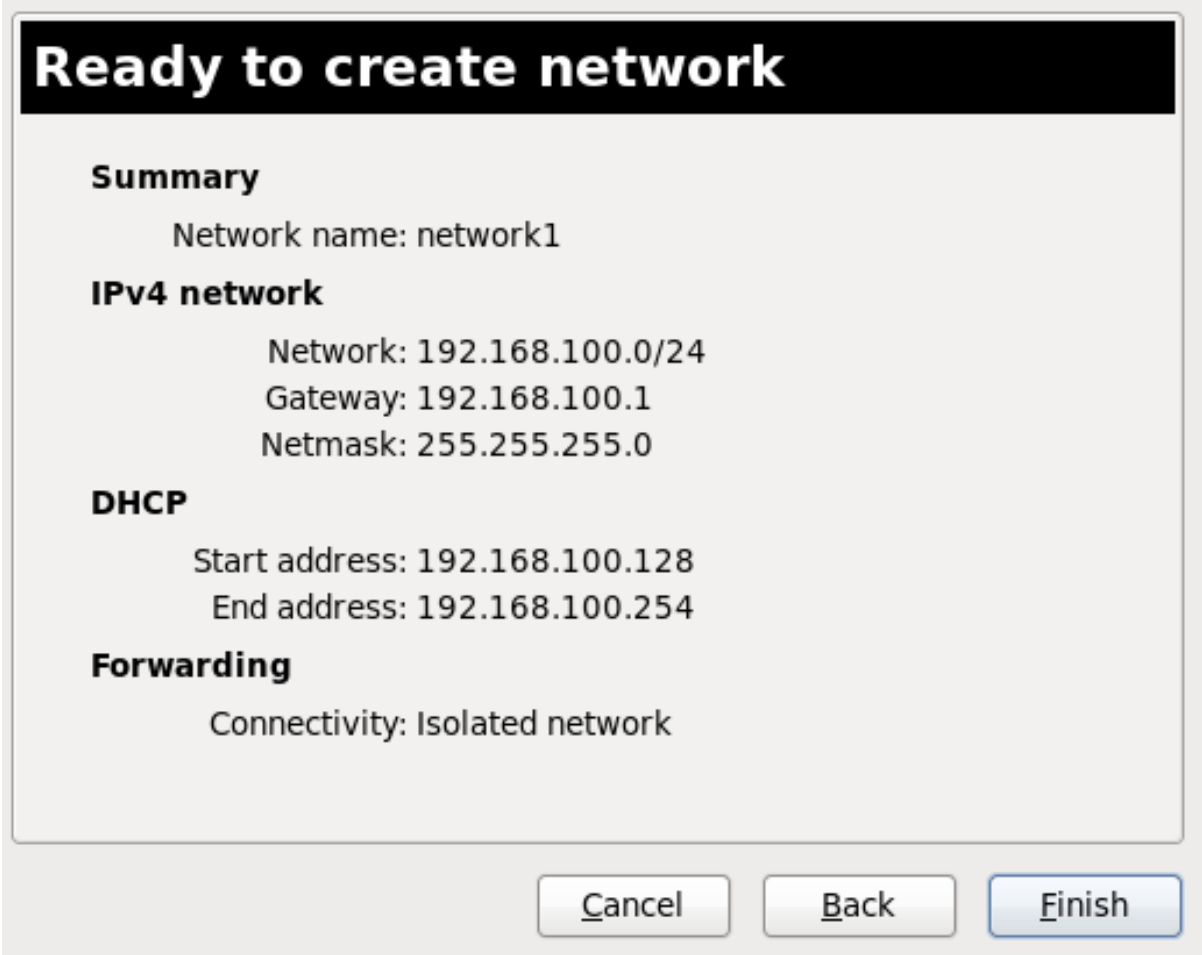


Figure 18.18. Ready to create network

7. The new virtual network is now available in the **Virtual Networks** tab of the **Connection Details** window.

18.10. ATTACHING A VIRTUAL NETWORK TO A GUEST

To attach a virtual network to a guest:

1. In the **Virtual Machine Manager** window, highlight the guest that will have the network assigned.

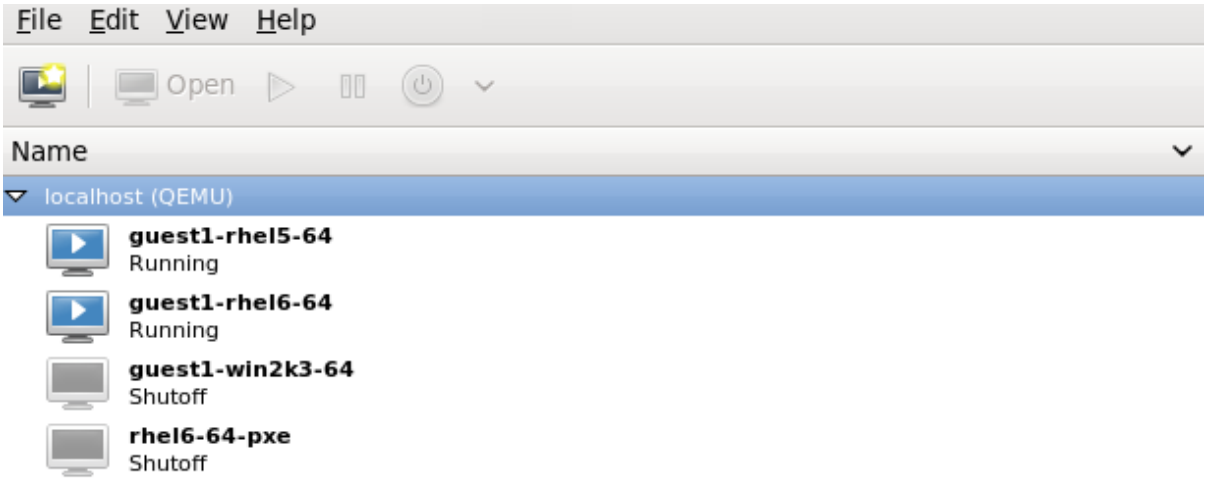


Figure 18.19. Selecting a virtual machine to display

2. From the Virtual Machine Manager **Edit** menu, select **Virtual Machine Details**.

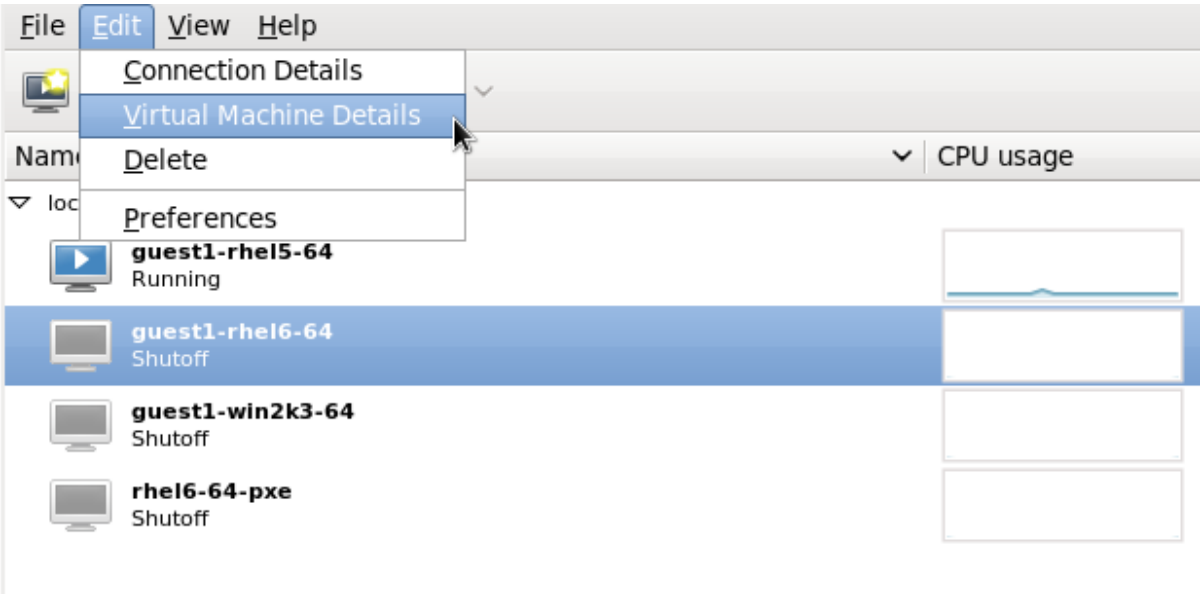


Figure 18.20. Displaying the virtual machine details

3. Click the **Add Hardware** button on the Virtual Machine Details window.

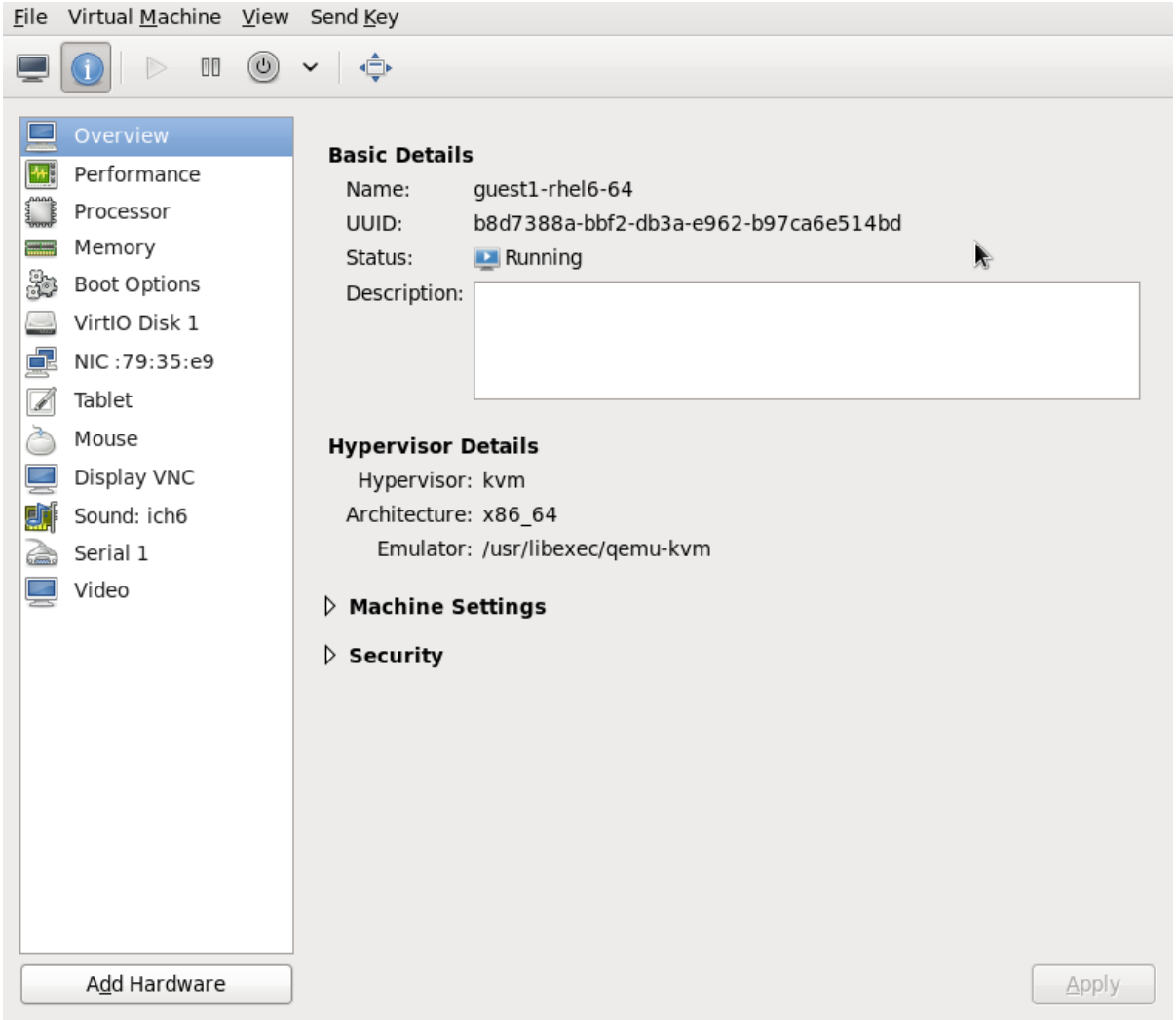


Figure 18.21. The Virtual Machine Details window

4. In the **Add new virtual hardware** window, select **Network** from the left pane, and select your network name (*network1* in this example) from the **Host device** menu and click **Finish**.

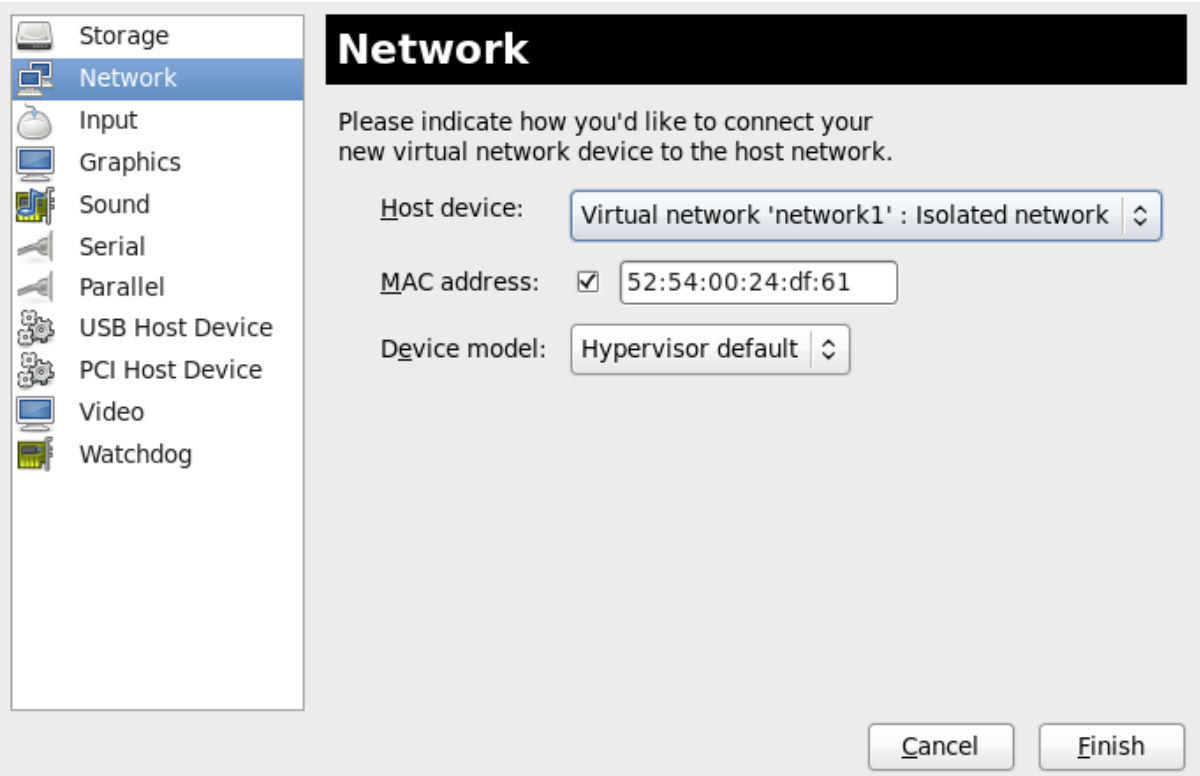


Figure 18.22. Select your network from the Add new virtual hardware window

5. The new network is now displayed as a virtual network interface that will be presented to the guest upon launch.

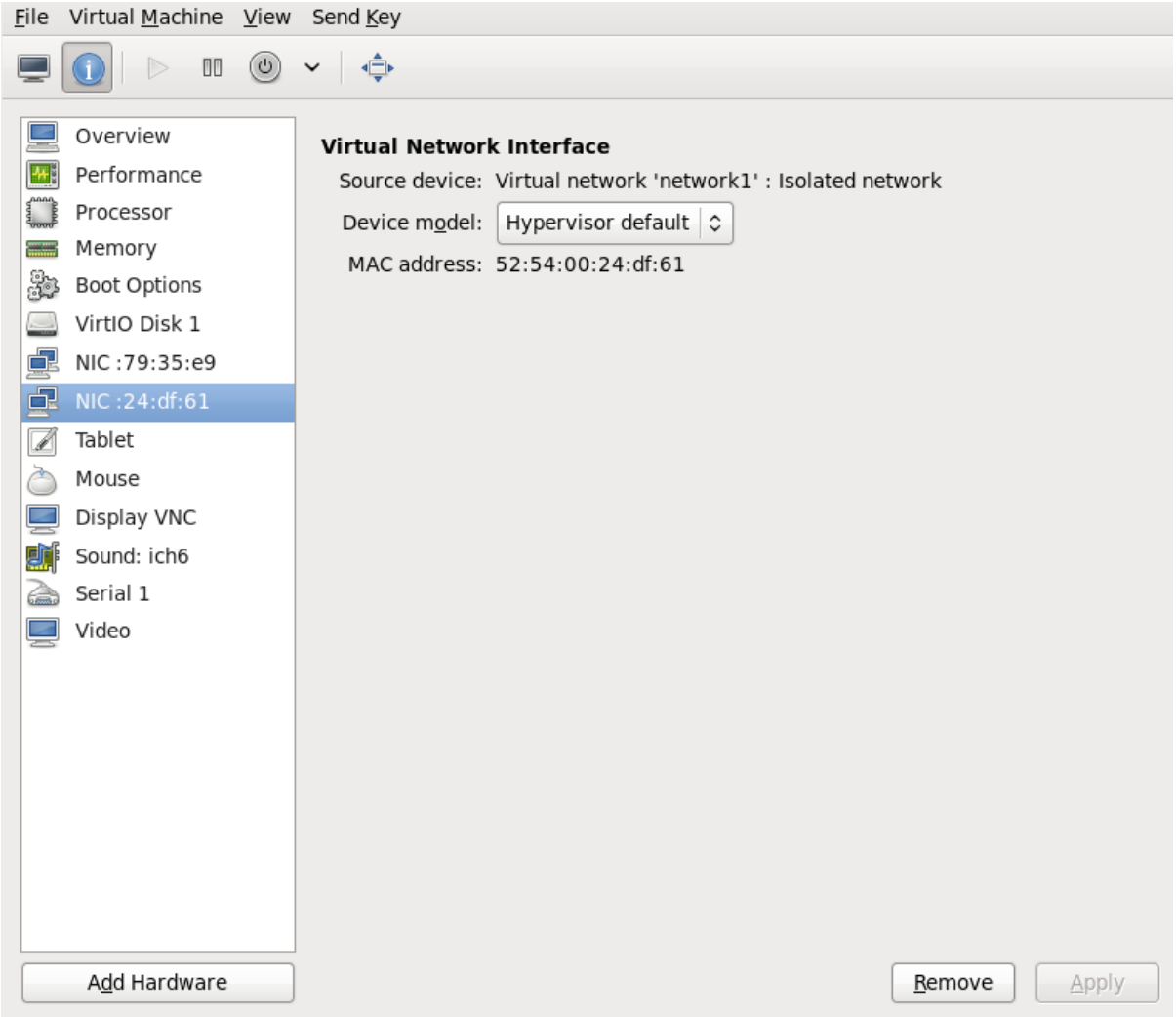


Figure 18.23. New network shown in guest hardware list

18.11. ATTACHING A VIRTUAL NIC DIRECTLY TO A PHYSICAL INTERFACE

As an alternative to the default NAT connection, you can use the *macvtap* driver to attach the guest's NIC directly to a specified physical interface of the host machine. This is not to be confused with device assignment (also known as passthrough). Macvtap connection has the following modes, each with different benefits and usecases:

Physical interface delivery modes

VEPA

In virtual ethernet port aggregator (VEPA) mode, all packets from the guests are sent to the external switch. This enables the user to force guest traffic through the switch. For VEPA mode to work correctly, the external switch must also support *hairpin mode*, which ensures that packets whose destination is a guest on the same host machine as their source guest are sent back to the host by the external switch.

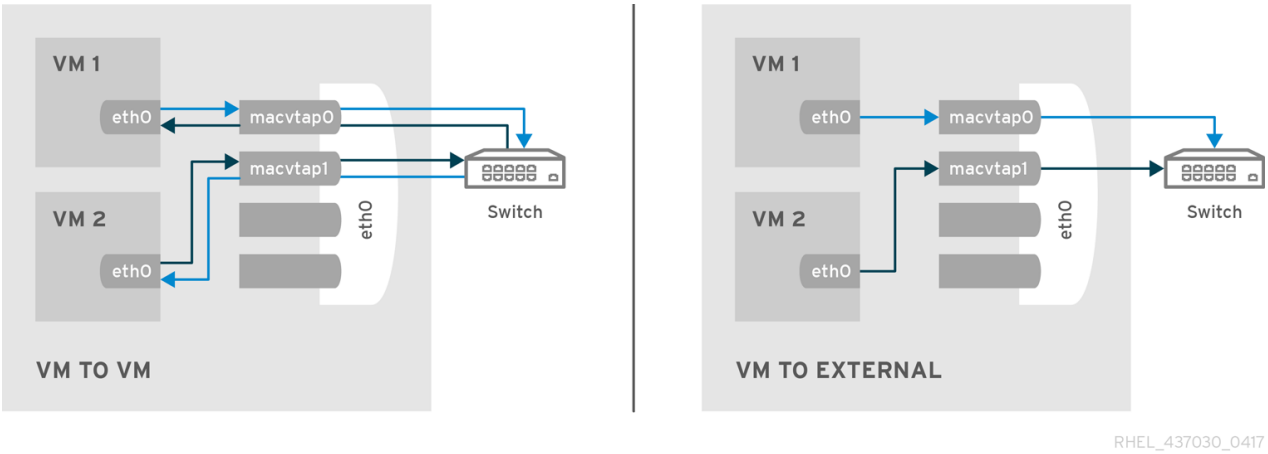


Figure 18.24. VEPA mode

bridge

Packets whose destination is on the same host machine as their source guest are directly delivered to the target macvtap device. Both the source device and the destination device need to be in bridge mode for direct delivery to succeed. If either one of the devices is in VEPA mode, a hairpin-capable external switch is required.

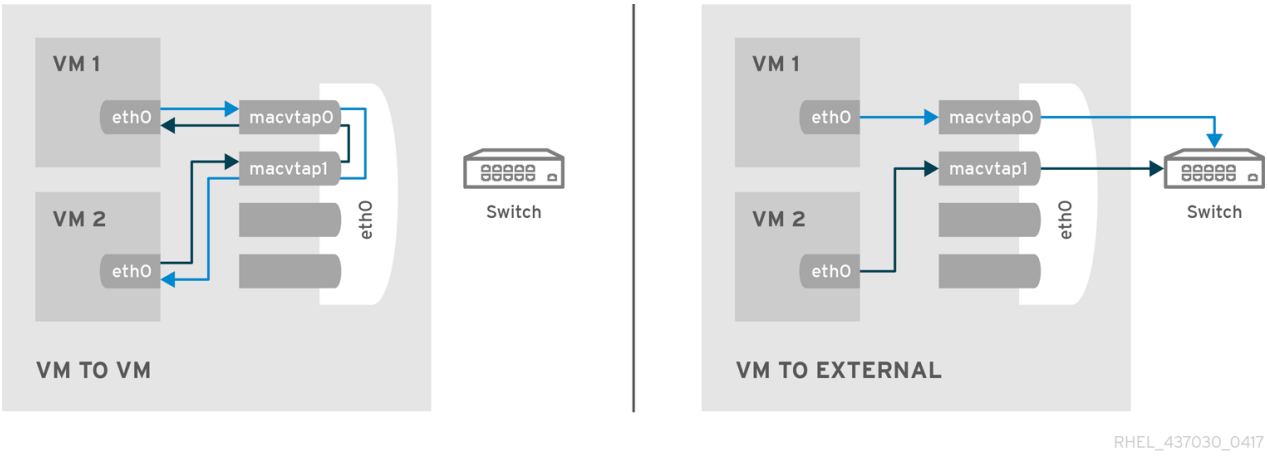


Figure 18.25. Bridge mode

private

All packets are sent to the external switch and will only be delivered to a target guest on the same host machine if they are sent through an external router or gateway and these send them back to the host. Private mode can be used to prevent the individual guests on the single host from communicating with each other. This procedure is followed if either the source or destination device is in private mode.

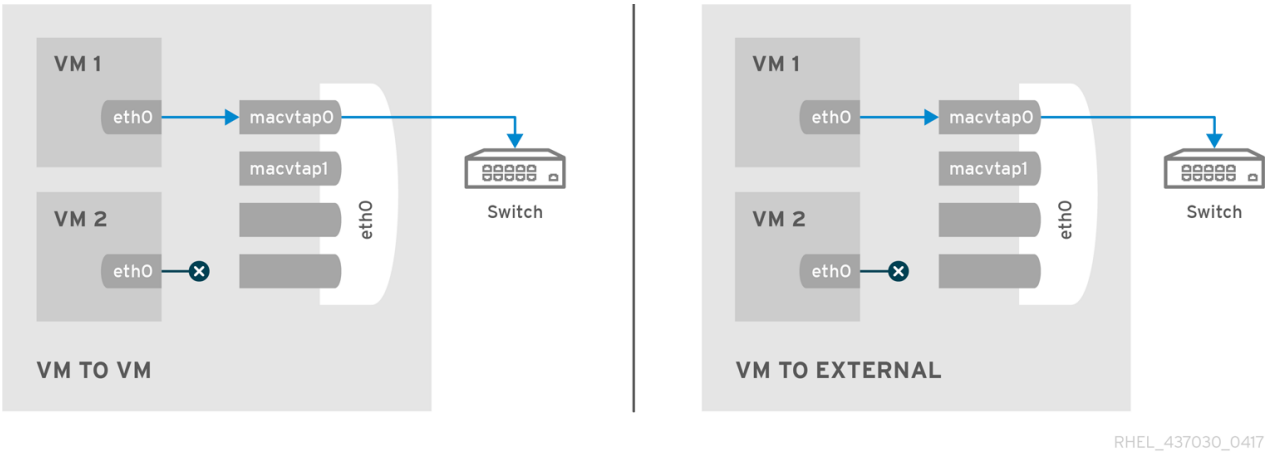


Figure 18.26. Private mode

passthrough

This feature attaches a physical interface device or a SR-IOV Virtual Function (VF) directly to a guest without losing the migration capability. All packets are sent directly to the designated network device. Note that a single network device can only be passed through to a single guest, as a network device cannot be shared between guests in passthrough mode.

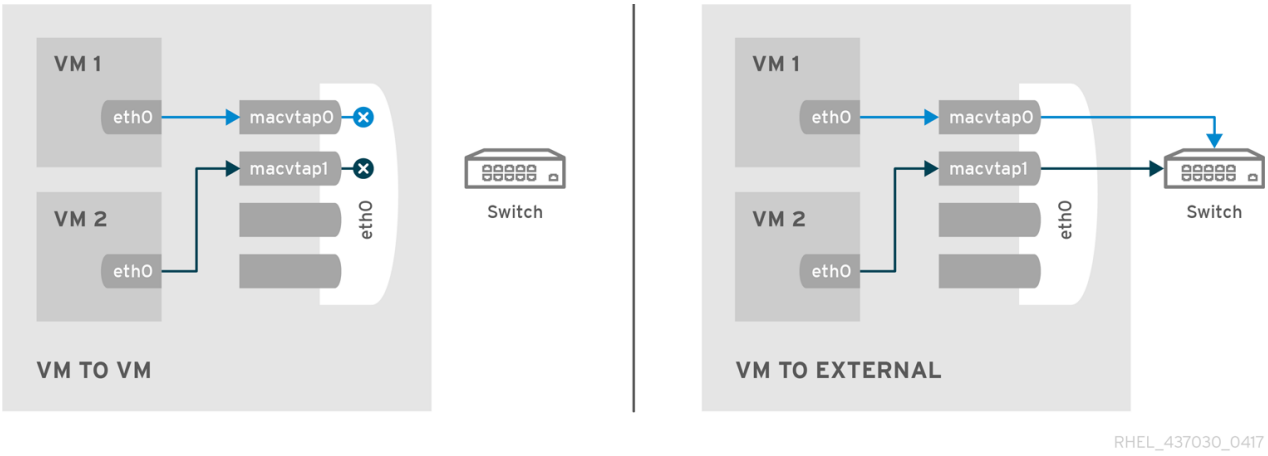


Figure 18.27. Passthrough mode

Each of the four modes is configured by changing the domain xml file. Once this file is opened, change the mode setting as shown:

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
```

The network access of direct attached guest virtual machines can be managed by the hardware switch to which the physical interface of the host physical machine is connected to.

The interface can have additional parameters as shown below, if the switch is conforming to the IEEE 802.1Qbg standard. The parameters of the virtualport element are documented in more detail in the IEEE 802.1Qbg standard. The values are network specific and should be provided by the network

administrator. In 802.1Qbg terms, the Virtual Station Interface (VSI) represents the virtual interface of a virtual machine.

Note that IEEE 802.1Qbg requires a non-zero value for the VLAN ID. Also if the switch is conforming to the IEEE 802.1Qbh standard, the values are network specific and should be provided by the network administrator.

Virtual Station Interface types

managerid

The VSI Manager ID identifies the database containing the VSI type and instance definitions. This is an integer value and the value 0 is reserved.

typeid

The VSI Type ID identifies a VSI type characterizing the network access. VSI types are typically managed by network administrator. This is an integer value.

typeidversion

The VSI Type Version allows multiple versions of a VSI Type. This is an integer value.

instanceid

The VSI Instance ID Identifier is generated when a VSI instance (that is a virtual interface of a virtual machine) is created. This is a globally unique identifier.

profileid

The profile ID contains the name of the port profile that is to be applied onto this interface. This name is resolved by the port profile database into the network parameters from the port profile, and those network parameters will be applied to this interface.

Each of the four types is configured by changing the domain xml file. Once this file is opened, change the mode setting as shown:

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0.2' mode='vepa'/>
    <virtualport type="802.1Qbg">
      <parameters managerid="11" typeid="1193047" typeidversion="2" instanceid="09b11c53-8b5c-
4eeb-8f00-d84eaa0aaa4f"/>
    </virtualport>
  </interface>
</devices>
```

The profile ID is shown here:

```
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private'/>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
```

```
    </virtualport>
  </interface>
</devices>
...
```

18.12. APPLYING NETWORK FILTERING

This section provides an introduction to libvirt's network filters, their goals, concepts and XML format.

18.12.1. Introduction

The goal of the network filtering, is to enable administrators of a virtualized system to configure and enforce network traffic filtering rules on virtual machines and manage the parameters of network traffic that virtual machines are allowed to send or receive. The network traffic filtering rules are applied on the host physical machine when a virtual machine is started. Since the filtering rules cannot be circumvented from within the virtual machine, it makes them mandatory from the point of view of a virtual machine user.

From the point of view of the guest virtual machine, the network filtering system allows each virtual machine's network traffic filtering rules to be configured individually on a per interface basis. These rules are applied on the host physical machine when the virtual machine is started and can be modified while the virtual machine is running. The latter can be achieved by modifying the XML description of a network filter.

Multiple virtual machines can make use of the same generic network filter. When such a filter is modified, the network traffic filtering rules of all running virtual machines that reference this filter are updated. The machines that are not running will update on start.

As previously mentioned, applying network traffic filtering rules can be done on individual network interfaces that are configured for certain types of network configurations. Supported network types include:

- network
- ethernet -- must be used in bridging mode
- bridge

Example 18.1. An example of network filtering

The interface XML is used to reference a top-level filter. In the following example, the interface description references the filter clean-traffic.

```
<devices>
 <interface type='bridge'>
   <mac address='00:16:3e:5d:c7:9e'/>
   <filterref filter='clean-traffic'/>
 </interface>
</devices>
```

Network filters are written in XML and may either contain: references to other filters, rules for traffic filtering, or hold a combination of both. The above referenced filter clean-traffic is a filter that only contains references to other filters and no actual filtering rules. Since references to other filters can be used, a tree of filters can be built. The clean-traffic filter can be viewed using the command: **# virsh nwfilter-dumpxml clean-traffic**.

As previously mentioned, a single network filter can be referenced by multiple virtual machines. Since interfaces will typically have individual parameters associated with their respective traffic filtering rules, the rules described in a filter's XML can be generalized using variables. In this case, the variable name is used in the filter XML and the name and value are provided at the place where the filter is referenced.

Example 18.2. Description extended

In the following example, the interface description has been extended with the parameter IP and a dotted IP address as a value.

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1'/>
    </filterref>
  </interface>
</devices>
```

In this particular example, the clean-traffic network traffic filter will be represented with the IP address parameter 10.0.0.1 and as per the rule dictates that all traffic from this interface will always be using 10.0.0.1 as the source IP address, which is one of the purpose of this particular filter.

18.12.2. Filtering Chains

Filtering rules are organized in filter chains. These chains can be thought of as having a tree structure with packet filtering rules as entries in individual chains (branches).

Packets start their filter evaluation in the root chain and can then continue their evaluation in other chains, return from those chains back into the root chain or be dropped or accepted by a filtering rule in one of the traversed chains.

Libvirt's network filtering system automatically creates individual root chains for every virtual machine's network interface on which the user chooses to activate traffic filtering. The user may write filtering rules that are either directly instantiated in the root chain or may create protocol-specific filtering chains for efficient evaluation of protocol-specific rules.

The following chains exist:

- root
- mac
- stp (spanning tree protocol)
- vlan
- arp and rarp
- ipv4
- ipv6

Multiple chains evaluating the mac, stp, vlan, arp, rarp, ipv4, or ipv6 protocol can be created using the protocol name only as a prefix in the chain's name.

Example 18.3. ARP traffic filtering

This example allows chains with names arp-xyz or arp-test to be specified and have their ARP protocol packets evaluated in those chains.

The following filter XML shows an example of filtering ARP traffic in the arp chain.

```
<filter name='no-arp-spoofing' chain='arp' priority='-500'>
  <uuid>f88f1932-debf-4aa1-9fbe-f10d3aa4bc95</uuid>
  <rule action='drop' direction='out' priority='300'>
    <mac match='no' srcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='350'>
    <arp match='no' arpsrcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='400'>
    <arp match='no' arpsrcipaddr='$IP'/>
  </rule>
  <rule action='drop' direction='in' priority='450'>
    <arp opcode='Reply'/>
    <arp match='no' arpdstmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='in' priority='500'>
    <arp match='no' arpdstipaddr='$IP'/>
  </rule>
  <rule action='accept' direction='inout' priority='600'>
    <arp opcode='Request'/>
  </rule>
  <rule action='accept' direction='inout' priority='650'>
    <arp opcode='Reply'/>
  </rule>
  <rule action='drop' direction='inout' priority='1000'/>
</filter>
```

The consequence of putting ARP-specific rules in the arp chain, rather than for example in the root chain, is that packets protocols other than ARP do not need to be evaluated by ARP protocol-specific rules. This improves the efficiency of the traffic filtering. However, one must then pay attention to only putting filtering rules for the given protocol into the chain since other rules will not be evaluated. For example, an IPv4 rule will not be evaluated in the ARP chain since IPv4 protocol packets will not traverse the ARP chain.

18.12.3. Filtering Chain Priorities

As previously mentioned, when creating a filtering rule, all chains are connected to the root chain. The order in which those chains are accessed is influenced by the priority of the chain. The following table shows the chains that can be assigned a priority and their default priorities.

Table 18.1. Filtering chain default priorities values

Chain (prefix)	Default priority
stp	-810
mac	-800
vlan	-750
ipv4	-700
ipv6	-600
arp	-500
rarp	-400

NOTE

A chain with a lower priority value is accessed before one with a higher value.

The chains listed in Table 18.1, "Filtering chain default priorities values" can be also be assigned custom priorities by writing a value in the range [-1000 to 1000] into the priority (XML) attribute in the filter node. Section 18.12.2, "Filtering Chains"filter shows the default priority of -500 for arp chains, for example.

18.12.4. Usage of Variables in Filters

There are two variables that have been reserved for usage by the network traffic filtering subsystem: MAC and IP.

MAC is designated for the MAC address of the network interface. A filtering rule that references this variable will automatically be replaced with the MAC address of the interface. This works without the user having to explicitly provide the MAC parameter. Even though it is possible to specify the MAC parameter similar to the IP parameter above, it is discouraged since libvirt knows what MAC address an interface will be using.

The parameter ***IP*** represents the IP address that the operating system inside the virtual machine is expected to use on the given interface. The IP parameter is special in so far as the libvirt daemon will try to determine the IP address (and thus the IP parameter's value) that is being used on an interface if the parameter is not explicitly provided but referenced. For current limitations on IP address detection, consult the section on limitations Section 18.12.12, "Limitations" on how to use this feature and what to expect when using it. The XML file shown in Section 18.12.2, "Filtering Chains" contains the filter ***no-arp-spoofing***, which is an example of using a network filter XML to reference the MAC and IP variables.

Note that referenced variables are always prefixed with the character ***$***. The format of the value of a variable must be of the type expected by the filter attribute identified in the XML. In the above example, the ***IP*** parameter must hold a legal IP address in standard format. Failure to provide the correct structure will result in the filter variable not being replaced with a value and will prevent a virtual machine from starting or will prevent an interface from attaching when hot plugging is being used. Some of the types that are expected for each XML attribute are shown in the example Example 18.4, "Sample variable types".

Example 18.4. Sample variable types

As variables can contain lists of elements, (the variable IP can contain multiple IP addresses that are valid on a particular interface, for example), the notation for providing multiple elements for the IP variable is:

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1'/>
      <parameter name='IP' value='10.0.0.2'/>
      <parameter name='IP' value='10.0.0.3'/>
    </filterref>
  </interface>
</devices>
```

This XML file creates filters to enable multiple IP addresses per interface. Each of the IP addresses will result in a separate filtering rule. Therefore using the XML above and the following rule, three individual filtering rules (one for each IP address) will be created:

```
<rule action='accept' direction='in' priority='500'>
  <tcp srpipaddr='$IP'/>
</rule>
```

As it is possible to access individual elements of a variable holding a list of elements, a filtering rule like the following accesses the 2nd element of the variable *DSTPORTS*.

```
<rule action='accept' direction='in' priority='500'>
  <udp dstportstart='$DSTPORTS[1]'/>
</rule>
```

Example 18.5. Using a variety of variables

As it is possible to create filtering rules that represent all possible combinations of rules from different lists using the notation **$VARIABLE[@<iterator id="x">]**. The following rule allows a virtual machine to receive traffic on a set of ports, which are specified in *DSTPORTS*, from the set of source IP address specified in *SRCIPADDRESSES*. The rule generates all combinations of elements of the variable *DSTPORTS* with those of *SRCIPADDRESSES* by using two independent iterators to access their elements.

```
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]'/>
</rule>
```

Assign concrete values to *SRCIPADDRESSES* and *DSTPORTS* as shown:

```
SRCIPADDRESSES = [ 10.0.0.1, 11.1.2.3 ]
DSTPORTS = [ 80, 8080 ]
```

Assigning values to the variables using **$SRCIPADDRESSES[@1]** and **$DSTPORTS[@2]** would then result in all combinations of addresses and ports being created as shown:

- 10.0.0.1, 80
- 10.0.0.1, 8080
- 11.1.2.3, 80
- 11.1.2.3, 8080

Accessing the same variables using a single iterator, for example by using the notation **$SRCIPADDRESSES[@1]** and **$DSTPORTS[@1]**, would result in parallel access to both lists and result in the following combination:

- 10.0.0.1, 80
- 11.1.2.3, 8080

NOTE

$VARIABLE is short-hand for **$VARIABLE[@0]**. The former notation always assumes the role of iterator with **iterator id="0"** added as shown in the opening paragraph at the top of this section.

18.12.5. Automatic IP Address Detection and DHCP Snooping

This section provides information about automatic IP address detection and DHCP snooping.

18.12.5.1. Introduction

The detection of IP addresses used on a virtual machine's interface is automatically activated if the variable IP is referenced but no value has been assigned to it. The variable ***CTRL_IP_LEARNING*** can be used to specify the IP address learning method to use. Valid values include: *any*, *dhcp*, or *none*.

The value *any* instructs libvirt to use any packet to determine the address in use by a virtual machine, which is the default setting if the variable ***TRL_IP_LEARNING*** is not set. This method will only detect a single IP address per interface. Once a guest virtual machine's IP address has been detected, its IP network traffic will be locked to that address, if for example, IP address spoofing is prevented by one of its filters. In that case, the user of the VM will not be able to change the IP address on the interface inside the guest virtual machine, which would be considered IP address spoofing. When a guest virtual machine is migrated to another host physical machine or resumed after a suspend operation, the first packet sent by the guest virtual machine will again determine the IP address that the guest virtual machine can use on a particular interface.

The value of *dhcp* instructs libvirt to only honor DHCP server-assigned addresses with valid leases. This method supports the detection and usage of multiple IP address per interface. When a guest virtual machine resumes after a suspend operation, any valid IP address leases are applied to its filters. Otherwise the guest virtual machine is expected to use DHCP to obtain a new IP addresses. When a guest virtual machine migrates to another physical host physical machine, the guest virtual machine is required to re-run the DHCP protocol.

If CTRL_IP_LEARNING is set to *none*, libvirt does not do IP address learning and referencing IP without assigning it an explicit value is an error.

18.12.5.2. DHCP Snooping

CTRL_IP_LEARNING=*dhcp* (DHCP snooping) provides additional anti-spoofing security, especially when combined with a filter allowing only trusted DHCP servers to assign IP addresses. To enable this, set the variable ***DHCPSERVER*** to the IP address of a valid DHCP server and provide filters that use this variable to filter incoming DHCP responses.

When DHCP snooping is enabled and the DHCP lease expires, the guest virtual machine will no longer be able to use the IP address until it acquires a new, valid lease from a DHCP server. If the guest virtual machine is migrated, it must get a new valid DHCP lease to use an IP address (for example, by bringing the VM interface down and up again).

NOTE

Automatic DHCP detection listens to the DHCP traffic the guest virtual machine exchanges with the DHCP server of the infrastructure. To avoid denial-of-service attacks on libvirt, the evaluation of those packets is rate-limited, meaning that a guest virtual machine sending an excessive number of DHCP packets per second on an interface will not have all of those packets evaluated and thus filters may not get adapted. Normal DHCP client behavior is assumed to send a low number of DHCP packets per second. Further, it is important to setup appropriate filters on all guest virtual machines in the infrastructure to avoid them being able to send DHCP packets. Therefore guest virtual machines must either be prevented from sending UDP and TCP traffic from port 67 to port 68 or the DHCPSERVER variable should be used on all guest virtual machines to restrict DHCP server messages to only be allowed to originate from trusted DHCP servers. At the same time anti-spoofing prevention must be enabled on all guest virtual machines in the subnet.

Example 18.6. Activating IPs for DHCP snooping

The following XML provides an example for the activation of IP address learning using the DHCP snooping method:

```
<interface type='bridge'>
  <source bridge='virbr0'/>
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp'/>
  </filterref>
</interface>
```

18.12.6. Reserved Variables

Table 18.2, “Reserved variables” shows the variables that are considered reserved and are used by libvirt:

Table 18.2. Reserved variables

Variable Name	Definition
MAC	The MAC address of the interface
IP	The list of IP addresses in use by an interface

Variable Name	Definition
IPV6	Not currently implemented: the list of IPV6 addresses in use by an interface
DHCPSERVER	The list of IP addresses of trusted DHCP servers
DHCPSERVERV6	Not currently implemented: The list of IPv6 addresses of trusted DHCP servers
CTRL_IP_LEARNING	The choice of the IP address detection mode

18.12.7. Element and Attribute Overview

The root element required for all network filters is named **<filter>** with two possible attributes. The **name** attribute provides a unique name of the given filter. The **chain** attribute is optional but allows certain filters to be better organized for more efficient processing by the firewall subsystem of the underlying host physical machine. Currently the system only supports the following chains: **root**, **ipv4**, **ipv6**, **arp** and **rarp**.

18.12.8. References to Other Filters

Any filter may hold references to other filters. Individual filters may be referenced multiple times in a filter tree but references between filters must not introduce loops.

Example 18.7. An Example of a clean traffic filter

The following shows the XML of the clean-traffic network filter referencing several other filters.

```
<filter name='clean-traffic'>
  <uuid>6ef53069-ba34-94a0-d33d-17751b9b8cb1</uuid>
  <filterref filter='no-mac-spoofing'/>
  <filterref filter='no-ip-spoofing'/>
  <filterref filter='allow-incoming-ipv4'/>
  <filterref filter='no-arp-spoofing'/>
  <filterref filter='no-other-l2-traffic'/>
  <filterref filter='qemu-announce-self'/>
</filter>
```

To reference another filter, the XML node filterref needs to be provided inside a filter node. This node must have the attribute filter whose value contains the name of the filter to be referenced.

New network filters can be defined at any time and may contain references to network filters that are not known to libvirt, yet. However, once a virtual machine is started or a network interface referencing a filter is to be hotplugged, all network filters in the filter tree must be available. Otherwise the virtual machine will not start or the network interface cannot be attached.

18.12.9. Filter Rules

The following XML shows a simple example of a network traffic filter implementing a rule to drop traffic if the IP address (provided through the value of the variable IP) in an outgoing IP packet is not the expected one, thus preventing IP address spoofing by the VM.

Example 18.8. Example of network traffic filtering

```
<filter name='no-ip-spoofing' chain='ipv4'>
  <uuid>fce8ae33-e69e-83bf-262e-30786c1f8072</uuid>
  <rule action='drop' direction='out' priority='500'>
    <ip match='no' srcipaddr='$IP'/>
  </rule>
</filter>
```

The traffic filtering rule starts with the rule node. This node may contain up to three of the following attributes:

- action is mandatory can have the following values:
  - drop (matching the rule silently discards the packet with no further analysis)
  - reject (matching the rule generates an ICMP reject message with no further analysis)
  - accept (matching the rule accepts the packet with no further analysis)
  - return (matching the rule passes this filter, but returns control to the calling filter for further analysis)
  - continue (matching the rule goes on to the next rule for further analysis)
- direction is mandatory can have the following values:
  - in for incoming traffic
  - out for outgoing traffic
  - inout for incoming and outgoing traffic
- priority is optional. The priority of the rule controls the order in which the rule will be instantiated relative to other rules. Rules with lower values will be instantiated before rules with higher values. Valid values are in the range of -1000 to 1000. If this attribute is not provided, priority 500 will be assigned by default. Note that filtering rules in the root chain are sorted with filters connected to the root chain following their priorities. This allows to interleave filtering rules with access to filter chains. Refer to Section 18.12.3, “Filtering Chain Priorities” for more information.
- statematch is optional. Possible values are '0' or 'false' to turn the underlying connection state matching off. The default setting is 'true' or 1

For more information see Section 18.12.11, “Advanced Filter Configuration Topics” .

The above example Example 18.7, “An Example of a clean traffic filter” indicates that the traffic of *type ip* will be associated with the chain *ipv4* and the rule will have **priority=*500***. If for example another filter is referenced whose traffic of *type ip* is also associated with the chain *ipv4* then that filter's rules will be ordered relative to the **priority=*500*** of the shown rule.

A rule may contain a single rule for filtering of traffic. The above example shows that traffic of type ip is to be filtered.

18.12.10. Supported Protocols

The following sections list and give some details about the protocols that are supported by the network filtering subsystem. This type of traffic rule is provided in the rule node as a nested node. Depending on the traffic type a rule is filtering, the attributes are different. The above example showed the single attribute ***srcipaddr*** that is valid inside the ip traffic filtering node. The following sections show what attributes are valid and what type of data they are expecting. The following datatypes are available:

- UINT8 : 8 bit integer; range 0-255
- UINT16: 16 bit integer; range 0-65535
- MAC_ADDR: MAC address in dotted decimal format, such as 00:11:22:33:44:55
- MAC_MASK: MAC address mask in MAC address format, such as FF:FF:FF:FC:00:00
- IP_ADDR: IP address in dotted decimal format, such as 10.1.2.3
- IP_MASK: IP address mask in either dotted decimal format (255.255.248.0) or CIDR mask (0-32)
- IPV6_ADDR: IPv6 address in numbers format, such as FFFF::1
- IPV6_MASK: IPv6 mask in numbers format (FFFF:FFFF:FC00::) or CIDR mask (0-128)
- STRING: A string
- BOOLEAN: 'true', 'yes', '1' or 'false', 'no', '0'
- IPSETFLAGS: The source and destination flags of the ipset described by up to 6 'src' or 'dst' elements selecting features from either the source or destination part of the packet header; example: src,src,dst. The number of 'selectors' to provide here depends on the type of ipset that is referenced

Every attribute except for those of type ***IP_MASK*** or ***IPV6_MASK*** can be negated using the match attribute with value *no*. Multiple negated attributes may be grouped together. The following XML fragment shows such an example using abstract attributes.

```
[...]
  <rule action='drop' direction='in'>
    <protocol match='no' attribute1='value1' attribute2='value2'/>
    <protocol attribute3='value3'/>
  </rule>
[...]
```

Rules behave evaluate the rule as well as look at it logically within the boundaries of the given protocol attributes. Thus, if a single attribute's value does not match the one given in the rule, the whole rule will be skipped during the evaluation process. Therefore, in the above example incoming traffic will only be dropped if: the protocol property ***attribute1*** does not match both ***value1*** and the protocol property ***attribute2*** does not match ***value2*** and the protocol property ***attribute3*** matches ***value3***.

18.12.10.1. MAC (Ethernet)

Protocol ID: mac

Rules of this type should go into the root chain.

Table 18.3. MAC protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
protocolid	UINT16 (0x600-0xffff), STRING	Layer 3 protocol ID. Valid strings include [arp, rarp, ipv4, ipv6]
comment	STRING	text string up to 256 characters

The filter can be written as such:

```
[...]
<mac match='no' srcmacaddr='$MAC'/>
[...]
```

18.12.10.2. VLAN (802.1Q)

Protocol ID: vlan

Rules of this type should go either into the root or vlan chain.

Table 18.4. VLAN protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
vlan-id	UINT16 (0x0-0xfff, 0 - 4095)	VLAN ID

Attribute Name	Datatype	Definition
encap-protocol	UINT16 (0x03c-0xfff), String	Encapsulated layer 3 protocol ID, valid strings are arp, ipv4, ipv6
comment	STRING	text string up to 256 characters

18.12.10.3. STP (Spanning Tree Protocol)

Protocol ID: stp

Rules of this type should go either into the root or stp chain.

Table 18.5. STP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
type	UINT8	Bridge Protocol Data Unit (BPDU) type
flags	UINT8	BPDU flagdstmacmask
root-priority	UINT16	Root priority range start
root-priority-hi	UINT16 (0x0-0xfff, 0 - 4095)	Root priority range end
root-address	MAC _ADDRESS	root MAC Address
root-address-mask	MAC _MASK	root MAC Address mask
roor-cost	UINT32	Root path cost (range start)
root-cost-hi	UINT32	Root path cost range end
sender-priority-hi	UINT16	Sender priority range end
sender-address	MAC_ADDRESS	BPDU sender MAC address
sender-address-mask	MAC_MASK	BPDU sender MAC address mask
port	UINT16	Port identifier (range start)
port_hi	UINT16	Port identifier range end

Attribute Name	Datatype	Definition
msg-age	UINT16	Message age timer (range start)
msg-age-hi	UINT16	Message age timer range end
max-age-hi	UINT16	Maximum age time range end
hello-time	UINT16	Hello time timer (range start)
hello-time-hi	UINT16	Hello time timer range end
forward-delay	UINT16	Forward delay (range start)
forward-delay-hi	UINT16	Forward delay range end
comment	STRING	text string up to 256 characters

18.12.10.4. ARP/RARP

Protocol ID: arp or rarp

Rules of this type should either go into the root or arp/rarp chain.

Table 18.6. ARP and RARP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
hwtype	UINT16	Hardware type
protocoltype	UINT16	Protocol type
opcode	UINT16, STRING	Opcode valid strings are: Request, Reply, Request_Reverse, Reply_Reverse, DRARP_Request, DRARP_Reply, DRARP_Error, InARP_Request, ARP_NAK

Attribute Name	Datatype	Definition
arpsrcmacaddr	MAC_ADDR	Source MAC address in ARP/RARP packet
arpdstmacaddr	MAC _ADDR	Destination MAC address in ARP/RARP packet
arpsrcipaddr	IP_ADDR	Source IP address in ARP/RARP packet
arpdstipaddr	IP_ADDR	Destination IP address in ARP/RARP packet
gratuitous	BOOLEAN	Boolean indiating whether to check for a gratuitous ARP packet
comment	STRING	text string up to 256 characters

18.12.10.5. IPv4

Protocol ID: ip

Rules of this type should either go into the root or ipv4 chain.

Table 18.7. IPv4 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address

Attribute Name	Datatype	Definition
protocol	UINT8, STRING	Layer 4 protocol identifier. Valid strings for protocol are: tcp, udp, udplite, esp, ah, icmp, igmp, sctp
srcportstart	UINT16	Start of range of valid source ports; requires protocol
srcportend	UINT16	End of range of valid source ports; requires protocol
dstportstart	UNIT16	Start of range of valid destination ports; requires protocol
dstportend	UNIT16	End of range of valid destination ports; requires protocol
comment	STRING	text string up to 256 characters

18.12.10.6. IPv6

Protocol ID: ipv6

Rules of this type should either go into the root or ipv6 chain.

Table 18.8. IPv6 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to MAC address of sender
dstmacaddr	MAC_ADDR	MAC address of destination
dstmacmask	MAC_MASK	Mask applied to MAC address of destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address

Attribute Name	Datatype	Definition
protocol	UINT8, STRING	Layer 4 protocol identifier. Valid strings for protocol are: tcp, udp, udplite, esp, ah, icmpv6, sctp
scrportstart	UNIT16	Start of range of valid source ports; requires protocol
srcportend	UINT16	End of range of valid source ports; requires protocol
dstportstart	UNIT16	Start of range of valid destination ports; requires protocol
dstportend	UNIT16	End of range of valid destination ports; requires protocol
comment	STRING	text string up to 256 characters

18.12.10.7. TCP/UDP/SCTP

Protocol ID: tcp, udp, sctp

The chain parameter is ignored for this type of traffic and should either be omitted or set to root. .

Table 18.9. TCP/UDP/SCTP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
scripto	IP_ADDR	Start of range of source IP address
srcipfrom	IP_ADDR	End of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address

Attribute Name	Datatype	Definition
dstipto	IP_ADDR	End of range of destination IP address
scrportstart	UNIT16	Start of range of valid source ports; requires protocol
srcportend	UINT16	End of range of valid source ports; requires protocol
dstportstart	UNIT16	Start of range of valid destination ports; requires protocol
dstportend	UNIT16	End of range of valid destination ports; requires protocol
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE
flags	STRING	TCP-only: format of mask/flags with mask and flags each being a comma separated list of SYN,ACK,URG,PSH,FIN,RST or NONE or ALL
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.12.10.8. ICMP

Protocol ID: icmp

Note: The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.10. ICMP protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to the MAC address of the sender

Attribute Name	Datatype	Definition
dstmacaddr	MAD_ADDR	MAC address of the destination
dstmacmask	MAC_MASK	Mask applied to the MAC address of the destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
type	UNIT16	ICMP type
code	UNIT16	ICMP code
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.12.10.9. IGMP, ESP, AH, UDPLITE, 'ALL'

Protocol ID: igmp, esp, ah, udplite, all

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.11. IGMP, ESP, AH, UDPLITE, 'ALL'

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcmacmask	MAC_MASK	Mask applied to the MAC address of the sender
dstmacaddr	MAD_ADDR	MAC address of the destination
dstmacmask	MAC_MASK	Mask applied to the MAC address of the destination
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.12.10.10. TCP/UDP/SCTP over IPV6

Protocol ID: tcp-ipv6, udp-ipv6, sctp-ipv6

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.12. TCP, UDP, SCTP over IPv6 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
srcportstart	UINT16	Start of range of valid source ports
srcportend	UINT16	End of range of valid source ports
dstportstart	UINT16	Start of range of valid destination ports
dstportend	UINT16	End of range of valid destination ports
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.12.10.11. ICMPv6

Protocol ID: icmpv6

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.13. ICMPv6 protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
type	UINT16	ICMPv6 type
code	UINT16	ICMPv6 code
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.12.10.12. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv6

Protocol ID: igmp-ipv6, esp-ipv6, ah-ipv6, udplite-ipv6, all-ipv6

The chain parameter is ignored for this type of traffic and should either be omitted or set to root.

Table 18.14. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv protocol types

Attribute Name	Datatype	Definition
srcmacaddr	MAC_ADDR	MAC address of sender
srcipaddr	IP_ADDR	Source IP address
srcipmask	IP_MASK	Mask applied to source IP address
dstipaddr	IP_ADDR	Destination IP address
dstipmask	IP_MASK	Mask applied to destination IP address
srcipfrom	IP_ADDR	start of range of source IP address
scripto	IP_ADDR	end of range of source IP address
dstipfrom	IP_ADDR	Start of range of destination IP address
dstipto	IP_ADDR	End of range of destination IP address
comment	STRING	text string up to 256 characters
state	STRING	comma separated list of NEW,ESTABLISHED,RELATED,INVALID or NONE
ipset	STRING	The name of an IPSet managed outside of libvirt
ipsetflags	IPSETFLAGS	flags for the IPSet; requires ipset attribute

18.12.11. Advanced Filter Configuration Topics

The following sections discuss advanced filter configuration topics.

18.12.11.1. Connection tracking

The network filtering subsystem (on Linux) makes use of the connection tracking support of IP tables. This helps in enforcing the directionality of network traffic (state match) as well as counting and limiting the number of simultaneous connections towards a guest virtual machine. As an example, if a guest virtual machine has TCP port 8080 open as a server, clients may connect to the guest virtual machine on port 8080. Connection tracking and enforcement of directionality then prevents the guest virtual machine from initiating a connection from (TCP client) port 8080 to the host physical machine back to a remote host physical machine. More importantly, tracking helps to prevent remote attackers from

establishing a connection back to a guest virtual machine. For example, if the user inside the guest virtual machine established a connection to port 80 on an attacker site, then the attacker will not be able to initiate a connection from TCP port 80 back towards the guest virtual machine. By default the connection state match that enables connection tracking and then enforcement of directionality of traffic is turned on.

Example 18.9. XML example for turning off connections to the TCP port

The following shows an example XML fragment where this feature has been turned off for incoming connections to TCP port 12345.

```
[...]
 <rule direction='in' action='accept' statematch='false'>
   <cp dstportstart='12345'/>
 </rule>
[...]
```

This now allows incoming traffic to TCP port 12345, but would also enable the initiation from (client) TCP port 12345 within the VM, which may or may not be desirable.

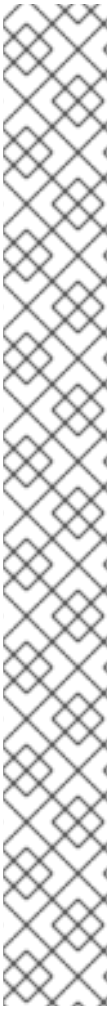
18.12.11.2. Limiting Number of Connections

To limit the number of connections a guest virtual machine may establish, a rule must be provided that sets a limit of connections for a given type of traffic. If for example a VM is supposed to be allowed to only ping one other IP address at a time and is supposed to have only one active incoming ssh connection at a time.

Example 18.10. XML sample file that sets limits to connections

The following XML fragment can be used to limit connections

```
[...]
<rule action='drop' direction='in' priority='400'>
  <tcp connlimit-above='1'/>
</rule>
<rule action='accept' direction='in' priority='500'>
  <tcp dstportstart='22'/>
</rule>
<rule action='drop' direction='out' priority='400'>
  <icmp connlimit-above='1'/>
</rule>
<rule action='accept' direction='out' priority='500'>
  <icmp/>
</rule>
<rule action='accept' direction='out' priority='500'>
  <udp dstportstart='53'/>
</rule>
<rule action='drop' direction='inout' priority='1000'>
  <all/>
</rule>
[...]
```



NOTE

Limitation rules must be listed in the XML prior to the rules for accepting traffic. According to the XML file in Example 18.10, "XML sample file that sets limits to connections", an additional rule for allowing DNS traffic sent to port 22 go out the guest virtual machine, has been added to avoid ssh sessions not getting established for reasons related to DNS lookup failures by the ssh daemon. Leaving this rule out may result in the ssh client hanging unexpectedly as it tries to connect. Additional caution should be used in regards to handling timeouts related to tracking of traffic. An ICMP ping that the user may have terminated inside the guest virtual machine may have a long timeout in the host physical machine's connection tracking system and will therefore not allow another ICMP ping to go through.

The best solution is to tune the timeout in the host physical machine's **sysfs** with the following command:# **echo 3 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout**. This command sets the ICMP connection tracking timeout to 3 seconds. The effect of this is that once one ping is terminated, another one can start after 3 seconds.

If for any reason the guest virtual machine has not properly closed its TCP connection, the connection to be held open for a longer period of time, especially if the TCP timeout value was set for a large amount of time on the host physical machine. In addition, any idle connection may result in a time out in the connection tracking system which can be re-activated once packets are exchanged.

However, if the limit is set too low, newly initiated connections may force an idle connection into TCP backoff. Therefore, the limit of connections should be set rather high so that fluctuations in new TCP connections do not cause odd traffic behavior in relation to idle connections.

18.12.11.3. Command line tools

virsh has been extended with life-cycle support for network filters. All commands related to the network filtering subsystem start with the prefix ***nwfilter***. The following commands are available:

- **nwfilter-list** : lists UUIDs and names of all network filters
- **nwfilter-define** : defines a new network filter or updates an existing one (must supply a name)
- **nwfilter-undefine** : deletes a specified network filter (must supply a name). Do not delete a network filter currently in use.
- **nwfilter-dumpxml** : displays a specified network filter (must supply a name)
- **nwfilter-edit** : edits a specified network filter (must supply a name)

18.12.11.4. Pre-existing network filters

The following is a list of example network filters that are automatically installed with libvirt:

Table 18.15. ICMPv6 protocol types

Command Name	Description

Command Name	Description
no-arp-spoofing	Prevents a guest virtual machine from spoofing ARP traffic; this filter only allows ARP request and reply messages and enforces that those packets contain the MAC and IP addresses of the guest virtual machine.
allow-dhcp	Allows a guest virtual machine to request an IP address via DHCP (from any DHCP server)
allow-dhcp-server	Allows a guest virtual machine to request an IP address from a specified DHCP server. The dotted decimal IP address of the DHCP server must be provided in a reference to this filter. The name of the variable must be *DHCPSERVER*.
no-ip-spoofing	Prevents a guest virtual machine from sending IP packets with a source IP address different from the one inside the packet.
no-ip-multicast	Prevents a guest virtual machine from sending IP multicast packets.
clean-traffic	Prevents MAC, IP and ARP spoofing. This filter references several other filters as building blocks.

These filters are only building blocks and require a combination with other filters to provide useful network traffic filtering. The most used one in the above list is the *clean-traffic* filter. This filter itself can for example be combined with the *no-ip-multicast* filter to prevent virtual machines from sending IP multicast traffic on top of the prevention of packet spoofing.

18.12.11.5. Writing your own filters

Since libvirt only provides a couple of example networking filters, you may consider writing your own. When planning on doing so there are a couple of things you may need to know regarding the network filtering subsystem and how it works internally. Certainly you also have to know and understand the protocols very well that you want to be filtering on so that no further traffic than what you want can pass and that in fact the traffic you want to allow does pass.

The network filtering subsystem is currently only available on Linux host physical machines and only works for Qemu and KVM type of virtual machines. On Linux, it builds upon the support for ebtables, iptables and ip6tables and makes use of their features. Considering the list found in Section 18.12.10, “Supported Protocols” the following protocols can be implemented using ebtables:

- mac
- stp (spanning tree protocol)
- vlan (802.1Q)
- arp, rarp

- ipv4
- ipv6

Any protocol that runs over IPv4 is supported using iptables, those over IPv6 are implemented using ip6tables.

Using a Linux host physical machine, all traffic filtering rules created by libvirt's network filtering subsystem first passes through the filtering support implemented by ebtables and only afterwards through iptables or ip6tables filters. If a filter tree has rules with the protocols including: mac, stp, vlan arp, rarp, ipv4, or ipv6; the ebtable rules and values listed will automatically be used first.

Multiple chains for the same protocol can be created. The name of the chain must have a prefix of one of the previously enumerated protocols. To create an additional chain for handling of ARP traffic, a chain with name arp-test, can for example be specified.

As an example, it is possible to filter on UDP traffic by source and destination ports using the IP protocol filter and specifying attributes for the protocol, source and destination IP addresses and ports of UDP packets that are to be accepted. This allows early filtering of UDP traffic with ebtables. However, once an IP or IPv6 packet, such as a UDP packet, has passed the ebtables layer and there is at least one rule in a filter tree that instantiates iptables or ip6tables rules, a rule to let the UDP packet pass will also be necessary to be provided for those filtering layers. This can be achieved with a rule containing an appropriate udp or udp-ipv6 traffic filtering node.

Example 18.11. Creating a custom filter

Suppose a filter is needed to fulfill the following list of requirements:

- prevents a VM's interface from MAC, IP and ARP spoofing
- opens only TCP ports 22 and 80 of a VM's interface
- allows the VM to send ping traffic from an interface but not let the VM be pinged on the interface
- allows the VM to do DNS lookups (UDP towards port 53)

The requirement to prevent spoofing is fulfilled by the existing ***clean-traffic*** network filter, thus the way to do this is to reference it from a custom filter.

To enable traffic for TCP ports 22 and 80, two rules are added to enable this type of traffic. To allow the guest virtual machine to send ping traffic a rule is added for ICMP traffic. For simplicity reasons, general ICMP traffic will be allowed to be initiated from the guest virtual machine, and will not be specified to ICMP echo request and response messages. All other traffic will be prevented to reach or be initiated by the guest virtual machine. To do this a rule will be added that drops all other traffic. Assuming the guest virtual machine is called **test** and the interface to associate our filter with is called **eth0**, a filter is created named **test-eth0**.

The result of these considerations is the following network filter XML:

```
<filter name='test-eth0'>
  <!- - This rule references the clean traffic filter to prevent MAC, IP and ARP spoofing. By not
providing an IP address parameter, libvirt will detect the IP address the guest virtual machine is
using. - ->
  <filterref filter='clean-traffic'/>

  <!- - This rule enables TCP ports 22 (ssh) and 80 (http) to be reachable - ->
```

```
  <rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
  </rule>

  <rule action='accept' direction='in'>
    <tcp dstportstart='80'/>
  </rule>

  <!- - This rule enables general ICMP traffic to be initiated by the guest virtual machine including
ping traffic - ->
  <rule action='accept' direction='out'>
    <icmp/>
  </rule>>

  <!- - This rule enables outgoing DNS lookups using UDP - ->
  <rule action='accept' direction='out'>
    <udp dstportstart='53'/>
  </rule>

  <!- - This rule drops all other traffic - ->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

18.12.11.6. Sample custom filter

Although one of the rules in the above XML contains the IP address of the guest virtual machine as either a source or a destination address, the filtering of the traffic works correctly. The reason is that whereas the rule's evaluation occurs internally on a per-interface basis, the rules are additionally evaluated based on which (tap) interface has sent or will receive the packet, rather than what their source or destination IP address may be.

Example 18.12. Sample XML for network interface descriptions

An XML fragment for a possible network interface description inside the domain XML of the test guest virtual machine could then look like this:

```
 [...]
  <interface type='bridge'>
    <source bridge='mybridge'/>
    <filterref filter='test-eth0'/>
  </interface>
 [...]
```

To more strictly control the ICMP traffic and enforce that only ICMP echo requests can be sent from the guest virtual machine and only ICMP echo responses be received by the guest virtual machine, the above ICMP rule can be replaced with the following two rules:

```
<!- - enable outgoing ICMP echo requests- ->
<rule action='accept' direction='out'>
  <icmp type='8'/>
```

```
</rule>

<!- - enable incoming ICMP echo replies- ->
<rule action='accept' direction='in'>
  <icmp type='0'/>
</rule>
```

Example 18.13. Second example custom filter

This example demonstrates how to build a similar filter as in the example above, but extends the list of requirements with an ftp server located inside the guest virtual machine. The requirements for this filter are:

- prevents a guest virtual machine's interface from MAC, IP, and ARP spoofing
- opens only TCP ports 22 and 80 in a guest virtual machine's interface
- allows the guest virtual machine to send ping traffic from an interface but does not allow the guest virtual machine to be pinged on the interface
- allows the guest virtual machine to do DNS lookups (UDP towards port 53)
- enables the ftp server (in active mode) so it can run inside the guest virtual machine

The additional requirement of allowing an FTP server to be run inside the guest virtual machine maps into the requirement of allowing port 21 to be reachable for FTP control traffic as well as enabling the guest virtual machine to establish an outgoing TCP connection originating from the guest virtual machine's TCP port 20 back to the FTP client (FTP active mode). There are several ways of how this filter can be written and two possible solutions are included in this example.

The first solution makes use of the state attribute of the TCP protocol that provides a hook into the connection tracking framework of the Linux host physical machine. For the guest virtual machine-initiated FTP data connection (FTP active mode) the RELATED state is used to enable detection that the guest virtual machine-initiated FTP data connection is a consequence of (or 'has a relationship with') an existing FTP control connection, thereby allowing it to pass packets through the firewall. The RELATED state, however, is only valid for the very first packet of the outgoing TCP connection for the FTP data path. Afterwards, the state is ESTABLISHED, which then applies equally to the incoming and outgoing direction. All this is related to the FTP data traffic originating from TCP port 20 of the guest virtual machine. This then leads to the following solution:

```
<filter name='test-eth0'>
  <!- - This filter (eth0) references the clean traffic filter to prevent MAC, IP, and ARP spoofing. By
not providing an IP address parameter, libvirt will detect the IP address the guest virtual machine
is using. - ->
  <filterref filter='clean-traffic'/>

  <!- - This rule enables TCP port 21 (FTP-control) to be reachable - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21'/>
  </rule>

  <!- - This rule enables TCP port 20 for guest virtual machine-initiated FTP data connection
related to an existing FTP control connection - ->
  <rule action='accept' direction='out'>
```

```
  <tcp srcportstart='20' state='RELATED,ESTABLISHED'/>
</rule>

<!- - This rule accepts all packets from a client on the FTP data connection - ->
<rule action='accept' direction='in'>
  <tcp dstportstart='20' state='ESTABLISHED'/>
</rule>

<!- - This rule enables TCP port 22 (SSH) to be reachable - ->
<rule action='accept' direction='in'>
  <tcp dstportstart='22'/>
</rule>

<!- -This rule enables TCP port 80 (HTTP) to be reachable - ->
<rule action='accept' direction='in'>
  <tcp dstportstart='80'/>
</rule>

<!- - This rule enables general ICMP traffic to be initiated by the guest virtual machine, including
ping traffic - ->
<rule action='accept' direction='out'>
  <icmp/>
</rule>

<!- - This rule enables outgoing DNS lookups using UDP - ->
<rule action='accept' direction='out'>
  <udp dstportstart='53'/>
</rule>

<!- - This rule drops all other traffic - ->
<rule action='drop' direction='inout'>
  <all/>
</rule>

</filter>
```

Before trying out a filter using the RELATED state, you have to make sure that the appropriate connection tracking module has been loaded into the host physical machine's kernel. Depending on the version of the kernel, you must run either one of the following two commands before the FTP connection with the guest virtual machine is established:

- **#modprobe nf_conntrack_ftp** – where available OR
- **#modprobe ip_conntrack_ftp** if above is not available

If protocols other than FTP are used in conjunction with the RELATED state, their corresponding module must be loaded. Modules are available for the protocols: ftp, tftp, irc, sip, sctp, and amanda.

The second solution makes use of the state flags of connections more than the previous solution did. This solution takes advantage of the fact that the NEW state of a connection is valid when the very first packet of a traffic flow is detected. Subsequently, if the very first packet of a flow is accepted, the flow becomes a connection and thus enters into the ESTABLISHED state. Therefore a general rule can be written for allowing packets of ESTABLISHED connections to reach the guest virtual machine or be sent by the guest virtual machine. This is done writing specific rules for the very first

packets identified by the NEW state and dictates the ports that the data is acceptable. All packets meant for ports that are not explicitly accepted are dropped, thus not reaching an ESTABLISHED state. Any subsequent packets sent from that port are dropped as well.

```
<filter name='test-eth0'>
  <!- - This filter references the clean traffic filter to prevent MAC, IP and ARP spoofing. By not providing and IP address parameter, libvirt will detect the IP address the VM is using. - ->
  <filterref filter='clean-traffic'/>

  <!- - This rule allows the packets of all previously accepted connections to reach the guest virtual machine - ->
  <rule action='accept' direction='in'>
    <all state='ESTABLISHED'/>
  </rule>

  <!- - This rule allows the packets of all previously accepted and related connections be sent from the guest virtual machine - ->
  <rule action='accept' direction='out'>
    <all state='ESTABLISHED,RELATED'/>
  </rule>

  <!- - This rule enables traffic towards port 21 (FTP) and port 22 (SSH)- ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21' dstportend='22' state='NEW'/>
  </rule>

  <!- - This rule enables traffic towards port 80 (HTTP) - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='80' state='NEW'/>
  </rule>

  <!- - This rule enables general ICMP traffic to be initiated by the guest virtual machine, including ping traffic - ->
  <rule action='accept' direction='out'>
    <icmp state='NEW'/>
  </rule>

  <!- - This rule enables outgoing DNS lookups using UDP - ->
  <rule action='accept' direction='out'>
    <udp dstportstart='53' state='NEW'/>
  </rule>

  <!- - This rule drops all other traffic - ->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

18.12.12. Limitations

The following is a list of the currently known limitations of the network filtering subsystem.

- VM migration is only supported if the whole filter tree that is referenced by a guest virtual

machine's top level filter is also available on the target host physical machine. The network filter **clean-traffic** for example should be available on all libvirt installations and thus enable migration of guest virtual machines that reference this filter. To assure version compatibility is not a problem make sure you are using the most current version of libvirt by updating the package regularly.

- Migration must occur between libvirt installations of version 0.8.1 or later in order not to lose the network traffic filters associated with an interface.
- VLAN (802.1Q) packets, if sent by a guest virtual machine, cannot be filtered with rules for protocol IDs arp, rarp, ipv4 and ipv6. They can only be filtered with protocol IDs, MAC and VLAN. Therefore, the example filter clean-traffic Example 18.1, "An example of network filtering" will not work as expected.

18.13. CREATING TUNNELS

This section will demonstrate how to implement different tunneling scenarios.

18.13.1. Creating Multicast Tunnels

A multicast group is set up to represent a virtual network. Any guest virtual machines whose network devices are in the same multicast group can talk to each other even across host physical machines. This mode is also available to unprivileged users. There is no default DNS or DHCP support and no outgoing network access. To provide outgoing network access, one of the guest virtual machines should have a second NIC which is connected to one of the first four network types, thus providing appropriate routing. The multicast protocol is compatible with the guest virtual machine user mode. Note that the source address that you provide must be from the address used for the multicast address block.

To create a multicast tunnel, specify the following XML details into the **<devices>** element:

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

Figure 18.28. Multicast tunnel XML example

18.13.2. Creating TCP Tunnels

A TCP client/server architecture provides a virtual network. In this configuration, one guest virtual machine provides the server end of the network while all other guest virtual machines are configured as clients. All network traffic is routed between the guest virtual machine clients via the guest virtual machine server. This mode is also available for unprivileged users. Note that this mode does not provide default DNS or DHCP support nor does it provide outgoing network access. To provide outgoing network access, one of the guest virtual machines should have a second NIC which is connected to one of the first four network types thus providing appropriate routing.

To create a TCP tunnel place the following XML details into the **<devices>** element:

```
    ...
<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
  ...
  <interface type='client'>
    <mac address='52:54:00:8b:c9:51'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
</devices>
...
```

Figure 18.29. TCP tunnel domain XMl example

18.14. SETTING VLAN TAGS

virtual local area network (vLAN) tags are added using the **virsh net-edit** command. This tag can also be used with PCI device assignment with SR-IOV devices. For more information, refer to Section 9.1.7, "Configuring PCI Assignment (Passthrough) with SR-IOV Devices".

```
<network>
  <name>ovs-net</name>
  <forward mode='bridge'/>
  <bridge name='ovsbr0'/>
  <virtualport type='openvswitch'>
    <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
  </virtualport>
  <vlan trunk='yes'>
    <tag id='42' nativeMode='untagged'/>
    <tag id='47'/>
  </vlan>
  <portgroup name='dontpanic'>
    <vlan>
      <tag id='42'/>
    </vlan>
  </portgroup>
</network>
```

Figure 18.30. vSetting VLAN tag (on supported network types only)

If (and only if) the network type supports vlan tagging transparent to the guest, an optional **<vlan>** element can specify one or more vlan tags to apply to the traffic of all guests using this network. (openvswitch and type='hostdev' SR-IOV networks do support transparent VLAN tagging of guest traffic; everything else, including standard linux bridges and libvirt's own virtual networks, do not support it. 802.1Qbh (vn-link) and 802.1Qbg (VEPA) switches provide their own way (outside of libvirt) to tag guest traffic onto specific vlans.) As expected, the tag attribute specifies which vlan tag to use. If a

network has more than one **<vlan>** element defined, it is assumed that the user wants to do VLAN trunking using all the specified tags. In the case that VLAN trunking with a single tag is desired, the optional attribute trunk='yes' can be added to the VLAN element.

For network connections using openvswitch it is possible to configure the 'native-tagged' and 'native-untagged' VLAN modes. This uses the optional nativeMode attribute on the **<tag>** element: nativeMode may be set to 'tagged' or 'untagged'. The id attribute of the element sets the native vlan.

<vlan> elements can also be specified in a **<portgroup>** element, as well as directly in a domain's **<interface>** element. In the case that a vlan tag is specified in multiple locations, the setting in **<interface>** takes precedence, followed by the setting in the **<portgroup>** selected by the interface config. The **<vlan>** in **<network>** will be selected only if none is given in **<portgroup>** or **<interface>**.

18.15. APPLYING QOS TO YOUR VIRTUAL NETWORK

Quality of Service (QoS) refers to the resource control systems that guarantees an optimal experience for all users on a network, making sure that there is no delay, jitter, or packet loss. QoS can be application specific or user / group specific. Refer to Section 20.16.9.14, “Quality of service” for more information.

CHAPTER 19. QEMU-KVM COMMANDS, FLAGS, AND ARGUMENTS

19.1. INTRODUCTION

NOTE

The primary objective of this chapter is to provide a list of the **qemu-kvm** utility commands, flags, and arguments that are used as an emulator and a hypervisor in Red Hat Enterprise Linux 6. This is a comprehensive summary of the options that are known to work but are to be used at your own risk. Red Hat Enterprise Linux 6 uses KVM as an underlying virtualization technology. The machine emulator and hypervisor used is a modified version of QEMU called qemu-kvm. This version does not support all configuration options of the original QEMU and it also adds some additional options.

Options **not** listed here should not be performed.

Whitelist Format

- **<name>** - When used in a syntax description, this string should be replaced by user-defined value.
- **[a|b|c]** - When used in a syntax description, only one of the strings separated by | is used.
- When no comment is present, an option is supported with all possible values.

19.2. BASIC OPTIONS

This section provides information about the basic options.

Emulated Machine

-M <machine-type>

-machine <machine-type>**[,<property>[=<value>][,..]]**

Processor Type

-cpu <model>[,<FEATURE>][...]

Additional models are visible by running **-cpu ?** command.

- **Opteron_G5** - AMD Opteron 63xx class CPU
- **Opteron_G4** - AMD Opteron 62xx class CPU
- **Opteron_G3** - AMD Opteron 23xx (AMD Opteron Gen 3)
- **Opteron_G2** - AMD Opteron 22xx (AMD Opteron Gen 2)
- **Opteron_G1** - AMD Opteron 240 (AMD Opteron Gen 1)
- **Westmere** - Westmere E56xx/L56xx/X56xx (Nehalem-C)
- **Haswell** - Intel Core Processor (Haswell)

- **SandyBridge** - Intel Xeon E312xx (Sandy Bridge)
- **Nehalem** - Intel Core i7 9xx (Nehalem Class Core i7)
- **Penryn** - Intel Core 2 Duo P9xxx (Penryn Class Core 2)
- **Conroe** - Intel Celeron_4x0 (Conroe/Merom Class Core 2)
- **cpu64-rhel5** - Red Hat Enterprise Linux 5 supported QEMU Virtual CPU version
- **cpu64-rhel6** - Red Hat Enterprise Linux 6 supported QEMU Virtual CPU version
- **default** - special option use default option from above.

Processor Topology

-smp <n>[,cores=<ncores>][,threads=<nthreads>][,sockets=<nsocks>][,maxcpus=<maxcpus>]

Hypervisor and guest operating system limits on processor topology apply.

NUMA System

-numa <nodes>[,mem=<size>][,cpus=<cpu[-cpu>]][,nodeid=<node>]

Hypervisor and guest operating system limits on processor topology apply.

Memory Size

-m <megs>

Supported values are limited by guest minimal and maximal values and hypervisor limits.

Keyboard Layout

-k <language>

Guest Name

-name <name>

Guest UUID

-uuid <uuid>

19.3. DISK OPTIONS

This section provides information about disk options.

Generic Drive

-drive <option>[,<option>[,<option>[,...]]]

Supported with the following options:

- **readonly**[on|off]
- **werror**[enospc|report|stop|ignore]
- **rerror**[report|stop|ignore]
- **id**=<id>

  Id of the drive has the following limitation for if=none:

  - IDE disk has to have <id> in following format: drive-ide0-<BUS>-<UNIT>

    Example of correct format:

    -drive if=none,id=drive-ide0-<BUS>-<UNIT>,... -device ide-drive,drive=drive-ide0-<BUS>-<UNIT>,bus=ide.<BUS>,unit=<UNIT>

- **file**=<file>

  Value of <file> is parsed with the following rules:

  - Passing floppy device as <file> is not supported.
  - Passing cd-rom device as <file> is supported only with cdrom media type (media=cdrom) and only as IDE drive (either if=ide or if=none + -device ide-drive).
  - If <file> is neither block nor character device, it must not contain ':'.

- **if**=<interface>

  The following interfaces are supported: none, ide, virtio, floppy.

- **index**=<index>
- **media**=<media>
- **cache**=<cache>

  Supported values: none, writeback or writethrough.

- **copy-on-read**=[on|off]
- **snapshot**=[yes|no]
- **serial**=<serial>
- **aio**=<aio>
- **format**=<format>

  This option is not required and can be omitted. However, this is not recommended for raw images because it represents security risk. Supported formats are:

  - **qcow2**
  - **raw**

Boot Option

-boot [order=<drives>][,menu=[on|off]]

Snapshot Mode

-snapshot

19.4. DISPLAY OPTIONS

This section provides information about display options.

Disable Graphics
-nographic

VGA Card Emulation
-vga <type>

Supported types:

- **cirrus** - Cirrus Logic GD5446 Video card.
- **std** - Standard VGA card with Bochs VBE extensions.
- **qxl** - Spice paravirtual card.
- **none** - Disable VGA card.

VNC Display
-vnc <display>[,<option>[,<option>[,...]]]

Supported display value:

- [<host>]:<port>
- unix:<path>
- **share**[allow-exclusive|force-shared|ignore]
- none - Supported with no other options specified.

Supported options are:

- **to**=<port>
- **reverse**
- **password**
- **tls**
- **x509**=</path/to/certificate/dir> - Supported when **tls** specified.
- **x509verify**=</path/to/certificate/dir> - Supported when **tls** specified.
- **sasl**
- **acl**

Spice Desktop
-spice option[,option[,...]]

Supported options are:

- **port**=<number>
- **addr**=<addr>
- **ipv4**

ipv6

- **password**=<secret>
- **disable-ticketing**
- **disable-copy-paste**
- **tls-port**=<number>
- **x509-dir**=</path/to/certificate/dir>
- **x509-key-file**=<file>

  x509-key-password=<file>

  x509-cert-file=<file>

  x509-cacert-file=<file>

  x509-dh-key-file=<file>
- **tls-cipher**=<list>
- **tls-channel**[main|display|cursor|inputs|record|playback]

  plaintext-channel[main|display|cursor|inputs|record|playback]
- **image-compression** =<compress>
- **jpeg-wan-compression** =<value>

  zlib-glz-wan-compression =<value>
- **streaming-video**=[off|all|filter]
- **agent-mouse**=[on|off]
- **playback-compression**=[on|off]
- **seamless-migratio**=[on|off]

19.5. NETWORK OPTIONS

This section provides information about network options.

TAP network

-netdev tap,id=<id>][,<options>...]

The following options are supported (all use name=value format):

- ifname
- fd
- script

- downscript
- sndbuf
- vnet_hdr
- vhost
- vhostfd
- vhostforce

19.6. DEVICE OPTIONS

This section provides information about device options.

General Device

-device <driver>[,<prop>[=<value>][,...]]

All drivers support following properties

- id
- bus

Following drivers are supported (with available properties):

- **pci-assign**
  - host
  - bootindex
  - configfd
  - addr
  - rombar
  - romfile
  - multifunction

  If the device has multiple functions, all of them need to be assigned to the same guest.
- **rtl8139**
  - mac
  - netdev
  - bootindex
  - addr
- **e1000**

  - mac
  - netdev
  - bootindex
  - addr
- **virtio-net-pci**
  - ioeventfd
  - vectors
  - indirect
  - event_idx
  - csum
  - guest_csum
  - gso
  - guest_tso4
  - guest_tso6
  - guest_ecn
  - guest_ufo
  - host_tso4
  - host_tso6
  - host_ecn
  - host_ufo
  - mrg_rxbuf
  - status
  - ctrl_vq
  - ctrl_rx
  - ctrl_vlan
  - ctrl_rx_extra
  - mac
  - netdev
  - bootindex

  - x-txtimer
  - x-txburst
  - tx
  - addr
- **qxl**
  - ram_size
  - vram_size
  - revision
  - cmdlog
  - addr
- **ide-drive**
  - unit
  - drive
  - physical_block_size
  - bootindex
  - ver
  - wwn
- **virtio-blk-pci**
  - class
  - drive
  - logical_block_size
  - physical_block_size
  - min_io_size
  - opt_io_size
  - bootindex
  - ioeventfd
  - vectors
  - indirect_desc
  - event_idx

  - scsi
  - addr
- **virtio-scsi-pci** - Technology Preview in 6.3, supported since 6.4.

  For Windows guests, Windows Server 2003, which was Technology Preview, is no longer supported since 6.5. However, Windows Server 2008 and 2012, and Windows desktop 7 and 8 are fully supported since 6.5.

  - vectors
  - indirect_desc
  - event_idx
  - num_queues
  - addr
- **isa-debugcon**
- **isa-serial**
  - index
  - iobase
  - irq
  - chardev
- **virtserialport**
  - nr
  - chardev
  - name
- **virtconsole**
  - nr
  - chardev
  - name
- **virtio-serial-pci**
  - vectors
  - class
  - indirect_desc
  - event_idx

  - max_ports
  - flow_control
  - addr
- **ES1370**
  - addr
- **AC97**
  - addr
- **intel-hda**
  - addr
- **hda-duplex**
  - cad
- **hda-micro**
  - cad
- **hda-output**
  - cad
- **i6300esb**
  - addr
- **ib700** - no properties
- **sga** - no properties
- **virtio-balloon-pci**
  - indirect_desc
  - event_idx
  - addr
- **usb-tablet**
  - migrate
  - port
- **usb-kbd**
  - migrate
  - port

- **usb-mouse**
  - migrate
  - port
- **usb-ccid** - supported since 6.2
  - port
  - slot
- **usb-host** - Technology Preview since 6.2
  - hostbus
  - hostaddr
  - hostport
  - vendorid
  - productid
  - isobufs
  - port
- **usb-hub** - supported since 6.2
  - port
- **usb-ehci** - Technology Preview since 6.2
  - freq
  - maxframes
  - port
- **usb-storage** - Technology Preview since 6.2
  - drive
  - bootindex
  - serial
  - removable
  - port
- **usb-redir** - Technology Preview for 6.3, supported since 6.4
  - chardev
  - filter

- **scsi-cd** - Technology Preview for 6.3, supported since 6.4
  - drive
  - logical_block_size
  - physical_block_size
  - min_io_size
  - opt_io_size
  - bootindex
  - ver
  - serial
  - scsi-id
  - lun
  - channel-scsi
  - wwn
- **scsi-hd** -Technology Preview for 6.3, supported since 6.4
  - drive
  - logical_block_size
  - physical_block_size
  - min_io_size
  - opt_io_size
  - bootindex
  - ver
  - serial
  - scsi-id
  - lun
  - channel-scsi
  - wwn
- **scsi-block** -Technology Preview for 6.3, supported since 6.4
  - drive
  - bootindex

- **scsi-disk** -Technology Preview for 6.3
  - drive=drive
  - logical_block_size
  - physical_block_size
  - min_io_size
  - opt_io_size
  - bootindex
  - ver
  - serial
  - scsi-id
  - lun
  - channel-scsi
  - wwn
- **piix3-usb-uhci**
- **piix4-usb-uhci**
- **ccid-card-passthru**

Global Device Setting

-global <device>.<property>=<value>

Supported devices and properties as in "General device" section with these additional devices:

- **isa-fdc**
  - driveA
  - driveB
  - bootindexA
  - bootindexB
- **qxl-vga**
  - ram_size
  - vram_size
  - revision
  - cmdlog
  - addr

Character Device
-chardev back end,id=<id>[,<options>]

Supported back ends are:

- **null**,id=<id> - null device
- **socket**,id=<id>,port=<port>[,host=<host>][,to=<to>][,ipv4][,ipv6][,nodelay][,server][,nowait][,telnet] - tcp socket
- **socket**,id=<id>,path=<path>[,server][,nowait][,telnet] - unix socket
- **file**,id=<id>,path=<path> - trafit to file.
- **stdio**,id=<id> - standard i/o
- **spicevmc**,id=<id>,name=<name> - spice channel

Enable USB
-usb

19.7. LINUX/MULTIBOOT BOOT

This section provides information about Linux and multiboot booting.

Kernel File
-kernel <bzImage>

Note: multiboot images are not supported

Ram Disk
-initrd <file>

Command Line Parameter
-append <cmdline>

19.8. EXPERT OPTIONS

This section provides information about expert options.

KVM Virtualization
-enable-kvm

QEMU-KVM supports only KVM virtualization and it is used by default if available. If -enable-kvm is used and KVM is not available, qemu-kvm fails. However, if -enable-kvm is not used and KVM is not available, **qemu-kvm** runs in TCG mode, which is not supported.

Disable Kernel Mode PIT Reinjection
-no-kvm-pit-reinjection

No Shutdown
-no-shutdown

No Reboot
-no-reboot

Serial Port, Monitor, QMP

-serial <dev>

-monitor <dev>

-qmp <dev>

Supported devices are:

- **stdio** - standard input/output
- **null** - null device
- **file**:<filename> - output to file.
- **tcp**:[<host>]:<port>[,server][,nowait][,nodelay] - TCP Net console.
- **unix**:<path>[,server][,nowait] - Unix domain socket.
- **mon**:<dev_string> - Any device above, used to multiplex monitor too.
- **none** - disable, valid only for -serial.
- **chardev**:<id> - character device created with -chardev.

Monitor Redirect

-mon <chardev_id>[,mode=[readline|control]][,default=[on|off]]

Manual CPU Start

-S

RTC

-rtc [base=utc|localtime|date][,clock=host|vm][,driftfix=none|slew]

Watchdog

-watchdog model

Watchdog Reaction

-watchdog-action <action>

Guest Memory Backing

-mem-prealloc -mem-path /dev/hugepages

SMBIOS Entry

-smbios type=0[,vendor=<str>][,<version=str>][,date=<str>][,release=%d.%d]

-smbios type=1[,manufacturer=<str>][,product=<str>][,version=<str>][,serial=<str>][,uuid=<uuid>][,sku=<str>][,family=<str>]

19.9. HELP AND INFORMATION OPTIONS

This section provides information about help and information options.

Help

-h

-help

Version
-version

Audio Help
-audio-help

19.10. MISCELLANEOUS OPTIONS

This section provides information about miscellaneous options.

Migration
-incoming

No Default Configuration
-nodefconfig

-nodefaults

Running without -nodefaults is not supported

Device Configuration File
-readconfig <file>

-writeconfig <file>

Loaded Saved State
-loadvm <file>

CHAPTER 20. MANIPULATING THE DOMAIN XML

This section describes the XML format used to represent domains. Here the term *domain* refers to the root **<domain>** element required for all guest virtual machine. The domain XML has two attributes: **type** specifies the hypervisor used for running the domain. The allowed values are driver specific, but include **KVM** and others. **id** is a unique integer identifier for the running guest virtual machine. Inactive machines have no id value. The sections in this chapter will address the components of the domain XML. Additional chapters in this manual may refer to this chapter when manipulation of the domain XML is required.

NOTE

This chapter is based on the libvirt upstream documentation.

20.1. GENERAL INFORMATION AND METADATA

This information is in this part of the domain XML:

```
<domain type='xen' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <title>A short description - title - of the domain</title>
  <description>Some human readable description</description>
  <metadata>
    <app1:foo xmlns:app1="http://app1.org/app1/">..</app1:foo>
    <app2:bar xmlns:app2="http://app1.org/app2/">..</app2:bar>
  </metadata>
  ...
</domain>
```

Figure 20.1. Domain XML metadata

The components of this section of the domain XML are as follows:

Table 20.1. General metadata elements

Element	Description
<name>	Assigns a name for the virtual machine. This name should consist only of alpha-numeric characters and is required to be unique within the scope of a single host physical machine. It is often used to form the filename for storing the persistent configuration files.
<uuid>	assigns a globally unique identifier for the virtual machine. The format must be RFC 4122-compliant, eg **3e3fce45-4f53-4fa7-bb32-11f34168b82b**. If omitted when defining/creating a new machine, a random UUID is generated. It is also possible to provide the UUID with a sysinfo specification.

Element	Description
<title>	**title** Creates space for a short description of the domain. The title should not contain any newlines.
<description>	Different from the title, This data is not used by libvirt in any way, it can contain any information the user wants to display.
<metadata>	Can be used by applications to store custom metadata in the form of XML nodes/trees. Applications must use custom namespaces on their XML nodes/trees, with only one top-level element per namespace (if the application needs structure, they should have sub-elements to their namespace element)

20.2. OPERATING SYSTEM BOOTING

There are a number of different ways to boot virtual machines each with their own pros and cons. Each one is described in the sub-sections that follow and include: BIOS boot loader, host physical machine boot loader, and direct kernel boot.

20.2.1. BIOS Boot loader

Booting through the BIOS is available for hypervisors supporting full virtualization. In this case the BIOS has a boot order priority (floppy, harddisk, cdrom, network) determining where to obtain/find the boot image. The OS section of the domain XML contains the information as follows:

```
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <boot dev='hd'/>
  <boot dev='cdrom'/>
  <bootmenu enable='yes'/>
  <smbios mode='sysinfo'/>
  <bios useserial='yes' rebootTimeout='0'/>
</os>
...
```

Figure 20.2. BIOS boot loader domain XML

The components of this section of the domain XML are as follows:

Table 20.2. BIOS boot loader elements

Element	Description

Element	Description
<type>	Specifies the type of operating system to be booted on the guest virtual machine. **hvm** indicates that the OS is one designed to run on bare metal, so requires full virtualization. **linux** refers to an OS that supports the Xen 3 hypervisor guest ABI. There are also two optional attributes, **arch** specifying the CPU architecture to virtualization, and **machine** referring to the machine type. Refer to *Driver Capabilities* for more information.
<loader>	refers to a piece of firmware that is used to assist the domain creation process. It is only needed for using Xen fully virtualized domains.
<boot>	takes one of the values**fd**, **hd**, **cdrom** or **network** and is used to specify the next boot device to consider. The boot element can be repeated multiple times to setup a priority list of boot devices to try in turn. Multiple devices of the same type are sorted according to their targets while preserving the order of buses. After defining the domain, its XML configuration returned by libvirt (through virDomainGetXMLDesc) lists devices in the sorted order. Once sorted, the first device is marked as bootable. For more information see *BIOS bootloader*.
<bootmenu>	determines whether or not to enable an interactive boot menu prompt on guest virtual machine startup. The **enable** attribute can be either**yes** or **no**. If not specified, the hypervisor default is used
<smbios>	determines how SMBIOS information is made visible in the guest virtual machine. The **mode** attribute must be specified, as either **emulate** (lets the hypervisor generate all values), **host**(copies all of Block 0 and Block 1, except for the UUID, from the host physical machine's SMBIOS values; the virConnectGetSysinfo call can be used to see what values are copied), or **sysinfo** (uses the values in the sysinfo element). If not specified, the hypervisor default setting is used.

Element	Description
<bios>	This element has attribute **useserial** with possible values **yes** or **no**. The attribute enables or disables Serial Graphics Adapter which allows users to see BIOS messages on a serial port. Therefore, one needs to have serial port defined. Note there is another attribute, **rebootTimeout** that controls whether and after how long the guest virtual machine should start booting again in case the boot fails (according to BIOS). The value is in milliseconds with maximum of **65535** and special value **-1** disables the reboot.

20.2.2. Host Physical Machine Boot Loader

Hypervisors employing paravirtualization do not usually emulate a BIOS, but instead the host physical machine is responsible for the operating system boot. This may use a pseudo-bootloader in the host physical machine to provide an interface to choose a kernel for the guest virtual machine. An example is pygrub with Xen.

```
...
<bootloader>/usr/bin/pygrub</bootloader>
<bootloader_args>--append single</bootloader_args>
...
```

Figure 20.3. Host physical machine boot loader domain XML

The components of this section of the domain XML are as follows:

Table 20.3. BIOS boot loader elements

Element	Description
<bootloader>	provides a fully qualified path to the boot loader executable in the host physical machine OS. This boot loader will choose which kernel to boot. The required output of the boot loader is dependent on the hypervisor in use.
<bootloader_args>	allows command line arguments to be passed to the boot loader (optional command)

20.2.3. Direct kernel boot

When installing a new guest virtual machine OS, it is often useful to boot directly from a kernel and initrd stored in the host physical machine OS, allowing command line arguments to be passed directly to the installer. This capability is usually available for both para and full virtualized guest virtual machines.

```
...
<os>
  <type>hvm</type>
  <loader>/usr/lib/xen/boot/hvmloader</loader>
  <kernel>/root/f8-i386-vmlinuz</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-i386/os/</cmdline>
  <dtb>/root/ppc.dtb</dtb>
</os>
...
```

Figure 20.4. Direct Kernel Boot

The components of this section of the domain XML are as follows:

Table 20.4. Direct kernel boot elements

Element	Description
<type>	same as described in the BIOS boot section
<loader>	same as described in the BIOS boot section
<kernel>	specifies the fully-qualified path to the kernel image in the host physical machine OS
<initrd>	specifies the fully-qualified path to the (optional) ramdisk image in the host physical machine OS.
<cmdline>	specifies arguments to be passed to the kernel (or installer) at boot time. This is often used to specify an alternate primary console (eg serial port), or the installation media source / kickstart file

20.3. SMBIOS SYSTEM INFORMATION

Some hypervisors allow control over what system information is presented to the guest virtual machine (for example, SMBIOS fields can be populated by a hypervisor and inspected using the **midecode** command in the guest virtual machine). The optional sysinfo element covers all such categories of information.

```
...
<os>
  <smbios mode='sysinfo'/>
  ...
</os>
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
  </bios>
  <system>
    <entry name='manufacturer'>Fedora</entry>
    <entry name='vendor'>Virt-Manager</entry>
  </system>
</sysinfo>
...
```

Figure 20.5. SMBIOS system information

The **<sysinfo>** element has a mandatory attribute **type** that determines the layout of sub-elements, and may be defined as follows:

- **smbios** - Sub-elements call out specific SMBIOS values, which will affect the guest virtual machine if used in conjunction with the smbios sub-element of the **<os>** element. Each sub-element of sysinfo names a SMBIOS block, and within those elements can be a list of entry elements that describe a field within the block. The following blocks and entries are recognized:
  - **bios** - This is block 0 of SMBIOS, with entry names drawn from **vendor**, **version**, **date**, and **release**.
  - **<system>** - This is block 1 of SMBIOS, with entry names drawn from **manufacturer**, **product**, **version**, **serial**, **uuid**, **sku**, and **family**. If a **uuid** entry is provided alongside a top-level uuid element, the two values must match.

20.4. CPU ALLOCATION

```
<domain>
  ...
  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
  ...
</domain>
```

Figure 20.6. CPU allocation

The **<cpu>** element defines the maximum number of virtual CPUs (vCPUs) allocated for the guest virtual machine operating system, which must be between 1 and the maximum supported by the hypervisor. This element can contain an optional **cpuset** attribute, which is a comma-separated list of physical CPU numbers that domain processes and virtual CPUs can be pinned to by default.

Note that the pinning policy of domain processes and virtual CPUs can be specified separately by using the **cputune** attribute. If the **emulatorpin** attribute is specified in **<cputune>**, the **cpuset** value specified by **<vcpu>** will be ignored.

Similarly, virtual CPUs that have set a value for **vcpupin** cause **cpuset** settings to be ignored. Virtual

CPUs where **vcpupin** is not specified will be pinned to the physical CPUs specified by **cpuset**. Each element in the **cpuset** list is either a single CPU number, a range of CPU numbers, or a caret (^) followed by a CPU number to be excluded from a previous range. The attribute **current** can be used to specify whether fewer than the maximum number of virtual CPUs should be enabled.

The optional attribute **placement** can be used to specify the CPU placement mode for the domain process. **placement** can be set as either **static** or **auto**. If you set **<vcpu placement='auto'>**, the system will query numad and use the settings specified in the **<numatune>** tag, and ignore any other settings in **<vcpu>** . If you set **<vcpu placement='static'>**, the system will use the settings specified in the **<vcpu placement>** tag instead of the settings in **<numatune>**.

20.5. CPU TUNING

```
<domain>
  ...
  <cputune>
    <vcpupin vcpu="0" cpuset="1-4,^2"/>
    <vcpupin vcpu="1" cpuset="0,1"/>
    <vcpupin vcpu="2" cpuset="2,3"/>
    <vcpupin vcpu="3" cpuset="0,4"/>
    <emulatorpin cpuset="1-3"/>
    <shares>2048</shares>
    <period>1000000</period>
    <quota>-1</quota>
    <emulator_period>1000000</emulator_period>
    <emulator_quota>-1</emulator_quota>
  </cputune>
  ...
</domain>
```

Figure 20.7. CPU tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.5. CPU tuning elements

Element	Description
<cputune>	Provides details regarding the CPU tunable parameters for the domain. This is optional.
<vcpupin>	Specifies which of host physical machine's physical CPUs the domain VCPU will be pinned to. If this is omitted, and attribute **cpuset** of element **<vcpu>** is not specified, the vCPU is pinned to all the physical CPUs by default. It contains two required attributes, the attribute **vcpu** specifies **id**, and the attribute **cpuset** is same as attribute **cpuset** of element **<vcpu>**.

Element	Description
<emulatorpin>	Specifies which of the host physical machine CPUs, the "emulator", a subset of a domains not including vcpu, will be pinned to. If this is omitted, and attribute **cpuset** of element **<vcpu>** is not specified, the "emulator" is pinned to all the physical CPUs by default. It contains one required attribute **cpuset** specifying which physical CPUs to pin to. **emulatorpin** is not allowed if attribute **placement** of element **<vcpu>** is **auto**.
<shares>	Specifies the proportional weighted share for the domain. If this is omitted, it defaults to the default value inherent in the operating system. If there is no unit for the value, it is calculated relative to the setting of other guest virtual machines. For example, if a guest virtual machine is configured with value of 2048, it will get twice as much processing time as a guest virtual machine configured with value of 1024.
<period>	Specifies the enforcement interval in microseconds. By using **period**, each of the domain's vcpu will not be allowed to consume more than its allotted quota worth of run time. This value should be within the following range: **1000-1000000**. A **period>** with a value of **0** means no value.
<quota>	Specifies the maximum allowed bandwidth in microseconds. A domain with **quota** as any negative value indicates that the domain has infinite bandwidth, which means that it is not bandwidth controlled. The value should be within the following range:**1000 - 18446744073709551** or less than **0**. A **quota** with value of **0** means no value. You can use this feature to ensure that all vcpus run at the same speed.
<emulator_period>	Specifies the enforcement interval in microseconds. Within an **<emulator_period>**, emulator threads (those excluding vcpus) of the domain will not be allowed to consume more than the **<emulator_quota>** worth of run time. The **<emulator_period>** value should be in the following range: **1000 - 1000000**. An **<emulator_period>** with value of **0**, means no value.

Element	Description
<emulator_quota>	Specifies the maximum allowed bandwidth in microseconds for the domain's emulator threads (those excluding vcpus). A domain with an **<emulator_quota>** as a negative value indicates that the domain has infinite bandwidth for emulator threads (those excluding vcpus), which means that it is not bandwidth controlled. The value should be in the following range: **1000 - 18446744073709551**, or less than **0**. An **<emulator_quota>** with value **0** means no value.

20.6. MEMORY BACKING

Memory backing allows the hypervisor to properly manage large pages within the guest virtual machine.

The optional **<memoryBacking>** element may have an **<hugepages>** element set within it. This tells the hypervisor that the guest virtual machine should have its memory allocated using hugepages instead of the normal native page size.

```
<domain>
  ...
  <memoryBacking>
    <hugepages/>
  </memoryBacking>
  ...
</domain>
```

Figure 20.8. Memory backing

20.7. MEMORY TUNING

```
<domain>
  ...
  <memtune>
    <hard_limit unit='G'>1</hard_limit>
    <soft_limit unit='M'>128</soft_limit>
    <swap_hard_limit unit='G'>2</swap_hard_limit>
    <min_guarantee unit='bytes'>67108864</min_guarantee>
  </memtune>
  ...
</domain>
```

Figure 20.9. Memory Tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.6. Memory tuning elements

Element	Description
<memtune>	Provides details regarding the memory tunable parameters for the domain. If this is omitted, it defaults to the OS provided defaults. The parameters are applied to the process as a whole therefore when setting limits, one needs to add up guest virtual machine RAM, guest virtual machine video RAM, and allow for some memory overhead. The last piece is hard to determine so one use trial and error. For each tunable, it is possible to designate which unit the number is in on input, using the same values as for **<memory>**. For backwards compatibility, output is always in KiB.
<hard_limit>	This is the maximum memory the guest virtual machine can use. The **unit** for this value is expressed in **kibibytes** (blocks of 1024 bytes)
<soft_limit>	This is the memory limit to enforce during memory contention. The **unit** for this value is expressed in kibibytes (blocks of 1024 bytes)
<swap_hard_limit>	This is the maximum memory plus swap the guest virtual machine can use. The **unit** for this value is expressed in kibibytes (blocks of 1024 bytes). This has to be more than **<hard_limit>** value provided
<min_guarantee>	This is the guaranteed minimum memory allocation for the guest virtual machine. The units for this value is expressed in kibibytes (blocks of 1024 bytes)

20.8. NUMA NODE TUNING

Once NUMA node tuning is done using conventional management tools the following domain XML parameters are effected:

```
>
<domain>
  ...
  <numatune>
    <memory mode="strict" nodeset="1-4,^3"/>
  </numatune>
  ...
</domain>
```

Figure 20.10. NUMA node tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.7. NUMA node tuning elements

Element	Description
<numatune>	Provides details of how to tune the performance of a NUMA host physical machine through controlling NUMA policy for domain process.
<memory>	Specifies how to allocate memory for the domain process on a NUMA host physical machine. It contains several optional attributes. Attribute **mode** is either **interleave**, **strict**, or **preferred**. If no value is given it defaults to **strict**. Attribute **nodeset** specifies the NUMA nodes, using the same syntax as attribute **cpuset** of element **<vcpu>**. Attribute **placement** can be used to indicate the memory placement mode for the domain process. Its value can be either **static** or **auto**. If attribute **<nodeset>** is specified it defaults to the **<placement>** of **<vcpu>**, or **static**. **auto** indicates the domain process will only allocate memory from the advisory nodeset returned from querying numad and the value of attribute nodeset will be ignored if it is specified. If attribute **placement** of **vcpu** is **auto**, and attribute **<numatune>** is not specified, a default numatune with **<placement>** **auto** and mode **strict** will be added implicitly.

20.9. BLOCK I/O TUNING

```
<domain>
  ...
  <blkiotune>
    <weight>800</weight>
    <device>
      <path>/dev/sda</path>
      <weight>1000</weight>
    </device>
    <device>
      <path>/dev/sdb</path>
      <weight>500</weight>
    </device>
  </blkiotune>
  ...
</domain>
```

Figure 20.11. Block I/O Tuning

Although all are optional, the components of this section of the domain XML are as follows:

Table 20.8. Block I/O tuning elements

Element	Description
<blkiotune>	This optional element provides the ability to tune Blkio cgroup tunable parameters for the domain. If this is omitted, it defaults to the OS provided defaults.
<weight>	This optional weight element is the overall I/O weight of the guest virtual machine. The value should be within the range 100 - 1000.
<device>	The domain may have multiple **<device>** elements that further tune the weights for each host physical machine block device in use by the domain. Note that multiple guest virtual machine disks can share a single host physical machine block device. In addition, as they are backed by files within the same host physical machine file system, this tuning parameter is at the global domain level, rather than being associated with each guest virtual machine disk device (contrast this to the **<iotune>** element which can be applied to a single **<disk>**). Each device element has two mandatory sub-elements, **<path>** describing the absolute path of the device, and **<weight>** giving the relative weight of that device, which has an acceptable range of 100 - 1000.

20.10. RESOURCE PARTITIONING

Hypervisors may allow for virtual machines to be placed into resource partitions, potentially with nesting of said partitions. The **<resource>** element groups together configuration related to resource partitioning. It currently supports a child element partition whose content defines the path of the resource partition in which to place the domain. If no partition is listed, then the domain will be placed in a default partition. It is the responsibility of the app/admin to ensure that the partition exists prior to starting the guest virtual machine. Only the (hypervisor specific) default partition can be assumed to exist by default.

```
<resource>
  <partition>/virtualmachines/production</partition>
 </resource>
```

Figure 20.12. Resource partitioning

Resource partitions are currently supported by the QEMU and LXC drivers, which map partition paths to cgroups directories in all mounted controllers.

20.11. CPU MODEL AND TOPOLOGY

This section covers the requirements for CPU model. Note that every hypervisor has its own policy for which CPU features guest will see by default. The set of CPU features presented to the guest by QEMU/KVM depends on the CPU model chosen in the guest virtual machine configuration. **qemu32**

and **qemu64** are basic CPU models but there are other models (with additional features) available. Each model and its topology is specified using the following elements from the domain XML:

```
<cpu match='exact'>
   <model fallback='allow'>core2duo</model>
   <vendor>Intel</vendor>
   <topology sockets='1' cores='2' threads='1'/>
   <feature policy='disable' name='lahf_lm'/>
  </cpu>
```

Figure 20.13. CPU model and topology example 1

```
<cpu mode='host-model'>
  <model fallback='forbid'/>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

Figure 20.14. CPU model and topology example 2

```
<cpu mode='host-passthrough'/>
```

Figure 20.15. CPU model and topology example 3

In cases where no restrictions are to be put on either the CPU model nor its features, a simpler cpu element such as the following may be used.

```
<cpu>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

Figure 20.16. CPU model and topology example 4

The components of this section of the domain XML are as follows:

Table 20.9. CPU model and topology elements

Element	Description
<cpu>	This element contains all parameters for the vCPU feature set.

Element	Description
<match>	Specifies how closely the features indicated in the **<cpu>** element must match the vCPUs that are available. The **match** attribute can be omitted if **<topology>** is the only element nested in the **<cpu>** element. Possible values for the **match** attribute are: • **minimum** - The features listed are the minimum requirement. There may be more features available in the vCPU then are indicated, but this is the minimum that will be accepted. This value will fail if the minimum requirements are not met. • **exact** - the virtual CPU provided to the guest virtual machine must exactly match the features specified. If no match is found, an error will result. • **strict** - the guest virtual machine will not be created unless the host physical machine CPU exactly matches the specification. If the **match** attribute is omitted from the **<cpu>** element, the default setting **match='exact'** is used.

Element	Description
<mode>	This optional attribute may be used to make it easier to configure a guest virtual machine CPU to be as close to the host physical machine CPU as possible. Possible values for the mode attribute are: • **custom** - describes how the CPU is presented to the guest virtual machine. This is the default setting when the **mode** attribute is not specified. This mode makes it so that a persistent guest virtual machine will see the same hardware no matter what host physical machine the guest virtual machine is booted on. • **host-model** - this is essentially a shortcut to copying host physical machine CPU definition from the capabilities XML into the domain XML. As the CPU definition is copied just before starting a domain, the same XML can be used on different host physical machines while still providing the best guest virtual machine CPU each host physical machine supports. Neither the **match** attribute nor any feature elements can be used in this mode. For more information see libvirt domain XML CPU models • **host-passthrough** With this mode, the CPU visible to the guest virtual machine is exactly the same as the host physical machine CPU including elements that cause errors within libvirt. The obvious the downside of this mode is that the guest virtual machine environment cannot be reproduced on different hardware and therefore this mode is recommended with great caution. Neither **model** nor **feature** elements are allowed in this mode. • Note that in both **host-model** and **host-passthrough** mode, the real (approximate in host-passthrough mode) CPU definition which would be used on current host physical machine can be determined by specifying VIR_DOMAIN_XML_UPDATE_CPU flag when calling virDomainGetXMLDesc API. When running a guest virtual machine that might be prone to operating system reactivation when presented with different hardware, and which will be migrated between host physical machines with different capabilities, you can use this output to rewrite XML to the custom mode for more robust migration.

Element	Description
<model>	Specifies CPU model requested by the guest virtual machine. The list of available CPU models and their definition can be found in **cpu_map.xml** file installed in libvirt's data directory. If a hypervisor is not able to use the exact CPU model, libvirt automatically falls back to a closest model supported by the hypervisor while maintaining the list of CPU features. An optional **fallback** attribute can be used to forbid this behavior, in which case an attempt to start a domain requesting an unsupported CPU model will fail. Supported values for fallback attribute are: **allow** (this is the default), and **forbid**. The optional **vendor_id** attribute can be used to set the vendor id seen by the guest virtual machine. It must be exactly 12 characters long. If not set, the vendor id of the host physical machine is used. Typical possible values are **AuthenticAMD** and **GenuineIntel**.
<vendor>	Specifies CPU vendor requested by the guest virtual machine. If this element is missing, the guest virtual machine runs on a CPU matching given features regardless of its vendor. The list of supported vendors can be found in **cpu_map.xml**.
<topology>	Specifies requested topology of virtual CPU provided to the guest virtual machine. Three non-zero values have to be given for sockets, cores, and threads: total number of CPU sockets, number of cores per socket, and number of threads per core, respectively.

Element	Description
<feature>	Can contain zero or more elements used to fine-tune features provided by the selected CPU model. The list of known feature names can be found in the same file as CPU models. The meaning of each feature element depends on its policy attribute, which has to be set to one of the following values: • **force** - forces the virtual to be supported regardless of whether it is actually supported by host physical machine CPU. • **require** - dictates that guest virtual machine creation will fail unless the feature is supported by host physical machine CPU. This is the default setting • **optional** - this feature is supported by virtual CPU but and only if it is supported by host physical machine CPU. • **disable** - this is not supported by virtual CPU. • **forbid** - guest virtual machine creation will fail if the feature is supported by host physical machine CPU.

20.11.1. Guest virtual machine NUMA topology

Guest virtual machine NUMA topology can be specified using the **<numa>** element and the following from the domain XML:

```
<cpu>
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
</cpu>
...
```

Figure 20.17. Guest Virtual Machine NUMA Topology

Each cell element specifies a NUMA cell or a NUMA node. **cpus** specifies the CPU or range of CPUs that are part of the node. **memory** specifies the node memory in kibibytes (blocks of 1024 bytes). Each cell or node is assigned **cellid** or **nodeid** in increasing order starting from 0.

20.12. EVENTS CONFIGURATION

Using the following sections of domain XML it is possible to override the default actions taken on various events.

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<on_lockfailure>poweroff</on_lockfailure>
```

Figure 20.18. Events configuration

The following collections of elements allow the actions to be specified when a guest virtual machine OS triggers a life cycle operation. A common use case is to force a reboot to be treated as a poweroff when doing the initial OS installation. This allows the VM to be re-configured for the first post-install bootup.

The components of this section of the domain XML are as follows:

Table 20.10. Event configuration elements

State	Description
<on_poweroff>	Specifies the action that is to be executed when the guest virtual machine requests a poweroff. Four possible arguments are possible: • **destroy** - this action terminates the domain completely and releases all resources • **restart** - this action terminates the domain completely and restarts it with the same configuration • **preserve** - this action terminates the domain completely but and its resources are preserved to allow for future analysis. • **rename-restart** - this action terminates the domain completely and then restarts it with a new name
<on_reboot>	Specifies the action that is to be executed when the guest virtual machine requests a reboot. Four possible arguments are possible: • **destroy** - this action terminates the domain completely and releases all resources • **restart** - this action terminates the domain completely and restarts it with the same configuration • **preserve** - this action terminates the domain completely but and its resources are preserved to allow for future analysis. • **rename-restart** - this action terminates the domain completely and then restarts it with a new name

State	Description
<on_crash>	Specifies the action that is to be executed when the guest virtual machine crashes. In addition, it supports these additional actions: ● **coredump-destroy** - the crashed domain's core is dumped, domain is terminated completely, and all resources are released. ● **coredump-restart** - the crashed domain's core is dumped, and the domain is restarted with the same configuration settings Four possible arguments are possible: ● **destroy** - this action terminates the domain completely and releases all resources ● **restart** - this action terminates the domain completely and restarts it with the same configuration ● **preserve** - this action terminates the domain completely but and its resources are preserved to allow for future analysis. ● **rename-restart** - this action terminates the domain completely and then restarts it with a new name
<on_lockfailure>	Specifies what action should be taken when a lock manager loses resource locks. The following actions are recognized by libvirt, although not all of them need to be supported by individual lock managers. When no action is specified, each lock manager will take its default action. The following arguments are possible: ● **poweroff** - forcefully powers off the domain ● **restart** - restarts the domain to reacquire its locks. ● **pause** - pauses the domain so that it can be manually resumed when lock issues are solved. ● **ignore** - keeps the domain running as if nothing happened.

20.13. POWER MANAGEMENT

It is possible to forcibly enable or disable BIOS advertisements to the guest virtual machine OS using conventional management tools which effects the following section of the domain XML:

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

Figure 20.19. Power management

The **<pm>** element can be enabled using the arguement **yes** or disabled using the argument **no**. BIOS support can be implemented for S3 using the argument **suspend-to-disk** and S4 using the argument **suspend-to-mem** ACPI sleep states. If nothing is specified, the hypervisor will be left with its default value.

20.14. HYPERVISOR FEATURES

Hypervisors may allow certain CPU / machine features to be enabled (**state='on')** or disabled (**state='off'**).

```
...
<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
  </hyperv>
</features>
...
```

Figure 20.20. Hypervisor features

All features are listed within the **<features>** element, if a **<state>** is not specified it is disabled. The available features can be found by calling the **capabilities** XML, but a common set for fully virtualized domains are:

Table 20.11. Hypervisor features elements

State	Description
<pae>	Physical address extension mode allows 32-bit guest virtual machines to address more than 4 GB of memory.
<acpi>	Useful for power management, for example, with KVM guest virtual machines it is required for graceful shutdown to work.

State	Description
<apic>	Allows the use of programmable IRQ management. For this element, there is an optional attribute **eoi** with values **on** and **off** which sets the availability of EOI (End of Interrupt) for the guest virtual machine.
<hap>	Enables the use of Hardware Assisted Paging if it is available in the hardware.
hyperv	Enables various features to improve the behavior of guest virtual machines running Microsoft Windows. Using the optional attribute **relaxed** with values **on** or **off** enables or disables the relax constraints on timers

20.15. TIMEKEEPING

The guest virtual machine clock is typically initialized from the host physical machine clock. Most operating systems expect the hardware clock to be kept in UTC, which is the default setting. Note that for Windows guest virtual machines the guest virtual machine must be set in **localtime**.

```
...
<clock offset='localtime'>
  <timer name='rtc' tickpolicy='catchup' track='guest'>
    <catchup threshold='123' slew='120' limit='10000'/>
  </timer>
  <timer name='pit' tickpolicy='delay'/>
</clock>
...
```

Figure 20.21. Timekeeping

The components of this section of the domain XML are as follows:

Table 20.12. Time keeping elements

State	Description

State	Description
<clock>	The **offset** attribute takes four possible values, allowing for fine grained control over how the guest virtual machine clock is synchronized to the host physical machine. Note that hypervisors are not required to support all policies across all time sources • **utc** - Synchronizes the clock to UTC when booted. **utc** mode can be converted to **variable** mode, which can be controlled by using the adjustment attribute. If the value is **reset**, the conversion is not done. A numeric value forces the conversion to **variable** mode using the value as the initial adjustment. The default adjustment is hypervisor specific. • **localtime** - Synchronizes the guest virtual machine clock with the host physical machine's configured timezone when booted. The adjustment attribute behaves the same as in 'utc' mode. • **timezone** - Synchronizes the guest virtual machine clock to the requested timezone using the timezone attribute. • **variable** - Gives the guest virtual machine clock an arbitrary offset applied relative to UTC or localtime, depending on the basis attribute. The delta relative to UTC (or localtime) is specified in seconds, using the **adjustment** attribute. The guest virtual machine is free to adjust the RTC over time and expect that it will be honored at next reboot. This is in contrast to **utc** and **localtime** mode (with the optional attribute **adjustment='reset'**), where the RTC adjustments are lost at each reboot. In addition the **basis** attribute can be either **utc** (default) or **localtime**. The **clock** element may have zero or more **<timer>** elements.
<timer>	See Note
<frequency>	This is an unsigned integer specifying the frequency at which **name**="**tsc"** runs.
<mode>	The **mode** attribute controls how the**name="tsc" <timer>** is managed, and can be set to:**auto**, **native**, **emulate**, **paravirt**, or **smpsafe**. Other timers are always emulated.

State	Description
<present>	Specifies whether a particular timer is available to the guest virtual machine. Can be set to **yes** or **no**

NOTE

Each **<timer>** element must contain a **name** attribute, and may have the following attributes depending on the name specified.

- **<name>** - selects which **timer** is being modified. The following values are acceptable:**kvmclock** (QEMU-KVM), **pit**(QEMU-KVM), or **rtc**(QEMU-KVM), or **tsc**(libxl only). Note that **platform** is currently unsupported.
- track - specifies the timer track. The following values are acceptable: **boot**, **guest**, or **wall**. **track** is only valid for **name="rtc"**.
- **tickpolicy** - determines what happens whens the deadline for injecting a tick to the guest virtual machine is missed. The following values can be assigned:
  - **delay** -will continue to deliver ticks at the normal rate. The guest virtual machine time will be delayed due to the late tick
  - **catchup** - delivers ticks at a higher rate in order to catch up with the missed tick. The guest virtual machine time is not displayed once catchup is complete. In addition, there can be three optional attributes, each a positive integer, as follows: threshold, slew, and limit.
  - **merge** - merges the missed tick(s) into one tick and injects them. The guest virtual machine time may be delayed, depending on how the merge is done.
  - **discard** - throws away the missed tick(s) and continues with future injection at its default interval setting. The guest virtual machine time may be delayed, unless there is an explicit statement for handling lost ticks

20.16. DEVICES

This set of XML elements are all used to describe devices provided to the guest virtual machine domain. All of the devices below are indicated as children of the main devices element.

The following virtual devices are supported:

- virtio-scsi-pci - PCI bus storage device
- virtio-9p-pci - PCI bus storage device
- virtio-blk-pci - PCI bus storage device
- virtio-net-pci - PCI bus network device also known as virtio-net
- virtio-serial-pci - PCI bus input device
- virtio-balloon-pci - PCI bus memory balloon device

- virtio-rng-pci - PCI bus virtual random number generator device

IMPORTANT

If a virtio device is created where the number of vectors is set to a value higher than 32, the device behaves as if it was set to a zero value on Red Hat Enterprise Linux 6, but not on Enterprise Linux 7. The resulting vector setting mismatch causes a migration error if the number of vectors on any virtio device on either platform is set to 33 or higher. It is therefore not reccomended to set the **vector** value to be greater than 32. All virtio devices with the exception of virtio-balloon-pci and virtio-rng-pci will accept a **vector** argument.

```
...
<devices>
  <emulator>/usr/lib/xen/bin/qemu-dm</emulator>
</devices>
...
```

Figure 20.22. Devices - child elements

The contents of the **<emulator>** element specify the fully qualified path to the device model emulator binary. The capabilities XML specifies the recommended default emulator to use for each particular domain type or architecture combination.

20.16.1. Hard Drives, Floppy Disks, CDROMs

This section of the domain XML specifies any device that looks like a disk, be it a floppy, harddisk, cdrom, or paravirtualized driver is specified via the disk element.

```
...
 <devices>
   <disk type='file' snapshot='external'>
     <driver name="tap" type="aio" cache="default"/>
     <source file='/var/lib/xen/images/fv0' startupPolicy='optional'>
       <seclabel relabel='no'/>
     </source>
     <target dev='hda' bus='ide'/>
     <iotune>
       <total_bytes_sec>10000000</total_bytes_sec>
       <read_iops_sec>400000</read_iops_sec>
       <write_iops_sec>100000</write_iops_sec>
     </iotune>
     <boot order='2'/>
     <encryption type='...'>
       ...
     </encryption>
     <shareable/>
     <serial>
       ...
     </serial>
   </disk>
     ...
```

```
  <disk type='network'>
    <driver name="qemu" type="raw" io="threads" ioeventfd="on" event_idx="off"/>
    <source protocol="sheepdog" name="image_name">
      <host name="hostname" port="7000"/>
    </source>
    <target dev="hdb" bus="ide"/>
    <boot order='1'/>
    <transient/>
    <address type='drive' controller='0' bus='1' unit='0'/>
  </disk>
  <disk type='network'>
    <driver name="qemu" type="raw"/>
    <source protocol="rbd" name="image_name2">
      <host name="hostname" port="7000"/>
    </source>
    <target dev="hdd" bus="ide"/>
    <auth username='myuser'>
      <secret type='ceph' usage='mypassid'/>
    </auth>
  </disk>
  <disk type='block' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <target dev='hdc' bus='ide' tray='open'/>
    <readonly/>
  </disk>
  <disk type='block' device='lun'>
    <driver name='qemu' type='raw'/>
    <source dev='/dev/sda'/>
    <target dev='sda' bus='scsi'/>
    <address type='drive' controller='0' bus='0' target='3' unit='0'/>
  </disk>
  <disk type='block' device='disk'>
    <driver name='qemu' type='raw'/>
    <source dev='/dev/sda'/>
    <geometry cyls='16383' heads='16' secs='63' trans='lba'/>
    <blockio logical_block_size='512' physical_block_size='4096'/>
    <target dev='hda' bus='ide'/>
  </disk>
  <disk type='volume' device='disk'>
    <driver name='qemu' type='raw'/>
    <source pool='blk-pool0' volume='blk-pool0-vol0'/>
    <target dev='hda' bus='ide'/>
  </disk>
</devices>
...
```

Figure 20.23. Devices - Hard drives, floppy disks, CDROMs

20.16.1.1. Disk element

The **<disk>** element is the main container for describing disks. The attribute **type** can be used with the **<disk>** element. The following types are allowed:

- **file**
- **block**

- **dir**
- **network**

For more information, see Disk Elements

20.16.1.2. Source element

If the **<disk type='file''>**, then the **file** attribute specifies the fully-qualified path to the file holding the disk. If the **<disk type='block'>**, then the **dev** attribute specifies the path to the host physical machine device to serve as the disk. With both **file** and **block**, one or more optional sub-elements **seclabel**, described below, can be used to override the domain security labeling policy for just that source file. If the disk type is **dir**, then the **dir** attribute specifies the fully-qualified path to the directory to use as the disk. If the disk type is **network**, then the protocol attribute specifies the protocol to access to the requested image; possible values are **nbd**, **rbd**, **sheepdog** or **gluster**.

If the protocol attribute is **rbd**, **sheepdog** or **gluster**, an additional attribute **name** is mandatory to specify which volume and or image will be used. When the disk type is **network**, the **source** may have zero or more **host** sub-elements used to specify the host physical machines to connect, including: **type='dir'** and **type='network'**. For a **file** disk type which represents a cdrom or floppy (the device attribute), it is possible to define policy what to do with the disk if the source file is not accessible. This is done by manipulating the **startupPolicy** attribute, with the following values:

- **mandatory** causes a failure if missing for any reason. This is the default setting.
- **requisite** causes a failure if missing on boot up, drops if missing on migrate/restore/revert
- **optional** drops if missing at any start attempt

20.16.1.3. Mirror element

This element is present if the hypervisor has started a **BlockCopy** operation, where the **<mirror>** location in the attribute file will eventually have the same contents as the source, and with the file format in attribute format (which might differ from the format of the source). If an attribute ready is present, then it is known the disk is ready to pivot; otherwise, the disk is probably still copying. For now, this element only valid in output; it is ignored on input.

20.16.1.4. Target element

The **<target>** element controls the bus / device under which the disk is exposed to the guest virtual machine OS. The dev attribute indicates the logical device name. The actual device name specified is not guaranteed to map to the device name in the guest virtual machine OS. The optional bus attribute specifies the type of disk device to emulate; possible values are driver specific, with typical values being **ide**, **scsi**, **virtio**, **xen**, **usb** or **sata**. If omitted, the bus type is inferred from the style of the device name. eg, a device named **'sda'** will typically be exported using a SCSI bus. The optional attribute **tray** indicates the tray status of the removable disks (such as CD-ROM or Floppy disk), the value can be either **open** or **closed**. The default setting is **closed**. For more information, see target Elements

20.16.1.5. iotune

The optional **<iotune>** element provides the ability to provide additional per-device I/O tuning, with values that can vary for each device (contrast this to the **blkiotune** element, which applies globally to the domain). This element has the following optional sub-elements. Note that any sub-element not specified or at all or specified with a value of **0** implies no limit.

- **<total_bytes_sec>** - the total throughput limit in bytes per second. This element cannot be used with **<read_bytes_sec>** or **<write_bytes_sec>**.
- **<read_bytes_sec>** - the read throughput limit in bytes per second.
- **<write_bytes_sec>** - the write throughput limit in bytes per second.
- **<total_iops_sec>** - the total I/O operations per second. This element cannot be used with **<read_iops_sec>** or **<write_iops_sec>**.
- **<read_iops_sec>** - the read I/O operations per second.
- **<write_iops_sec>** - the write I/O operations per second.

20.16.1.6. driver

The optional **<driver>** element allows specifying further details related to the hypervisor driver that is used to provide the disk. The following options may be used:

- If the hypervisor supports multiple back-end drivers, then the **name** attribute selects the primary back-end driver name, while the optional type attribute provides the sub-type. For a list of possible types refer to Driver Elements
- The optional **cache** attribute controls the cache mechanism, possible values are: **default**, **none**, **writethrough**, **writeback**, **directsync** (similar to **writethrough**, but it bypasses the host physical machine page cache) and **unsafe** (host physical machine may cache all disk io, and sync requests from guest virtual machine virtual machines are ignored).
- The optional **error_policy** attribute controls how the hypervisor behaves on a disk read or write error, possible values are **stop**, **report**, **ignore**, and **enospace**. The default setting of **error_policy** is **report**. There is also an optional **rerror_policy** that controls behavior for read errors only. If no **rerror_policy** is given, **error_policy** is used for both read and write errors. If **rerror_policy** is given, it overrides the **error_policy** for read errors. Also note that **enospace** is not a valid policy for read errors, so if **error_policy** is set to **enospace** and **no rerror_policy** is given, the read error the default setting, **report** will be used.
- The optional **io** attribute controls specific policies on I/O; **qemu** guest virtual machine virtual machines support **threads** and **native**. The optional **ioeventfd** attribute allows users to set domain I/O asynchronous handling for disk device. The default is left to the discretion of the hypervisor. Accepted values are **on** and **off**. Enabling this allows the guest virtual machine virtual machine to be executed while a separate thread handles I/O. Typically guest virtual machine virtual machines experiencing high system CPU utilization during I/O will benefit from this. On the other hand, an overloaded host physical machine can increase guest virtual machine virtual machine I/O latency. Unless you are absolutely certian that the **io** needs to be manipulated, it is highly recommended that you not change the default setting and allow the hypervisor to dictate the setting.
- The optional **event_idx** attribute controls some aspects of device event processing and can be set to either **on** or **off** - if it is on, it will reduce the number of interrupts and exits for the guest virtual machine virtual machine. The default is determined by the hypervisor and the default setting is **on**. In cases that there is a situation where this behavior is suboptimal, this attribute provides a way to force the feature **off**. Unless you are absolutely certian that the **event_idx** needs to be manipulated, it is highly recommended that you not change the default setting and allow the hypervisor to dictate the setting.
- The optional **copy_on_read** attribute controls whether to copy the read backing file into the image file. The accepted values can be either **on** or **<off>**. **copy-on-read** avoids accessing the

same backing file sectors repeatedly and is useful when the backing file is over a slow network. By default **copy-on-read** is **off**.

20.16.1.7. Additional device elements

The following attributes may be used within the **device** element:

- **<boot>** - Specifies that the disk is bootable.

  Additional boot values

  - **<order>** - Determines the order in which devices will be tried during boot sequence.
  - **<per-device>** boot elements cannot be used together with general boot elements in BIOS boot loader section

- **<encryption>** - Specifies how the volume is encrypted. See the Storage Encryption page for more information.
- **<readonly>** - Indicates the device cannot be modified by the guest virtual machine virtual machine. This setting is the default for disks with **attribute device='cdrom'**.
- **shareable** Indicates the device is expected to be shared between domains (as long as hypervisor and OS support this). If **shareable** is used, **cache='no'** should be used for that device.
- **<transient>**- Indicates that changes to the device contents should be reverted automatically when the guest virtual machine virtual machine exits. With some hypervisors, marking a disk **transient** prevents the domain from participating in migration or snapshots.
- **<serial>**- Specifies the serial number of guest virtual machine virtual machine virtual machine's hard drive. For example, **<serial>**WD-WMAP9A966149**</serial>**.
- **<wwn>** - Specifies the WWN (World Wide Name) of a virtual hard disk or CD-ROM drive. It must be composed of 16 hexadecimal digits.
- **<vendor>** - Specifies the vendor of a virtual hard disk or CD-ROM device. It must not be longer than 8 printable characters.
- **<product>** - Specifies the product of a virtual hard disk or CD-ROM device. It must not be longer than 16 printable characters
- **<host>** - Supports 4 attributes: **viz**, **name**, **port**, **transport** and **socket**, which specify the host name, the port number, transport type and path to socket, respectively. The meaning of this element and the number of the elements depend on the **protocol** attribute as shown here:

  Additional host attributes

  - **nbd** - Specifies a server running nbd-server and may only be used for only one host physical machine
  - **rbd** - Monitors servers of RBD type and may be used for one or more host physical machines
  - **sheepdog** - Specifies one of the sheepdog servers (default is localhost:7000) and can be used one or none of the host physical machines

  - **gluster** - Specifies a server running a glusterd daemon and may be used for only only one host physical machine. The valid values for transport attribute are **tcp**, **rdma** or **unix**. If nothing is specified, **tcp** is assumed. If transport is **unix**, the **socket** attribute specifies path to unix socket.

- **<address>** - Ties the disk to a given slot of a controller. The actual **<controller>** device can often be inferred by but it can also be explicitly specified. The **type** attribute is mandatory, and is typically **pci** or **drive**. For a **pci** controller, additional attributes for **bus**, **slot**, and **function** must be present, as well as optional **domain** and **multifunction**. **multifun ction** defaults to **off**. For a **drive** controller, additional attributes **controller**, **bus**, **target**, and **unit** are available, each with a default setting of **0**.

- **auth** - Provides the authentication credentials needed to access the source. It includes a mandatory attribute **username**, which identifies the user name to use during authentication, as well as a sub-element **secret** with mandatory attribute **type**. More information can be found here at Device Elements

- **geometry** - Provides the ability to override geometry settings. This mostly useful for S390 DASD-disks or older DOS-disks.

- **cyls** - Specifies the number of cylinders.

- **heads** - Specifies the number of heads.

- **secs** - Specifies the number of sectors per track.

- **trans** - Specifies the BIOS-Translation-Modus and can have the following values: **none**, **lba** or **auto**

- **blockio** - Allows the block device to be overridden with any of the block device properties listed below:

  blockio options

  - **logical_block_size**- reports to the guest virtual machine virtual machine OS and describes the smallest units for disk I/O.

  - **physical_block_size** - reports to the guest virtual machine virtual machine OS and describes the disk's hardware sector size which can be relevant for the alignment of disk data.

20.16.2. Filesystems

A filesystems directory on the host physical machine that can be accessed directly from the guest virtual machine virtual machine

```
...
 <devices>
  <filesystem type='template'>
    <source name='my-vm-template'/>
    <target dir='/'/>
  </filesystem>
  <filesystem type='mount' accessmode='passthrough'>
    <driver type='path' wrpolicy='immediate'/>
    <source dir='/export/to/guest'/>
    <target dir='/import/from/host'/>
    <readonly/>
  </filesystem>
  ...
 </devices>
 ...
```

Figure 20.24. Devices - filesystems

The **filesystem** attribute has the following possible values:

- **type='mount'** - Specifies the host physical machine directory to mount in the guest virtual machine. This is the default type if one is not specified. This mode also has an optional sub-element **driver**, with an attribute **type='path'** or **type='handle'**. The driver block has an optional attribute **wrpolicy** that further controls interaction with the host physical machine page cache; omitting the attribute reverts to the default setting, while specifying a value immediate means that a host physical machine writeback is immediately triggered for all pages touched during a guest virtual machine file write operation

- **type='template'** - Specifies the OpenVZ filesystem template and is only used by OpenVZ driver.

- **type='file'** - Specifies that a host physical machine file will be treated as an image and mounted in the guest virtual machine. This filesystem format will be autodetected and is only used by LXC driver.

- **type='block'** - Specifies the host physical machine block device to mount in the guest virtual machine. The filesystem format will be autodetected and is only used by LXC driver.

- **type='ram'** - Specifies that an in-memory filesystem, using memory from the host physical machine OS will be used. The source element has a single attribute **usage** which gives the memory usage limit in kibibytes and is only used by LXC driver.

- **type='bind'** - Specifies a directory inside the guest virtual machine which will be bound to another directory inside the guest virtual machine. This element is only used by LXC driver.

- **accessmode** which specifies the security mode for accessing the source. Currently this only works with type='mount' for the QEMU/KVM driver. The possible values are:

  - **passthrough** - Specifies that the source is accessed with the User's permission settings that are set from inside the guest virtual machine. This is the default access mode if one is not specified.

  - **mapped** - Specifies that the source is accessed with the permission settings of the hypervisor.

  - **squash** - Similar to **'passthrough'**, the exception is that failure of privileged operations like **chown** are ignored. This makes a passthrough-like mode usable for people who run the hypervisor as non-root.

- **<source>** - Specifies that the resource on the host physical machine that is being accessed in the guest virtual machine. The **name** attribute must be used with **<type='template'>**, and the **dir** attribute must be used with **<type='mount'>**. The **usage** attribute is used with **<type='ram'>** to set the memory limit in KB.

- **target** - Dictates where the source drivers can be accessed in the guest virtual machine. For most drivers this is an automatic mount point, but for QEMU-KVM this is merely an arbitrary string tag that is exported to the guest virtual machine as a hint for where to mount.

- **readonly** - Enables exporting the file sydtem as a readonly mount for guest virtual machine, by default **read-write** access is given.

- **space_hard_limit** - Specifies the maximum space available to this guest virtual machine's filesystem

- **space_soft_limit** - Specifies the maximum space available to this guest virtual machine's filesystem. The container is permitted to exceed its soft limits for a grace period of time. Afterwards the hard limit is enforced.

20.16.3. Device Addresses

Many devices have an optional **<address>** sub-element to describe where the device placed on the virtual bus is presented to the guest virtual machine. If an address (or any optional attribute within an address) is omitted on input, libvirt will generate an appropriate address; but an explicit address is required if more control over layout is required. See below for device examples including an address element.

Every address has a mandatory attribute ***type*** that describes which bus the device is on. The choice of which address to use for a given device is constrained in part by the device and the architecture of the guest virtual machine. For example, a disk device uses **type='disk'**, while a console device would use **type='pci'** on the 32-bit AMD and Intel architecture or AMD64 and Intel 64 guest virtual machines, or **type='spapr-vio'** on PowerPC64 pseries guest virtual machines. Each address **<type>** has additional optional attributes that control where on the bus the device will be placed. The additional attributes are as follows:

- **type='pci'** - PCI addresses have the following additional attributes:

  - **domain** (a 2-byte hex integer, not currently used by qemu)

  - **bus** (a hex value between 0 and 0xff, inclusive)

  - **slot** (a hex value between 0x0 and 0x1f, inclusive)

  - **function** (a value between 0 and 7, inclusive)

  - Also available is the **multifunction** attribute, which controls turning on the multifunction bit for a particular slot/function in the PCI control register. This multifunction attribute defaults to **'off'**, but should be set to **'on'** for function 0 of a slot that will have multiple functions used.

- **type='drive** - drive addresses have the following additional attributes:

  - **controller**- (a 2-digit controller number)

  - **bus** - (a 2-digit bus number)
  - **target** - (a 2-digit bus number)
  - **unit** - (a 2-digit unit number on the bus)
- **type='virtio-serial'** - Each virtio-serial address has the following additional attributes:
  - **controller** - (a 2-digit controller number)
  - **bus** - (a 2-digit bus number)
  - **slot** - (a 2-digit slot within the bus)
- **type='ccid'** - A CCID address, used for smart-cards, has the following additional attributes:
  - **bus** - (a 2-digit bus number)
  - **slot** attribute - (a 2-digit slot within the bus)
- **type='usb'** - USB addresses have the following additional attributes:
  - **bus** - (a hex value between 0 and 0xfff, inclusive)
  - **port** - (a dotted notation of up to four octets, such as 1.2 or 2.1.3.1)
- **type='spapr-vio** - On PowerPC pseries guest virtual machines, devices can be assigned to the SPAPR-VIO bus. It has a flat 64-bit address space; by convention, devices are generally assigned at a non-zero multiple of 0x1000, but other addresses are valid and permitted by libvirt. The additional attribute: reg (the hex value address of the starting register) can be assigned to this attribute.

20.16.4. Controllers

Depending on the guest virtual machine architecture, it is possible to assign many virtual devices to a single bus. Under normal circumstances libvirt can automatically infer which controller to use for the bus. However, it may be necessary to provide an explicit **<controller>** element in the guest virtual machine XML:

```
...
<devices>
  <controller type='ide' index='0'/>
  <controller type='virtio-serial' index='0' ports='16' vectors='4'/>
  <controller type='virtio-serial' index='1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0'/>
  <controller type='scsi' index='0' model='virtio-scsi' num_queues='8'/>
  </controller>
  ...
</devices>
...
```

Figure 20.25. Controller Elements

Each controller has a mandatory attribute **type**, which must be one of **"ide", "fdc", "scsi", "sata",**

"usb", "ccid", or "virtio-serial", and a mandatory attribute **index** which is the decimal integer describing in which order the bus controller is encountered (for use in controller attributes of **address** elements). The **"virtio-serial"** controller has two additional optional attributes, **ports** and **vectors**, which control how many devices can be connected through the controller.

A **<controller type='scsi'>** has an optional attribute **model**, which is one of **"auto"**, **"buslogic"**, **"ibmvscsi"**, **"lsilogic"**, **"lsias1068"**, **"virtio-scsi** or **"vmpvscsi"**. It should be noted that virtio-scsi controllers and drivers will work on both KVM and Windows guest virtual machines. The **<controller type='scsi'>** also has an attribute **num_queues** which enables multi-queue support for the number of queues specified.

A **"usb"** controller has an optional attribute **model**, which is one of **"piix3-uhci"**, **"piix4-uhci"**, **"ehci"**, **"ich9-ehci1"**, **"ich9-uhci1"**, **"ich9-uhci2"**, **"ich9-uhci3"**, **"vt82c686b-uhci"**, **"pci-ohci"** or **"nec-xhci"**. Additionally, if the USB bus needs to be explicitly disabled for the guest virtual machine, **model='none'** may be used. The PowerPC64 "spapr-vio" addresses do not have an associated controller.

For controllers that are themselves devices on a PCI or USB bus, an optional sub-element **address** can specify the exact relationship of the controller to its master bus, with semantics given above.

USB companion controllers have an optional sub-element **master** to specify the exact relationship of the companion to its master controller. A companion controller is on the same bus as its master, so the companion index value should be equal.

```
...
<devices>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0' bus='0' slot='4' function='7'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0'/>
    <address type='pci' domain='0' bus='0' slot='4' function='0' multifunction='on'/>
  </controller>
  ...
</devices>
...
```

Figure 20.26. Devices - controllers - USB

20.16.5. Device Leases

When using a lock manager, you have the option to record device leases against a guest virtual machine. The lock manager will ensure that the guest virtual machine does not start unless the leases can be acquired. When configured using conventional management tools, the following section of the domain xml is effected:

```
...
<devices>
  ...
  <lease>
    <lockspace>somearea</lockspace>
    <key>somekey</key>
    <target path='/some/lease/path' offset='1024'/>
  </lease>
  ...
</devices>
...
```

Figure 20.27. Devices - device leases

The **lease** section can have the following arguments:

- **lockspace** - an arbitrary string that identifies lockspace within which the key is held. Lock managers may impose extra restrictions on the format, or length of the lockspace name.
- **key** - an arbitrary string, that uniquely identies the lease to be acquired. Lock managers may impose extra restrictions on the format, or length of the key.
- **target** - the fully qualified path of the file associated with the lockspace. The offset specifies where the lease is stored within the file. If the lock manager does not require a offset, set this value to **0**.

20.16.6. Host Physical Machine Device Assignment

This section provides information about host physical machine device assignment.

20.16.6.1. USB / PCI Devices

The host physical machine's USB and PCI devices can be passed through to the guest virtual machine using the **hostdev** element, by modifying the host physical machine using a management tool the following section of the domain xml file is configured:

```
...
<devices>
  <hostdev mode='subsystem' type='usb'>
    <source startupPolicy='optional'>
      <vendor id='0x1234'/>
      <product id='0xbeef'/>
    </source>
    <boot order='2'/>
  </hostdev>
</devices>
...
```

Figure 20.28. Devices - host physical machine device assignment

Alternatively the following can also be done:

```
...
<devices>
  <hostdev mode='subsystem' type='pci' managed='yes'>
    <source>
      <address bus='0x06' slot='0x02' function='0x0'/>
    </source>
    <boot order='1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </hostdev>
</devices>
...
```

Figure 20.29. Devices - host physical machine device assignment alternative

The components of this section of the domain XML are as follows:

Table 20.13. Host physical machine device assignment elements

Parameter	Description
hostdev	This is the main container for describing host physical machine devices. For USB device passthrough **mode** is always **subsystem** and **type** is **usb** for a USB device and **pci** for a PCI device. When **managed** is **yes** for a PCI device, it is detached from the host physical machine before being passed on to the guest virtual machine, and reattached to the host physical machine after the guest virtual machine exits. If **managed** is omitted or **no** for PCI and for USB devices, the user is responsible to use the argument **virNodeDeviceDettach** (or **virsh nodedev-dettach**) before starting the guest virtual machine or hot-plugging the device, and **virNodeDeviceReAttach** (or **virsh nodedev-reattach**) after hot-unplug or stopping the guest virtual machine.

Parameter	Description
source	Describes the device as seen from the host physical machine. The USB device can either be addressed by vendor / product id using the **vendor** and **product** elements or by the device's address on the host physical machines using the **address** element. PCI devices on the other hand can only be described by their address. Note that the source element of USB devices may contain a **startupPolicy** attribute which can be used to define a rule for what to do if the specified host physical machine USB device is not found. The attribute accepts the following values: • **mandatory** - fails if missing for any reason (the default) • **requisite** - fails if missing on boot up, drops if missing on migrate/restore/revert • **optional** - drops if missing at any start attempt
vendor, product	These elements each have an **id** attribute that specifies the USB vendor and product id. The IDs can be given in decimal, hexadecimal (starting with 0x) or octal (starting with 0) form.
boot	Specifies that the device is bootable. The attribute's order determines the order in which devices will be tried during boot sequence. The per-device boot elements cannot be used together with general boot elements in BIOS boot loader section.
rom	Used to change how a PCI device's ROM is presented to the guest virtual machine. The optional **bar** attribute can be set to **on** or **off**, and determines whether or not the device's ROM will be visible in the guest virtual machine's memory map. (In PCI documentation, the **rombar** setting controls the presence of the Base Address Register for the ROM). If no rom bar is specified, the default setting will be used. The optional **file** attribute is used to point to a binary file to be presented to the guest virtual machine as the device's ROM BIOS. This can be useful, for example, to provide a PXE boot ROM for a virtual function of an sr-iov capable ethernet device (which has no boot ROMs for the VFs).

Parameter	Description
address	Also has a **bus** and **device** attribute to specify the USB bus and device number the device appears at on the host physical machine. The values of these attributes can be given in decimal, hexadecimal (starting with 0x) or octal (starting with 0) form. For PCI devices the element carries 3 attributes allowing to designate the device as can be found with **lspci** or with **virsh nodedev-list**

20.16.6.2. Block / character devices

The host physical machine's block / character devices can be passed through to the guest virtual machine by using management tools to modify the domain xml **hostdev** element. Note that this is only possible with container based virtualization.

```
...
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
```

Figure 20.30. Devices - host physical machine device assignment block character devices

An alternative approach is this:

```
...
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
```

Figure 20.31. Devices - host physical machine device assignment block character devices alternative 1

Another alternative approach is this:

```
...
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
```

Figure 20.32. Devices - host physical machine device assignment block character devices alternative 2

The components of this section of the domain XML are as follows:

Table 20.14. Block / character device elements

Parameter	Description
hostdev	This is the main container for describing host physical machine devices. For block/character devices passthrough **mode** is always **capabilities** and **type** is **block** for a block device and **char** for a character device.
source	This describes the device as seen from the host physical machine. For block devices, the path to the block device in the host physical machine OS is provided in the nested **block** element, while for character devices the **char** element is used

20.16.7. Redirected Devices

USB device redirection through a character device is supported by configuring it with management tools that modify the following section of the domain xml:

```
...
<devices>
  <redirdev bus='usb' type='tcp'>
    <source mode='connect' host='localhost' service='4000'/>
    <boot order='1'/>
  </redirdev>
  <redirfilter>
    <usbdev class='0x08' vendor='0x1234' product='0xbeef' version='2.00' allow='yes'/>
    <usbdev allow='no'/>
  </redirfilter>
</devices>
...
```

Figure 20.33. Devices - redirected devices

The components of this section of the domain XML are as follows:

Table 20.15. Redirected device elements

Parameter	Description
redirdev	This is the main container for describing redirected devices. **bus** must be **usb** for a USB device. An additional attribute type is required, matching one of the supported serial device types, to describe the host physical machine side of the tunnel; **type='tcp'** or **type='spicevmc'** (which uses the usbredir channel of a SPICE graphics device) are typical. The redirdev element has an optional sub-element **address** which can tie the device to a particular controller. Further sub-elements, such as **source**, may be required according to the given **type**, although a **target** sub-element is not required (since the consumer of the character device is the hypervisor itself, rather than a device visible in the guest virtual machine).
boot	Specifies that the device is bootable. The order attribute determines the order in which devices will be tried during boot sequence. The per-device boot elements cannot be used together with general boot elements in BIOS boot loader section.
redirfilter	This is used for creating the filter rule to filter out certain devices from redirection. It uses sub-element **usbdev** to define each filter rule. The **class** attribute is the USB Class code.

20.16.8. Smartcard Devices

A virtual smartcard device can be supplied to the guest virtual machine via the **smartcard** element. A USB smartcard reader device on the host machine cannot be used on a guest with simple device passthrough, as it cannot be made available to both the host and guest, and can lock the host computer when it is removed from the guest. Therefore, some hypervisors provide a specialized virtual device that can present a smartcard interface to the guest virtual machine, with several modes for describing how the credentials are obtained from the host machine, or from a channel created by a third-party smartcard provider. To set parameters for USB device redirection through a character device, edit the following section of the domain XML:

```
...
<devices>
  <smartcard mode='host'/>
  <smartcard mode='host-certificates'>
    <certificate>cert1</certificate>
    <certificate>cert2</certificate>
    <certificate>cert3</certificate>
    <database>/etc/pki/nssdb/</database>
  </smartcard>
  <smartcard mode='passthrough' type='tcp'>
    <source mode='bind' host='127.0.0.1' service='2001'/>
    <protocol type='raw'/>
    <address type='ccid' controller='0' slot='0'/>
  </smartcard>
  <smartcard mode='passthrough' type='spicevmc'/>
</devices>
...
```

Figure 20.34. Devices - smartcard devices

The **smartcard** element has a mandatory attribute **mode**. The following modes are supported; in each mode, the guest virtual machine sees a device on its USB bus that behaves like a physical USB CCID (Chip/Smart Card Interface Device) card.

The mode attributes are as follows:

Table 20.16. Smartcard mode elements

Parameter	Description
mode='host'	In this mode, the hypervisor relays all direct access requests from the guest virtual machine to the host physical machine's smartcard via NSS. No other attributes or sub-elements are required. See below about the use of an optional **address** sub-element.
mode='host-certificates'	This mode allows you to provide three NSS certificate names residing in a database on the host physical machine, rather than requiring a smartcard to be plugged into the host physical machine. These certificates can be generated using the command **certutil -d /etc/pki/nssdb -x -t CT,CT,CT -S -s CN=cert1 -n cert1,** and the resulting three certificate names must be supplied as the content of each of three **certificate** sub-elements. An additional sub-element **database** can specify the absolute path to an alternate directory (matching the **-d** option of the **certutil** command when creating the certificates); if not present, it defaults to **/etc/pki/nssdb**.

Parameter	Description
mode='passthrough'	This mode allows you to tunnel all requests through a secondary character device to a third-party provider (which may in turn be talking to a smartcard or using three certificate files), rather than having the hypervisor directly communicate with the host physical machine. In this mode, an additional attribute **type** is required, matching one of the supported serial device types, to describe the host physical machine side of the tunnel; **type='tcp'** or **type='spicevmc'** (which uses the smartcard channel of a SPICE graphics device) are typical. Further sub-elements, such as **source**, may be required according to the given type, although a **target** sub-element is not required (since the consumer of the character device is the hypervisor itself, rather than a device visible in the guest virtual machine).

Each mode supports an optional sub-element **address**, which fine-tunes the correlation between the smartcard and a ccid bus controller (Refer to Section 20.16.3, "Device Addresses").

20.16.9. Network Interfaces

The network interface devices are modified using management tools that will configure the following part of the Domain XML:

```
...
<devices>
  <interface type='bridge'>
    <source bridge='xenbr0'/>
    <mac address='00:16:3e:5d:c7:9e'/>
    <script path='vif-bridge'/>
    <boot order='1'/>
    <rom bar='off'/>
  </interface>
</devices>
...
```

Figure 20.35. Devices - network interfaces

There are several possibilities for specifying a network interface visible to the guest virtual machine. Each subsection below provides more details about common setup options. Additionally, each **<interface>** element has an optional **<address>** sub-element that can tie the interface to a particular pci slot, with attribute **type='pci'** (Refer to Section 20.16.3, "Device Addresses").

20.16.9.1. Virtual networks

This is the recommended configuration for general guest virtual machine connectivity on host physical machines with dynamic / wireless networking configurations (or multi-host physical machine environments where the host physical machine hardware details are described separately in a

<network> definition). In addition, it provides a connection whose details are described by the named network definition. Depending on the virtual network's **forward mode** configuration, the network may be totally isolated (no **<forward>** element given), NAT'ing to an explicit network device or to the default route (**forward mode='nat'**), routed with no NAT (**forward mode='route'**/), or connected directly to one of the host physical machine's network interfaces (using macvtap) or bridge devices (**forward mode=' bridge|private|vepa|passthrough'**/)

For networks with a forward mode of bridge, private, vepa, and passthrough, it is assumed that the host physical machine has any necessary DNS and DHCP services already setup outside the scope of libvirt. In the case of isolated, nat, and routed networks, DHCP and DNS are provided on the virtual network by libvirt, and the IP range can be determined by examining the virtual network config with **virsh net-dumpxml [networkname]**. There is one virtual network called 'default' setup out of the box which does NAT'ing to the default route and has an IP range of 192.168.122.0/255.255.255.0. Each guest virtual machine will have an associated tun device created with a name of vnetN, which can also be overridden with the **<target>** element (refer to Section 20.16.9.11, "Overriding the target element").

When the source of an interface is a network, a portgroup can be specified along with the name of the network; one network may have multiple portgroups defined, with each portgroup containing slightly different configuration information for different classes of network connections. Also, similar to **<direct>** network connections (described below), a connection of type **network** may specify a **<virtualport>** element, with configuration data to be forwarded to a vepa (802.1Qbg) or 802.1Qbh-compliant switch, or to an Open vSwitch virtual switch.

Since the actual type of switch may vary depending on the configuration in the **<network>** on the host physical machine, it is acceptable to omit the virtualport type attribute, and specify attributes from multiple different virtualport types (and also to leave out certain attributes); at domain startup time, a complete **<virtualport>** element will be constructed by merging together the type and attributes defined in the network and the portgroup referenced by the interface. The newly-constructed virtualport is a combination of both. The attributes from lower virtualport cannot make changes on the ones defined in higher virtualport. Interfaces take the highest priority, portgroup is lowest priority.

For example, to create a properly working network with both an 802.1Qbh switch and an Open vSwitch switch, you may choose to specify no type, but both **profileid** and an **interfaceid** must be supplied. The other attributes to be filled in from the virtual port, such as such as **managerid**, **typeid**, or **profileid**, are optional.

If you want to limit a guest virtual machine to connecting only to certain types of switches, you can specify the virtualport type, and only switches with the specified port type will connect. You can also further limit switch connectivity by specifying additional parameters. As a result, if the port was specified and the host physical machine's network has a different type of virtualport, the connection of the interface will fail. The virtual network parameters are defined using management tools that modify the following part of the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
  </interface>
  ...
  <interface type='network'>
    <source network='default' portgroup='engineering'/>
    <target dev='vnet7'/>
    <mac address="00:11:22:33:44:55"/>
    <virtualport>
      <parameters instanceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
    </virtualport>

  </interface>
</devices>
...
```

Figure 20.36. Devices - network interfaces- virtual networks

20.16.9.2. Bridge to LAN

Note that this is the recommended configuration setting for general guest virtual machine connectivity on host physical machines with static wired networking configurations.

Bridge to LAN provides a bridge from the guest virtual machine directly onto the LAN. This assumes there is a bridge device on the host physical machine which has one or more of the host physical machines physical NICs enslaved. The guest virtual machine will have an associated **tun** device created with a name of **<vnetN>**, which can also be overridden with the **<target>** element (refer to Section 20.16.9.11, "Overriding the target element"). The **<tun>** device will be enslaved to the bridge. The IP range / network configuration is whatever is used on the LAN. This provides the guest virtual machine full incoming and outgoing net access just like a physical machine.

On Linux systems, the bridge device is normally a standard Linux host physical machine bridge. On host physical machines that support Open vSwitch, it is also possible to connect to an open vSwitch bridge device by adding a **virtualport type='openvswitch'**/ to the interface definition. The Open vSwitch type virtualport accepts two parameters in its **parameters** element - an **interfaceid** which is a standard uuid used to uniquely identify this particular interface to Open vSwitch (if you do no specify one, a random **interfaceid** will be generated for you when you first define the interface), and an optional **profileid** which is sent to Open vSwitch as the interfaces **<port-profile>**. To set the bridge to LAN settings, use a management tool that will configure the following part of the domain XML:

```
...
<devices>
  ...
  <interface type='bridge'>
    <source bridge='br0'/>
  </interface>
  <interface type='bridge'>
    <source bridge='br1'/>
    <target dev='vnet7'/>
    <mac address="00:11:22:33:44:55"/>
  </interface>
  <interface type='bridge'>
    <source bridge='ovsbr'/>
    <virtualport type='openvswitch'>
      <parameters profileid='menial' interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
    </virtualport>
  </interface>
  ...
</devices>
```

Figure 20.37. Devices - network interfaces- bridge to LAN

20.16.9.3. Setting a port masquerading range

In cases where you want to set the port masquerading range, the port can be set as follows:

```
<forward mode='nat'>
  <address start='192.0.2.1' end='192.0.2.10'/>
</forward>  ...
```

Figure 20.38. Port Masquerading Range

These values should be set using the **iptables** commands as shown in Section 18.3, “Network Address Translation Mode”

20.16.9.4. User-space SLIRP stack

Setting the user-space SLIRP stack parameters provides a virtual LAN with NAT to the outside world. The virtual network has DHCP and DNS services and will give the guest virtual machine an IP addresses starting from 10.0.2.15. The default router will be 10.0.2.2 and the DNS server will be 10.0.2.3. This networking is the only option for unprivileged users who need their guest virtual machines to have outgoing access.

The user-space SLIP stack parameters are defined in the following part of the domain XML::

```
...
<devices>
  <interface type='user'/>
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55"/>
  </interface>
</devices>
...
```

Figure 20.39. Devices - network interfaces- User-space SLIRP stack

20.16.9.5. Generic Ethernet connection

Provides a means for the administrator to execute an arbitrary script to connect the guest virtual machine's network to the LAN. The guest virtual machine will have a **tun** device created with a name of **vnetN**, which can also be overridden with the **target** element. After creating the **tun** device a shell script will be run which is expected to do whatever host physical machine network integration is required. By default this script is called **/etc/qemu-ifup** but can be overridden (refer to Section 20.16.9.11, "Overriding the target element").

The generic Ethernet connection parameters are defined in the following part of the domain XML:

```
...
<devices>
  <interface type='ethernet'/>
  ...
  <interface type='ethernet'>
    <target dev='vnet7'/>
    <script path='/etc/qemu-ifup-mynet'/>
  </interface>
</devices>
...
```

Figure 20.40. Devices - network interfaces- generic Ethernet connection

20.16.9.6. Direct attachment to physical interfaces

Using **<interface type='direct'>** attaches a virtual machine's NIC to a specified physical interface on the host.

This set up requires the Linux macvtap driver to be available. One of the following modes can be chosen for the operation mode of the macvtap device: **vepa** ('Virtual Ethernet Port Aggregator'), which is the default mode, **bridge** or **private**.

To set up direct attachment to physical interface, use the following parameters in the domain XML:

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
...
```

Figure 20.41. Devices - network interfaces- direct attachment to physical interfaces

The individual modes cause the delivery of packets to behave as shown in Table 20.17, "Direct attachment to physical interface elements":

Table 20.17. Direct attachment to physical interface elements

Element	Description
vepa	All of the guest virtual machines' packets are sent to the external bridge. Packets whose destination is a guest virtual machine on the same host physical machine as where the packet originates from are sent back to the host physical machine by the VEPA capable bridge (today's bridges are typically not VEPA capable).
bridge	Packets whose destination is on the same host physical machine as where they originate from are directly delivered to the target macvtap device. Both origin and destination devices need to be in bridge mode for direct delivery. If either one of them is in **vepa** mode, a VEPA capable bridge is required.
private	All packets are sent to the external bridge and will only be delivered to a target VM on the same host physical machine if they are sent through an external router or gateway and that device sends them back to the host physical machine. This procedure is followed if either the source or destination device is in private mode.
passthrough	This feature attaches a virtual function of a SRIOV capable NIC directly to a guest virtual machine without losing the migration capability. All packets are sent to the VF/IF of the configured network device. Depending on the capabilities of the device additional prerequisites or limitations may apply; for example, this requires kernel 2.6.38 or newer.

The network access of direct attached virtual machines can be managed by the hardware switch to which the physical interface of the host physical machine machine is connected to.

The interface can have additional parameters as shown below, if the switch is conforming to the IEEE 802.1Qbg standard. The parameters of the virtualport element are documented in more detail in the IEEE 802.1Qbg standard. The values are network specific and should be provided by the network administrator. In 802.1Qbg terms, the Virtual Station Interface (VSI) represents the virtual interface of a virtual machine.

Note that IEEE 802.1Qbg requires a non-zero value for the VLAN ID.

Additional elements that can be manipulated are described in Table 20.18, "Direct attachment to physical interface additional elements":

Table 20.18. Direct attachment to physical interface additional elements

Element	Description
managerid	The VSI Manager ID identifies the database containing the VSI type and instance definitions. This is an integer value and the value **0** is reserved.
typeid	The VSI Type ID identifies a VSI type characterizing the network access. VSI types are typically managed by network administrator. This is an integer value.
typeidversion	The VSI Type Version allows multiple versions of a VSI Type. This is an integer value.
instanceid	The VSI Instance ID Identifier is generated when a VSI instance (that is a virtual interface of a virtual machine) is created. This is a globally unique identifier.
profileid	The profile ID contains the name of the port profile that is to be applied onto this interface. This name is resolved by the port profile database into the network parameters from the port profile, and those network parameters will be applied to this interface.

Additional parameters in the domain XML include:

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0.2' mode='vepa'/>
    <virtualport type="802.1Qbg">
      <parameters managerid="11" typeid="1193047" typeidversion="2" instanceid="09b11c53-8b5c-
4eeb-8f00-d84eaa0aaa4f"/>
    </virtualport>
  </interface>
</devices>
...
```

Figure 20.42. Devices - network interfaces- direct attachment to physical interfaces additional parameters

The interface can have additional parameters as shown below if the switch is conforming to the IEEE 802.1Qbh standard. The values are network specific and should be provided by the network administrator.

Additional parameters in the domain XML include:

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private'/>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
</devices>
...
```

Figure 20.43. Devices - network interfaces- direct attachment to physical interfaces more additional parameters

The **profileid** attribute, contains the name of the port profile that is to be applied to this interface. This name is resolved by the port profile database into the network parameters from the port profile, and those network parameters will be applied to this interface.

20.16.9.7. PCI passthrough

A PCI network device (specified by the **source** element) is directly assigned to the guest virtual machine using generic device passthrough, after first optionally setting the device's MAC address to the configured value, and associating the device with an 802.1Qbh capable switch using an optionally specified **virtualport** element (see the examples of virtualport given above for type='direct' network devices). Note that - due to limitations in standard single-port PCI ethernet card driver design - only SR-IOV (Single Root I/O Virtualization) virtual function (VF) devices can be assigned in this manner; to assign a standard single-port PCI or PCIe ethernet card to a guest virtual machine, use the traditional **hostdev** device definition

Note that this "intelligent passthrough" of network devices is very similar to the functionality of a

standard **hostdev** device, the difference being that this method allows specifying a MAC address and **virtualport** for the passed-through device. If these capabilities are not required, if you have a standard single-port PCI, PCIe, or USB network card that does not support SR-IOV (and hence would anyway lose the configured MAC address during reset after being assigned to the guest virtual machine domain), or if you are using a version of libvirt older than 0.9.11, you should use standard **hostdev** to assign the device to the guest virtual machine instead of **interface type='hostdev'**/.

```
...
<devices>
  <interface type='hostdev'>
    <driver name='vfio'/>
    <source>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </source>
    <mac address='52:54:00:6d:90:02'>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
</devices>
...
```

Figure 20.44. Devices - network interfaces- PCI passthrough

20.16.9.8. Multicast tunnel

A multicast group may be used to represent a virtual network. Any guest virtual machine whose network devices are within the same multicast group will talk to each other, even if they reside across multiple physical host physical machines. This mode may be used as an unprivileged user. There is no default DNS or DHCP support and no outgoing network access. To provide outgoing network access, one of the guest virtual machines should have a second NIC which is connected to one of the first 4 network types in order to provide appropriate routing. The multicast protocol is compatible with protocols used by **user mode** linux guest virtual machines as well. Note that the source address used must be from the multicast address block. A multicast tunnel is created by manipulating the **interface type** using a management tool and setting/changing it to **mcast**, and providing a mac and source address. The result is shown in changes made to the domain XML:

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

Figure 20.45. Devices - network interfaces- multicast tunnel

20.16.9.9. TCP tunnel

Creating a TCP client/server architecture is another way to provide a virtual network where one guest virtual machine provides the server end of the network and all other guest virtual machines are

configured as clients. All network traffic between the guest virtual machines is routed via the guest virtual machine that is configured as the server. This model is also available for use to unprivileged users. There is no default DNS or DHCP support and no outgoing network access. To provide outgoing network access, one of the guest virtual machines should have a second NIC which is connected to one of the first 4 network types thereby providing the appropriate routing. A TCP tunnel is created by manipulating the **interface type** using a management tool and setting/changing it to **server** or **client**, and providing a mac and source address. The result is shown in changes made to the domain XML:

```
...
<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
  ...
  <interface type='client'>
    <mac address='52:54:00:8b:c9:51'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
</devices>
...
```

Figure 20.46. Devices - network interfaces- TCP tunnel

20.16.9.10. Setting NIC driver-specific options

Some NICs may have tunable driver-specific options. These options are set as attributes of the **driver** sub-element of the interface definition. These options are set by using management tools to configuring the following sections of the domain XML:

```
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <model type='virtio'/>
    <driver name='vhost' txmode='iothread' ioeventfd='on' event_idx='off'/>
  </interface>
</devices>
...
```

Figure 20.47. Devices - network interfaces- setting NIC driver-specific options

Currently the following attributes are available for the "virtio" NIC driver:

Table 20.19. virtio NIC driver elements

Parameter	Description

Parameter	Description
name	The optional **name** attribute forces which type of back-end driver to use. The value can be either **qemu** (a user-space back end) or **vhost** (a kernel back end, which requires the vhost module to be provided by the kernel); an attempt to require the vhost driver without kernel support will be rejected. The default setting is **vhost** if the vhost driver present, but will silently fall back to **qemu** if not.
txmode	Specifies how to handle transmission of packets when the transmit buffer is full. The value can be either **iothread** or **timer**. If set to **iothread**, packet tx is all done in an iothread in the bottom half of the driver (this option translates into adding **"tx=bh"** to the **qemu** commandline -device virtio-net-pci option). If set to **timer**, tx work is done in qemu, and if there is more tx data than can be sent at the present time, a timer is set before qemu moves on to do other things; when the timer fires, another attempt is made to send more data. In general you should leave this option alone, unless you are very certain you that changing it is an absolute necessity.
ioeventfd	Allows users to set domain I/O asynchronous handling for interface device. The default is left to the discretion of the hypervisor. Accepted values are **on** and **off** . Enabling this option allows qemu to execute a guest virtual machine while a separate thread handles I/O. Typically guest virtual machines experiencing high system CPU utilization during I/O will benefit from this. On the other hand, overloading the physical host physical machine may also increase guest virtual machine I/O latency. Therefore, you should leave this option alone, unless you are very certain you that changing it is an absolute necessity.
event_idx	The event_idx attribute controls some aspects of device event processing. The value can be either **on** or **off**. Choosing **on**, reduces the number of interrupts and exits for the guest virtual machine. The default is **on**. In case there is a situation where this behavior is suboptimal, this attribute provides a way to force the feature off. You should leave this option alone, unless you are very certain you that changing it is an absolute necessity.

20.16.9.11. Overriding the target element

To override the target element, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
  </interface>
</devices>
...
```

Figure 20.48. Devices - network interfaces- overriding the target element

If no target is specified, certain hypervisors will automatically generate a name for the created tun device. This name can be manually specified, however the name must not start with either 'vnet' or 'vif', which are prefixes reserved by libvirt and certain hypervisors. Manually specified targets using these prefixes will be ignored.

20.16.9.12. Specifying boot order

To specify the boot order, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <boot order='1'/>
  </interface>
</devices>
...
```

Figure 20.49. Specifying boot order

For hypervisors which support it, you can set a specific NIC to be used for the network boot. The order of attributes determine the order in which devices will be tried during boot sequence. Note that the per-device boot elements cannot be used together with general boot elements in BIOS boot loader section.

20.16.9.13. Interface ROM BIOS configuration

To specify the ROM BIOS configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </interface>
</devices>
...
```

Figure 20.50. Interface ROM BIOS configuration

For hypervisors which support it, you can change how a PCI Network device's ROM is presented to the guest virtual machine. The **bar** attribute can be set to **on** or **off**, and determines whether or not the device's ROM will be visible in the guest virtual machine's memory map. (In PCI documentation, the "rombar" setting controls the presence of the Base Address Register for the ROM). If no rom bar is specified, the qemu default will be used (older versions of qemu used a default of **off**, while newer qemus have a default of **on**). The optional **file** attribute is used to point to a binary file to be presented to the guest virtual machine as the device's ROM BIOS. This can be useful to provide an alternative boot ROM for a network device.

20.16.9.14. Quality of service

This section of the domain XML provides setting quality of service. Incoming and outgoing traffic can be shaped independently. The **bandwidth** element can have at most one inbound and at most one outbound child elements. Leaving any of these children element out results in no QoS being applied on that traffic direction. Therefore, when you want to shape only domain's incoming traffic, use inbound only, and vice versa.

Each of these elements has one mandatory attribute **average** (or **floor** as described below). **average** specifies average bit rate on the interface being shaped. Then there are two optional attributes: **peak**, which specifies maximum rate at which interface can send data, and **burst**, which specifies the amount of bytes that can be burst at peak speed. Accepted values for attributes are integer numbers.

The units for **average** and **peak** attributes are kilobytes per second, whereas **burst** is only set in kilobytes. In addition, inbound traffic can optionally have a **floor** attribute. This guarantees minimal throughput for shaped interfaces. Using the **floor** requires that all traffic goes through one point where QoS decisions can take place. As such it may only be used in cases where the **interface type='network'**/ with a **forward** type of **route**, **nat**, or no forward at all). It should be noted that within a virtual network, all connected interfaces are required to have at least the inbound QoS set (**average** at least) but the floor attribute does not require specifying **average**. However, **peak** and **burst** attributes still require **average**. At the present time, ingress qdiscs may not have any classes, and therefore **floor** may only be applied only on inbound and not outbound traffic.

To specify the QoS configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <bandwidth>
      <inbound average='1000' peak='5000' floor='200' burst='1024'/>
      <outbound average='128' peak='256' burst='256'/>
    </bandwidth>
  </interface>
<devices>
...
```

Figure 20.51. Quality of service

20.16.9.15. Setting VLAN tag (on supported network types only)

To specify the VLAN tag configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='bridge'>
    <vlan>
      <tag id='42'/>
    </vlan>
    <source bridge='ovsbr0'/>
    <virtualport type='openvswitch'>
      <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
    </virtualport>
  </interface>
<devices>
...
```

Figure 20.52. Setting VLAN tag (on supported network types only)

If (and only if) the network connection used by the guest virtual machine supports vlan tagging transparent to the guest virtual machine, an optional **vlan** element can specify one or more vlan tags to apply to the guest virtual machine's network traffic (openvswitch and **type='hostdev'** SR-IOV interfaces do support transparent vlan tagging of guest virtual machine traffic; everything else, including standard Linux bridges and libvirt's own virtual networks, do not support it. 802.1Qbh (vn-link) and 802.1Qbg (VEPA) switches provide their own way (outside of libvirt) to tag guest virtual machine traffic onto specific vlans.) To allow for specification of multiple tags (in the case of vlan trunking), a subelement, **tag**, specifies which vlan tag to use (for example: **tag id='42'**/. If an interface has more than one **vlan** element defined, it is assumed that the user wants to do VLAN trunking using all the specified tags. In the case that vlan trunking with a single tag is desired, the optional attribute **trunk='yes'** can be added to the toplevel vlan element.

20.16.9.16. Modifying virtual link state

This element provides means of setting state of the virtual network link. Possible values for attribute **state** are **up** and **down**. If **down** is specified as the value, the interface behaves as if it had the network cable disconnected. Default behavior if this element is unspecified is to have the link state **up**.

To specify the virtual link state configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <link state='down'/>
  </interface>
<devices>
...
```

Figure 20.53. Modifying virtual link state

20.16.10. Input Devices

Input devices allow interaction with the graphical framebuffer in the guest virtual machine virtual machine. When enabling the framebuffer, an input device is automatically provided. It may be possible to add additional devices explicitly, for example, to provide a graphics tablet for absolute cursor movement.

To specify the input devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <input type='mouse' bus='usb'/>
</devices>
...
```

Figure 20.54. Input devices

The **<input>** element has one mandatory attribute: **type** which can be set to: **mouse** or **tablet**. The latter provides absolute cursor movement, while the former uses relative movement. The optional **bus** attribute can be used to refine the exact device type and can be set to: **xen** (paravirtualized), **ps2**, and **usb**.

The input element has an optional sub-element **<address>**, which can tie the device to a particular PCI slot, as documented above.

20.16.11. Hub Devices

A hub is a device that expands a single port into several so that there are more ports available to connect devices to a host physical machine system.

To specify the hub devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <hub type='usb'/>
</devices>
...
```

Figure 20.55. Hub devices

The hub element has one mandatory attribute, the type whose value can only be **usb**. The hub element has an optional sub-element **address** with **type='usb'**which can tie the device to a particular controller.

20.16.12. Graphical framebuffers

A graphics device allows for graphical interaction with the guest virtual machine OS. A guest virtual machine will typically have either a framebuffer or a text console configured to allow interaction with the admin.

To specify the graphical framebuffer devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <graphics type='sdl' display=':0.0'/>
  <graphics type='vnc' port='5904'>
    <listen type='address' address='192.0.2.1'/>
  </graphics>
  <graphics type='rdp' autoport='yes' multiUser='yes' />
  <graphics type='desktop' fullscreen='yes'/>
  <graphics type='spice'>
    <listen type='network' network='rednet'/>
  </graphics>
</devices>
...
```

Figure 20.56. Graphical Framebuffers

The **graphics** element has a mandatory **type** attribute which takes the value **sdl**, **vnc**, **rdp** or **desktop** as explained below:

Table 20.20. Graphical framebuffer elements

Parameter	Description
sdl	This displays a window on the host physical machine desktop, it can take 3 optional arguments: a **display** attribute for the display to use, an **xauth** attribute for the authentication identifier, and an optional **fullscreen** attribute accepting values **yes** or **no**

Parameter	Description
vnc	Starts a VNC server. The **port** attribute specifies the TCP port number (with **-1** as legacy syntax indicating that it should be auto-allocated). The **autoport** attribute is the new preferred syntax for indicating autoallocation of the TCP port to use. The **listen** attribute is an IP address for the server to listen on. The **passwd** attribute provides a VNC password in clear text. The **keymap** attribute specifies the keymap to use. It is possible to set a limit on the validity of the password be giving an **timestamp passwdValidTo='2010-04-09T15:51:00'** assumed to be in UTC. The **connected** attribute allows control of connected client during password changes. VNC accepts **keep** value only and note that it may not be supported by all hypervisors. Rather than using listen/port, QEMU supports a socket attribute for listening on a unix domain socket path.
spice	Starts a SPICE server. The **port** attribute specifies the TCP port number (with -1 as legacy syntax indicating that it should be auto-allocated), while **tlsPort** gives an alternative secure port number. The **autoport** attribute is the new preferred syntax for indicating auto-allocation of both port numbers. The **listen** attribute is an IP address for the server to listen on. The **passwd** attribute provides a SPICE password in clear text. The **keymap** attribute specifies the keymap to use. It is possible to set a limit on the validity of the password be giving an **timestamp passwdValidTo='2010-04-09T15:51:00'** assumed to be in UTC. The **connected** attribute allows control of connected client during password changes. SPICE accepts keep to keep client connected, disconnect to disconnect client and fail to fail changing password. Note it is not be supported by all hypervisors. The **defaultMode** attribute sets the default channel security policy, valid values are **secure**, **insecure** and the default **any** (which is **secure** if possible, but falls back to **insecure** rather than erroring out if no secure path is available).

When SPICE has both a normal and TLS secured TCP port configured, it may be desirable to restrict what channels can be run on each port. This is achieved by adding one or more **channel** elements inside the main **graphics** element. Valid channel names include **main**, **display**, **inputs**, **cursor**, **playback**, **record**; **smartcard**; and **usbredir**.

To specify the SPICE configuration settings, use a mangement tool to make the following changes to the domain XML:

```
<graphics type='spice' port='-1' tlsPort='-1' autoport='yes'>
  <channel name='main' mode='secure'/>
  <channel name='record' mode='insecure'/>
  <image compression='auto_glz'/>
  <streaming mode='filter'/>
  <clipboard copypaste='no'/>
  <mouse mode='client'/>
</graphics>
```

Figure 20.57. SPICE configuration

SPICE supports variable compression settings for audio, images and streaming. These settings are accessible by using the compression attribute in all following elements: **image** to set image compression (accepts auto_glz, auto_lz, quick, glz, lz, off), **jpeg** for JPEG compression for images over wan (accepts auto, never, always), **zlib** for configuring wan image compression (accepts auto, never, always) and **playback** for enabling audio stream compression (accepts on or off).

Streaming mode is set by the **streaming** element, settings its **mode** attribute to one of **filter**, **all** or **off**.

In addition, Copy and paste functionality (using the SPICE agent) is set by the **clipboard** element. It is enabled by default, and can be disabled by setting the **copypaste** property to **no**.

Mouse mode is set by the **mouse** element, setting its **mode** attribute to one of **server** or **client**. If no mode is specified, the qemu default will be used (**client** mode).

Additional elements include:

Table 20.21. Additional graphical framebuffer elements

Parameter	Description
rdp	Starts a RDP server. The port attribute specifies the TCP port number (with -1 as legacy syntax indicating that it should be auto-allocated). The autoport attribute is the new preferred syntax for indicating autoallocation of the TCP port to use. The replaceUser attribute is a boolean deciding whether multiple simultaneous connections to the VM are permitted. The multiUser whether the existing connection must be dropped and a new connection must be established by the VRDP server, when a new client connects in single connection mode.
desktop	This value is reserved for VirtualBox domains for the moment. It displays a window on the host physical machine desktop, similarly to "sdl", but uses the VirtualBox viewer. Just like "sdl", it accepts the optional attributes display and fullscreen.

Parameter	Description
listen	Rather than putting the address information used to set up the listening socket for graphics types vnc and spice in the **graphics**, the **listen** attribute, a separate subelement of **graphics**, called **listen** can be specified (see the examples above). **listen** accepts the following attributes: • **type** - Set to either address or network. This tells whether this listen element is specifying the address to be used directly, or by naming a network (which will then be used to determine an appropriate address for listening). • **address** - this attribute will contain either an IP address or host name (which will be resolved to an IP address via a DNS query) to listen on. In the "live" XML of a running domain, this attribute will be set to the IP address used for listening, even if **type='network'**. • network - if **type='network'**, the network attribute will contain the name of a network in libvirt's list of configured networks. The named network configuration will be examined to determine an appropriate listen address. For example, if the network has an IPv4 address in its configuration (for instance if it has a forward type of route, nat, or no forward type (isolated)), the first IPv4 address listed in the network's configuration will be used. If the network is describing a host physical machine bridge, the first IPv4 address associated with that bridge device will be used, and if the network is describing one of the 'direct' (macvtap) modes, the first IPv4 address of the first forward dev will be used.

20.16.13. Video Devices

A video device.

To specify the video devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <video>
    <model type='vga' vram='8192' heads='1'>
      <acceleration accel3d='yes' accel2d='yes'/>
    </model>
  </video>
</devices>
...
```

Figure 20.58. Video devices

The **graphics** element has a mandatory **type** attribute which takes the value "sdl", "vnc", "rdp" or "desktop" as explained below:

Table 20.22. Graphical framebuffer elements

Parameter	Description
video	The **video** element is the container for describing video devices. For backwards compatibility, if no video is set but there is a **graphics** element in domain xml, then libvirt will add a default **video** according to the guest virtual machine type. If "ram" or "vram" are not supplied a default value is used.
model	This has a mandatory**type** attribute which takes the value **vga**, **cirrus**, **vmvga**, **xen**, **vbox**, or **qxl** depending on the hypervisor features available. You can also provide the amount of video memory in kibibytes (blocks of 1024 bytes) using vram and the number of figure with heads.
acceleration	If acceleration is supported it should be enabled using the **accel3d** and **accel2d** attributes in the **acceleration** element.
address	The optional address sub-element can be used to tie the video device to a particular PCI slot.

20.16.14. Consoles, Serial, Parallel, and Channel Devices

A character device provides a way to interact with the virtual machine. Paravirtualized consoles, serial ports, parallel ports and channels are all classed as character devices and so represented using the same syntax.

To specify the consols, channel and other devices configuration settings, use a management tool to make the following changes to the domain XML:

```
...
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2'/>
    <target port='0'/>
  </parallel>
  <serial type='pty'>
    <source path='/dev/pts/3'/>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <source path='/dev/pts/4'/>
    <target port='0'/>
  </console>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd'/>
    <target type='guestfwd' address='10.0.2.1' port='4600'/>
  </channel>
</devices>
...
```

Figure 20.59. Consoles, serial, parallel, and channel devices

In each of these directives, the top-level element name (parallel, serial, console, channel) describes how the device is presented to the guest virtual machine. The guest virtual machine interface is configured by the target element. The interface presented to the host physical machine is given in the type attribute of the top-level element. The host physical machine interface is configured by the source element. The source element may contain an optional seclabel to override the way that labelling is done on the socket path. If this element is not present, the security label is inherited from the per-domain setting. Each character device element has an optional sub-element **address** which can tie the device to a particular controller or PCI slot.

20.16.15. Guest Virtual Machine Interfaces

A character device presents itself to the guest virtual machine as one of the following types.

To set the parallel port, use a management tool to make the following change to the domain XML

```
...
<devices>
  <parallel type='pty'>
    <source path='/dev/pts/2'/>
    <target port='0'/>
  </parallel>
</devices>
...
```

Figure 20.60. Guest virtual machine interface Parallel Port

<target> can have a **port** attribute, which specifies the port number. Ports are numbered starting from 0. There are usually 0, 1 or 2 parallel ports.

To set the serial port use a management tool to make the following change to the domain XML:

```
...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3'/>
    <target port='0'/>
  </serial>
</devices>
...
```

Figure 20.61. Guest virtual machine interface serial port

<target> can have a **port** attribute, which specifies the port number. Ports are numbered starting from 0. There are usually 0, 1 or 2 serial ports. There is also an optional **type** attribute, which has two choices for its value, one is **isa-serial**, the other is **usb-serial**. If **type** is missing, **isa-serial** will be used by default. For usb-serial an optional sub-element **<address>** with **type='usb'** can tie the device to a particular controller, documented above.

The **<console>** element is used to represent interactive consoles. Depending on the type of guest virtual machine in use, the consoles might be paravirtualized devices, or they might be a clone of a serial device, according to the following rules:

- If no **targetType** attribute is set, then the default device **type** is according to the hypervisor's rules. The default **type** will be added when re-querying the XML fed into libvirt. For fully virtualized guest virtual machines, the default device type will usually be a serial port.
- If the **targetType** attribute is **serial**, and if no **<serial>** element exists, the console element will be copied to the **<serial>** element. If a **<serial>** element does already exist, the console element will be ignored.
- If the **targetType** attribute is not **serial**, it will be treated normally.
- Only the first **<console>** element may use a **targetType** of **serial**. Secondary consoles must all be paravirtualized.
- On s390, the console element may use a targetType of sclp or sclplm (line mode). SCLP is the native console type for s390. There's no controller associated to SCLP consoles.

In the example below, a virtio console device is exposed in the guest virtual machine as /dev/hvc[0-7] (for more information, see http://fedoraproject.org/wiki/Features/VirtioSerial):

```
...
<devices>
  <console type='pty'>
    <source path='/dev/pts/4'/>
    <target port='0'/>
  </console>

  <!-- KVM virtio console -->
  <console type='pty'>
    <source path='/dev/pts/5'/>
    <target type='virtio' port='0'/>
  </console>
</devices>
...

...
<devices>
  <!-- KVM s390 sclp console -->
  <console type='pty'>
    <source path='/dev/pts/1'/>
    <target type='sclp' port='0'/>
  </console>
</devices>
...
```

Figure 20.62. Guest virtual machine interface - virtio console device

If the console is presented as a serial port, the **<target>** element has the same attributes as for a serial port. There is usually only one console.

20.16.16. Channel

This represents a private communication channel between the host physical machine and the guest virtual machine and is manipulated by making changes to your guest virtual machine virtual machine using a management tool that results in changes made to the following section of the domain xml

```
...
<devices>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd'/>
    <target type='guestfwd' address='10.0.2.1' port='4600'/>
  </channel>

  <!-- KVM virtio channel -->
  <channel type='pty'>
    <target type='virtio' name='arbitrary.virtio.serial.port.name'/>
  </channel>
  <channel type='unix'>
    <source mode='bind' path='/var/lib/libvirt/qemu/f16x86_64.agent'/>
    <target type='virtio' name='org.qemu.guest_agent.0'/>
  </channel>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0'/>
  </channel>
</devices>
...
```

Figure 20.63. Channel

This can be implemented in a variety of ways. The specific type of **<channel>** is given in the **type** attribute of the **<target>** element. Different channel types have different target attributes as follows:

- **guestfwd** - Dictates that TCP traffic sent by the guest virtual machine to a given IP address and port is forwarded to the channel device on the host physical machine. The **target** element must have address and port attributes.

- **virtio** - Paravirtualized virtio channel. **<channel>** is exposed in the guest virtual machine under **/dev/vport***, and if the optional element **name**is specified, **/dev/virtio-ports/$name** (for more info, see http://fedoraproject.org/wiki/Features/VirtioSerial). The optional element **address** can tie the channel to a particular **type='virtio-serial'** controller, documented above. With QEMU, if name is "org.qemu.guest_agent.0", then libvirt can interact with a guest virtual machine agent installed in the guest virtual machine, for actions such as guest virtual machine shutdown or file system quiescing.

- **spicevmc** - Paravirtualized SPICE channel. The domain must also have a SPICE server as a graphics device, at which point the host physical machine piggy-backs messages across the main channel. The **target** element must be present, with attribute **type='virtio';** an optional attribute **name** controls how the guest virtual machine will have access to the channel, and defaults to **name='com.redhat.spice.0'**. The optional **<address>** element can tie the channel to a particular **type='virtio-serial'** controller.

20.16.17. Host Physical Machine Interface

A character device presents itself to the host physical machine as one of the following types:

Table 20.23. Character device elements

Parameter	Description	XML snippet

Domain logfile	Disables all input on the character device, and sends output into the virtual machine's logfile	`<devices>`
		`<console type='stdio'>`
		`<target port='1'/>`
		`</console>`
		`</devices>`

Device logfile	A file is opened and all data sent to the character device is written to the file.	`<devices>`
		`<serial type="file">`
		`<source path="/var/log/vm/vm-serial.log"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

Virtual console	Connects the character device to the graphical framebuffer in a virtual console. This is typically accessed via a special hotkey sequence such as "ctrl+alt+3"	`<devices>`
		`<serial type='vc'>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

Null device	Connects the character device to the void. No data is ever provided to the input. All data written is discarded.	`<devices>`
		`<serial type='null'>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

Parameter	Description	XML snippet

Pseudo TTY	A Pseudo TTY is allocated using **/dev/ptmx**. A suitable client such as **virsh console** can connect to interact with the serial port locally.	`<devices>`
		`<serial type="pty">`
		`<source path="/dev/pts/3"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

NB Special case	NB special case if **<console type='pty'>**, then the TTY path is also duplicated as an attribute **tty='/dev/pts/3'** on the top level **<console>** tag. This provides compat with existing syntax for **<console>** tags.	

Host physical machine device proxy	The character device is passed through to the underlying physical character device. The device types must match, eg the emulated serial port should only be connected to a host physical machine serial port - do not connect a serial port to a parallel port.	`<devices>`
		`<serial type="dev">`
		`<source path="/dev/ttyS0"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

Named pipe	The character device writes output to a named pipe. See pipe(7) man page for more info.	`<devices>`
		`<serial type="pipe">`
		`<source path="/tmp/mypipe"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

TCP client/server	The character device acts as a TCP client connecting to a remote server.	`<devices>`
		`<serial type="tcp">`
		`<source mode="connect" host="0.0.0.0" service="2445"/>`
		`<protocol`

Parameter	Description	XML snippet
		type="raw"/> <target port="1"/> </serial> </devices> Or as a TCP server waiting for a client connection. <devices> <serial type="tcp">\ <source mode="bind" host="127.0.0.1" service="2445"/> <protocol type="raw"/> <target port="1"/> </serial> </devices> Alternatively you can use telnet instead of raw TCP. In addition, you can also use telnets (secure telnet) and tls. <devices> <serial type="tcp">\ <source mode="connect" host="0.0.0.0" service="2445"/> <protocol type="telnet"/> <target port="1"/> </serial> <serial type="tcp">\ <source mode="bind" host="127.0.0.1" service="2445"/> <protocol type="telnet"/> <target port="1"/> </serial> </devices>

Parameter	Description	XML snippet

UDP network console	The character device acts as a UDP netconsole service, sending and receiving packets. This is a lossy service.	`<devices>`
		`<serial type="udp">`
		`<source mode="bind" host="0.0.0.0" service="2445"/>`
		`<source mode="connect" host="0.0.0.0" service="2445"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

UNIX domain socket client/server	The character device acts as a UNIX domain socket server, accepting connections from local clients.	`<devices>`
		`<serial type="unix">`
		`<source mode="bind" path="/tmp/foo"/>`
		`<target port="1"/>`
		`</serial>`
		`</devices>`

20.17. SOUND DEVICES

A virtual sound card can be attached to the host physical machine by using the sound element.

```
...
<devices>
  <sound model='es1370'/>
</devices>
...
```

Figure 20.64. Virtual sound card

The sound element has one mandatory attribute, **model**, which specifies what real sound device is emulated. Valid values are specific to the underlying hypervisor, though typical choices are **'es1370'**, **'sb16'**, **'ac97'**, and **'ich6'**. In addition, a sound element with ich6 model can have optional sub-elements **codec** to attach various audio codecs to the audio device. If not specified, a default codec will be attached to allow playback and recording. Valid values are **'duplex'** (advertises a line-in and a line-out) and **'micro'** (advertises a speaker and a microphone).

```
...
<devices>
  <sound model='ich6'>
    <codec type='micro'/>
  <sound/>
</devices>
...
```

Figure 20.65. Sound devices

Each sound element has an optional sub-element **<address>** which can tie the device to a particular PCI slot, documented above.

20.18. WATCHDOG DEVICE

A virtual hardware watchdog device can be added to the guest virtual machine via the **<watchdog>** element. The watchdog device requires an additional driver and management daemon in the guest virtual machine. As merely enabling the watchdog in the libvirt configuration does not do anything useful on its own. Currently there is no support notification when the watchdog fires.

```
...
<devices>
  <watchdog model='i6300esb'/>
</devices>
...

...
<devices>
  <watchdog model='i6300esb' action='poweroff'/>
</devices>
</domain>
```

Figure 20.66. Watchdog Device

The following attributes are declared in this XML:

- **model** - The required **model** attribute specifies what real watchdog device is emulated. Valid values are specific to the underlying hypervisor.
- The **model** attribute may take the following values:
  - **i6300esb** – the recommended device, emulating a PCI Intel 6300ESB
  - **ib700** – emulates an ISA iBase IB700
- **action** - The optional **action** attribute describes what action to take when the watchdog expires. Valid values are specific to the underlying hypervisor. The **action** attribute can have the following values:
  - **reset** – default setting, forcefully resets the guest virtual machine
  - **shutdown** – gracefully shuts down the guest virtual machine (not recommended)

- **poweroff** – forcefully powers off the guest virtual machine
- **pause** – pauses the guest virtual machine
- **none** – does nothing
- **dump** – automatically dumps the guest virtual machine.

Note that the 'shutdown' action requires that the guest virtual machine is responsive to ACPI signals. In the sort of situations where the watchdog has expired, guest virtual machines are usually unable to respond to ACPI signals. Therefore using 'shutdown' is not recommended. In addition, the directory to save dump files can be configured by auto_dump_path in file /etc/libvirt/qemu.conf.

20.19. MEMORY BALLOON DEVICE

A virtual memory balloon device is added to all Xen and KVM/QEMU guest virtual machines. It will be seen as **<memballoon>** element. It will be automatically added when appropriate, so there is no need to explicitly add this element in the guest virtual machine XML unless a specific PCI slot needs to be assigned. Note that if the memballoon device needs to be explicitly disabled, **model='none'** may be used.

The following example automatically added device with KVM

```
...
<devices>
  <memballoon model='virtio'/>
</devices>
...
```

Figure 20.67. Memory balloon device

Here is an example where the device is added manually with static PCI slot 2 requested

```
...
<devices>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </memballoon>
</devices>
</domain>
```

Figure 20.68. Memory balloon device added manually

The required **model** attribute specifies what type of balloon device is provided. Valid values are specific to the virtualization platform are: **'virtio'** which is the default setting with the KVM hypervisor or **'xen'** which is the default setting with the Xen hypervisor.

20.20. SECURITY LABEL

The **<seclabel>** element allows control over the operation of the security drivers. There are three basic modes of operation, **'dynamic'** where libvirt automatically generates a unique security label, **'static'** where the application/administrator chooses the labels, or **'none'** where confinement is disabled. With

dynamic label generation, libvirt will always automatically relabel any resources associated with the virtual machine. With static label assignment, by default, the administrator or application must ensure labels are set correctly on any resources. However, automatic relabeling can be enabled if desired.

If more than one security driver is used by libvirt, multiple seclabel tags can be used, one for each driver and the security driver referenced by each tag can be defined using the attribute **model** Valid input XML configurations for the top-level security label are:

```
<seclabel type='dynamic' model='selinux'/>

<seclabel type='dynamic' model='selinux'>
  <baselabel>system_u:system_r:my_svirt_t:s0</baselabel>
</seclabel>

<seclabel type='static' model='selinux' relabel='no'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='static' model='selinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='none'/>
```

Figure 20.69. Security label

If no **'type'** attribute is provided in the input XML, then the security driver default setting will be used, which may be either **'none'** or **'dynamic'**. If a **<baselabel>** is set but no **'type'** is set, then the type is presumed to be **'dynamic'**. When viewing the XML for a running guest virtual machine with automatic resource relabeling active, an additional XML element, **imagelabel**, will be included. This is an output-only element, so will be ignored in user supplied XML documents.

The following elements can be manipulated with the following values:

- **type** - Either **static**, **dynamic** or **none** to determine whether libvirt automatically generates a unique security label or not.
- **model** - A valid security model name, matching the currently activated security model
- **relabel** - Either **yes** or **no**. This must always be **yes** if dynamic label assignment is used. With static label assignment it will default to **no**.
- **<label>** - If static labelling is used, this must specify the full security label to assign to the virtual domain. The format of the content depends on the security driver in use:
  - **SELinux**: a SELinux context.
  - **AppArmor**: an AppArmor profile.
  - **DAC**: owner and group separated by colon. They can be defined both as user/group names or uid/gid. The driver will first try to parse these values as names, but a leading plus sign can used to force the driver to parse them as uid or gid.
- **<baselabel>** - If dynamic labelling is used, this can optionally be used to specify the base security label. The format of the content depends on the security driver in use.

- **<imagelabel>** - This is an output only element, which shows the security label used on resources associated with the virtual domain. The format of the content depends on the security driver in use When relabeling is in effect, it is also possible to fine-tune the labeling done for specific source file names, by either disabling the labeling (useful if the file lives on NFS or other file system that lacks security labeling) or requesting an alternate label (useful when a management application creates a special label to allow sharing of some, but not all, resources between domains). When a seclabel element is attached to a specific path rather than the top-level domain assignment, only the attribute relabel or the sub-element label are supported.

20.21. EXAMPLE DOMAIN XML CONFIGURATION

QEMU emulated guest virtual machine on AMD64 and Intel

```
<domain type='qemu'>
  <name>QEmu-fedora-i686</name>
  <uuid>c7a5fdbd-cdaf-9455-926a-d65c16db1809</uuid>
  <memory>219200</memory>
  <currentMemory>219200</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='cdrom'/>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='cdrom'>
      <source file='/home/user/boot.iso'/>
      <target dev='hdc'/>
      <readonly/>
    </disk>
    <disk type='file' device='disk'>
      <source file='/home/user/fedora.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <graphics type='vnc' port='-1'/>
  </devices>
</domain>
```

Figure 20.70. Example domain XML config

KVM hardware accelerated guest virtual machine on i686

```
<domain type='kvm'>
  <name>demo2</name>
  <uuid>4dea24b3-1d52-d8f3-2516-782e98a23fa0</uuid>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="i686">hvm</type>
  </os>
  <clock sync="localtime"/>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/demo2.img'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <mac address='24:42:53:21:52:45'/>
    </interface>
    <graphics type='vnc' port='-1' keymap='de'/>
  </devices>
</domain>
```

Figure 20.71. Example domain XML config

CHAPTER 21. TROUBLESHOOTING

This chapter covers common problems and solutions for Red Hat Enterprise Linux 6 virtualization issues.

Read this chapter to develop an understanding of some of the common problems associated with virtualization technologies. Troubleshooting takes practice and experience which are difficult to learn from a book. It is recommended that you experiment and test virtualization on Red Hat Enterprise Linux 6 to develop your troubleshooting skills.

If you cannot find the answer in this document there may be an answer online from the virtualization community. Refer to Section B.1, "Online Resources" for a list of Linux virtualization websites.

21.1. DEBUGGING AND TROUBLESHOOTING TOOLS

This section summarizes the System Administrator applications, the networking utilities, and debugging tools. You can employ these standard System administration tools and logs to assist with troubleshooting:

- **kvm_stat** - refer to Section 21.4, "kvm_stat"
- **trace-cmd**
- **ftrace** Refer to the *Red Hat Enterprise Linux Developer Guide*
- **vmstat**
- **iostat**
- **lsof**
- **systemtap**
- **crash**
- **sysrq**
- **sysrq t**
- **sysrq w**

These networking tools can assist with troubleshooting virtualization networking problems:

- **ifconfig**
- **tcpdump**

  The **tcpdump** command 'sniffs' network packets. **tcpdump** is useful for finding network abnormalities and problems with network authentication. There is a graphical version of **tcpdump** named **wireshark**.

- **brctl**

  brctl is a networking tool that inspects and configures the Ethernet bridge configuration in the Linux kernel. You must have root access before performing these example commands:

  ```
  # brctl show
  ```

```
bridge-name    bridge-id          STP  enabled  interfaces
-----------------------------------------------------------------------------
virtbr0            8000.feffffff       yes        eth0

# brctl showmacs virtbr0
port-no           mac-addr                  local?       aging timer
1                 fe:ff:ff:ff:ff:           yes           0.00
2                 fe:ff:ff:fe:ff:           yes           0.00
# brctl showstp virtbr0
virtbr0
bridge-id              8000.fefffffffff
designated-root        8000.fefffffffff
root-port              0                    path-cost            0
max-age                20.00                bridge-max-age       20.00
hello-time             2.00                 bridge-hello-time    2.00
forward-delay          0.00                 bridge-forward-delay 0.00
aging-time             300.01
hello-timer            1.43                 tcn-timer            0.00
topology-change-timer  0.00                 gc-timer             0.02
```

Listed below are some other useful commands for troubleshooting virtualization.

- **strace** is a command which traces system calls and events received and used by another process.
- **vncviewer**: connect to a VNC server running on your server or a virtual machine. Install **vncviwer** using the **yum install tigervnc** command.
- **vncserver**: start a remote desktop on your server. Gives you the ability to run graphical user interfaces such as virt-manager via a remote session. Install **vncserver** using the **yum install tigervnc-server** command.

21.2. PREPARING FOR DISASTER RECOVERY

If at all possible, it is best to prepare for situations where your equipment will be compromised due to weather, or other reasons. It is highly recommended that you perform a backup of the following files and directories on your host physical machine:

- From the **/etc/libvirt** directory, all files.
- From the **/var/lib/libvirt** directory, back up the following items:
  - The current dnsmasq DHCP leases found in **/var/lib/libvirt/dnsmasq**
  - The running virtual network configuration files found in **/var/lib/libvirt/network**
  - Guest virtual machine files created by **virt-manager** when saving the current state of a guest, if there are any. These can be found in the **/var/lib/libvirt/qemu/save/** directory. If the virsh **save command** was used to create virtual machines, the files can be found in the location specified for **virsh save** by the user.
  - The guest virtual machine snapshot files, created by the **qemu-img create** and **virsh snapshot-create** commands and found in the locations specified for the commands by the user.
  - Guest virtual machine disk images created by **virt-manager**, if any, which can be found in

the **/var/lib/libvirt/images/** directory. If the **virsh pool-define** command was used to create virtual storage, the image file can be found in the location specified for **virsh pool-define** by the user. For instructions on how to back up the guest image files, use the steps described in Procedure 21.1, "Creating a backup of the guest virtual machine's disk image for disaster recovery purposes".

- If you are using bridges, you will also need to back up the files located in **/etc/sysconfig/network-scripts/ifcfg-<bridge_name>**
- Optionally, the guest virtual machine core dump files found in **/var/lib/libvirt/qemu/dump** can also be backed up to be used for analysing the causes of the failure. Note, however, that these files can be very large for some systems.

Procedure 21.1. Creating a backup of the guest virtual machine's disk image for disaster recovery purposes

This procedure will cover how to back up several different disk image types.

1. To back up only the guest virtual machine disk image, back up the files located in **/var/lib/libvirt/images**. To back up guest virtual machine disk images with LVM logical volumes, run the following command:

   ```
   # lvcreate --snapshot --name snap --size 8G /dev/vg0/data
   ```

   This command creates a snapshot volume named snap with a size of 8G as part of a 64G volume.

2. Create a file for the snapshots using a command similar to this one:

   ```
   # mkdir /mnt/virt.snapshot
   ```

3. Mount the directory you created and the snapshot volume using the following command:

   ```
   # mount /dev/vg0/snap /mnt/virt.snapshot
   ```

4. Use one of the following commands to back up the volume:

   a. ```
      # tar -pzc -f /mnt/backup/virt-snapshot-MM-DD-YYYY.tgz
      /mnt/virt.snapshot++++++++++++
      ```

   b. ```
      # rsync -a /mnt/virt.snapshot/ /mnt/backup/virt-snapshot.MM-DD-YYYY/
      ```

21.3. CREATING VIRSH DUMP FILES

Executing a **virsh dump** command sends a request to dump the core of a guest virtual machine to a file so errors in the virtual machine can be diagnosed. Running this command may require you to manually ensure proper permissions on file and path specified by the argument **corefilepath**. The **virsh dump** command is similar to a coredump (or the **crash** utility). To create the **virsh dump** file, run:

```
#virsh dump <domain> <corefilepath> [--bypass-cache] { [--live] | [--crash] | [--reset] } [--verbose] [--
memory-only]
```

While the domain (guest virtual machine domain name) and corefilepath (location of the newly created core dump file) are mandatory, the following arguments are optional:

- **--live** creates a dump file on a running machine and does not pause it.
- **--crash** stops the guest virtual machine and generates the dump file. The main difference is that the guest virtual machine will not be listed as Stopped, with the reason as Crashed. Note that in **virt-manager** the status will be listed as Paused.
- **--reset** will reset the guest virtual machine following a successful dump. Note, these three switches are mutually exclusive.
- **--bypass-cache** uses O_DIRECT to bypass the file system cache.
- **--memory-only** the dump file will be saved as an elf file, and will only include domain's memory and cpu common register value. This option is very useful if the domain uses host devices directly.
- **--verbose** displays the progress of the dump

The entire dump process may be monitored using **virsh domjobinfo** command and can be canceled by running **virsh domjobabort**.

21.4. KVM_STAT

The **kvm_stat** command is a python script which retrieves runtime statistics from the **kvm** kernel module. The **kvm_stat** command can be used to diagnose guest behavior visible to **kvm**. In particular, performance related issues with guests. Currently, the reported statistics are for the entire system; the behavior of all running guests is reported. To run this script you need to install the qemu-kvm-tools package.

The **kvm_stat** command requires that the **kvm** kernel module is loaded and **debugfs** is mounted. If either of these features are not enabled, the command will output the required steps to enable **debugfs** or the **kvm** module. For example:

```
# kvm_stat
Please mount debugfs ('mount -t debugfs debugfs /sys/kernel/debug')
and ensure the kvm modules are loaded
```

Mount **debugfs** if required:

```
# mount -t debugfs debugfs /sys/kernel/debug
```

kvm_stat Output

The **kvm_stat** command outputs statistics for all guests and the host. The output is updated until the command is terminated (using **Ctrl**+**c** or the **q** key).

```
# kvm_stat

kvm statistics

efer_reload              94       0
exits               4003074   31272
fpu_reload          1313881   10796
```

```
halt_exits              14050     259
halt_wakeup              4496     203
host_state_reload 1638354   24893
hypercalls                  0       0
insn_emulation         1093850    1909
insn_emulation_fail          0       0
invlpg                   75569       0
io_exits               1596984   24509
irq_exits                21013     363
irq_injections           48039    1222
irq_window               24656     870
largepages                   0       0
mmio_exits               11873       0
mmu_cache_miss           42565       8
mmu_flooded              14752       0
mmu_pde_zapped           58730       0
mmu_pte_updated              6       0
mmu_pte_write           138795       0
mmu_recycled                 0       0
mmu_shadow_zapped        40358       0
mmu_unsync                 793       0
nmi_injections               0       0
nmi_window                   0       0
pf_fixed                697731    3150
pf_guest                279349       0
remote_tlb_flush             5       0
request_irq                  0       0
signal_exits                 1       0
tlb_flush               200190       0
```

Explanation of variables:

efer_reload

The number of Extended Feature Enable Register (EFER) reloads.

exits

The count of all **VMEXIT** calls.

fpu_reload

The number of times a **VMENTRY** reloaded the FPU state. The **fpu_reload** is incremented when a guest is using the Floating Point Unit (FPU).

halt_exits

Number of guest exits due to **halt** calls. This type of exit is usually seen when a guest is idle.

halt_wakeup

Number of wakeups from a **halt**.

host_state_reload

Count of full reloads of the host state (currently tallies MSR setup and guest MSR reads).

hypercalls

Number of guest hypervisor service calls.

insn_emulation

Number of guest instructions emulated by the host.

insn_emulation_fail

Number of failed **insn_emulation** attempts.

io_exits

Number of guest exits from I/O port accesses.

irq_exits

Number of guest exits due to external interrupts.

irq_injections

Number of interrupts sent to guests.

irq_window

Number of guest exits from an outstanding interrupt window.

largepages

Number of large pages currently in use.

mmio_exits

Number of guest exits due to memory mapped I/O (MMIO) accesses.

mmu_cache_miss

Number of KVM MMU shadow pages created.

mmu_flooded

Detection count of excessive write operations to an MMU page. This counts detected write operations not of individual write operations.

mmu_pde_zapped

Number of page directory entry (PDE) destruction operations.

mmu_pte_updated

Number of page table entry (PTE) destruction operations.

mmu_pte_write

Number of guest page table entry (PTE) write operations.

mmu_recycled

Number of shadow pages that can be reclaimed.

mmu_shadow_zapped

Number of invalidated shadow pages.

mmu_unsync

Number of non-synchronized pages which are not yet unlinked.

nmi_injections

Number of Non-maskable Interrupt (NMI) injections to the guest.

nmi_window

Number of guest exits from (outstanding) Non-maskable Interrupt (NMI) windows.

pf_fixed

Number of fixed (non-paging) page table entry (PTE) maps.

pf_guest

Number of page faults injected into guests.

remote_tlb_flush

Number of remote (sibling CPU) Translation Lookaside Buffer (TLB) flush requests.

request_irq

Number of guest interrupt window request exits.

signal_exits

Number of guest exits due to pending signals from the host.

tlb_flush

Number of **tlb_flush** operations performed by the hypervisor.

NOTE

The output information from the **kvm_stat** command is exported by the KVM hypervisor as pseudo files located in the **/sys/kernel/debug/kvm/** directory.

21.5. GUEST VIRTUAL MACHINE FAILS TO SHUTDOWN

Traditionally, executing a **virsh shutdown** command causes a power button ACPI event to be sent, thus copying the same action as when someone presses a power button on a physical machine. Within every physical machine, it is up to the OS to handle this event. In the past operating systems would just silently shutdown. Today, the most usual action is to show a dialog asking what should be done. Some operating systems even ignore this event completely, especially when no users are logged in. When such operating systems are installed on a guest virtual machine, running **virsh shutdown** just does not work (it is either ignored or a dialog is shown on a virtual display). However, if a qemu-guest-agent channel is added to a guest virtual machine and this agent is running inside the guest virtual machine's OS, the **virsh shutdown** command will ask the agent to shutdown the guest OS instead of sending the ACPI event. The agent will call for a shutdown from inside the guest virtual machine OS and everything works as expected.

Procedure 21.2. Configuring the guest agent channel in a guest virtual machine

1. Stop the guest virtual machine.

2. Open the Domain XML for the guest virtual machine and add the following snippet:

```
<channel type='unix'>
   <source mode='bind'/>
   <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

Figure 21.1. Configuring the guest agent channel

3. Start the guest virtual machine, by running **virsh start [domain]**.

4. Install qemu-guest-agent on the guest virtual machine (**yum install qemu-guest-agent**) and make it run automatically at every boot as a service (qemu-guest-agent.service). Refer to Chapter 10, *QEMU-img and QEMU Guest Agent* for more information.

21.6. TROUBLESHOOTING WITH SERIAL CONSOLES

Linux kernels can output information to serial ports. This is useful for debugging kernel panics and hardware issues with video devices or headless servers. The subsections in this section cover setting up serial console output for host physical machines using the KVM hypervisor.

This section covers how to enable serial console output for fully virtualized guests.

Fully virtualized guest serial console output can be viewed with the **virsh console** command.

Be aware fully virtualized guest serial consoles have some limitations. Present limitations include:

- output data may be dropped or scrambled.

The serial port is called **ttyS0** on Linux or **COM1** on Windows.

You must configure the virtualized operating system to output information to the virtual serial port.

To output kernel information from a fully virtualized Linux guest into the domain, modify the **/boot/grub/grub.conf** file. Append the following to the ***kernel*** line: ***console=tty0 console=ttyS0,115200***.

```
title Red Hat Enterprise Linux Server (2.6.32-36.x86-64)
 root (hd0,0)
 kernel /vmlinuz-2.6.32-36.x86-64 ro root=/dev/volgroup00/logvol00 \
 console=tty0 console=ttyS0,115200
 initrd /initrd-2.6.32-36.x86-64.img
```

Reboot the guest.

On the host, access the serial console with the following command:

```
# virsh console
```

You can also use **virt-manager** to display the virtual text console. In the guest console window, select **Serial 1** in **Text Consoles** from the **View** menu.

21.7. VIRTUALIZATION LOG FILES

- Each fully virtualized guest log is in the **/var/log/libvirt/qemu/** directory. Each guest log is named as *GuestName*.log and will be periodically compressed once a size limit is reached.

If you encounter any errors with the Virtual Machine Manager, you can review the generated data in the **virt-manager.log** file that resides in the **$HOME/.virt-manager** directory.

21.8. LOOP DEVICE ERRORS

If file-based guest images are used you may have to increase the number of configured loop devices. The default configuration allows up to eight active loop devices. If more than eight file-based guests or loop devices are needed the number of loop devices configured can be adjusted in the **/etc/modprobe.d**/directory. Add the following line:

```
options loop max_loop=64
```

This example uses 64 but you can specify another number to set the maximum loop value. You may also have to implement loop device backed guests on your system. To use a loop device backed guests for a full virtualized system, use the **phy: device** or **file: file** commands.

21.9. LIVE MIGRATION ERRORS

There may be cases where a live migration causes the memory contents to be re-transferred over and over again. This process causes the guest to be in a state where it is constantly writing to memory and therefore will slow down migration. If this should occur, and the guest is writing more than several tens of MBs per second, then live migration may fail to finish (converge). This issue is not scheduled to be resolved at the moment for Red Hat Enterprise Linux 6, and is scheduled to be fixed in Red Hat Enterprise Linux 7.

The current live-migration implementation has a default migration time configured to 30ms. This value determines the guest pause time at the end of the migration in order to transfer the leftovers. Higher values increase the odds that live migration will converge

21.10. ENABLING INTEL VT-X AND AMD-V VIRTUALIZATION HARDWARE EXTENSIONS IN BIOS

NOTE

To expand your expertise, you might also be interested in the Red Hat Virtualization (RH318) training course.

This section describes how to identify hardware virtualization extensions and enable them in your BIOS if they are disabled.

The Intel VT-x extensions can be disabled in the BIOS. Certain laptop vendors have disabled the Intel VT-x extensions by default in their CPUs.

The virtualization extensions cannot be disabled in the BIOS for AMD-V.

Refer to the following section for instructions on enabling disabled virtualization extensions.

Verify the virtualization extensions are enabled in BIOS. The BIOS settings for Intel VT or AMD-V are usually in the **Chipset** or **Processor** menus. The menu names may vary from this guide, the virtualization extension settings may be found in **Security Settings** or other non standard menu names.

Procedure 21.3. Enabling virtualization extensions in BIOS

1. Reboot the computer and open the system's BIOS menu. This can usually be done by pressing the **delete** key, the **F1** key or **Alt** and **F4** keys depending on the system.

2. **Enabling the virtualization extensions in BIOS**

   NOTE

   Many of the steps below may vary depending on your motherboard, processor type, chipset and OEM. Refer to your system's accompanying documentation for the correct information on configuring your system.

   a. Open the **Processor** submenu The processor settings menu may be hidden in the **Chipset**, **Advanced CPU Configuration** or **Northbridge**.

   b. Enable **Intel Virtualization Technology** (also known as Intel VT-x). **AMD-V** extensions cannot be disabled in the BIOS and should already be enabled. The virtualization extensions may be labeled **Virtualization Extensions**, **Vanderpool** or various other names depending on the OEM and system BIOS.

   c. Enable Intel VT-d or AMD IOMMU, if the options are available. Intel VT-d and AMD IOMMU are used for PCI device assignment.

   d. Select **Save & Exit**.

3. Reboot the machine.

4. When the machine has booted, run **cat /proc/cpuinfo |grep -E "vmx|svm"**. Specifying ***--color*** is optional, but useful if you want the search term highlighted. If the command outputs, the virtualization extensions are now enabled. If there is no output your system may not have the virtualization extensions or the correct BIOS setting enabled.

21.11. KVM NETWORKING PERFORMANCE

By default, KVM virtual machines are assigned a virtual Realtek 8139 (rtl8139) NIC (network interface controller). Whereas Red Hat Enterprise Linux guests are assigned a virtio NIC by default, Windows guests or the guest type is not specified.

The rtl8139 virtualized NIC works fine in most environments,but this device can suffer from performance degradation problems on some networks, such as a 10 Gigabit Ethernet.

To improve performance, you can switch to the paravirtualized network driver.

NOTE

Note that the virtualized Intel PRO/1000 (**e1000**) driver is also supported as an emulated driver choice. To use the **e1000** driver, replace **virtio** in the procedure below with **e1000**. For the best performance it is recommended to use the **virtio** driver.

Procedure 21.4. Switching to the virtio driver

1. Shutdown the guest operating system.
2. Edit the guest's configuration file with the **virsh** command (where ***GUEST*** is the guest's name):

```
# virsh edit GUEST
```

   The **virsh edit** command uses the **$EDITOR** shell variable to determine which editor to use.

3. Find the network interface section of the configuration. This section resembles the snippet below:

```
<interface type='network'>
  [output truncated]
  <model type='rtl8139' />
</interface>
```

4. Change the type attribute of the model element from ***'rtl8139'*** to ***'virtio'***. This will change the driver from the rtl8139 driver to the e1000 driver.

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

5. Save the changes and exit the text editor
6. Restart the guest operating system.

Creating New Guests Using Other Network Drivers

Alternatively, new guests can be created with a different network driver. This may be required if you are having difficulty installing guests over a network connection. This method requires you to have at least one guest already created (possibly installed from CD or DVD) to use as a template.

1. Create an XML template from an existing guest (in this example, named *Guest1*):

```
# virsh dumpxml Guest1 > /tmp/guest-template.xml
```

2. Copy and edit the XML file and update the unique fields: virtual machine name, UUID, disk image, MAC address, and any other unique parameters. Note that you can delete the UUID and MAC address lines and virsh will generate a UUID and MAC address.

```
# cp /tmp/guest-template.xml /tmp/new-guest.xml
# vi /tmp/new-guest.xml
```

   Add the model line in the network interface section:

```
<interface type='network'>
 [output truncated]
 <model type='virtio' />
</interface>
```

3. Create the new virtual machine:

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

21.12. WORKAROUND FOR CREATING EXTERNAL SNAPSHOTS WITH LIBVIRT

There are two classes of snapshots for QEMU guests. Internal snapshots are contained completely within a qcow2 file, and fully supported by libvirt, allowing for creating, deleting, and reverting of snapshots. This is the default setting used by libvirt when creating a snapshot, especially when no option is specified. Although this file type takes a bit longer than others in creating the snapshot, it is required by libvirt to use qcow2 disks. Another drawback to this file type is that qcow2 disks are not subject to receive improvements from QEMU.

External snapshots, on the other hand work with any type of original disk image, can be taken with no guest downtime, and are able to receive active improvements from QEMU. In libvirt, they are created when using the **--disk-only** option to **snapshot-create-as** (or when specifying an explicit XML file to **snapshot-create** that does the same). At the moment external snapshots are a one-way operation as libvirt can create them but cannot do anything further with them.

21.13. MISSING CHARACTERS ON GUEST CONSOLE WITH JAPANESE KEYBOARD

On a Red Hat Enterprise Linux 6 host, connecting a Japanese keyboard locally to a machine may result in typed characters such as the underscore (the _ character) not being displayed correctly in guest consoles. This occurs because the required keymap is not set correctly by default.

With Red Hat Enterprise Linux 3 and Red Hat Enterprise Linux 6 guests, there is usually no error message produced when pressing the associated key. However, Red Hat Enterprise Linux 4 and Red Hat Enterprise Linux 5 guests may display an error similar to the following:

```
atkdb.c: Unknown key pressed (translated set 2, code 0x0 on isa0060/serio0).
atkbd.c: Use 'setkeycodes 00 <keycode>' to make it known.
```

To fix this issue in **virt-manager**, perform the following steps:

- Open the affected guest in **virt-manager**.
- Click **View** → **Details**.
- Select **Display VNC** in the list.
- Change **Auto** to **ja** in the **Keymap** pull-down menu.
- Click the **Apply** button.

Alternatively, to fix this issue using the **virsh edit** command on the target guest:

- Run **virsh edit <target guest>**
- Add the following attribute to the <graphics> tag: **keymap='ja'**. For example:

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

21.14. VERIFYING VIRTUALIZATION EXTENSIONS

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

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

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

2. Analyze the output.

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

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

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

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

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

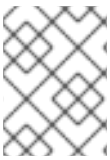
   The "**flags:**" output content may appear multiple times, once for each hyperthread, core or CPU on the system.

   The virtualization extensions may be disabled in the BIOS. If the extensions do not appear or full virtualization does not work refer to Procedure 21.3, “Enabling virtualization extensions in BIOS” .

3. **Ensure KVM subsystem is loaded**
   As an additional check, verify that the **kvm** modules are loaded in the kernel:

```
# lsmod | grep kvm
```

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



NOTE

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

APPENDIX A. THE VIRTUAL HOST METRICS DAEMON (VHOSTMD)

vhostmd (the Virtual Host Metrics Daemon) allows virtual machines to see limited information about the host they are running on. This daemon is only supplied with Red Hat Enterprise Linux for SAP.

In the host, a daemon (**vhostmd**) runs which writes metrics periodically into a disk image. This disk image is exported read-only to guest virtual machines. Guest virtual machines can read the disk image to see metrics. Simple synchronization stops guest virtual machines from seeing out of date or corrupt metrics.

The system administrator chooses which metrics are available for use on a per guest virtual machine basis. In addition, the system administrator may block one or more guest virtual machines from having any access to metric configurations.

Customers who want to use **vhostmd** and **vm-dump-metrics** therefore need subscriptions for "RHEL for SAP Business Applications" to be able to subscribe their RHEL systems running SAP to the "RHEL for SAP" channel on the Customer Portal or Red Hat Subscription Management to install the packages. The following kbase article in the customer portal describes the setup of vhostmd on RHEL: https://access.redhat.com/knowledge/solutions/41566

APPENDIX B. ADDITIONAL RESOURCES

To learn more about virtualization and Red Hat Enterprise Linux, refer to the following resources.

B.1. ONLINE RESOURCES

- http://www.libvirt.org/ is the official website for the **libvirt** virtualization API.
- https://virt-manager.org/ is the project website for the **Virtual Machine Manager** (virt-manager), the graphical application for managing virtual machines.
- Red Hat Virtualization - http://www.redhat.com/products/cloud-computing/virtualization/
- Red Hat product documentation - https://access.redhat.com/documentation/en/
- Virtualization technologies overview - http://virt.kernelnewbies.org

B.2. INSTALLED DOCUMENTATION

- **man virsh** and **/usr/share/doc/libvirt-*<version-number>*** – Contains sub-commands and options for the **virsh** virtual machine management utility as well as comprehensive information about the **libvirt** virtualization library API.
- **/usr/share/doc/gnome-applet-vm-*<version-number>*** – Documentation for the GNOME graphical panel applet that monitors and manages locally-running virtual machines.
- **/usr/share/doc/libvirt-python-*<version-number>*** – Provides details on the Python bindings for the **libvirt** library. The **libvirt-python** package allows python developers to create programs that interface with the **libvirt** virtualization management library.
- **/usr/share/doc/python-virtinst-*<version-number>*** – Provides documentation on the **virt-install** command that helps in starting installations of Fedora and Red Hat Enterprise Linux related distributions inside of virtual machines.
- **/usr/share/doc/virt-manager-*<version-number>*** – Provides documentation on the Virtual Machine Manager, which provides a graphical tool for administering virtual machines.

APPENDIX C. REVISION HISTORY

Revision 1-502	**Mon Mar 08 2017**	**Jiri Herrmann**
Updates for the 6.9 GA release		
Revision 1-501	**Mon May 02 2016**	**Jiri Herrmann**
Updates for the 6.8 GA release		
Revision 1-500	**Thu Mar 01 2016**	**Jiri Herrmann**
Multiple updates for the 6.8 beta release		
Revision 1-449	**Thu Oct 08 2015**	**Jiri Herrmann**
Cleaned up the Revision History		
Revision 1-447	**Fri Jul 10 2015**	**Dayle Parker**
Updates for the 6.7 GA release.		