Linux on Z and LinuxONE

KVM Virtual Server Management

November 2017

SC34-2752-04

Linux on Z and LinuxONE

KVM Virtual Server Management

November 2017

SC34-2752-04

This edition applies to the Linux on z Systems Development stream, libvirt version, and QEMU release as available at the time of writing, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document ix
How this document is organized ix
Conventions and assumptions used in this publication x
Where to get more information x
Other publications for Linux on z Systems and LinuxONE. xi

Part 1. General concepts 1

Chapter 1. Overview 3
Virtual server management tasks 4
Virtualization components 6

Chapter 2. Virtual block devices 9
DASDs and SCSI disks 9
Image files and logical volumes 14
  Storage pools 14

Chapter 3. SCSI tapes and medium changers as virtual SCSI devices 17

Chapter 4. Network devices as virtual Ethernet devices. 21

Part 2. Device setup 25

Chapter 5. Preparing DASDs 27

Chapter 6. Preparing SCSI disks. 29

Chapter 7. Preparing SCSI tape and medium changer devices 33

Chapter 8. Preparing network devices 37
Creating a network interface. 38
Preparing a network interface for a direct MacVTap connection 40
  Preparing a bonded interface 40
Preparing a virtual switch 43

Part 3. Configuration. 47

Chapter 9. Configuring a virtual server 49
Domain configuration-XML 51
Configuring the boot process 53
  Configuring a DASD or SCSI disk as IPL device 53
  Configuring an ISO image as IPL device. 54
  Configuring a kernel image file as IPL device 55
| Configuring a network IPL device 56
  Example of an initial installation 57
Configuring virtual CPUs 61
  Configuring the number of virtual CPUs 61
  Tuning virtual CPUs 62
  Configuring the CPU model 63
Configuring virtual memory. 65
  Configuring the amount of virtual memory. 65

Tuning virtual memory . . . 65
Configuring the collection of QEMU core dumps . . . 67
Configuring the user space . . . 68
Configuring devices with the virtual server . . . 69
Configuring the console . . . 70
Configuring a watchdog device . . . 71
Disabling protected key encryption . . . 72
Suppressing the automatic configuration of a default memory balloon device . . . 74

Chapter 10. Configuring devices . . . 75
Device configuration-XML . . . 76
Configuring virtual block devices . . . 78
Configuring a DASD or SCSI disk . . . 78
Configuring an image file as storage device . . . 84
Configuring a volume as storage device . . . 86
Configuring virtual SCSI devices . . . 88
Configuring a virtual HBA . . . 88
Configuring a SCSI tape or medium changer device . . . 90
Example of a multipathed SCSI tape and medium changer device configuration . . . 93
Configuring a virtual SCSI-attached CD/DVD drive . . . 95
Configuring virtual Ethernet devices . . . 98
Configuring a MacVTap interface . . . 98
Configuring a virtual switch . . . 100
Configuring a random number generator . . . 102

Chapter 11. Configuring storage pools . . . 103
Storage pool and volume configuration-XMLs . . . 103

| **Chapter 12. Configuring virtual networks . . . 105**

Part 4. Operation . . . 107

Chapter 13. Creating, modifying, and deleting persistent virtual server definitions . . . 109
Defining a virtual server . . . 110
Modifying a virtual server definition . . . 110
Undefining a virtual server . . . 111

Chapter 14. Managing the virtual server life cycle . . . 113
Starting a virtual server . . . 114
Terminating a virtual server . . . 114
Suspending a virtual server . . . 116
Resuming a virtual server . . . 116

Chapter 15. Monitoring virtual servers . . . 119
Browsing virtual servers . . . 120
Displaying information about a virtual server . . . 120
Displaying the current libvirt-internal configuration . . . 122

Chapter 16. Migration . . . 125
Definition of a virtual server on different hosts using the same configuration-XML . . . 126
Live virtual server migration . . . 127
Hypervisor release . . . 127
| IBM Z hardware model . . . 128
Live migration setup . . . 129
Phases of a live migration . . . 133
Performing a live migration . . . 134

Chapter 17. Managing system resources 137
Managing virtual CPUs 138
  Modifying the number of virtual CPUs. 138
  Modifying the virtual CPU weight 141
Managing virtual memory 143

Chapter 18. Managing devices 145
Attaching a device. 146
Detaching a device 147
Replacing a virtual DVD 148
Connecting to the console of a virtual server 149

Chapter 19. Managing storage pools 151
Storage pool management commands 152
Volume management commands 153

| **Chapter 20. Managing virtual networks 155**

Part 5. Best practices and performance considerations 157

Chapter 21. CPU management 159
Linux scheduling 159
CPU weight 160

Chapter 22. Memory management 163

Chapter 23. Storage management. 167
I/O threads 167
Logical volume management 167

Part 6. Diagnostics and troubleshooting 171

Chapter 24. Logging 173
Log messages 173
Specifying the logging level of the libvirt log messages 173

Chapter 25. Dumping 175
Creating a virtual server dump on the host 175
Creating a dump on the virtual server 175

Chapter 26. Collecting performance metrics 177

Part 7. Reference 181

Chapter 27. Virtual server life cycle 183
shut off 184
running 185
paused 186
crashed 187
in shutdown. 188

Chapter 28. Selected libvirt XML elements 189
<adapter> as child element of <source> 191
<address> as child element of <controller>, <disk>, <interface>, and <memballoon> 192
<address> as child element of <hostdev> or <disk> 193
<address> as child element of <source> 194

<backend> . . . 195
<boot> . . . 196
| <bridge> . . . 197
<cipher> . . . 198
<cmdline> . . . 199
<console> . . . 200
<controller> . . . 201
| <cpu> . . . 202
<cputune> . . . 203
<device> . . . 204
<devices> . . . 205
| <dhcp> . . . 206
<disk> . . . 207
<domain> . . . 208
<driver> as child element of <controller> . . . 209
<driver> as child element of <disk> . . . 210
<emulator> . . . 212
| <feature> . . . 213
<format> . . . 214
| <forward> . . . 215
<geometry> . . . 216
<host> . . . 217
<hostdev> . . . 218
<initrd> . . . 219
<interface> . . . 220
<iothreads> . . . 221
| <ip> . . . 222
<kernel> . . . 223
<key> . . . 224
<keywrap> . . . 225
<log> . . . 226
<mac> . . . 227
<memballoon> . . . 228
<memory> . . . 229
<memtune> . . . 231
| <model> as a child element of <cpu> . . . 232
<model> as a child element of <interface> . . . 233
<name> as a child element of <domain> . . . 234
| <name> as a child element of <network> . . . 235
| <network> . . . 236
<on_crash> . . . 237
<on_reboot> . . . 238
<os> . . . 239
<path> as child element of <pool><target> . . . 240
<path> as child element of <volume><target> . . . 241
<pool> . . . 242
<readonly> . . . 244
<rng> . . . 245
<shareable> . . . 246
<shares> . . . 247
<soft_limit> . . . 248
<source> as child element of <disk> . . . 249
<source> as child element of <hostdev> . . . 251
<source> as child element of <interface> . . . 252
<source> as child element of <pool> . . . 253
<target> as child element of <console> . . . 254
<target> as child element of <disk> . . . 255
<target> as child element of <pool> . . . 256
<target> as child element of <volume> . . . 257
<type> . . . 258
<vcpu> . . . 259

<virtualport> as a child element of <interface> . . . 260
| <virtualport> as a child element of <network> . . . 261
<volume> . . . 262
<watchdog> . . . 263

Chapter 29. Selected virsh commands. . . . 265
attach-device . . . 268
change-media . . . 270
console . . . 272
define . . . 273
destroy . . . 274
detach-device . . . 275
domblklist . . . 277
domblkstat . . . 278
domcapabilities. . . . 280
domiflist . . . 281
domifstat . . . 282
dominfo . . . 283
domjobabort. . . . 284
domstate . . . 285
dump . . . 286
dumpxml. . . . 287
edit. . . . 288
inject-nmi . . . 289
iothreadadd . . . 290
iothreaddel . . . 292
iothreadinfo . . . 294
list . . . 295
managedsave . . . 297
memtune . . . 299
migrate . . . 300
migrate-getspeed . . . 303
migrate-setmaxdowntime . . . 304
migrate-setspeed . . . 305
| net-autostart. . . . 306
| net-define . . . 307
| net-destroy . . . 308
| net-dumpxml . . . 309
| net-edit . . . 310
| net-info . . . 311
| net-list. . . . 312
| net-name . . . 313
| net-start . . . 314
| net-undefine. . . . 315
| net-uuid . . . 316
pool-autostart . . . 317
pool-define . . . 318
pool-delete . . . 319
pool-destroy. . . . 320
pool-dumpxml . . . 321
pool-edit . . . 322
pool-info . . . 323
pool-list . . . 324
pool-name . . . 325
pool-refresh . . . 326
pool-start . . . 327
pool-undefine . . . 328
pool-uuid . . . 329
reboot . . . 330
resume . . . 331
schedinfo . . . 332

shutdown . . . 333
setvcpus . . . 334
start . . . 336
suspend . . . 338
undefine . . . 339
vcpucount . . . 340
vol-create. . . . 341
vol-delete. . . . 342
vol-dumpxml . . . 343
vol-info . . . 344
vol-key . . . 345
vol-list. . . . 346
vol-name . . . 347
vol-path . . . 348
vol-pool . . . 349

Chapter 30. Selected QEMU commands . . . 351

QEMU monitor commands . . . 351
Examples for the use of the qemu-img command . . . 351

Chapter 31. Hypervisor information for the virtual server user . . . 353

Part 8. Appendixes . . . 359

Accessibility. . . . 361

Notices . . . 363

Trademarks . . . 364

Index . . . 365

About this document

This document describes the tasks that are performed by the KVM virtual server administrator to set up, configure, and operate Linux on KVM instances and their virtual devices running on the KVM host on IBM® Z hardware.

For KVM host setup information, see the host administration documentation of your distribution.

For a description of Linux on KVM and tasks that are performed by the KVM virtual server user, see *Device Drivers, Features, and Commands for Linux as a KVM Guest*, SC34-2754.

This document describes a selection of helpful libvirt XML elements and virsh commands that can be used to perform the documented administration tasks for a KVM host on IBM Z hardware. The described subset is not complete.

KVM users familiar with other platforms should be aware that:

- Configuration elements might be used differently on the IBM Z platform.
- Not all available commands, command options or command output are relevant to the IBM Z platform.

You can find the latest version of the complete references on libvirt.org at:

- libvirt.org/format.html
- libvirt.org/sources/virshcmdref

How this document is organized

The first part of this document contains general and overview information for the KVM virtual server management tasks and concepts.

Part two contains chapters that describe how to change the current setup of IBM Z devices on the KVM host in order to provide them as virtual devices for a KVM virtual server.

Part three contains chapters about the configuration of a KVM virtual server and the specification of the IBM Z hardware on which the virtual resources are based.

Part four contains chapters about the lifecycle management and operation of a KVM virtual server.

Part five contains chapters that describe how to display information that helps to diagnose and solve problems associated with the operation of a KVM virtual server.

Part six contains a selection of configuration elements and operation commands that are useful for the described tasks on the IBM Z platform.

Conventions and assumptions used in this publication

This summarizes the styles, highlighting, and assumptions used throughout this publication.

Authority

Most of the tasks described in this document require a user with root authority. Throughout this document, it is assumed that you have root authority.

Persistent configuration

Device and interface setups as described in this document do not persist across host reboots. For information about persistently setting up devices and interfaces, see the administration documentation of your host distribution.

Terminology

This document uses the following terminology:

KVM virtual server, virtual server
Virtualized IBM Z resources that comprise processor, memory, and I/O capabilities as provided and managed by KVM. A virtual server can include an operating system.

KVM guest, guest, guest operating system
An operating system of a virtual server.

KVM host, host
The Linux instance that runs the KVM virtual servers and manages their resources.

Highlighting

This publication uses the following highlighting styles:

- Paths and file names are highlighted in `monospace`.
- Variables are highlighted in *italics.*
- Commands in text are highlighted in **`monospace bold`**.
- Input and output as normally seen on a computer screen is shown

```
within a screen frame.
Prompts on the KVM host are shown as hash signs:
#
Prompts on the KVM virtual server are shown as hash signs preceeded by an indication:
[root@guest:] #
```

Where to get more information

This section provides links to information about KVM virtual server management.

Kernel based virtual machine (KVM)

For general documentation around KVM, see linux-kvm.org/page/Main_Page. The documentation mainly focuses on KVM internals and feature sets. There are also more general documents that describe administration and tuning aspects. Of particular interest is the KVM HowTo page at linux-kvm.org/page/HOWTO.

For information about KVM on x86, see the IBM Knowledge Center at www.ibm.com/support/knowledgecenter/linuxonibm/liaat/liaatkvm.htm.

libvirt virtualization API

libvirt provides the management API on the host.

For internal and external documentation of libvirt, see libvirt.org. Of particular interest are:

- The FAQ section at wiki.libvirt.org/page/FAQ. This section provides a good general introduction to libvirt.
- The XML reference at libvirt.org/format.html. This XML configures a virtual server.
- The virsh command reference at libvirt.org/virshcmdref.html. The virsh commands are used on the host to manage virtual servers.

QEMU

QEMU is the user space process that implements the virtual server hardware on the host.

For QEMU documentation, see wiki.qemu.org.

Other publications

- Open vSwitch: openvswitch.org
- SCSI Architecture Model (SAM): t10.org

Other publications for Linux on z Systems and LinuxONE

You can find publications for Linux on z Systems® and LinuxONE on IBM Knowledge Center and on developerWorks®.

These publications are available on IBM Knowledge Center at www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_lib.html

- *Device Drivers, Features, and Commands* (distribution-specific editions)
- *Using the Dump Tools* (distribution-specific editions)
- *Running Docker Containers on IBM Z*, SC34-2781
- *KVM Virtual Server Quick Start*, SC34-2753
- *KVM Virtual Server Management*, SC34-2752
- *KVM Virtual Server Management Tools*, SC34-2763
- *Device Drivers, Features, and Commands for Linux as a KVM Guest* (distribution-specific editions)
- *Installing SUSE Linux Enterprise Server 12 as a KVM Guest*, SC34-2755
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *libica Programmer's Reference*, SC34-2602
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Linux on z Systems Troubleshooting*, SC34-2612
- *Linux Health Checker User's Guide*, SC34-2609
- *Kernel Messages*, SC34-2599

- *How to use Execute-in-Place Technology with Linux on z/VM®*, SC34-2594
- *How to Improve Performance with PAV*, SC33-8414
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596

You can also find these publications on developerWorks at
www.ibm.com/developerworks/linux/linux390/documentation_dev.html

For versions of documents that have been adapted to a particular distribution, see one of the following web pages:
www.ibm.com/developerworks/linux/linux390/documentation_red_hat.html
www.ibm.com/developerworks/linux/linux390/documentation_suse.html
www.ibm.com/developerworks/linux/linux390/documentation_ubuntu.html

Part 1. General concepts

Chapter 1. Overview 3
Virtual server management tasks 4
Virtualization components 6

Chapter 2. Virtual block devices 9
DASDs and SCSI disks 9
Image files and logical volumes 14
Storage pools 14

Chapter 3. SCSI tapes and medium changers as virtual SCSI devices 17

Chapter 4. Network devices as virtual Ethernet devices 21

As KVM virtual server administrator, you prepare devices for the use of virtual servers, configure virtual servers, and manage the operation of virtual servers.

Chapter 1. Overview

Set up, configure, and manage the operation of virtual servers.

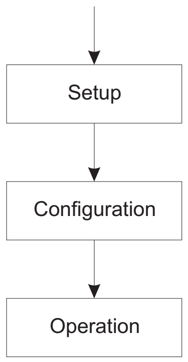


Figure 1. Virtual server administrator's tasks

A *KVM virtual server* consists of virtualized IBM Z resources that comprise processor, memory, and I/O capabilities as provided and managed by KVM. A virtual server can include an operating system. Throughout this book, the term *virtual server* is used for a KVM virtual server. In the libvirt documentation, a virtual server is called a *domain*.

A *KVM guest* or simply *guest* is an operating system of a virtual server. In the QEMU or libvirt documentation, sometimes a virtual server is also referred to as a guest. Do not confuse this term with the preceding definitions.

The *KVM host* is the Linux instance that runs the KVM virtual servers and manages their resources. In the libvirt documentation, a host is also called a *node*.

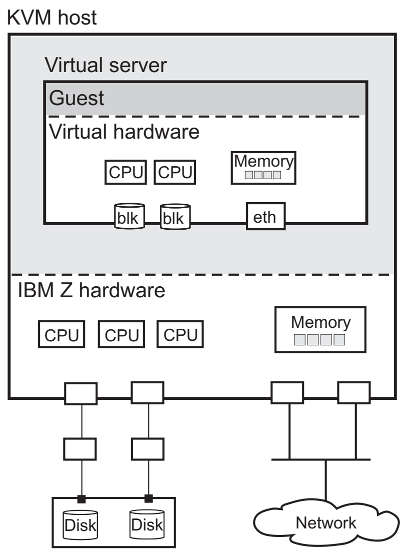


Figure 2. KVM host with a virtual server including a guest operating system

Virtual server management tasks

As a virtual server administrator, you are responsible for the following tasks.

1. Device setup

   Device virtualization hides the specifics of real devices from virtual servers. As a consequence, real devices cannot be configured from virtual servers. You need to prepare the adapter hardware, the physical disk devices, and the network devices to be used by virtual servers.

   For a detailed description of this task, see Part 2, "Device setup," on page 25.

2. Virtual server and device configuration

   You configure a virtual server with a *domain configuration-XML*. The configuration includes the specification of a name, which is used to identify the virtual server, system resources, and devices to be defined with the virtual server.

   You can configure devices that can be attached to an already defined virtual server by using separate *device configuration-XMLs*.

   For a detailed description of this task, see Part 3, "Configuration," on page 47.

3. Virtual server and device operation

   This document describes how to manage the operation of virtual servers by using *virsh commands* based on *configuration-XML files*.

   a. After you have configured a virtual server, you create a persistent virtual server definition:

      Defining the virtual server passes its domain configuration-XML file to *libvirt*. libvirt associates the defined virtual server with the name specified in the domain configuration-XML and with an internal representation of the configuration (see Figure 3).

      This internal representation may differ from the domain configuration-XML with regard to the order of configuration elements, and automatically generated additional configuration elements and values.

      The current libvirt-internal configuration may vary depending on resource operations that you perform on the running virtual server.

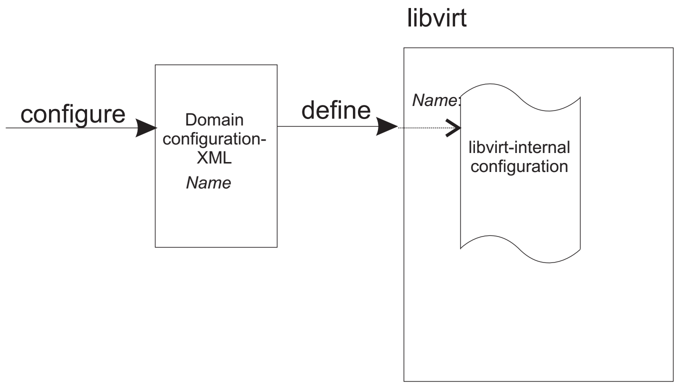


Figure 3. Creating a persistent virtual server definition

   b. Now you can manage the *operation* of the virtual server. This consists of:
      - Life cycle management:

A virtual server is either shut off, running or paused. (There are other states as well, which will be mentioned in a later topic.)

You can issue virsh commands to start, terminate, suspend, or resume a virtual server (see Figure 4).

- Monitoring, which allows you to display:
  - Lists of the defined virtual servers.
  - Specific information about a defined virtual server, such as its state or scheduling information.
  - The current libvirt-internal configuration of a defined virtual server.
- Live migration, which allows you to migrate a defined virtual server to another host.
- System resource management, which allows you to manage the virtual system resources of a virtual server, such as its virtual CPUs.
- Device management, which allows you to attach devices to or detach devices from a defined virtual server. If the virtual server is running, the devices can be hotplugged or unplugged.

c. *Undefining* a virtual server from libvirt results in the deletion of the virtual server name and the libvirt-internal configuration.

For a detailed description of these tasks, see Part 4, "Operation," on page 107.

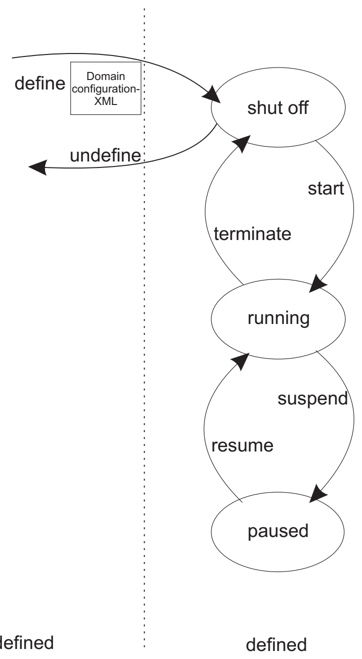


Figure 4. Simplified state-transition diagram of a virtual server

Virtualization components

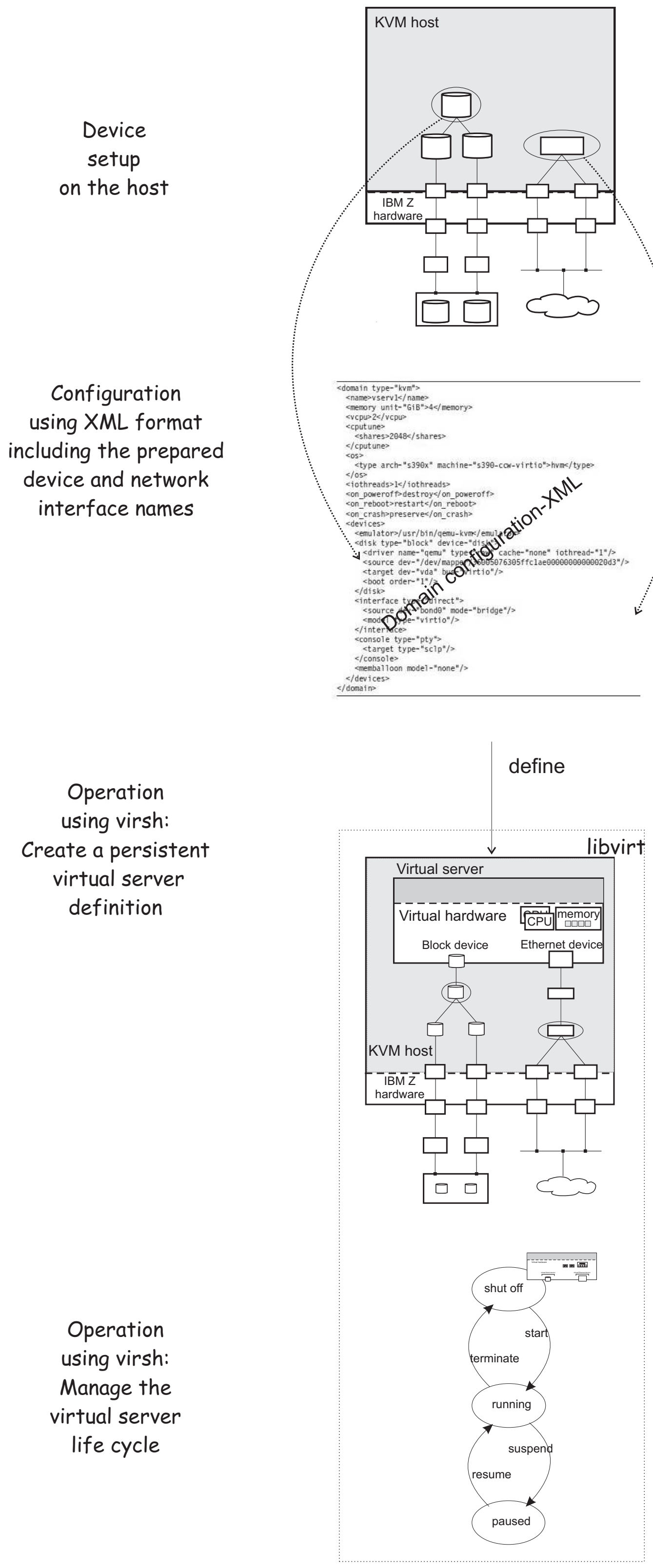
The virtual server management as described in this document is based on the following virtualization components.

Linux kernel including the kvm kernel module (KVM)
Provides the core virtualization infrastructure to run multiple virtual servers on a Linux host.

QEMU
User space component that implements virtual servers on the host using KVM functionality.

libvirt Provides a toolkit for the virtual server management:
- The *XML format* is used to configure virtual servers.
- The *virsh command-line interface* is used to operate virtual servers and devices.

Figure 5 on page 7 shows the virtual server management tasks using the XML format and the virsh command-line interface.



```
<domain type="kvm">
  <name>vserv1</name>
  <memory unit="GiB">4</memory>
  <vcpu>2</vcpu>
  <cputune>
    <shares>2048</shares>
  </cputune>
  <os>
    <type arch="s390x" machine="s390-ccw-virtio">hvm</type>
  </os>
  <iothreads>1</iothreads>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>preserve</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
    </disk>
    <interface type="direct">
      <source dev="bond0" mode="bridge"/>
      <model type="virtio"/>
    </interface>
    <console type="pty">
      <target type="sclp"/>
    </console>
    <memballoon model="none"/>
  </devices>
</domain>
```

Figure 5. Virtual server administrator tasks using XML format and the virsh command-line interface

Chapter 2. Virtual block devices

DASDs, FC-attached SCSI disks, image files and logical volumes are virtualized as virtio block devices.

Related publications

- *Device Drivers, Features, and Commands*, SC33-8411
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413

DASDs and SCSI disks

DASDs and FC-attached SCSI disks are virtualized as virtio block devices.

On the host, you manage various types of disk devices and their configuration topology. For production systems, DASDs and FC-attached SCSI disks are typically set up with multipathing to boost availability through path redundancy.

From the virtual server point of view, these are virtual block devices which are attached by one virtual channel path. There is no difference whether a virtual block device is implemented as a DASD, a SCSI disk, or an image file on the host.

QEMU uses the current libvirt-internal configuration to assign the virtual devices of a virtual server to the underlying host devices.

To provide DASDs and FC-attached SCSI disks as virtual block devices for a virtual server:

1. Set up the DASDs and FC-attached SCSI disks.

   Prepare multipathing, because virtual block devices cannot be multipathed on the virtual server.

   It is also important that you provide unique device nodes that are persistent across host reboots. Unique device nodes ensure that your configuration remains valid after a host reboot. In addition, device nodes that are unique for a disk device on different hosts allow the live migration of a virtual server to a different host, or the migration of a disk to a different storage server or storage controller.

   See Chapter 5, "Preparing DASDs," on page 27 and Chapter 6, "Preparing SCSI disks," on page 29.
2. Configure the DASDs and FC-attached SCSI disks as virtual block devices.

   You configure devices that are to be defined with the virtual server in its domain configuration-XML file. You can also define devices in a separate device configuration-XML file. Such devices can be attached to an already defined virtual server.

   See Chapter 10, "Configuring devices," on page 75 and "Configuring virtual block devices" on page 78.

DASD and SCSI disk configuration topology

Figure 6 on page 10 shows how multipathed DASD and SCSI disks are configured as virtual block devices.

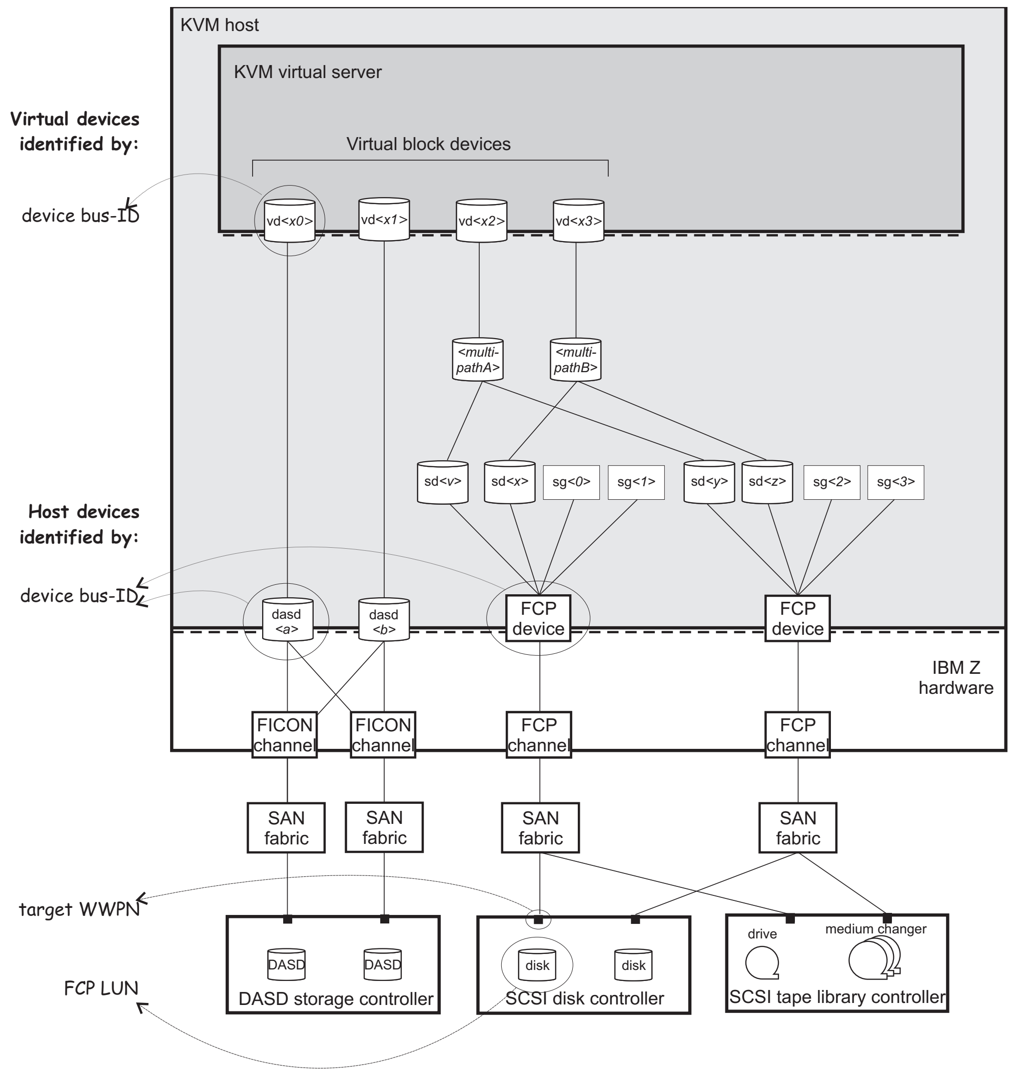


Figure 6. Multipathed DASD and SCSI disks configured as virtual block devices

Disk device identification

There are multiple ways to identify a disk device on the host or on the virtual server.

Device bus-ID and device number of an FCP device

On the host, a SCSI device is connected to an FCP device, which has a device bus-ID of the form:

```
0.m.dddd
```

Where:

`0`	is the channel subsystem-ID.
`m`	is the subchannel set-ID.
`dddd`	is the device number of the FCP device.

Example:

`0.0.1700`	device bus-ID of the FCP device.

`1700`	device number of the FCP device.

Device bus-ID and device number of a DASD

On the host, a DASD is attached to a FICON® channel. It has a device bus-ID of the form:

`0.m.dddd`

Example:

`0.0.e717`	device bus-ID of the DASD.
`e717`	device number of the DASD.

Unique ID (UID) of a DASD

PAV and HyperPAV provide means to create unique IDs to identify DASDs.

Example:

`IBM.75000000010671.5600.00`

Device bus-ID and device number of a virtual block device

On the virtual server, all virtual block devices are accessed through a single virtual channel subsystem. The virtual server directly identifies a virtual block device through its device bus-ID, which is of the form:

`0.m.dddd`

Where:

`0`	is the channel subsystem-ID.
`m`	is the subchannel set-ID.
`dddd`	is the device number of the virtual block device.

Example:

`0.0.1a12`	device bus-ID of the virtual device.
`1a12`	device number of the virtual device.

Standard device name

Standard device names are of the form:

`dasd<`*x*`>`	for DASDs on the host.
`sd<`*x*`>`	for SCSI disks on the host.
`vd<`*x*`>`	for virtual block devices on the virtual server.

Where <*x*> can be one or more letters.

They are assigned in the order in which the devices are detected and thus can change across reboots.

Example:

`dasda`	on the host.
`sda`	on the host.
`vda`	on the virtual server.

If there is only one attached SCSI disk, you can be sure that host device `sda` is mapped to virtual server device `vda`.

Standard device node

User space programs access devices through device nodes. Standard device nodes are of the form:

`/dev/<standard-device-name>`

Example:

`/dev/sda`	for SCSI disks on the host.
`/dev/dasda`	for DASDs on the host.
`/dev/vda`	for virtual block devices on the virtual server.

udev-created device node

If udev is available with your product or distribution, it creates device nodes which are based on unique properties of a device and so identify a particular device. udev creates various device nodes for a device which are based on the following information:

- Hardware / storage server (by-uid device node)
- Device bus-ID (by-path device node)
- SCSI identifier for SCSI disks or disk label (VOLSER) for DASDs (by-ID device node)
- File system information (by-uuid device node)

Example for DASDs on the host:

`/dev/disk/by-path/ccw-0.0.1607`
`/dev/disk/by-path/ccw-0.0.1607-part1`

where:

`0.0.1607`	is the device bus-ID of the DASD.
`part1`	denotes the first partition of the DASD.

`/dev/disk/by-id/ccw-IBM.750000000R0021.1600.07`
`/dev/disk/by-id/ccw-IBM.750000000R0021.1600.07-part1`

where:

`IBM.750000000R0021.1600.07`	is the UID of the DASD.
`part1`	denotes the first partition of the DASD.

`/dev/disk/by-uuid/a6563ff0-9a0f-4ed3-b382-c56ad4653637`

where:

`a6563ff0-9a0f-4ed3-b382-c56ad4653637`	is the universally unique identifier (UUID) of a file system.

Example for SCSI devices on the host:

`/dev/disk/by-path/ccw-0.0.3c40-zfcp-0x500507630300c562:0x401040ea00000000`

where:

`0.0.3c40`	is the device bus-ID of the FCP device.
`0x500507630300c562`	is the worldwide port name (WWPN) of the storage controller port.
`0x401040ea00000000`	is the FCP LUN.

`/dev/disk/by-id/scsi-36005076303ffc56200000000000010ea`

where:

`scsi-36005076303ffc56200000000000010ea`	is the SCSI identifier.

`/dev/disk/by-uuid/7eaf9c95-55ac-4e5e-8f18-065b313e63ca` where:

`7eaf9c95-55ac-4e5e-8f18-065b313e63ca`	is the universally unique identifier (UUID) of a file system.

Since device-specific information is hidden from the virtual server, udev creates by-path device nodes on the virtual server. They are derived from the device number of the virtual block device, which you can specify in the domain configuration-XML or in the device configuration-XML.

The udev rules to derive by-path device nodes depend on your product or distribution.

Tip: Prepare a strategy for specifying device numbers for the virtio block devices, which you provide for virtual servers. This strategy makes it easy to identify the virtualized disk from the device bus-ID or device number of the virtual block device.

Virtual server example:

```
/dev/disk/by-path/ccw-0.0.1a12
/dev/disk/by-path/ccw-0.0.1a12-part1
```

where:

`0.0.1a12`	is the device bus-ID.
`part1`	denotes the first partition of the device.

Device mapper-created device node

The *multipath device mapper support* assigns a unique device mapper-created device node to a SCSI disk. The device mapper-created device node can be used on different hosts to access the same SCSI disk.

Example:

```
/dev/mapper/36005076305ffc1ae00000000000021d5
/dev/mapper/36005076305ffc1ae00000000000021d5p1
```

where

`p1`	denotes the first partition of the device.

Tip: Use device mapper-created device nodes for SCSI disks and udev-created device nodes for DASDs in your configuration-XML files to support a smooth live migration of virtual servers to a different host.

Image files and logical volumes

Image files and logical volumes are virtualized as virtio block devices.

To provide image files as virtual block devices for a virtual server:

1. Create and initialize the image files
2. Make the image files accessible for the virtual server.
3. Configure the image files as virtual block devices.

   You configure devices that are to be defined with the virtual server in its domain configuration-XML file. You can also define devices in a separate device configuration-XML file. Such devices can be attached to an already defined virtual server.

   See Chapter 10, "Configuring devices," on page 75 and "Configuring an image file as storage device" on page 84.

Storage pools

Alternatively, you can configure *storage pools*, leaving the resource management of step 1 to libvirt. A storage pool consists of a set of *volumes*, such as

- The image files of a host directory
- The image files residing on a disk or the partition of a disk
- The image files residing on a network file system
- The logical volumes of a volume group

A live virtual server migration is only possible for storage pools backed by image files residing on a network file system.

Figure 7 shows a storage pool backed by the image files of a directory:

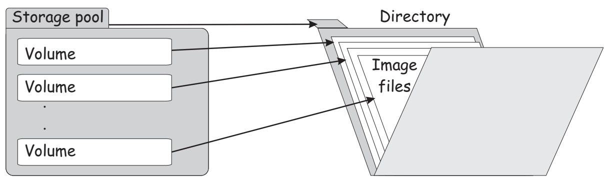


Figure 7. Storage pool backed by the image files of a directory

Figure 8 shows a storage pool backed by the logical volumes of a volume group:

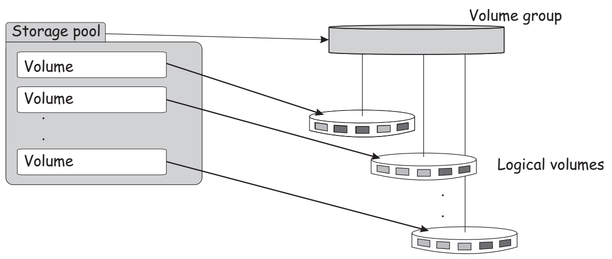


Figure 8. Storage pool backed by the logical volumes of a volume group

To provide the volumes of a storage pool as virtual block devices for a virtual server:

1. Create the resources which back the storage pool.

2. Make resources backing the volumes accessible for the virtual server.
3. Configure the storage pool including its volumes.

   See Chapter 11, “Configuring storage pools,” on page 103.
4. Configure volumes as virtual storage devices for the virtual server.

   See “Configuring a volume as storage device” on page 86.

   Figure 9 shows a storage pool backed by a host directory. The volumes of the storage pool are configured as virtual block devices of different virtual servers:

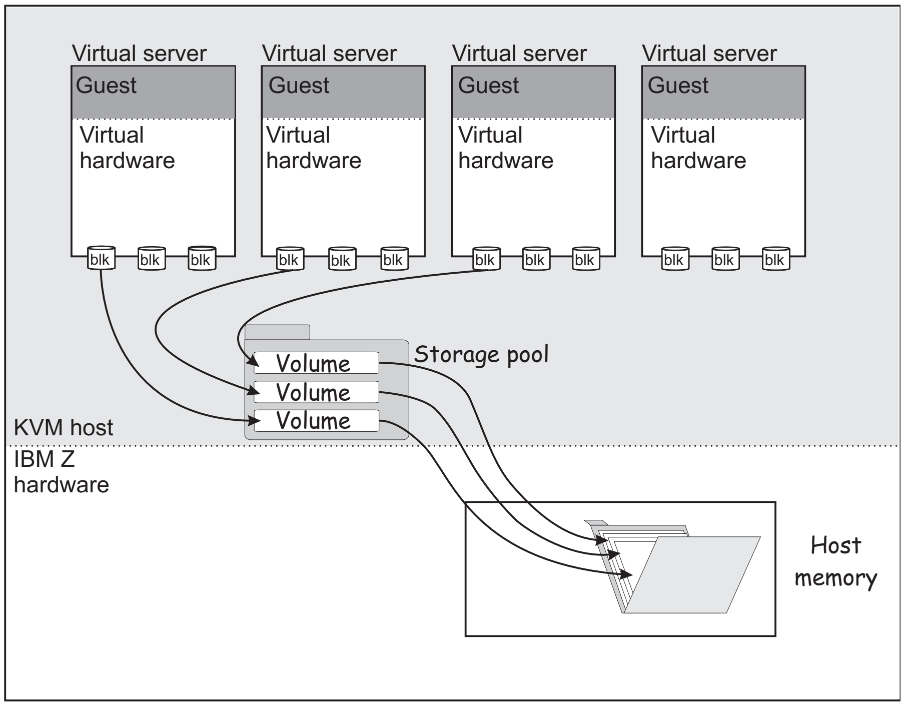


Figure 9. Storage pool volumes configured as virtual block devices

5. Define and start the storage pool before defining the virtual server.

   Manage the storage pool and its volumes by using the commands described in Chapter 19, “Managing storage pools,” on page 151.

Chapter 3. SCSI tapes and medium changers as virtual SCSI devices

FC-attached SCSI tape and medium changer devices are virtualized as virtio SCSI devices.

To provide high reliability, be sure to set up redundant paths for SCSI tape or medium changer devices on the host. A device configuration for a SCSI tape or medium changer device provides one virtual SCSI device for each path. Figure 10 on page 18 shows one virtual SCSI device for sg<0>, and one for sg<1>, although these devices represent different paths to the same device. The lin_tape device driver models path redundancy on the virtual server. lin_tape reunites the virtual SCSI devices that represent different paths to the same SCSI tape or medium changer device.

To provide a SCSI tape or medium changer device for a virtual server:

1. Set up the SCSI tape or medium changer device.

   See Chapter 7, "Preparing SCSI tape and medium changer devices," on page 33.
2. Configure the SCSI tape or medium changer device in separate device configuration-XML files.

   You need to check this configuration after a host reboot, a live migration, or when an FCP device or a SCSI tape or medium changer device in the configuration path is set offline and back online.

   See Chapter 10, "Configuring devices," on page 75 and "Configuring virtual SCSI devices" on page 88.

Virtual SCSI device configuration topology

Figure 10 on page 18 shows one SCSI tape and one SCSI medium changer, which are accessible via two different configuration paths. They are configured as virtual SCSI devices on a virtual server.

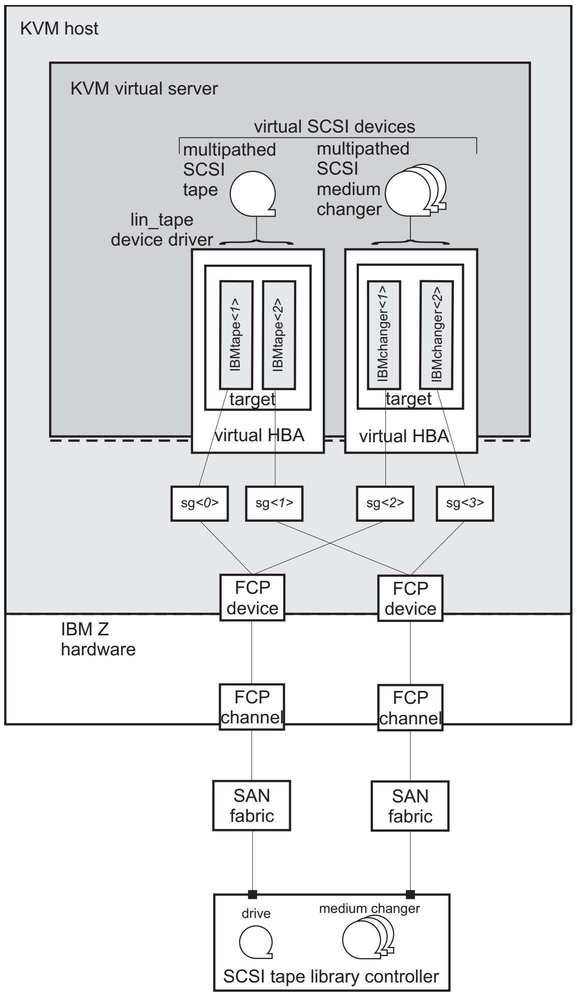


Figure 10. Multipathed SCSI tapes and SCSI medium changer devices configured as virtual SCSI devices

Each generic SCSI host device is configured as a virtual SCSI device.

SCSI device identification

For a SCSI tape or medium changer device configuration, the following device names are relevant:

Standard device name

Standard device names are of the form:

`sg<x>`	for SCSI tape or medium changer devices on the host using the SCSI generic device driver.
`IBMtape<x>`	for SCSI tape devices on the virtual server using the lin_tape device driver.
`IBMchanger<x>`	for SCSI medium changer devices on the virtual server using the lin_tape device driver.

Where <*x*> can be one or more digits.

They are assigned in the order in which the devices are detected and thus can change across reboots.

SCSI device name

SCSI device names are of the form:

```
<SCSI-host-number>:0:<SCSI-ID>:<SCSI-LUN>
```

Where:

<SCSI-host-number>	is assigned to the FCP device in the order in which the FCP device is detected.
<SCSI-ID>	is the SCSI ID of the target port.
<SCSI-LUN>	is assigned to the SCSI device by conversion from the corresponding FCP LUN.

SCSI device names are freshly assigned when the host reboots, or when an FCP device or a SCSI tape or medium changer device is set offline and back online.

SCSI device names are also referred to as *SCSI stack addresses*.

Example:	`0:0:1:7`

Related publication

- *Device Drivers, Features, and Commands*, SC33-8411

Chapter 4. Network devices as virtual Ethernet devices

Virtualize network devices as virtual Ethernet devices by configuring direct MacVTap connections or virtual switches.

In a typical virtual network device configuration, you will want to isolate the virtual server communication paths from the communication paths of the host. There are two ways to provide network isolation:

- You set up separate network devices for the virtual servers that are not used for the host network traffic. This method is called *full isolation*. It allows the virtual network device configuration using a direct MacVTap connection or a virtual switch.
- If the virtual server network traffic shares network interfaces with the host, you can provide isolation by configuring the virtual network device using a MacVTap interface. Direct MacVTap connections guarantee the isolation of virtual server and host communication paths.

Whatever configuration you choose, be sure to provide high reliability through path redundancy as shown in Figure 11:

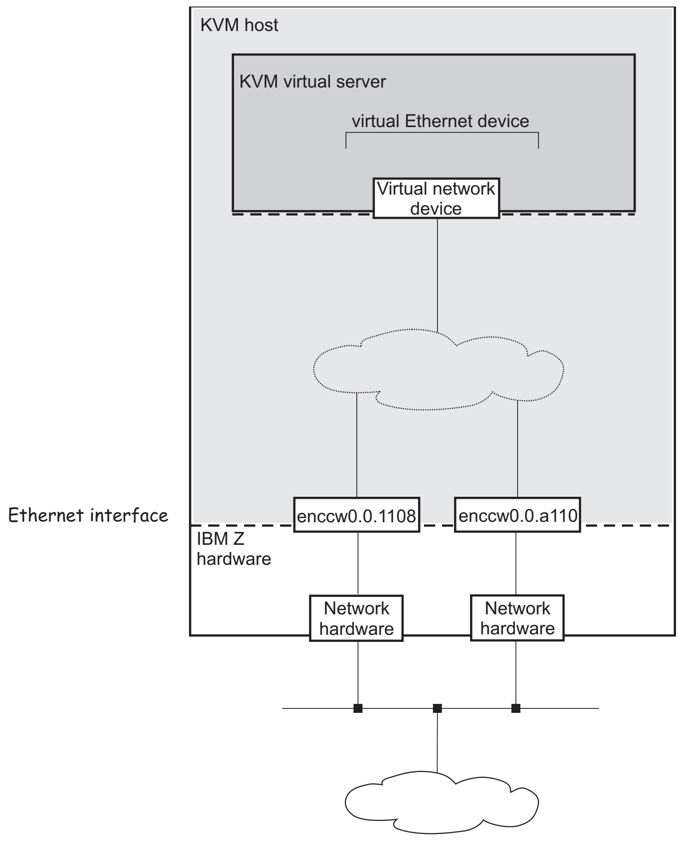


Figure 11. Highly reliable virtual network device configuration

Network device configuration using a direct MacVTap connection

MacVTap provides a high speed network interface to the virtual server. The MacVTap network device driver virtualizes Ethernet devices and provides MAC addresses for virtual network devices.

If you decide to configure a MacVTap interface, be sure to set up a bonded interface which aggregates multiple network interfaces into a single entity, balancing traffic and providing failover capabilities. In addition, you can set up a virtual LAN interface, which provides an isolated communication between the virtual servers that are connected to it.

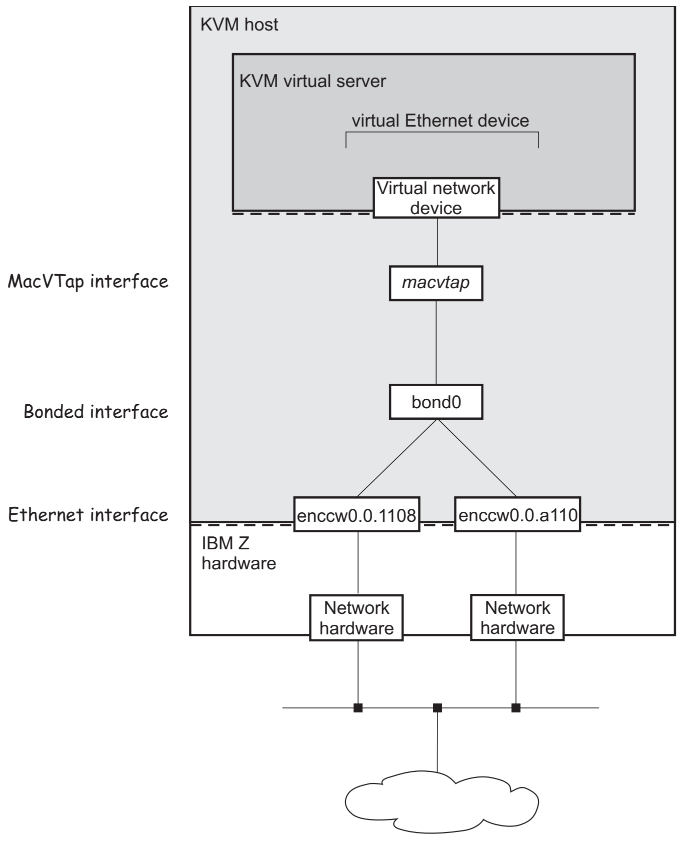


Figure 12. Configuration using a direct MacVTap connection

When you configure a virtual Ethernet device, you associate it with a network interface name on the host in the configuration-XML. In Figure 12, this is `bond0`. libvirt then creates a MacVTap interface from your network configuration.

Use persistent network interface names to ensure that the configuration-XMLs are still valid after a host reboot or after you unplug or plug in a network adapter. Your product or distribution might provide a way to assign meaningful names to your network interfaces. When you intend to migrate a virtual server, use network interface names that are valid for the hosts that are part of the migration.

Network device configuration using virtual switches

Virtual switches are implemented using Open vSwitch. Virtual switches can be used to virtualize Ethernet devices. They provide means to configure path redundancy, and isolated communication between selected virtual servers.

With virtual switches, the configuration outlined in Figure 11 on page 21 can be realized as follows:

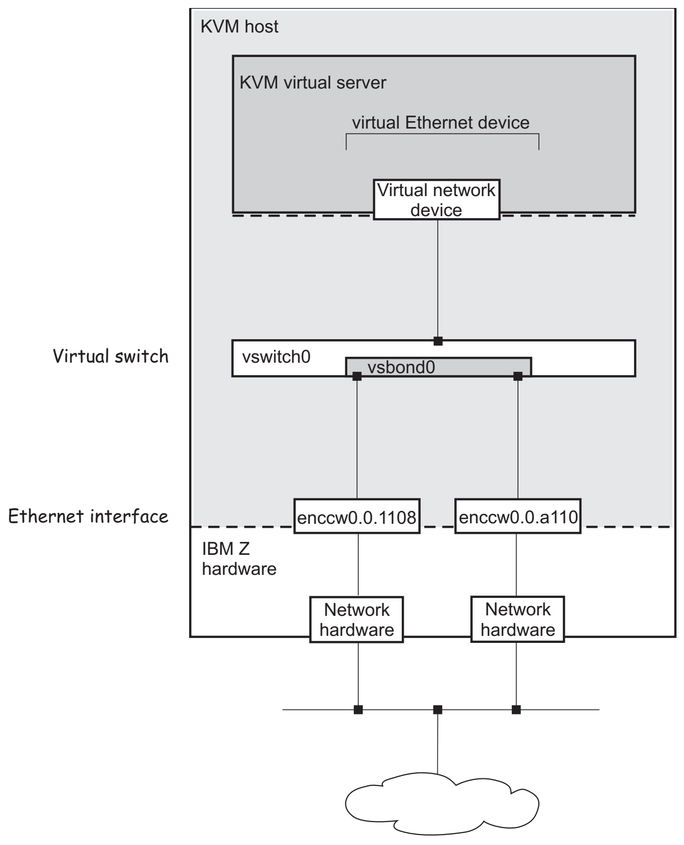


Figure 13. Configuration using a virtual switch

Note: Libvirt also provides a default bridged network, called virbr0, which is not covered in this document. See the libvirt networking documentation reference in the related publications section for more details.

Related publications

- *Device Drivers, Features, and Commands*, SC33-8411
- Libvirt networking documentation at wiki.libvirt.org/page/Networking

Related tasks:

Chapter 8, “Preparing network devices,” on page 37
Consider these aspects when setting up network interfaces for the use of virtual servers.

“Configuring virtual Ethernet devices” on page 98
Configure network interfaces, such as Ethernet interfaces, bonded interfaces, virtual LANs, or virtual switches as virtual Ethernet devices for a virtual server.

Part 2. Device setup

Chapter 5. Preparing DASDs 27

Chapter 6. Preparing SCSI disks 29

Chapter 7. Preparing SCSI tape and medium changer devices 33

Chapter 8. Preparing network devices. 37
Creating a network interface. 38
Preparing a network interface for a direct MacVTap connection. 40
Preparing a bonded interface 40
Preparing a virtual switch 43

Prepare devices on the host for the use of a virtual server.

Chapter 5. Preparing DASDs

Consider these aspects when setting up ECKD™ DASDs for the use of a virtual server.

Before you begin

- You need to know the device number of the base device as defined on the storage system and configured in the IOCDS.
- If you intend to identify the DASD using the device bus-ID (by-path device node) and you intend to migrate the virtual server accessing the DASD, make sure that you use the same IOCDS configuration for the DASD on both the source and the destination host.
- Make sure that the DASD is accessible, for example by entering the following command:

```
# lsdasd -a
Bus-ID Status Name Device Type BlkSz Size Blocks
==============================================================================
0.0.7500 offline
```

- If the PAV or the HyperPAV feature is enabled on your storage system, it assigns unique IDs to its DASDs and manages the alias devices.

About this task

The following publication describes how to configure, prepare, and work with DASDs:
- *Device Drivers, Features, and Commands*, SC33-8411

Procedure

The following steps describe a DASD setup on the host that does not persist across host reboots.
For a persistent setup, see your host administration documentation (see also "Persistent configuration" on page x).

1. Set the DASD base device and its alias devices online.
2. Obtain the device node of the DASD.
3. You need to format the DASD, because the virtual server cannot format DASDs by itself.

   You can use CDL, and LDL formats.
4. Do not create partitions on behalf of the virtual server.

   Establish a process to let the virtual server user know which virtual block devices are backed up by DASDs, because these devices have to be partitioned using the Linux command **fdasd** for CDL formats. The inadvertent use of the **fdisk** command to partition the device could lead to data corruption.

Example

1. Set the DASD online using the Linux command **chccwdev** and the device bus-ID of the DASD.

   For example, for device `0.0.7500`, issue:

```
# chccwdev –e 0.0.7500
```

2. To obtain the DASD name from the device bus-ID, you can use the Linux command **`lsdasd`**:

```
# lsdasd
Bus-ID     Status      Name      Device  Type  BlkSz  Size      Blocks
==============================================================================
0.0.7500   active      dasde     94:0    ECKD  4096   7043MB    1803060
...
```

   The udev-created by-path device node for device `0.0.7500` is `/dev/disk/by-path/ccw-0.0.7500`. You can verify this name by issuing:

```
# ls /dev/disk/by-path -l
total 0
lrwxrwxrwx 1 root root 11 Mar 11  2014 ccw-0.0.7500 -> ../../dasde
```

3. Format the DASD using the Linux command **`dasdfmt`** and the device name.

```
# dasdfmt -b 4096 /dev/disk/by-path/ccw-0.0.7500 -p
```

4. Establish a procedure to let the virtual server user know which virtual devices are backed up by DASDs.

What to do next

Configure the DASDs as described in "Configuring a DASD or SCSI disk" on page 78.

Related concepts:

Chapter 2, "Virtual block devices," on page 9
DASDs, FC-attached SCSI disks, image files and logical volumes are virtualized as virtio block devices.

Chapter 6. Preparing SCSI disks

Consider these aspects when setting up FC-attached SCSI disks for the use of a virtual server.

Before you begin

1. If you want to allow a migration of a virtual server to another host, use unique names for the virtualized SCSI disks, which can be used from different hosts.

   Device-mapper multipathing groups two or more paths to the same SCSI disk, thus providing failover redundancy and load balancing. It assigns unique device mapper-created device nodes to SCSI disks, which are valid for all hosts that access the SCSI disks.

   According to your product or distribution mechanism:

   a. Make sure that multipath support is enabled.
   b. Configure the multipath device mapper not to use user-friendly names. User friendly names are symbolic names, which are not necessarily equal on different hosts.

   See your host administration documentation to find out how to prepare multipath support.
2. Provide either of the following information:
   - The device bus-IDs of the FCP devices, target WWPNs, and the FCP LUNs of the SCSI disk.
   - The device mapper-created device node of the SCSI disk.

About this task

The following publications describe in detail how to configure, prepare, and work with FC-attached SCSI disks:

- *Fibre Channel Protocol for Linux and z/VM on IBM System z*®, SG24-7266
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *Device Drivers, Features, and Commands*, SC33-8411

Procedure

The following steps describe a SCSI disk setup on the host that does not persist across host reboots.
For a persistent setup, see your host administration documentation (see also "Persistent configuration" on page x).

1. Linux senses the available FCP devices.

   You can use the **lscss** command to display the available FCP devices.

   The **-t** option can be used to restrict the output to a particular device type. FCP devices are listed as 1732/03 devices with control unit type 1731/03.
2. Set the FCP device online.

   You can use the **chccwdev** command to set an FCP device online or offline.
3. Configure the SCSI disks on the host.

   For details about this step, refer to your host administration documentation and *Device Drivers, Features, and Commands*, SC33-8411.

If your FCP setup uses N_Port ID virtualization (NPIV), the SCSI LUNs are automatically detected. If you do not use NPIV or if automatic LUN scanning is disabled, write the LUN to the sysfs **unit_add** attribute of the applicable target port:

```
# echo <fcp_lun> > /sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/unit_add
```

4. Verify the configuration and display the multipath device mapper-created device node of the SCSI disk.
5. Do not partition SCSI disks for a virtual server, because the virtual server user might want to partition its virtual block devices.

Example

For one example path, you provide the device bus-ID of the FCP device, the target WWPN, and the FCP LUN of the SCSI disk:

`/sys/bus/ccw/drivers/zfcp/0.0.1700/0x500507630513c1ae/0x402340bc00000000` provides the information:

Device bus-ID of the FCP device	`0.0.1700`
WWPN	`0x500507630513c1ae`
FCP LUN	`0x402340bc00000000`

1. Display the available FCP devices.

```
# lscss -t 1732/03 | fgrep '1731/03'
0.0.1700 0.0.06d4  1732/03 1731/03      80  80  ff   50000000 00000000
0.0.1740 0.0.0714  1732/03 1731/03      80  80  ff   51000000 00000000
0.0.1780 0.0.0754  1732/03 1731/03 yes  80  80  ff   52000000 00000000
0.0.17c0 0.0.0794  1732/03 1731/03 yes  80  80  ff   53000000 00000000
0.0.1940 0.0.08d5  1732/03 1731/03      80  80  ff   5c000000 00000000
0.0.1980 0.0.0913  1732/03 1731/03      80  80  ff   5d000000 00000000
```

2. Set the FCP device online.

```
# chccwdev -e 0.0.1700
Setting device 0.0.1700 online
Done
```

3. Configure the SCSI disk on the host.

```
# echo 0x402340bc00000000 > /sys/bus/ccw/drivers/zfcp/0.0.1700/0x500507630513c1ae/unit_add
```

4. Figure out the device mapper-created device node of the SCSI disk.
   a. You can use the **lszfcp** command to display the SCSI device name of a SCSI disk:

   ```
   # lszfcp -D -b 0.0.1700 -p 0x500507630513c1ae -l 0x402340bc00000000
   0.0.1700/0x500507630513c1ae/0x402340bc00000000 2:0:17:1086079011
   ```

   b. The **lsscsi -i** command displays the multipathed SCSI disk related to the SCSI device name:

```
# lsscsi -i
...
[1:0:16:1086144547]disk   IBM   2107900   .166  /dev/sdg   36005076305ffc1ae00000000000023bd
[1:0:16:1086210083]disk   IBM   2107900   .166  /dev/sdk   36005076305ffc1ae00000000000023be
[1:0:16:1086275619]disk   IBM   2107900   .166  /dev/sdo   36005076305ffc1ae00000000000023bf
[2:0:17:1086079011]disk   IBM   2107900   2440  /dev/sdq   36005076305ffc1ae00000000000023bc
...
```

The device mapper-created device node that you can use to uniquely reference the multipathed SCSI disk `36005076305ffc1ae00000000000023bc` is:

`/dev/mapper/36005076305ffc1ae00000000000023bc`

What to do next

Configure the SCSI disks as described in "Configuring a DASD or SCSI disk" on page 78.

Related concepts:

Chapter 2, "Virtual block devices," on page 9
DASDs, FC-attached SCSI disks, image files and logical volumes are virtualized as virtio block devices.

Chapter 7. Preparing SCSI tape and medium changer devices

Consider these aspects when setting up FC-attached SCSI tapes and SCSI medium changers for the use of a virtual server.

Before you begin

Provide the device bus-IDs of the FCP devices, the target WWPNs, and the FCP LUNs of the SCSI tape or medium changer devices.

You can use the information that is provided as directory names:

```
/sys/bus/ccw/drivers/zfcp/<device_bus_id>/<wwpn>/<fcp_lun>
```

The virtual server user can install and use the IBM *lin_tape* package on the virtual server for actions such as the mounting and unmounting of tape cartridges into the affected tape drive. The use of the lin_tape device driver is documented in the *IBM Tape Device Drivers Installation and User's Guide*, GC27-2130.

About this task

The following publications describe in detail how to configure, prepare, and work with FC-attached SCSI devices:

- *Fibre Channel Protocol for Linux and z/VM on IBM System z*, SG24-7266
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *Device Drivers, Features, and Commands*, SC33-8411

Note: In the libvirt documentation, the term "LUN" is often referenced as "unit".

Procedure

The following steps describe a SCSI tape or medium changer setup on the host that does not persist across host reboots.
For a persistent setup, see your host administration documentation (see also "Persistent configuration" on page x).

1. Linux senses the available FCP devices.

   You can use the **lscss** command to display the available FCP devices. The **-t** option can be used to restrict the output to a particular device type. FCP devices are listed as 1732/03 devices with control unit type 1731/03.
2. Set the FCP device to which your SCSI device is attached online.

   You can use the **chccwdev** command to set an FCP device online or offline.
3. Register the SCSI tape or medium changer device on the host.

   For details about this step, refer to your host administration documentation and *Device Drivers, Features, and Commands*, SC33-8411.

   If your LUN is not automatically detected, you might add the LUN of the SCSI tape or medium changer device to the file system by issuing:

   ```
   # echo <fcp_lun> > /sys/bus/ccw/devices/<device_bus_id>/<wwpn>/unit_add
   ```

   This step registers the SCSI tape or medium changer device in the Linux SCSI stack and creates a sysfs entry for it in the SCSI branch.

4. Obtain the following information to be able to configure the SCSI tape or medium changer device:
   - The SCSI host number that corresponds to the FCP device
   - The SCSI ID of the target port
   - The SCSI LUN

   You obtain this information by issuing:

   ```
   # lszfcp -D -b <device_bus_ID> -p <wwpn> -l <fcp_lun>
   ```

   This command displays the SCSI device name of the SCSI tape or the SCSI medium changer:

   `<scsi_host_number>:0:<scsi_ID>:<scsi_lun>`

Example

For one example path, you provide the device bus-ID of the FCP device, the target WWPN, and the FCP LUN of the SCSI tape or medium changer device:

`/sys/bus/ccw/drivers/zfcp/0.0.1cc8/0x5005076044840242/0x0000000000000000` provides the information:

Device bus-ID of the FCP device	`0.0.1cc8`
WWPN	`0x5005076044840242`
FCP LUN	`0x0000000000000000`

1. Display the available FCP devices:

   ```
   # lscss -t 1732/03 | fgrep '1731/03'
   0.0.1cc8 0.0.0013  1732/03 1731/03      80  80  ff  f0000000 00000000
   0.0.1f08 0.0.0015  1732/03 1731/03 yes  80  80  ff  1e000000 00000000
   0.0.3b58 0.0.0016  1732/03 1731/03      80  80  ff  68000000 00000000
   ```

2. Bring the FCP device online:

   ```
   # chccwdev -e 0.0.1cc8
   Setting device 0.0.1cc8 online
   Done
   ```

3. Register the SCSI tape device on the host:

   ```
   # echo 0x0000000000000000 > /sys/bus/ccw/devices/0.0.1cc8/0x5005076044840242/unit_add
   ```

4. Obtain the SCSI host number, the SCSI ID, and the SCSI LUN of the registered SCSI tape device:

   ```
   # lszfcp -D -b 0.0.1cc8 -p 0x5005076044840242 -l 0x0000000000000000
    0.0.1cc8/0x5005076044840242/0x0000000000000000 1:0:2:0
   ```

   where:

SCSI host number	1
SCSI channel	0 (*always*)
SCSI ID	2
SCSI LUN	0

What to do next

Configure the SCSI tape and medium changer devices as described in "Configuring a SCSI tape or medium changer device" on page 90.

Related concepts:

Chapter 3, "SCSI tapes and medium changers as virtual SCSI devices," on page 17
FC-attached SCSI tape and medium changer devices are virtualized as virtio SCSI devices.

Chapter 8. Preparing network devices

Consider these aspects when setting up network interfaces for the use of virtual servers.

About this task

Set up the network carefully and be aware that any performance lost in the host setup usually cannot be recovered in the virtual server.

For information about how to set up network devices on the host, see *Device Drivers, Features, and Commands*, SC33-8411.

For performance relevant information about setting up a network in Linux on z Systems, see www.ibm.com/developerworks/linux/linux390/perf/tuning_networking.shtml.

Procedure

1. Create network interfaces as described in "Creating a network interface" on page 38.
2. Prepare the configuration-specific setup.
   a. To configure a MacVTap interface, perform the steps described in "Preparing a network interface for a direct MacVTap connection" on page 40.
   b. To configure a virtual switch, perform the steps described in "Preparing a virtual switch" on page 43.

      Virtual switches provide means to configure highly available or isolated connections. Nevertheless, you may set up a bonded interface or a virtual LAN interface.

What to do next

Configure the network interfaces as described in "Configuring virtual Ethernet devices" on page 98.

Related concepts:

Chapter 4, "Network devices as virtual Ethernet devices," on page 21
Virtualize network devices as virtual Ethernet devices by configuring direct MacVTap connections or virtual switches.

Creating a network interface

Create a network interface for a network device.

Before you begin

You need to know the IP address of the network device and its network interface name.

To find the interface name of a qeth device, issue:

```
# lsqeth -p
```

About this task

The following steps describe a network interface setup on the host that does not persist across host reboots.

For a persistent setup, see your host administration documentation (see also "Persistent configuration" on page x).

Procedure

1. Determine the available network devices as defined in the IOCDS.

   You can use the **znetconf -u** command to list the unconfigured network devices and to determine their device bus-IDs.

   ```
   # znetconf -u
   ```

2. Configure the network devices in layer 2 mode and set them online.

   To provide a good network performance, set the buffer count value to 128.

   For a non-persistent configuration, use the **znetconf -a** command with the **layer2** sysfs attribute set to 1 and the **buffer_count** attribute set to 128:

   ```
   # znetconf -a <device-bus-ID> -o layer2=1 -o buffer_count=128
   ```

   You can use the **znetconf -c** command to list the configured network interfaces and to display their interface names:

   ```
   # znetconf -c
   ```

3. Activate the network interfaces.

   For example, you can use the **ip** command to activate a network interface. Using this command can also verify your results.

   ```
   # ip addr add <IP-address> dev <network-interface-name>
   # ip link set <network-interface-name> up
   ```

   Issue the first command only if the interface has not already been activated and subsequently deactivated.

4. To exploit best performance, increase the transmit queue length of the network device (txqueuelen) to the recommended value of 2500.

   ```
   ip link set <network-interface-name> qlen 2500
   ```

Example

In the following example, you determine that OSA-Express CCW group devices with, for example, device bus-IDs 0.0.8050, 0.0.8051, and 0.0.8052 are to be used, and you set up the network interface.

1. Determine the available network devices.

```
# znetconf -u
Scanning for network devices...
Device IDs                 Type    Card Type      CHPID Drv.
------------------------------------------------------------
...
0.0.8050,0.0.8051,0.0.8052 1731/01 OSA (QDIO)        90 qeth
...
```

2. Configure the network devices and set them online.

```
# znetconf -a 0.0.8050 -o layer2=1 -o buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.8050 (enccw0.0.8050)

# znetconf -c
Device IDs                 Type    Card Type     CHPID Drv. Name            State
-----------------------------------------------------------------------------------
...
0.0.8050,0.0.8051,0.0.8052 1731/01 OSD_1000         A0 qeth enccw0.0.8050   online
...
```

3. Activate the network interfaces.

```
# ip link show enccw0.0.8050
32: enccw0.0.8050: <BROADCAST,MULTICAST> mtu 1492 qdisc pfifo_fast state DOWN qlen 1000
    link/ether 02:00:00:6c:db:72 brd ff:ff:ff:ff:ff:ff

# ip link set enccw0.0.8050 up
```

4. Increase the transmit queue length.

```
# ip link set enccw0.0.8050 qlen 2500
# ip link show enccw0.0.8050
32: enccw0.0.8050: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1492 qdisc pfifo_fast state UNKNOWN
    qlen 2500
    link/ether 02:00:00:6c:db:72 brd ff:ff:ff:ff:ff:ff
```

What to do next

Prepare the configuration-specific setup as described in:

- "Preparing a network interface for a direct MacVTap connection" on page 40
- or "Preparing a virtual switch" on page 43

Preparing a network interface for a direct MacVTap connection

Prepare a network interface for a configuration as direct MacVTap connection.

Before you begin

libvirt will automatically create a MacVTap interface when you configure a direct connection.

Make sure that the MacVTap kernel modules are loaded, for example by using the **lsmod | grep macvtap** command.

Procedure

1. Create a bonded interface to provide high availability.

   See "Preparing a bonded interface."
2. Optional: Create a virtual LAN (VLAN) interface.

   VLAN interfaces provide an isolated communication between the virtual servers that are connected to it.

   Use the **ip link add** command to create a VLAN on a network interface and to specify a VLAN ID:

   ```
   # ip link add link <base-network-if-name> name <vlan-network-if-name>
    type vlan id <VLAN-ID>
   ```

   Example:

   Create a virtual LAN interface with VLAN ID 623.

   ```
   # ip link add link bond0 name bond0.623 type vlan id 623
   # ip link show bond0.623
   17: bond0.623@bond0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
   qdisc noqueue state UP mode DEFAULT group default
   link/ether 02:00:00:f7:a7:c2 brd ff:ff:ff:ff:ff:ff
   ```

Preparing a bonded interface

A bonded network interface allows multiple physical interfaces to be aggregated into a single link, balancing traffic and providing failover capabilities based on the selected mode, such as round-robin or active-backup.

Before you begin

Ensure that the channel bonding module is loaded, for example using the following commands:

```
# modprobe bonding
# lsmod | grep bonding
bonding              156908  0
```

About this task

The following steps describe a bonded interface setup on the host that does not persist across host reboots.

For a persistent setup, see your host administration documentation (see also "Persistent configuration" on page x).

Procedure

1. Define the bonded interface.

   If you configure the bonded interface in a configuration-XML that is intended for a migration, choose an interface name policy which you also provide on the destination host.
2. Set the bonding parameters for the desired bonding mode.

   Dedicate OSA devices planned for 802.3ad mode to a target LPAR. For more information, see *Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935-17.
3. Configure slave devices.
4. Activate the interface.

Example

This example shows how to set up bonded interface bond1. In your distribution, bond0 might be automatically created and registered. In this case, omit step 1 to make use of bond0.

1. Add a new master bonded interface:

```
# echo "+bond1" > /sys/class/net/bonding_masters
# ip link show bond1
8: bond1: <BROADCAST,MULTICAST,MASTER> mtu 1500 qdisc noop state DOWN mode DEFAULT
    link/ether 9a:80:45:ba:50:90 brd ff:ff:ff:ff:ff:ff
```

2. Set the bonding parameters for the desired bonding mode. To set the mode to `active-backup`:

```
# echo "active-backup 1" > /sys/class/net/bond1/bonding/mode
# echo "100" > /sys/class/net/bond1/bonding/miimon
# echo "active 1" > /sys/class/net/bond1/bonding/fail_over_mac
```

3. Add slave interfaces to the bonded interface:

```
# ip link set enccw0.0.8050 master bond1
# ip link set enccw0.0.1108 master bond1
# ip link show enccw0.0.8050
5: enccw0.0.8050: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond1 state UNKNOWN
   mode DEFAULT qlen 1000
    link/ether 02:11:10:66:1f:fb brd ff:ff:ff:ff:ff:ff
# ip link show enccw0.0.1108
6: enccw0.0.1108: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master bond1 state UNKNOWN
   mode DEFAULT qlen 1000
    link/ether 02:00:bb:66:1f:ec brd ff:ff:ff:ff:ff:ff
```

4. Activate the interface:

```
# ip link set bond1 up
# ip link show bond1
8: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT
    link/ether 02:11:10:66:1f:fb brd ff:ff:ff:ff:ff:ff
```

To verify the bonding settings, issue:

```
# cat /proc/net/bonding/bond1
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup) (fail_over_mac active)
Primary Slave: None
Currently Active Slave: enccw0.0.8050
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: enccw0.0.8050
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 02:11:10:66:1f:fb
Slave queue ID: 0

Slave Interface: enccw0.0.1108
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 02:00:bb:66:1f:ec
Slave queue ID: 0
```

Related tasks:

"Configuring a MacVTap interface" on page 98
Configure network interfaces, such as Ethernet interfaces, bonded interfaces, virtual LANs, through a direct MacVTap interface.

Preparing a virtual switch

Consider these aspects when setting up a virtual switch for the use of a virtual server.

Before you begin

Make sure that:

- All OSA network devices used by a virtual switch are *active bridge ports*. Active bridge ports receive all frames addressed to unknown MAC addresses.

  You achieve this by enabling the *bridge port* role of the OSA network devices.

  Please note that only one CCW group device sharing the same OSA adapter port can be configured as a *primary* bridge port. If available, the primary bridge port becomes the active bridge port.

  To verify whether an OSA network device is an active bridge port, display the **bridge_state** sysfs attribute of the device. It should be active:

  ```
  cat /sys/devices/qeth/<ccwgroup>/bridge_state
  active
  ```

  If an OSA network device is not an active bridge port, use the **znetconf** command with the **-o** option to enable the bridge port role:

  ```
  # znetconf -a <device-bus-ID> -o layer2=1 -o bridge_role=primary
  ```

  For more information about active bridge ports, see *Device Drivers, Features, and Commands*, SC33-8411
- Security-Enhanced Linux (SELinux) is enabled.
- An Open vSwitch package is installed and running. The **status openvswitch** command displays the Open vSwitch status:

  ```
  # systemctl status openvswitch
  ovsdb-server is not running
  ovs-vswitchd is not running
  ```

  If Open vSwitch is not running, enter the **start openvswitch** command:

  ```
  # systemctl start openvswitch
  Starting openvswitch (via systemctl): [ OK ]
  # systemctl status openvswitch
  ovsdb-server is running with pid 18727
  ovs-vswitchd is running with pid 18737
  ```

About this task

Further information:

- Open vSwitch command reference: openvswitch.org/support/dist-docs

Procedure

1. Create a virtual switch.

   Use the **ovs-vsctl add-br** command to create a virtual switch.

   ```
   # ovs-vsctl add-br <vswitch>
   ```

The **ovs-vsctl show** command displays the available virtual switches and their state.

To delete a virtual switch, use the **ovs-vsctl del-br** command.

2. Create an uplink port.

   To provide high availability, use the **ovs-vsctl add-bond** command to create a bonded port. Alternatively, the **ovs-vsctl add-port** command creates a single port.

   ```
   # ovs-vsctl add-bond <vswitch> <bonded-interface> <slave1> <slave2>
   ```

Example

Set up a virtual switch vswitch0, which groups the network interfaces enccw0.0.1108 and enccw0.0.a112 to a bonded interface vsbond0:

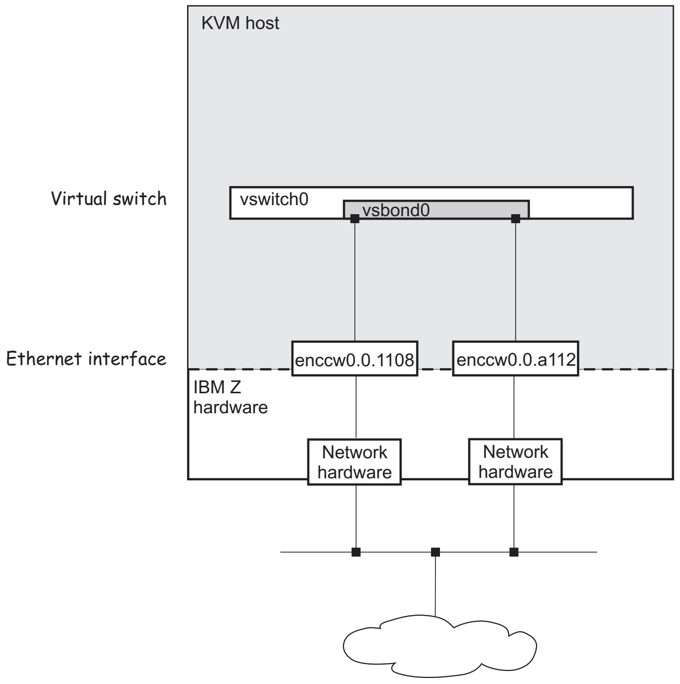


Figure 14. Virtual switch with a bonded interface

Verify that the OSA network devices are configured as bridge ports:

```
cat /sys/devices/qeth/0.0.1108/bridge_state
active

cat /sys/devices/qeth/0.0.a112/bridge_state
active
```

1. Create a virtual switch:

```
# ovs-vsctl add-br vswitch0
# ovs-vsctl show
3935bfec-241e-4610-a555-9e6f60987f87
    Bridge "vswitch0"
        Port "vswitch0"
            Interface "vswitch0"
                type: internal
    ovs_version: ...
```

2. Create an uplink port:

```
# ovs-vsctl add-bond vswitch0 vsbond0 enccw0.0.1108 enccw0.0.a112
# ovs-vsctl show
...
   Bridge "vswitch0"
        Port "vsbond0"
            Interface "enccw0.0.1108"
            Interface "enccw0.0.a112"
        Port "vswitch0"
            Interface "vswitch0"
                type: internal
...
```

Related tasks:

"Configuring a virtual switch" on page 100
Configure virtual switches as virtual Ethernet devices.

Part 3. Configuration

Chapter 9. Configuring a virtual server 49
Domain configuration-XML 51
Configuring the boot process 53
Configuring a DASD or SCSI disk as IPL device 53
Configuring an ISO image as IPL device. . . . 54
Configuring a kernel image file as IPL device . . 55
| Configuring a network IPL device 56
Example of an initial installation 57
Configuring virtual CPUs 61
Configuring the number of virtual CPUs . . . 61
Tuning virtual CPUs 62
Configuring the CPU model 63
Configuring virtual memory. 65
Configuring the amount of virtual memory. . . 65
Tuning virtual memory 65
Configuring the collection of QEMU core dumps . . 67
Configuring the user space 68
Configuring devices with the virtual server. . . . 69
Configuring the console 70
Configuring a watchdog device. 71
Disabling protected key encryption 72
Suppressing the automatic configuration of a default memory balloon device 74

Chapter 10. Configuring devices 75
Device configuration-XML 76
Configuring virtual block devices 78
Configuring a DASD or SCSI disk 78
Example of a DASD configuration 81
Example of a SCSI disk configuration. 82
Configuring an image file as storage device . . 84
Configuring a volume as storage device 86
Configuring virtual SCSI devices 88
Configuring a virtual HBA 88
Configuring a SCSI tape or medium changer device 90
Example of a multipathed SCSI tape and medium changer device configuration 93
Configuring a virtual SCSI-attached CD/DVD drive 95
Configuring virtual Ethernet devices 98
Configuring a MacVTap interface 98
Configuring a virtual switch 100
Configuring a random number generator 102

Chapter 11. Configuring storage pools 103
Storage pool and volume configuration-XMLs . . 103

| **Chapter 12. Configuring virtual networks** . . . 105

Create configuration-XML files to configure virtual servers and devices.

Chapter 9. Configuring a virtual server

The configuration of a virtual server includes the configuration of properties, such as a name, system resources, such as CPUs, memory, and a boot device, and devices, such as storage, and network devices.

Procedure

1. Create a domain configuration-XML file.

   See "Domain configuration-XML" on page 51.
2. Specify a name for the virtual server.

   Use the name element to specify a unique name according to your naming conventions.
3. Configure system resources, such as virtual CPUs, or the virtual memory.
   a. Configure a boot process.

      See "Configuring the boot process" on page 53.

   b. Configure virtual CPUs.

      See "Configuring virtual CPUs" on page 61.

   c. Configure memory.

      See "Configuring virtual memory" on page 65.

   d. Optional: Configure the collection of QEMU core dumps.

      See "Configuring the collection of QEMU core dumps" on page 67.
4. In the domain configuration-XML file, enter the virtual server device configuration.
   a. Optional: Configure the user space.

      If you do not configure the user space, libvirt configures an existing user space automatically.

      See "Configuring the user space" on page 68.

   b. Configure persistent devices.

      See "Configuring devices with the virtual server" on page 69.

   c. Configure the console device.

      See "Configuring the console" on page 70.

   d. Optional: Configure a watchdog device.

      See "Configuring a watchdog device" on page 71.

   e. Optional: Disable the generation of cryptographic wrapping keys and the use of protected key management operations on the virtual server.

      See "Disabling protected key encryption" on page 72.

   f. Optional: Libvirt automatically generates a default memory balloon device for the virtual server.

      To prohibit this automatism, see "Suppressing the automatic configuration of a default memory balloon device" on page 74.
5. Save the domain configuration-XML file according to your virtual server administration policy.

What to do next

Define the virtual server to libvirt based on the created domain configuration-XML file as described in "Defining a virtual server" on page 110.

Domain configuration-XML

Configure a virtual server with a domain configuration-XML file.

Root element

domain
: Specify `kvm` as the domain type.

domain type attribute:	kvm

Selected child elements

name
: Assigns a unique name to the virtual server. You use this name to manage the virtual server.

memory
: Specifies the amount of memory that is allocated for a virtual server at boot time.

vcpu
: Specifies the maximum number of CPUs for a virtual server.

cputune
: Groups the CPU tuning parameters:

  shares
  : Optionally specifies the initial CPU weight. The default is 1024.

os
: Groups the operating system parameters:

  type
  : Specifies the machine type.

  kernel
  : Optionally specifies the kernel image file on the host.

  initrd
  : Optionally specifies the initial ramdisk on the host.

  cmdline
  : Optionally specifies command-line arguments.

iothreads
: Assigns threads that are dedicated to I/O operations on virtual block devices to the virtual server.

on_poweroff
: Configures the behavior of the virtual server when it is shut down.

on_reboot
: Configures the behavior of the virtual server when it is rebooted.

on_crash
: Configures the behavior of the virtual server if it crashes. The `preserve` value prevents debug data from being discarded.

on_crash element:	preserve

devices
: Configures the devices that are persistent across virtual server reboots.

Example

```
<domain type="kvm">
  <name>vserv1</name>
  <memory unit="GiB">4</memory>
  <vcpu>2</vcpu>
  <cputune>
    <shares>2048</shares>
  </cputune>

  <os>
    <type arch="s390x" machine="s390-ccw-virtio">hvm</type>
  </os>
  <iothreads>1</iothreads>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>preserve</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-system-s390x</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
    </disk>
    <interface type="direct">
      <source dev="bond0" mode="bridge"/>
      <model type="virtio"/>
    </interface>
    <console type="pty">
      <target type="sclp"/>
    </console>
    <memballoon model="none"/>
  </devices>
</domain>
```

Related reference:

Chapter 28, "Selected libvirt XML elements," on page 189
These libvirt XML elements might be useful for you. You find the complete libvirt XML reference at libvirt.org.

Configuring the boot process

Specify the device that contains a root file system, or a prepared kernel image file.

Before you begin

Ensure that there is a way to boot a guest.

About this task

When you start a virtual server, an Initial Program Load (IPL) is performed to boot the guest. You specify the boot process in the domain configuration-XML file:

- If a guest is installed, you usually boot it from a disk.

  You specify the boot device as described in "Configuring a DASD or SCSI disk as IPL device."
- Alternatively, you can specify an ISO image or an initial ramdisk and a kernel image file for a guest IPL.

  For a description, see "Configuring an ISO image as IPL device" on page 54 or "Configuring a kernel image file as IPL device" on page 55.

The running virtual server is able to reboot from different devices.

Configuring a DASD or SCSI disk as IPL device

Boot a guest from a configured disk device.

Before you begin

Prepare a DASD or a SCSI disk, which contains a root file system with a bootable kernel as described in Chapter 5, "Preparing DASDs," on page 27 or Chapter 6, "Preparing SCSI disks," on page 29.

Procedure

1. Configure the DASD or SCSI disk containing the root file system as a persistent device.

   See "Configuring devices with the virtual server" on page 69 and "Configuring a DASD or SCSI disk" on page 78.
2. Per default, the guest is booted from the first specified disk device in the current libvirt-internal configuration. To avoid possible errors, explicitly specify the boot device with the boot element in the disk device definition (see "<boot>" on page 196).

boot order attribute:	<*number*>

   | The guest is booted from the disk with the lowest specified boot order value. If the specified device has a boot menu configuration, you can use the loadparm attribute of the boot element to specify a particular menu entry to be booted.

boot loadparm attribute:	<*selection*>

Example

The following domain configuration-XML configures V1, which is booted from the
| virtual block device `0xe714` on the virtual subchannel set "0x1":

```
<domain type="kvm">
    <name>V1</name>
    ...
    <devices>
        <emulator>/usr/bin/qemu-system-s390x</emulator>
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
            <target dev="vda" bus="virtio"/>
            <address type="ccw" cssid="0xfe" ssid="0x1" devno="0xe714"/>
            <boot order="1"/>
        </disk>
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
            <target dev="vdb" bus="virtio"/>
            <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
        </disk>
        ...
    </devices>
</domain>
```

The following domain configuration-XML configures V2, which is booted from a boot menu configuration on a virtual block device `0xe716`:

```
<domain type="kvm">
    <name>V2</name>
    ...
    <devices>
        <emulator>/usr/bin/qemu-system-s390x</emulator>
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
            <target dev="vda" bus="virtio"/>
            <address type="ccw" cssid="0xfe" ssid="0x1" devno="0xe714"/>
        </disk>
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
            <target dev="vdb" bus="virtio"/>
            <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
            <boot order="1" loadparm="2"/>
        </disk>
        ...
    </devices>
</domain>
```

The `loadparm` attribute selects the second entry in the boot menu.

Configuring an ISO image as IPL device

Boot a guest from an ISO 9660 image following the EL Torito specification.

Before you begin

Usually, your distribution provides an ISO image of the installation DVD.

Procedure

1. Configure a virtual SCSI-attached CD/DVD drive as a persistent device, which contains the ISO image as virtual DVD.

   See “Configuring a virtual SCSI-attached CD/DVD drive” on page 95.

You can also configure the ISO image as a storage device, but usually you might want to take advantage of the capability to change the virtual media.

2. Per default, the guest is booted from the first specified disk device in the current libvirt-internal configuration. To avoid possible errors, explicitly specify the boot device with the boot element in the disk device definition (see "<boot>" on page 196).

boot order attribute:	<*number*>

The guest is booted from the disk with the lowest specified boot order value.

Example

1. Specify the ISO image.

   Configure the ISO image as a virtual DVD:

```
<devices>
   ...
   <controller type="scsi" model="virtio-scsi" index="4"/>
   <disk type="file" device="cdrom">
      <driver name="qemu" type="raw" io="native" cache="none"/>
      <source file="/root/SLE12SP1ServerDVDs390xGMCDVD1.iso"/>
      <target dev="sda" bus="scsi"/>
      <address type="drive" controller="4" bus="0" target="0" unit="0"/>
      <readonly/>
      <boot order="1"/>
   </disk>
   ...
</devices>
```

When you start the virtual server, it will be booted from this ISO image:

```
# virsh start vserv1 --console
Domain vserv1 started
Initializing cgroup subsys cpuacct
Linux version 3.12.4911default (geeko@buildhost) (gcc version 4.8.5
 (SUSE Linux) ) #1 SMP Wed Nov 11 20:52:43 UTC 2015 (8d714a0)
setup.289988: Linux is running under KVM in 64bit mode
Zone ranges:
 DMA      [mem 0x000000000x7fffffff]
 Normal   empty
...
```

2. Provide a disk for the guest installation:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
</disk>
```

Configuring a kernel image file as IPL device

As an alternative to booting an installed guest from a DASD or a SCSI disk, you might want to boot from a kernel image file residing on the host for setup purposes.

Procedure

1. Specify the initial ramdisk, the kernel image file, and the kernel parameters.

You get this information from the installation file and the parameter file of your product or distribution.

a. Specify the fully qualified path to the initial ramdisk on the host with the initrd element, which is a child of the os element (see "<initrd>" on page 219).

initrd element:	*<initial-ramdisk>*

b. Specify the fully qualified path to the kernel image file in the kernel element, which is a child of the os element (see "<kernel>" on page 223).

kernel element:	*<kernel-image-file>*

c. Pass command-line arguments to the installer by using the cmdline element, which is a child of the os element (see "<cmdline>" on page 199).

   You can use the command line parameters that are supported by your product or distribution.

cmdline element:	*<command-line-arguments>*

2. Configure all disks that are needed for the boot process as persistent devices.

   If you are booting from the kernel image file as an initial installation, make sure to provide a disk for the guest installation.

Example

1. Specify the kernel image file in the os element:

```
<os>
    ...
    <initrd>initial-ramdisk</initrd>
    <kernel>kernel-image</kernel>
    <cmdline>command-line-parameters</cmdline>
</os>
```

2. Provide a disk for the guest installation:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
</disk>
```

Configuring a network IPL device

You can boot the operating system in a KVM virtual server from a network boot server.

Before you begin

A network boot server and a connection from your KVM host to that server must be in place.

Procedure

1. Configure an interface to a virtual network, to an Open vSwitch, or for a direct MacVTap connection (see "<interface>" on page 220).

interface type attribute:	network | bridge | direct

2. Use the source element as a child of the interface element, to specify the network or bridge that provides the connection to the network boot server (see "<source> as child element of <interface>" on page 252).
   - For a virtual network:

source network attribute:	*<network-name>*

   - For an Open vSwitch or a direct MacVTap connection:

source dev attribute:	*<bridge-interface-name>*
source mode attribute:	bridge

3. Specify `virtio` as the interface type with model element, which is a child of the interface element (see "<model> as a child element of <interface>" on page 233).

model type attribute:	virtio

4. Per default, the guest is booted from the first specified disk device in the current libvirt-internal configuration. To avoid possible errors, explicitly specify the boot device with the boot element, which is a child of the interface element (see "<boot>" on page 196).

boot order attribute:	*<number>*

   The guest is booted from the device with the lowest specified boot order value.
5. Specify the device type as CCW and a device bus-ID with the address element as a child of the interface element (see "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 192).

address type attribute:	CCW
address cssid attribute:	0xfe
address ssid attribute:	*<ssid>*
address devno attribute:	*<devno>*

Example

```
<domain name="vs003n">
  ...
  <interface type="network">
    <source network="boot-net"/>
    <model type="virtio"/>
    <boot order="1"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xb001"/>
  </interface>
  ...
</domain>
```

In the example, the first boot device in the boot order of the KVM virtual server `vs003n` is the CCW network device with bus ID 0.0.b001.

Example of an initial installation

The guest installation process depends on your product or distribution.

Procedure

1. For an initial installation, you need to provide installation files for the virtual server, such as an ISO image of the installation DVD, the kernel image file, and the initial ramdisk.

   The name and the location of these files depend on your product, your distribution or your installation process.

   You can either mount the ISO image containing the installation files during the guest installation process, copy the required files to the host file system, or connect to an FTP server.
2. Create a domain configuration-XML file.
   a. If you intend to boot from an ISO image, the domain configuration-XML file should contain:
      - The fully qualified path and filename of the ISO image.
      - A persistent device configuration for the device that will contain the bootable installed guest.

      Example:

```
<domain>
   ...
   <os>
       ...
   </os>
   ...
   <devices>
      <emulator>/usr/bin/qemu-system-s390x</emulator>

      <!-- IPL device -->
      <controller type="scsi" model="virtio-scsi" index="4"/>
      <disk type="file" device="cdrom">
         <driver name="qemu" type="raw" io="native" cache="none"/>
         <source file="/root/SLE12SP1ServerDVDs390xGMCDVD1.iso"/>
         <target dev="sda" bus="scsi"/>
         <address type="drive" controller="4" bus="0" target="0" unit="0"/>
         <readonly/>
         <boot order="1"/>
      </disk>

      <!-- guest installation device -->
      <disk type="block" device="disk">
         <driver name="qemu" type="raw" cache="none"
                 io="native" iothread="1"/>
         <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
         <target dev="vda" bus="virtio"/>
      </disk>

      <console type="pty">
         <target type="sclp"/>
      </console>
   </devices>
</domain>
```

   b. If you intend to boot from a kernel image file and an initial ramdisk, the domain configuration-XML file should contain:
      - The fully qualified path and filename of the kernel image.
      - The fully qualified path and filename of the initial ramdisk.
      - The kernel command-line parameters.

- A persistent device configuration for the device that will contain the bootable installed guest.

Example:

```
<domain>
    ...
    <os>
        ...
        <!-- Boot kernel - remove 3 lines                    -->
        <!-- after a successful initial installation         -->

        <initrd>initial-ramdisk</initrd>
        <kernel>kernel-image</kernel>
        <cmdline>command-line-parameters</cmdline>
        ...
    </os>
    ...
    <devices>
        <emulator>/usr/bin/qemu-system-s390x</emulator>

        <!-- guest installation device -->
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none"
                    io="native" iothread="1"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
            <target dev="vda" bus="virtio"/>
        </disk>

        <console type="pty">
            <target type="sclp"/>
        </console>
    </devices>
</domain>
```

3. Start the virtual server for the initial installation.
4. Install the guest as described in your distribution documentation.
5. When a bootable guest is installed, modify the domain configuration-XML using **virsh edit** to boot from the IPL disk containing the boot record.
   a. In case you installed the guest using the ISO image:

   Example:

```
<domain>
   ...
   <os>
       ...
   </os>
   ...
   <devices>
      <emulator>/usr/bin/qemu-system-s390x</emulator>

      <!-- IPL device -->
      <controller type="scsi" model="virtio-scsi" index="4"/>
      <disk type="file" device="cdrom">
         <driver name="qemu" type="raw" io="native" cache="none"/>
         <source file="/root/SLE12SP1ServerDVDs390xGMCDVD1.iso"/>
      <target dev="sda" bus="scsi"/>
      <address type="drive" controller="4" bus="0" target="0" unit="0"/>
      <readonly/>
   </disk>
   <!-- guest IPL disk -->
   <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none"
              io="native" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
   </disk>

   <console type="pty">
      <target type="sclp"/>
   </console>
  </devices>
</domain>
```

b. In case you installed the guest using the kernel image and the initial ramdisk:

Example:

```
<domain>
    ...
    <os>
        ...
    </os>
    ...
    <devices>
        <emulator>/usr/bin/qemu-system-s390x</emulator>
        <!-- guest IPL disk -->
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none"
                    io="native" iothread="1"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
            <target dev="vda" bus="virtio"/>
            <boot order="1"/>
        </disk>

        <console type="pty">
            <target type="sclp"/>
        </console>
    </devices>
</domain>
```

6. From now on, you can start the virtual server using this domain configuration-XML. The virtual server boots the installed guest from the IPL disk.

Configuring virtual CPUs

Configure virtual CPUs for a virtual server.

Related concepts:

Chapter 21, “CPU management,” on page 159
Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks:

“Managing virtual CPUs” on page 138
Modify the number of virtual CPUs and the portion of the run time that is assigned to the virtual CPUs of a defined virtual server.

Configuring the number of virtual CPUs

Configure the number of virtual CPUs for a virtual server.

Procedure

1. You can configure the number of virtual CPUs that are available for the defined virtual server by using the vcpu element (see “<vcpu>” on page 259).

   If you do not specify the vcpu element, the maximum number of virtual CPUs available for a virtual server is 1.

vcpu element:	*<number-of-CPUs>*

   Note: It is not useful to configure more virtual CPUs than available host CPUs.
2. To configure the actual number of virtual CPUs that are available for the virtual server when it is started, specify the current attribute. The value of the current attribute is limited by the maximum number of available virtual CPUs.

   If you do not specify the current attribute, the maximum number of virtual CPUs is available at startup.

vcpu current attribute:	*<number>*

Example

This example configures 5 virtual CPUs, which are all available at startup:

```
<domain type="kvm">
    ...
    <vcpu>5</vcpu>
    ...
</domain>
```

This example configures a maximum of 5 available virtual CPUs for the virtual server. When the virtual server is started, only 2 virtual CPUs are available. You can modify the number of virtual CPUs that are available for the running virtual server using the virsh **setvcpus** command (see “Modifying the number of virtual CPUs” on page 138).

```
<domain type="kvm">
    ...
    <vcpu current="2">5</vcpu>
    ...
</domain>
```

Tuning virtual CPUs

Regardless of the number of its virtual CPUs, the CPU weight determines the shares of CPU time which is dedicated to a virtual server.

About this task

For more information about the CPU weight, see "CPU weight" on page 160.

Procedure

Use the cputune element to group CPU tuning elements.

You specify the CPU weight by using the shares element (see "<shares>" on page 247).

shares element:	*<CPU-weight>*

Example

```
<domain>
    ...
    <cputune>
        <shares>2048</shares>
    </cputune>
    ...
</domain>
```

Configuring the CPU model

The CPU model configuration specifies the features of the virtual CPUs that are provided to the virtual server.

About this task

You can use a generic specification that resolves to a basic set of CPU features on any hardware model. Use an explicit configuration if you must satisfy special requirements, for example:

- Disable a CPU feature that causes problems for a particular application.
- Keep the option for a live migration to an earlier hardware model that does not support all CPU features of the current hardware (see “IBM Z hardware model” on page 128).
- Keep the option for a live migration to a KVM host with an earlier QEMU version that does not support all CPU features of the current version.

Procedure

- To configure the basic set of CPU features that is provided by the hardware, specify:

cpu mode attribute:	host-model

  (see “<cpu>” on page 202)
- To define a CPU model with a specific set of hardware features, specify:
  1. Declare that a specific CPU model is to be configured.

cpu mode attribute:	custom
cpu match attribute:	exact

  2. Specify an existing CPU model with the <model> element as a child of the <cpu> element.

model element:	*<cpu_model>*

     (see “<model> as a child element of <cpu>” on page 232)

     Where *<cpu_model>* is one of the models listed in the <domainCapabilities> XML. Issue **`virsh domcapabilities`** to display the contents of the XML file. Eligible values are specified with <model> tags that have the attribute `useable="yes"`.

     Example: This example identifies `z14` as an eligible CPU model.

     ```
     <model usable="yes">z14</model>
     ```

     The model specifications are in one of the forms that follow:

     <mainframe_model>
     : Specifies the default CPU features for an original mainframe hardware release. This default is the subset of features that are supported by the QEMU version of the KVM host. For example, `z13` specifies the QEMU supported CPU features of an IBM z13® mainframe when it first became available in February 2015.

     <mainframe_model>.<n>
     : Specifies the default CPU features for the *<n>*th major hardware release of a mainframe model. This default is the subset of features that are supported by the QEMU version of the KVM host. For

example, `z13.2` specifies the QEMU supported CPU features of an IBM z13 mainframe with its first major update in February 2016 (informally also known as GA2).

<*mainframe_model*>-base or <*mainframe_model*>.<*n*>-base
Other than the default specifications, which depend on the QEMU version and can resolve to different subsets of features, the `-base` suffix specifies a fixed subset. This subset is supported by any QEMU version that supports the mainframe model. At the peril of not using the full hardware potential, the `-base` suffix reduces QEMU dependencies for an intended live migration.

3. Optionally, use one or more <feature> elements as child elements of the <cpu> element. With each <feature> element, you can add or remove an individual CPU feature from the CPU model of the previous step (see "<feature>" on page 213).

feature policy attribute:	require | disable
feature name attribute:	<*cpu_feature*>

Where <*cpu_feature*> is one of the features as listed by the **`qemu-system-s390x -cpu help`** command.

Example

- To use all available QEMU supported CPU features of any mainframe model:

```
<cpu mode="host-model"/>
```

- To require the QEMU supported CPU features of a z14 mainframe, but without the `iep` feature:

```
<cpu mode="custom">
  <model>z14</model>
  <feature policy="disable" name="iep">
</cpu>
```

As for other parts of the domain configuration-XML, the CPU model specification is expanded in the internal XML representation of a defined and of a started virtual server.

Configuring virtual memory

Configure the virtual server memory.

Related concepts:

Chapter 22, “Memory management,” on page 163
The memory configured for a virtual server appears as physical memory to the guest operating system but is realized as a Linux virtual address space.

Related tasks:

“Managing virtual memory” on page 143
Specify a soft limit for the amount of physical host memory used by a virtual server.

Configuring the amount of virtual memory

Configure the amount of memory that is available for the virtual server at startup time.

Procedure

Use the memory element which is a child of the domain element (see “<memory>” on page 229).

memory element:	*<memory-size>*
memory unit attribute:	*<memory-unit>*

Example

```
<domain type="kvm">
    <name>vserv1</name>
    <memory unit="MB">512</memory>
    ...
<domain>
```

The memory that is configured for the virtual server when it starts up is 512 MB.

Tuning virtual memory

A configured soft limit allows the host to limit the physical host memory resources used for the virtual server memory in case the host experiences high swapping activity.

About this task

For more information about memory tuning, see Chapter 22, “Memory management,” on page 163.

Procedure

Use the memtune element to group memory tuning elements.

Specify a soft limit by using the soft_limit element (see “<soft_limit>” on page 248).

soft_limit element:	*<soft-limit-size>*
soft_limit unit attribute:	*<unit of the soft-limit-size>*

Example

```
<domain type="kvm">
    <name>vserv1</name>
    ....
    <memory unit="MB">512</memory>
    <memtune>
        <soft_limit unit="MB">256</soft_limit>
    </memtune>
    ...
<domain>
```

The memory configured for virtual server vserv1 is 512 MB. In case the host is under memory pressure, it might limit the physical host memory usage of vserv1 to 256 MB.

Configuring the collection of QEMU core dumps

Exclude the memory of a virtual server when collecting QEMU core dumps on the host.

Procedure

To exclude the memory of a virtual server from a QEMU core dump, specify:

memory dumpCore attribute:	off

(see “<memory>” on page 229)

Example

```
<domain type="kvm">
    <name>vserv1</name>
    <memory unit="MB" dumpCore="off">512</memory>
    ...
<domain>
```

Configuring the user space

The user space process qemu-system-s390x realizes the virtual server on the IBM Z host. You might want to configure it explicitly.

Procedure

The optional emulator element contains path and file name of the user space process (see "<emulator>" on page 212).
The emulator element is a child of the devices element. If you do not specify it, libvirt automatically inserts the user space configuration to the libvirt-internal configuration when you define it.

emulator element:	*<emulator-file>*

Example:

```
<devices>
   <emulator>/usr/bin/qemu-system-s390x</emulator>
   ...
</devices>
```

Configuring devices with the virtual server

The domain configuration-XML file specifies virtual devices that are defined along with the virtual server.

Before you begin

- Ensure that the devices are prepared for the use of the virtual server.
- Devices that can be attached to an already defined virtual server are configured in separate device configuration-XML files.

Procedure

1. Optional: To improve the performance of I/O operations on DASDs and SCSI disks, specify the number of I/O threads to be supplied for the virtual server.

   For more information about I/O threads, see "I/O threads" on page 167.

iothreads element:	*<number-of-IOthreads>*

   (see "<iothreads>" on page 221)

   Example:

   ```
   <domain>
       ...
       <iothreads>1</iothreads>
       ...
   </domain>
   ```

2. Specify a configuration-XML for each device.

   Chapter 10, "Configuring devices," on page 75 describes how to specify a configuration-XML for a device.

3. For each device to be defined with the virtual server, place the configuration-XML as a child element of the devices element in the domain configuration-XML file.

Example

```
<domain type="kvm">
    <iothreads>1</iothreads>
    ...
    <devices>
        ...
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
            <target dev="vda" bus="virtio"/>
        </disk>
        ...
    </devices>
</domain>
```

Configuring the console

Configure the console by using the console element.

Procedure

1. You configure the host representation of the console by using the console type attribute (see "<console>" on page 200).

   To configure a pty console, enter:

console type attribute:	pty

2. You configure the virtual server representation of the console by using the target type attribute (see "<target> as child element of <console>" on page 254).

   To configure a service-call logical processor (SCLP) console interface, enter the "sclp" value.

target type attribute:	sclp

   You can also configure a virtio console by entering the target type attribute value "virtio".

3. Optional: Specify a log file which collects the console output in addition to the display in the console window.

   Use the log element to specify the log file (see "<log>" on page 226). Optionally, you can specify whether or not the log file will be overwritten in case of a virtual server restart. By default, the log file is overwritten.

log file attribute:	*<log-file>*
log append attribute:	off | on

Example

This example configures a pty console. The console output is collected in the file `/var/log/libvirt/qemu/vserv-cons0.log`. A virtual server restart overwrites the log file.

```
<devices>
    ...
    <console type="pty">
       <target type="sclp" port="0"/>
       <log file="/var/log/libvirt/qemu/vserv-cons0.log" append="off"/>
    </console>
<devices/>
```

Related tasks:

"Connecting to the console of a virtual server" on page 149
Open a console when you start a virtual server, or connect to the console of a running virtual server.

Configuring a watchdog device

A watchdog device provides a guest watchdog application with access to a watchdog timer.

About this task

When the guest is loading the watchdog module, it provides the new device node `/dev/watchdog` for the watchdog device. The watchdog timer is started when the watchdog device is opened by the guest watchdog application. The watchdog application confirms a healthy system state by writing to `/dev/watchdog` at regular intervals. If nothing is written to the device node for a specified time, the watchdog timer elapses, and QEMU assumes that the guest is in an error state. QEMU then triggers a predefined action against the guest. For example, the virtual server might be terminated and rebooted, or a dump might be initiated.

Procedure

Use the watchdog element as child of the devices element to configure a watchdog device (see "<watchdog>" on page 263).

watchdog model attribute:	diag288
watchdog action attribute:	<*timeout-action*>

Example

```
<devices>
    ...
    <watchdog model="diag288" action="inject-nmi"/>
    ...
</devices>
```

Disabling protected key encryption

The generation of cryptographic wrapping keys and the use of protected key management operations on the virtual server is enabled by default.

Before you begin

The use of cryptographic protected key management operations on the virtual server is enabled by default, if:

1. IBM Z Central Processor Assist for Cryptographic Functions (CPACF) is installed.
2. The logical partition running the host is enabled for CPACF key management operations.

   You enable CPACF key management operations on the security page of the Customize Activation Profiles task, which is part of the CPC Operational Customization tasks list.

About this task

The CPACF hardware provides a set of key management operations for clear key encryption, pseudo random number generation, hash functions, and protected key encryption. The use of protected key management operations on the virtual server can be configured.

Symmetric encryption uses a cryptographic key to encrypt messages, files, or disks, and the identical key to decrypt them. A cryptographic key is created using a specific algorithm:

- Data Encryption Algorithm (DEA), also known as Data Encryption Standard (DES)
- Triple DEA (3DEA, TDEA), which is based on DEA and is also known as Triple DES, 3DES, or TDES
- Advanced Encryption Standard (AES)

A *protected key* is a cryptographic key which is itself encrypted by a so-called *wrapping key*, thus protecting it from unauthorized access.

The unique wrapping keys are associated with the lifetime of a virtual server. Each time the virtual server is started, its wrapping keys are regenerated. There are two wrapping keys: one for DEA or TDEA keys, and one for AES keys.

A set of key management operations can be performed on the virtual server. *Protected key management operations* are used to encrypt a clear key using a wrapping key.

If you disable the generation of wrapping keys for DEA/TDEA or for AES, you also disable the access to the respective protected key management operations on the virtual server.

Procedure

You configure the generation of wrapping keys by using the keywrap element (see "<keywrap>" on page 225).
Its child element cipher (see "<cipher>" on page 198) enables or disables the generation of a wrapping key and the use of the respective protected key

management operations. By default, both the AES and DEA/TDEA wrapping keys are generated.

Specify the wrapping key generation that is to be disabled or enabled.

cipher name attribute:	aes | dea
cipher state attribute:	*<state>*

<state>

on Default; enables the wrapping key generation.

off Disables the wrapping key generation.

Example

This example disables the generation of an AES wrapping key. The DEA/TDEA wrapping key is generated by default.

```
<keywrap>
    <cipher name="aes" state="off"/>
</keywrap>
```

The example is equivalent to this one:

```
<keywrap>
    <cipher name="aes" state="off"/>
    <cipher name="dea" state="on"/>
</keywrap>
```

Suppressing the automatic configuration of a default memory balloon device

By default, libvirt automatically defines a default memory balloon device for a virtual server configuration.

Procedure

To avoid the automatic creation of a default memory balloon device, specify:

memballoon model attribute:	none

(see “<memballoon>” on page 228)

Example

```
<devices>
    ...
    <memballoon model="none"/>
    ...
</devices>
```

Chapter 10. Configuring devices

When you configure storage and network devices, you specify the physical hardware on which the resources are based.

About this task

From the virtual server point of view, all disks, tapes, CD-ROMs, DVDs, or image files you provide for it as storage devices, and all devices you provide for it as network devices, are accessed as CCW devices. All CCW devices are accessed through a virtual channel subsystem.

The virtual channel subsystem provides only one virtual channel path that is shared by all CCW devices. The virtual server views the virtual channel subsystem-ID 0x00. When you define a device for a virtual server, you use the reserved channel subsystem-ID 0xfe.

The virtual control unit model is used to reflect the device type.

The virtual server sees the following predefined values:

Virtual channel subsystem-ID	0x00
Virtual channel path type	0x32
Virtual control unit type	0x3832
Virtual control unit model for:	
• Network (virtio-net) devices	0x01
• Block (virtio-block) devices (SCSI disks, DASD disks, CD-ROMs, DVDs, or image files)	0x02
• Serial devices Deprecated	0x03
• Random number generators (RNGs) Do not configure a virtual random number generator for a virtual server, unless the host is equipped with a hardware random number generator, such as the secure IBM CCA coprocessor of a Crypto Express adapter.	0x04
• Balloon devices This device can be suppressed in the configuration of the virtual server	0x05
• SCSI Host Bus Adapter (virtio-scsi)	0x08

Procedure

1. Configure the device as described in:
   - “Configuring virtual block devices” on page 78
   - “Configuring virtual SCSI devices” on page 88
   - “Configuring virtual Ethernet devices” on page 98

2. To configure a device that is defined along with the virtual server, enter the device configuration as a child element of the devices element in the domain configuration-XML file.
3. To configure a device that can be attached to an already defined virtual server, enter the device configuration in a separate device configuration-XML file.

Device configuration-XML

Devices that are configured with separate device configuration-XML files can be attached to an already defined virtual server.

Virtual block device

Root element
disk

Selected child elements
driver, source, target, address

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vda" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x3c1b"/>
</disk>
```

Virtual SCSI device

Root element
hostdev

Selected child elements
source, address

Example

```
<hostdev mode="subsystem" type="scsi">
    <source>
       <adapter name="scsi_host0"/>
       <address bus="0" target="0" unit="0"/>
    </source>
    <address type="scsi" controller="0" bus="0" target="1" unit="1"/>
</hostdev>
```

Virtual Host Bus Adapter

Root element
controller

Selected child elements
address

Example

```
<controller type="scsi" model="virtio-scsi" index="0">
    <address type="ccw" cssid="0xfe" ssid="0" devno="0x0002"/>
</controller>
```

Virtual Ethernet device

Root element
interface

Selected child elements
mac, source, model

Example

```
<interface type="direct">
    <source dev="bond0" mode="bridge"/>
    <model type="virtio"/>
</interface>
```

Related reference:

Chapter 28, “Selected libvirt XML elements,” on page 189
These libvirt XML elements might be useful for you. You find the complete libvirt XML reference at libvirt.org.

Configuring virtual block devices

Configure storage devices, such as DASDs, SCSI disks, or image files, as virtual block devices for a virtual server.

About this task

- "Configuring a DASD or SCSI disk"
- "Configuring an image file as storage device" on page 84
- "Configuring a volume as storage device" on page 86

Configuring a DASD or SCSI disk

Specify DASDs and FC-attached SCSI disks as virtio block devices in the configuration-XML.

Before you begin

Make sure that

- DASDs are prepared as described in Chapter 5, "Preparing DASDs," on page 27.
- SCSI disks are prepared as described in Chapter 6, "Preparing SCSI disks," on page 29.

If the virtual server uses Logical Volume Manager (LVM), be sure to exclude these devices from the host LVM configuration. Otherwise, the host LVM might interpret the LVM metadata on the disk as its own and cause data corruption. For more information, see "Logical volume management" on page 167.

About this task

You specify DASDs or SCSI disks by a device node. If you want to identify the device on the host as it appears to the virtual server, specify a device number for the virtual block device.

Procedure

1. Configure the device.
   a. Configure the device as virtio block device.

disk type attribute:	block
disk device attribute:	disk

(see "<disk>" on page 207)

   b. Specify the user space process that implements the device.

driver name attribute:	qemu
driver type attribute:	raw
driver cache attribute:	none
driver io attribute:	native
driver iothread attribute:	*<IOthread-ID>*

(see "<driver> as child element of <disk>" on page 210)

<IOthread-ID> indicates the I/O thread dedicated to perform the I/O operations on the device.

For devices that are defined in the domain configuration-XML file:
Specify a value between 1 and the number of I/O threads

configured by the iothreads element in the domain configuration-XML file. To improve performance, be sure that there is an I/O thread dedicated for this device.

For devices that are defined in separate device configuration-XML files:
Specify the ID of the I/O thread that is created when the device is attached.

Example:

```
<domain>
    ...
    <iothreads>2</iothreads>
    ...
    <devices>
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
            ...
        </disk>
    </devices>
    ....
</domain>
```

In this example, I/O thread with ID 2 is dedicated to perform the input operations to and the output operations from the device.

For more information about I/O threads, see "I/O threads" on page 167.

c. Specify virtio as the virtual server disk device type.

target bus attribute:	virtio

(see "<target> as child element of <disk>" on page 255)

2. Identify the device on the host.

Specify a device node of the device.

source dev attribute:	*<device-node>*

(see "<source> as child element of <disk>" on page 249)

Note: You should be aware that the selection of the specified device node determines whether or not you will be able to:
- Perform a live migration of the virtual server accessing the device.
- Migrate the storage to another storage server or another storage controller.

For DASDs:
Use udev-created device nodes.

All udev-created device nodes support live migration. By-uuid device nodes support also storage migration, because they are hardware-independent.

For SCSI disks:
Use device mapper-created device nodes.

Device mapper-created device nodes are unique and always specify the same device, irrespective of the host which runs the virtual server.

Please be aware that setting up multipathing on the host without passing the device mapper-created device nodes to the virtual server leads to the loss of all multipath advantages regarding high availability and performance.

3. Identify the device on the virtual server.

a. Specify a unique logical device name.

Logical device names are of the form `vd<x>`, where <*x*> can be one or more letters. Do not confuse the logical device name with the standard device name. The standard device name is assigned to the device on the virtual server in the order the device is detected. It is not persistent across guest reboots.

target dev attribute:	*<logical-device-name>*

(see "<target> as child element of <disk>" on page 255)

b. Optional: Specify a unique device number.

You specify a device bus-ID, which is of the form

```
fe.n.dddd
```

where `n` is the subchannel set-ID and `dddd` is the device number. The channel subsystem-ID `0xfe` is reserved to the virtual channel.

The virtual server sees the channel subsystem-ID `0x0` instead.

Tip: Do not mix device specifications with and without device numbers.

address type attribute:	ccw
address cssid attribute:	0xfe (reserved channel subsystem-ID)
address ssid attribute:	*<subchannel-set-ID>*
address devno attribute:	*<device-number>*

(see "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 192)

Example: KVM host device bus-ID `fe.0.1a12` is seen by the virtual server as device bus-ID `0.0.1a12`.

If you do not specify a device number, a device bus-ID is automatically generated by using the first available device bus-ID starting with subchannel set-ID `0x0` and device number `0x0000`.

Assign device numbers depending on your policy, such as:

- Assigning identical device numbers on the virtual server and on the host enable the virtual server user to identify the real device.
- Assigning identical device numbers on the virtual servers allows you to create identical virtual servers.

Related concepts:

Chapter 2, "Virtual block devices," on page 9
DASDs, FC-attached SCSI disks, image files and logical volumes are virtualized as virtio block devices.

Example of a DASD configuration

To see the device nodes of the prepared DASDs on the host, enter:

```
# lsdasd
Bus-ID     Status      Name      Device  Type  BlkSz  Size      Blocks
==============================================================================
0.0.7500   active      dasda     94:0    ECKD  4096   7043MB    1803060
0.0.7600   active      dasdb     94:4    ECKD  4096   7043MB    1803060
```

The udev-created by-path device node for device `0.0.7500` is `/dev/disk/by-path/ccw-0.0.7500`.

Define the devices:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/disk/by-path/ccw-0.0.7500"/>
    <target dev="vda" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x7500"/>
</disk>
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
    <source dev="/dev/disk/by-path/ccw-0.0.7600"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x7600"/>
</disk>
```

This example follows the policy to assign the host device number to the virtual server.

The virtual server sees the standard device nodes, which are of the form `/dev/vd<x>`, where *<x>* represents one or more letters. The mapping between a name and a certain device is not persistent across guest reboots. To see the current mapping between the standard device nodes and the udev-created by-path device nodes, enter:

```
[root@guest:] # ls /dev/disk/by-path -l
total 0
lrwxrwxrwx 1 root root  9 May 15 15:20 ccw-0.0.7500 -> ../../vda
lrwxrwxrwx 1 root root 10 May 15 15:20 ccw-0.0.7600 -> ../../vdb
```

The virtual server always sees the control unit type `3832`. The control unit model indicates the device type, where `02` is a block device:

```
[root@guest:] # lscss
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.7500 0.0.0000  0000/00 3832/02 yes  80  80  ff   00000000 00000000
0.0.7600 0.0.0001  0000/00 3832/02 yes  80  80  ff   00000000 00000000
```

Example of a SCSI disk configuration

To see the device mapper-created device nodes of the prepared devices on the host, enter:

```
# multipathd -k'show topology'
36005076305ffc1ae00000000000021df dm-3 IBM     ,2107900
size=30G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=0 status=active
  |- 1:0:7:1088372769 sdm   8:192  active ready running
  |- 1:0:3:1088372769 sdn   8:208  active ready running
  |- 1:0:5:1088372769 sdo   8:224  active ready running
  |- 1:0:4:1088372769 sdl   8:176  active ready running
  |- 0:0:3:1088372769 sdbd  67:112 active ready running
  |- 0:0:4:1088372769 sdax  67:16  active ready running
  |- 0:0:8:1088372769 sdbj  67:208 active ready running
  `- 0:0:6:1088372769 sdbp  68:48  active ready running
...
36005076305ffc1ae00000000000021d5 dm-0 IBM     ,2107900
size=30G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=0 status=active
  |- 1:0:4:1087717409 sdg   8:96   active ready running
  |- 1:0:7:1087717409 sdq   65:0   active ready running
  |- 1:0:5:1087717409 sdi   8:128  active ready running
  |- 1:0:3:1087717409 sdf   8:80   active ready running
  |- 0:0:4:1087717409 sdaw  67:0   active ready running
  |- 0:0:3:1087717409 sdbc  67:96  active ready running
  |- 0:0:6:1087717409 sdbo  68:32  active ready running
  `- 0:0:8:1087717409 sdbi  67:192 active ready running
```

Define the devices:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021df"/>
    <target dev="vda" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x1a10"/>
</disk>

<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x1a12"/>
</disk>
```

The virtual server sees the standard device nodes, which are of the form `/dev/vd<x>`, where *<x>* represents one or more letters. The mapping between a name and a certain device is not persistent across guest reboots. To see the current mapping between the standard device nodes and the udev-created by-path device nodes, enter:

```
[root@guest:] # ls /dev/disk/by-path -l
total 0
lrwxrwxrwx 1 root root  9 May 15 15:20 ccw-0.0.1a10 -> ../../vda
lrwxrwxrwx 1 root root 10 May 15 15:20 ccw-0.0.1a12 -> ../../vdb
```

The virtual server always sees the control unit type `3832`. The control unit model indicates the device type, where `02` is a block device:

```
[root@guest:] # lscss
Device   Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
----------------------------------------------------------------------
0.0.1a10 0.0.0000  0000/00 3832/02 yes  80  80  ff   00000000 00000000
0.0.1a12 0.0.0001  0000/00 3832/02 yes  80  80  ff   00000000 00000000
```

Configuring an image file as storage device

Typically, you provide an image file as storage device when you intend to boot the virtual server from a boot image file.

Before you begin

Make sure that the image file exists, is initialized and accessible for the virtual server. You can provide raw image files or qcow2 image files. qcow2 image files occupy only the amount of storage that is really in use.

Use the QEMU command **`qemu-img create`** to create a qcow2 image file. See "Examples for the use of the qemu-img command" on page 351 for examples.

Procedure

1. Configure the image file.
   a. Configure the image file as virtual disk.

	raw image file:	qcow2 image file:
disk type attribute:	file	file
disk device attribute:	disk	disk

   (see "<disk>" on page 207)

   b. Specify the user space process that implements the device.

	raw image file:	qcow2 image file:
driver name attribute:	qemu	qemu
driver io attribute:	native	native
driver type attribute:	raw	qcow2
driver cache attribute:	*<cache-mode>*	*<cache-mode>*

   (see "<driver> as child element of <disk>" on page 210)

   Where *<cache-mode>* determines the QEMU caching strategy.

   Tip: For most configurations, the "none" value is appropriate.

   c. Specify virtio as the virtual server disk device type.

target bus attribute:	virtio

   (see "<target> as child element of <disk>" on page 255)

2. Identify the image file on the host.

   Specify the image file name.

source file attribute:	*<image-file-name>*

   (see "<source> as child element of <disk>" on page 249)

3. Identify the device on the virtual server.
   a. Specify a unique logical device name.

   Logical device names are of the form `vd`*<x>*, where *<x>* can be one or more letters. Do not confuse the logical device name with the standard device name. The standard device name is assigned to the device on the virtual server in the order the device is detected. It is not persistent across guest reboots.

target dev attribute:	*<logical-device-name>*

(see “<target> as child element of <disk>” on page 255)

b. Optional: Specify a device number.

You specify a device bus-ID of the form

```
fe.n.dddd
```

where `n` is the subchannel set-ID and `dddd` is the device number. The channel subsystem-ID `0xfe` is reserved to the virtual channel.

The virtual server sees the channel subsystem-ID `0x0` instead.

address type attribute:	ccw
address cssid attribute:	0xfe (reserved channel subsystem-ID)
address ssid attribute:	*<subchannel-set-ID>*
address devno attribute:	*<device-number>*

(see “<address> as child element of <controller>, <disk>, <interface>, and <memballoon>” on page 192)

Example: KVM host device bus-ID `fe.0.0009` is seen by the virtual server as device bus-ID `0.0.0009`.

If you do not specify a device number, a device bus-ID is automatically generated by using the first available device bus-ID starting with subchannel set-ID `0x0` and device number `0x0000`.

Example

This example configures the image file `/var/lib/libvirt/images/disk.img` as storage device. This image might as well be the volume of a storage pool.

```
<disk type="file" device="disk">
    <driver name="qemu" type="raw" io="native" cache="none"/>
    <source file="/var/lib/libvirt/images/disk.img"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0009"/>
</disk>
```

Related tasks:

“Configuring the boot process” on page 53
Specify the device that contains a root file system, or a prepared kernel image file.

Configuring a volume as storage device

An alternative to configuring storage pool volumes like image files is to configure them as virtual disks of type volume. Use this variation only if you rely completely on the libvirt storage pool management, and if you do not intend to migrate virtual servers accessing this device to a different host.

Before you begin

Make sure that the storage pool and the volume are configured and defined.

Procedure

1. Configure the volume.
   a. Configure the volume as virtual disk.

	raw image file:	qcow2 image file:
disk type attribute:	volume	volume
disk device attribute:	disk	disk

   (see "<disk>" on page 207)

   b. Specify the user space process that implements the device.

	raw image file:	qcow2 image file:
driver name attribute:	qemu	qemu
driver io attribute:	native	native
driver type attribute:	raw	qcow2
driver cache attribute:	*<cache-mode>*	*<cache-mode>*

   (see "<driver> as child element of <disk>" on page 210)

   Where *<cache-mode>* determines the QEMU caching strategy.

   Tip: For most configurations, the "none" value is appropriate.

   c. Specify virtio as the virtual server disk device type.

target bus attribute:	virtio

   (see "<target> as child element of <disk>" on page 255)

2. Identify the image file on the host.

   Specify the image file name.

source pool attribute:	*<pool-name>*
source volume attribute:	*<volume-name>*

   (see "<source> as child element of <disk>" on page 249)

3. Identify the device on the virtual server.
   a. Specify a unique logical device name.

   Logical device names are of the form `vd<x>`, where *<x>* can be one or more letters. Do not confuse the logical device name with the standard device name. The standard device name is assigned to the device on the virtual server in the order the device is detected. It is not persistent across guest reboots.

target dev attribute:	*<logical-device-name>*

(see “<target> as child element of <disk>” on page 255)

b. Optional: Specify a device number.

   You specify a device bus-ID of the form

   ```
   fe.n.dddd
   ```

   where `n` is the subchannel set-ID and `dddd` is the device number. The channel subsystem-ID `0xfe` is reserved to the virtual channel.

   The virtual server sees the channel subsystem-ID `0x0` instead.

address type attribute:	ccw
address cssid attribute:	0xfe (reserved channel subsystem-ID)
address ssid attribute:	*<subchannel-set-ID>*
address devno attribute:	*<device-number>*

   (see “<address> as child element of <controller>, <disk>, <interface>, and <memballoon>” on page 192)

   Example: KVM host device bus-ID `fe.0.0009` is seen by the virtual server as device bus-ID `0.0.0009`.

   If you do not specify a device number, a device bus-ID is automatically generated by using the first available device bus-ID starting with subchannel set-ID `0x0` and device number `0x0000`.

Example

This example configures logical volume blk-pool0-vol0 from the LVM pool blk-pool0 as a virtual block device.

```
<disk type="volume" device="disk">
    <driver name="qemu" type="raw" io="native" cache="none"/>
    <source pool="blk-pool0" volume="blk-pool0-vol0"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0009"/>
</disk>
```

Related tasks:

“Configuring the boot process” on page 53
Specify the device that contains a root file system, or a prepared kernel image file.

Configuring virtual SCSI devices

Configure SCSI tape devices, SCSI medium changer devices, and DVD drives as virtual SCSI devices for a virtual server.

Procedure

1. Configure a virtual HBA.

   See "Configuring a virtual HBA"
2. Configure a SCSI tape of medium changer device or a virtual SCSI-attached CD/DVD drive being attached to the virtual HBA.

   See one of the following:
   - "Configuring a SCSI tape or medium changer device" on page 90
   - "Configuring a virtual SCSI-attached CD/DVD drive" on page 95

Example

See "Example of a multipathed SCSI tape and medium changer device configuration" on page 93.

Configuring a virtual HBA

Configure virtual Host Bus Adapters (HBAs) for virtual SCSI devices.

Procedure

1. Use the controller element, which is a child of the devices element (see "<controller>" on page 201).

controller type attribute:	scsi
controller model attribute:	virtio-scsi
controller index attribute:	*<index>*

   Where *<index>* is a unique decimal integer designating in which order the virtual HBA is set online.

   Example:
   ```
   <devices>
       <controller type="scsi" model="virtio-scsi" index="0"/>
   </devices>
   ```
2. Optional: To improve performance, specify an I/O thread dedicated to perform the I/O operations on the device.

   Use the driver element, which is a child of the controller element (see "<driver> as child element of <controller>" on page 209):

driver iothread attribute:	*<IOthread-ID>*

   <IOthread-ID> indicates the I/O thread dedicated to perform the I/O operations on the virtual SCSI device which is attached to the virtual HBA. Specify a value between 1 and the number of I/O threads configured by the iothreads element in the domain configuration-XML file. To improve performance, be sure that there is an I/O thread dedicated for this device.

   See also "I/O threads" on page 167.

   Example:

```
<domain>
    ...
    <iothreads>2</iothreads>
    ...
    <devices>
        <controller type="scsi" model="virtio-scsi" index="0">
            <driver iothread="2"/>
        </controller>
    </devices>
    ....
</domain>
```

In this example, I/O thread with ID 2 is dedicated to perform the input operations to and the output operations from the device.

3. Optional: Specify the address of the device to be created.

   The controller element creates the virtual device and subchannel numbers sequentially. This can be overwritten by expanding the controller element to include an address element. The device number is used to create the virtual HBA.

address type attribute:	ccw
address cssid attribute:	0xfe (reserved channel subsystem-ID)
address ssid attribute:	*<subchannel-set-ID>*
address devno attribute:	*<device-number>*

   (see "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 192)

   Example:

```
<devices>
    <controller type="scsi" model="virtio-scsi" index="0">
        <address type="ccw" cssid="0xfe" ssid="0" devno="0x1111"/>
    </controller>
</devices>
```

Example

If you do not configure an address for an HBA, libvirt creates an address for you. You can retrieve this address with the virsh **dumpxml** command.

1. Domain configuration-XML file:

```
<domain type="kvm">
    ...
    <devices>
        <controller type="scsi" model="virtio-scsi" index="0"/>
        ...
    </devices>
</domain>
```

2. Define the virtual server to libvirt.
3. Issue the command:

```
# virsh dumpxml vserv1
```

   The current libvirt-internal configuration is displayed:

```
<domain type="kvm">
    ...
    <devices>
        <controller type="scsi" model="virtio-scsi" index="0">
            <address type="ccw" cssid="0xfe" ssid="0" devno="0x0000"/>
        </controller>
        ...
    </devices>
</domain>
```

Configuring a SCSI tape or medium changer device

Configure FC-attached SCSI tape devices and SCSI medium changers as host devices for a virtual server.

Before you begin

Make sure that, as described in Chapter 7, "Preparing SCSI tape and medium changer devices," on page 33:

- The SCSI tape or medium changer device is set up.
- You provide the SCSI device name of the SCSI tape or medium changer device.

You need a virtual HBA to connect to.

- Either use a configured virtual HBA (see "Configuring a virtual HBA" on page 88), or
- Connect to a new virtual HBA which will be automatically configured for you.

About this task

SCSI device names are freshly assigned after a host reboot or when a device is set offline and back online. This means that you have to verify an FC-attached SCSI tape or medium changer device configuration after one of these events. This limitation is also important if you plan a live migration.

Tip: Configure both FC-attached SCSI tape and medium changer devices in separate device configuration-XML files. Attach these devices only when necessary, and detach them before you migrate the virtual server, or set one of the devices in the configuration path offline.

Procedure

1. Configure the SCSI tape or medium changer device using the hostdev element (see "<hostdev>" on page 218).

hostdev mode attribute:	subsystem
hostdev type attribute:	scsi

2. Specify the SCSI tape or medium changer device on the host as child of the source element.

adapter name attribute:	scsi_host<*SCSI-host-number*>
address bus attribute:	0
address target attribute:	<*SCSI-ID*>
address unit attribute:	<*SCSI-LUN*>

(see "<adapter> as child element of <source>" on page 191 and "<address> as child element of <source>" on page 194)

3. Optional: Connect to a virtual HBA and specify a freely selectable SCSI device name on the virtual server.

address type attribute:	scsi
address controller attribute:	<*controller-index*>
address bus attribute:	0
address target attribute:	<*target*>
address unit attribute:	<*unit*>

(see "<address> as child element of <hostdev> or <disk>" on page 193)

Where

<controller-index>
   specifies the virtual HBA to which the SCSI device is connected.

   Enter the value of the controller index attribute of a configured virtual HBA or a new index value. The allocated index values must be contiguous without gaps. If you specify a new index value, a new virtual HBA is automatically configured.

   The virtual HBA is also called the *SCSI host* of the SCSI device on the virtual server.

<target>
   is a freely selectable natural number: 0 ≤ <*target*> < 256

<unit> determines the SCSI LUN on the virtual server according to the rules specified in the SCSI Architecture Model (SAM):

   0 ≤ *<unit>* < 256
      SCSI LUN := <*unit*>

   256 ≤ *<unit>* ≤ 16383
      SCSI LUN := 0x<*unit*> ∨ 0x4000

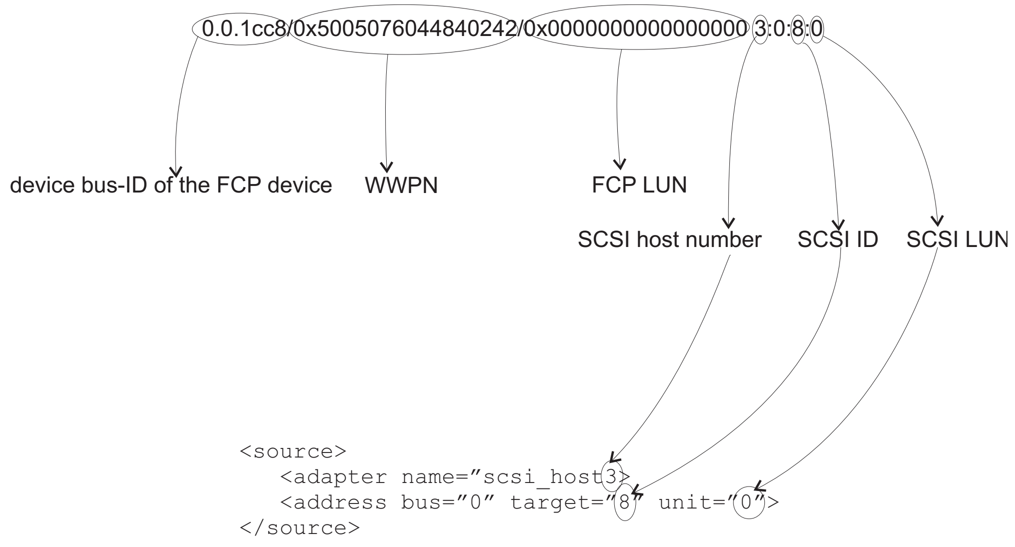
   Tip: Choose a value between 0 and 255, because these values are identically mapped to the SCSI LUN on the virtual server.

Example

Obtain the SCSI host number, the SCSI ID, and the SCSI LUN of the FC-attached SCSI tape or medium changer device:

```
# lszfcp -D
0.0.1cc8/0x5005076044840242/0x0000000000000000 3:0:8:0
```

where:


0.0.1cc8/0x5005076044840242/0x0000000000000000 3:0:8:0
device bus-ID of the FCP device
WWPN
FCP LUN
SCSI host number
SCSI ID
SCSI LUN
<source>
<adapter name="scsi_host3">
<address bus="0" target="8" unit="0">
</source>

Assign a SCSI device name to the virtual SCSI device on the virtual server. The controller attribute of the address element refers to the index attribute of the controller element.

- Domain configuration-XML file:

```
<domain type="kvm">
  <name>VM1</name>
  ...
  <devices>
     ...
    <controller type="scsi" model="virtio-scsi" index="0">
       <address type="ccw" cssid="0xfe" ssid="0" devno="0x0002"/>
    </controller>

    ...
  </devices>
</domain>
```

- Device configuration-XML file:

```
<hostdev mode="subsystem" type="scsi">
    <source>
       <adapter name="scsi_host3"/>
       <address bus="0" target="8" unit="0"/>
    </source>
    <address type="scsi" controller="0" bus="0" target="1" unit="1"/>
</hostdev>
```

Display the SCSI tape on the host:

```
# lsscsi
[3:0:8:0] tape IBM 03592E07 35CD
```

On the virtual server, the SCSI tape will be displayed like this:

```
[root@guest:] # lsscsi
[0:0:1:1] tape IBM 03592E07 35CD
```

Example of a multipathed SCSI tape and medium changer device configuration

Provide one virtual SCSI device for each configuration path.

About this task

This example provides a configuration for the topology as shown in Figure 10 on page 18.

Procedure

1. Create a domain configuration-XML file with one configured virtual HBA for each host device. This configuration groups all virtual SCSI devices that represent the same host device in an own virtual HBA.

```
<domain type="kvm">
  <name>VM1</name>
  ...
  <devices>
     ...
    <controller type="scsi" model="virtio-scsi" index="0">
       <address type="ccw" cssid="0xfe" ssid="0" devno="0x0002"/>
    </controller>
    <controller type="scsi" model="virtio-scsi" index="1">
       <address type="ccw" cssid="0xfe" ssid="0" devno="0x0004"/>
    </controller>
    ...
  </devices>
</domain>
```

2. Create separate device configuration-XML files for the SCSI tape device, both connected to the virtual HBA 0.
   a. The first file configures SCSI device name 0:0:0:0, which is the path of SCSI LUN 0 via SCSI host 0.

```
<hostdev mode="subsystem" type="scsi">
     <source>
        <adapter name="scsi_host0"/>
        <address bus="0" target="0" unit="0"/>
     </source>
     <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
</hostdev>
```

   b. The second file configures SCSI device name 1:0:0:0, which is the path via SCSI host 1.

```
<hostdev mode="subsystem" type="scsi">
     <source>
        <adapter name="scsi_host1"/>
        <address bus="0" target="0" unit="0"/>
     </source>
     <address type="scsi" controller="0" bus="0" target="0" unit="100"/>
</hostdev>
```

3. Create separate device configuration-XML files for the SCSI medium changer device, both connected to the virtual HBA 1.
   a. The first file configures SCSI device name 0:0:0:1, which is the path of SCSI LUN 1 via SCSI host 0.

```
<hostdev mode="subsystem" type="scsi">
     <source>
        <adapter name="scsi_host0"/>
        <address bus="0" target="0" unit="1"/>
     </source>
     <address type="scsi" controller="1" bus="0" target="0" unit="1"/>
</hostdev>
```

b. The second file configures SCSI device name 1:0:0:1, which is the path via SCSI host 1.

```
<hostdev mode="subsystem" type="scsi">
     <source>
        <adapter name="scsi_host1"/>
        <address bus="0" target="0" unit="1"/>
     </source>
     <address type="scsi" controller="1" bus="0" target="0" unit="101"/>
</hostdev>
```

Configuring a virtual SCSI-attached CD/DVD drive

The configuration of a virtual DVD drive as virtual SCSI device allows the virtual server to access various ISO images as virtual DVDs during its life cycle. You can replace a provided ISO image during virtual server operation.

Before you begin

You need a virtual HBA to connect to.

- Either use a configured virtual HBA (see "Configuring a virtual HBA" on page 88), or
- Connect to a new virtual HBA which will be automatically configured for you.

About this task

The virtual server accesses a virtual DVD as a virtual block device. You configure an ISO image, which represents the virtual DVD, and connect it through a controller as a virtual SCSI device. This allows the virtual server access to a virtual SCSI-attached CD/DVD drive, and to mount and unmount the file system which is contained on the currently provided virtual DVD.

You can remove the configured ISO image and provide a different one during the life cycle of the virtual server.

The virtual server can load it, and then reboot using the new ISO image.

Procedure

1. Configure the virtual DVD.
   a. Configure the ISO image, which represents the virtual DVD, as a file of type cdrom (see "<disk>" on page 207).

disk type attribute:	file
disk device attribute:	cdrom

   b. Specify the user space process that implements the virtual DVD (see "<driver> as child element of <disk>" on page 210).

driver name attribute:	qemu
driver io attribute:	native
driver type attribute:	raw
driver cache attribute:	none

   c. Specify the ISO image as virtual block device (see "<target> as child element of <disk>" on page 255).

target bus attribute:	scsi

   d. Specify the virtual DVD as read-only using the readonly element (see "<readonly>" on page 244).
2. Identify the ISO image on the host.

   Specify the fully qualified ISO image file name on the host (see "<source> as child element of <disk>" on page 249). If the virtual SCSI-attached CD/DVD drive is empty, omit this step.

source file attribute:	*<iso-image>*

3. Identify the virtual SCSI-attached CD/DVD drive on the virtual server.
   a. Specify a unique logical device name (see "<target> as child element of <disk>" on page 255).

target dev attribute:	*<logical-device-name>*

   Do not confuse the logical device name with its device name on the virtual server.
   b. Optional: Connect to a virtual HBA and specify a freely selectable SCSI device name on the virtual server.

address type attribute:	drive
address controller attribute:	*<controller-index>*
address bus attribute:	0
address target attribute:	*<target>*
address unit attribute:	*<unit>*

   (see "<address> as child element of <hostdev> or <disk>" on page 193)

   Where

   <controller-index>
   specifies the virtual HBA to which the SCSI device is connected.

   Enter the value of the controller index attribute of a configured virtual HBA or a new index value. The allocated index values must be contiguous without gaps. If you specify a new index value, a new virtual HBA is automatically configured.

   The virtual HBA is also called the *SCSI host* of the SCSI device on the virtual server.

   <target>
   is a freely selectable natural number: 0 ≤ *<target>* < 256

   <unit> determines the SCSI LUN on the virtual server according to the rules specified in the SCSI Architecture Model (SAM):

   0 ≤ *<unit>* < 256
   SCSI LUN := *<unit>*

   256 ≤ *<unit>* ≤ 16383
   SCSI LUN := 0x*<unit>* ∨ 0x4000

   Tip: Choose a value between 0 and 255, because these values are identically mapped to the SCSI LUN on the virtual server.

Example

```
<devices>
    ...
    <controller type="scsi" model="virtio-scsi" index="4"/>
    <disk type="file" device="cdrom">
        <driver name="qemu" type="raw" io="native" cache="none"/>
        <source file="/var/lib/libvirt/images/cd.iso"/>
        <target dev="sda" bus="scsi"/>
        <address type="drive" controller="4" bus="0" target="0" unit="0"/>
        <readonly/>
    </disk>
    ...
</devices>
```

Related tasks:

"Replacing a virtual DVD" on page 148
The virtual server accesses a provided ISO image as a virtual DVD through the virtual SCSI-attached CD/DVD drive. You can remove a virtual DVD, and provide a different one.

Configuring virtual Ethernet devices

Configure network interfaces, such as Ethernet interfaces, bonded interfaces, virtual LANs, or virtual switches as virtual Ethernet devices for a virtual server.

Before you begin

Provide network interfaces as described in Chapter 8, "Preparing network devices," on page 37.

Procedure

- To configure a MacVTap interface, follow the steps described in "Configuring a MacVTap interface."
- To configure a virtual switch, follow the steps described in "Configuring a virtual switch" on page 100

Configuring a MacVTap interface

Configure network interfaces, such as Ethernet interfaces, bonded interfaces, virtual LANs, through a direct MacVTap interface.

Procedure

You configure a network interface as direct MacVTap connection by using the interface element (see "<interface>" on page 220).
Libvirt automatically creates a MacVTap interface when you define the network device.

interface type attribute:	direct

By default, the virtual server cannot change its assigned MAC address and, as a result, cannot join multicast groups. To enable multicasting, you need set the interface trustGuestRxFilters attribute to yes. This has security implications, because it allows the virtual server to change its MAC address and thus to receive all frames delivered to this address.

1. Optional: Specify a freely selectable Media Access Control (MAC) address for the virtual server's virtual NIC.

mac address attribute:	*<MAC-address>*

   (see "<mac>" on page 227)

   If you do not specify the mac address attribute, libvirt assigns a MAC address to the interface.
2. Specify the host network interface.

   To allow virtual server migration to another host, ensure that an interface with the chosen name is configured on both the source and destination host.

source dev attribute:	*<interface-name>*
source mode attribute:	bridge

   (see "<source> as child element of <interface>" on page 252)
3. Specify the model type (see "<model> as a child element of <interface>" on page 233).

model type attribute:	virtio

Example

- To configure bonded interface bond0:

```
<interface type="direct">
      <source dev="bond0" mode="bridge"/>
      <model type="virtio"/>
</interface>
```

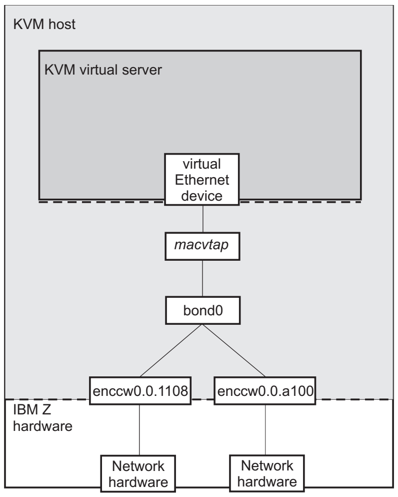


Figure 15. Direct interface type which configures a bonded interface

- To configure virtual LAN bond0.623:

```
<interface type="direct">
      <source dev="bond0.623" mode="bridge"/>
      <model type="virtio"/>
</interface>
```

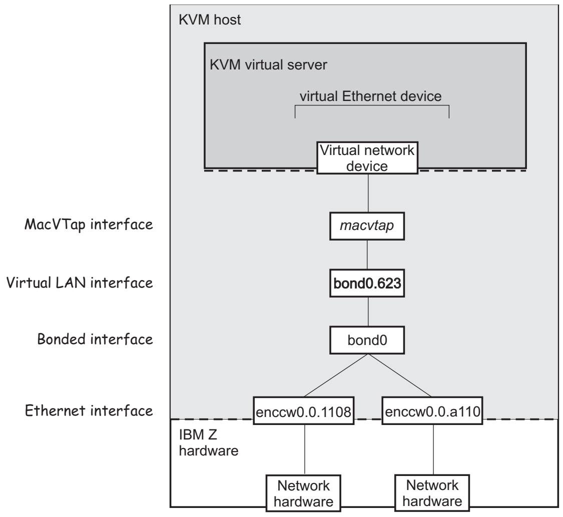


Figure 16. Direct interface type which configures a virtual LAN interface

Configuring a virtual switch

Configure virtual switches as virtual Ethernet devices.

Procedure

You configure a virtual switch by using the interface element (see “<interface>” on page 220).

interface type attribute:	bridge

1. Optional: Specify a freely selectable Media Access Control (MAC) address for the virtual server's virtual NIC.

mac address attribute:	*<MAC-address>*

   (see “<mac>” on page 227)
2. Specify the virtual switch that you created before as described in “Preparing a virtual switch” on page 43.

source bridge attribute:	*<vswitch>*

   (see “<source> as child element of <interface>” on page 252)
3. Specify the type.

virtualport type attribute:	openvswitch

   (see “<virtualport> as a child element of <interface>” on page 260)
4. Specify the model type.

model type attribute:	virtio

   (see “<model> as a child element of <interface>” on page 233)

Example

Display the available virtual switches:

```
# ovs-vsctl show
...
   Bridge "vswitch0"
        Port "vsbond0"
            Interface "enccw0.0.1108"
            Interface "enccw0.0.a112"
        Port "vswitch0"
            Interface "vswitch0"
                type: internal
...
```

Configure the virtual switch which is shown in Figure 13 on page 23:

```
<interface type="bridge">
      <source bridge="vswitch0"/>
      <virtualport type="openvswitch"/>
      <model type="virtio"/>
</interface>
```

After the creation and the start of the virtual server, the virtual switch is displayed as follows:

```
# ovs-vsctl show
...
   Bridge "vswitch0"
        Port "vnet0"
            Interface "vnet0"
        Port "vsbond0"
            Interface "enccw0.0.1108"
            Interface "enccw0.0.a112"
        Port "vswitch0"
            Interface "vswitch0"
                type: internal
...
```

Configuring a random number generator

Provide a virtual random number generator only if the host is equipped with a hardware random number generator, such as the secure IBM CCA coprocessor of a Crypto Express adapter.

Procedure

Use the rng element to configure a random number generator (see “<rng>” on page 245).

rng model attribute:	virtio

Use the backend element as child of the rng element to specify the device node of the input character device (see “<backend>” on page 195).
Currently, `/dev/random` is the only valid device node.

backend model attribute:	random
backend element:	*<device-node>*

Example

```
<devices>
    ...
    <rng model="virtio">
        <backend model="random">/dev/random</backend>
    </rng>
    ...
</devices>
```

Chapter 11. Configuring storage pools

A storage pool consists of a set of similar volumes. The storage pool volumes are backed by the image files of a directory, a disk, a partition, or a network file system, or by the logical volumes of a volume group.

Storage pool and volume configuration-XMLs

Configure storage pools with storage pool configuration-XML files, and configure storage pool volumes with volume configuration-XML files.

Storage pool

Root element
pool

Selected child elements
name, source, target

Example

```
<pool type="dir">
  <name>myPool</name>
  <target>
    <path>/var/lib/libvirt/images</path>
  </target>
</pool>
```

Storage pool volume

Root element
volume

Selected child elements
name, key, allocation, capacity

Example

```
<volume type="file">
  <name>federico.img</name>
  <key>/var/lib/libvirt/images/federico.img</key>
  <target>
    <path>/var/lib/libvirt/images/federico.img</path>
    <format type="qcow2"/>
  </target>
</volume>
```

Related reference:
"<pool>" on page 242
Is the root element of a storage pool configuration-XML.
"<volume>" on page 262
Is the root element of a volume configuration-XML.

Chapter 12. Configuring virtual networks

Use the network configuration-XML to configure virtual networks that connect KVM virtual servers among themselves and to an external network.

KVM hosts on IBM Z support networks with three types of Linux bridges. All types make a communication setup addressable as a network or bridge.

- Bridge with network address translation (NAT)
- Open vSwitch bridge
- Bridge with IP routing

Each bridge type has a different forwarding mode as specified with the <forward> element. Omitting the <forward> element results in a virtual network among the virtual servers, without a connection to a physical network.

Bridge with network address translation (NAT)

With network address translation, traffic of all virtual servers to the physical network is routed through the host's routing stack and uses the host's public IP address. This type of network supports outbound traffic only.

Forwarding mode
nat

Example

```
<network>
  <name>net0</name>
  <uuid>fec14861-35f0-4fd8-852b-5b70fdc112e3</uuid>
  <forward mode="nat">
    <nat>
      <port start="1024" end="65535"/>
    </nat>
  </forward>
  <bridge name="virbr0" stp="on" delay="0"/>
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

Open vSwitch bridge

With an Open vSwitch bridge, the switch implements a subnet. The <bridge> element must reference an already existing Open vSwitch (see "Preparing a virtual switch" on page 43).

Forwarding mode
bridge

Example

```
<network>
  <name>ovs</name>
  <uuid>58681f9f-20e1-4673-97a0-5c819660db3e</uuid>
  <forward mode="bridge"/>
  <bridge name="ovs-br0"/>
  <virtualport type="openvswitch"/>
</network>
```

Bridge with IP routing

Bridges with IP routing link to a virtual IP subnet on the host. Traffic to and from virtual servers that are connected to that subnet are then handled by the IP protocol.

Forwarding mode
route

Example

```
<network>
  <name>net1</name>
  <uuid>34fc97f4-86c5-4d65-887a-cc8b33d2a260</uuid>
  <forward mode="route"/>
  <bridge name="iedn" stp="off" delay="0"/>
  <mac address="f6:2b:85:a9:bf:d9"/>
  <ip address="198.51.100.1" netmask="255.255.255.0">
  </ip>
</network>
```

Related reference:

"<bridge>" on page 197
Configures the bridge device that is used to set up the virtual network.

"<dhcp>" on page 206
Configures DHCP services for the virtual network.

"<forward>" on page 215
Configures the forwarding mode for the bridge that connects the virtual network to a physical LAN. Omitting this tag results in an isolated network that can connect guests.

"<ip>" on page 222
Configures IP addresses for the virtual network.

"<name> as a child element of <network>" on page 235
Assigns a short name to a virtual network.

"<network>" on page 236
Is the root element of a network configuration-XML.

Part 4. Operation

Chapter 13. Creating, modifying, and deleting persistent virtual server definitions 109
Defining a virtual server. 110
Modifying a virtual server definition 110
Undefining a virtual server 111

Chapter 14. Managing the virtual server life cycle 113
Starting a virtual server 114
Terminating a virtual server 114
Suspending a virtual server. 116
Resuming a virtual server 116

Chapter 15. Monitoring virtual servers 119
Browsing virtual servers. 120
Displaying information about a virtual server . . 120
Displaying the current libvirt-internal configuration 122

Chapter 16. Migration 125
Definition of a virtual server on different hosts using the same configuration-XML 126
Live virtual server migration 127
Hypervisor release 127
| IBM Z hardware model 128
Live migration setup 129
Preservation of the virtual server resources 129
Host environments 132
Phases of a live migration 133
Performing a live migration 134

Chapter 17. Managing system resources . . . 137
Managing virtual CPUs 138
Modifying the number of virtual CPUs. . . . 138
Modifying the virtual CPU weight 141
Managing virtual memory 143

Chapter 18. Managing devices 145
Attaching a device. 146
Detaching a device 147
Replacing a virtual DVD 148
Connecting to the console of a virtual server . . . 149

Chapter 19. Managing storage pools. 151
Storage pool management commands 152
Volume management commands 153

| **Chapter 20. Managing virtual networks 155**

Manage the operation of virtual servers using virsh commands.

Chapter 13. Creating, modifying, and deleting persistent virtual server definitions

Pass a virtual server configuration to libvirt, modify the libvirt-internal configuration, or delete it.

Before you begin

- Ensure that the libvirt daemon is running on the host:

```
# systemctl status libvirtd
libvirtd.service - Virtualization daemon
Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled)
Active: active (running) since Thu 2015-04-16 10:55:29 CEST; 2 months 3 days ago
Docs: man:libvirtd(8)
http://libvirt.org
Main PID: 5615 (libvirtd)
CGroup: /system.slice/libvirtd.service
├─5615 /usr/sbin/libvirtd
├─6750 /sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro ...
└─6751 /sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro ...
```

  If the libvirt daemon is not running, enter:

```
# systemctl start libvirtd.service
```

- Ensure that a domain configuration-XML file, which configures the virtual server, is created.

About this task

1. To create a persistent virtual server definition, you pass its domain configuration-XML file to libvirt. From the domain configuration-XML file, libvirt creates a libvirt-internal configuration, which may differ from the domain configuration-XML. For example, libvirt generates a UUID or MAC addresses for virtual Ethernet devices, if they are not specified.

   See "Defining a virtual server" on page 110.
2. You can modify the libvirt-internal configuration without deleting the virtual server definition. Modifications come into effect with the next virtual server restart.

   See "Modifying a virtual server definition" on page 110.
3. When you delete the definition of a virtual server, libvirt destroys the libvirt-internal configuration. When you create a virtual server definition again, the generated values, such as UUID or MAC addresses, will differ from the previous ones.

   See "Undefining a virtual server" on page 111.

Related reference:

Chapter 29, "Selected virsh commands," on page 265
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Defining a virtual server

Create a persistent definition of a virtual server configuration.

Procedure

Define a virtual server to libvirt using the virsh **define** command (see "define" on page 273):

```
# virsh define <domain-configuration-XML-filename>
```

<domain-configuration-XML-filename>
is the path and file name of the domain configuration-XML file.

Results

libvirt creates a persistent virtual server definition and a libvirt-internal configuration. The name of the virtual server is the unique name specified in the domain configuration-XML file. The virtual server is in the state "shut off" with reason "unknown".

What to do next

To verify your definition, you may:

1. Browse all defined virtual servers (see "Browsing virtual servers" on page 120) by issuing:

   ```
   # virsh list --all
   ```

   Virtual servers that are defined but not yet started are listed with state "shut off".
2. Display the current libvirt-internal configuration as described in "Displaying the current libvirt-internal configuration" on page 122.
3. Start the virtual server as described in "Starting a virtual server" on page 114.
4. Check your connection to the virtual server via the configured console as described in "Connecting to the console of a virtual server" on page 149.

Related reference:
Chapter 27, "Virtual server life cycle," on page 183
Display the state of a defined virtual server including the reason with the virsh **domstate --reason** command.

Modifying a virtual server definition

Edit the libvirt-internal configuration of a defined virtual server.

About this task

Editing the libvirt-internal configuration modifies the virtual server definition persistently across host reboots. The modification is effective with the next virtual server restart.

Procedure

Modify the libvirt-internal configuration of a virtual server by using the virsh **edit** command (see "edit" on page 288):

```
# virsh edit <VS>
```

<VS> Is the name of the virtual server as specified in its domain configuration-XML file.

By default, the virsh **edit** command uses the vi editor. You can modify the editor by setting the environment variables $VISUAL or $EDITOR.

Results

If your configuration does not contain necessary elements, they will be inserted automatically when you quit the editor. Also, the virsh **edit** command does not allow to save and quit corrupted files.

The libvirt-internal configuration is modified and will be effective with the next virtual server restart.

What to do next

To make the modification of the configuration effective, you might want to terminate the virtual server and restart it afterwards (see "Terminating a virtual server" on page 114 and "Starting a virtual server" on page 114).

Undefining a virtual server

Delete the persistent libvirt definition of a virtual server.

Before you begin

- Ensure that the virtual server is in state "shut off".

  To view information about the current state of a virtual server, use the virsh **domstate** command.

Procedure

Delete the definition of a virtual server from libvirt by using the virsh **undefine** command (see "undefine" on page 339):

```
# virsh undefine <VS>
```

<VS> Is the name of the virtual server as specified in its domain configuration-XML file.

Chapter 14. Managing the virtual server life cycle

Use libvirt commands to start, terminate, suspend, or resume a defined virtual server.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 295) to verify whether the virtual server is defined:

  ```
  # virsh list --all
  ```

  If the virtual server is not displayed, see "Defining a virtual server" on page 110.

About this task

- "Starting a virtual server" on page 114

  Start a defined virtual server.
- "Terminating a virtual server" on page 114

  Properly shut down a virtual server, save a system image, or, if necessary, immediately terminate it.
- "Suspending a virtual server" on page 116

  Pause a virtual server.
- "Resuming a virtual server" on page 116

  Transfer a paused virtual server to the running state.

Related reference:

Chapter 27, "Virtual server life cycle," on page 183
Display the state of a defined virtual server including the reason with the virsh **domstate --reason** command.

Chapter 29, "Selected virsh commands," on page 265
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Starting a virtual server

Use the virsh **start** command to start a shut off virtual server.

About this task

When you start a virtual server, usually, an Initial Program Load (IPL) is performed, for example to boot the guest. But if there is a saved system image for the virtual server, the guest is restored from this system image. It depends on the command that terminated a virtual server whether the system image was saved or not (see "Terminating a virtual server").

The "saved shut off" state indicates the availability of a saved system image. To display the state and the reason of a virtual server, enter the command:

```
# virsh domstate <VS> --reason
shut off (saved)
```

where <*VS*> is the name of the virtual server.

Refer to Chapter 27, "Virtual server life cycle," on page 183 to see the effect of the virsh **start** command depending on the virtual server state.

Procedure

Start a defined virtual server in "shut off" state using the virsh **start** command (see "start" on page 336):

```
# virsh start <VS>
```

Using the `--console` option grants initial access to the virtual server console and displays all messages that are issued to the console:

```
# virsh start <VS> --console
```

<VS> Is the name of the virtual server as specified in its domain configuration-XML file.

If there is a saved system image, you can avoid that the virtual server is restored from this image by using the `--force-boot` option.

Terminating a virtual server

Terminate a running, paused, or crashed virtual server with or without saving its system image.

About this task

Refer to Chapter 27, "Virtual server life cycle," on page 183 to see the effect of the virsh commands to terminate a virtual server depending on its state.

Procedure

Description	Command	Comments
To properly terminate a virtual server:	"shutdown" on page 333	
To save a system image and terminate a virtual server properly:	"managedsave" on page 297	
To terminate a virtual server immediately:	"destroy" on page 274	Use the `--graceful` option to try to properly terminate the virtual server before terminating it forcefully.

- In most cases, you use the virsh **`shutdown`** command to properly terminate a virtual server.

  If the virtual server does not respond, it is not terminated. While the virtual server is shutting down, it traverses the state "in shutdown" and finally enters the "shutdown shut off" state.

  ```
  # virsh shutdown <VS>
  ```

  Example:

  To properly shut down virtual server vserv1, issue:

  ```
  # virsh shutdown vserv1
  Domain vserv1 is being shutdown
  ```

- Save the system image of a running or a paused virtual server and terminate it thereafter with the virsh **`managedsave`** command.

  ```
  # virsh managedsave <VS>
  ```

  Example:

  To save the system image of virtual server vserv2 and properly shut it down, issue:

  ```
  # virsh managedsave vserv2
  Domain vserv2 state saved by libvirt
  ```

  The system image of the virtual server is resumed at the time of the next start. Then, the state of the virtual server is either running or paused, depending on the last state of the virtual server and the **`managedsave`** command options.

  Note: The managedsave operation will save the virtual server state in a file in the host filesystem. This file has at least the size of the virtual server memory. Make sure the host filesystem has enough space to hold the virtual server state.

- When a virtual server is not responding, you can terminate it immediately with the virsh **`destroy`** command.

  The virtual server enters the "destroyed shut off" state. This command might cause a loss of data.

  ```
  # virsh destroy <VS>
  ```

The `--graceful` option tries to properly terminate the virtual server, and only if it is not responding in a reasonable amount of time, it is forcefully terminated:

```
# virsh destroy <VS> --graceful
```

Example:

To force a shutdown of virtual server vserv3, issue:

```
# virsh destroy vserv3
Domain vserv3 destroyed
```

<VS> Is the name of the virtual server as specified in its domain configuration-XML file.

Suspending a virtual server

Transfer a virtual server into the paused state.

Before you begin

Use the virsh **domstate** command to display the state of the virtual server.

About this task

Refer to Chapter 27, "Virtual server life cycle," on page 183 to see the effect of the virsh **suspend** command depending on the virtual server state.

Procedure

Suspend a virtual server by using the virsh **suspend** command (see "suspend" on page 338):

```
# virsh suspend <VS>
```

<VS> Is the name of the virtual server.

What to do next

To transfer the virtual server back to the running state, issue the virsh **resume** command.

Resuming a virtual server

Transfer a virtual server from the paused into the running state.

Before you begin

The virsh **list** command with the `--state-paused` option displays a list of paused virtual servers.

About this task

Refer to Chapter 27, "Virtual server life cycle," on page 183 to see the effect of the virsh **resume** command depending on the virtual server state.

Procedure

Resume a virtual server using the virsh **resume** command (see "resume" on page 331):

```
# virsh resume <VS>
```

<VS> Is the name of the virtual server.

Chapter 15. Monitoring virtual servers

Use libvirt commands to display information about a defined virtual server.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 295) to verify whether the virtual server is defined:

  ```
  # virsh list --all
  ```

  If the virtual server is not displayed, see "Defining a virtual server" on page 110.

About this task

- "Browsing virtual servers" on page 120

  View lists of all defined or of all running virtual servers.
- "Displaying information about a virtual server" on page 120

  View information about a virtual server, its state, its devices, or scheduling properties.
- "Displaying the current libvirt-internal configuration" on page 122

  The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, complemented with libvirt-internal information, and modified as devices are attached or detached.

Related reference:

Chapter 29, "Selected virsh commands," on page 265
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Browsing virtual servers

View lists of all defined or of all running virtual servers.

Procedure

- To view a list of all defined virtual servers, use the virsh **list** command with the `--all` option (see “list” on page 295):

```
# virsh list --all
```

- To view a list of all running or paused virtual servers, enter:

```
# virsh list
```

Example

View a list of all running or paused virtual servers:

```
# virsh list
Id    Name              State
----------------------------------

3     vserv1            paused
8     vserv2            running
```

Displaying information about a virtual server

View information about a virtual server, its state, its devices, or scheduling properties.

Procedure

You can display information about a defined virtual server using one of the following commands:

Displayed information	Command	Comments
General information	“dominfo” on page 283	
Current state	“domstate” on page 285	Display the reason of the current state by using the `--reason` option.
Scheduling information	“schedinfo” on page 332	
Number of virtual CPUs	“vcpucount” on page 340	
Virtual block devices	“domblkstat” on page 278	To retrieve the device name, use the virsh **domblklist** command.
Virtual Ethernet interfaces	“domifstat” on page 282	To retrieve the interface name, use the virsh **domiflist** command.
I/O threads	“iothreadinfo” on page 294	

Example

- View information about a defined virtual server:

```
# virsh dominfo vserv2
Id:             8
Name:           vserv2
UUID:           f4fbc391-717d-4c58-80d5-1cae505f89c8
OS Type:        hvm
State:          running
CPU(s):         4
CPU time:       164.6s
Max memory:     2097152 KiB
Used memory:    2097152 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c383,c682 (enforcing)
```

- View information about the current state:

```
# virsh domstate vserv2
running

# virsh domstate vserv2 --reason
running (unpaused)
```

- View scheduling information:

```
# virsh schedinfo vserv1
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
```

- Display the number of virtual CPUs:

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        3
current      live          3
```

- View information about the virtual block devices:

```
# virsh domblklist vserv1
Target     Source
------------------------------------------------
vda        /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc

# virsh domblkstat vserv1 /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc rd_req 17866
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc rd_bytes 180311040
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc wr_req 11896
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc wr_bytes 126107648
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc flush_operations 3884
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc rd_total_times 14496884715
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc wr_total_times 9834388979
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae000000000000023bc flush_total_times 755568088
```

- View information about the virtual Ethernet interfaces:

```
# virsh domiflist vserv1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    iedn       virtio      02:17:12:01:ff:01

# virsh domifstat vserv1 vnet0
vnet0 rx_bytes 2377970
vnet0 rx_packets 55653
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 831453
vnet0 tx_packets 18690
vnet0 tx_errs 0
vnet0 tx_drop 0
```

- View information about the I/O threads of a virtual server with 8 virtual CPUs:

```
# virsh iothreadinfo vserv1
  IOThread ID     CPU Affinity
  ---------------------------------------------------
   1               0-7
   2               0-7
   3               0-7
```

Displaying the current libvirt-internal configuration

The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, complemented with libvirt-internal information, and modified as devices are attached or detached.

Procedure

To display the current libvirt-internal configuration of a defined virtual server, use the virsh **dumpxml** command (see "dumpxml" on page 287):

```
# virsh dumpxml <VS>
```

<VS> Is the name of the virtual server as specified in its domain configuration-XML.

Example

Domain configuration-XML file vserv1.xml configures virtual server vserv1:

vserv1.xml

```
<domain type="kvm">
  <name>vserv1</name>
  <memory unit="GiB">4</memory>
  <vcpu>2</vcpu>
  <cputune>
    <shares>2048</shares>
  </cputune>

  <os>
    <type arch="s390x" machine="s390-ccw-virtio">hvm</type>
  </os>
  <iothreads>2</iothreads>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>preserve</on_crash>
  <devices>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
    </disk>
    <interface type="direct">
      <source dev="bond0" mode="bridge"/>
      <model type="virtio"/>
    </interface>
    <console type="pty">
      <target type="sclp"/>
    </console>
    <memballoon model="none"/>
  </devices>
</domain>
```

Device configuration-XML file `dev1.xml` configures a separate device:

dev1.xml

```
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
  <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
  <target dev="vdb" bus="virtio"/>
</disk>
```

You can define and start the virtual server and then attach the configured device with the commands:

```
# virsh define vserv1.xml
# virsh start vserv1 --console
# virsh attach-device vserv1 dev1.xml
```

The virsh **dumpxml** command displays the current libvirt-internal configuration, as for example:

```
# virsh dumpxml vserv1
<domain type="kvm">
  <name>quickstart1</name>
  <uuid>4a461da8-0253-4989-b267-bd4db02bfac4</uuid>
  <memory unit="KiB">4194304</memory>
  <currentMemory unit="KiB">4194304</currentMemory>
  <vcpu placement="static">2</vcpu>
  <iothreads>2</iothreads>
  <os>
    <type arch="s390x" machine="s390-ccw-virtio-2.10">hvm</type>
  </os>
  <clock offset="utc"/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>preserve</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-system-s390x</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0000"/>
    </disk>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
      <target dev="vdb" bus="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0002"/>
    </disk>
    <interface type="direct">
      <mac address="52:54:00:6a:0b:53"/>
      <source dev="bond0" mode="bridge"/>
      <model type="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0001"/>
    </interface>
    <console type="pty">
      <target type="sclp" port="0"/>
    </console>
    <memballoon model="none"/>
  </devices>
</domain>
```

libvirt added a number of XML elements to the current representation of the virtual server configuration. They are shown in **bold** typeface: a UUID, the current machine type which depends on the host setup and might be of the form "s390-virtio-ccw-<*x.x*>" as well, the emulator, mac address and address elements, and the attached device.

Chapter 16. Migration

Deploy virtual servers or ensure high availability during a hypervisor upgrade.

There are two ways to migrate a virtual server:

- “Definition of a virtual server on different hosts using the same configuration-XML” on page 126

  To deploy a virtual server to a different host, you can copy its domain configuration-XML file to the destination host, make any necessary adjustments, and finally define a virtual server on the basis of this configuration-XML.
- “Live virtual server migration” on page 127

  To provide high availability, for example during a hypervisor upgrade, you can migrate a running virtual server to a different host. You can decide whether or not the virtual server will be removed from the source host after the migration, so you can use live migration as a deployment vehicle, too.

  “Live” migration also provides means to migrate virtual servers offline. This option might be interesting for you if you want to deploy virtual servers and benefit from the migration automatism.

Migration to a different hypervisor release

The hypervisor release is defined by the installed QEMU release, by the hypervisor product or by your distribution on the host.

The virtual server's machine type determines which hypervisor release runs the virtual server on the host.

Be sure to configure the machine type with the alias value “s390-ccw-virtio” in the domain configuration-XML unless you intend to migrate the virtual server to a destination host with an earlier hypervisor release.

Definition of a virtual server on different hosts using the same configuration-XML

Deploy virtual servers by copying their domain configuration-XML files to different hosts and defining them afterwards.

When you define a virtual server using the alias machine type, libvirt replaces the alias machine type by the machine type which reflects the current hypervisor release of the host running the virtual server. The libvirt-internal configuration reflects the installed hypervisor release.

Example:

Domain configuration-XML using the alias machine type:

```
<type arch="s390x" machine="s390-ccw-virtio">hvm</type>
```

Libvirt-internal configuration for QEMU release 2.10:

```
<type arch="s390x" machine="s390-ccw-virtio-2.10">hvm</type>
```

Depending on your distribution, there may be additional machine types. The following command displays the available machine types:

```
# qemu-kvm --machine help
```

Figure 17 shows that creating virtual servers from the same domain

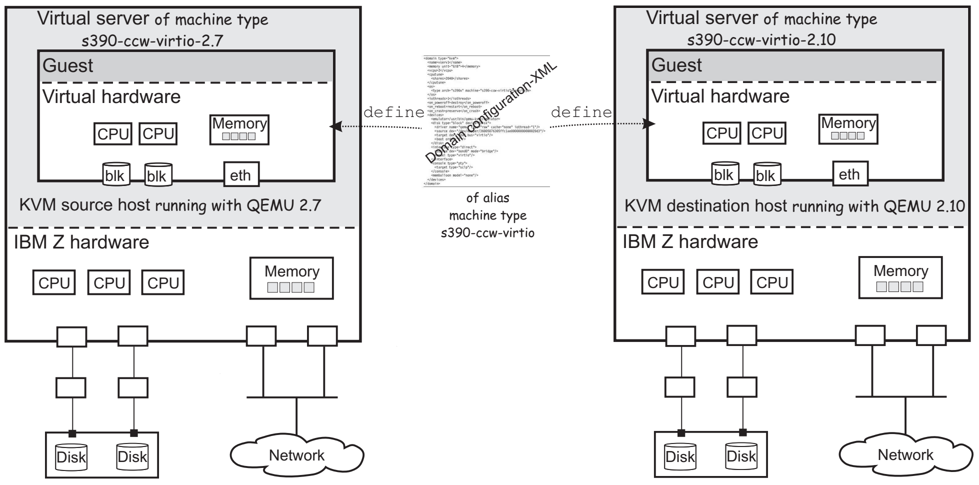

Virtual server of machine type s390-ccw-virtio-2.7
Guest
Virtual hardware
CPU
CPU
Memory
blk
blk
eth
KVM source host running with QEMU 2.7
IBM Z hardware
CPU
CPU
CPU
Memory
Disk
Disk
Network
define
Domain configuration-XML
of alias machine type s390-ccw-virtio
define
Virtual server of machine type s390-ccw-virtio-2.10
Guest
Virtual hardware
CPU
CPU
Memory
blk
blk
eth
KVM destination host running with QEMU 2.10
IBM Z hardware
CPU
CPU
CPU
Memory
Disk
Disk
Network

Figure 17. Defining virtual servers on different hosts

configuration-XML file on different hosts results in different machine types.

Live virtual server migration

Migrate a running virtual server from one host to another without affecting the virtual server. The literature also uses the terms "virtual server, virtual machine, or guest *relocation*".

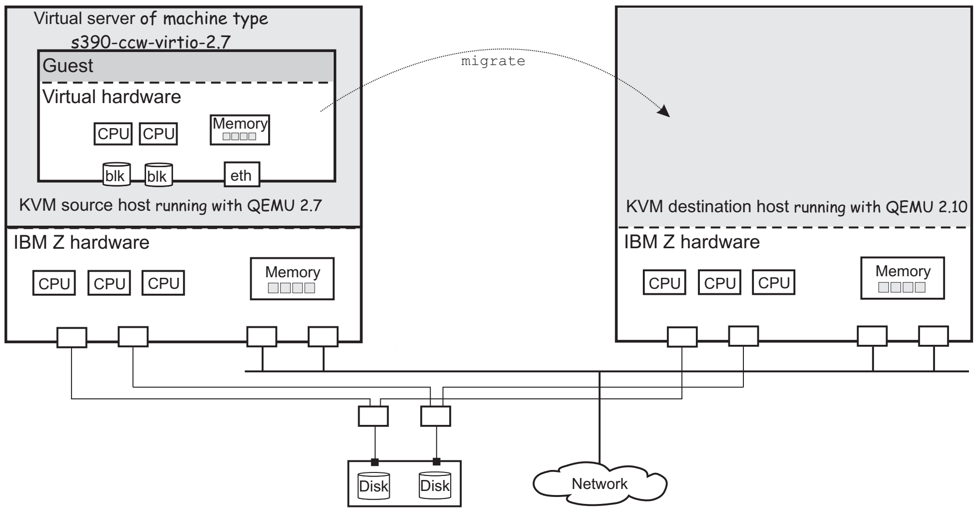
Hypervisor release

A live virtual server migration preserves the machine type of the virtual server. The libvirt-internal configuration is not changed, that is, the machine type still reflects the hypervisor release of the source host. Newer hypervisor releases are compatible with earlier versions.

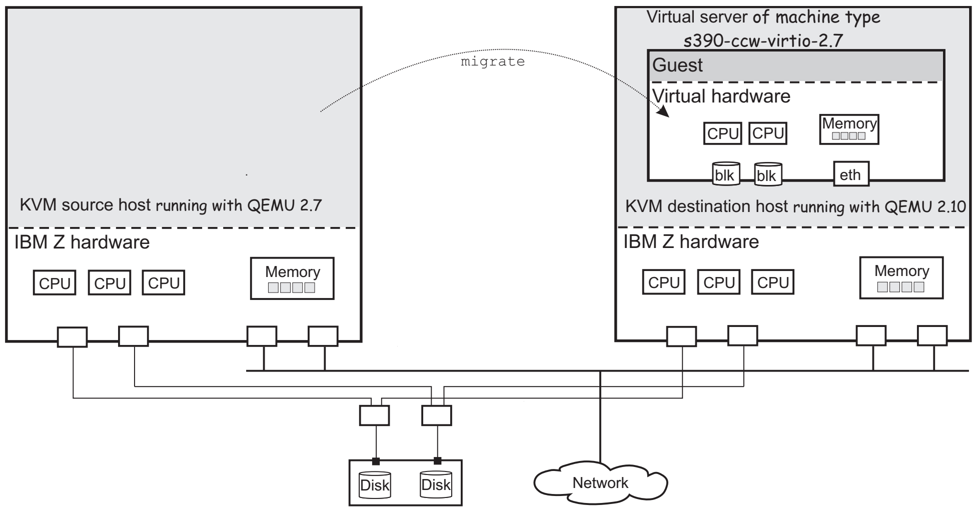
However, if you try to migrate a virtual server to a destination host with an earlier hypervisor release than the currently reflected machine type, you need to explicitly specify this earlier machine type in the virtual server definition before the migration.

Example:

1. Before the migration, the virtual server is running on the source host with a hypervisor release based on QEMU 2.7. The virtual server's machine type is s390-ccw-virtio-2.7.


Virtual server of machine type s390-ccw-virtio-2.7
Guest
Virtual hardware
CPU
CPU
Memory
blk
blk
eth
KVM source host running with QEMU 2.7
IBM Z hardware
CPU
CPU
CPU
Memory
migrate
KVM destination host running with QEMU 2.10
IBM Z hardware
CPU
CPU
CPU
Memory
Disk
Disk
Network

2. After the migration, the virtual server is running on the destination host with a hypervisor release based on QEMU 2.10. The virtual server's machine type is still s390-ccw-virtio-2.7.


migrate
Virtual server of machine type s390-ccw-virtio-2.7
Guest
Virtual hardware
CPU
CPU
Memory
blk
blk
eth
KVM source host running with QEMU 2.7
IBM Z hardware
CPU
CPU
CPU
Memory
KVM destination host running with QEMU 2.10
IBM Z hardware
CPU
CPU
CPU
Memory
Disk
Disk
Network

The virtual server runs on the earlier hypervisor release and does not exploit the features of the current release.

As long as you do not change the machine type to the new release, a migration of this virtual server back to its original source host will succeed.

IBM Z hardware model

The destination host must offer the same or a later CPU model than the CPU model that is used on the original host.

You can perform virtual server live migrations across IBM Z hardware of the same model and upgrade level. You can also migrate to a later hardware model, for example from an IBM z13 to an IBM z14 mainframe.

Migration to a prior hardware model is possible only if the virtual server on the newer hardware is restricted to CPU features that are also available on the older destination hardware. By default, a virtual server uses the latest CPU model of the hardware. Use the <cpu> element in the domain XML to configure a specific backlevel CPU model (see "Configuring the CPU model" on page 63).

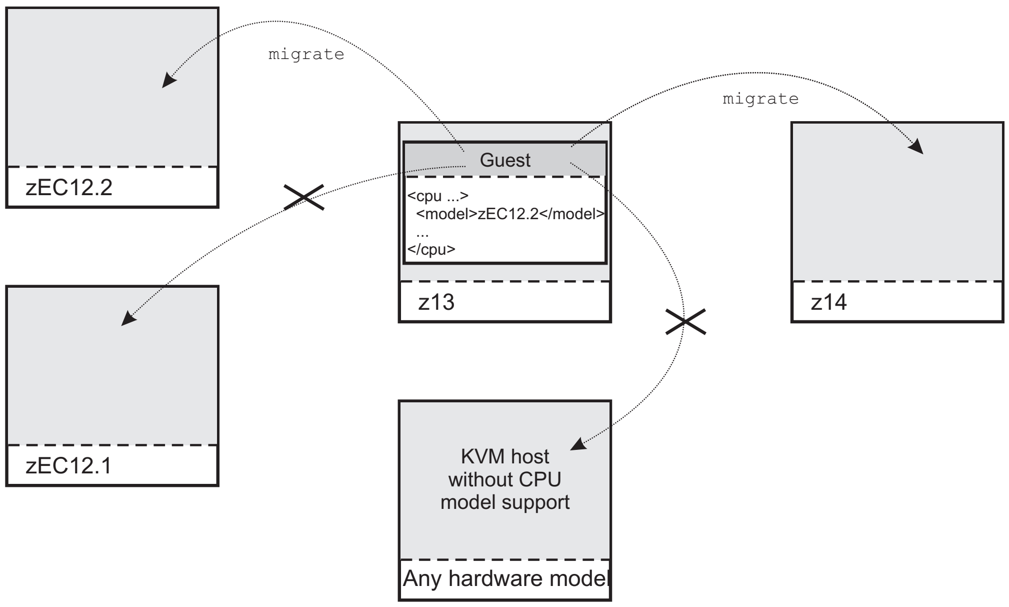
After a live migration to a later hardware model, the virtual server keeps running with the CPU model of the original hardware. This behavior preserves the option for a live migration back to the original hardware. To use new CPU features on the destination hardware, stop the virtual server, modify the domain configuration-XML, and then restart the virtual server.

Regardless of the destination hardware, live guest migration is possible only to KVM hosts with CPU model support.

If you cannot perform live guest migration, migrate by shutting down the virtual server and then starting it on the destination hardware (see "Definition of a virtual server on different hosts using the same configuration-XML" on page 126).

Example

The following figure illustrates the rules for a live migration. A virtual server on z13 hardware runs a guest operating system with the CPU features of zEC12 with upgrade level 2.



- Guest live migration is possible to z14 hardware.
- Guest live migration is possible to zEC12 hardware with upgrade level 2.
- Guest live migration is not possible to zEC12 hardware with upgrade level 1.

Live migration setup

To perform a live migration, the source and destination hosts must be connected and must have access to the same or equivalent system resources, the same storage devices and networks.

Preservation of the virtual server resources

Prepare a migration carefully to preserve the resources of the virtual server.

System resources

Provide access to the same or equivalent system resources, such as memory and CPUs, on both hosts.

Storage

Storage devices that are configured for the virtual server must be accessible from the destination host.

DASDs:

- Make sure that DASDs are configured using udev-created device nodes.
- If the DASDs are configured using the device bus-ID (by-path device node), make sure that you use identical device numbers in the IOCDS of both hosts.
- Make sure that there is a migration process for setting both the base devices and the alias devices online on the destination host.

SCSI disks:

- Make sure that SCSI disks are configured using device mapper-created device nodes.

Image files residing on a network file system (NFS):

- Make sure that both hosts have a shared access to the image files.

  If Security-Enhanced Linux (SELinux) is enabled on the destination host, using the following command can provide access to the NFS:

  ```
  # setsebool -P virt_use_nfs 1
  ```

  Please note that depending on the NFS configuration the image files could be accessible by other virtual servers.

Disk images residing on the host:
There are options to migrate image files that back up virtual block devices to the destination host. This process is called *disk migration.*

For each image file which is to be migrated:

- Make sure that the image file has write permission. That is, the virtual block device which is backed by the image file is not configured as a virtual DVD or by using the readonly element.

SCSI tapes or medium changer devices:

- When you migrate a virtual server that uses a configured virtual SCSI device, be aware that the SCSI device name, which is used to specify the source device, might change on the destination host.

  Tip: Make sure that SCSI tapes or medium changer devices are configured in separate device configuration-XML files. Detach them before you perform a migration. After the migration, reconfigure the devices before you reattach them.

"Disk device identification" on page 10 and "SCSI device identification" on page 18 explain various device nodes.

Networking

To ensure that the virtual server's network access is not interrupted by the migration:

- Make sure that the network administrator uses identical network interface names for the access to identical networks on both hosts.
- Make sure that the OSA channels are not shared between the source and the destination host.

Example

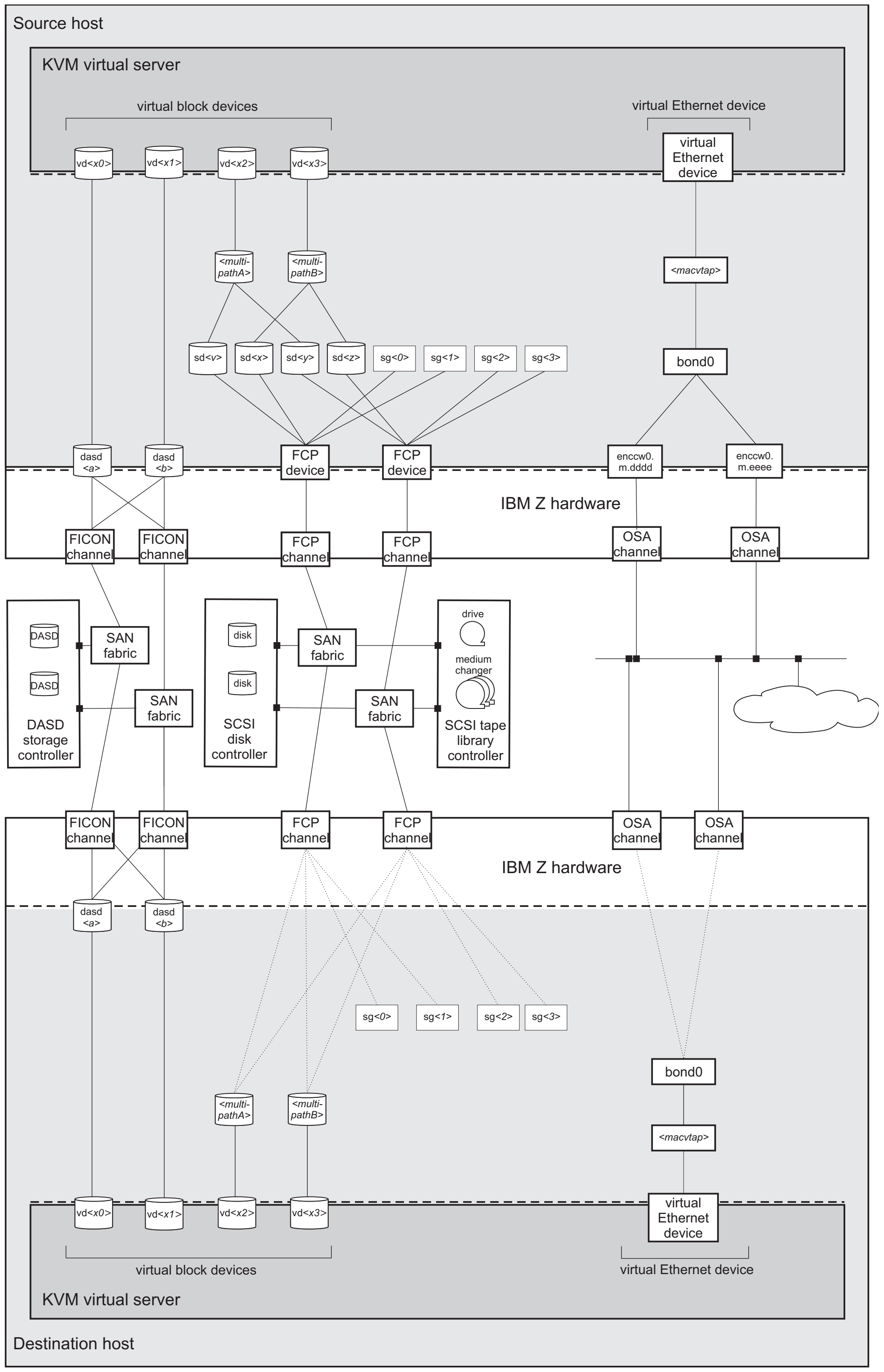


Figure 18. Example of a device setup on the source and destination hosts that allows the migration of the virtual server using these devices

Host environments

These settings and conditions on the involved hosts are relevant for a successful migration.

Concurrency

Maximum number of concurrent connections

If you connect to the destination host using ssh, increase the maximum number of unauthenticated concurrent connections to perform more than 10 concurrent migrations.

1. On the destination host, modify the OpenSSH SSH daemon configuration file `/etc/ssh/sshd_config`. The **`MaxStartups`** parameter specifies the maximum number of concurrent connections that have not yet been authenticated. The default is 10, which is specified as follows:

   ```
   #MaxStartups 10:30:100
   ```

   To allow a maximum number of 100 unauthenticated concurrent connections, change the **`MaxStartups`** parameter to:

   ```
   #MaxStartups 100
   ```

2. Restart the SSH daemon:

   ```
   [root@destination]# systemctl restart sshd.service
   ```

Migration port range

In a non-tunneled migration which has an URI of the form `qemu+ssh://<destination-host>/system`, each virtual server that is migrated uses a distinct destination port.

In addition, both tunneled and non-tunneled migrations use a separate destination port for each virtual disk that is to be migrated.

By default, libvirt uses the destination ports in the range from 49152 to 49215 for a migration. If you need more than 64 destination ports concurrently, increase the migration port range.

To allow for a backward migration, you might want to modify the migration port range of the source host, too.

To increase the migration port range:

- Change the **`migration_port_max`** parameter in `/etc/libvirt/qemu.conf` to a higher value than the default 49215.
- Make sure that the firewall configuration is changed to reflect the higher destination port number (see "Firewall configuration").

Firewall configuration

Make sure that the firewall configuration of the involved systems allows access to all required network resources.

Open the required migration port range in the firewall of the destination host. If you modified the migration port range which is used by libvirt, open the additional destination ports as well.

Example:

```
[root@destination]# firewall-cmd --zone=public --add-port=49152-49215/tcp \
--permanent
[root@destination]# firewall-cmd --reload
```

Deadlock prevention

Make sure that the migration is not blocked. In particular:

- Close all tape device nodes and unload online tape drives.
- A virtual server program should not be blocked by time-consuming or stalled I/O operations, such as rewinding a tape.

Performance considerations

In most cases, live virtual server migration does not directly affect the host system performance. However, it might have an impact if either the source system or the destination system is heavily loaded or constrained in the areas of CPU utilization, paging, or network bandwidth.

Phases of a live migration

The migration of a virtual server from a source to a destination host consists of two phases, the live phase and the stopped phase.

Live phase

While the virtual server is running, its memory pages are transferred to the destination host. During the live phase, the virtual server might continue to modify memory pages. These pages are called *dirty pages*, which must be retransmitted.

QEMU continuously estimates the time it will need to complete the migration during the stopped phase. If this estimated time is less than the specified maximum downtime for the virtual server, the virtual server enters the stopped phase of the migration.

If the virtual server changes memory pages faster than the host can transfer them to the destination, the migration command option `--auto-converge` can be used to throttle down the CPU time of the virtual server until the estimated downtime is less than the specified maximum downtime. If you do not specify this option, it might happen that the virtual server never enters the stopped phase because there are too many dirty pages to migrate.

This mechanism works for average virtual server workloads. Workloads that are very memory intensive might require the additional specification of the `--timeout` option. This option suspends the virtual server after a specified amount of time and avoids the situation where throttling down the CPU cannot catch up with the memory activity and thus, in the worst case, the migration operation never stops.

Stopped phase

During the stopped phase, the virtual server is paused. The host uses this downtime to transfer the rest of the dirty pages and the virtual server's system image to the destination.

If the virtual server makes use of storage keys, they are also migrated during this phase.

Performing a live migration

These commands are useful in the context of a live migration.

Procedure

1. Optional: You may specify a tolerable downtime for a virtual server during a migration operation by using the virsh **migrate-setmaxdowntime** command (see "migrate-setmaxdowntime" on page 304). The specified value is used to estimate the point in time when to enter the stopped phase.

   You can still issue this command during the process of a migration operation:

   ```
   # virsh migrate-setmaxdowntime <VS> <milliseconds>
   ```

2. Optional: You might want to limit the bandwidth that is provided for a migration.

   To set or to modify the maximum bandwidth, use the virsh **migrate-setspeed** command (see "migrate-setspeed" on page 305):

   ```
   # virsh migrate-setspeed <VS> --bandwidth <mebibyte-per-second>
   ```

   You can display the maximum bandwidth that is used during a migration with the virsh **migrate-getspeed** command (see "migrate-getspeed" on page 303):

   ```
   # virsh migrate-getspeed <VS>
   ```

3. To start a live migration of a virtual server, use the virsh **migrate** command with the `--live` option (see "migrate" on page 300):

   ```
   # virsh migrate --live <command-options> <VS> qemu+ssh://<destination-host>/system
   ```

   When virsh connects to the destination host via SSH, you will be prompted for a password. See libvirt.org/remote.html to avoid entering a password.

   <command-options>
   Are options of the virsh **migrate** command.

   <destination-host>
   Is the name of the destination host.

   <mebibyte-per-second>
   Is the migration bandwidth limit in MiB/s.

   <milliseconds>
   Is the number of milliseconds used to estimate the point in time when the virtual server enters the stopped phase.

   <VS> Is the name of the virtual server as specified in its domain configuration-XML file.

   a. Optional: The use of the `--auto-converge` and the `--timeout` options ensure that the migration operation completes.

   b. Optional: To avoid a loss of connectivity during a time-consuming migration process, increase the virsh keepalive interval (see Chapter 29, "Selected virsh commands," on page 265):

      ```
      # virsh --keepalive-interval <interval-in-seconds>
      ```

The use of the virsh `--keepalive-interval` and `--keepalive-count` options preserves the communication connection between the host that initiates the migration and the libvirtd service on the source host during time-consuming processes.

Use the keepalive options if:

- The virtual server is running a memory intensive workload, so that it might need to be suspended to complete the migration.
- You make use of an increased timeout interval.

Defaults:

keepalive interval	5 seconds
keepalive count	6

These defaults can be changed in `/etc/libvirt/libvirtd.conf`.

Example:

```
# virsh --keepalive-interval 10 migrate --live --persistent --undefinesource \
--timeout 1200 --verbose vserv1 qemu+ssh://kvmhost/system
```

This example increases the keepalive interval of the connection to the host to 10 seconds.

c. Optional: If the virtual server accesses virtual block devices that are backed by an image fileon the source host, these disks have to be migrated to the destination host (*disk migration*).

Specify the option `--copy-storage-all` or `--copy-storage-inc` in combination with the option `--migrate-disks` to copy image files that back up virtual block devices to the destination host.

Restriction:

- Disk migration is only possible for writable virtual disks.

  One example of a read-only disk is a virtual DVD. If in doubt, check your domain configuration-XML. If the disk device attribute of a disk element is configured as cdrom, or contains a readonly element, then the disk cannot be migrated.

Example:

This example copies the qcow2 image `/var/libvirt/images/vdd.qcow2` to the destination host, assuming that vdd is configured as follows:

```
<disk type="file" device="disk">
    <driver name="qemu" type="qcow2" io="native" cache="none"/>
    <source file="/var/lib/libvirt/images/vdd.qcow2"/>
    <target dev="vdd" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0004"/>
</disk>
```

1) Create a qcow2 image on the destination host:

```
[root@destination]# qemu-img create -f qcow2 \
/var/lib/libvirt/images/vdd.qcow2 1G
```

2) Issue the virsh **`migrate`** command on the source host:

```
[root@source]# virsh migrate --live --copy-storage-all --migrate-disks vdd \
vserv2 qemu+ssh://zhost/system
```

Results

The virtual server is not destroyed on the source host until it has been completely migrated to the destination host.

In the event of an error during migration, the resources on the destination host are cleaned up and the virtual server continues to run on the source host.

Example

- This example starts a live migration of the virtual server vserv3 to the destination host zhost. The virtual server will be transient on zhost, that is, after vserv3 is stopped on zhost, its definition will be deleted. After a successful migration, the virtual server will be destroyed on the source host, but still be defined.

  If the migration operation is not terminated within three hundred seconds, the virtual server is suspended while the migration continues.

  ```
  # virsh migrate --live --auto-converge --timeout 300 vserv3 qemu+ssh://zhost/system
  ```

- This example starts a live migration of vserv3 to the destination host zhost. After a successful migration, vserv3 will be destroyed and undefined on the source host. The virtual server definition will be persistent on the destination host.

  If the migration operation is not terminated within three hundred seconds, the virtual server is suspended while the migration continues.

  ```
  # virsh migrate --live --auto-converge --timeout 300 --undefinesource --persistent \
  vserv3 qemu+ssh://zhost/system
  ```

What to do next

- You can verify whether the migration completed successfully by looking for a running status of the virtual server on the destination, for example by using the virsh **list** command:

  ```
  # virsh list

  Id Name                State
  ----------------------------------
  10 kvm1               running
  ```

- You can cancel an ongoing migration operation by using the virsh **domjobabort** command:

  ```
  # virsh domjobabort <VS>
  ```

Chapter 17. Managing system resources

Use libvirt commands to manage the system resources of a defined virtual server, such as virtual CPUs.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 295) to verify whether the virtual server is defined:

  ```
  # virsh list --all
  ```

  If the virtual server is not displayed, see "Defining a virtual server" on page 110.

About this task

- "Managing virtual CPUs" on page 138

  Modify the portion of the run time that is assigned to the CPUs of a defined virtual server.
- "Managing virtual memory" on page 143

  Restrict the amount of physical memory used by a virtual server.

Related reference:

Chapter 29, "Selected virsh commands," on page 265
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Managing virtual CPUs

Modify the number of virtual CPUs and the portion of the run time that is assigned to the virtual CPUs of a defined virtual server.

About this task

- "Modifying the number of virtual CPUs"

  describes how to modify the number of virtual CPUs of a running virtual server.
- "Modifying the virtual CPU weight" on page 141

  describes how to modify the portion of the run time that is assigned to the virtual server CPUs.

Related concepts:

Chapter 21, "CPU management," on page 159
Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks:

"Configuring virtual CPUs" on page 61
Configure virtual CPUs for a virtual server.

Modifying the number of virtual CPUs

Modify the number of virtual CPUs or the maximum number of available virtual CPUs for a defined virtual server.

About this task

The number of virtual CPUs that you can assign to a virtual server is limited by the maximum number of available virtual CPUs. Both numbers are configured with the vcpu element and can be modified during operation.

To display the number of virtual CPUs, use the virsh **vcpucount** command. For example, issue:

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        3
current      live          3
```

where

maximum config
: Specifies the maximum number of virtual CPUs that can be made available for the virtual server after the next restart.

maximum live
: Specifies the maximum number of virtual CPUs that can be made available for the running or paused virtual server.

current config
: Specifies the actual number of virtual CPUs which will be available for the virtual server with the next restart.

current live
: Specifies the actual number of virtual CPUs which are available for the running or paused virtual server.

You can modify the following values:

maximum config
: The maximum value can be modified only in combination with a virtual server restart.

  The maximum number of available virtual CPUs is not limited. If no value is specified, the maximum number of available virtual CPUs is 1.

current config
: The current value can be modified in combination with a virtual server restart. It is limited by the maximum number of available virtual CPUs. Consider to set the surplus virtual CPUs offline until the next restart.

current live
: You can increase the actual number of virtual CPUs for a running or paused virtual server. This number is limited by the maximum number of available CPUs.

  Additional virtual CPUs are provided in the halted state. Depending on the guest setup, the virtual server user has to bring them online.

Procedure

Use the virsh **setvcpus** command to modify the number of virtual CPUs or the maximum number of available virtual CPUs for a defined virtual server (see “setvcpus” on page 334).

- Modify **maximum config**:

  To modify the maximum number of available virtual CPUs with the next virtual server restart, use the `--maximum` and the `--config` options:

  ```
  # virsh setvcpus <VS> <max-number-of-CPUs> --maximum --config
  ```

  This modification takes effect after the termination of the virtual server and a subsequent restart. Please note that a virtual server reboot does not modify the libvirt-internal configuration.
- Modify **current config:**

  To increase or reduce the number of virtual CPUs with the next virtual server restart, use the `--config` option:

  ```
  # virsh setvcpus <VS> <number-of-CPUs> --config
  ```

  The virtual CPUs are not removed until the next virtual server reboot. Until then, the virtual server user might set the corresponding number of virtual CPUs offline.
- Modify **current live:**

  To increase the number of virtual CPUs of a running or paused virtual server, use the `--live` option:

  ```
  # virsh setvcpus <VS> <number-of-CPUs> --live
  ```

  The virtual server user has to bring the additional virtual CPUs online.

  <VS> Is the name of the virtual server as specified in its domain configuration-XML file.

<max-number-of-CPUs>
: Is the maximum number of available virtual CPUs for the virtual server after the next restart.

<number-of-CPUs>
: Is the number of virtual CPUs assigned to the virtual server.

Example

- Change the maximum number of available virtual CPUs with the next virtual server restart.

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        4
current      live          4

# virsh setvcpus vserv1 6 --maximum --config

# virsh vcpucount vserv1
maximum      config        6
maximum      live          5
current      config        4
current      live          4
```

- You cannot remove virtual CPUs from a running virtual server.
  1. This example removes two virtual CPUs from the virtual server vserv1 with the next virtual server restart:

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        4
current      live          4

# virsh setvcpus vserv1 2 --config

# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        2
current      live          4
```

  2. To set the CPUs offline until the next virtual server restart, the virtual server user might set the virtual CPUs offline:

```
[root@guest:] # chcpu -d 2
CPU 2 disabled
[root@guest:] # chcpu -d 3
CPU 3 disabled
```

- Add virtual CPUs to a running virtual server.
  1. This example adds a virtual CPU to the virtual server vserv1:

```
# virsh vcpucount vserv1
maximum      config         5
maximum      live           5
current      config         3
current      live           3

# virsh setvcpus vserv1 4 --live

# virsh vcpucount vserv1
maximum      config         5
maximum      live           5
current      config         3
current      live           4
```

2. To set the additional CPU online, the virtual server user might enter:

```
[root@guest:] # chcpu -e 3
CPU 3 enabled
```

Modifying the virtual CPU weight

Modify the share of run time that is assigned to a virtual server.

About this task

The available CPU time is shared between the running virtual servers. Each virtual server receives the share that is configured with the shares element, or the default value.

To display the current CPU weight of a virtual server, enter:

```
# virsh schedinfo <VS>
```

You can modify this share for a running virtual server or persistently across virtual server restarts.

Procedure

- To modify the current CPU weight of a running virtual server, use the virsh **schedinfo** command with the `--live` option (see “schedinfo” on page 332):

```
# virsh schedinfo <VS> --live cpu_shares=<number>
```

- To modify the CPU weight in the libvirt-internal configuration of the virtual server, which will persistently affect the CPU weight beginning with the next restart, use the `--config` option:

```
# virsh schedinfo <VS> --config cpu_shares=<number>
```

<number>
Specifies the CPU weight.

<VS> Is the name of the virtual server.

Example

- A virtual server with a CPU weight of 2048 receives twice as much run time as a virtual server with a CPU weight of 1024.

- The following example modifies the CPU weight of vserv1 to 2048 while it is running:

```
virsh schedinfo vserv1 --live cpu_shares=2048
Scheduler : posix
cpu_shares : 2048
vcpu_period : 100000
vcpu_quota : -1
emulator_period: 100000
emulator_quota : -1
```

- The following example changes the libvirt-internal configuration, which will persistently affect the CPU weight, beginning with the next restart of vserv1.

```
virsh schedinfo vserv1 --config cpu_shares=2048
Scheduler : posix
cpu_shares : 2048
vcpu_period : 0
vcpu_quota : 0
emulator_period: 0
emulator_quota : 0
```

Related tasks:

"Tuning virtual CPUs" on page 62
Regardless of the number of its virtual CPUs, the CPU weight determines the shares of CPU time which is dedicated to a virtual server.

Managing virtual memory

Specify a soft limit for the amount of physical host memory used by a virtual server.

Procedure

Specify a soft limit for physical host memory usage with the virsh **memtune** command (see "memtune" on page 299):

```
# virsh memtune <VS> --soft-limit <limit-in-KB>
```

<limit-in-KB>
Specifies the soft limit in kilobytes.

<VS> Is the name of the virtual server as defined in the domain configuration-XML file.

Related concepts:

Chapter 22, "Memory management," on page 163
The memory configured for a virtual server appears as physical memory to the guest operating system but is realized as a Linux virtual address space.

Related tasks:

"Tuning virtual memory" on page 65
A configured soft limit allows the host to limit the physical host memory resources used for the virtual server memory in case the host experiences high swapping activity.

Chapter 18. Managing devices

Add, remove, or access devices of a running virtual server.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 295) to verify whether the virtual server is defined:

  ```
  # virsh list --all
  ```

  If the virtual server is not displayed, see "Defining a virtual server" on page 110.

About this task

- "Attaching a device" on page 146

  Attach a device to a virtual server. If the virtual server is running, you can hotplug the device.
- "Detaching a device" on page 147

  Detach a device from a virtual server. If the virtual server is running, you can unplug the device.
- "Replacing a virtual DVD" on page 148

  Remove the currently provided ISO image, or provide a different one.
- "Connecting to the console of a virtual server" on page 149

  Connect to the console of a virtual server.

Related reference:

Chapter 29, "Selected virsh commands," on page 265
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Attaching a device

You can hotplug devices to a running virtual server, add devices to the persistent virtual server configuration, or both.

Before you begin

- Ensure that the new device is not already assigned to the virtual server.

  To list the devices that are assigned to a virtual server, you can
  - Display the current libvirt-internal configuration.
  - Use the virsh **`domblklist`** command to display a list of currently assigned block devices or the virsh **`domiflist`** command to display a list of currently assigned interface devices.
- You need a device configuration-XML file for the device.

Procedure

1. Optional: If you attach a virtual block device, and the current libvirt-internal configuration does not provide an I/O thread for the device:

   Add an I/O thread dedicated to the device by using the virsh **`iothreadadd`** command (see "iothreadadd" on page 290):

   ```
   # virsh iothreadadd <VS> <IOthread-ID>
   ```

   <VS>
   Is the name of the virtual server as defined in the domain configuration-XML file.

   <IOthread-ID>
   Is the ID of the I/O thread to be added to the virtual server. Be sure that the I/O thread ID matches the I/O thread ID in the device configuration-XML.
2. Attach the device using the virsh **`attach-device`** command (see "attach-device" on page 268).

   ```
   # virsh attach-device <VS> <device-configuration-XML-filename> <scope>
   ```

   <device-configuration-XML-filename>
   Is the name of the device configuration-XML file.

   <VS>
   Is the name of the virtual server as defined in the domain configuration-XML file.

   <scope>
   Specifies the scope of the command:

   `--live`
   Hotplugs the device to a running virtual server. This configuration change does not persist across stopping and starting the virtual server.

   `--config`
   Adds the device to the persistent virtual server configuration. The device becomes available when the virtual server is next started. This configuration change persists across stopping and starting the virtual server.

   `--persistent`
   Adds the device to the persistent virtual server configuration and

hotplugs it if the virtual server is running. This configuration change persists across stopping and starting the virtual server. This option is equivalent to specifying both **--live** and **--config**.

Related concepts:

"I/O threads" on page 167
I/O threads are dedicated to perform I/O operations on virtual block devices.

Related tasks:

Chapter 10, "Configuring devices," on page 75
When you configure storage and network devices, you specify the physical hardware on which the resources are based.

"Displaying the current libvirt-internal configuration" on page 122
The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, complemented with libvirt-internal information, and modified as devices are attached or detached.

Detaching a device

You can unplug devices from a running virtual server, remove devices from the persistent virtual server configuration, or both.

Before you begin

You need a device configuration-XML file to detach a device from a virtual server. If the device has previously been attached to the virtual server, use the device configuration-XML file that was used to attach the device.

Procedure

1. Detach the device using the virsh **detach-device** command (see "detach-device" on page 275):

```
# virsh detach-device <VS> <device-configuration-XML-filename> <scope>
```

<device-configuration-XML-filename>
Is the name of the device configuration-XML file.

<VS>
Is the name of the virtual server as defined in the domain configuration-XML file.

<scope>
Specifies the scope of the command:

--live
Unplugs the device from a running virtual server. This configuration change does not persist across stopping and starting the virtual server.

--config
Removes the device from the persistent virtual server configuration. The device becomes unavailable when the virtual server is next started. This configuration change persists across stopping and starting the virtual server.

--persistent
Removes the device from the persistent virtual server configuration and unplugs it if the virtual server is running. This configuration change persists across stopping and starting the virtual server. This option is equivalent to specifying both **--live** and **--config**.

2. Optional: If you detach a virtual block device, you might want to remove the I/O thread which is dedicated to the device.

   The virsh **iothreadinfo** command displays the I/O threads that are available for a virtual server.

   Use the virsh **iothreaddel** command to remove an I/O thread (see "iothreaddel" on page 292):

   ```
   # virsh iothreaddel <VS> <IOthread-ID>
   ```

   <VS>
   Is the name of the virtual server as defined in the domain configuration-XML file.

   <IOthread-ID>
   Is the ID of the I/O thread to be deleted from the virtual server.

Replacing a virtual DVD

The virtual server accesses a provided ISO image as a virtual DVD through the virtual SCSI-attached CD/DVD drive. You can remove a virtual DVD, and provide a different one.

Before you begin

Make sure that the virtual DVD drive is configured as a virtual SCSI device (see "Configuring a virtual SCSI-attached CD/DVD drive" on page 95).

About this task

The guest is able to mount and to unmount the file system residing on a virtual DVD. You can remove the ISO image which represents the virtual DVD and provide a different one during the life time of the virtual server. If you try to remove an ISO image that is still in use by the guest, QEMU forces the guest to release the file system.

Procedure

1. Optional: Remove the current ISO image by using the virsh **change-media** command with the `--eject` option (see "change-media" on page 270):

   ```
   # virsh change-media <VS> <logical-device-name> --eject
   ```

2. Provide a different ISO image by using the virsh **change-media** command with the `--insert` option:

   ```
   # virsh change-media <VS> <logical-device-name> --insert <iso-image>
   ```

   In case the current ISO image has not been removed before, it is replaced by the new one.

   <iso-image>
   Is the fully qualified path to the ISO image on the host.

   <logical-device-name>
   Identifies the virtual SCSI-attached CD/DVD drive by its logical device name, which was specified with the target dev attribute in the domain configuration-XML file.

<*VS*> Is the name of the virtual server as defined in the domain configuration-XML file.

Example

After the guest has unmounted the file system on the virtual DVD, this example removes the currently provided virtual DVD from the virtual DVD drive:

```
# virsh domblklist vserv1
Target     Source
------------------------------------------------
vda        /dev/storage1/vs1_disk1
sda        /var/lib/libvirt/images/cd2.iso

# virsh change-media vserv1 sda --eject
Successfully ejected media.

# virsh domblklist vserv1
Target     Source
------------------------------------------------
vda        /dev/storage1/vs1_disk1
sda        -
```

If the virtual DVD is still in use by the guest, the **change-media** command with the `--eject` option forces the guest to unmount the file system.

This example inserts a virtual DVD, which is represented by the ISO image, into a virtual DVD drive:

```
# virsh change-media vserv1 sda --insert /var/lib/libvirt/images/cd2.iso
Successfully inserted media.
```

Connecting to the console of a virtual server

Open a console when you start a virtual server, or connect to the console of a running virtual server.

Procedure

Connect to a pty console of a running virtual server by using the virsh **console** command (see “console” on page 272):

```
# virsh console <VS>
```

However, if you want to be sure that you do not miss any console message, connect to the console when you start a virtual server by using the `--console` option (see “start” on page 336):

```
# virsh start <VS> --console
```

What to do next

To leave the console, press Control and Right bracket (Ctrl+]) when using the US keyboard layout.

Related tasks:

“Starting a virtual server” on page 114
Use the virsh **start** command to start a shut off virtual server.

"Configuring the console" on page 70
Configure the console by using the console element.

Chapter 19. Managing storage pools

Before you begin

1. Configure one or more storage pools as described in Chapter 11, "Configuring storage pools," on page 103.
2. For each storage pool, configure a set of volumes. Alternatively, you can create volumes by using virsh commands.

About this task

Once a storage pool is defined to libvirt, it can enter the states "inactive", "active", or "destroyed".

Figure 19 shows the state-transition diagram of a storage pool:

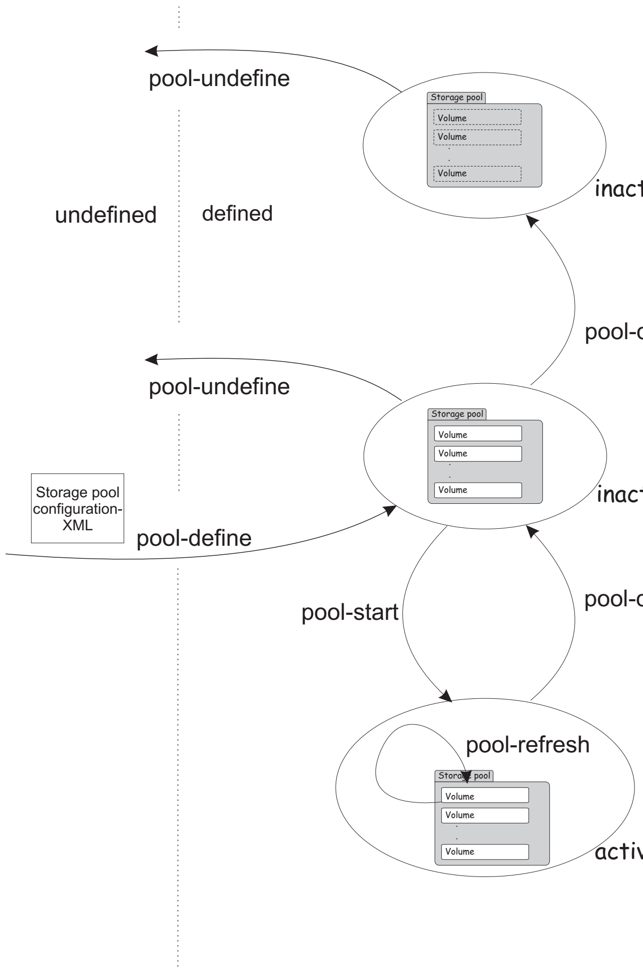


Figure 19. Storage pool state-transition diagram

There are virsh commands to:

- Create a persistent storage pool definition, modify the definition, and delete the storage pool definition
- Manage the storage pool life cycle

- Monitor a storage pool

These commands are described in "Storage pool management commands."

There are also virsh commands to manage the volumes of a storage pool. Use the commands described in "Volume management commands" on page 153.

Storage pool management commands

Before you begin

- Ensure that the libvirt daemon is running on the host.

About this task

Procedure

- Creating, modifying, and deleting a persistent storage pool definition:

Functionality	Command
Create a persistent definition of a storage pool configuration	"pool-define" on page 318
Edit the libvirt-internal configuration of a defined storage pool	"pool-edit" on page 322
Delete the persistent libvirt definition of a storage pool	"pool-undefine" on page 328

- Managing the storage pool life cycle:

Functionality	Command
Enable or disable the automatic start of a storage pool when the libvirt daemon is started	"pool-autostart" on page 317
Start a defined inactive storage pool	"pool-start" on page 327
Update the volume list of a storage pool	"pool-refresh" on page 326
Shut down a storage pool - the pool can be restarted by using the virsh **`pool-start`** command	"pool-destroy" on page 320
Delete the volumes of a storage pool **Attention:** This command is intended for expert users. Depending on the pool type, the results range from no effect to loss of data. In particular, data is lost when a zfs or LVM group pool is deleted.	"pool-delete" on page 319

- Monitoring storage pools:

Functionality	Command
View a list of all defined storage pools	"pool-list" on page 324
Display the current libvirt-internal configuration of a storage pool	"pool-dumpxml" on page 321
Display information about a defined storage pool	"pool-info" on page 323
Retrieve the name of a storage pool from its UUID	"pool-name" on page 325
Retrieve the UUID of a storage pool from its name	"pool-uuid" on page 329

Volume management commands

Before you begin

- Ensure that the libvirt daemon is running on the host.

Procedure

- Creating, modifying, and deleting volumes:

Functionality	Command
Create a volume for a storage pool from a volume configuration-XML file	“vol-create” on page 341
Remove a volume from a storage pool	“vol-delete” on page 342

  Other useful commands to create volumes are: vol-create-as, vol-create-from, or vol-clone.

- Monitoring volumes:

Functionality	Command
Display a list of the volumes of a storage pool	“vol-list” on page 346
Display the current libvirt-internal configuration of a storage volume	“vol-dumpxml” on page 343
Display information about a defined volume	“vol-info” on page 344
Display the key of a volume from its name or path	“vol-key” on page 345
Display the name of a volume from its key or path	“vol-name” on page 347
Display the path of a volume from its name or key	“vol-path” on page 348
Display the storage pool name or UUID which hosts the volume	“vol-pool” on page 349

Chapter 20. Managing virtual networks

Use **`virsh`** commands to manage virtual networks.

Before you begin

KVM hosts provide a preconfigured virtual network named `default`. You can configure more virtual networks as described in Chapter 12, "Configuring virtual networks," on page 105.

About this task

Virtual networks that are defined to libvirt can be in one of the states "inactive" or "active".

The following figure shows the state-transition diagram of a virtual network:

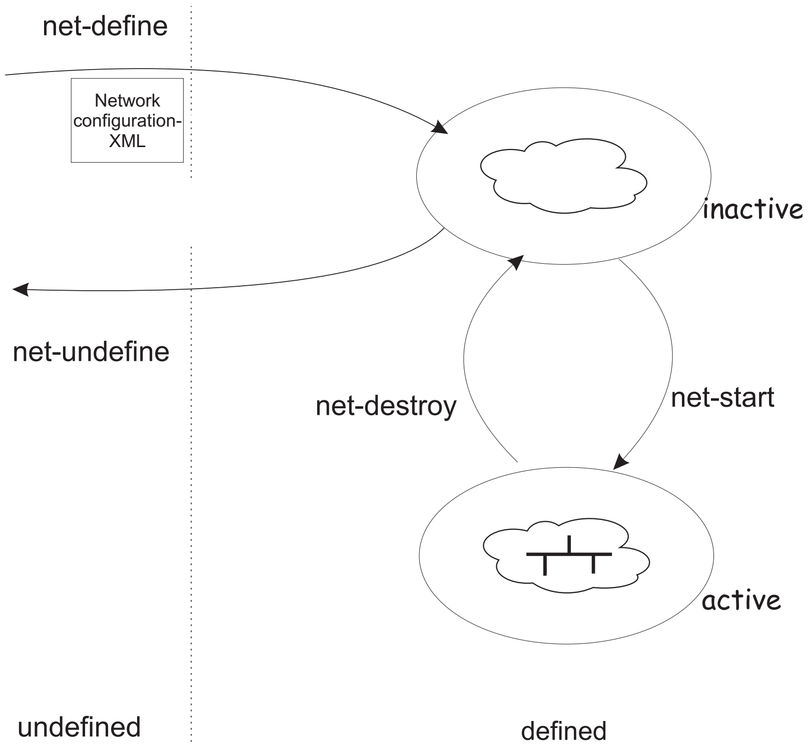


Figure 20. Virtual network state-transition diagram

Procedure

- Creating, modifying, and deleting a persistent network definition:

Task	Command
Create a persistent definition of a virtual network configuration.	"net-define" on page 307
Edit the libvirt-internal configuration of a defined virtual network.	"net-edit" on page 310

Task	Command
Delete the persistent libvirt definition of a virtual network.	“net-undefine” on page 315

- Managing the virtual network lifecycle:

Task	Command
Enable or disable the automatic activation of a virtual network when the libvirt daemon is started.	“net-autostart” on page 306
Activate a defined, inactive virtual network.	“net-start” on page 314
Deactivate a virtual network.	“net-destroy” on page 308

- Monitoring networks:

Task	Command
View a list of all defined virtual networks.	“net-list” on page 312
Display the current configuration of a virtual network.	“net-dumpxml” on page 309
Display information about a defined virtual network.	“net-info” on page 311
Retrieve the name of a virtual network from its UUID.	“net-name” on page 313
Retrieve the UUID of a virtual network from its name.	“net-uuid” on page 316

Part 5. Best practices and performance considerations

Chapter 21. CPU management 159
Linux scheduling 159
CPU weight 160

Chapter 22. Memory management. 163

Chapter 23. Storage management 167
I/O threads 167
Logical volume management 167

Avoid common pitfalls and tune the virtual server.

Chapter 21. CPU management

Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks:

"Configuring virtual CPUs" on page 61
Configure virtual CPUs for a virtual server.

"Managing virtual CPUs" on page 138
Modify the number of virtual CPUs and the portion of the run time that is assigned to the virtual CPUs of a defined virtual server.

Linux scheduling

Based on the hardware layout of the physical cores, the Linux scheduler maintains hierarchically ordered *scheduling domains*.

Basic scheduling domains consist of those processes that are run on physically adjacent cores, such as the cores on the same chip. Higher level scheduling domains group physically adjacent scheduling domains, such as the chips on the same book.

The Linux scheduler is a multi-queue scheduler, which means that for each of the logical host CPUs, there is a *run queue* of processes waiting for this CPU. Each virtual CPU waits for its execution in one of these run queues.

Moving a virtual CPU from one run queue to another is called a *(CPU) migration*. Be sure not to confuse the term "CPU migration" with a "live migration", which is the migration of a virtual server from one host to another. The Linux scheduler might decide to migrate a virtual CPU when the estimated wait time until the virtual CPU will be executed is too long, the run queue where it is supposed to be waiting is full, or another run queue is empty and needs to be filled up.

Migrating a virtual CPU within the same scheduling domain is less cost intensive than to a different scheduling domain because of the caches being moved from one core to another. The Linux scheduler has detailed information about the *migration costs* between different scheduling domains or CPUs. Migration costs are an important factor for the decision if the migration of a virtual CPU to another host CPU is valuable.

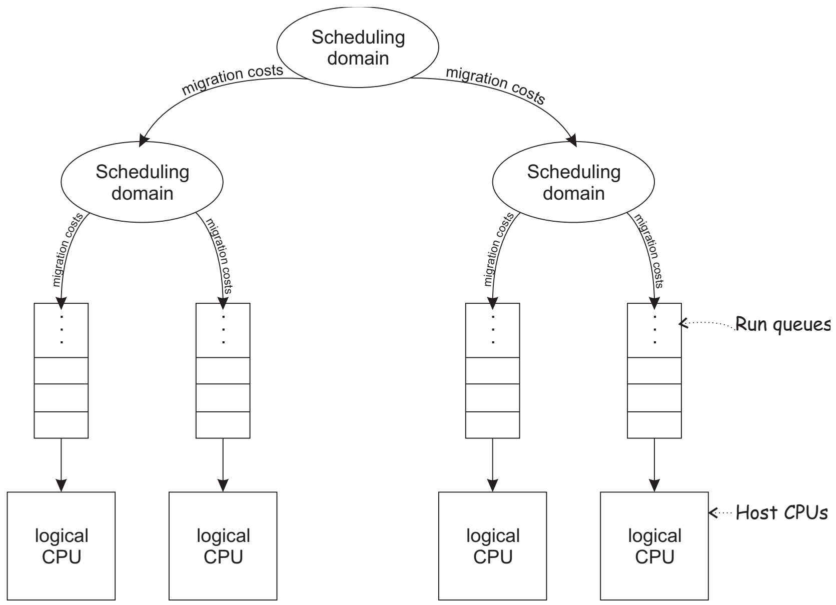


Figure 21. Linux scheduling

libvirt provides means to assign virtual CPUs to groups of host CPUs in order to minimize migration costs. This process is called *CPU pinning*. CPU pinning forces the Linux scheduler to migrate virtual CPUs only between those host CPUs of the specified group. Likewise, the execution of the user space process or I/O threads can be assigned to groups of host CPUs.

Attention: Do not use CPU pinning, because a successful CPU pinning depends on a variety of factors which can change over time:

- CPU pinning can lead to the opposite effect of what was desired when the circumstances for which it was designed change. This may occur, for example, when the host reboots, the workload on the host changes, or the virtual servers are modified.
- Deactivating operating CPUs and activating standby CPUs (CPU hotplug) on the host may lead to a situation where host CPUs are no longer available for the execution of virtual server threads after their reactivation.

CPU weight

The host CPU time which is available for the execution of the virtual CPUs depends on the system utilization.

The available CPU time is divided up between the virtual servers running on the host.

The Linux scheduler and the Linux kernel feature cgroups allocate the upper limit of *CPU time shares* (or simply: *CPU shares*) which a virtual server is allowed to use based on the CPU weight of all virtual servers running on the host.

You can configure the CPU weight of a virtual server, and you can modify it during operation.

The CPU shares of a virtual server are calculated by forming the virtual server's weight-fraction.

Example:

Virtual server	CPU weight	Weight-sum	Weight-fraction	CPU shares
A	1024	3072	1024/3072	1/3
B	2048	3072	2048/3072	2/3

The number of virtual CPUs does not affect the CPU shares of a virtual server.

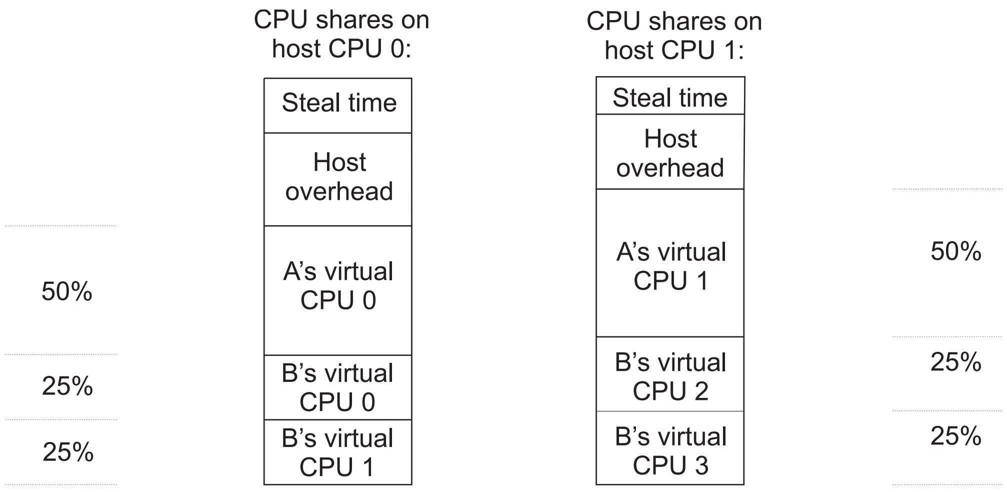
Example:

Virtual server	CPU weight	Number of virtual CPUs
A	1024	2
B	1024	4

The CPU shares are the same for both virtual servers:

Virtual server	CPU weight	Weight-sum	Weight-fraction	CPU shares
A	1024	2048	1024/2048	1/2
B	1024	2048	1024/2048	1/2

The CPU shares of each virtual server are spread across its virtual CPUs, such as:



Chapter 22. Memory management

The memory configured for a virtual server appears as physical memory to the guest operating system but is realized as a Linux virtual address space.

Virtual server memory has the same characteristics as virtual memory used by other Linux processes. For example, it is protected from access by other virtual servers or applications running on the host. It also allows for *memory overcommitment*, that is, the amount of virtual memory for one or more virtual servers may exceed the amount of physical memory available on the host.

Memory is organized in fixed size blocks called *pages*. Each virtual server memory page must be backed by a physical page of the host. Since more virtual pages than physical pages can exist, it is necessary that the content of currently unused virtual pages can be temporarily stored on a storage volume (*swap device*) and retrieved upon access by the guest. The activity of storing pages to and retrieving them from the disk is called *swapping*.

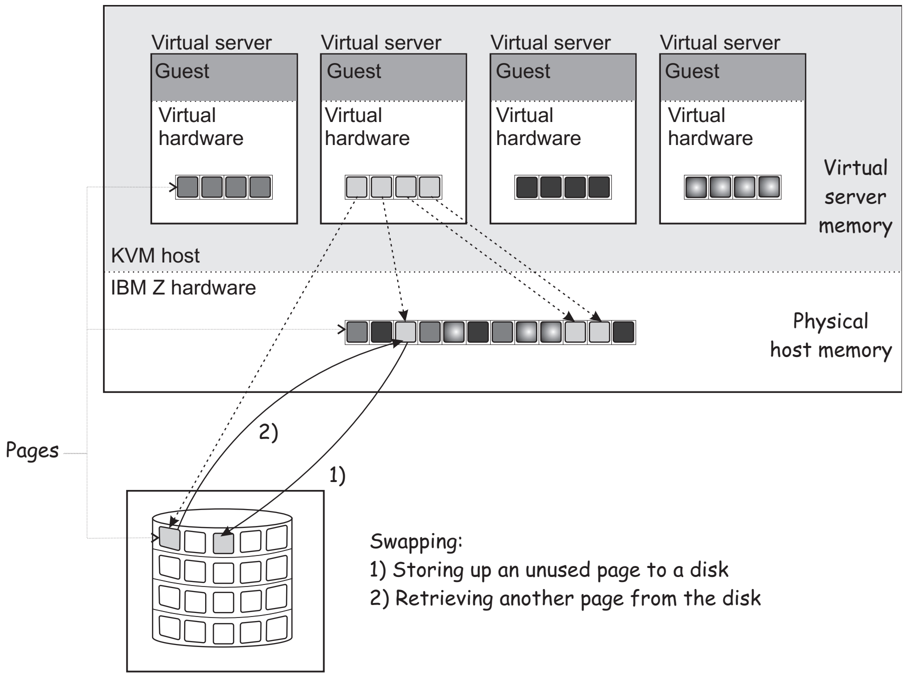


Figure 22. Swapping

Since disk storage access is significantly slower than memory access, swapping will slow down the execution of a virtual server even though it happens transparently for the guest. Careful planning of virtual server memory handling is therefore essential for an optimal system performance.

Tip:
- Plan a memory ratio of not more than virtual-to-real to 2:1
- Configure the minimum amount of memory necessary for each virtual server

Even if the defined virtual server memory exceeds the physical host memory significantly, the actual memory usage of a virtual server may be considerably less than the defined amount. There are multiple techniques allowing the host to efficiently deal with memory overcommitment:

Collaborative memory management

A guest operating system can mark memory pages as *unused* or *volatile* with the IBM Z Collaborative Memory Management Assist (CMMA) facility. This allows the host to avoid unnecessary disk swapping because unused pages can simply be discarded. Current Linux operating systems make use of CMMA. The subset of the CMMA facility as used by Linux is enabled in KVM, therefore transparently ensuring efficient physical host memory usage, while still allowing the virtual server to use all of the defined virtual memory if needed.

Ballooning

KVM implements a virtual memory balloon device that serves the purpose of controlling the physical host memory usage of a virtual server. With the balloon device, the host can request that the guest gives up memory. This could be done to re-balance the resource allocations between virtual servers to adapt to changing resource needs.

Whether and to which extent the guest honors the request depends on a few factors not controlled by the host, such as, whether or not a balloon device driver is installed in the guest, or whether there's enough memory that can be freed.

Unlike for CMMA, the memory given up by the balloon device is removed from the virtual server and cannot be reclaimed by the guest. As this can cause adverse effects and even lead to program or operating system failures due to low memory conditions, it should only be used in well-understood situations. By default, you should disable the balloon device by configuring `<memballoon model="none"/>`.

Memory tuning

Another way to control virtual server memory usage is by means of the Linux cgroups memory controller. By specifying a soft limit the amount of physical host memory used by a virtual server can be restricted once the host is under high memory pressure, that is, the host is experiencing high swapping activity. Again, this would typically be done to re-balance resource allocations between virtual servers.

Since the virtual server memory available to the guest is not modified, applying a soft limit is transparent, except for the performance penalty caused by swapping. If swapping becomes excessive, time-critical processes may be affected, causing program failures. Therefore the soft limit should be applied carefully as well.

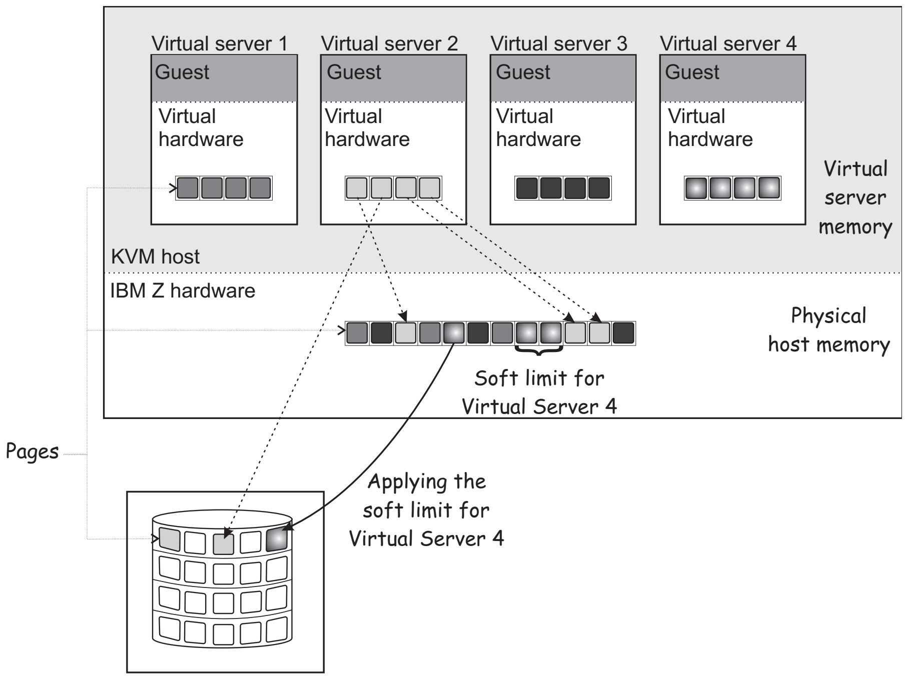


Figure 23. Applying the soft limit

The virtual server memory soft limit can be controlled statically using the soft_limit child element of the memtune element or dynamically using the virsh **memtune** command.

Related tasks:

"Configuring virtual memory" on page 65
Configure the virtual server memory.

"Managing virtual memory" on page 143
Specify a soft limit for the amount of physical host memory used by a virtual server.

Chapter 23. Storage management

Consider these aspects when setting up and configuring the virtual server storage.

I/O threads

I/O threads are dedicated to perform I/O operations on virtual block devices.

For a good performance of I/O operations, provide one I/O thread for each virtual block device. Estimate no more than one or two I/O threads per host CPU and no more I/O threads than virtual block I/O devices that will be available for the virtual server. Too many I/O threads will reduce system performance by increasing the system overhead.

You can configure I/O threads in the domain configuration-XML of a virtual server. For more information, see:

- “Configuring devices with the virtual server” on page 69
- “Configuring a DASD or SCSI disk” on page 78
- “Configuring a virtual HBA” on page 88

When you attach a virtual block device to a virtual server, you can provide an I/O thread for this device during operation and remove it after use. For more information, see:

- “Attaching a device” on page 146
- “Detaching a device” on page 147

Logical volume management

Consider these aspects when the virtual server utilizes logical volumes.

Path redundancy

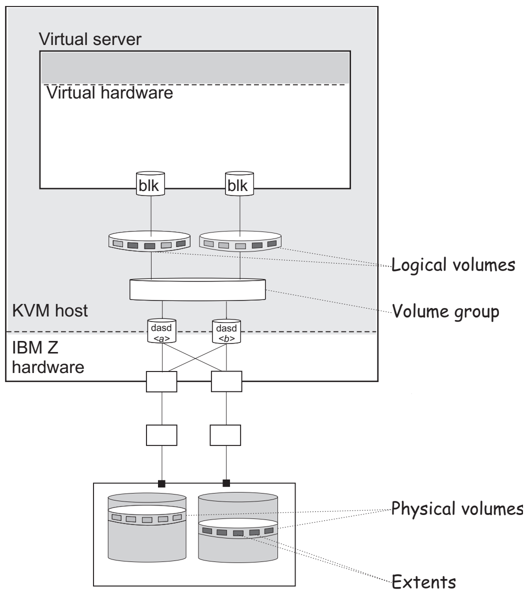
As discussed in Chapter 2, “Virtual block devices,” on page 9, it is important to ensure that you provide path redundancy for all physical volumes. Especially, all LVM physical volumes on SCSI disks have to be assembled from device mapper-created device nodes.

Data integrity

There are two ways to manage logical volumes:

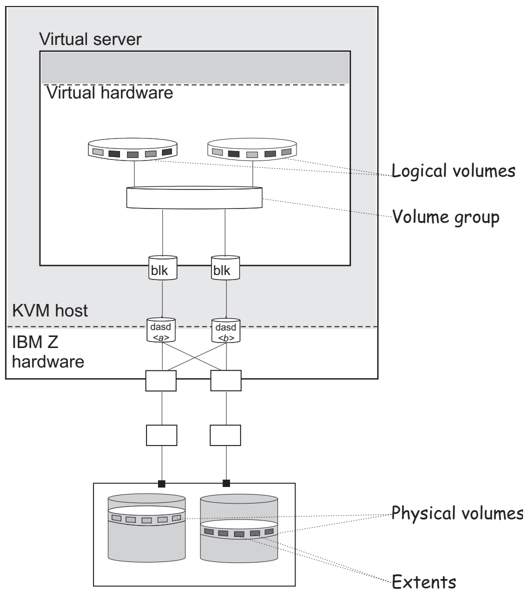
- On the host:

  This example shows multipathed DASDs. The logical volumes that are managed on the host are configured as virtual block devices.



- On the virtual server:

  When you configure physical volumes as virtual block devices, the logical volumes are managed on the virtual server. In this case you need to prohibit a logical volume management of the configured physical volumes on the host. Else, the host might detect the physical volumes and try to manage them on the host, too. Storing host metadata on the physical volumes might cause a loss of virtual server data.



To prohibit a logical volume management for physical volumes that are managed on the virtual server, provide an explicit whitelist in `/etc/lvm/lvm.conf` which explicitly contains all disk block devices to be managed on the host, or a blacklist that contains all physical volumes that are to be managed on the virtual server.

The filter section in the device settings allows to specify a whitelist using the prefix “a”, and to specify a blacklist using the prefix “r”.

Example

This whitelist in `/etc/lvm/lvm.conf` filters the physical volumes which are to be managed on the host. The last line ("`r|.*|`") denotes that all other physical volumes that are not listed here are not to be managed on the host.

```
devices
       {filter = [ "a|/dev/mapper/36005076305ffc1ae00000000000021d5p1|",
                   "a|/dev/mapper/36005076305ffc1ae00000000000021d7p1|",
                   "a|/dev/disk/by-path/ccw-0.0.1607-part1|",
                   "r|.*|" ]
       }
```

The following physical volumes are to be managed on the host:

- `/dev/mapper/36005076305ffc1ae00000000000021d5p1`
- `/dev/mapper/36005076305ffc1ae00000000000021d7p1`
- `/dev/disk/by-path/ccw-0.0.1607-part1`

You can verify that SCSI disks are referenced correctly by issuing the following **pvscan** command:

```
# pvscan -vvv 2>&1 | fgrep '/dev/sd'
...
   /dev/sda: Added to device cache
   /dev/block/8:0: Aliased to /dev/sda in device cache
   /dev/disk/by-path/ccw-0.0.50c0-zfcp-0x1234123412341234:\
     0x0001000000000000: Aliased to /dev/sda in device cache
   ...
   /dev/sda: Skipping (regex)
```

The output must contain the string "Skipping (regex)" for each SCSI disk standard device name which is configured for the virtual server.

Part 6. Diagnostics and troubleshooting

Chapter 24. Logging 173
Log messages 173
Specifying the logging level of the libvirt log messages 173

Chapter 25. Dumping 175
Creating a virtual server dump on the host . . . 175
Creating a dump on the virtual server 175

Chapter 26. Collecting performance metrics . . 177

Monitor and display information that helps to diagnose and solve problems.

Chapter 24. Logging

Adapt the logging facility to your needs.

Log messages

These logs are created.

libvirt log messages
By default, libvirt log messages are stored in the system journal. You can specify a different location in the libvirt configuration file at `/etc/libvirt/libvirtd.conf`. For more information, see libvirt.org/logging.html.

QEMU log file of a virtual server
`/var/log/libvirt/qemu/<VS>.log`, where <*VS*> is the name of the virtual server.

Console log file
If the log element is specified in the console configuration, the log file attribute indicates the console log file.

Example:

The following console configuration specifies the console log file `/var/log/libvirt/qemu/vserv-cons0.log`:

```
<devices>
    ...
    <console type="pty">
       <target type="sclp" port="0"/>
       <log file="/var/log/libvirt/qemu/vserv-cons0.log" append="on"/>
    </console>
<devices/>
```

Specifying the logging level of the libvirt log messages

Specify the level of logging information that is displayed in the libvirt log messages file.

About this task

For further information, see: libvirt.org/logging.html

Procedure

1. In the libvirt configuration file `/etc/libvirt/libvirtd.conf`, specify:

   `log_level = <n>`

   Where <*n*> is the logging level:

   4 Displays errors.

   3 Is the default logging level, which logs errors and warnings.

   2 Provides more information than logging level 3.

   1 Is the most verbose logging level.
2. Restart the libvirt daemon to enable the changes.

```
# systemctl restart libvirtd.service
```

Chapter 25. Dumping

Create dumps of a crashed virtual server on the host or on the virtual server.

Creating a virtual server dump on the host

When the virtual server is crashed, you can create a dump on the host.

Procedure

Create a dump of the crashed virtual server using the virsh **dump** command with the `--memory-only` option:

```
# virsh dump --memory-only <VS> <dumpfile>
```

<dumpfile>
: Is the name of the dump file. If no fully qualified path to the dump file is specified, it is written to the current working directory of the user who issues the virsh **dump** command.

<VS>
: Is the name of the virtual server as specified in its domain configuration-XML file.

Results

The dump is written to the file *<dumpfile>*.

What to do next

To inspect the dump, enter the command:

```
# crash <dumpfile> <kernel-image-filename>
```

<kernel-image-filename>
: Is the name of the kernel image file of the guest running on the dumped virtual server.

Creating a dump on the virtual server

When a virtual server is crashed, you can provide a dump for the virtual server user.

Before you begin

Ensure that kdump is installed and enabled on the virtual server.

If kdump is not enabled on the virtual server, the following procedure causes only a restart of the virtual server.

For more information about kdump, see *Using the Dump Tools*, SC33-8412.

Procedure

- In case of a virtual server kernel panic, a dump is automatically created.
- In case of a non-responding virtual server, you can trigger a restart interrupt.

  The interrupt handling of a restart interrupt depends on the PSW restart configuration and ends up in a dump.

  To trigger a restart interrupt, use the virsh **inject-nmi** command:

  ```
  # virsh inject-nmi <VS>
  ```

  <VS> Is the name of the virtual server as specified in its domain configuration-XML file.

Results

The virtual server creates a dump and then restarts in kdump mode.

What to do next

To verify your action, you might want to see the dump on the virtual server:

1. Log in to the virtual server as root.
2. Use the **makedumpfile** command to create a dump file from the `vmcore` file:

   ```
   [root@guest:] # makedumpfile -c <vmcore> <dumpfile>
   ```

3. To inspect the dump, enter:

   ```
   [root@guest:] # crash <dumpfile> <kernel-image-filename>
   ```

   The **`crash`** command is available with the kernel-debuginfo package.

   <dumpfile>
   Is the fully qualified path and file name of the dump file.

   <kernel-image-filename>
   Is the name of the kernel image file of the guest running on the dumped virtual server.

   <vmcore>
   Is the fully qualified path and file name of the `vmcore` file of the guest.

Chapter 26. Collecting performance metrics

You can monitor virtual server machine code instructions.

Before you begin

- Make sure that your kernel is built using the common source options `CONFIG_TRACEPOINTS`, `CONFIG_HAVE_PERF_EVENTS`, and `CONFIG_PERF_EVENTS`.
- Make sure that the **perf** tool is installed.

  You can check this by issuing:

  ```
  # perf list
  ...
  kvm:kvm_s390_sie_enter                                [Tracepoint event]
  ...
  ```

  If the command returns a list of supported events, such as the tracepoint event `kvm_s390_sie_enter`, the tool is installed.

Procedure

You collect, record, and display performance metrics with the **perf kvm stat** command.

- The **record** subcommand records performance metrics and stores them in the file `perf.data.guest`.
  - The **perf** tool records events until you terminate it by pressing Control and c (Ctrl+c).
  - To display the recorded data, use the **report** subcommand.
  - It is recommended to save `perf.data.guest` before you collect new statistics, because a new record may overwrite this file.
- The **live** subcommand displays the current statistics without saving them.

  The **perf** tool displays events until you terminate it by pressing Control and c (Ctrl+c).

Example

```
# ./perf kvm stat record -a
^C[ perf record: Woken up 7 times to write data ]
[ perf record: Captured and wrote 13.808 MB perf.data.guest (~603264 samples) ]

# ./perf kvm stat report


Analyze events for all VMs, all VCPUs:

                             VM-EXIT    Samples  Samples%     Time%    Min Time     Max Time         Avg time

                   Host interruption      14999    35.39%     0.39%      0.45us    1734.88us      0.82us ( +-  19.59% )
          DIAG (0x44) time slice end      13036    30.76%     0.57%      1.06us    1776.08us      1.39us ( +-   9.81% )
   DIAG (0x500) KVM virtio functions      13011    30.70%     1.90%      1.15us    2144.75us      4.65us ( +-   5.08% )
                          0xE5 TPROT        512     1.21%     0.01%      0.79us       2.18us      0.83us ( +-   0.42% )
                           0xB2 TSCH        406     0.96%     0.19%      7.35us     109.43us     14.95us ( +-   2.97% )
                          0xB2 SERVC        117     0.28%     0.15%     10.97us     339.00us     40.46us ( +-   9.17% )
                    External request        113     0.27%     0.01%      0.75us       2.58us      1.56us ( +-   1.55% )
                          0xB2 STSCH         57     0.13%     0.02%      7.30us      26.40us      9.47us ( +-   5.99% )
                          Wait state         40     0.09%    96.48%   3334.30us  464600.00us  76655.28us ( +-  32.97% )
                           0xB2 MSCH         14     0.03%     0.00%      7.22us       9.19us      7.74us ( +-   2.13% )
                           0xB2 SSCH         14     0.03%     0.01%      8.67us      35.41us     16.16us ( +-  16.38% )
                           0xB2 CHSC         10     0.02%     0.00%      7.51us      22.90us     11.06us ( +-  15.20% )
                         I/O request          8     0.02%     0.00%      1.37us       1.97us      1.55us ( +-   5.77% )
                           0xB2 STPX          8     0.02%     0.00%      1.04us       7.10us      1.98us ( +-  37.25% )
                           0xB2 STSI          7     0.02%     0.00%      1.65us      62.09us     22.26us ( +-  41.95% )
                          0xB2 STIDP          4     0.01%     0.00%      1.12us       3.62us      2.62us ( +-  21.07% )
               SIGP set architecture          3     0.01%     0.00%      1.05us       2.68us      1.60us ( +-  33.74% )
                           0xB2 STAP          3     0.01%     0.00%      1.05us       7.61us      3.39us ( +-  62.25% )
                           0xB2 STFL          3     0.01%     0.00%      1.78us       3.88us      2.84us ( +-  21.31% )
    DIAG (0x204) logical-cpu utilization      2     0.00%     0.00%      4.58us      39.48us     22.03us ( +-  79.19% )
          DIAG (0x308) ipl functions          2     0.00%     0.01%     19.34us     329.25us    174.30us ( +-  88.90% )
   DIAG (0x9c) time slice end directed        1     0.00%     0.00%      1.09us       1.09us      1.09us ( +-   0.00% )
                            0xB2 SPX          1     0.00%     0.00%      4.58us       4.58us      4.58us ( +-   0.00% )
                           0xB2 SETR          1     0.00%     0.00%     56.97us      56.97us     56.97us ( +-   0.00% )
                           0xB2 SSKE          1     0.00%     0.25%   7957.94us    7957.94us   7957.94us ( +-   0.00% )
                          0xB2 STCRW          1     0.00%     0.00%     11.24us      11.24us     11.24us ( +-   0.00% )
    DIAG (0x258) page-reference services      1     0.00%     0.00%      4.87us       4.87us      4.87us ( +-   0.00% )
                           0xB9 ESSA          1     0.00%     0.00%      8.72us       8.72us      8.72us ( +-   0.00% )
                          0xEB LCTLG          1     0.00%     0.00%      9.27us       9.27us      9.27us ( +-   0.00% )

Total Samples:42377, Total events handled time:3178166.35us.
```

What to do next

For more information about the **perf** subcommand **kvm stat**, see the man page or issue the full subcommand with the `--help` option:

```
►►──perf──kvm──stat──┬─record─┬── --help──────────────────────────────────►◄
                     ├─report─┤
                     └─live───┘
```

With the collected statistics, you can watch the virtual server behavior and time consumption and then analyze the recorded events. So you may find hints for possible sources of error.

- You can find a description of the general instructions in the *z/Architecture®* *Principles of Operation*, SA22-7832, for example:

Mnemonic	Instruction	Opcode
TPROT	TEST PROTECTION	E501
TSCH	TEST SUBCHANNEL	B235

- Signal-processor orders (SIGP) are also described in the *z/Architecture Principles of Operation*, SA22-7832.
- Table 1 on page 179 lists all diagnoses (DIAG) as supported by KVM on IBM Z.

Table 1. Supported Linux diagnoses

Number	Description	Linux use	Required/ Optional
0x010	Release pages	CMM	Required
0x044	Voluntary time-slice end	In the kernel for spinlock and udelay	Required
0x09c	Voluntary time slice yield	Spinlock	Optional
0x258	Page-reference services	In the kernel, for pfault	Optional
0x288	Virtual server time bomb	The watchdog device driver	Required
0x308	Re-ipl	Re-ipl and dump code	Required
0x500	Virtio functions	Operate virtio-ccw devices	Required

Required means that a function is not available without the diagnose; optional means that the function is available but there might be a performance impact.

You may also find other DIAG events on your list, but those are not supported by KVM on IBM Z. A list of all Linux diagnoses is provided in *Device Drivers, Features, and Commands*, SC33-8411.

Part 7. Reference

Chapter 27. Virtual server life cycle 183
shut off . . . 184
running . . . 185
paused . . . 186
crashed . . . 187
in shutdown. . . . 188

Chapter 28. Selected libvirt XML elements . . . 189
<adapter> as child element of <source> . . . 191
<address> as child element of <controller>, <disk>, <interface>, and <memballoon> . . . 192
<address> as child element of <hostdev> or <disk> 193
<address> as child element of <source> . . . 194
<backend> . . . 195
<boot> . . . 196
| <bridge> . . . 197
<cipher> . . . 198
<cmdline> . . . 199
<console> . . . 200
<controller> . . . 201
| <cpu> . . . 202
<cputune> . . . 203
<device> . . . 204
<devices> . . . 205
| <dhcp> . . . 206
<disk> . . . 207
<domain> . . . 208
<driver> as child element of <controller> . . . 209
<driver> as child element of <disk> . . . 210
<emulator> . . . 212
| <feature> . . . 213
<format> . . . 214
| <forward> . . . 215
<geometry> . . . 216
<host> . . . 217
<hostdev> . . . 218
<initrd> . . . 219
<interface> . . . 220
<iothreads> . . . 221
| <ip> . . . 222
<kernel> . . . 223
<key> . . . 224
<keywrap> . . . 225
<log> . . . 226
<mac> . . . 227
<memballoon> . . . 228
<memory> . . . 229
<memtune> . . . 231
| <model> as a child element of <cpu> . . . 232
<model> as a child element of <interface> . . . 233
<name> as a child element of <domain> . . . 234
| <name> as a child element of <network> . . . 235
| <network> . . . 236
<on_crash> . . . 237
<on_reboot> . . . 238
<os> . . . 239
<path> as child element of <pool><target> . . . 240
<path> as child element of <volume><target> . . . 241
<pool> . . . 242
<readonly> . . . 244
<rng> . . . 245
<shareable> . . . 246
<shares> . . . 247
<soft_limit> . . . 248
<source> as child element of <disk> . . . 249
<source> as child element of <hostdev> . . . 251
<source> as child element of <interface> . . . 252
<source> as child element of <pool> . . . 253
<target> as child element of <console> . . . 254
<target> as child element of <disk> . . . 255
<target> as child element of <pool> . . . 256
<target> as child element of <volume> . . . 257
<type> . . . 258
<vcpu> . . . 259
<virtualport> as a child element of <interface> . . 260
| <virtualport> as a child element of <network> . . 261
<volume> . . . 262
<watchdog> . . . 263

Chapter 29. Selected virsh commands 265
attach-device . . . 268
change-media . . . 270
console . . . 272
define . . . 273
destroy . . . 274
detach-device . . . 275
domblklist . . . 277
domblkstat . . . 278
domcapabilities. . . . 280
domiflist . . . 281
domifstat . . . 282
dominfo . . . 283
domjobabort. . . . 284
domstate . . . 285
dump . . . 286
dumpxml. . . . 287
edit. . . . 288
inject-nmi . . . 289
iothreadadd . . . 290
iothreaddel . . . 292
iothreadinfo . . . 294
list . . . 295
managedsave . . . 297
memtune . . . 299
migrate . . . 300
migrate-getspeed . . . 303
migrate-setmaxdowntime . . . 304
migrate-setspeed . . . 305
| net-autostart . . . 306
| net-define . . . 307
| net-destroy . . . 308
| net-dumpxml . . . 309

| net-edit 310
| net-info 311
| net-list. 312
| net-name 313
| net-start 314
| net-undefine. 315
| net-uuid 316
pool-autostart 317
pool-define 318
pool-delete 319
pool-destroy. 320
pool-dumpxml 321
pool-edit 322
pool-info 323
pool-list 324
pool-name 325
pool-refresh 326
pool-start. 327
pool-undefine 328
pool-uuid 329
reboot 330
resume 331
schedinfo. 332
shutdown 333
setvcpus 334
start 336
suspend 338
undefine 339
vcpucount 340
vol-create. 341
vol-delete. 342
vol-dumpxml 343
vol-info 344
vol-key 345
vol-list. 346
vol-name 347
vol-path 348
vol-pool 349

Chapter 30. Selected QEMU commands. . . . 351
QEMU monitor commands 351
Examples for the use of the qemu-img command 351

Chapter 31. Hypervisor information for the virtual server user 353

Get an overview of the virtual server states and the elements and commands that are specific to configure and operate a virtual server on IBM Z. The virtual server user can retrieve information about the IBM Z hardware and the LPAR on which the KVM host runs.

Chapter 27. Virtual server life cycle

Display the state of a defined virtual server including the reason with the virsh **`domstate --reason`** command.

Figure 24 shows the life cycle of a defined virtual server: States, their reasons, and state transitions which are caused by the virsh virtual server management commands. The state transitions shown in this figure do not comprise command options that you can use to further influence the state transition.

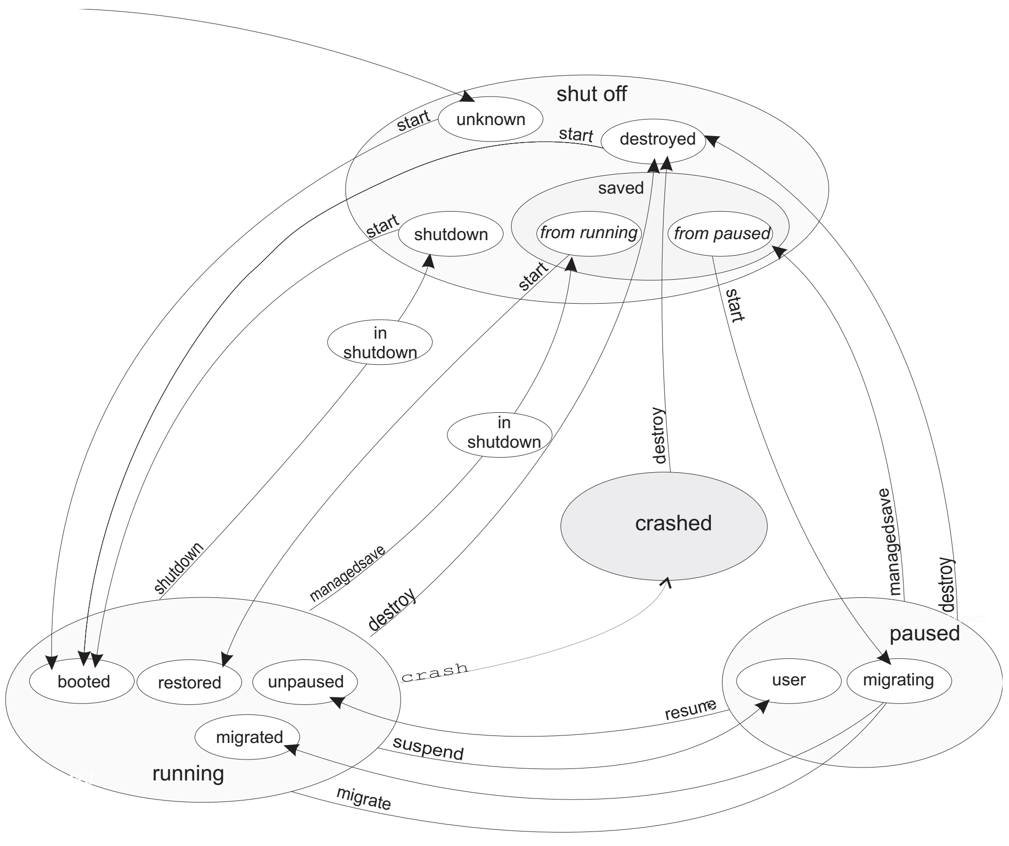


Figure 24. State-transition diagram of a virtual server including reasons

shut off

The virtual server is defined to libvirt and has not yet been started, or it was terminated.

Reasons

unknown	The virtual server is defined to the host.
saved	The system image of the virtual server is saved in the file `/var/lib/libvirt/qemu/save/<`*VS*`>.save` and can be restored. The system image contains state information about the virtual server. Depending on this state, the virtual server is started in the state running or paused.
shutdown	The virtual server was properly terminated. The virtual server's resources were released.
destroyed	The virtual server was immediately terminated. The virtual server's resources were released.

Commands

Command	From state (reason)	To state (reason)
`start`	shut off (unknown)	running (booted)
`start`	shut off (saved *from running*)	running (restored)
`start`	shut off (saved *from paused*)	paused (migrating)
`start`	shut off (shutdown)	running (booted)
`start`	shut off (destroyed)	running (booted)
`start --force-boot`	shut off (unknown)	running (booted)
`start --force-boot`	shut off (saved *from running*)	running (booted)
`start --force-boot`	shut off (saved *from paused*)	paused (user)
`start --force-boot`	shut off (shutdown)	running (booted)
`start --force-boot`	shut off (destroyed)	running (booted)
`start --paused`	shut off (unknown)	paused (user)
`start --paused`	shut off (saved *from running*)	paused (migrating)
`start --paused`	shut off (saved *from paused*)	paused (migrating)
`start --paused`	shut off (shutdown)	paused (user)
`start --paused`	shut off (destroyed)	paused (user)

running

The virtual server was started.

Reasons

booted	The virtual server was started from scratch.
migrated	The virtual server was restarted on the destination host after the stopped phase of a live migration.
restored	The virtual server was started at the state indicated by the stored system image.
unpaused	The virtual server was resumed from the paused state.

Commands

Command	Transition state	To state (reason)
`destroy`	n/a	shut off (destroyed)
`managedsave`	n/a	shut off (saved *from running*)
`managedsave --running`	n/a	shut off (saved *from running*)
`managedsave --paused`	n/a	shut off (saved *from paused*)
`migrate`	paused (migrating)	running (migrated)
`migrate --suspend`	paused (migrating)	paused (user)
`shutdown`	in shutdown	shut off (shutdown)
`suspend`	n/a	paused (user)

paused

The virtual server has been suspended.

Reasons

user	The virtual server was suspended with the virsh **suspend** command.
migrating	The virtual server's system image is saved and the virtual server is halted - either because it is being migrated, or because it is started from a saved shut off state.

Commands

Command	Transition state	To state (reason)
destroy	n/a	shut off (destroyed)
managedsave	n/a	shut off (saved *from paused*)
managedsave --running	n/a	shut off (saved *from running*)
managedsave --paused	n/a	shut off (saved *from paused*)
resume	n/a	running (unpaused)
shutdown	in shutdown	shut off (shutdown)

crashed

The virtual server crashed and is not prepared for a reboot.

You can create memory dumps of the virtual server.

Then, you can terminate the virtual server and restart it.

For testing purposes, you can crash a virtual server with the virsh **`inject-nmi`** command.

Commands

Command	To state (reason)
`destroy`	shut off (destroyed)

in shutdown

While the virtual server is shutting down, it traverses the “in shutdown” state.

Chapter 28. Selected libvirt XML elements

These libvirt XML elements might be useful for you. You find the complete libvirt XML reference at libvirt.org.

- “<adapter> as child element of <source>” on page 191
- “<address> as child element of <controller>, <disk>, <interface>, and <memballoon>” on page 192
- “<address> as child element of <hostdev> or <disk>” on page 193
- “<address> as child element of <source>” on page 194
- “<backend>” on page 195
- “<boot>” on page 196
- “<bridge>” on page 197
- “<cipher>” on page 198
- “<cmdline>” on page 199
- “<console>” on page 200
- “<controller>” on page 201
- “<cpu>” on page 202
- “<cputune>” on page 203
- “<device>” on page 204
- “<devices>” on page 205
- “<dhcp>” on page 206
- “<disk>” on page 207
- “<domain>” on page 208
- “<driver> as child element of <controller>” on page 209
- “<driver> as child element of <disk>” on page 210
- “<emulator>” on page 212
- “<feature>” on page 213
- “<format>” on page 214
- “<forward>” on page 215
- “<geometry>” on page 216
- “<host>” on page 217
- “<hostdev>” on page 218
- “<initrd>” on page 219
- “<interface>” on page 220
- “<iothreads>” on page 221
- “<ip>” on page 222
- “<kernel>” on page 223
- “<key>” on page 224
- “<keywrap>” on page 225
- “<log>” on page 226
- “<mac>” on page 227
- “<memballoon>” on page 228
- “<memory>” on page 229

- “<memtune>” on page 231
- “<model> as a child element of <cpu>” on page 232
- “<model> as a child element of <interface>” on page 233
- “<name> as a child element of <domain>” on page 234
- “<name> as a child element of <network>” on page 235
- “<network>” on page 236
- “<on_crash>” on page 237
- “<on_reboot>” on page 238
- “<os>” on page 239
- “<path> as child element of <pool><target>” on page 240
- “<path> as child element of <volume><target>” on page 241
- “<pool>” on page 242
- “<readonly>” on page 244
- “<rng>” on page 245
- “<shareable>” on page 246
- “<shares>” on page 247
- “<soft_limit>” on page 248
- “<source> as child element of <disk>” on page 249
- “<source> as child element of <hostdev>” on page 251
- “<source> as child element of <interface>” on page 252
- “<source> as child element of <pool>” on page 253
- “<target> as child element of <console>” on page 254
- “<target> as child element of <disk>” on page 255
- “<target> as child element of <pool>” on page 256
- “<target> as child element of <volume>” on page 257
- “<type>” on page 258
- “<vcpu>” on page 259
- “<virtualport> as a child element of <interface>” on page 260
- “<virtualport> as a child element of <network>” on page 261
- “<volume>” on page 262
- “<watchdog>” on page 263

<adapter> as child element of <source>

Specifies an FCP device (Host Bus Adapter).

Text content

None.

Selected attributes

name=scsi_host<*n*>
: Specifies the name of the FCP device, where <*n*> is a nonnegative integer.

Usage

"Configuring a SCSI tape or medium changer device" on page 90

Parent elements

"<source> as child element of <hostdev>" on page 251.

Child elements

None.

Example

```
<devices>
   ...
   <controller type="scsi" model="virtio-scsi" index="0"/>
   <hostdev mode="subsystem" type="scsi">
      <source>
         <adapter name="scsi_host0"/>
         <address bus="0" target="0" unit="0"/>
      </source>
      <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
   ...
</devices>
```

<address> as child element of <controller>, <disk>, <interface>, and <memballoon>

Specifies the address of a device on the virtual server.

Text content

None.

Selected attributes

type=ccw
: Specifies a virtio CCW device, such as a block device or a network device.

  You can specify the device bus-ID with the address attributes cssid, ssid, and devno.

cssid
: Specifies the channel subsystem number of the virtual device. Must be “0xfe”.

ssid
: Specifies the subchannel set of the virtual device. Valid values are between “0x0” and “0x3”.

devno
: Specifies the device number of the virtio device. Must be a unique value between “0x0000” and “0xffff”.

Usage

- “Configuring a DASD or SCSI disk” on page 78
- “Configuring an image file as storage device” on page 84

Parent elements

- “<controller>” on page 201
- “<disk>” on page 207
- “<interface>” on page 220
- “<memballoon>” on page 228

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vda" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x1108"/>
</disk>
```

<address> as child element of <hostdev> or <disk>

Specifies the address of a device, which is connected to the virtual server through a controller.

Text content

None.

Selected attributes

type=scsi
Specifies a SCSI device.

controller
Specifies the virtual controller of the virtual device. Enter the index attribute value of the respective controller element.

bus Specifies the virtual SCSI bus of the virtual device.

target Specifies the virtual SCSI target of the virtual device. This value can be between 0 and 255.

unit Specifies the unit number (LUN) of the virtual SCSI device.

Usage

- "Configuring a SCSI tape or medium changer device" on page 90
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 95

Parent elements

- "<hostdev>" on page 218
- "<disk>" on page 207

Child elements

None.

Example

```
<devices>
    ...
    <controller type="scsi" model="virtio-scsi" index="0"/>
    <hostdev mode="subsystem" type="scsi">
        <source>
            <adapter name="scsi_host0"/>
            <address bus="0" target="0" unit="0"/>
        </source>
        <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
    </hostdev>
    ...
    <controller type="scsi" model="virtio-scsi" index="1"/>
    <disk type="file" device="cdrom">
        <driver name="qemu" type="raw" io="native" cache="none"/>
        <source file="/var/lib/libvirt/images/cd.iso"/>
        <target dev="vda" bus="scsi"/>
        <address type="drive" controller="1" bus="0" target="0" unit="0"/>
        <readonly/>
    </disk>
    ...
</devices>
```

<address> as child element of <source>

Specifies a device address from the host point of view.

Text content

None.

Selected attributes

bus=0	For a SCSI device the value is zero.
target	Specifies the SCSI ID.
unit	Specifies the SCSI LUN.

Usage

"Configuring a SCSI tape or medium changer device" on page 90

Parent elements

"<source> as child element of <hostdev>" on page 251

Child elements

None.

Example

```
<devices>
   ...
   <controller type="scsi" model="virtio-scsi" index="0"/>
   <hostdev mode="subsystem" type="scsi">
      <source>
         <adapter name="scsi_host0"/>
         <address bus="0" target="0" unit="0"/>
      </source>
      <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
   ...
</devices>
```

<backend>

Specifies the character device which generates the random numbers.

Text content

Specifies the device node of the input character device. The default value and currently the only valid value is `/dev/random`.

Selected attributes

model=random
Specifies the source model.

Usage

"Configuring a random number generator" on page 102

Parent elements

"<rng>" on page 245

Child elements

None.

Example

```
<devices>
    ...
    <rng model="virtio">
        <backend model="random">/dev/random</backend>
    </rng>
    ...
</devices>
```

<boot>

Indicates that the virtual block device is bootable.

Text content

None.

Selected attributes

order=*number*
: Specifies the order in which a device is considered as boot device during the boot sequence.

loadparm=*number*
: For IPL devices with a boot menu configuration: Specifies the boot menu entry. If this parameter is omitted, the default entry is booted.

Usage

"Configuring the boot process" on page 53

Parent elements

"<disk>" on page 207

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x1" devno="0xa30e"/>
    <boot order="1" loadparm="2"/>
</disk>
```

<bridge>

Configures the bridge device that is used to set up the virtual network.

Text content

None.

Selected attributes

name Specifies a name for the bridge.

Usage

Chapter 12, “Configuring virtual networks,” on page 105

Parent elements

“<network>” on page 236

Child elements

None.

Example

```
<network>
  ...
  <bridge name="virbr2"/>
  ...
</network>
```

<cipher>

Configures the generation of an AES or DEA/TDEA wrapping key and the use of the respective protected key management operations on the virtual server.

Text content

None.

Selected attributes

name=aes | dea
Specifies the AES or DEA/TDEA wrapping key.

state=on | off

on Enables wrapping key generation.

The respective protected key management operations are available on the virtual server.

off Disables wrapping key generation.

The respective protected key management operations are not available on the virtual server.

Usage

"Disabling protected key encryption" on page 72

Parent elements

"<keywrap>" on page 225

Child elements

None.

Example

```
<domain type="kvm">
    ...
    <keywrap>
        <cipher name="aes" state="off"/>
    </keywrap>
    ...
</domain>
```

<cmdline>

Specifies arguments to be passed to the kernel (or installer) at boot time.

Text content

Command line arguments using the same syntax as if they were specified in the command line.

Selected attributes

None.

Usage

"Configuring a kernel image file as IPL device" on page 55

Parent elements

"<os>" on page 239

Child elements

None.

Example

```
<os>
    <type arch='s390x' machine='s390-virtio'>hvm</type>
    <kernel>/boot/vmlinuz-3.1.0-7.fc16.s390x</kernel>
    <initrd>/boot/initramfs-3.1.0-7.fc16.s390x.img</initrd>
    <cmdline>printk.time=1</cmdline>
</os>
```

<console>

Configures the host representation of the virtual server console.

Text content

None.

Selected attributes

type=pty
Configures a console which is accessible via PTY.

Usage

"Configuring the console" on page 70

Parent elements

"<devices>" on page 205

Child elements

- "<log>" on page 226
- <protocol>
- "<target> as child element of <console>" on page 254

Example

```
<devices>
    ...
    <console type="pty">
       <target type="sclp" port="0"/>
       <log file="/var/log/libvirt/qemu/vserv-cons0.log" append="off"/>
    </console>
<devices/>
```

<controller>

Specifies a device controller for a virtual server.

Text content

None.

Selected attributes

type=scsi | virtio-serial
: Specifies the type of controller.

index
: This decimal integer specifies the controller index, which is referenced by the attached host device.

  To reference a controller, use the controller attribute of the address element as child of the hostdev element.

scsi type-specific attributes:

  model=virtio-scsi
  : Optional; specifies the model of the controller.

Usage

"Configuring a SCSI tape or medium changer device" on page 90

Parent elements

"<devices>" on page 205

Child elements

- "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 192
- "<driver> as child element of <controller>" on page 209

Example

```
<devices>
   <controller type="scsi" model="virtio-scsi" index="0"/>
   <hostdev mode="subsystem" type="scsi">
      <source>
        <adapter name="scsi_host0"/>
        <address bus="0" target="0" unit="0"/>
      </source>
      <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
    </hostdev>
 </devices>
```

<cpu>

Specifies the features of the virtual CPUs of a virtual server.

Text content

None.

Selected attributes

match=exact
: The virtual CPU provided to the virtual server must match the specification. The virtual server can be started only if the specified CPU model is supported. This is the default.

mode= **custom** | **host-model**

custom
: The <cpu> element and its nested elements define the CPU to be presented to the virtual server. This mode ensures that a persistent virtual server uses the same CPU model on any KVM host. The virtual server can be started only if the KVM host supports the specified CPU model. This is the default.

host-model
: The CPU definition is derived from the KVM host CPU.

Usage

"Configuring the CPU model" on page 63

Parent elements

"<domain>" on page 208

Child elements

- "<model> as a child element of <cpu>" on page 232
- "<feature>" on page 213

Example

This example sets the CPU model to the default for z14:

```
<cpu mode="custom">
  <model>z14</model>
</cpu>
```

<cputune>

Groups CPU tuning parameters.

Text content

None.

Selected attributes

None.

Usage

"Tuning virtual CPUs" on page 62

Parent elements

"<domain>" on page 208

Child elements

"<shares>" on page 247

The use of the emulator_period, emulator_quota, period, and quota elements might affect the runtime behavior of the virtual server and interfere with the use of the shares element. Use the shares element for CPU tuning unless there is a specific need for the use of one of those elements.

Example

```
<domain>
    ...
    <cputune>

        <shares>2048</shares>

    </cputune>
    ...
</domain>
```

<device>

Specifies the device that stores a network file system or a logical volume backing a storage pool.

Text content

None.

Selected attributes

path

Usage

Chapter 11, "Configuring storage pools," on page 103

Parent elements

"<source> as child element of <pool>" on page 253

Child elements

None.

Example

```
<pool type="netfs">
    <name>nfspool01</name>
    <source>
        <format type="nfs"/>
        <host name="sandbox.example.com"/>
        <device path="/srv/nfsexport"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/nfspool01</path>
    </target>
</pool>
```

<devices>

Specifies the virtual network and block devices of the virtual server.

Text content

None.

Selected attributes

None.

Usage

Chapter 10, “Configuring devices,” on page 75

Parent elements

“<domain>” on page 208

Child elements

- “<console>” on page 200
- “<controller>” on page 201
- “<disk>” on page 207
- “<emulator>” on page 212
- “<hostdev>” on page 218
- “<interface>” on page 220
- “<memballoon>” on page 228
- “<watchdog>” on page 263

Example

```
<devices>
    <interface type="direct">
        <source dev="enccw0.0.1108" mode="bridge"/>
        <model type="virtio"/>
    </interface>

    <disk type="block" device="disk">
        <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
        <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
        <target dev="vdb" bus="virtio"/>
        <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x3c1b"/>
    </disk>
</devices>
```

<dhcp>

Configures DHCP services for the virtual network.

Text content

None.

Selected attributes

None.

Usage

Chapter 12, “Configuring virtual networks,” on page 105

Parent elements

“<ip>” on page 222

Child elements

<range>

Example

```
<network>
  ...
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

<disk>

Specifies a virtual block device, such as a SCSI device, or an image file.

Text content

None.

Selected attributes

type=block | file
Specifies the underlying disk source.

device=disk | cdrom
Optional; Indicates how the virtual block device is to be presented to the virtual server.

Usage

- Chapter 10, "Configuring devices," on page 75
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 95

Parent elements

"<devices>" on page 205

Child elements

- "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 192
- <blockio>
- "<boot>" on page 196
- "<driver> as child element of <disk>" on page 210
- "<geometry>" on page 216
- "<readonly>" on page 244
- "<shareable>" on page 246
- "<source> as child element of <disk>" on page 249
- "<target> as child element of <disk>" on page 255

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0009"/>
</disk>
```

<domain>

Is the root element of a domain configuration-XML.

Text content

None.

Selected attributes

type=kvm
Specifies the virtual server type.

Usage

"Domain configuration-XML" on page 51

Parent elements

None.

Child elements

- <clock>
- "<console>" on page 200
- "<controller>" on page 201
- "<cputune>" on page 203
- <currentMemory>
- "<devices>" on page 205
- "<iothreads>" on page 221
- <memory>
- <name>
- "<on_crash>" on page 237
- <on_poweroff>
- <on_reboot>
- <os>
- <uuid>
- "<vcpu>" on page 259

<driver> as child element of <controller>

Specifies details that are related to the user space process used to implement the controller.

Text content

None.

Selected attributes

iothread=<*IOthread-ID*>
Assigns a certain I/O thread to the user space process. Use this attribute to ensure best performance.

<*IOthread-ID*> is a value between 1 and the number of I/O threads which is specified by the iothreads element.

Usage

"Configuring a virtual HBA" on page 88

Parent elements

"<controller>" on page 201

Child elements

None.

Example

```
<domain>
    ...
    <iothreads>2</iothreads>
    ...
    <devices>
         <controller type="scsi" model="virtio-scsi" index="0">
            <driver iothread="2"/>
            ..
        </controller>
    </devices>
    ....
</domain>
```

<driver> as child element of <disk>

Specifies details that are related to the user space process used to implement the block device.

Text content

None.

Selected attributes

name=qemu
: Name of the user space process. Use "qemu".

type=raw | qcow2
: Use subtype "raw", except for qcow2 image files, which require the "qcow2" subtype.

iothread=<*IOthread-ID*>
: Assigns a certain I/O thread to the user space process. Use this attribute to ensure best performance.

  <*IOthread-ID*> is a value between 1 and the number of I/O threads which is specified by the iothreads element.

cache=none
: Optional; controls the cache mechanism.

error_policy=report | stop | ignore | enospace
: Optional; the error_policy attribute controls how the host will behave if a disk read or write error occurs.

rerror_policy=report | stop | ignore
: Optional; controls the 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. Therefore, if error_policy is set to "enospace" and no rerror_policy is given, the read error policy is left at its default ("report").

io=threads | native
: Optional; controls specific policies on I/O. For a better performance, specify "native".

ioeventfd=on | off
: Optional; allows users to set domain I/O asynchronous handling for the disk device. The default is left to the discretion of the host. Enabling this attribute allows QEMU to run the virtual server while a separate thread handles I/O. Typically virtual servers experiencing high system CPU utilization during I/O will benefit from this. On the other hand, on overloaded host it could increase virtual server I/O latency. **Note:** Only very experienced users should attempt to use this option!

event_idx=on | off
: Optional; controls some aspects of device event processing. If it is on, it will reduce the number of interrupts and exits for the virtual server. The default is determined by QEMU; usually if the feature is supported, the default is "on". If the situation occurs where this behavior is suboptimal, this attribute provides a way to force the feature "off". **Note:** Only experienced users should attempt to use this option!

Usage

- “Configuring a DASD or SCSI disk” on page 78
- “Configuring a virtual SCSI-attached CD/DVD drive” on page 95

Parent elements

“<disk>” on page 207

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xd501"/>
</disk>
```

<emulator>

Specifies the user space process.

Text content

Fully qualified path and file name of the user space process.

Selected attributes

None.

Usage

- "Configuring the user space" on page 68
- "Displaying the current libvirt-internal configuration" on page 122

Parent elements

"<devices>" on page 205

Child elements

None.

Example

```
<emulator>/usr/bin/qemu-system-s390x</emulator>
```

<feature>

Adds or removes a CPU feature in a CPU model specification.

Text content

None.

Selected attributes

name Specifies one of the features as shown in the output of the **`qemu-system-s390x -cpu help`** command.

policy= require | disable

require
Adds the feature to the CPU definition specified with the name attribute.

disable
Removes the feature from the CPU definition specified with the name attribute.

Usage

"Configuring the CPU model" on page 63

Parent elements

"<cpu>" on page 202

Child elements

None.

Example

This example uses the default for CPU model z14 as a starting point, but removes the `iep` feature:

```
<cpu mode="custom">
  <model>z14</model>
  <feature name="iep" policy="disable"/>
</cpu>
```

<format>

Specifies the image file format backing the storage pool volume.

Text content

None.

Selected attributes

type=raw | qcow2

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<target> as child element of <volume>” on page 257

Child elements

None.

Example

```
<volume type="file">
    <name>federico.img</name>
    <key>/var/lib/libvirt/images/federico.img</key>
    <target>
        <path>/var/lib/libvirt/images/federico.img</path>
        <format type="qcow2"/>
    </target>
</volume>
```

<forward>

Configures the forwarding mode for the bridge that connects the virtual network to a physical LAN. Omitting this tag results in an isolated network that can connect guests.

Text content

None.

Selected attributes

mode=nat | bridge | route
Specifies the method of forwarding for the LAN connection.

nat Configures a bridge with network address translation (NAT). All guest traffic to the physical network is routed through the host's routing stack and uses the host's public IP address. This mode supports only outbound traffic.

bridge Configures a bridge based on an already configured Open vSwitch (see “Preparing a virtual switch” on page 43).

route Configures a bridge with IP routing. Bridges with IP routing link to a virtual IP subnet on the host. Traffic to and from virtual servers that are connected to that subnet are then handled by the IP protocol.

Usage

Chapter 12, “Configuring virtual networks,” on page 105

Parent elements

“<network>” on page 236

Child elements

<nat>

Example

```
<network>
  <name>net0</name>
  <uuid>fec14861-35f0-4fd8-852b-5b70fdc112e3</uuid>
  <forward mode="nat">
    <nat>
      <port start="1024" end="65535"/>
    </nat>
  </forward>
  <bridge name="virbr0" stp="on" delay="0"/>
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

<geometry>

Overrides the geometry settings of DASDs or FC-attached SCSI disks.

Text content

None.

Selected attributes

cyls	Specifies the number of cylinders.
heads	Specifies the number of heads.
secs	Specifies the number of sectors per track.

Usage

"Configuring a DASD or SCSI disk" on page 78

Parent elements

"<disk>" on page 207

Child elements

None.

Example

```
<geometry cyls="16383" heads="16" secs="64" trans="lba"/>
```

<host>

Specifies the host that stores the network file system backing a storage pool.

Text content

None.

Selected attributes

name

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<source> as child element of <pool>” on page 253

Child elements

None.

Example

```
<pool type="netfs">
    <name>nfspool01</name>
    <source>
        <format type="nfs"/>
        <host name="sandbox.example.com"/>
        <device path="/srv/nfsexport"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/nfspool01</path>
    </target>
</pool>
```

<hostdev>

Passes host-attached devices to a virtual server.

Ensure that the device that is passed through to the virtual server is not in use by the host.

Text content

None.

Selected attributes

mode=subsystem
Specifies the pass-through mode.

type=scsi
Specifies the type of device that is assigned to a virtual server.

rawio=no | yes
Indicates whether the device needs raw I/O capability. If any device in a device configuration-XML file is specified in raw I/O mode, this capability is enabled for all such devices of the virtual server.

sgio=filtered | unfiltered
Indicates whether the kernel will filter unprivileged SG_IO commands for the device.

Usage

"Configuring a SCSI tape or medium changer device" on page 90

Parent elements

"<devices>" on page 205

Child elements

- "<address> as child element of <hostdev> or <disk>" on page 193
- "<readonly>" on page 244
- "<shareable>" on page 246
- "<source> as child element of <hostdev>" on page 251

Example

```
<devices>
   <controller type="scsi" model="virtio-scsi" index="0"/>
   <hostdev mode="subsystem" type="scsi">
      <source>
        <adapter name="scsi_host0"/>
        <address bus="0" target="0" unit="0"/>
      </source>
      <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
    </hostdev>
 </devices>
```

<initrd>

Specifies the fully qualified path of the ramdisk image in the host operating system.

Text content

Fully qualified path and file name of the initial ramdisk.

Selected attributes

None.

Usage

"Configuring a kernel image file as IPL device" on page 55

Parent elements

"<os>" on page 239

Child elements

None.

Example

```
<os>
    <type arch='s390x' machine='s390-virtio'>hvm</type>
    <kernel>/boot/vmlinuz-3.1.0-7.fc16.s390x</kernel>
    <initrd>/boot/initramfs-3.1.0-7.fc16.s390x.img</initrd>
    <cmdline>printk.time=1</cmdline>
</os>
```

<interface>

Specifies a virtual Ethernet device for a virtual server.

Text content

None.

Selected attributes

type = direct | bridge | network

Specifies the type of connection:

direct Creates a MacVTap interface.

bridge Attaches to a bridge, as for example implemented by a virtual switch.

network
Attaches to a virtual network as configured with a network configuration-XML.

trustGuestRxFilters = <u>no</u> | yes
Only valid if type = "direct".

Set this attribute to "yes" to allow the virtual server to change its MAC address. As a consequence, the virtual server can join multicast groups. The ability to join multicast groups is a prerequisite for the IPv6 Neighbor Discovery Protocol (NDP).

Setting trustGuestRxFilters to "yes" has security implications, because it allows the virtual server to change its MAC address and thus to receive all frames delivered to this address.

Usage

"Configuring virtual Ethernet devices" on page 98

Parent elements

"<devices>" on page 205

Child elements

- "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 192
- "<mac>" on page 227
- "<model> as a child element of <interface>" on page 233
- "<source> as child element of <interface>" on page 252
- "<virtualport> as a child element of <interface>" on page 260

Example

```
<interface type="direct">
    <source dev="bond0" mode="bridge"/>
    <model type="virtio"/>
</interface>
```

<iothreads>

Assigns threads that are dedicated to I/O operations on virtual block devices to a virtual server.

The use of I/O threads improves the performance of I/O operations of the virtual server. If this element is not specified, no I/O threads are provided.

Text content

Natural number specifying the number of threads.

Selected attributes

None.

Usage

"Configuring devices with the virtual server" on page 69

Parent elements

"<domain>" on page 208

Child elements

None.

Example

```
<iothreads>3</iothreads>
```

<ip>

Configures IP addresses for the virtual network.

Text content

None.

Selected attributes

address
: Sets the IP address for the bridge device. The value must be a valid IPv4 address.

netmask
: Specifies a subnet mask for the virtual network.

Usage

Chapter 12, “Configuring virtual networks,” on page 105

Parent elements

“<network>” on page 236

Child elements

“<dhcp>” on page 206

Example

```
<network>
  ...
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

<kernel>

Specifies the kernel image file.

Text content

Fully qualified path and file name of the kernel image file.

Selected attributes

None.

Usage

“Configuring a kernel image file as IPL device” on page 55

Parent elements

“<os>” on page 239

Child elements

None.

Example

```
<kernel>/boot/vmlinuz-3.9.3-60.x.20130605-s390xrhel</kernel>
```

<key>

Specifies the image file backing the volume.

Text content

Fully qualified path and file name of the image file backing the volume.

Selected attributes

None.

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<volume>” on page 262

Child elements

None.

Example

```
<volume type="file">
    <name>federico.img</name>
    <key>/var/lib/libvirt/images/federico.img</key>
    <target>
        <path>/var/lib/libvirt/images/federico.img</path>
        <format type="qcow2"/>
    </target>
</volume>
```

<keywrap>

Groups the configuration of the AES and DEA/TDEA wrapping key generation.

The keywrap element must contain at least one cipher element.

Text content

None.

Selected attributes

None.

Usage

"Disabling protected key encryption" on page 72

Parent elements

"<domain>" on page 208

Child elements

"<cipher>" on page 198

Example

```
<domain type="kvm">
    ...
    <keywrap>
        <cipher name="aes" state="off"/>
    </keywrap>
    ...
</domain>
```

<log>

Specifies a log file which is associated with the virtual server console output.

Text content

None.

Selected attributes

file Specifies the fully qualified path and filename of the log file.

append=off | on
Specifies whether the information in the file is preserved (append="on") or overwritten (append="off") on a virtual server restart.

Usage

"Configuring the console" on page 70

Parent elements

"<console>" on page 200

Child elements

None.

Example

```
<devices>
    ...
    <console type="pty">
        <target type="sclp"/>
        <log file="/var/log/libvirt/qemu/vserv-cons0.log" append="off"/>
    </console>
</devices>
```

<mac>

Specifies a host network interface for a virtual server.

Text content

None.

Selected attributes

address
Specifies the mac address of the interface.

Usage

"Configuring virtual Ethernet devices" on page 98

Parent elements

"<interface>" on page 220

Child elements

None.

Example

```
<interface type='direct'>
    <mac address='02:10:10:f9:80:00'/>
    <model type='virtio'/>
</interface>
```

<memballoon>

Specifies memory balloon devices.

Text content

None.

Selected attributes

model=none
Suppresses the automatic creation of a default memory balloon device.

Usage

"Suppressing the automatic configuration of a default memory balloon device" on page 74

Parent elements

"<devices>" on page 205

Child elements

None.

Example

```
<memballoon model="none"/>
```

<memory>

Specifies the amount of memory allocated for a virtual server at boot time and configures the collection of QEMU core dumps.

Text content

Natural number specifying the amount of memory. The unit is specified with the unit attribute.

Selected attributes

dumpCore=on | off
Specifies whether the memory of a virtual server is included in a generated core dump.

on	Specifies that the virtual server memory is included.
off	Specifies that the virtual server memory is excluded.

unit=b | KB | k | KiB | MB | M | MiB | GB | G | GiB | TB | T | TiB
Specifies the units of memory used:

b	bytes
KB	kilobytes (1,000 bytes)
k or KiB	kibibytes (1024 bytes), the default
MB	megabytes (1,000,000 bytes)
M or MiB	mebibytes (1,048,576 bytes)
GB	gigabytes (1,000,000,000 bytes)
G or GiB	gibibytes (1,073,741,824 bytes)
TB	terabytes (1,000,000,000,000 bytes)
T or TiB	tebibytes (1,099,511,627,776 bytes

Usage

- “Configuring virtual memory” on page 65
- “Configuring the collection of QEMU core dumps” on page 67

Parent elements

“<domain>” on page 208

Child elements

None.

Example

This example:
- Configures 524,288 KB of virtual memory.

- Excludes the virtual memory from QEMU core dumps.

```
<memory dumpCore="off" unit="KB">524288</memory>
```

<memtune>

Groups memory tuning parameters.

Text content

None.

Selected attributes

None.

Usage

- Chapter 22, “Memory management,” on page 163
- “Tuning virtual memory” on page 65

Parent elements

“<domain>” on page 208

Child elements

“<soft_limit>” on page 248

Example

This example ...

```
<domain>
    ...
    <memtune>
        <soft_limit unit="M">128</soft_limit>
    </memtune>
</domain>
```

<model> as a child element of <cpu>

Specifies a CPU model.

Text content

CPU model as shown in the output of the **virsh domcapabilities** command. Eligible values are specified with <model> tags that have the attribute `useable="yes"`.

Example: This example identifies the value `z14` as an eligible CPU model.

```
<model usable="yes">z14</model>
```

Selected attributes

None.

Usage

"Configuring the CPU model" on page 63

Parent elements

"<cpu>" on page 202

Child elements

None.

Example

This example sets the CPU model to the default for z14:

```
<cpu mode="custom">
  <model>z14</model>
</cpu>
```

<model> as a child element of <interface>

Specifies the interface model type.

Text content

None.

Selected attributes

type=virtio
Specifies the interface model type virtio.

Usage

- “Configuring a MacVTap interface” on page 98
- “Configuring a virtual switch” on page 100

Parent elements

“<interface>” on page 220

Child elements

None.

Example

This example configures a virtio interface:

```
<interface type="direct">
    <source dev="enccw0.0.a100" mode="bridge"/>
    <model type="virtio"/>
</interface>
```

<name> as a child element of <domain>

Assigns a unique name to the virtual server.

Text content

Unique alphanumeric name for the virtual server.

Selected attributes

None.

Usage

"Domain configuration-XML" on page 51

Parent elements

"<domain>" on page 208

Child elements

None.

Example

```
<domain type="kvm">
    <name>Virtual_server_25</name>
    <uuid>12345678abcd12341234abcdefabcdef</uuid>
    ....
</domain>
```

On the virtual server, the name will display as follows:

```
[root@guest:] # cat /proc/sysinfo | grep VM
VM00 Name: Virtual_
VM00 Control Program: KVM/Linux
...
VM00 Extended Name: Virtual_server_25
VM00 UUID: 12345678abcd12341234abcdefabcdef
```

<name> as a child element of <network>

Assigns a short name to a virtual network.

Text content

Alphanumeric name for the virtual network. The name must be unique for the scope of the KVM host.

Selected attributes

None.

Usage

Chapter 12, “Configuring virtual networks,” on page 105

Parent elements

“<network>” on page 236

Child elements

None.

Example

```
<network>
  <name>net0</name>
  ...
</network>
```

<network>

Is the root element of a network configuration-XML.

Text content

None.

Selected attributes

None.

Usage

Chapter 12, “Configuring virtual networks,” on page 105

Parent elements

None.

Child elements

- “<bridge>” on page 197
- “<forward>” on page 215
- “<ip>” on page 222
- “<name> as a child element of <network>” on page 235
- <uuid>
- “<virtualport> as a child element of <network>” on page 261

Example

```
<network>
  <name>ovs</name>
  <forward mode="bridge"/>
  <bridge name="ovs-br0"/>
  <virtualport type="openvswitch"/>
</network>
```

<on_crash>

Configures the behavior of the virtual server in the crashed state.

Set to preserve to ensure that virtual server crashes are detected.

Text content

preserve
Preserves the crashed state.

Selected attributes

None.

Usage

"Domain configuration-XML" on page 51

Parent elements

"<domain>" on page 208

Child elements

None.

Example

```
<on_crash>preserve</on_crash>
```

<on_reboot>

Configures the behavior of the virtual server when it is rebooted.

See also “reboot” on page 330.

Text content

restart Terminates the virtual server using the **shutdown** command and then boots the guest using the previous libvirt-internal configuration without modifying it.

destroy
Terminates the virtual server using the **destroy** command and then boots the guest using the previous libvirt-internal configuration without modifying it.

Selected attributes

None.

Usage

“Domain configuration-XML” on page 51

Parent elements

“<domain>” on page 208

Child elements

None.

Example

```
<on_reboot>restart</on_reboot>
```

<os>

Groups the operating system parameters.

Text content

None.

Selected attributes

None.

Usage

“Domain configuration-XML” on page 51

Parent elements

“<domain>” on page 208

Child elements

- “<type>” on page 258
- “<kernel>” on page 223
- “<initrd>” on page 219
- “<cmdline>” on page 199

Example

```
<os>
    <type arch="s390x" machine="s390-ccw-virtio">hvm</type>
    <initrd>/boot/initramfs-3.9.3-60.x.20130605-s390xrhel.img</initrd>
    <kernel>/boot/vmlinuz-3.9.3-60.x.20130605-s390xrhel</kernel>
    <cmdline>rd.md=0 rd.lvm=0 LANG=en_US.UTF-8
             KEYTABLE=us SYSFONT=latarcyrheb-sun16 rd.luks=0
             root=/dev/disk/by-path/ccw-0.0.e714-part1
             rd.dm=0 selinux=0 CMMA=on
             crashkernel=128M plymouth.enable=0
    </cmdline>
</os>
```

<path> as child element of <pool><target>

Specifies the path to the device backing a storage pool.

Text content

The text content depends on the pool type:

dir | netfs
Specifies the fully qualified path of the host or network directory.

fs Specifies the device node of the disk or the partition.

Selected attributes

None.

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<target> as child element of <pool>” on page 256

Child elements

None.

Example

This example specifies a directory backing a storage pool of type directory:

```
<pool type="dir">
  <name>directoryPool</name>
  <target>
    <path>/var/lib/libvirt/images</path>
  </target>
</pool>
```

This example specifies an FC-attached SCSI disk backing a storage pool of type file system:

```
<pool type="fs">
    <name>fspool01</name>
    <source>
        <device path="/dev/s356001/fspool"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/fspool01</path>
    </target>
</pool>
```

<path> as child element of <volume><target>

Specifies the fully qualified path of the image file.

Text content

Selected attributes

None.

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<target> as child element of <volume>” on page 257

Child elements

None.

Example

```
<volume type="file">
    <name>federico.img</name>
    <key>/var/lib/libvirt/images/federico.img</key>
    <target>
        <path>/var/lib/libvirt/images/federico.img</path>
        <format type="qcow2"/>
    </target>
</volume>
```

<pool>

Is the root element of a storage pool configuration-XML.

Text content

None.

Selected attributes

type=dir | fs | netfs | logical
where

dir Specifies a directory. All image files located in this directory are volumes of the storage pool.

fs Specifies a file system. The file system may be located on a DASD or SCSI disk or on a disk partition. libvirt will mount the file system and make all image files contained in the file system available as volumes of the storage pool.

netfs Specifies a network file system, such as NFS or CIFS. libvirt will mount the file system and make all image files contained in the file system available as volumes of the storage pool.

logical
Specifies a volume group. Each logical volume of this volume group will be available as volume of the storage pool.

Usage

Chapter 11, "Configuring storage pools," on page 103

Parent elements

None.

Child elements

- <name>
- "<source> as child element of <pool>" on page 253
- "<target> as child element of <pool>" on page 256

Example

- This example configures a storage pool backed by a directory as shown in Figure 7 on page 14:

```
<pool type="dir">
    <name>directoryPool</name>
    <target>
        <path>/var/lib/libvirt/images</path>
    </target>
</pool>
```

- This example configures a storage pool backed by a file system:

```
<pool type="fs">
    <name>fspool01</name>
    <source>
        <device path="/dev/s356001/fspool"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/fspool01</path>
    </target>
</pool>
```

- This example configures a storage pool backed by a network file system:

```
<pool type="netfs">
    <name>nfspool01</name>
    <source>
        <format type="nfs"/>
        <host name="sandbox.example.com"/>
        <device path="/srv/nfsexport"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/nfspool01</path>
    </target>
</pool>
```

- This example configures a storage pool backed by a volume group as shown in Figure 8 on page 14:

```
<pool type="logical">
    <name>lvPool01</name>
    <source>
        <name>lvpool01</name>
        <format type="lvm2"/>
    </source>
</pool>
```

<readonly>

Indicates that a device is readonly.

Text content

None.

Selected attributes

None.

Usage

"Configuring a virtual SCSI-attached CD/DVD drive" on page 95

Parent elements

- "<disk>" on page 207
- "<hostdev>" on page 218

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <readonly/>
</disk>
```

<rng>

Specifies a random number generator.

Text content

None.

Selected attributes

model=virtio
Specifies the random number generator device type.

Usage

"Configuring a random number generator" on page 102

Parent elements

"<devices>" on page 205

Child elements

"<backend>" on page 195

Example

```
<devices>
    ...
    <rng model="virtio">
        <backend model="random">/dev/random</backend>
    </rng>
    ...
</devices>
```

<shareable>

Indicates that a device can be shared between various virtual servers.

Text content

None.

Selected attributes

None.

Parent elements

- "<disk>" on page 207
- "<hostdev>" on page 218

Child elements

None.

Example

```
<devices>
    <controller type="scsi" model="virtio-scsi" index="0"/>
    <hostdev mode="subsystem" type="scsi">
        <source>
            <adapter name="scsi_host0"/>
            <address bus="0" target="0" unit="0"/>
        </source>
        <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
        <shareable/>
    </hostdev>
</devices>
```

<shares>

Specifies the initial CPU weight.

The CPU shares of a virtual server are calculated from the CPU weight of all virtual servers running on the host. For example, a virtual server that is configured with value 2048 gets twice as much CPU time as a virtual server that is configured with value 1024.

Text content

Natural number specifying the CPU weight.
- Valid values are in the natural numbers between 2 and 262144.
- The default value is 1024.

Selected attributes

None.

Usage
- "Tuning virtual CPUs" on page 62
- "CPU weight" on page 160

Parent elements

"<cputune>" on page 203

Child elements

None.

Example
```
<cputune>
   <shares>2048</shares>
</cputune>
```

<soft_limit>

Specifies a soft limit for the physical host memory requirements of the virtual server memory.

Text content

None.

Selected attributes

unit=b | KB | k | KiB | MB | M | MiB | GB | G | GiB | TB | T | TiB
Specifies the units of memory used:

b	bytes
KB	kilobytes (1,000 bytes)
k or KiB	kibibytes (1024 bytes), the default
MB	megabytes (1,000,000 bytes)
M or MiB	mebibytes (1,048,576 bytes)
GB	gigabytes (1,000,000,000 bytes)
G or GiB	gibibytes (1,073,741,824 bytes)
TB	terabytes (1,000,000,000,000 bytes)
T or TiB	tebibytes (1,099,511,627,776 bytes

Usage

- Chapter 22, “Memory management,” on page 163
- “Configuring virtual memory” on page 65

Parent elements

“<memtune>” on page 231

Child elements

None.

Example

This example configures a memory soft limit of 128 mebibytes:

```
<memtune>
    <soft_limit unit="M">128</soft_limit>
</memtune>
```

<source> as child element of <disk>

Specifies the host view of a device configuration.

Text content

None.

Selected attributes

dev Must be specified for disk type="block". Specifies a host device node of the block device.

file Must be specified for disk type="file". Specifies the fully qualified host file name.

pool Must be specified for disk type="volume". Specifies the name of the defined pool.

volume
Must be specified for disk type="volume". Specifies the name of the defined volume, which must be part of the specified pool.

startupPolicy=mandatory | requisite | optional
For disk type file that represents a CD or diskette, you may define a policy what to do with the disk if the source file is not accessible:

mandatory
fail if missing for any reason

requisite
fail if missing on boot up, drop if missing on migrate/restore/revert

optional
drop if missing at any start attempt

Usage

- "Configuring a DASD or SCSI disk" on page 78
- "Configuring an image file as storage device" on page 84
- "Configuring a volume as storage device" on page 86
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 95

Parent elements

"<disk>" on page 207

See also:
- "<source> as child element of <interface>" on page 252

Child elements

<seclabel>

Examples

- This example configures a SCSI disk as virtual block device:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
</disk>
```

- This example configures a file as virtual block device:

```
<disk type="file" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native"/>
    <source file="/var/lib/libvirt/images/disk.img"/>
    <target dev="vda1" bus="virtio"/>
</disk>
```

- This example configures a storage pool as virtual block device:

```
<disk type="volume" device="disk">
    <driver name="qemu" type="raw" io="native" cache="none"/>
    <source pool="blk-pool0" volume="blk-pool0-vol0"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0009"/>
</disk>
```

<source> as child element of <hostdev>

Specifies the host view of a host device configuration.

Text content

None.

Selected attributes

None.

Usage

"Configuring a SCSI tape or medium changer device" on page 90

Parent elements

"<hostdev>" on page 218

Child elements

- "<address> as child element of <source>" on page 194
- "<adapter> as child element of <source>" on page 191

Example

```
<devices>
   ...
   <hostdev mode="subsystem" type="scsi">
      <source>
         <adapter name="scsi_host0"/>
         <address bus="0" target="0" unit="0"/>
      </source>
      <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
   ...
</devices>
```

<source> as child element of <interface>

Specifies the host view of a network interface configuration.

Text content

None.

Selected attributes

dev Specifies the network interface.

mode=bridge | vepa
Optional and mutually exclusive with network; indicates whether packets are delivered to the target device or to the external bridge.

bridge If packets have a destination on the host from which they originated, they are delivered directly to the target. For direct delivery, both origin and destination devices need to be in bridge mode. If either the origin or destination is in vepa mode, a VEPA-capable bridge is required.

vepa All packets are sent to the external bridge. If packets have a destination on the host from which they originated, the VEPA-capable bridge will return the packets to the host.

| **network**
| Optional and mutually exclusive with mode; specifies the name of a virtual
| network.

Usage

"Configuring virtual Ethernet devices" on page 98

Parent elements

"<interface>" on page 220

Child elements

None.

Example

```
<interface type="direct">
    <source dev="bond0" mode="bridge"/>
    <model type="virtio"/>
</interface>
```

<source> as child element of <pool>

Specifies file system or network file system resources backing a storage pool.

Text content

None.

Selected attributes

None.

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<pool>” on page 242

Child elements

- <host>
- <device>

Example

```
<pool type="netfs">
    <name>nfspool01</name>
    <source>
        <format type="nfs"/>
        <host name="sandbox.example.com"/>
        <device path="/srv/nfsexport"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/nfspool01</path>
    </target>
</pool>
```

<target> as child element of <console>

Specifies the virtual server view of a console that is provided from the host.

Text content

None.

Selected attributes

type=virtio | sclp
Must be specified for the console.

virtio	Specifies a virtio console.
sclp	Specifies an SCLP console.

Usage

"Configuring the console" on page 70

Parent elements

"<console>" on page 200

See also:
- "<target> as child element of <disk>" on page 255

Child elements

None.

Example

```
<console type="pty">
    <target type="sclp"/>
</console>
```

<target> as child element of <disk>

Specifies the virtual server view of a device that is provided from the host.

Text content

None.

Selected attributes

dev Unique name for the device of the form `vd<x>`, where *<x>* can be one or more letters.

If no address element is specified, the order in which device bus-IDs are assigned to virtio block devices is determined by the order of the target dev attributes.

bus=virtio
Specifies the device type on the virtual server. Specify "virtio".

Usage

- "Configuring a DASD or SCSI disk" on page 78
- "Configuring an image file as storage device" on page 84
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 95

Parent elements

"<disk>" on page 207

See also: "<target> as child element of <console>" on page 254

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xa30e"/>
</disk>
```

<target> as child element of <pool>

Specifies the directory backing a storage pool.

Text content

None.

Selected attributes

None.

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<pool>” on page 242

Child elements

- “<path> as child element of <pool><target>” on page 240
- <permissions>

Example

```
<pool type="dir">
  <name>directoryPool</name>
  <target>
    <path>/var/lib/libvirt/images</path>
  </target>
</pool>
```

<target> as child element of <volume>

Specifies the image file backing a volume.

Text content

None.

Selected attributes

None.

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

“<volume>” on page 262

Child elements

- “<path> as child element of <volume><target>” on page 241
- “<format>” on page 214

Example

```
<volume type="file">
    <name>federico.img</name>
    <key>/var/lib/libvirt/images/federico.img</key>
    <target>
        <path>/var/lib/libvirt/images/federico.img</path>
        <format type="qcow2"/>
    </target>
</volume>
```

<type>

Specifies the machine type.

The use of this element is mandatory.

Text content

hvm Indicates that the operating system needs full virtualization.

Selected attributes

arch=s390x
Specifies the system architecture.

machine=s390-ccw-virtio | <*machine-type*>
Specifies the machine type. If you specify the alias machine type "s390-ccw-virtio", libvirt replaces this value by the current machine type, which depends on the installed QEMU release on the host or on the hypervisor release. Use this value unless you intend to migrate to a host with an earlier hypervisor release.

If you intend to migrate the virtual server to a destination host with earlier hypervisor release than the source host, specify the machine type reflecting this earlier release.

To display the available machine types, enter:

```
# qemu-kvm --machine help
```

Usage

- "Domain configuration-XML" on page 51
- "Definition of a virtual server on different hosts using the same configuration-XML" on page 126

Parent elements

"<os>" on page 239

Child elements

None.

Example

```
<type arch="s390x" machine="s390-ccw-virtio">hvm</type>
```

<vcpu>

Specifies the number of virtual CPUs for a virtual server.

Text content

Natural number specifying the maximum number of available virtual CPUs.

Selected attributes

current
Optional; specifies the number of virtual CPUs available at startup.

The value of the current attribute is limited by the maximum number of available virtual CPUs. If you do not specify the current attribute, the maximum number of virtual CPUs is available at startup.

Usage

"Configuring virtual CPUs" on page 61

Parent elements

"<domain>" on page 208

Child elements

None.

Example

```
<domain type="kvm">
    <name>vserv1</name>
    <memory>524288</memory>
    <vcpu current="2">5</vcpu>
    ....
</domain>
```

<virtualport> as a child element of <interface>

Specifies the type of a virtual switch.

Text content

None.

Selected attributes

type=openvswitch
Specifies the type of the virtual switch.

Usage

- "Configuring a virtual switch" on page 100

Parent elements

"<interface>" on page 220

Child elements

None.

Example

```
<interface>
   ...
   <virtualport type="openvswitch">
</interface>
```

<virtualport> as a child element of <network>

Identifies the bridge of a virtual network as an Open vSwitch.

Text content

None.

Selected attributes

type=openvswitch
: In combination with forwarding mode `bridge` and a <bridge> element, identifies the bridge as an Open vSwitch.

Usage

Chapter 12, "Configuring virtual networks," on page 105

Parent elements

"<network>" on page 236

Child elements

None.

Example

```
<network>
  <name>ovs</name>
  <forward mode="bridge"/>
  <bridge name="ovs-br0"/>
  <virtualport type="openvswitch"/>
</network>
```

<volume>

Is the root element of a volume configuration-XML.

Text content

None.

Selected attributes

type=file

Usage

Chapter 11, “Configuring storage pools,” on page 103

Parent elements

None.

Child elements

- <name>
- <key>
- <source>
- <target>

Example

```
<volume type="file">
    <name>federico.img</name>
    <key>/var/lib/libvirt/images/federico.img</key>
    <target>
        <path>/var/lib/libvirt/images/federico.img</path>
        <format type="qcow2"/>
    </target>
</volume>
```

<watchdog>

Specifies a watchdog device, which provides a guest watchdog application with access to a watchdog timer.

You can specify no more than one diag288 watchdog device. A watchdog device can be configured only as persistent device.

Text content

None.

Selected attributes

model=diag288
Specifies the diag288 watchdog device.

action=reset | poweroff | pause | dump | inject-nmi | none | shutdown
Optional; specifies an action that is automatically performed when the watchdog timer expires:

reset Default; immediately terminates the virtual server and restarts it afterwards.

poweroff
Immediately terminates the virtual server.

pause Suspends the virtual server.

dump Creates a virtual server dump on the host.

inject-nmi
Causes a restart interrupt for the virtual server including a dump on the virtual server, if it is configured respectively.

none Does not perform any command.

shutdown
Tries to properly shut down the virtual server.

Since the usage of this action assumes that the virtual server is not responding, it is unlikely that the virtual server will respond to the shutdown command. It is recommended not to use this action.

Usage

"Configuring a watchdog device" on page 71

Parent elements

"<devices>" on page 205

Child elements

None.

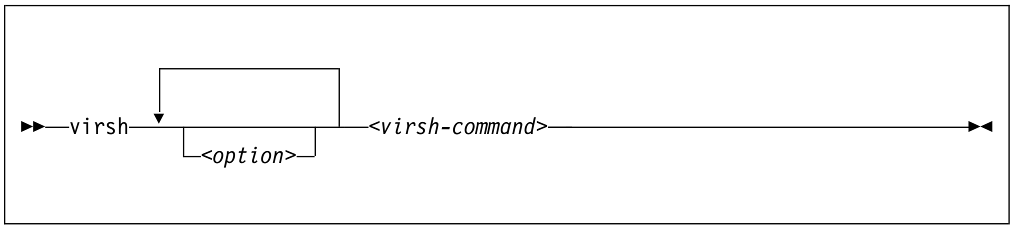
Example

```
<devices>
    ...
    <watchdog model="diag288" action="inject-nmi"/>
    ...
</devices>
```

Chapter 29. Selected virsh commands

These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Syntax



Where:

<*option*>
Is a command option.

<*VS*> Is the name, the ID, or the UUID of the virtual server.

<*virsh-command*>
Is a virsh command.

For a complete list of the virsh commands, see libvirt.org/virshcmdref.html.

<*XML-filename*>
Is the name of the XML file, which defines the device to be attached to the running virtual server.

Selected options

--help Displays the virsh online help.

--keepalive-interval <*interval-in-seconds*>
Sets an interval for sending keepalive messages to the virtual server to confirm the connection between the host and the virtual server. If the virtual server does not answer for a number of times which is defined by the `--keepalive-count` option, the host closes the connection. Setting the interval to 0 disables this mechanism. The default is 5 seconds.

--keepalive-count <*keepalive-count*>
Sets the number of times keepalive message can be sent without getting an answer from the virtual server without closing the connection. If the keepalive interval is set to 0, this option has no effect. The default is 6.

--version
Displays the installed libvirt version.

Selected virsh commands

These virsh commands are listed in the following chapters:
- "attach-device" on page 268
- "change-media" on page 270

- “console” on page 272
- “define” on page 273
- “destroy” on page 274
- “detach-device” on page 275
- “domblklist” on page 277
- “domblkstat” on page 278
- “domcapabilities” on page 280
- “domiflist” on page 281
- “domifstat” on page 282
- “dominfo” on page 283
- “domjobabort” on page 284
- “domstate” on page 285
- “dump” on page 286
- “dumpxml” on page 287
- “edit” on page 288
- “inject-nmi” on page 289
- “iothreadadd” on page 290
- “iothreaddel” on page 292
- “iothreadinfo” on page 294
- “list” on page 295
- “managedsave” on page 297
- “memtune” on page 299
- “migrate” on page 300
- “migrate-getspeed” on page 303
- “migrate-setmaxdowntime” on page 304
- “migrate-setspeed” on page 305
- “net-autostart” on page 306
- “net-define” on page 307
- “net-destroy” on page 308
- “net-dumpxml” on page 309
- “net-edit” on page 310
- “net-info” on page 311
- “net-list” on page 312
- “net-name” on page 313
- “net-start” on page 314
- “net-undefine” on page 315
- “net-uuid” on page 316
- “pool-autostart” on page 317
- “pool-define” on page 318
- “pool-delete” on page 319
- “pool-destroy” on page 320
- “pool-dumpxml” on page 321
- “pool-edit” on page 322
- “pool-info” on page 323
- “pool-list” on page 324

- “pool-name” on page 325
- “pool-refresh” on page 326
- “pool-start” on page 327
- “pool-undefine” on page 328
- “pool-uuid” on page 329
- “reboot” on page 330
- “resume” on page 331
- “schedinfo” on page 332
- “shutdown” on page 333
- “setvcpus” on page 334
- “start” on page 336
- “suspend” on page 338
- “undefine” on page 339
- “vcpucount” on page 340
- “vol-create” on page 341
- “vol-delete” on page 342
- “vol-dumpxml” on page 343
- “vol-info” on page 344
- “vol-key” on page 345
- “vol-list” on page 346
- “vol-name” on page 347
- “vol-path” on page 348
- “vol-pool” on page 349

Example

This example displays the virsh online help of the virsh **migrate** command:

```
# virsh help migrate
```

This example increases the keepalive interval of the connection to the host to 10 seconds during a live migration:

```
# virsh --keepalive-interval 10 migrate --live --persistent --undefinesource \
--timeout 1200 --verbose vserv1 qemu+ssh://kvmhost/system
```

attach-device

Attaches a device to a defined virtual server.

Syntax

```
                      ┌ --domain┐         ┌ --file┐
►►──attach-device─────┴─────────┴──<VS>───┴───────┴──<XML-filename>──────────►

   ┌ --live┐
►──┼───────┴─────────┬─────────────────────────────────────────────────────►◄
   │       └ --config┘
   ├ --current───────┤
   └ --persistent────┘
```

Where:

<VS> Is the name, the ID, or the UUID of the virtual server.

<XML-filename>
Is the name of the XML file, which defines the device to be attached to the running virtual server.

Selected options

--config
Persistently attaches the device to the virtual server with the next restart.

--current
Depending on the virtual server state:

running, paused
Attaches the device to the virtual server until it is detached or the virtual server is terminated.

shut off
Persistently attaches the device to the virtual server with the next restart.

--domain
Specifies the virtual server.

--file Specifies the device configuration-XML file.

--live Attaches the device to the running virtual server until it is detached or the virtual server is terminated.

--persistent
Depending on the virtual server state:

running, paused
Attaches the device to the virtual server.

The device remains persistently attached across restarts.

shut off
Persistently attaches the device to the virtual server with the next restart.

Usage

"Attaching a device" on page 146

Example

This example attaches the devices that are defined in device configuration-XML file dev1.xml to the virtual server vserv1.

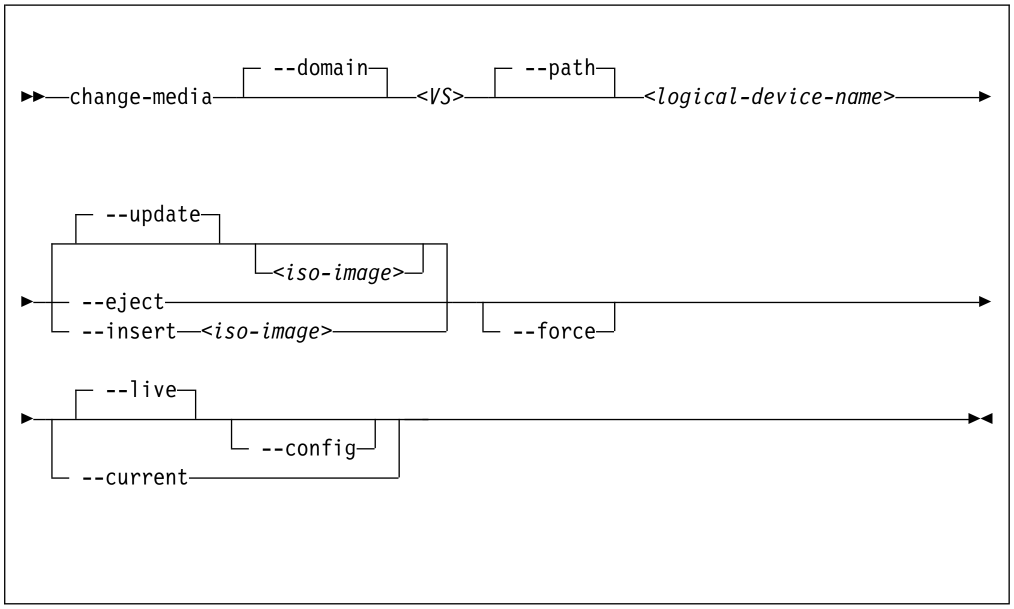
```
# virsh attach-device vserv1 dev1.xml
```

See also the example on page 123.

change-media

Removes a currently provided ISO image from a virtual SCSI-attached CD/DVD drive, or provides a different ISO image.

Syntax



Where:

<logical-device-name>
: Identifies the virtual SCSI-attached CD/DVD drive as specified with the target dev attribute in the domain configuration-XML file.

<iso-image>
: Is the fully qualified path to the ISO image on the host.

<VS> Is the name, ID or UUID of the virtual server.

Selected options

--config
: Persistently adds or removes the ISO image with the next virtual server restart.

--current
: Depending on the virtual server state:

  running, paused
  : Adds or removes the ISO image until the virtual server is terminated.

  shut off
  : Persistently removes the ISO image from the virtual server or provides a different one with the next restart.

--domain
: Specifies the virtual server.

--eject Removes the currently provided ISO image from the virtual SCSI-attached CD/DVD drive.

--force Forces the guest to release the file system residing on the virtual DVD, even if it is currently in use.

--insert
Provides a different ISO image for the virtual server.

--live Removes an ISO image from the running virtual server or provides an ISO image for a running virtual server until the virtual server is terminated.

--path Specifies the virtual SCSI-attached CD/DVD drive.

--update

If no ISO image is specified:
Removes the currently provided ISO image, just like the `--eject` option.

If an ISO image is specified:
Provides the specified ISO image. In case the current disk image has not been removed before, it is replaced by the new one.

Usage

"Replacing a virtual DVD" on page 148

Example

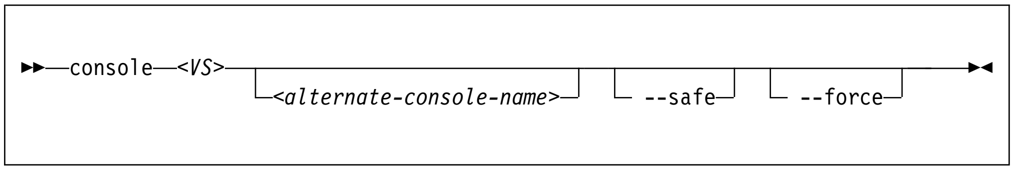
This command replaces the currently provided virtual DVD by a different one:

```
# virsh change-media vserv1 vdc -update /var/lib/libvirt/images/cd2.iso
Successfully inserted media.
```

console

Displays the console of a virtual server.

Syntax



Where:

<alternate-console-name>
: Is the device alias name of an alternative console that is configured for the virtual server.

<VS>
: Is the name, the ID, or the UUID of the virtual server.

Selected options

--force Disconnects any session in a case the connection is disrupted.

--safe Only connects to the console if the host ensures exclusive access to the console.

Usage

"Connecting to the console of a virtual server" on page 149

Example

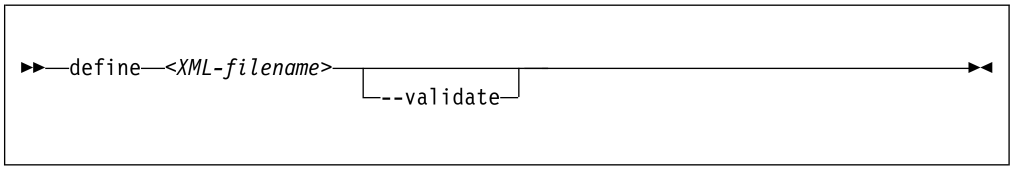
This example connects to the console of virtual server `vserv1`.

```
# virsh console vserv1
```

define

Creates a persistent virtual server definition.

Syntax



Where:

<XML-filename>
: Is the name of the domain configuration-XML file.

Selected options

--validate
: Validates the domain configuration-XML file against the XML schema.

Usage

- Chapter 1, “Overview,” on page 3
- “Defining a virtual server” on page 110

Example

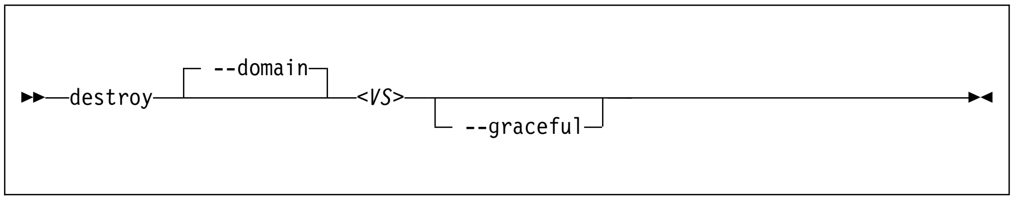
This example defines the virtual server, which is configured in domain configuration-XML file `vserv1.xml`.

```
# virsh define vserv1.xml
```

destroy

Immediately terminates a virtual server and releases any used resources.

Syntax



Where:

<VS> Is the name, the ID, or the UUID of the virtual server.

Selected options

--domain
Specifies the virtual server.

--graceful
Tries to properly terminate the virtual server, and only if it is not responding in a reasonable amount of time, it is forcefully terminated.

Virtual server state transitions

From State	To State (reason)
running	shut off (destroyed)
paused	shut off (destroyed)
crashed	shut off (destroyed)

Usage

"Terminating a virtual server" on page 114

Example

This example immediately terminates virtual server vserv1.

```
# virsh destroy vserv1
```

detach-device

Detaches a device from a defined virtual server.

Syntax

```
                         ┌─ --domain─┐       ┌─ --file─┐
►►──detach-device────────┴───────────┴─<VS>──┴─────────┴─<XML-filename>──────────►

   ┌─ --live─┐
►──┬─┴─────────┴───────────┬─────────────────────────────────────────────────────►◄
   │           └─ --config─┘ │
   ├─ --current──────────────┤
   └─ --persistent───────────┘
```

Where:

<*VS*> Is the name, the ID, or the UUID of the virtual server.

<*XML-filename*>
Is the name of the XML file, which defines the device to be detached from the running virtual server.

Selected options

--config
Persistently detaches the device with the next restart.

--current
Depending on the virtual server state:

running, paused
Immediately detaches the device from the virtual server.

If the device was attached persistently, it will be reattached with the next restart.

shut off
Persistently detaches the device from the virtual server with the next restart.

--domain
Specifies the virtual server.

--file Specifies the device configuration-XML file.

--live Detaches the device from the running virtual server.

--persistent
Depending on the virtual server state:

running, paused
Immediately detaches the device from the virtual server.

The device remains persistently detached across restarts.

shut off
Persistently detaches the device from the virtual server with the next restart.

Usage

"Detaching a device" on page 147

Example

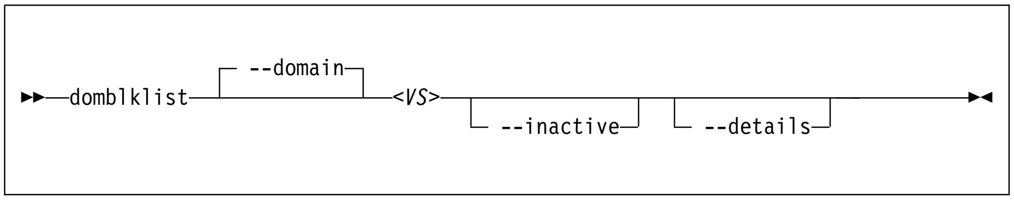
This example detaches the device that is defined in device configuration-XML file `vda.xml` from virtual server vserv1.

```
# virsh detach-device vserv1 vda.xml
```

domblklist

Displays information about the virtual block devices of a virtual server.

Syntax



Where:

<*VS*> Is the name, the ID, or the UUID of the virtual server.

Selected options

--details
Display details, such as device type and value.

--domain
Specifies the virtual server.

--inactive
Lists the block devices that will be used with the next virtual server reboot.

Usage

"Displaying information about a virtual server" on page 120

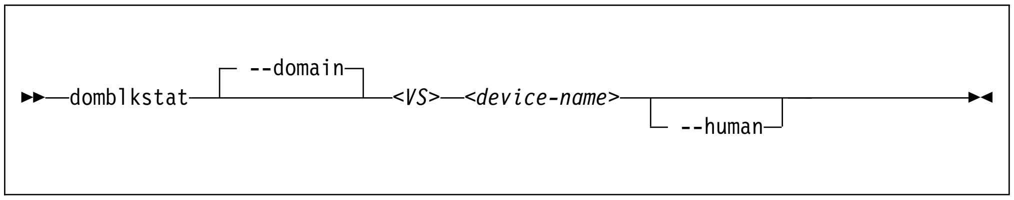
Example

```
#  virsh domblklist vserv1
Target     Source
------------------------------------------------
vda        /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023be
```

domblkstat

Displays status information about a virtual block device.

Syntax



Where:

<device-name>
: Is the name of the virtual block device.

<VS> Is the name, the ID, or the UUID of the virtual server.

Selected options

--domain
: Specifies the virtual server.

--human
: Replaces abbreviations by written-out information.

Usage

"Displaying information about a virtual server" on page 120

Example

Obtain the device names of the block devices of virtual server vserv1:

```
# virsh domblklist vserv1
Target     Source
------------------------------------------------
vda        /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023be
```

Obtain information about the virtual block device vda:

```
#  virsh domblkstat vserv1 vda
vda rd_req 20359
vda rd_bytes 235967488
vda wr_req 4134
vda wr_bytes 52682752
vda flush_operations 1330
vda rd_total_times 49294200385
vda wr_total_times 4403369039
vda flush_total_times 256032781
```

Alternatively, display written-out information:

```
# virsh domblkstat vserv vda --human
Device: vda
 number of read operations:      20359
 number of bytes read:           235967488
 number of write operations:     4348
 number of bytes written:        54353920
 number of flush operations:     1372
 total duration of reads (ns):   49294200385
 total duration of writes (ns):  4626108064
 total duration of flushes (ns): 265417103
```

domcapabilities

Prints an XML document that describes the hypervisor capabilities.

Syntax

```
►►──domcapabilities──┬─────────────────────────┬──────────────────────────────►◄
                     └─ --machine──<machine>──┘
```

Selected options

<machine>
: Is a supported hypervisor version as listed by the **`qemu-kvm --machine help`** command. If this option is omitted, the XML document for the default hypervisor version is displayed.

Usage

"Configuring the CPU model" on page 63

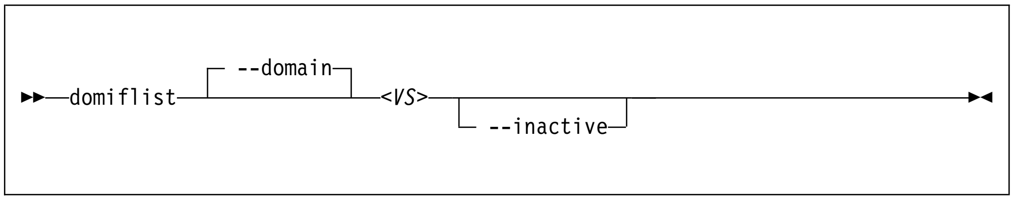
Example

```
# virsh domcapabilities --machine s390-ccw-virtio-2.8
<domainCapabilities>
  <path>/usr/bin/qemu-system-s390x</path>
  <domain>kvm</domain>
  <machine>s390-ccw-virtio-2.8</machine>
  <arch>s390x</arch>
  <vcpu max='248'/>
  ...
  <cpu>
    <mode name='host-passthrough' supported='yes'/>
    <mode name='host-model' supported='yes'>
      <model fallback='forbid'>z13s-base</model>
      <feature policy='require' name='aefsi'/>
      <feature policy='require' name='msa5'/>
      <feature policy='require' name='msa4'/>
      ...
    </mode>
    <mode name='custom' supported='yes'>
      <model usable='yes'>z10EC-base</model>
      <model usable='yes'>z9EC-base</model>
      <model usable='yes'>z196.2-base</model>
      ...
    </mode>
  </cpu>
  ...
</domainCapabilities>
```

domiflist

Displays network interface information for a running virtual server.

Syntax



Where:

<*VS*> Is the name, the ID, or the UUID of the virtual server.

Selected options

--domain
Specifies the virtual server.

--inactive
Lists the interfaces that will be used with the next virtual server reboot.

Usage

"Displaying information about a virtual server" on page 120

Example

```
#  virsh domiflist vserv1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet2      network    iedn       virtio      02:17:12:03:ff:01
```

domifstat

Displays network interface statistics for a running virtual server.

Syntax

```
                ┌─ --domain─┐
►►──domifstat───┴───────────┴──<VS>──<interface>──────────────────────►◄
```

Where:

<VS> Is the name, the ID, or the UUID of the virtual server.

<interface>
Is the name of the network interface as specified as target dev attribute in the configuration-XML file.

Selected options

--domain
Specifies the virtual server.

Usage

"Displaying information about a virtual server" on page 120

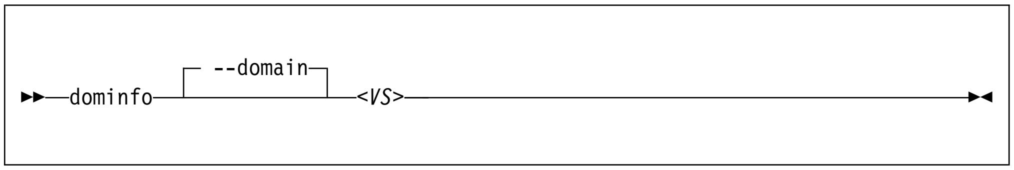
Example

```
# virsh domifstat vserv1 vnet0
vnet0 rx_bytes 7766280
vnet0 rx_packets 184904
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 5772
vnet0 tx_packets 130
vnet0 tx_errs 0
vnet0 tx_drop 0
```

dominfo

Displays information about a virtual server.

Syntax



Where:

<*VS*> Is the name, ID, or UUID of the virtual server.

Selected options

--domain
Specifies the virtual server.

Usage

"Displaying information about a virtual server" on page 120

Example

```
# virsh dominfo e20
Id:             55
Name:           e20
UUID:           65d6cee0-ca0a-d0c1-efc7-faacb8631497
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       1.2s
Max memory:     4194304 KiB
Used memory:    4194304 KiB
Persistent:     yes
Autostart:      enable
Managed save:   no
Security model: none
Security DOI:   0
```

domjobabort

Aborts the currently running virsh command related to the specified virtual server.

Syntax

```
                 ┌─ --domain─┐
►►──domjobabort──┴───────────┴──<VS>──────────────────────────────────►◄
```

Where:

<VS> Is the name, ID or UUID of the virtual server.

Selected options

None.

Usage

"Live virtual server migration" on page 127

Example

This example aborts the currently running dump request for vserv1.

```
# virsh dump vserv1 vserv1.txt
error: Failed to core dump domain vserv1 to vserv1.txt
error: operation aborted: domain core dump job: canceled by client

# virsh domjobabort vserv1
```

domstate

Displays the state of a virtual server.

Syntax

```
►►──domstate──<VS>──┬──────────┬──────────────────────────────────────►◄
                    └─ --reason─┘
```

Where:

<*VS*> Is the name, ID, or UUID of the virtual server.

Selected options

--reason
Displays information about the reason why the virtual server entered the current state.

Usage

"Displaying information about a virtual server" on page 120

Example

```
# virsh domstate vserv1
crashed
# virsh domstate vserv1 --reason
crashed (panicked)
```

dump

Creates a virtual server dump on the host.

Syntax

```
►►──dump──┬───────────────┬──<VS>──<filename>──────────────────────────►◄
          └─ --memory-only─┘
```

Where:

<VS> Is the name, ID, or UUID of the virtual server.

<filename>
Is the name of the target dump file.

Selected options

--memory-only
Issues ELF dumps, which can be inspected by using the **crash** command.

Usage

"Creating a virtual server dump on the host" on page 175

Example

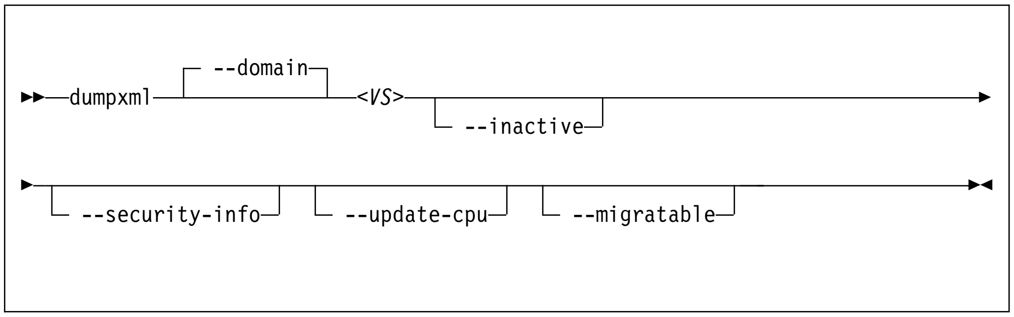
This example dumps the virtual server vserv1 to the file `dumpfile.name`.

```
# virsh dump --memory-only vserv1 dumpfile.name
```

dumpxml

Displays the current libvirt-internal configuration of a defined virtual server.

Syntax



Where:

<VS> Is the name, the ID, or the UUID of the virtual server.

Selected options

--domain
: Specifies the virtual server.

--migratable
: Displays a version of the current libvirt-internal configuration that is compatible with older libvirt releases.

--inactive
: Displays a defined virtual server, which is not in "running" state.

--security-info
: Includes security-sensitive information.

--update-cpu
: Updates the virtual server according to the host CPU.

Usage

"Displaying the current libvirt-internal configuration" on page 122

Example

This example displays the current domain configuration-XML of virtual server vserv1.

```
# virsh dumpxml vserv1
```

edit

Edits the libvirt-internal configuration of a virtual server.

Syntax

```
             ┌─ --domain─┐
►►──edit─────┴───────────┴──<VS>──────────────────────────────────────────►◄
```

Where:

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

--domain
Specifies the virtual server.

Usage

"Modifying a virtual server definition" on page 110

Example

This example edits the libvirt-internal configuration of virtual server vserv1.

```
# virsh edit vserv1
```

inject-nmi

Causes a restart interrupt for a virtual server including a dump on the virtual server, if it is configured respectively.

The dump is displayed in the virtual server file `/proc/vmcore`.

Syntax

```
►►──inject-nmi──<VS>──────────────────────────────────────────────────►◄
```

Where:

<VS> Is the name, the ID, or the UUID of the virtual server.

Selected options

None.

Usage

"Creating a dump on the virtual server" on page 175

Example

This example causes a restart interrupt for the virtual server vserv1 including a core dump.

```
# virsh inject-nmi vserv1
```

iothreadadd

Provides an additional I/O thread for a virtual server.

Syntax

```
                     ┌─ --domain─┐       ┌─ --id─┐
►►──iothreadadd──────┴───────────┴──<VS>─┴───────┴──<IOthread-ID>──────────────►

►──┬────────────┬──────────────────────────────────────────────────────────────►◄
   ├─ --config──┤
   ├─ --live────┤
   └─ --current─┘
```

Where:

<IOthread-ID>
: Is the ID of the I/O thread to be added to the virtual server. The I/O thread ID must be beyond the range of available I/O threads.

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

--config
: Affects the virtual server the next time it is restarted.

--current
: Affects the current virtual server.

--domain
: Specifies the virtual server.

--id Specifies the ID of the I/O thread that will be added to the I/O threads of the virtual server.

--live Affects the current virtual server only if it is running.

Usage

"Attaching a device" on page 146

Example

This example shows the **iothreadinfo** command for 8 virtual CPUs:

```
# virsh iothreadinfo vserv1
  IOThread ID     CPU Affinity
  ---------------------------------------------------
   1               0-7
   2               0-7
   3               0-7

# virsh iothreadadd vserv1 4

# virsh iothreadinfo vserv1
  IOThread ID     CPU Affinity
  ---------------------------------------------------
   1               0-7
   2               0-7
   3               0-7
   4               0-7
```

iothreaddel

Removes an I/O thread from a virtual server.

If the specified I/O thread is assigned to a virtual block device that belongs to the current configuration of the virtual server, it is not removed.

Syntax

```
                  ┌─ --domain─┐          ┌─ --id─┐
►►──iothreaddel───┴───────────┴──<VS>────┴───────┴──<IOthread-ID>──────────────►

►──┬─────────────┬─────────────────────────────────────────────────────────────►◄
   ├─ --config───┤
   ├─ --live─────┤
   └─ --current──┘
```

Where:

<IOthread-ID>
Is the ID of the I/O thread to be deleted from the virtual server.

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

--config
Affects the virtual server the next time it is restarted.

--current
Affects the current virtual server.

--domain
Specifies the virtual server.

--id Specifies the ID of the I/O thread that will be removed from the I/O threads of the virtual server.

--live Affects the current virtual server only if it is running.

Usage

"Detaching a device" on page 147

Example

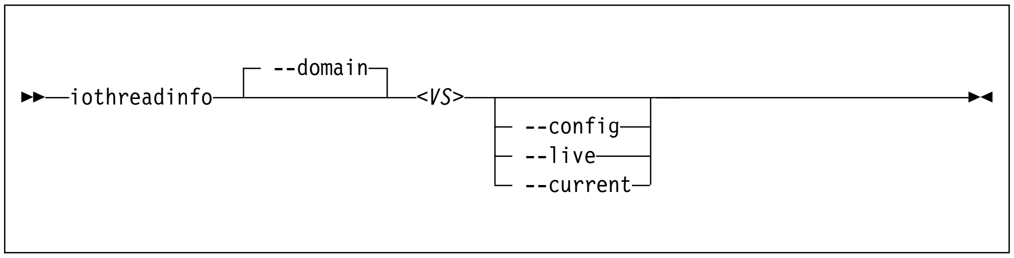
This example shows the **iothreadinfo** command for 8 virtual CPUs:

```
# virsh iothreadinfo vserv1
  IOThread ID     CPU Affinity
  ---------------------------------------------------
   1               0-7
   2               0-7
   3               0-7

# virsh iothreaddel vserv1 3

# virsh iothreadinfo vserv1
  IOThread ID     CPU Affinity
  ---------------------------------------------------
   1               0-7
   2               0-7
```

iothreadinfo

Displays information about the I/O threads of a virtual server.

Syntax



Where:

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

--config
Affects the virtual server the next time it is restarted.

--current
Affects the current virtual server.

--domain
Specifies the virtual server.

--live Affects the current virtual server only if it is running.

Usage

"Displaying information about a virtual server" on page 120

Example

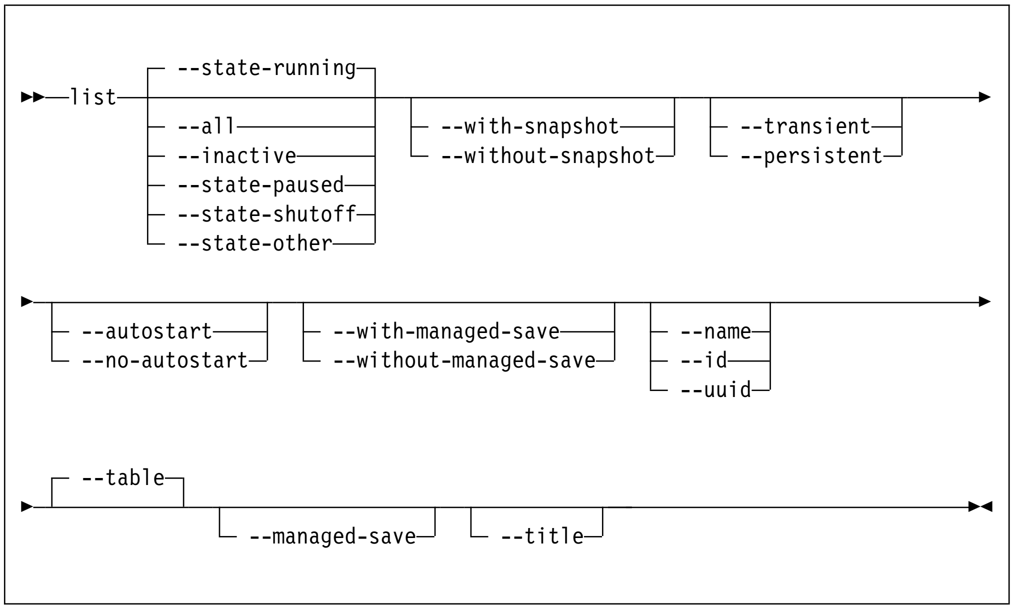
This example shows the **iothreadinfo** command for 8 virtual CPUs:

```
# virsh iothreadinfo vserv1
  IOThread ID     CPU Affinity
  ---------------------------------------------------
   1               0-7
   2               0-7
   3               0-7
```

list

Browses defined virtual servers.

Syntax



Selected options

--all Lists all defined virtual servers.

--autostart
Lists all defined virtual servers with autostart enabled.

--inactive
Lists all defined virtual servers that are not running.

--managed-save
Only when **--table** is specified.

--name
Lists only virtual server names.

--no-autostart
Lists only virtual servers with disabled autostart option.

--persistent
Lists persistent virtual servers.

--state-other
Lists virtual servers in state “shutting down”.

--state-paused
Lists virtual servers in state “paused”.

--state-running
Lists virtual servers in state “running”.

--state-shutoff
Lists virtual servers in state “shut off”.

--table Displays the listing as a table.

--title
Displays only a short virtual server description.

--transient
Lists transient virtual servers.

--uuid
Lists only UUIDs.

--with-managed-save
Lists virtual servers with managed save state.

--with-snapshot
Lists virtual servers with existing snapshot.

--without-managed-save
Lists virtual servers without managed save state.

--without-snapshot
Lists virtual servers without existing snapshot.

Usage

"Browsing virtual servers" on page 120

Example

This example lists all defined virtual servers.

```
# virsh list --all
```

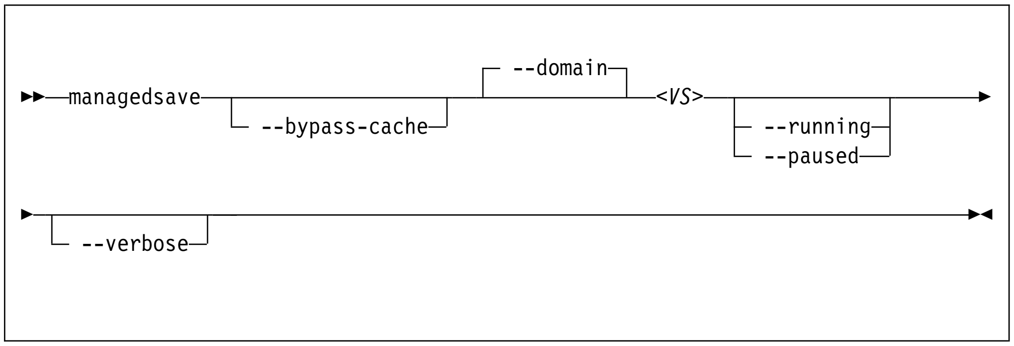
managedsave

Saves the system image of a running or a paused virtual server and terminates it thereafter. When the virtual server is started again, the saved system image is resumed.

Per default, the virtual server is in the same state as it was when it was terminated.

Use the **dominfo** command to see whether the system image of a shut off virtual server was saved.

Syntax



Where:

<*VS*> Is the name, ID, or UUID of the virtual server.

Selected options

--bypass-cache
Writes virtual server data directly to the disk bypassing the file system cache. This sacrifices write speed for data integrity by getting the data written to the disk faster.

--running
When you restart the virtual server, it will be running.

--paused
When you restart the virtual server, it will be paused.

--verbose
Displays the progress of the save operation.

Virtual server state transitions

Command option	From state	To state (reason)
`managedsave`	running	shut off (saved *from running*)
`managedsave`	paused	shut off (saved *from paused*)
`managedsave --running`	running	shut off (saved *from running*)
`managedsave --running`	paused	shut off (saved *from running*)
`managedsave --paused`	running	shut off (saved *from paused*)
`managedsave --paused`	paused	shut off (saved *from paused*)

Usage

- "Terminating a virtual server" on page 114
- Chapter 27, "Virtual server life cycle," on page 183

Example

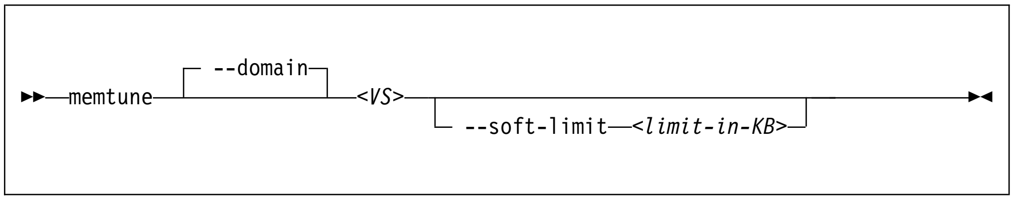
```
# virsh managedsave vserv1 --running
Domain vserv1 state saved by libvirt

# virsh dominfo vserv1
Id:             -
Name:           vserv1331
UUID:           d30a4c80-2670-543e-e73f-30c1fa7c9c20
OS Type:        hvm
State:          shut off
CPU(s):         2
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   yes
Security model: none
Security DOI:   0

# virsh start vserv1
Domain vserv1 started

# virsh list
 Id    Name                           State
----------------------------------------------------
 13    vserv1                        running
```

```
# virsh managedsave vserv1 --paused --verbose
Managedsave: [100 %]
Domain vserv1 state saved by libvirt

# virsh domstate vserv1
shut off

# virsh start vserv1
Domain vserv1 started

# virsh list
 Id    Name                           State
----------------------------------------------------
 13    vserv1                        paused
```

memtune

Specifies a soft limit for the physical host memory requirements of the virtual server memory.

Syntax



Where:

<limit-in-KB>
Is the minimum physical host memory in kilobytes remaining available for the virtual server memory in case the physical host memory resources are reduced.

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

--soft-limit
Specifies the minimum physical host memory remaining available for the virtual server in case the memory resources are reduced.

Note: Do not use the options `--hard-limit` and `--swap_hard_limit`. Their use might lead to a virtual server crash.

Usage

- Chapter 22, “Memory management,” on page 163
- “Managing virtual memory” on page 143

Example

This example allows the host to limit the physical host memory usage of vserv1 memory to 256 MB in case the host is under memory pressure:

```
# virsh memtune vserv1 --soft-limit 256000
```

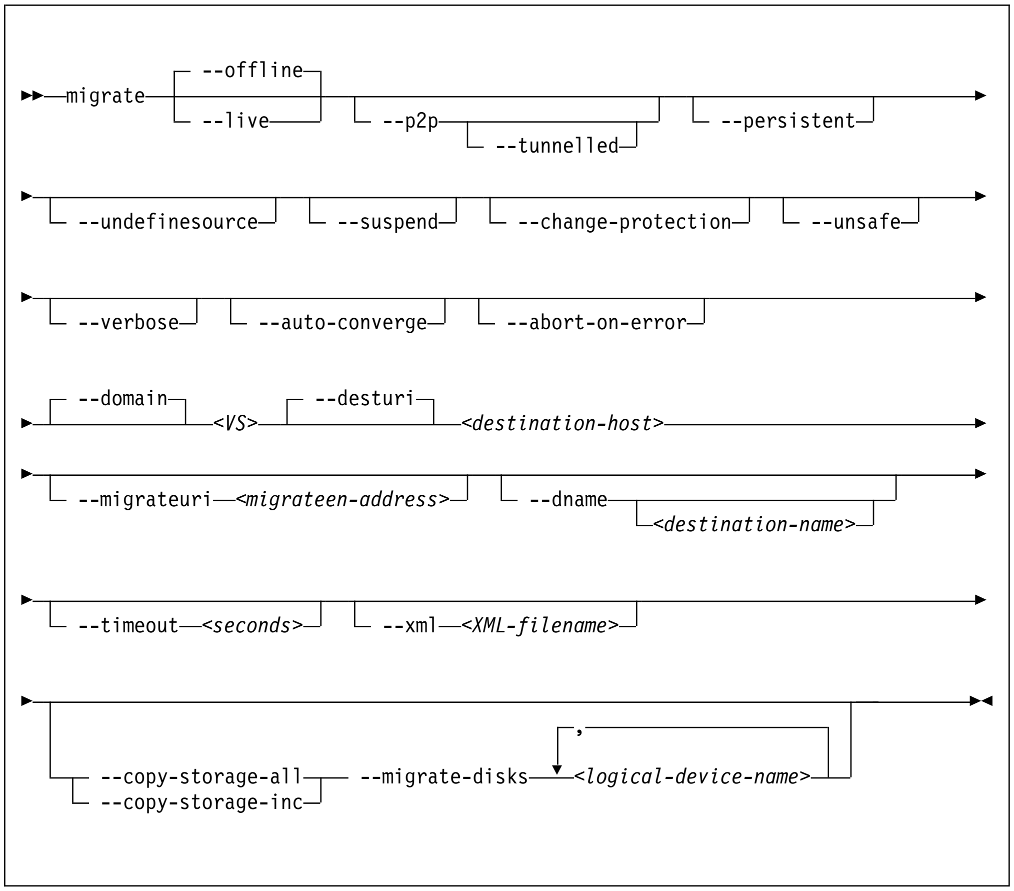
This example displays the memory tuning parameters of vserv1. Be sure not to modify the hard_limit and swap_hard_limit parameters.

```
# virsh memtune vserv1
hard_limit     : unlimited
soft_limit     : 256000
swap_hard_limit: unlimited
```

migrate

Migrates a virtual server to a different host.

Syntax



where

<destination-host>
: The libvirt connection URI of the destination host.

  Normal migration:
  : Specify the address of the destination host as seen from the virtual server.

  Peer to-peer migration:
  : Specify the address of the destination host as seen from the source host.

<destination-name>
: Is the new name of the virtual server on the destination host.

<logical-device-name>
: The logical device name of the virtual block device.

<migrateen-address>
: The host specific URI of the destination host.

<VS> Is the name, ID, or UUID of the virtual server.

<XML-filename>
: The domain configuration-XML for the source virtual server.

Selected options

--abort-on-error
: Causes an abort on soft errors during migration.

--auto-converge
: Forces auto convergence during live migration.

--change-protection
: Prevents any configuration changes to the virtual server until the migration ends

--copy-storage-all
: Copies image files that back up virtual block devices to the destination. Make sure that an image file with the same path and filename exists on the destination host before you issue the virsh **`migrate`** command. The regarding virtual block devices are specified by the `--migrate-disks` option.

--copy-storage-inc
: Incrementally copies non-readonly image files that back up virtual block devices to the destination. Make sure that an image file with the same path and filename exists on the destination host before you issue the virsh **`migrate`** command. The regarding virtual block devices are specified by the `--migrate-disks` option.

--dname
: Specifies that the virtual server is renamed during migration (if supported).

--domain
: Specifies the virtual server.

--live
: Specifies the migration of a running or a paused virtual server.

--migrate-disks
: Copies the files which back up the specified virtual block devices to the destination host. Use the `--copy-storage-all` or the `--copy-storage-inc` option in conjunction with this option. The regarding files must be writable. Please note that virtual DVDs are read-only disks. If in doubt, check your domain configuration-XML. If the disk device attribute of a disk element is configured as cdrom, or contains a readonly element, then the disk cannot be migrated.

--migrateuri
: Specifies the host specific URI of the destination host.

  If not specified, libvirt automatically processes the host specific URI from the libvirt connection URI. In some cases, it is useful to specify a destination network interface or port manually.

--offline
: Specifies the migration of the virtual server in “shut off” state. A copy of the libvirt-internal configuration of the virtual server on the source host is defined on the destination host.

  If you specify this option, specify the `--persistent` option, too.

--persistent
: Specifies to persistent the virtual server on the destination system.

--p2p Specifies peer-to-peer migration:

libvirt establishes a connection from the source to the destination host and controls the migration process. The migration continues even if virsh crashes or loses the connection.

Without the `--p2p` option, virsh handles the communication between the source and the destination host.

--suspend
Specifies that the virtual server will not be restarted on the destination system.

--timeout *seconds*
The number of seconds allowed before the virtual server is suspended while live migration continues.

--tunnelled
Specifies a tunneled migration:

libvirt pipes the migration data through the libvirtd communication socket. Thus, no extra ports are required to be opened on the destination host. This simplifies the networking setup required for migration.

The tunneled migration has a slight performance impact, because the data is copied between the libvirt daemons of the source host and the destination host.

Nevertheless, also in a tunneled migration, disk migration requires one extra destination port per disk.

--undefinesource
Specifies to undefine the virtual server on the source system.

--unsafe
Forces a migration even if it may cause data loss or corruption on the virtual server.

--verbose
Displays messages which indicate the migration progress.

Usage

"Live virtual server migration" on page 127

Example

This example migrates the virtual server vserv1 to the host zhost.

```
# virsh migrate --auto-converge --timeout 300 vserv1 qemu+ssh://zhost/system
```

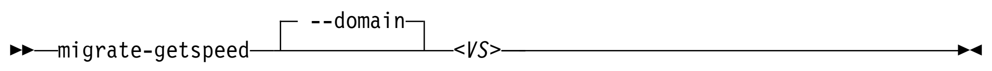
More information

libvirt.org/migration.html

migrate-getspeed

Displays the maximum migration bandwidth for a virtual server in MiB/s.

Syntax



Where:

<*VS*> Is the name, ID or UUID of the virtual server.

Selected options

None.

Usage

"Live virtual server migration" on page 127

Example

```
# virsh migrate-getspeed vserv1
8796093022207
```

migrate-setmaxdowntime

Specifies a tolerable downtime for the virtual server during the migration, which is used to estimate the point in time when to suspend it.

Syntax

```
                             ┌─ --domain─┐
►►──migrate-setmaxdowntime───┴───────────┴──<VS>──────────────────────────────►

   ┌─ --downtime─┐
►──┴─────────────┴──<milliseconds>───────────────────────────────────────────►◄
```

where

<milliseconds>
: Is the tolerable downtime of the virtual server during migration in milliseconds.

<VS>
: Is the name, ID, or UUID of the virtual server.

Selected options

None.

Usage

"Live virtual server migration" on page 127

Example

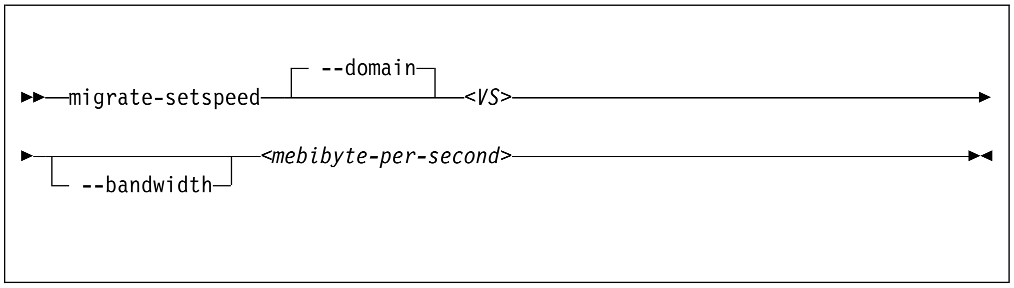
This example specifies a tolerable downtime of 100 milliseconds for the virtual server vserv1 in case it is migrated to another host.

```
# virsh migrate-setmaxdowntime vserv1 --downtime 100
```

migrate-setspeed

Sets the maximum migration bandwidth for a virtual server in MiB/s.

Syntax



Where:

<mebibyte-per-second>
Is the migration bandwidth limit in MiB/s.

<*VS*> Is the name, ID or UUID of the virtual server.

Selected options

--bandwidth
Sets the bandwidth limit during a migration in MiB/s.

Usage

"Live virtual server migration" on page 127

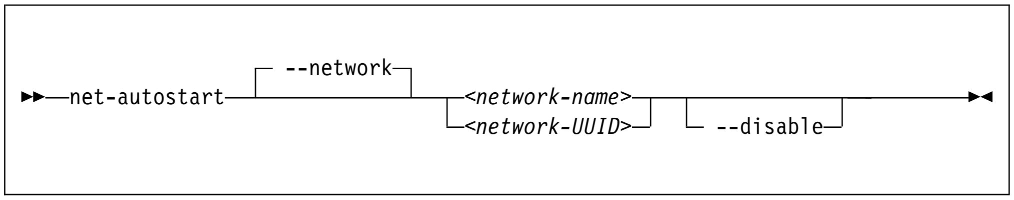
Example

```
# virsh migrate-setspeed vserv1 --bandwidth 100
# virsh migrate-getspeed vserv1
100
```

net-autostart

Enables or disables the automatic start of a virtual network when the libvirt daemon is started.

Syntax



Where:

<network-name>
Is the name of the virtual network.

<network-UUID>
Is the UUID of the virtual network.

Selected options

--network
Specifies the virtual network.

--disable
Disables the automatic start of the virtual network when the libvirt daemon is started.

Usage

Chapter 20, "Managing virtual networks," on page 155

Example

This example configures the automatic start of virtual network `net0` when the libvirt daemon is started.

```
# virsh net-autostart net0
```

net-define

Creates a persistent definition of a virtual network.

Syntax

```
                     ┌─ --file─┐
►►──net-define───────┴─────────┴──<XML-filename>──────────────────────────────►◄
```

Where:

<XML-filename>
: Is the name of the network configuration-XML file.

Selected options

--file specifies the network configuration-XML file.

Usage

Chapter 20, "Managing virtual networks," on page 155

Example

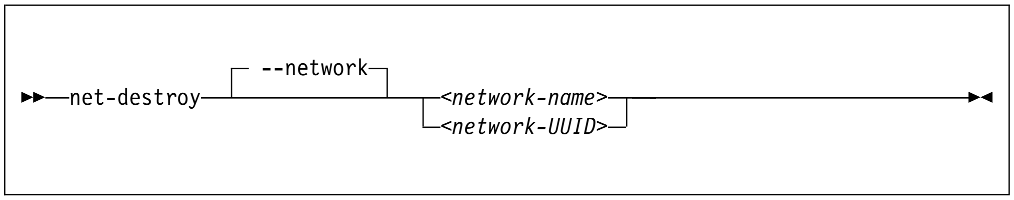
This example defines the virtual network that is configured by the `net0.xml` network configuration-XML file.

```
# virsh net-define net0.xml
```

net-destroy

Deactivates an active virtual network.

Syntax



Where:

<network-name>
Is the name of the virtual network.

<network-UUID>
Is the UUID of the virtual network.

Selected options

--network
Specifies the virtual network.

Usage

Chapter 20, "Managing virtual networks," on page 155

Example

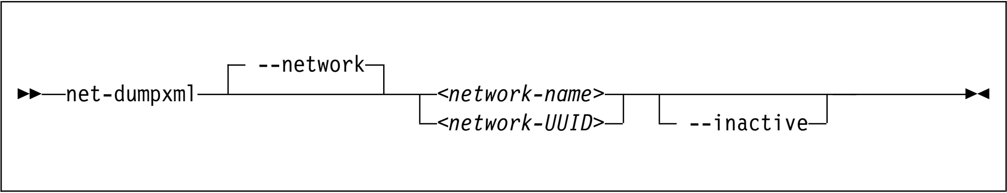
This example shuts down the virtual network with name `net0`.

```
# virsh net-destroy net0
```

net-dumpxml

Displays the current configuration of a virtual network.

Syntax



Where:

<network-name>
: Is the name of the virtual network.

<network-UUID>
: Is the UUID of the virtual network.

Selected options

--network
: Specifies the virtual network.

--inactive
: Displays the network XML without the automatic expansions in the libvirt-internal representation.

Usage

Chapter 20, "Managing virtual networks," on page 155

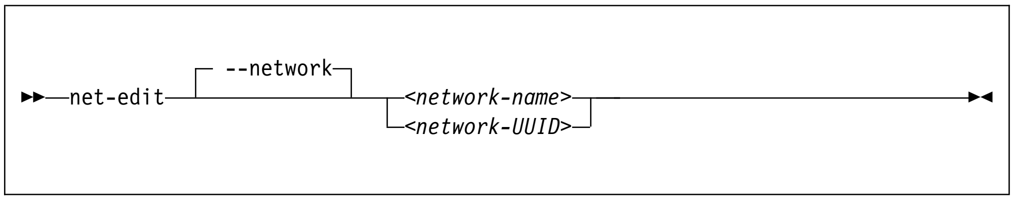
Example

```
# virsh net-dumpxml net0
<network>
  <name>net0</name>
  <uuid>fec14861-35f0-4fd8-852b-5b70fdc112e3</uuid>
  <forward mode="nat">
    <nat>
      <port start="1024" end="65535"/>
    </nat>
  </forward>
  <bridge name="virbr0" stp="on" delay="0"/>
  <mac address="aa:25:9e:d9:55:13"/>
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

net-edit

Edits the configuration of a defined virtual network. When the update is saved, both the libvirt-internal configuration and the Network configuration-XML are updated.

Syntax



Where:

<network-name>
: Is the name of the virtual network.

<network-UUID>
: Is the UUID of the virtual network.

Selected options

--network
: Specifies the virtual network.

Usage

Chapter 20, "Managing virtual networks," on page 155

Example

This example edits the configuration of `net0`.

```
# virsh net-edit net0
```

net-info

Displays information about a defined virtual network.

Syntax

```
                 ┌─ --network─┐
►►──net-info─────┴────────────┴──┬─<network-name>─┬─────────────────────────►◄
                                 └─<network-UUID>─┘
```

Where:

<network-name>
: Is the name of the virtual network.

<network-UUID>
: Is the UUID of the virtual network.

Selected options

--network
: Specifies the virtual network.

Usage

Chapter 20, “Managing virtual networks,” on page 155

Example

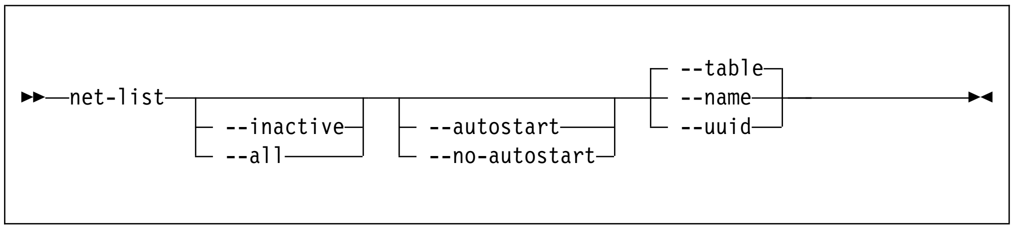
```
# virsh net-info net0
Name:           net0
UUID:           fec14861-35f0-4fd8-852b-5b70fdc112e3
Active:         yes
Persistent:     yes
Autostart:      yes
Bridge:         virbr0
```

net-list

Displays a list of defined virtual networks.

By default, a list of active virtual networks is displayed.

Syntax



Selected options

--all Displays active and inactive virtual networks.

--autostart
Displays only virtual networks that start automatically when the libvirt daemon is started.

--inactive
Displays only inactive virtual networks.

--name
Lists the network names instead of displaying a table of virtual networks.

--no-autostart
Displays only virtual networks that do not start automatically when the libvirt daemon is started.

--table Displays the virtual network information in table format.

--uuid Lists the virtual network UUIDs instead of displaying a table of virtual networks.

Usage

Chapter 20, “Managing virtual networks,” on page 155

Example

```
# virsh net-list
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
 net0                 active     no            yes
```

net-name

Displays the name of a virtual network that is specified with its UUID.

Syntax

```
                 ┌─ --network─┐
►►──net-name─────┴────────────┴──<network-UUID>──────────────────────────────►◄
```

Where:

<network-UUID>
: Is the UUID of the virtual network.

Selected options

--network
: Specifies the virtual network.

Usage

Chapter 20, "Managing virtual networks," on page 155

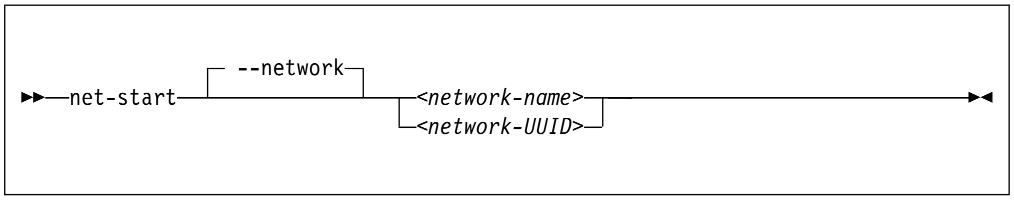
Example

```
# virsh net-name fec14861-35f0-4fd8-852b-5b70fdc112e3
net0
```

net-start

Activates a defined, inactive virtual network.

Syntax



Where:

<network-name>
Is the name of the virtual network.

<network-UUID>
Is the UUID of the virtual network.

Selected options

--network
Specifies the virtual network.

Usage

Chapter 20, "Managing virtual networks," on page 155

Example

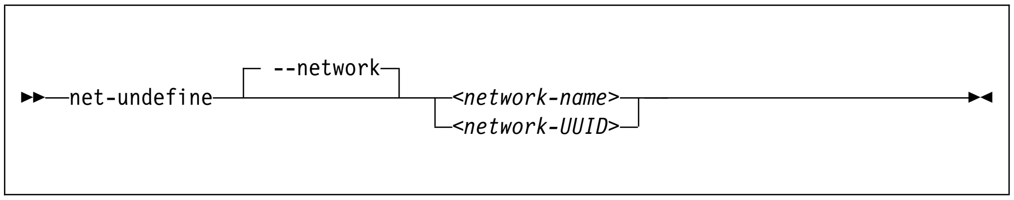
This example starts the virtual network with the name `net0`.

```
# virsh net-start net0
```

net-undefine

Deletes the persistent libvirt definition of a virtual network.

Syntax



Where:

<network-name>
: Is the name of the virtual network.

<network-UUID>
: Is the UUID of the virtual network.

Selected options

--network
: Specifies the virtual network.

Usage

Chapter 20, "Managing virtual networks," on page 155

Example

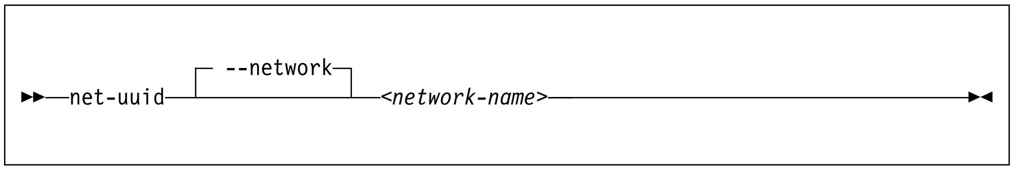
This example removes the virtual network with name `net0` from the libvirt definition.

```
# virsh net-undefine net0
```

net-uuid

Displays the UUID of a virtual network that is specified with its name.

Syntax



Where:

<network-name>
: Is the name of the virtual network.

Selected options

--network
: Specifies the virtual network.

Usage

Chapter 20, “Managing virtual networks,” on page 155

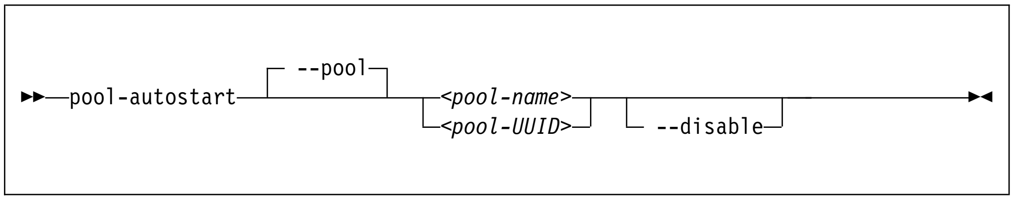
Example

```
# virsh net-uuid net0
fec14861-35f0-4fd8-852b-5b70fdc112e3
```

pool-autostart

Enables or disables the automatic start of a storage pool when the libvirt daemon is started.

Syntax



Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

--disable
: Disables the automatic start of the storage pool when the libvirt daemon is started.

Usage

Chapter 19, “Managing storage pools,” on page 151

Example

This example specifies the automatic start of storage pool `pool1` when the libvirt daemon is started.

```
# virsh pool-autostart pool1
```

pool-define

Creates a persistent definition of a storage pool configuration.

Syntax

```
                      ┌─ --file─┐
►►──pool-define───────┴─────────┴──<XML-filename>──────────────────────────►◄
```

Where:

<XML-filename>
Is the name of the storage pool configuration-XML file.

Selected options

--file Specifies the storage pool configuration-XML file.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

This example defines the storage pool that is configured by the storage pool configuration-XML file named `pool1.xml`.

```
# virsh pool-define pool1.xml
```

pool-delete

Deletes the volumes of a storage pool.

Syntax

Attention: This command is intended for expert users. Depending on the pool type, the results range from no effect to loss of data. In particular, data is lost when a zfs or LVM group pool is deleted.

```
                   ┌ --pool┐
►►──pool-delete────┴───────┴──┬─<pool-name>─┬─────────────────────────────►◄
                              └─<pool-UUID>─┘
```

Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

This example deletes the volumes of storage pool `pool1`.

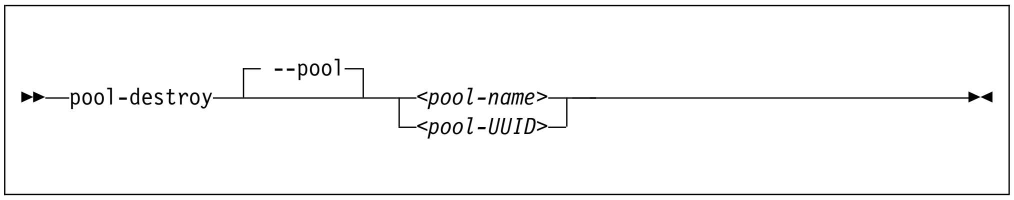
```
# virsh pool-delete pool1
```

pool-destroy

Shut down a storage pool.

The pool can be restarted by using the virsh **`pool-start`** command.

Syntax



Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Selected options

None.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

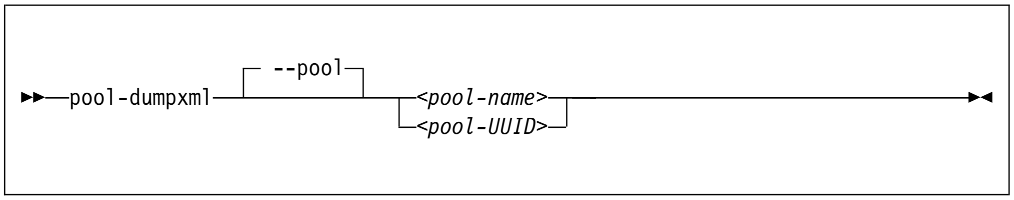
This example shuts down storage pool `pool1`.

```
# virsh pool-destroy pool1
```

pool-dumpxml

Displays the current libvirt-internal configuration of a storage pool.

Syntax



Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, “Managing storage pools,” on page 151

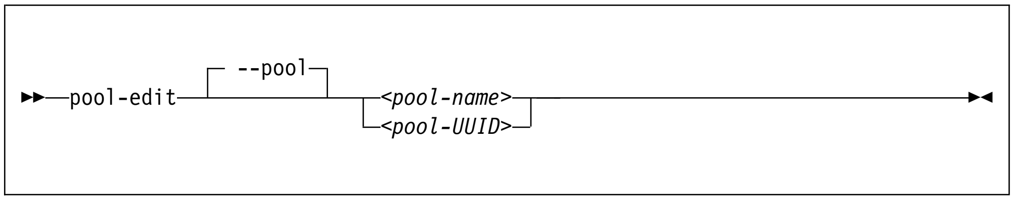
Example

```
# virsh pool-dumpxml pool1 default
<pool type="dir">
  <name>default</name>
  <uuid>09382b31-03ac-6726-45be-dfcaaf7b01cc</uuid>
  <capacity unit="bytes">243524067328</capacity>
  <allocation unit="bytes">109275693056</allocation>
  <available unit="bytes">134248374272</available>
  <source>
  </source>
  <target>
    <path>/var/lib/libvirt/images</path>
    <permissions>
      <mode>0711</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>
```

pool-edit

Edits the libvirt-internal configuration of a defined storage pool.

Syntax



Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, “Managing storage pools,” on page 151

Example

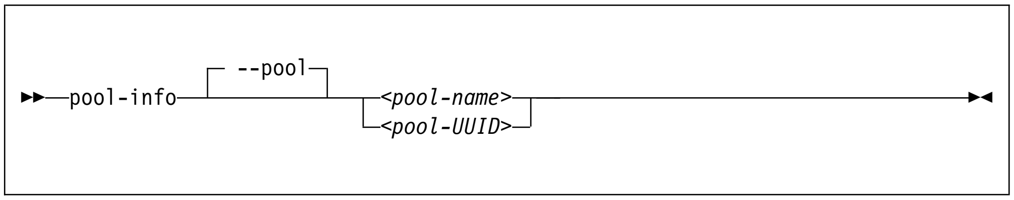
This example edits the libvirt-internal configuration of pool1.xml.

```
# virsh pool-edit pool1
```

pool-info

Displays information about a defined storage pool.

Syntax



Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

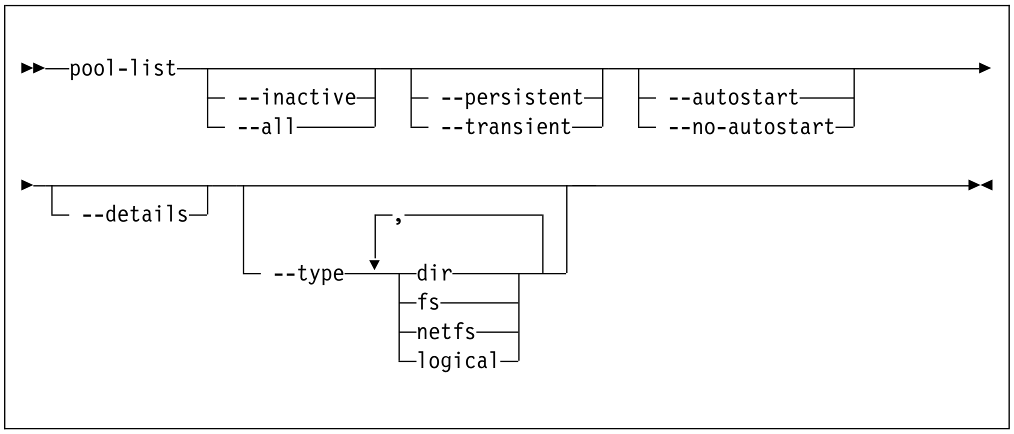
```
# virsh pool-info pool1
```

pool-list

Displays a list of defined storage pools.

By default, a list of active storage pools is displayed.

Syntax



Selected options

--all Displays all defined storage pools.

--autostart
Displays all storage pools that start automatically when the libvirt daemon is started.

--details
Displays pool persistence and capacity related information.

--inactive
Displays all inactive storage pools.

--no-autostart
Displays all storage pools that do not start automatically when the libvirt daemon is started.

--persistent
Displays all persistent storage pools.

--transient
Displays all transient storage pools.

--type Displays all storage pools of the specified types.

Usage

Chapter 19, “Managing storage pools,” on page 151

Example

```
# virsh pool-list
```

pool-name

Displays the name of a storage pool specified by its UUID.

Syntax

```
                  ┌─ --pool─┐
►►──pool-name─────┴─────────┴──<pool-UUID>──────────────────────────────────►◄
```

Where:

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, "Managing storage pools," on page 151

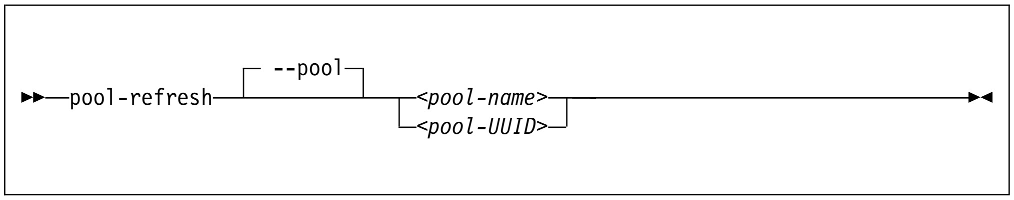
Example

```
# virsh pool-name bc403958-a355-4d3c-9d5d-872c16b205ca
```

pool-refresh

Updates the volume list of a storage pool.

Syntax



Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

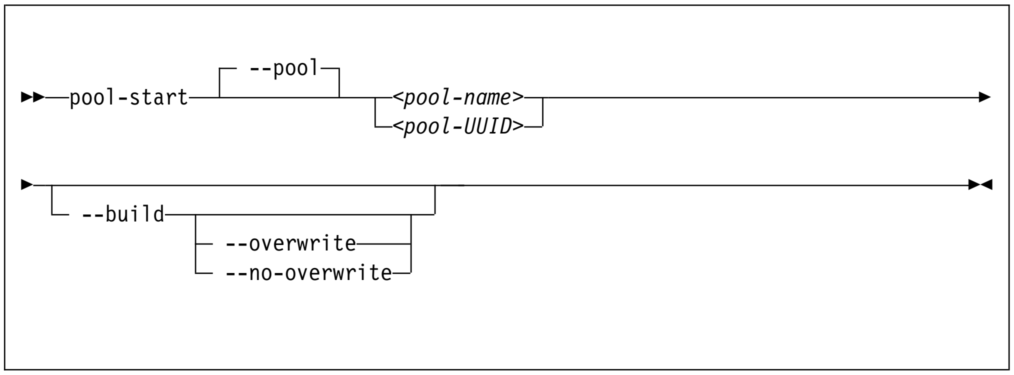
This example updates the list of volumes contained in storage pool `pool1`.

```
# virsh pool-refresh pool1
```

pool-start

Starts a defined inactive storage pool.

Syntax



Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

--build
: Creates the directory or the file system or creates the label for a disk (depending on the storage pool type) before starting the storage pool.

--overwrite
: Only valid for storage pools of type dir, fs, or netfs.

  Creates the directory, the file system or the label, overwriting existing ones.

--no-overwrite
: Only valid for storage pools of type dir, fs, or netfs.

  Creates the directory or the file system only if it does not exist. Returns an error if it does exist.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

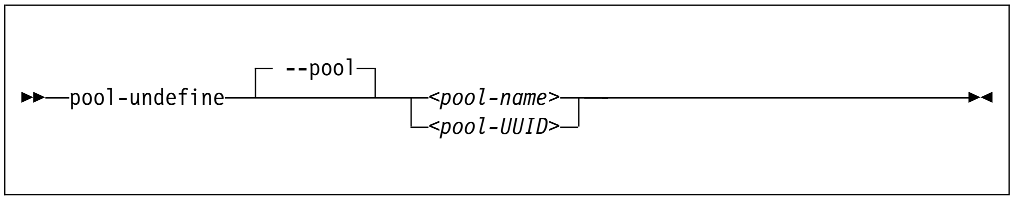
This example starts storage pool `pool1`.

```
# virsh pool-start pool1
```

pool-undefine

Deletes the persistent libvirt definition of a storage pool.

Syntax



Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

This example removes storage pool `pool1` from the libvirt definition.

```
# virsh pool-undefine pool1
```

pool-uuid

Displays the UUID of a storage pool specified by its name.

Syntax

```
                   ┌─ --pool─┐
►►──pool-uuid──────┴─────────┴──<pool-name>────────────────────────────────────►◄
```

Where:

<pool-name>
Is the name of the storage pool.

Selected options

--pool Specifies the storage pool.

Usage

Chapter 19, "Managing storage pools," on page 151

Example

```
# virsh pool-uuid pool1
```

reboot

Reboots a guest using the current libvirt-internal configuration.

For making virtual server configuration changes effective, shut down the virtual server and start it again instead of rebooting it.

The exact reboot behavior of a virtual server is configured by the on_reboot element in the domain configuration-XML (see “<on_reboot>” on page 238.

Syntax

```
►►──reboot──<VS>──────────────────────────────────────────────────────►◄
```

Where:

<VS> Is the name, ID, or UUID of the virtual server.

Virtual server state transition

If on_reboot is configured as “restart”:

From State	Transfer State (reason)	To State (reason)
running	shut off (shutdown)	running (booted)
paused	shut off (shutdown)	running (booted)

If on_reboot is configured as “destroy”:

From State	Transfer State (reason)	To State (reason)
running	shut off (destroyed)	running (booted)
paused	shut off (destroyed)	running (booted)

Example

```
# virsh reboot vserv1
Domain vserv1 is being rebooted
```

resume

Resumes a virtual server from the paused to the running state.

Syntax

```
▶▶──resume──<VS>──────────────────────────────────────────────────▶◀
```

Where:

<*VS*> Is the name, ID, or UUID of the virtual server.

Selected options

None.

Virtual server state transition

From State	To State (reason)
paused	running (unpaused)

Usage

"Resuming a virtual server" on page 116

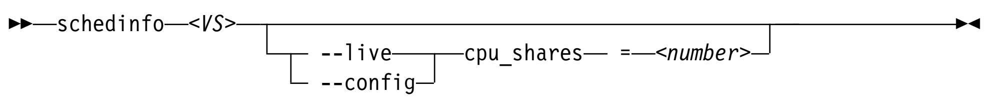
Example

```
# virsh list
 Id    Name                           State
----------------------------------------------------
 13    vserv1                        paused

# virsh resume vserv1
Domain vserv1 resumed

# virsh list
 Id    Name                           State
----------------------------------------------------
 13    vserv1                        running
```

schedinfo

Displays scheduling information about a virtual server, and can modify the portion of CPU time that is assigned to it.

Syntax



Where:

<number>
Specifies the CPU weight.

<VS> Is the name, the ID, or the UUID of the virtual server.

Selected options

--live Specifies the modification of the current CPU weight of the running virtual server.

--config
Specifies the modification of the virtual server's CPU weight after the next restart.

Usage

"Modifying the virtual CPU weight" on page 141

Examples

This example sets the CPU weight of the running virtual server vserv1 to 2048.

```
# virsh schedinfo vserv1 --live cpu_shares=2048
```

This example modifies the domain configuration-XML, which will be effective from the next restart.

```
# virsh schedinfo vserv1 --config cpu_shares=2048
```

This example displays scheduling information about the virtual server vserv1.

```
# virsh schedinfo vserv1
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
```

shutdown

Properly shuts down a running virtual server.

Syntax

```
                  ┌─ --domain─┐
►►──shutdown──────┴───────────┴──<VS>──────────────────────────────────►◄
```

Where:

<VS> Is the name, the ID, or the UUID of the virtual server.

Selected options

--domain
Specifies the virtual server.

Virtual server state transitions

From State	To State (reason)
running	shut off (shutdown)

Usage

- Chapter 1, “Overview,” on page 3
- “Terminating a virtual server” on page 114

Example

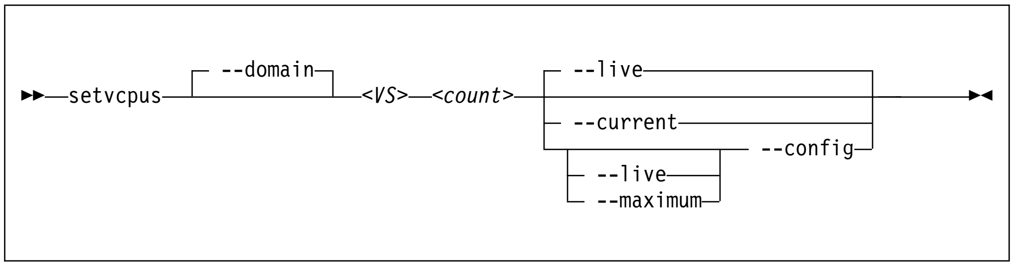
This example terminates virtual server vserv1.

```
# virsh shutdown vserv1
Domain vserv1 is being shutdown
```

setvcpus

Changes the number of virtual CPUs of a virtual server.

Syntax



Where:

<count>

If the `--maximum` option is not specified:
: Specifies the actual number of virtual CPUs which are made available for the virtual server.

  This value is limited by the maximum number of virtual CPUs. This number is configured with the vcpu element and can be modified during operation. If no number is specified, the maximum number of virtual CPUs is 1.

  If <*count*> is less than the actual number of available virtual CPUs, specify the `--config` option to remove the appropriate number of virtual CPUs with the next virtual server reboot. Until then, the virtual server user might set the corresponding number of virtual CPUs offline.

If the `--maximum` option is specified:
: Specifies the maximum number of virtual CPUs which can be made available after the next virtual server reboot.

  Do not specify more virtual CPUs than available host CPUs.

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

--config
: Changes the number the next time the virtual server is started.

--current, --live
: Changes the number of available virtual CPUs immediately.

--domain
: Specifies the virtual server.

--maximum
: Changes the maximum number of virtual CPUs that can be made available after the next virtual server reboot.

Usage

"Modifying the number of virtual CPUs" on page 138

Example

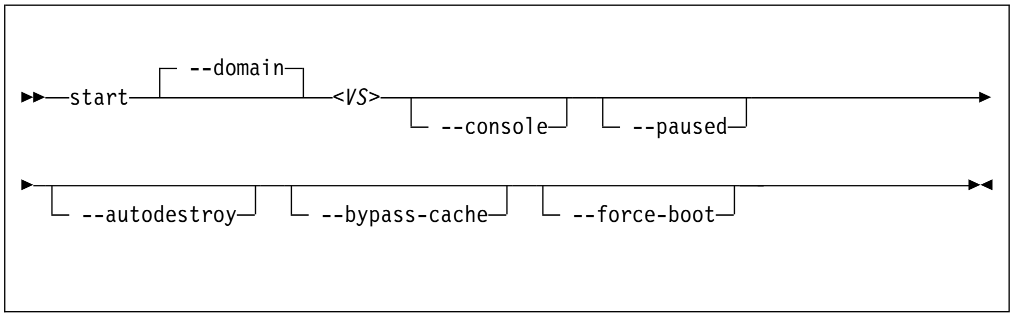
This example persistently adds a virtual CPU to the running virtual server vserv1:

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        3
current      live          3

# virsh setvcpus vserv1 4 --live --config

# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        4
current      live          4
```

start

Starts a defined virtual server that is shut off or crashed.

Syntax



Where:

<*VS*> Is the name, ID, or UUID of the virtual server.

Selected options

--autodestroy
Destroys the virtual server when virsh disconnects from libvirt.

--bypass-cache
Does not load the virtual server from the cache.

--console
Connects to a configured pty console.

--domain
Specifies the virtual server.

--force-boot
Any saved system image is discarded before booting.

--paused
Suspends the virtual server as soon as it is started.

Virtual server state transitions

Command option	From state (reason)	To state (reason)
`start`	shut off (unknown)	running (booted)
`start`	shut off (saved *from running*)	running (restored)
`start`	shut off (saved *from paused*)	paused (migrating)
`start`	shut off (shutdown)	running (booted)
`start`	shut off (destroyed)	running (booted)
`start`	crashed	running (booted)
`start --force-boot`	shut off (unknown)	running (booted)
`start --force-boot`	shut off (saved *from running*)	running (booted)
`start --force-boot`	shut off (saved *from paused*)	paused (user)
`start --force-boot`	shut off (shutdown)	running (booted)
`start --force-boot`	shut off (destroyed)	running (booted)

Command option	From state (reason)	To state (reason)
`start --paused`	shut off (unknown)	paused (user)
`start --paused`	shut off (saved *from running*)	paused (migrating)
`start --paused`	shut off (saved *from paused*)	paused (migrating)
`start --paused`	shut off (shutdown)	paused (user)
`start --paused`	shut off (destroyed)	paused (user)

Usage

- Chapter 1, "Overview," on page 3
- "Starting a virtual server" on page 114
- "Connecting to the console of a virtual server" on page 149

Example

This example starts virtual server vserv1 with initial console access.

```
# virsh start vserv1 --console
Domain vserv1 started
```

suspend

Transfers a virtual server from the running to the paused state.

Syntax

```
►►──suspend──<VS>──────────────────────────────────────────────────────►◄
```

Where:

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

None.

Virtual server state transition

From State	To State (reason)
running	paused (user)

Usage

"Suspending a virtual server" on page 116

Example

This example suspends virtual server vserv1.

```
# virsh list
 Id    Name                           State
----------------------------------------------------
 13    vserv1                        running

# virsh suspend vserv1
Domain vserv1 suspended

# virsh list
 Id    Name                           State
----------------------------------------------------
 13    vserv1                        paused
```

undefine

Deletes a virtual server from libvirt.

Purpose

Syntax

```
►►──undefine──<VS>──────────────────────────────────────────────────►◄
```

Where:

<*VS*> Is the name, ID, or UUID of the virtual server.

Selected options

None.

Usage

- Chapter 1, "Overview," on page 3
- "Undefining a virtual server" on page 111

Example

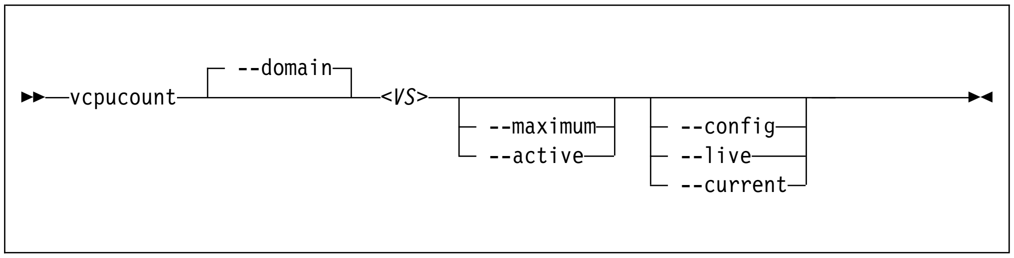
This example removes virtual server vserv1 from the libvirt definition.

```
# virsh undefine vserv1
```

vcpucount

Displays the number of virtual CPUs associated with a virtual server.

Syntax



where

<VS> Is the name, ID, or UUID of the virtual server.

Selected options

--active
Displays the number of virtual CPUs being used by the virtual server.

--config
Displays the number of virtual CPUs available to an inactive virtual server the next time it is restarted.

--current
Displays the number of virtual CPUs for the current virtual server.

--domain
Specifies the virtual server.

--live Displays the number of CPUs for the active virtual server.

--maximum
Displays information on the maximum cap of virtual CPUs that a virtual server can add.

Usage

"Modifying the number of virtual CPUs" on page 138

Example

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        3
current      live          3
```

vol-create

Creates a volume for a storage pool from a volume configuration-XML file.

Syntax

```
►►──vol-create──<pool-name>──<volume-XML-filename>──────────────────────►◄
```

Where:

<pool-name>
Is the name of the storage pool.

<volume-XML-filename>
Is the name of the volume configuration-XML file.

Selected options

None.

Usage

"Volume management commands" on page 153

Example

```
# virsh vol-create pool1 vol1.xml
```

vol-delete

Remove a volume from a storage pool.

Syntax

```
►►──vol-delete── --pool──┬─<pool-name>─┬──┬─<vol-key>───┬──────────────────►◄
                         └─<pool-UUID>─┘  ├─<vol-name>──┤
                                          └─<vol-path>──┘
```

Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

<vol-key>
Is the key of the volume.

<vol-name>
Is the name of the volume.

<vol-path>
Is the path of the volume.

Selected options

--pool Specifies the storage pool.

Usage

"Volume management commands" on page 153

Example

```
# virsh vol-delete --pool pool1 vol1
```

vol-dumpxml

Displays the current libvirt-internal configuration of a storage volume.

Syntax

```
▶▶──vol-dumpxml── --pool──┬─<pool-name>─┬──┬─<vol-key──┬──────────────────────▶◀
                          └─<pool-UUID>─┘  ├─<vol-name>─┤
                                           └─<vol-path>─┘
```

Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

<vol-key>
: Is the key of the volume.

<vol-name>
: Is the name of the volume.

<vol-path>
: Is the path of the volume.

Selected options

--pool Specifies the storage pool.

Usage

"Volume management commands" on page 153

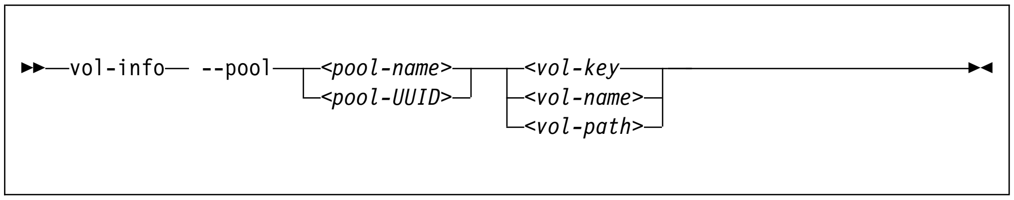
Example

```
# virsh vol-dumpxml --pool default federico.img
<volume type="file">
  <name>federico.img</name>
  <key>/var/lib/libvirt/images/federico.img</key>
  <source>
  </source>
  <capacity unit="bytes">12582912000</capacity>
  <allocation unit="bytes">2370707456</allocation>
  <target>
    <path>/var/lib/libvirt/images/federico.img</path>
    <format type="qcow2"/>
    <permissions>
      <mode>0600</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
    <timestamps>
      <atime>1481535271.342162944</atime>
      <mtime>1481292068.444109102</mtime>
      <ctime>1481292068.916109091</ctime>
    </timestamps>
  </target>
</volume>
```

vol-info

Displays information about a defined volume.

Syntax



Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

<vol-key>
Is the key of the volume.

<vol-name>
Is the name of the volume.

<vol-path>
Is the path of the volume.

Selected options

--pool Specifies the storage pool.

Usage

"Volume management commands" on page 153

Example

```
# virsh vol-info --pool pool1 vol1
```

vol-key

Displays the key of a volume from its name or path.

Syntax

```
►►──vol-key── --pool──┬─<pool-name>─┬──┬─<vol-name>─┬──────────────────────────►◄
                      └─<pool-UUID>─┘  └─<vol-path>─┘
```

Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

<vol-name>
Is the name of the volume.

<vol-path>
Is the path of the volume.

Selected options

--pool Specifies the storage pool.

Usage

"Volume management commands" on page 153

Example

This example displays the volume key of vol1 as a volume of storage pool pool1.

```
# virsh vol-key --pool pool1 vol1
/var/lib/libvirt/images/federico.img
```

vol-list

Displays a list of defined storage pools.

By default, a list of active storage pools is displayed.

Syntax

```
►►──vol-list── --pool──┬─<pool-name>─┬──┬───────────┬──────────────────────►◄
                       └─<pool-UUID>─┘  └─ --details─┘
```

Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

Selected options

--details
: Displays volume type and capacity related information.

--pool Specifies the storage pool.

Usage

"Volume management commands" on page 153

Example

```
# virsh vol-list --pool pool1
```

vol-name

Displays the name of a volume from its key or path.

Syntax

```
►►──vol-name── --pool──┬─<pool-name>─┬──┬─<vol-key>──┬──────────────────────────►◄
                       └─<pool-UUID>─┘  └─<vol-path>─┘
```

Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

<vol-key>
: Is the key of the volume.

<vol-path>
: Is the path of the volume.

Selected options

--pool Specifies the storage pool.

Usage

"Volume management commands" on page 153

Example

This example displays the volume name of vol1 as a volume of storage pool pool1.

```
# virsh vol-name --pool pool1 /var/lib/libvirt/images/federico.img
vol1
```

vol-path

Displays the path of a volume from its name or key.

Syntax

```
►►──vol-path── --pool──┬─<pool-name>─┬──┬─<vol-key>──┬──────────────────────►◄
                       └─<pool-UUID>─┘  └─<vol-name>─┘
```

Where:

<pool-name>
: Is the name of the storage pool.

<pool-UUID>
: Is the UUID of the storage pool.

<vol-key>
: Is the key of the volume.

<vol-name>
: Is the name of the volume.

Selected options

None.

Usage

"Volume management commands" on page 153

Example

This example displays the volume key of vol1 as a volume of storage pool pool1.

```
# virsh vol-path --pool pool1 vol1
/var/lib/libvirt/images/federico.img
```

vol-pool

Displays the name or the UUID of the storage pool containing a given volume.

By default, the strorage pool name is displayed.

Syntax

```
►►──vol-pool──┬────────┬──┬─<vol-key>──┬──────────────────────────────►◄
              └─--uuid─┘  ├─<vol-name>─┤
                          └─<vol-path>─┘
```

Where:

<vol-key>
: Is the key of the volume.

<vol-name>
: Is the name of the volume.

<vol-path>
: Is the path of the volume.

Selected options

--pool Specifies the storage pool.

--uuid Returns the storage pool UUID.

Usage

"Volume management commands" on page 153

Example

```
# virsh vol-pool vol1
pool1
```

Chapter 30. Selected QEMU commands

QEMU monitor commands

Do not use the QEMU monitor commands, because their use can change the state of a virtual server, might disturb the correct operation of libvirt and lead to inconsistent states or even a crash of the virtual server.

Examples for the use of the qemu-img command

- This example creates a qcow2 image with a maximum size of 10GB:

```
# qemu-img create -f qcow2 /var/lib/libvirt/images/disk1.img 10G
Formatting '/var/lib/libvirt/images/disk1.img', fmt=qcow2
size=10737418240 encryption=off cluster_size=65536
lazy_refcounts=off
Format specific information:
compat: 1.1
lazy refcounts: false
refcount bits: 16
corrupt: false
```

- This example displays attributes of a qcow2 image:

```
# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: qcow2
virtual size: 10G (10737418240 bytes)
disk size: 136K
cluster_size: 65536
```

- This example increases the size of a qcow2 image:

```
# qemu-img resize /var/lib/libvirt/images/disk1.img 20G
Image resized.

# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 140K
cluster_size: 65536
```

- This example creates a RAW image with a maximum size of 10GB:

```
# qemu-img create -f raw /var/lib/libvirt/images/disk1.img 10G
Formatting '/var/lib/libvirt/images/disk1.img', fmt=raw
size=10737418240
```

- This example displays attributes of a RAW image:

```
# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: raw
virtual size: 10G (10737418240 bytes)
disk size: 0
```

- This example increases the size of a RAW image:

```
# qemu-img resize -f raw /var/lib/libvirt/images/disk1.img 20G
Image resized.

# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: raw
virtual size: 20G (21474836480 bytes)
disk size: 0
```

Chapter 31. Hypervisor information for the virtual server user

The virtual server user can use the emulated Store Hypervisor Information (STHYI) instruction to retrieve information about the IBM Z hardware and the LPAR on which the KVM host runs.

The information includes:
- The CPU count, by type (CP or IFL)
- Limitations for shared CPUs
- CEC and LPAR identifiers

KVM guests use the `qclib` and the GCC inline assembly to run the emulated instruction. For an example, see `arch/s390/kvm/sthyi.c` in the Linux source tree.

The emulated STHYI instruction provides information through a response buffer with three data sections:
- The header section, at the beginning of the response buffer, which identifies the locations and length of the sections that follow.
- The machine section.
- The partition section.

Header section

Length	Data Type	Offset (dec)	Name	Contents
1	Bitstring	0	INFHFLG1	Header Flag Byte 1 These flag settings indicate the environment that the instruction was executed in and may influence the value of the validity bits. The validity bits, and not these flags, should be used to determine if a field is valid. **0x80** Global Performance Data unavailable. **0x40** One or more hypervisor levels below this level does not support the STHYI instruction. When this flag is set the value of INFGPDU is not meaningful because the state of the Global Performance Data setting cannot be determined. **0x20** Virtualization stack is incomplete. This bit indicates one of two cases: • One or more hypervisor levels does not support the STHYI instruction. For this case, INFSTHYI will also be set. • There were more than three levels of guest/hypervisor information to report. **0x10** Execution environment is not within a logical partition.

Length	Data Type	Offset (dec)	Name	Contents
1	Bitstring	1	INFHFLG2	Header Flag Byte 2 reserved for IBM use
1	Bitstring	2	INFHVAL1	Header Validity Byte 1 reserved for IBM use
1	Bitstring	3	INFHVAL2	Header Validity Byte 2 reserved for IBM use
3		4		Reserved for future IBM use
1	Unsigned Binary Integer	7	INFHYGCT	Count of Hypervisor and Guest Sections
2	Unsigned Binary Integer	8	INFHTOTL	Total length of response buffer
2	Unsigned Binary Integer	10	INFHDLN	Length of Header Section mapped by INF0HDR
2	Unsigned Binary Integer	12	INFMOFF	Offset to Machine Section mapped by INF0MAC
2	Unsigned Binary Integer	14	INFMLEN	Length of Machine Section
2	Unsigned Binary Integer	16	INFPOFF	Offset to Partition Section mapped by INF0PAR
2	Unsigned Binary Integer	18	INFPLEN	Length of Partition Section
2	Unsigned Binary Integer	20	INFHOFF1	Offset to Hypervisor Section1 mapped by INF0HYP
2	Unsigned Binary Integer	22	INFHLEN1	Length of Hypervisor Section1
2	Unsigned Binary Integer	24	INFGOFF1	Offset to Guest Section1 mapped by INF0GST
2	Unsigned Binary Integer	26	INFGLEN1	Length of Guest Section1
2	Unsigned Binary Integer	28	INFHOFF2	Offset to Hypervisor Section2 mapped by INF0HYP
2	Unsigned Binary Integer	30	INFHLEN2	Length of Hypervisor Section2
2	Unsigned Binary Integer	32	INFGOFF2	Offset to Guest Section2 mapped by INF0GST
2	Unsigned Binary Integer	34	INFGLEN2	Length of Guest Section2
2	Unsigned Binary Integer	36	INFHOFF3	Offset to Hypervisor Section3 mapped by INF0HYP
2	Unsigned Binary Integer	38	INFHLEN3	Length of Hypervisor Section3
2	Unsigned Binary Integer	40	INFGOFF3	Offset to Guest Section3 mapped by INF0GST
2	Unsigned Binary Integer	42	INFGLEN3	Length of Guest Section3
4		44		Reserved for future IBM use

Format machine section

Length	Data Type	Offset (dec)	Name	Contents
1	Bitstring	0	INFMFLG1	Machine Flag Byte 1 reserved for IBM use
1	Bitstring	1	INFMFLG2	Machine Flag Byte 2 reserved for IBM use
1	Bitstring	2	INFMVAL1	Machine Validity Byte 1 **0x80** Processor Count Validity. When this bit is on, it indicates that INFMSCPS, INFMDCPS, INFMSIFL, and INFMDIFL contain valid counts. The validity bit may be off when: • STHYI support is not available on a lower level hypervisor, or • Global Performance Data is not enabled. **0x40** Machine ID Validity. This bit being on indicates that a SYSIB 1.1.1 was obtained from STSI and information reported in the following fields is valid: INFMTYPE, INFMMANU, INFMSEQ, and INFMPMAN. **0x20** Machine Name Validity. This bit being on indicates that the INFMNAME field is valid.
1	Bitstring	3	INFMVAL2	Machine Validity Byte 2 reserved for IBM use
2	Unsigned Binary Integer	4	INFMSCPS	Number of shared CPs configured in the machine or in the physical partition if the system is physically partitioned
2	Unsigned Binary Integer	6	INFMDCPS	Number of dedicated CPs configured in this machine or in the physical partition if the system is physically partitioned
2	Unsigned Binary Integer	8	INFMSIFL	Number of shared IFLs configured in this machine or in the physical partition if the system is physically partitioned.
2	Unsigned Binary Integer	10	INFMDIFL	Number of dedicated IFLs configured in this machine or in the physical partition if the system is physically partitioned.
8	EBCDIC	12	INFMNAME	Machine Name
4	EBCDIC	20	INFMTYPE	Type
16	EBCDIC	24	INFMMANU	Manufacturer
16	EBCDIC	40	INFMSEQ	Sequence Code
4	EBCDIC	56	INFMPMAN	Plant of Manufacture
4		60		Reserved for future IBM use

Format partition section

Length	Data Type	Offset (dec)	Name	Contents
1	Bitstring	0	INFPFLG1	Partition Flag Byte 1 **0x80** Multithreading (MT) is enabled.
1	Bitstring	1	INFPFLG2	Partition Flag Byte 2 reserved for IBM use
1	Bitstring	2	INFPVAL1	Partition Validity Byte 1 **0x80** This bit being on indicates that INFPSCPS, INFPDCPS, INFPSIFL, and INFPDIFL contain valid counts. **0x40** This bit being on indicates that INFPWBCP and INFPWBIF are valid **0x20** This bit being on indicates that INFPABCP and INFPABIF are valid. **0x10** This bit being on indicates that a SYSIB 2.2.2 was obtained from STSI and information reported in the following fields is valid: INFPPNUM and INFPPNAM. **0x08** This bit being on indicates that INFPLGNM, INFPLGCP, and INFPLGIF are valid.
1	Bitstring	3	INFPVAL2	Partition Validity Byte 2 reserved for IBM use
2	Unsigned Binary Integer	4	INFPPNUM	Logical partition number
2	Unsigned Binary Integer	6	INFPSCPS	Number of shared logical CPs configured for this partition. Count of cores when MT is enabled.
2	Unsigned Binary Integer	8	INFPDCPS	Number of dedicated logical CPs configured for this partition. Count of cores when MT is enabled.
2	Unsigned Binary Integer	10	INFPSIFL	Number of shared logical IFLs configured for this partition. Count of cores when MT is enabled.
2	Unsigned Binary Integer	12	INFPDIFL	Number of dedicated logical IFLs configured for this partition. Count of cores when MT is enabled.
2		14		Reserved for future IBM use
8	EBCIDIC	16	INFPPNAM	Logical partition name
4	Unsigned Binary Integer	24	INFPWBCP	Partition weight-based capped capacity for CPs, a scaled number where X'00010000' represents one core. Zero if not capped.
4	Unsigned Binary Integer	28	INFPABCP	Partition absolute capped capacity for CPs, a scaled number where X'00010000' represents one core. Zero if not capped.

Length	Data Type	Offset (dec)	Name	Contents
4	Unsigned Binary Integer	32	INFPWBIF	Partition weight-based capped capacity for IFLs, a scaled number where X'00010000' represents one core. Zero if not capped.
4	Unsigned Binary Integer	36	INFPABIF	Partition absolute capped capacity for IFLs, a scaled number where X'00010000' represents one core. Zero if not capped.
8	EBCIDIC	40	INFPLGNM	LPAR group name. Binary zeros when the partition is not in an LPAR group. EBCDIC and padded with blanks on the right when in a group. The group name is reported only when there is a group cap on CP or IFL CPU types and the partition has the capped CPU type.
4	Unsigned Binary Integer	48	INFPLGCP	LPAR group absolute capacity value for CP CPU type when nonzero. This field will be nonzero only when INFPLGNM is nonzero and a cap is defined for the LPAR group for the CP CPU type. When nonzero, contains a scaled number where X'00010000' represents one core.
4	Unsigned Binary Integer	52	INFPLGIF	LPAR group absolute capacity value for IFL CPU type when nonzero. This field will be nonzero only when INFPLGNM is nonzero and a cap is defined for the LPAR group for the IFL CPU type. When nonzero, contains a scaled number where X'00010000' represents one core.

Part 8. Appendixes

Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on z Systems and LinuxONE publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication, use the Readers' Comments form in the back of this publication, send an email to eservdoc@de.ibm.com, or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

`www.ibm.com/able`

Notices

Trademarks

Index

Special characters

--active option
  of the vcpucount virsh command 340
--all option
  of the list virsh command 111, 120, 295
  of the net-list virsh command 312
--auto-converge option
  of the migrate virsh command 133, 134
--autodestroy option
  of the start virsh command 336
--autostart option
  of the list virsh command 295
  of the net-list virsh command 312
--bandwidth option
  of the migrate-setspeed virsh command 134, 305
--build option
  of the pool-start virsh command 327
--bypass-cache option
  of the managedsave virsh command 297
  of the start virsh command 336
--change-protection option
  of the migrate virsh command 300
--compressed option
  of the migrate virsh command 300
--config option
  of the attach-device virsh command 146, 268
  of the change-media virsh command 270
  of the detach-device virsh command 275
  of the iothreadadd virsh command 290
  of the iothreaddel virsh command 292
  of the iothreadinfo virsh command 294
  of the schedinfo virsh command 141, 332
  of the setvcpus virsh command 334
  of the vcpucount virsh command 340
--console option
  of the start virsh command 114, 149, 336
--copy-storage-all option
  of the migrate virsh command 134, 300
--copy-storage-inc option
  of the migrate virsh command 134, 300
--current option
  of the change-media virsh command 270
  of the detach-device virsh command 275
  of the iothreadadd virsh command 290
  of the iothreaddel virsh command 292
  of the iothreadinfo virsh command 294
  of the setvcpus virsh command 334
  of the vcpucount virsh command 340
--direct option
  of the migrate virsh command 300
--disable option
  of the net-autostart virsh command 306
  of the pool-autostart virsh command 317
--domain option
  of the attach-device virsh command 268
  of the change-media virsh command 270
  of the destroy virsh command 274
  of the detach-device virsh command 275
  of the dumpxml virsh command 287
  of the shutdown virsh command 333
  of the start virsh command 336
--eject option
  of the change-media virsh command 148, 270
--file option
  of the attach-device virsh command 268
  of the detach-device virsh command 275
--force option
  of the change-media virsh command 270
  of the console virsh command 272
--force-boot option
  of the start virsh command 114, 336
--graceful option
  of the destroy virsh command 114, 274
--id option
  of the iothreadadd virsh command 290
  of the iothreaddel virsh command 292
  of the list virsh command 295
--inactive option
  of the dumpxml virsh command 287
  of the list virsh command 295
  of the net-dumpxml virsh command 309
  of the net-list virsh command 312
--insert option
  of the change-media virsh command 148, 270
--keepalive-count option
  of the virsh command 134
--keepalive-interval option
  of the virsh command 134
--live option
  of the change-media virsh command 270
  of the detach-device virsh command 275
  of the iothreadadd virsh command 290
  of the iothreaddel virsh command 292
  of the iothreadinfo virsh command 294
  of the migrate virsh command 300
  of the schedinfo virsh command 141, 332
  of the setvcpus virsh command 334
  of the vcpucount virsh command 340
--machine option
  of the domcapabilities virsh command 280
--managed-save option
  of the list virsh command 295
--maximum option
  of the setvcpus virsh command 334
  of the vcpucount virsh command 340
--memory-only option
  of the dump virsh command 175, 286
--migratable option
  of the dumpxml virsh command 287
--migrate-disks option
  of the migrate virsh command 134, 300
--mode option
  of the shutdown virsh command 333
--name option
  of the list virsh command 295
  of the net-list virsh command 312
--no-autostart option
  of the list virsh command 295
  of the net-list virsh command 312
--no-overwrite option
  of the pool-start virsh command 327

--offline option
  of the migrate virsh command 300
--overwrite option
  of the pool-start virsh command 327
--p2p option
  of the migrate virsh command 300
--path option
  of the change-media virsh command 270
--paused option
  of the managedsave virsh command 297
  of the start virsh command 336
--persisent option
  of the migrate virsh command 300
--persistent option
  of the detach-device virsh command 275
  of the list virsh command 295
  of the migrate virsh command 300
--reason option
  of the domstate virsh command 120, 285
--running option
  of the managedsave virsh command 297
--safe option
  of the console virsh command 272
--security-info option
  of the dumpxml virsh command 287
--soft-limit option
  of the memtune virsh command 299
--state-other option
  of the list virsh command 295
--state-paused option
  of the list virsh command 295
--state-running option
  of the list virsh command 295
--state-shutoff option
  of the list virsh command 295
--suspend option
  of the migrate virsh command 300
--table option
  of the list virsh command 295
  of the migrate virsh command 300
  of the net-list virsh command 312
--timeout option
  of the migrate virsh command 133, 134, 300
--title option
  of the list virsh command 295
--transient option
  of the list virsh command 295
  of the migrate virsh command 300
--tunnelled option
  of the migrate virsh command 300
--undefinesource option
  of the migrate virsh command 300
--unsafe option
  of the migrate virsh command 300
--update option
  of the change-media virsh command 270
--update-cpu option
  of the dumpxml virsh command 287
--uuid option
  of the list virsh command 295
  of the net-list virsh command 312
--verbose option
  of the managedsave virsh command 297
  of the migrate virsh command 300
--with-managed-save option
  of the list virsh command 295
--with-snapshot option
  of the list virsh command 295
--without-managed-save option
  of the list virsh command 295
--without-snapshot option
  of the list virsh command 295
--xml option
  of the migrate virsh command 300
/etc/hosts 110
/etc/libvirt/libvirtd.conf 134, 173
/etc/libvirt/qemu.conf 132
/etc/lvm/lvm.conf 167
/etc/ssh/sshd_config 132
/etc/zfcp.conf 29
/var/log/libvirtd/qemu/<VS>.log 173
/var/log/messages 173
$EDITOR environment variable 110
$VISUAL environment variable 110

Numerics

0xfe value
  of the address cssid attribute 88
3DEA 72
3DES 72

A

accessibility 361
action attribute
  of the watchdog element 263
activating
  bonded interfaces 40
  network interfaces 38
active bridge port 43
adapter element 90, 191
  name attribute 90, 191
adding
  CPUs 138
  I/O threads 146
address attribute
  of the ip element 222
  of the mac element 98, 100, 227
  of the target element 254
address element 76, 78, 84, 86, 88, 90, 95, 192, 193, 194
  bus attribute 90, 95, 193, 194
  controller attribute 90, 95, 193
  cssid attribute 78, 84, 86, 88, 192
    0xfe value 88
  devno attribute 78, 84, 86, 88, 192
  ssid attribute 78, 84, 86, 88, 192
  target attribute 90, 95, 193, 194
  type attribute 78, 84, 86, 88, 90, 192, 193
    ccw value 78, 84, 86, 88, 192
    drive value 95
    scsi value 90
  unit attribute 90, 95, 193, 194
Advanced Encryption Standard 72
AES 72
aes value
  of the cipher name attribute 72, 198
alias device 27
alias machine type 126
append attribute
  of the log element 70, 226

arch attribute
  of the target element 258
  of the type element 51
attach-device command
  --config option 146
attach-device virsh command 146, 268
attaching
  devices 146
auto value
  of the controller model attribute 201
  of the geometry trans attribute 216

B

backend element 195
  model attribute 195
balloon device 74
base device 27
block device 9, 14, 75
  attaching 146
  detaching 147
  displaying 120
  hotplugging 146
  unplugging 147
block device driver 4
block value
  of the disk type attribute 78, 207
bonded interface 21, 40
  activating 40
  configuring 98
  preparing 40
  setting up 40
bonded network interface 40
bonding
  mode 40
  parameter 40
boot element 53, 196
  loadparm attribute 196
  order attribute 53, 196
boot menu selection 196
boot process
  configuring 53
bootable kernel 53, 55
booted reason
  of the running state 185
booting
  from a DASD 53
  from a kernel image file 55
  from a SCSI disk 53
  from an ISO image 54
  over the network 56
bridge attribute
  of the source element 100
bridge element 197
  name attribute 197
bridge port 43
  active 43
  primary 43
bridge port role
  enabling 43
bridge value
  of the forward mode attribute 215
  of the interface type attribute 100, 220
  of the source mode attribute 98
bridge_state sysfs attribute 43
browsing
  virtual servers 120
bus attribute
  of the address element 90, 95, 193, 194
  of the target element 78, 84, 86, 95, 255
by-ID device node 9, 14
by-path device node 9, 14, 27, 78
by-uuid device node 9, 14

C

cache attribute
  of the driver element 78, 84, 86, 95, 210
canceling
  a live migration 134
CCW device 75
CCW group device 38, 43
ccw value
  of the address type attribute 78, 84, 86, 88, 192
CD-ROM 75
cdrom value
  of the disk device attribute 95, 207
Central Processor Assist for Cryptographic Functions 72
cgroups 160
change-media command
  --eject option 148
  --insert option 148
change-media virsh command 148, 270
channel bonding 40
channel bonding module 40
channel path 9, 14, 75
channel path type 75
channel subsystem 9, 14, 75
channel subsystem-ID 9, 14, 75, 78, 84, 86
chccwdev command 27, 29, 33
chcpu command 138
cipher element 72, 198
  name attribute 72, 198
    aes value 72, 198
    dea value 72, 198
  state attribute 72, 198
    off value 72, 198
    on value 72, 198
clear key 72
cmdline element 51, 55, 199
collecting
  performance metrics 177
command
  chccwdev 29, 33
  chcpu 138
  crash 175
  fdisk 29
  ip 37, 38
  ls 78
  lscss 33, 78
  lsqeth 38, 43
  lsscsi 29
  lszfcp 29, 33
  modprobe 40
  multipath 29, 78
  ovs-vsctl add-bond 43
  ovs-vsctl add-br 43
  ovs-vsctl add-port 43
  ovs-vsctl del-br 43
  ovs-vsctl show 43
  perf kvm stat 177
  pvscan 167
  QEMU monitor info 351
  qemu-kvm 126

command *(continued)*
  start openvswitch 43
  status openvswitch 43
  virsh attach-device 146, 268
  virsh change-media 148, 270
  virsh console 149, 272
  virsh define 110, 273
  virsh destroy 114, 274
  virsh detach-device 147, 275
  virsh domblklist 120, 277
  virsh domblkstat 120, 278
  virsh domcapabilities 280
  virsh domiflist 120, 281
  virsh domifstat 120, 282
  virsh dominfo 120, 283
  virsh domjobabort 134, 284
  virsh domstate 120, 285
  virsh dump 175, 286
  virsh dumpxml 88, 122, 287
  virsh edit 110, 288
  virsh inject-nmi 289
  virsh iothreadadd 290
  virsh iothreaddel 292
  virsh iothreadinfo 294
  virsh list 111, 120, 145, 149, 295
  virsh managedsave 114, 297
  virsh memtune 299
  virsh migrate 134, 300
  virsh migrate-getspeed 134, 303
  virsh migrate-setmaxdowntime 134
  virsh migrate-setspeed 134, 305
  virsh net-autostart 306
  virsh net-define 307
  virsh net-destroy 308
  virsh net-dumpxml 309
  virsh net-edit 310
  virsh net-info 311
  virsh net-list 312
  virsh net-name 313
  virsh net-start 314
  virsh net-undefine 315
  virsh net-uuid 316
  virsh pool-autostart 317
  virsh pool-define 318
  virsh pool-delete 319
  virsh pool-destroy 320
  virsh pool-dumpxml 321
  virsh pool-edit 322
  virsh pool-info 323
  virsh pool-list 152, 324
  virsh pool-name 325
  virsh pool-refresh 326
  virsh pool-start 327
  virsh pool-undefine 328
  virsh pool-uuid 329
  virsh reboot 330
  virsh resume 116, 331
  virsh schedinfo 120, 141, 332
  virsh setvcpus 138, 334
  virsh shutdown 114, 333
  virsh start 114, 149, 336
  virsh suspend 116, 338
  virsh undefine 111, 339
  virsh vcpucount 120, 138, 340
  virsh vol-create 341
  virsh vol-delete 342
  virsh vol-dumpxml 343
command *(continued)*
  virsh vol-info 344
  virsh vol-key 345
  virsh vol-list 153, 346
  virsh vol-name 347
  virsh vol-path 348
  virsh vol-pool 349
  zipl 53, 55
  znetconf 37, 38, 43, 98
concurrency 132
concurrent connections 132
CONFIG_HAVE_PERF_EVENTS 177
CONFIG_PERF_EVENTS 177
CONFIG_TRACEPOINTS 177
configuration
  libvirt-internal 122
  of devices 4
  of virtual servers 4
configuration file
  libvirt 173
  of the OpenSSH SSH daemon configuration file 132
configuration topology 9, 14
configuration-XML file 9, 14, 75
  of a device 4, 9, 14, 76
  of a domain 4, 9, 14, 51
  of a storage pool 103
  of a volume 103
configuring
  an ISO image as IPL device 54
  bonded interfaces 98
  boot devices 49, 53
  boot process 53, 54, 55
  boot process, network 56
  consoles 70
  CPU model 63
  CPUs 49, 61
  DASDs 78
  devices 4, 75
  devices with virtual server 69
  Ethernet interfaces 98
  FC-attached SCSI tape devices 90
  I/O threads 69
  image files as storage devices 84
  logical volumes 167
  memory 49, 65
  multipath device mapper support 29
  network devices 49
  network interfaces 98
  operating systems 49
  physical volumes 167
  protected key encryption 72
  random number generators 102
  removable ISO images 95
  SCSI disks 78
  SCSI medium changer devices 88, 90
  SCSI tapes 88, 90
  storage devices 49
  storage pools 103
  user space 49, 68
  virtual CPUs 61
  virtual Host Bus Adapters 88
  virtual memory 65
  virtual networks 105
  virtual servers 4
  virtual switches 98, 100
  volumes as storage devices 86
  watchdog devices 71

connecting
  to the console of a virtual server 149
connections 132
console 70, 149
  configuring 70
  connecting 149
console element 70, 200
  type attribute 70, 200
    pty value 70, 200
console log file 70, 173
console output
  logging 70
console virsh command 149, 272
control unit model 75
controller attribute
  of the address element 90, 95, 193
controller element 76, 88, 201
  index attribute 88, 201
  model attribute 88, 201
    virtio-scsi value 88
  ports attribute 201
  type attribute 88, 201
    scsi value 88
  vectors attribute 201
core dump
  configurable 67
CPACF 72
CPU
  adding 138
  configuring 61
  configuring model 63
  management 159
  modifying the number 138
  pinning 159, 160
cpu element 63, 202
  match attribute 63, 202
  mode attribute 63, 202
CPU migration 159
CPU shares 141, 160
CPU time 141, 160
CPU weight 62, 141, 160
  modifying 141
cpu_shares parameter
  of the schedinfo virsh command 141, 332
cputune element 51, 62, 203
crash command 175
crashed state 183, 187
creating
  file systems 29
  network interfaces 38
  partitions 29
  persistent virtual server definitions 109
  uplink ports 43
  virtual server dumps 175
  virtual servers 4
  virtual switches 43
cryptographic key 72
cssid attribute
  of the address element 78, 84, 86, 88, 192
current attribute
  of the vcpu element 259
current live 138
custom value
  of the cpu mode attribute 63, 202
cyls attribute
  of the geometry element 216

D

DASD 78
  configuring 78
  preparing 27
  setting up 27
  virtualization 9, 14
DASD configuration
  example 81
dasdfmt command 27
Data Encryption Algorithm 72
Data Encryption Standard 72
DEA 72
dea value
  of the cipher name attribute 72, 198
deadlock prevention 132
default value
  of the driver cache attribute 210
define virsh command 110, 273
defined virtual server 4
defining
  virtual servers 4, 110
definition 4, 126, 127
  creating 110
  deleting 111
  modifying 110
deleting
  virtual server definitions 109
DES 72
destination port 132
destroy text content
  of the on_reboot element 238
destroy virsh command 114, 274
  --graceful option 114, 274
destroyed reason
  of the shut off state 184
destroying
  virtual servers 114
detach-device virsh command 147, 275
detaching
  devices 147
dev attribute
  of the source element 78, 84, 86, 98, 249, 252
  of the target element 78, 84, 86, 95, 255
device
  attaching 146
  configuring 75
  detaching 147
  hotplugging 146
  managing 145
  unplugging 147
device attribute
  of the disk element 78, 84, 86, 95, 207
device bus-ID 9, 14, 37
  of a virtual block device 9, 14, 78, 84, 86
  of an FCP device 9, 14, 29
device configuration topology 9, 14
device configuration-XML 4
device configuration-XML file 4, 75, 76
  child elements 76
  root element 76
device driver
  lin_tape 17, 33
  MacVTap 21
  SCSI generic 17
  watchdog 177
device element 204
  path attribute 204

device mapper-created device node 9, 14, 29, 78
device name 9, 14
  logical 78, 84, 86
  standard 9, 14
device node 9, 14, 78
  device mapper 9, 14
  device mapper-created 29, 78
  standard 9, 14, 78
  udev-created 9, 14, 78
device number 9, 14
  of a virtual block device 9, 14, 78, 84, 86
  of an FCP device 9, 14
device type 9, 14, 75
device-mapper multipathing 29
devices element 51, 205
devno attribute
  of the address element 78, 84, 86, 88, 192
dhcp element 206
DIAG event 177
diagnose 177
direct connection 98
direct MacVTap connection 40
direct value
  of the interface type attribute 98, 220
directsync value
  of the driver cache attribute 210
dirty pages 133
disable value
  of the feature policy attribute 63, 213
disabling
  protected key encryption 72
disk 75
disk element 76, 78, 84, 86, 95, 207
  device attribute 78, 84, 86, 95, 207
    cdrom value 95, 207
    disk value 78, 84, 86, 207
  type attribute 78, 84, 86, 95, 207
    block value 78, 207
    file value 84, 86, 95, 207
disk migration 129, 134
disk value
  of the disk device attribute 78, 84, 86, 207
displaying
  block devices 120
  information about a virtual server 120
  network interfaces 120
  performance metrics 177
  scheduling information 120
  states 120
  the libvirt-internal configuration 122
domain 3, 4
domain configuration-XML 4, 9, 14
domain configuration-XML file 4, 49, 51, 75
  child elements 51
  root element 51
domain element 208
  type attribute 51, 208
domblklist virsh command 120, 277
domblkstat virsh command 120, 278
domcapabilities virsh command 280
domiflist virsh command 120, 281
domifstat virsh command 120, 282
dominfo virsh command 120, 283
domjobabort virsh command 134, 284
domstate virsh command 120, 285
  --reason option 120, 285
drive value
  of the address type attribute 95
driver element 76, 78, 84, 86, 95, 209, 210
  cache attribute 78, 84, 86, 95, 210
    default value 210
    directsync value 210
    none value 84, 86, 95, 210
    unsafe value 210
    writeback value 210
    writethrough value 84, 86, 210
  error_policy attribute 210
    enospace value 210
    ignore value 210
    report value 210
    stop value 210
  event_idx attribute 210
    off value 210
    on value 210
  io attribute 78, 84, 86, 95, 210
    native value 84, 86, 95, 210
    of the driver element 84, 86
    threads value 210
  ioeventfd attribute 210
    off value 210
    on value 210
  iothread attribute 78, 209
  name attribute 78, 84, 86, 95, 210
    qemu value 95, 210
  rerror_policy attribute 210
    ignore value 210
    report value 210
    stop value 210
  type attribute 78, 84, 86, 95, 210
    qcow2 value 84, 86
    raw value 84, 86, 95, 210
driver value
  of the error_policy attribute 210
  of the rerror_policy attribute 210
dump 175
  configurable 67
dump file 175
dump location 175
dump virsh command 175, 286
  --memory-only option 175, 286
dumpCore attribute
  of the memory element 67
dumping
  virtual servers 175
dumpxml virsh command 88, 122, 287
DVD 75

E

edit virsh command 110, 288
editing
  libvirt-internal configurations 110
  persistent virtual server definitions 110
emulator element 68, 212
enabling
  bridge port roles 43
encryption 72
enospace value
  of the driver error_policy attribute 210
environment variable
  $EDITOR 110
  $VISUAL 110

error_policy attribute
  of the driver element 210
Ethernet interface
  configuring 98
  preparing 37
event_idx attribute
  of the driver element 210
exact value
  of the cpu match attribute 63, 202
example
  of a DASD configuration 81
  of a multipathed SCSI device configuration 93
  of a SCSI disk configuration 82
  of an initial installation 58

F

failover redundancy 29
FC-attached SCSI medium changer device 17
FC-attached SCSI tape device 17
  configuring 90
  preparing 33
FCP device 9, 14, 29
FCP LUN 9, 14, 29
fdasd command 27
fdisk command 27, 29
feature element 63, 213
file 75
file attribute
  of the log element 70, 226
  of the source attribute 95
  of the source element 84, 86, 249
file value
  of the disk type attribute 84, 86, 95, 207
filtered value
  of the hostdev sgio attribute 218
firewall configuration 132
format element 214
  type attribute 214
forward element 215
  mode attribute 215
full isolation 21

G

geometry element 216
  cyls attribute 216
  heads attribute 216
  secs attribute 216
  trans attribute 216
GRE tunnel 43
guest x, 3, 4
  relocation
    See live migration

H

hardware information 353
HBA 75
  configuring 88
heads attribute
  of the geometry element 216
high reliability 9, 14, 17, 21
host x, 4, 40
Host Bus Adapter 75
  configuring 88
host device 90
host element 217
  name attribute 217
host network
  OSA-Express device 40
host-model value
  of the cpu mode attribute 63, 202
hostdev element 76, 90, 218
  mode attribute 90, 218
    subsystem value 90
  rawio attribute 218
  sgio attribute 218
  type attribute 90, 218
    scsi value 90
hotplug device 4, 76, 146
hotplugging
  devices 75, 146
hvm text content
  of the type element 258
hypervisor
  release 126, 127
hypervisor information for the virtual server user 353
hypervisor release 132

I

I/O operations
  improving performance 69, 167
I/O thread 78, 167
  adding 146
  configuring 69
  providing 69
  removing 147
ignore value
  of the driver error_policy attribute 210
  of the driver rerror_policy attribute 210
image file 75
  qcow2 84, 86
  raw 84, 86
improving
  the performance of I/O operations 69, 167
index attribute
  of the controller element 88, 201
info QEMU command 351
initial installation
  example 58
Initial Program Load 53, 114
initial ramdisk 53, 55, 58
initrd element 51, 55, 219
inject-nmi virsh command 175, 289
installation DVD 54
installation file 55
installing a guest
  from a kernel image file 55
  from an ISO image 54
  over the network 56
interface
  bonded 40
  MacVTap 40
  virtual LAN 40
  VLAN 40
interface element 76, 98, 100, 220
  trustGuestRxFilters attribute 98, 220
    no value 220
    yes value 220
  type attribute 98, 100, 220
    bridge value 100, 220

interface element *(continued)*
  type attribute *(continued)*
    direct value 98, 220
    network value 220
interface name 37
io attribute
  of the driver element 78, 95, 210
IOCDS 27, 37, 38
ioeventfd attribute
  of the driver element 210
iothread attribute
  of the driver element 78, 209
iothreadadd virsh command 290
  --config option 290
  --current option 290
  --id option 290
  --live option 290
iothreaddel virsh command 292
  --config option 292
  --current option 292
  --id option 292
  --live option 292
iothreadinfo virsh command 294
  --config option 294
  --current option 294
  --live option 294
iothreads element 51, 69, 78, 221
IP address 37
ip command 37, 38
ip element 222
  address attribute 222
  netmask attribute 222
IPL 53, 114
ISO image 17, 58, 95, 148
  as IPL device 54
  removable 95, 148

K

kdump 175
keepalive interval
  of the virsh command 134
kernel element 51, 53, 55, 223
kernel image file 53, 55, 58
kernel parameters 53, 55
  specifying 55
key element 224
keywrap element 72, 225
KVM 6
KVM guest x, 3, 4
KVM host x
kvm kernel module 6
kvm value
  of the domain type attribute 51, 208
KVM virtual server x, 3, 4
kvm_s390_sie_enter tracepoint event 177

L

layer 2 mode 38
lba value
  of the geometry trans attribute 216
libvirt 4, 6, 33
libvirt configuration file 173
libvirt daemon 113, 145
  starting 113, 145
libvirt XML attribute
  adapter name 90
  address bus 90, 95, 193, 194
  address controller 90, 95, 193
  address cssid 78, 84, 86, 88, 192
  address devno 78, 84, 86, 88, 192
  address ssid 78, 84, 86, 88, 192
  address target 90, 95, 193, 194
  address type 78, 84, 86, 88, 90, 95, 192, 193
  address unit 90, 95, 193, 194
  backend model 195
  boot loadparm 196
  boot order 53, 196
  bridge name 197
  cipher name 72, 198
  cipher state 72, 198
  console type 70, 200
  controller index 88, 201
  controller model 88, 201
  controller ports 201
  controller type 88, 201
  controller vectors 201
  cpu match 202
  cpu mode 202
  device path 204
  disk device 78, 84, 86, 95, 207
  disk type 78, 84, 86, 95, 207
  domain type 51, 208
  driver cache 78, 84, 86, 95, 210
  driver error_policy 210
  driver event_idx 210
  driver io 78, 84, 86, 95, 210
  driver ioeventfd 210
  driver iothread 78, 209
  driver name 78, 84, 86, 95, 210
  driver rerror_policy 210
  driver type 78, 84, 86, 95, 210
  feature name 213
  feature policy 213
  format type 214
  forward mode 215
  geometry cyls 216
  geometry heads 216
  geometry secs 216
  geometry trans 216
  host name 217
  hostdev mode 90, 218
  hostdev rawio 218
  hostdev sgio 218
  hostdev type 90, 218
  interface trustGuestRxFilters 220
  interface type 98, 100, 220
  ip address 222
  ip netmask 222
  log append 70, 226
  log file 70, 226
  mac address 98, 100
  memballoon model 74, 228
  memory dumpCore 67
  memory unit 65
  model type 100
  pool type 242
  source bridge 100
  source dev 78, 98, 249
  source file 84, 86, 95, 249
  source mode 98
  source pool 249

libvirt XML attribute *(continued)*
  source startupPolicy 249
  source volume 249
  target address 254
  target bus 78, 84, 86, 95, 255
  target dev 78, 84, 86, 95, 255
  target port 254
  target type 70, 254
  type arch 51, 258
  type machine 51, 258
  vcpu current 259
  virtualport type 100, 260, 261
  volume type 262
  watchdog action 263
  watchdog model 263
libvirt XML element
  adapter 90
  address 76, 78, 84, 86, 88, 90, 95, 192, 193, 194
  backend 195
  boot 53, 196
  bridge 197
  cipher 72, 198
  cmdline 51, 55, 199
  console 70, 200
  controller 76, 88, 201
  cpu 63, 202
  cputune 51, 62, 203
  device 204
  devices 51, 205
  dhcp 206
  disk 76, 78, 84, 86, 95, 207
  domain 208
  driver 76, 78, 84, 86, 95, 209, 210
  emulator 68, 212
  feature 63, 213
  format 214
  forward 215
  geometry 216
  host 217
  hostdev 76, 90, 218
  initrd 51, 55, 219
  interface 76, 98, 100, 220
  iothreads 51, 69, 78, 221
  ip 222
  kernel 51, 55, 223
  key 224
  keywrap 72, 225
  log 70, 226
  mac 76, 98, 100, 227
  memballoon 74, 228
  memory 51, 65, 67, 229
  memtune 65, 231
  model 76, 100, 232
  model as a child element of cpu 63
  model as a child element of interface 233
  name 51, 234, 235
  network 236
  on_crash 51, 237
  on_poweroff 51
  on_reboot 51, 238
  os 51, 55, 239
  path 240, 241
  pool 242
  readonly 95, 244
  rng 245
  shareable 246
  shares 51, 62, 247

libvirt XML element *(continued)*
  soft_limit 65, 248
  source 76, 78, 84, 86, 95, 98, 100, 249, 253
  target 70, 76, 78, 84, 86, 95, 254, 255, 256, 257
  type 51, 258
  vcpu 51, 61, 259
  virtualport 100, 260, 261
  volume 262
  watchdog 263
libvirt-internal configuration 109, 110
  displaying 122
libvirtd log messages 173
lin_tape device driver 17, 33
Linux scheduling 159
list virsh command 111, 120, 145, 149, 295
  --all option 111, 120
live migration 78, 90, 126, 127
  cancellation 134
  concurrency 132
  concurrent connections 132
  connections 132
  deadlock prevention 132
  destination port 132
  firewall configuration 132
  hardware dependencies 128
  host environments 132
  host setup 132
  hypervisor release 132
  image files 129
  live phase 133
  maximum downtime 133
  MaxStartups parameter 132
  memory intensive workload 133
  memory intensive workloads 134
  messages 300
  migration port range 132
  migration_port_max parameter 132
  non-tunneled 132
  performance considerations 132
  phases 133
  port range 132
  prerequisites 129
  preserving virtual server resources 129
  process 134
  setup 129
  stopped phase 133
  storage keys 133
  verification 134
  virtual server CPUs 129
  virtual server memory 129
  virtual server network 129
  virtual server storage 129
live phase of a migration 133
live subcommand
  or perf kvm stat 177
live virtual server migration
  See live migration
load balancing 29
loadparm attribute
  of the boot element 196
log element 70, 226
  append attribute 226
    off value 226
    on value 226
  file attribute 226
log file
  for console output 70

log messages 173
logging
  console output 70
logging level 173
logical device name 78, 84, 86
logical volume
  configuration 167
  management 167
Logical Volume Manager 78, 167
ls command 78
lscss command 33, 78
lsdasd command 27
lsqeth command 38, 43
lsscsi command 29
lsscss command 29
lszfcp command 29, 33
LUN 33
LVM 78, 167

M

MAC address 21, 43, 100
mac element 76, 98, 100, 227
  address attribute 98, 100, 227
machine attribute
  of the target element 258
  of the type element 51, 126
machine type
  alias value 126
machine type of the virtual server 132
MacVTap
  direct connection 40
  kernel modules 40
  network device driver 21
MacVTap interface 21, 98
  preparing 40
  setting up 40
makedumpfile command 175
managedsave state 114, 183
managedsave virsh command 114, 297
  --bypass-cache option 297
  --paused option 297
  --running option 297
  --verbose option 297
managing
  devices 145
  storage pools 152
  system resources 137
  virtual CPUs 138
  virtual memory 143
  virtual servers 113
  volumes 153
mandatory value
  of the source startupPolicy attribute 249
mapping a virtio block device to a host device 9, 14
master bonded interface 40
match attribute
  of the cpu element 63, 202
maximum downtime 133
maximum number of available virtual CPUs 138
MaxStartups parameter 132
memballoon element 74, 228
  model attribute 74, 228
    none value 74, 228
memory 65
  configuring 65
memory balloon device 74, 75
memory element 51, 65, 67, 229
  dumpCore attribute 67, 229
  unit attribute 65, 229
memory intensive workloads 133, 134
memtune element 65
memtune virsh command 299
  --soft-limit option 299
migrate virsh command 134, 300
  --auto-converge option 134
  --timeout option 134
migrate-getspeed virsh command 134, 303
migrate-setmaxdowntime virsh command 134
migrate-setspeed virsh command 134, 305
  --bandwidth option 134, 305
migrated reason
  of the running state 185
migrating
  CPUs 159
  image files 129
  running virtual servers 127, 134
  storage 9, 14
  virtual servers 125
migrating reason
  of the paused state 186
migration 159
  of a running virtual server to another host 78, 90
    See live migration
  of the storage server 9, 14, 78
  of virtual disks
    See disk migration
  preparing virtual servers for 9, 14, 29
  to a different hypervisor release 126, 127
  to a different IBM Z model 128
migration costs 159
migration port range 132
migration_port_max parameter 132
mode attribute 218
  of the cpu element 63, 202
  of the forward element 215
  of the hostdev element 90, 218
  of the source element 98, 252
  subsystem value 218
model attribute
  of the backend element 195
  of the controller element 88, 201
  of the memballoon element 74, 228
  of the rng element 245
  of the watchdog element 263
model element 63, 76, 98, 100, 232, 233
  type attribute 98, 100, 233
    virtio value 98, 100
model, CPU 63
modifying
  persistent virtual server definitions 110
  the CPU weight 141
  the number of virtual CPUs 138
  virtual server definitions 109
modprobe command 40
monitoring
  virtual servers 119
multipath command 29, 78
multipath device mapper support 9, 14, 29
  configuring 29
  preparing 29
multipathed SCSI device configuration
  example 93

N

N_Port ID virtualization 29
name as a child element of domain 234
name as a child element of network 235
name attribute
  of the adapter element 90, 191
  of the bridge element 197
  of the cipher element 72, 198
  of the driver element 78, 84, 86, 95, 210
  of the feature element 63, 213
  of the host element 217
name element 51
name property
  of virtual servers 4, 49
nat value
  of the forward mode attribute 215
native value
  of the driver io attribute 78, 84, 86, 95, 210
NDP 220
Neighbor Discovery Protocol 220
net-autostart virsh command 306
net-define virsh command 307
net-destroy virsh command 308
net-dumpxml virsh command 309
net-edit virsh command 310
net-info virsh command 311
net-list virsh command 312
net-name virsh command 313
net-start virsh command 314
net-undefine virsh command 315
net-uuid virsh command 316
netmask attribute
  of the ip element 222
network
  configuring 105
network attribute
  of the source element 252
network configuration-XML file 105
network device 21, 38, 75
  attaching 146
  detaching 147
  hotplugging 146
  preparing 37
  unplugging 147
network device driver 4
network element 236
network file system 129
network interface
  activating 38
  configuring 98
  creating 38
  displaying 120
  preparing 38
  setting up 37, 38
network isolation 21
network value
  of the interface type attribute 220
NFS
  See network file system
NIC 100
no value
  of the hostdev rawio attribute 218
  of the interface trustGuestRxFilters attribute 220
node 3
non-tunneled migration 132
none value
  of the driver cache attribute 78, 84, 86, 95, 210
none value *(continued)*
  of the geometry trans attribute 216
  of the memballoon model attribute 74, 228
NPIV 29
number of virtual CPUs 138

O

off value
  of the cipher state attribute 72, 198
  of the driver event_idx attribute 210
  of the driver ioeventfd attribute 210
  of the log append attribute 70, 226
  of the memory dumpCore attribute 67
on value
  of the cipher state attribute 72, 198
  of the driver event_idx attribute 210
  of the driver ioeventfd attribute 210
  of the log append attribute 70, 226
on_crash element 51, 237
on_poweroff element 51
on_reboot element 51, 238
Open vSwitch 21, 43
  package 43
openvswitch command
  start 43
  status 43
openvswitch value
  of the virtualport type attribute 100, 260, 261
optional value
  of the source startupPolicy attribute 249
order attribute
  of the boot element 53, 196
os element 51, 55, 239
OSA adapter port 43
OSA network device 43
OSA-Express feature 37
ovs-vsctl command
  add-bond 43
  add-br 43
  add-port 43
  del-br 43
  show 43

P

para-virtualized device driver 4
parameter file 55
partition 9, 14
path attribute
  of the device element 204
path element 240, 241
path redundancy
  of DASDs 9, 14
  of network devices 21
  of physical volumes 167
  of SCSI disks 9, 14
  of SCSI medium changer devices 17
  of SCSI tape devices 17
paused state 4, 183
  migrating reason 186
  user reason 186
perf kvm stat command
  live subcommand 177
  record subcommand 177
  report subcommand 177

perf tool 177
perf.data.guest 177
performance considerations 132
performance metrics
  collecting 177
  displaying 177
  recording 177
performing
  a live migration 134
physical volume
  configuration 167
  filter 167
policy attribute
  of the feature element 63, 213
pool attribute
  of the source element 249
pool element 242
  type attribute 242
pool-autostart virsh command 317
pool-define virsh command 318
pool-delete virsh command 319
pool-destroy virsh command 320
pool-dumpxml virsh command 321
pool-edit virsh command 322
pool-info virsh command 323
pool-list virsh command 152, 324
pool-name virsh command 325
pool-refresh virsh command 326
pool-start virsh command 327
pool-undefine virsh command 328
pool-uuid virsh command 329
port attribute
  of the target element 254
port range for migration 132
ports attribute
  of the controller element 201
preparing
  bonded interfaces 40
  DASDs 27
  devices for virtual servers 4
  host devices 27
  MacVTap interfaces 40
  multipath device mapper support 29
  network devices 37
  network interfaces 40
  physical volumes 167
  SCSI disks 29
  SCSI medium changer devices 33
  SCSI tapes 33
  virtual Ethernet devices 37
  virtual LAN interfaces 40
  virtual servers for migration 9, 14, 29
  virtual switches 37, 43
  VLAN interfaces 40
preserve text content
  of the on_crash element 51, 237
  of the on_reboot element 238
preserve value
  of the on_crash element 51
primary bridge port 43
property
  name 49
protected key 72
  management operations 72
providing
  I/O threads 69, 146
  ISO images 148
pty value
  of the console type attribute 70, 200
pvscan command 167

Q

qcow2 image file 84, 86
qcow2 value
  of the driver type attribute 84, 86
  of the format type attribute 214
QEMU 6, 9, 14, 33
QEMU command
  info 351
  qemu-img create 84, 86
QEMU core dump
  configuring 67
qemu value
  of the driver name attribute 78, 84, 86, 95, 210
qemu-kvm command 126
qemu-system-s390x user space process 68
qethconf 43

R

ramdisk 53, 55
random number generator 75
  configuring 102
random value
  of the backend model attribute 195
raw image file 84, 86
raw value
  of the driver type attribute 78, 84, 86, 95, 210
  of the format type attribute 214
rawio attribute 218
  no value 218
  of the hostdev element 218
  yes value 218
readonly element 95, 244
reboot virsh command 330
record subcommand
  or perf kvm stat 177
recording
  performance metrics 177
redundant paths 9, 14, 17, 21
relocating
  virtual servers
    See live migration
relocation
  See live migration
removable ISO image
  configuring 95
  removing 148
  replacing 148
removing
  I/O threads 147
  ISO images 148
replacing
  ISO images 148
report subcommand
  or perf kvm stat 177
report value
  of the driver error_policy attribute 210
  of the driver rerror_policy attribute 210
require value
  of the feature policy attribute 63, 213

requisite value
    of the source startupPolicy attribute 249
rerror_policy attribute
    of the driver element 210
restart text content
    of the on_reboot element 238
restored reason
    of the running state 185
resume virsh command 116, 331
resuming
    virtual servers 4, 116
retrieving hardware information
    as a virtual server user 353
RNG device 75, 102
rng element 245
    model attribute 245
root file system 53, 55
root path 53, 55
route value
    of the forward mode attribute 215
run queue 159
running state 4, 183
    booted reason 185
    migrated reason 185
    restored reason 185
    unpaused reason 185

S

s390-ccw-virtio value
    of the type machine attribute 51, 126, 258
s390x value
    of the type arch attribute 51, 258
SAM 90
SAN fabric 9, 14
saved reason
    of the shut off state 184
saved system image 114
saving
    system images 114
schedinfo virsh command 120, 141, 332
    --config option 141, 332
    --live option 141, 332
    cpu_shares attribute 141, 332
scheduling domain 159
scheduling information
    displaying 120
sclp value
    of the target type attribute 70, 254
SCSI Architecture Model 90
SCSI device driver 4
SCSI device name 17, 33, 90
SCSI disk 9, 14, 78
    configuring 78
    preparing 29
    setting up 29
SCSI disk configuration
    example 82
SCSI generic device driver 17
SCSI host 90
SCSI Host Bus Adapter 75
SCSI host number 17, 90
SCSI ID 17, 90
SCSI identifier 9, 14
SCSI LUN 17, 90
SCSI medium changer 17
    configuring 88, 90
SCSI medium changer *(continued)*
    preparing 33
    setting up 33
SCSI stack 33
SCSI stack address 17
SCSI tape 17
    configuring 88, 90
    preparing 33
    setting up 33
scsi value
    of the address type attribute 90, 193
    of the controller type attribute 88, 201
    of the hostdev type attribute 90, 218
    of the target bus attribute 95
secs attribute
    of the geometry element 216
Security-Enhanced Linux 43
SELinux 43
serial device 75
setting up
    bonded interfaces 40
    host devices 27
    MacVTap interfaces 40
    network interfaces 37, 40
    the host for live migration 132
    virtual LAN interfaces 40
    virtual switches 43
    VLAN interfaces 40
setvcpus command
    setvcpus 334
setvcpus virsh command 138, 334
sgio attribute 218
    filtered value 218
    of the hostdev element 218
    unfiltered value 218
shareable element 246
shares element 51, 62, 247
shut off state 4, 111, 183
    destroyed reason 184
    saved reason 184
    shutdown reason 184
    unknown reason 184
shutdown reason
    of the shut off state 184
shutdown virsh command 114, 333
shutting down
    virtual servers 4, 114
shutting down state 183
slave device 40
soft_limit element 65, 248
    unit attribute 248
source element 76, 78, 84, 86, 95, 98, 100, 249, 251, 252, 253
    bridge attribute 100
    dev attribute 78, 84, 86, 98, 249, 252
    file attribute 84, 86, 95, 249
    mode attribute 98, 252
        bridge value 98
    pool attribute 249
    startupPolicy attribute 249
        mandatory value 249
        optional value 249
        requisite value 249
    volume attribute 249
SSH daemon 132
    configuration file 132
ssid attribute
    of the address element 78, 84, 86, 88, 192

standard device name 9, 14
  of a SCSI medium changer 17
  of a SCSI tape 17
standard device node 9, 14, 78
standard interface name 37
start openvswitch command 43
start virsh command 114, 149, 336
  --console option 114, 149
  --force-boot option 114
starting
  libvirt daemon 113, 145
  virtual servers 4, 114
startupPolicy attribute
  of the source element 249
state 4, 183
  crashed 183, 187
  displaying 120
  managedsave 114, 183
  paused 4, 183, 186
  running 4, 183, 185
  shut off 4, 111, 183, 184
  shutting down 183
state attribute
  of the cipher element 72, 198
state-transition diagram 183
  simplified 4
status openvswitch command 43
STHYI instruction 353
stop value
  of the driver error_policy attribute 210
  of the driver rerror_policy attribute 210
stopped phase of a migration 133
storage controller 9, 14
  port 9, 14
storage keys 133
storage migration 78, 167
storage pool
  configuring 103
  managing 152
storage pool configuration-XML file 103
Store Hypervisor Information instruction 353
subchannel set-ID 9, 14, 78, 84, 86
subsystem value
  of the hostdev mode attribute 90, 218
suspend virsh command 116, 338
suspending
  virtual servers 4, 116
symmetric encryption 72
sysfs attribute
  bridge_state 43
system image 297
  saved 114
  saving 114
system journal 173
system resources
  configuring 49
  managing 137

T

tape 75
target attribute
  of the address element 90, 95, 193, 194
target element 70, 76, 78, 84, 86, 95, 254, 255, 256, 257
  address attribute 254
  bus attribute 78, 84, 86, 255
    scsi value 95
target element *(continued)*
  bus attribute *(continued)*
    virtio value 78, 84, 86, 255
  dev attribute 78, 84, 86, 95, 255
  port attribute 254
  type attribute 70, 254
    sclp value 70, 254
    virtio value 70, 254
TDEA 72
TDES 72
terminating
  virtual servers 4, 114
threads value
  of the driver io attribute 210
topology 9, 14
trans attribute
  of the geometry element 216
Triple DEA 72
Triple DES 72
trustGuestRxFilters attribute
  of the interface element 98, 220
tuning
  virtual CPUs 62
  virtual memory 65
type
  of the virtual channel path 75
type attribute 218
  of the address element 78, 84, 86, 88, 90, 95, 192, 193
  of the console element 70, 200
  of the controller element 88, 201
  of the disk element 78, 84, 86, 95, 207
  of the domain element 51, 208
  of the driver element 78, 84, 86, 95, 210
  of the format element 214
  of the hostdev element 90, 218
  of the interface element 98, 100, 220
  of the model element 98, 100, 233
  of the pool element 242
  of the target element 70, 254
  of the virtualport element 100, 260, 261
  of the volume element 262
  scsi value 193, 201, 218
  virtio-serial value 201
type element 51, 258
  arch attribute 258
  machine attribute 258

U

udev-created by-path device node 78
udev-created device node 9, 14, 78
UID 9, 14
undefine virsh command 111, 339
undefined virtual server 4
undefining
  virtual servers 4, 111
unfiltered value
  of the hostdev sgio attribute 218
unique ID 9, 14
unit 33
unit attribute
  of the address element 90, 95, 193, 194
  of the memory element 65
  of the soft_limit element 248
universally unique identifier 9, 14
unknown reason
  of the shut off state 184

unpaused reason
  of the running state 185
unplugging
  devices 147
unsafe value
  of the driver cache attribute 210
uplink port
  creating 43
user reason
  of the paused state 186
user space 49
  configuring 68
user space process 209, 210
user-friendly name 29
UUID 9, 14

V

vcpu element 51, 61, 259
  current attribute 259
vcpucount virsh command 120, 138, 340
  --active option 340
  --config option 340
  --current option 340
  --live option 340
  --maximum option 340
vectors attribute
  of the controller element 201
verifying
  a live migration 134
virsh command 4
  attach-device 146, 268
  change-media 148, 270
  console 149, 272
  define 110, 273
  destroy 114, 274
  detach-device 147, 275
  domblklist 120, 277
  domblkstat 120, 278
  domcapabilities 280
  domiflist 120, 281
  domifstat 120, 282
  dominfo 120, 283
  domjobabort 134, 284
  domstate 120, 285
  dump 175, 286
  dumpxml 88, 122, 287
  edit 110, 288
  inject-nmi 175, 289
  iothreadadd 290
  iothreaddel 292
  iothreadinfo 294
  list 111, 120, 145, 149, 295
  makedumpfile 175
  managedsave 114, 297
  memtune 299
  migrate 134
  migrate-getspeed 134, 303
  migrate-setmaxdowntime 134
  migrate-setspeed 134, 305
  net-autostart 306
  net-define 307
  net-destroy 308
  net-dumpxml 309
  net-edit 310
  net-info 311
  net-list 312
virsh command *(continued)*
  net-name 313
  net-start 314
  net-undefine 315
  net-uuid 316
  pool-autostart 317
  pool-define 318
  pool-delete 319
  pool-destroy 320
  pool-dumpxml 321
  pool-edit 322
  pool-info 323
  pool-list 152, 324
  pool-name 325
  pool-refresh 326
  pool-start 327
  pool-undefine 328
  pool-uuid 329
  reboot 330
  resume 116, 331
  schedinfo 120, 141, 332
  setvcpus 138
  shutdown 114, 333
  start 114, 149, 336
  suspend 116, 338
  undefine 111, 339
  vcpucount 120, 138, 340
  vol-create 341
  vol-delete 342
  vol-dumpxml 343
  vol-info 344
  vol-key 345
  vol-list 153, 346
  vol-name 347
  vol-path 348
  vol-pool 349
virsh command option
  --all 295, 312
  --autodestroy 336
  --autostart 295, 312
  --build 327
  --bypass-cache 297, 336
  --config 141, 268, 270, 275, 332
  --console 336
  --current 270, 275
  --disable 306, 317
  --domain 268, 270, 274, 275, 287, 333, 336
  --eject 270
  --file 268, 275
  --force 270, 272
  --force-boot 336
  --graceful 274
  --id 295
  --inactive 287, 295, 309, 312
  --insert 270
  --live 141, 270, 275, 332
  --machine 280
  --managed-save 295
  --memory-only 286, 299
  --migratable 287
  --mode 333
  --name 295, 312
  --no-autostart 295, 312
  --no-overwrite 327
  --overwrite 327
  --path 270
  --paused 297, 336

virsh command option *(continued)*
  --persistent 275, 295
  --reason 120, 285
  --running 297
  --safe 272
  --security-info 287
  --state-other 295
  --state-paused 295
  --state-running 295
  --state-shutoff 295
  --table 295, 312
  --title 295
  --transient 295
  --update 270
  --update-cpu 287
  --uuid 295, 312
  --verbose 297
  --with-managed-save 295
  --with-snapshot 295
  --without-managed-save 295
  --without-snapshot 295
virsh command-line interface 6
virtio 4
  block device 9, 14, 78, 84, 86
  block device driver 4
  device driver 4
  network device driver 4
virtio value
  of the model type attribute 98, 100
  of the target bus attribute 78, 84, 86, 255
  of the target type attribute 70, 254
virtio-block device 75
virtio-net device 75
virtio-scsi device 75
virtio-scsi value
  of the controller model attribute 88, 201
virtio-serial value
  of the controller type attribute 201
virtual block device 9, 14, 78
  attaching 146
  detaching 147
  device configuration-XML 76
  hotplugging 146
  unplugging 147
virtual channel 78, 84, 86
virtual channel path 9, 14, 75
virtual channel path type 75
virtual channel subsystem 75
virtual channel subsystem-ID 75
virtual control unit model 75
virtual CPU 159
  configuring 61
  configuring the number 61
  Linux management of 159
  managing 138
  model 63
  modifying the weight 141
  tuning 62
virtual CPUs
  actual number 138
  current config 138
  maximum config 138
  maximum live 138
  maximum number 138
virtual DVD 17, 95, 148
virtual DVD drive 17, 148
virtual Ethernet device 21

virtual Ethernet device *(continued)*
  attaching 146
  detaching 147
  device configuration-XML 76
  hotplugging 146
  unplugging 147
virtual Ethernet interface
  preparing 37
virtual HBA 75
  attaching 146
  configuring 88
  detaching 147
  device configuration-XML 76
  hotplugging 146
  unplugging 147
virtual Host Bus Adapter
  configuring 88
  device configuration-XML 76
virtual LAN interface 21, 40
virtual machine relocation
  See live migration
virtual memory 65
  configuring 65
  managing 143
  tuning 65
virtual network
  configuring 105
  for booting 56
virtual SCSI device 17, 88
  attaching 146
  detaching 147
  device configuration-XML 76
  hotplugging 146
  unplugging 147
virtual SCSI-attached CD/DVD drive 95
virtual server x, 3, 4
  browsing 120
  configuration 9, 14
  configuring 4
  crashed 175
  defining 4, 110
  destroying 114
  devices 49
  displaying block devices 120
  displaying information 120
  displaying network interfaces 120
  displaying scheduling information 120
  displaying the state 120
  dumping 175
  kernel panic 175
  managing 113
  migrating 125, 127
  monitoring 119
  name 4, 49, 51
  persistent definition
    creating 109
    defining 109
    deleting 109
    editing 110
    modifying 109, 110
  preparing 4
  properties 49
  relocation
    See live migration
  resuming 4, 116
  shutting down 4, 114
  starting 4, 114

virtual server *(continued)*
  state 4, 183
    crashed 183, 187
    paused 4, 183, 186
    running 4, 183, 185
    shut off 4, 183, 184
    shutting down 183
  stopping 4
  suspending 4, 116
  system resources 49
  terminating 114
  undefining 4, 111
virtual switch 21
  configuring 98, 100
  creating 43
  preparing 37, 43
  setting up 43
virtualization
  of DASDs 9, 14
  of network devices 21
  of SCSI disks 9, 14
  of SCSI medium changer devices 17
  of SCSI tapes 17
virtualization components 6
virtualport element 100, 260, 261
  type attribute 100, 260, 261
    openvswitch value 100, 260
VLAN ID 40
VLAN interface 40
vol-create virsh command 341
vol-delete virsh command 342
vol-dumpxml virsh command 343
vol-info virsh command 344
vol-key virsh command 345
vol-list virsh command 153, 346
vol-name virsh command 347
vol-path virsh command 348
vol-pool virsh command 349
volume
  managing 153
volume attribute
  of the source element 249
volume configuration-XML file 103
volume element 262
  type attribute 262
VXLAN tunnel 43

W

watchdog device
  configuring 71
watchdog device driver 177
watchdog element 263
  action attribute 263
  model attribute 263
watchdog timer 71
weight-fraction 160
workload
  memory intensive 133, 134
worldwide port name 9, 14
wrapping key 72
writeback value
  of the driver cache attribute 210
writethrough value
  of the driver cache attribute 84, 86, 210
WWPN 9, 14, 29

X

XML format 6

Y

yes value
  of the hostdev rawio attribute 218
  of the interface trustGuestRxFilters attribute 220

Z

zipl
  command 53, 55
  configuration file 53, 55
znetconf command 37, 38, 43, 98

Readers’ Comments — We'd Like to Hear from You

Linux on Z and LinuxONE
KVM Virtual Server Management
November 2017

Publication No. SC34-2752-04

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send your comments via email to: eservdoc@de.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You

SC34-2752-04

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SC34-2752-04

Cut or Fold
Along Line

SC34-2752-04