

Linux on z Systems and LinuxONE



KVM Virtual Server Management

March 2017

Linux on z Systems and LinuxONE



KVM Virtual Server Management

March 2017

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

© **Copyright IBM Corporation 2015, 2017.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document	ix
How this document is organized	ix
Conventions and assumptions used in this publication	x
Where to get more information	xi
Other publications for Linux on z Systems	xi
<hr/>	
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
<hr/>	
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
<hr/>	
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
Example of an initial installation	56
Configuring virtual CPUs	60
Configuring the number of virtual CPUs	60
Tuning virtual CPUs	61
Configuring virtual memory.	62
Configuring the amount of virtual memory.	62
Tuning virtual memory	62
Configuring the collection of QEMU core dumps.	64

Configuring the user space	65
Configuring persistent devices	66
Configuring the console	67
Configuring a watchdog device.	68
Disabling protected key encryption	69
Suppressing the automatic configuration of a default memory balloon device	71
Chapter 10. Configuring devices.	73
Device configuration-XML	74
Configuring virtual block devices	76
Configuring a DASD or SCSI disk.	76
Configuring an image file as storage device	82
Configuring a volume as storage device.	84
Configuring virtual SCSI devices	86
Configuring a virtual HBA	86
Configuring a SCSI tape or medium changer device.	88
Example of a multipathed SCSI tape and medium changer device configuration	91
Configuring a virtual SCSI-attached CD/DVD drive.	93
Configuring virtual Ethernet devices	96
Configuring a MacVTap interface	96
Configuring a virtual switch.	98
Configuring a random number generator	100
Chapter 11. Configuring storage pools	101
Storage pool and volume configuration-XMLs	101
<hr/>	
Part 4. Operation	103
Chapter 12. Creating, modifying, and deleting persistent virtual server definitions.	105
Defining a virtual server	106
Modifying a virtual server definition	106
Undefined a virtual server.	107
Chapter 13. Managing the virtual server life cycle.	109
Starting a virtual server	110
Terminating a virtual server	110
Suspending a virtual server.	112
Resuming a virtual server	112
Chapter 14. Monitoring virtual servers.	115
Browsing virtual servers.	116
Displaying information about a virtual server	116
Displaying the current libvirt-internal configuration	118
Chapter 15. Migration	121
Definition of a virtual server on different hosts using the same configuration-XML	122
Live virtual server migration	123
Hypervisor release	123
Live migration setup	124
Phases of a live migration	128
Performing a live migration	129
Chapter 16. Managing system resources	133
Managing virtual CPUs	134
Modifying the number of virtual CPUs.	134
Modifying the virtual CPU weight	137
Managing virtual memory	139

Chapter 17. Managing devices	141
Attaching a device.	142
Detaching a device	143
Replacing a virtual DVD	143
Connecting to the console of a virtual server	145
Chapter 18. Managing storage pools	147
Storage pool management commands	148
Volume management commands	149
<hr/>	
Part 5. Best practices and performance considerations	151
Chapter 19. CPU management	153
Linux scheduling	153
CPU weight	154
Chapter 20. Memory management	157
Chapter 21. Storage management.	161
I/O threads	161
Logical volume management	161
<hr/>	
Part 6. Diagnostics and troubleshooting	165
Chapter 22. Logging	167
Log messages	167
Specifying the logging level of the libvirt log messages	167
Chapter 23. Dumping	169
Creating a virtual server dump on the host	169
Creating a dump on the virtual server	169
Chapter 24. Collecting performance metrics	171
<hr/>	
Part 7. Reference	175
Chapter 25. Virtual server life cycle	177
shut off	178
running	179
paused	180
crashed	181
in shutdown.	182
Chapter 26. Selected libvirt XML elements	183
<adapter> as child element of <source>	185
<address> as child element of <controller>, <disk>, <interface>, and <memballoon>	186
<address> as child element of <hostdev> or <disk>	187
<address> as child element of <source>	188
<backend>	189
<boot>	190
<cipher>	191
<cmdline>	192
<console>	193
<controller>	194
<cputune>	195
<device>	196
<devices>	197

<disk>	198
<domain>	199
<driver> as child element of <controller>	200
<driver> as child element of <disk>	201
<emulator>	203
<format>	204
<geometry>	205
<host>	206
<hostdev>	207
<initrd>	208
<interface>	209
<iothreads>	210
<kernel>	211
<key>	212
<keywrap>	213
<log>	214
<mac>	215
<memballoon>	216
<memory>	217
<memtune>	219
<model>	220
<name>	221
<on_crash>	222
<on_reboot>	223
<os>	224
<path> as child element of <pool><target>	225
<path> as child element of <volume><target>	226
<pool>	227
<readonly>	229
<rng>	230
<shareable>	231
<shares>	232
<soft_limit>	233
<source> as child element of <disk>	234
<source> as child element of <hostdev>	236
<source> as child element of <interface>	237
<source> as child element of <pool>	238
<target> as child element of <console>	239
<target> as child element of <disk>	240
<target> as child element of <pool>	241
<target> as child element of <volume>	242
<type>	243
<vcpu>	244
<virtualport>	245
<volume>	246
<watchdog>	247

Chapter 27. Selected virsh commands. 249

attach-device	252
change-media	254
console	256
define	257
destroy	258
detach-device	259
dombklist	261
dombkstat	262
domiflist	264
domifstat	265
dominfo	266
domjobabort	267
domstate	268

dump	269
dumpxml.	270
edit.	271
inject-nmi	272
iothreadadd	273
iothreaddel	275
iothreadinfo	277
list	278
managedsave	280
memtune	282
migrate	283
migrate-getspeed	286
migrate-setmaxdowntime	287
migrate-setspeed	288
pool-autostart	289
pool-define	290
pool-delete	291
pool-destroy.	292
pool-dumpxml	293
pool-edit	294
pool-info	295
pool-list	296
pool-name	297
pool-refresh	298
pool-start.	299
pool-undefine	300
pool-uuid	301
reboot	302
resume	303
schedinfo.	304
shutdown	305
setvcpus	306
start	308
suspend	310
undefine	311
vcpucount	312
vol-create.	313
vol-delete.	314
vol-dumpxml	315
vol-info	316
vol-key	317
vol-list.	318
vol-name	319
vol-path	320
vol-pool	321
Chapter 28. Selected QEMU commands	323
QEMU monitor commands	323
Examples for the use of the qemu-img command	323
Chapter 29. Hypervisor information for the virtual server user	325
<hr/>	
Part 8. Appendixes	331
Accessibility.	333
Notices	335
Trademarks	336

Index 337

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 z Systems™ hardware.

For an appropriate KVM host setup, refer to your host administration documentation. Depending on the product or distribution you use, this is *KVM for IBM z Systems: System Administration, SC27-8237*, or your distribution documentation.

For a scenario of defining and operating a KVM virtual server, see *KVM Virtual Server Quick Start, SC34-2753*.

For a description of the installation of SLES 12 as a guest operating system, see *Installing SUSE Linux Enterprise Server 12 as a KVM Guest, SC34-2755*.

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 on z Systems. The described subset is not complete.

KVM users familiar with other platforms should be aware that:

- The use of some configuration elements might be different on the z Systems platform.
- Not all available commands, command options or command output are relevant on a z Systems 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 z Systems 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 z Systems 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 z Systems 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

This document describes how to set up devices and interfaces for Linux on z Systems which are not persistent. If you need to make your changes persistent, refer to your host administration documentation, or use commonly available tools.

Depending on the product or distribution you use, your host administration documentation is *KVM for IBM z Systems: System Administration, SC27-8237* or your distribution documentation.

Terminology

This document uses the following terminology:

KVM virtual server, virtual server

Virtualized z Systems 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 URLs 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 preceded 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

You can find publications for Linux on z Systems 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)
- *KVM Virtual Server Management*, SC34-2752
- *KVM Virtual Server Quick Start*, SC34-2753
- *KVM Virtual Server Management Tools*, SC34-2763
- *Installing SUSE Linux Enterprise Server 12 as a KVM Guest*, SC34-2755

- *Device Drivers, Features, and Commands for Linux as a KVM Guest* (distribution-specific editions)
- *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 Set up a Terminal Server Environment on z/VM[®]*, SC34-2596

These publications are available on developerWorks at

www.ibm.com/developerworks/linux/linux390/documentation_dev.html

- *Device Drivers, Features, and Commands*, SC33-8411
- *Using the Dump Tools*, SC33-8412
- *KVM Virtual Server Management*, SC34-2752
- *KVM Virtual Server Quick Start*, SC34-2753
- *KVM Virtual Server Management Tools*, SC34-2763
- *Installing SUSE Linux Enterprise Server 12 as a KVM Guest*, SC34-2755
- *Device Drivers, Features, and Commands for Linux as a KVM Guest*, SC34-2754
- *How to Improve Performance with PAV*, SC33-8414
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *Kernel Messages*, SC34-2599
- *libica Programmer's Reference*, SC34-2602
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Linux on z Systems Troubleshooting*, SC34-2612
- *Linux Health Checker User's Guide*, SC34-2609

Tuning hints and tips for Linux on z Systems are available at

www.ibm.com/developerworks/linux/linux390/perf

Part 1. General concepts

Chapter 1. Overview	3	Chapter 3. SCSI tapes and medium changers as virtual SCSI devices	17
Virtual server management tasks	4		
Virtualization components	6		
Chapter 2. Virtual block devices	9	Chapter 4. Network devices as virtual Ethernet devices	21
DASDs and SCSI disks	9		
Image files and logical volumes	14		
Storage pools	14		

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.

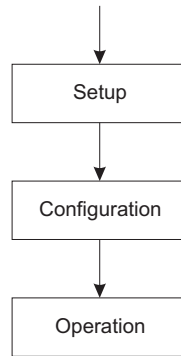


Figure 1. Virtual server administrator's tasks

A *KVM virtual server* consists of virtualized z Systems 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*.

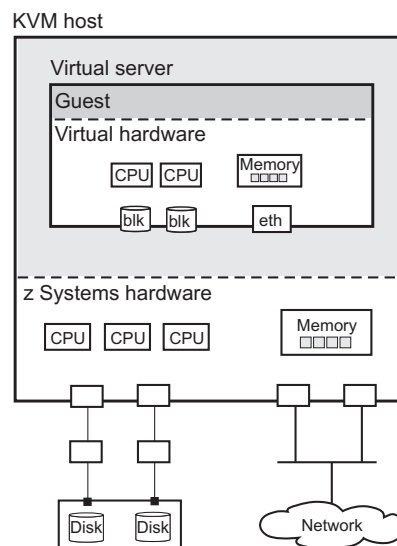


Figure 2. KVM host with a virtual server including a guest operating system

Virtual server management tasks

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

1. Device setup

The virtual server user does not see the device specifics of the devices that you provide for it. It can handle them only on an abstraction layer that does not allow them to be configured. You need to prepare the adapter hardware, the physical disk devices, and the network devices to be used by the virtual server. 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 persistent devices.

You can also configure *hotplug devices* by using *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.

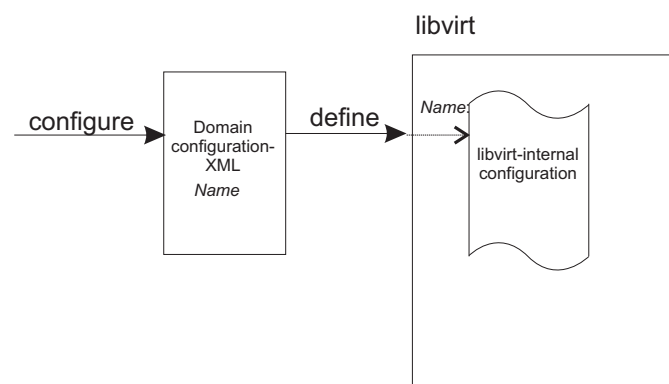


Figure 3. Creating a persistent virtual server definition

- 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 dynamically attach devices to or detach devices from a defined virtual server. If the virtual server is running, the devices are 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 103.

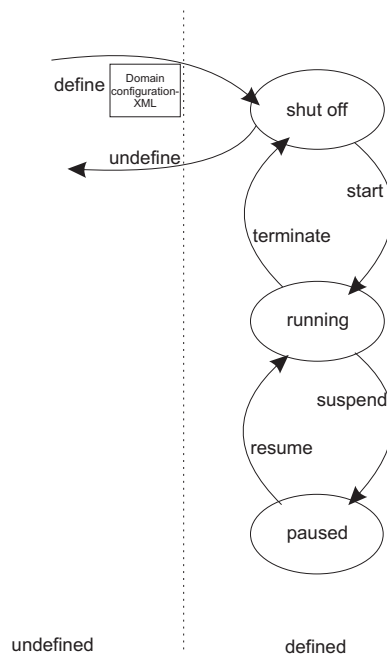


Figure 4. Simplified state-transition diagram of a virtual server

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.

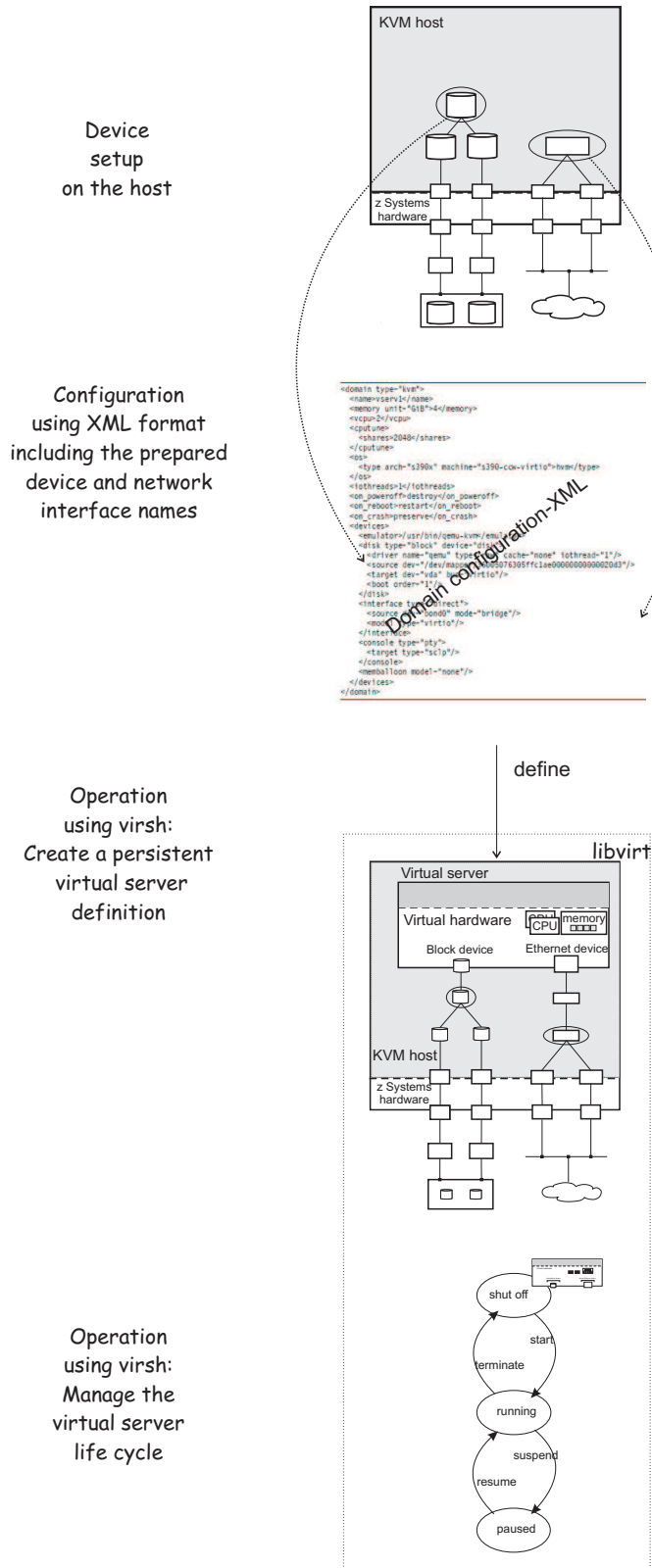


Figure 5. Virtual server administrator tasks using XML format and the virsh command-line interface

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
- *KVM for IBM z Systems: System Administration*, SC27-8237

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. Path redundancy in the setup of FC-attached SCSI disks guarantees high availability of the devices. Analogous, multipathing is implemented in the z Systems hardware of DASDs.

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.

In particular, prepare multipathing to guarantee high availability, 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 persistent for a virtual server in its domain configuration-XML file and hotplug devices in a separate device configuration-XML file.

See Chapter 10, “Configuring devices,” on page 73 and “Configuring virtual block devices” on page 76.

DASD and SCSI disk configuration topology

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

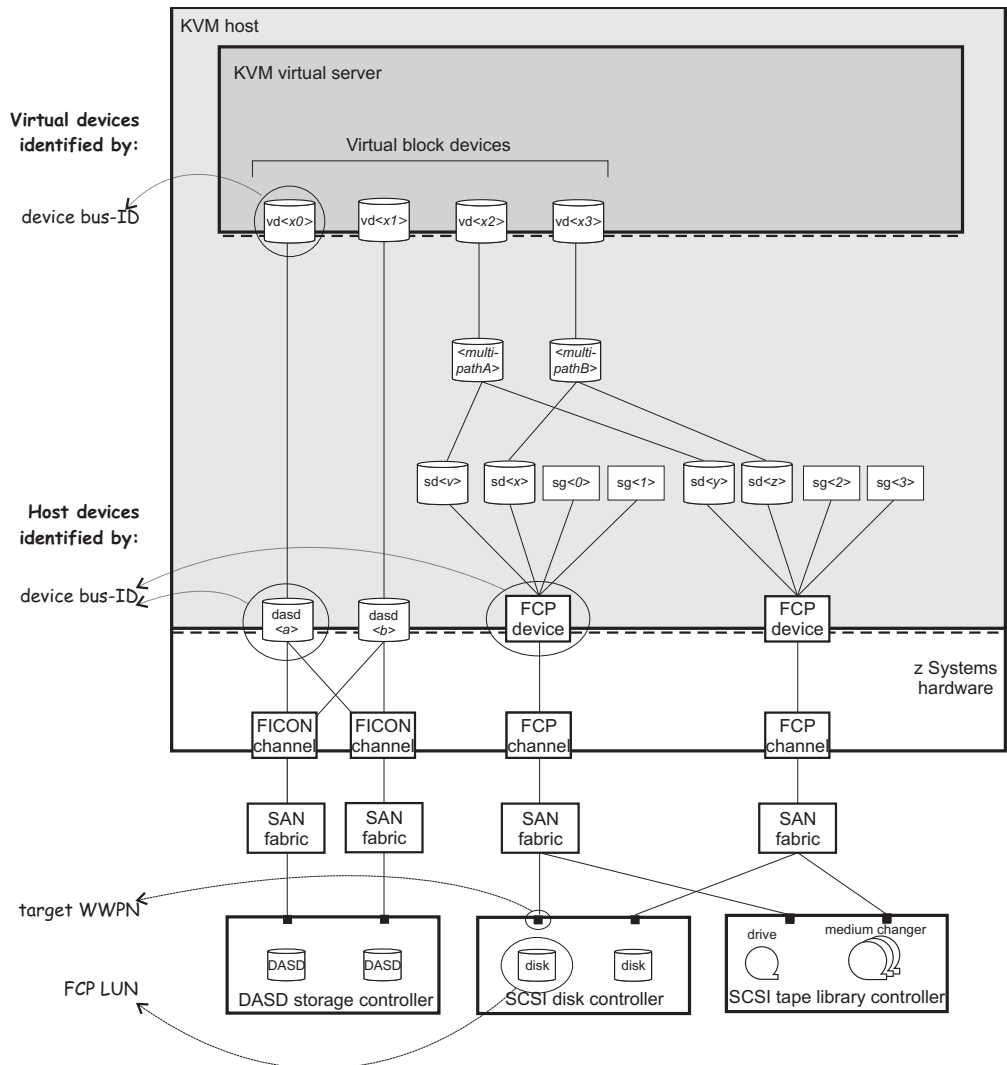


Figure 6. Multipathed DASD and SCSI disks configured as virtual block devices

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:

<code>/dev/sda</code>	for SCSI disks on the host.
<code>/dev/dasda</code>	for DASDs on the host.
<code>/dev/vda</code>	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-zfc-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-36005076303ffc5620000000000010ea`
where:
 `scsi-36005076303ffc562000000000000010ea` 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/36005076305ffc1ae0000000000021d5
/dev/mapper/36005076305ffc1ae0000000000021d5p1
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 persistent for a virtual server in its domain configuration-XML file and hotplug devices in a separate device configuration-XML file.

See Chapter 10, “Configuring devices,” on page 73 and “Configuring an image file as storage device” on page 82.

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:

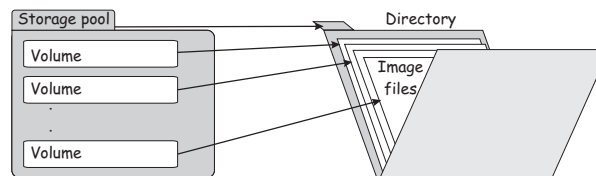


Figure 7. Storage pool backed by the image files of a directory

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

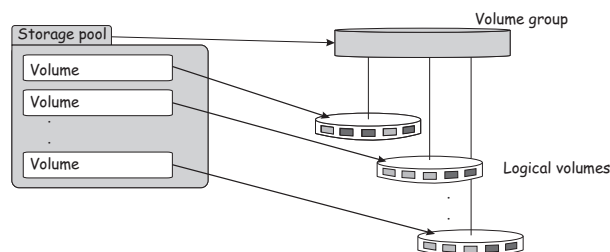


Figure 8. Storage pool backed by the logical volumes of a volume group

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.
See Chapter 11, “Configuring storage pools,” on page 101.
3. Configure the storage pool including its volumes.
See “Configuring a volume as storage device” on page 84.

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:

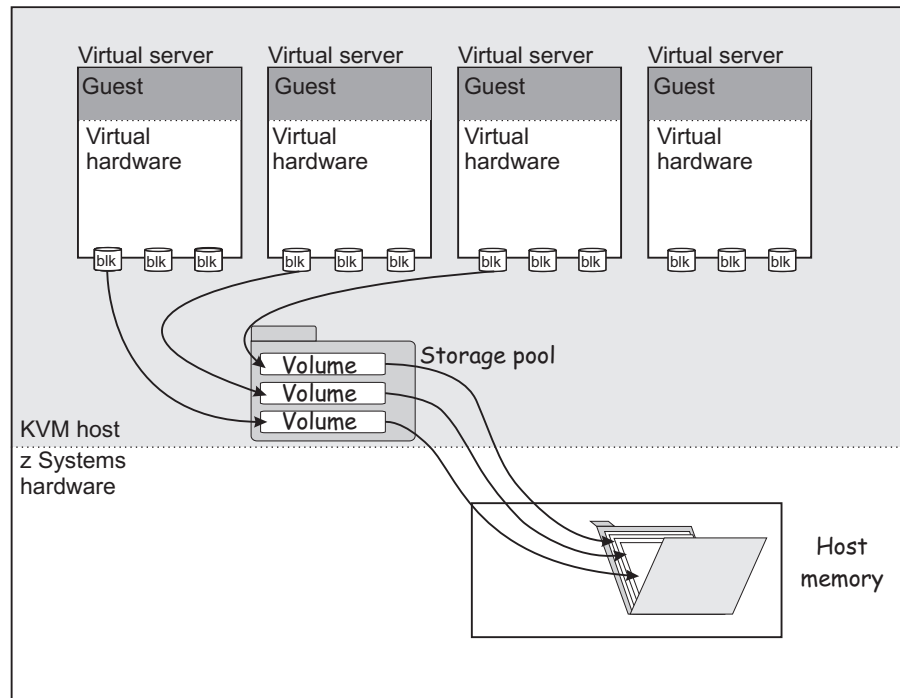


Figure 9. Storage pool volumes configured as virtual block devices

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 18, “Managing storage pools,” on page 147.

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 as hotplug device.
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 73 and “Configuring virtual SCSI devices” on page 86.

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.

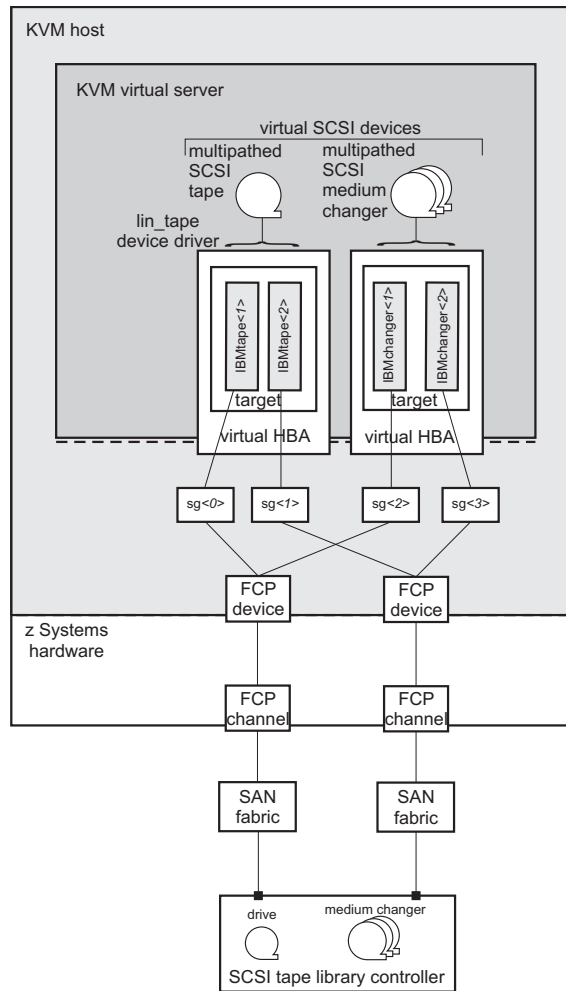


Figure 10. Multipath SCSI tapes and SCSI medium changer devices configured as virtual SCSI devices

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 publications

- *Device Drivers, Features, and Commands*, SC33-8411
- *KVM for IBM z Systems: System Administration*, SC27-8237

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:

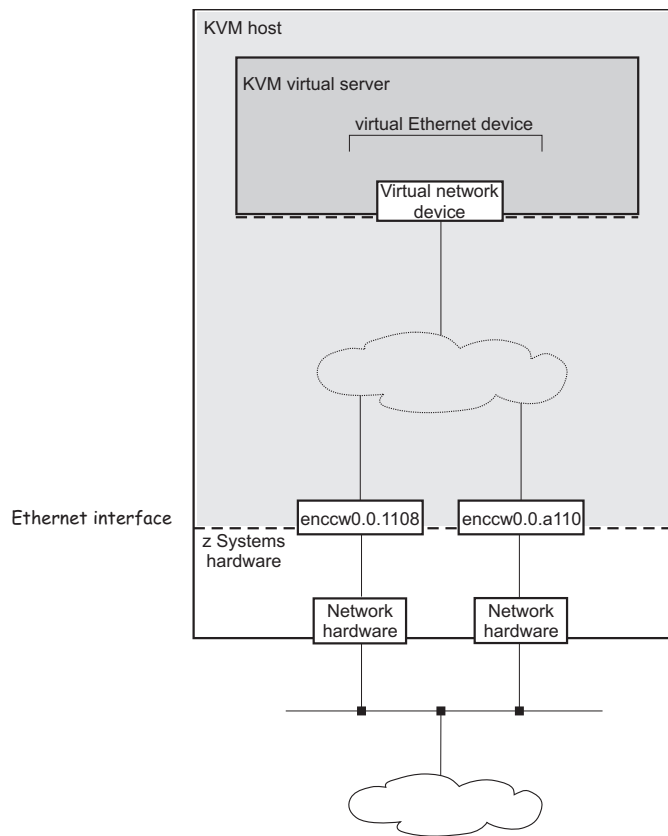


Figure 11. Highly reliable virtual network device configuration

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.

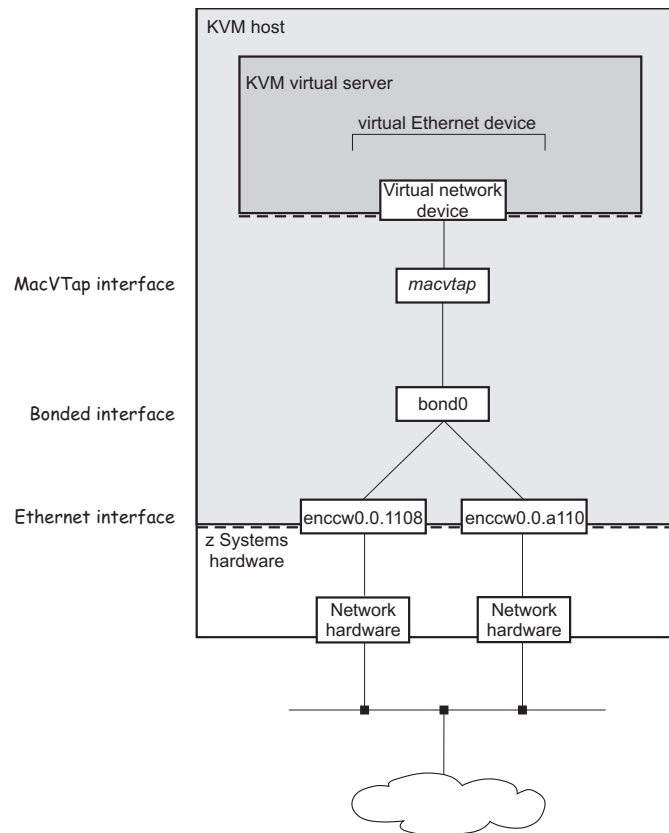


Figure 12. Configuration using a direct MacVTap connection

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:

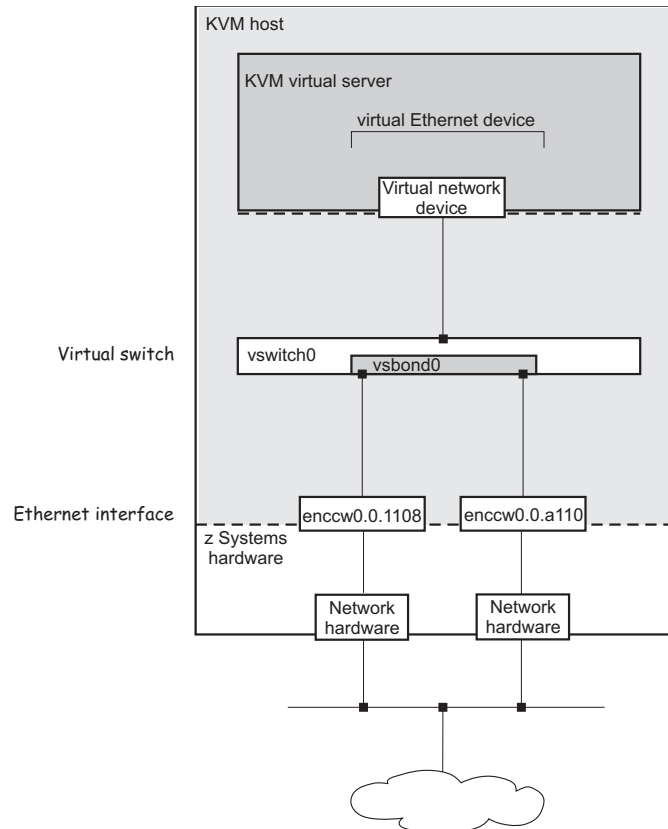


Figure 13. Configuration using a virtual switch

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
- *KVM for IBM z Systems: System Administration*, SC27-8237
- 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 96

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 8. Preparing network devices.	37
Chapter 6. Preparing SCSI disks	29	Creating a network interface.	38
Chapter 7. Preparing SCSI tape and medium changer devices	33	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 publications describe in detail how to configure, prepare, and work with DASDs:

- *Device Drivers, Features, and Commands*, SC33-8411
- *KVM for IBM z Systems: System Administration*, SC27-8237

Procedure

The following steps describe a dynamic DASD setup on the host.

To set up DASDs persistently across host reboots, refer to 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 76.

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
- *KVM for IBM z Systems: System Administration*, SC27-8237

Procedure

The following steps describe a dynamic SCSI disk setup on the host.

If you want to set up a SCSI disk persistently across host reboots, refer to 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 36005076305ffc1ae0000000000023bd
[1:0:16:1086210083] disk IBM 2107900 .166 /dev/sdk 36005076305ffc1ae0000000000023be
[1:0:16:1086275619] disk IBM 2107900 .166 /dev/sdo 36005076305ffc1ae0000000000023bf
[2:0:17:1086079011] disk IBM 2107900 2440 /dev/sdq 36005076305ffc1ae0000000000023bc
...
```

The device mapper-created device node that you can use to uniquely reference the multipathed SCSI disk 36005076305ffc1ae0000000000023bc is: `/dev/mapper/36005076305ffc1ae0000000000023bc`

What to do next

Configure the SCSI disks as described in “Configuring a DASD or SCSI disk” on page 76.

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/zfc/<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
- *KVM for IBM z Systems: System Administration*, SC27-8237

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

Procedure

The following steps describe a dynamic SCSI tape or medium changer setup on the host.

If you want to set up a SCSI tape or medium changer persistently across host reboots, refer to 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 (<i>always</i>)
SCSI ID	2

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 88.

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.

The following publications describe in detail how to set up network devices on the host:

- *Device Drivers, Features, and Commands*, SC33-8411
- *KVM for IBM z Systems: System Administration*, SC27-8237

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 96.

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 dynamic network interface setup on the host. If this network device is expected to persist over subsequent host reboots, you need to configure it persistently.

For a description of the necessary steps, refer to 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 dynamic 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 dynamic bonded interface setup on the host. If this bonded interface is expected to persist over subsequent reboots, you need to configure it persistently.

For a description of the necessary steps, refer to 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 96
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:

- *KVM for IBM z Systems: System Administration*, SC27-8237
- 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>
```

3. Optional: If you want to create GRE or VXLAN tunnels, see *KVM for IBM z Systems: System Administration, SC27-8237*.

Example

Set up a virtual switch vswitch0, which groups the network interfaces encw0.0.1108 and encw0.0.a112 to a bonded interface vsbond0:

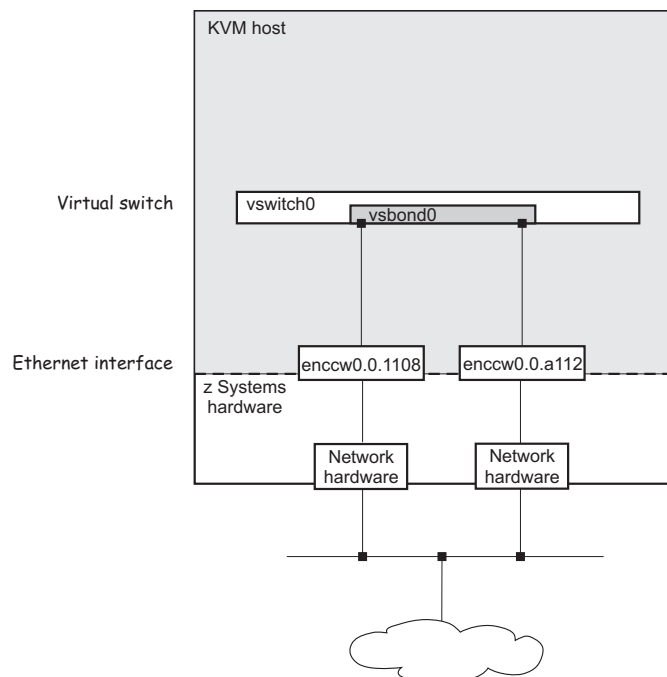


Figure 14. Virtual switch with a bonded interface

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 encw0.0.1108 encw0.0.a112
# ovs-vsctl show
...
  Bridge "vswitch0"
    Port "vsbond0"
      Interface "encw0.0.1108"
      Interface "encw0.0.a112"
    Port "vswitch0"
      Interface "vswitch0"
        type: internal
  ...
```

Related tasks:

“Configuring a virtual switch” on page 98
Configure virtual switches as virtual Ethernet devices.

Part 3. Configuration

Chapter 9. Configuring a virtual server	49	Device configuration-XML	74
Domain configuration-XML	51	Configuring virtual block devices	76
Configuring the boot process	53	Configuring a DASD or SCSI disk.	76
Configuring a DASD or SCSI disk as IPL device	53	Example of a DASD configuration.	79
Configuring an ISO image as IPL device.	54	Example of a SCSI disk configuration.	80
Configuring a kernel image file as IPL device	55	Configuring an image file as storage device	82
Example of an initial installation	56	Configuring a volume as storage device	84
Configuring virtual CPUs	60	Configuring virtual SCSI devices	86
Configuring the number of virtual CPUs	60	Configuring a virtual HBA	86
Tuning virtual CPUs	61	Configuring a SCSI tape or medium changer device	88
Configuring virtual memory.	62	Example of a multipathed SCSI tape and medium changer device configuration	91
Configuring the amount of virtual memory.	62	Configuring a virtual SCSI-attached CD/DVD drive	93
Tuning virtual memory	62	Configuring virtual Ethernet devices	96
Configuring the collection of QEMU core dumps.	64	Configuring a MacVTap interface	96
Configuring the user space	65	Configuring a virtual switch.	98
Configuring persistent devices	66	Configuring a random number generator	100
Configuring the console	67	Chapter 11. Configuring storage pools	101
Configuring a watchdog device.	68	Storage pool and volume configuration-XMLs	101
Disabling protected key encryption	69		
Suppressing the automatic configuration of a default memory balloon device	71		
Chapter 10. Configuring devices	73		

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 60.
 - c. Configure memory.
See “Configuring virtual memory” on page 62.
 - d. Optional: Configure the collection of QEMU core dumps.
See “Configuring the collection of QEMU core dumps” on page 64.
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 65.
 - b. Configure persistent devices.
See “Configuring persistent devices” on page 66.
 - c. Configure the console device.
See “Configuring the console” on page 67.
 - d. Optional: Configure a watchdog device.
See “Configuring a watchdog device” on page 68.
 - 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 69.
 - 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 71.
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 106.

Domain configuration-XML

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

Root element

domain

Specify:

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 when it crashes. Specify the preserve value.

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-kvm</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iotread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae0000000000020d3"/>
      <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="sc1p"/>
    </console>
    <memballoon model="none"/>
  </devices>
</domain>
```

Related reference:

Chapter 26, “Selected libvirt XML elements,” on page 183

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.

For a description of the guest installation process, see:

- *KVM for IBM z Systems: System Administration, SC27-8237*
- *Installing SUSE Linux Enterprise Server 12 as a KVM Guest, SC34-2755*

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 persistent devices” on page 66 and “Configuring a DASD or SCSI disk” on page 76.
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 190).

boot order attribute:	<number>
-----------------------	----------

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

Example

The following domain configuration-XML configures V1, which is booted from the virtual block device 0xe714:

```

<domain type="kvm">
  <name>V1</name>
  ...
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iotread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae0000000000021d5"/>
      <target dev="vda" bus="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe714"/>
    </disk>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iotread="2"/>
      <source dev="/dev/mapper/36005076305ffc1ae0000000000021d7"/>
      <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 the virtual block device 0xe716:

```

<domain type="kvm">
  <name>V2</name>
  ...
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iotread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae0000000000021d5"/>
      <target dev="vda" bus="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe714"/>
    </disk>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iotread="2"/>
      <source dev="/dev/mapper/36005076305ffc1ae0000000000021d7"/>
      <target dev="vdb" bus="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
      <boot order="1"/>
    </disk>
    ...
  </devices>
</domain>

```

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 93.

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.

- 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 190).

boot order attribute: <number>

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

Example

- 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
...
```

- 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

- 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 208).

```
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 211).

```
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 192).

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/36005076305ffclae0000000000021d7"/>
  <target dev="vdb" bus="virtio"/>
  <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
</disk>
```

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-kvm</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" iotthread="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-kvm</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 administration documentation or in *Installing SUSE Linux Enterprise Server 12 as a KVM Guest, SC34-2755*.
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-kvm</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>
  </devices>
</domain>

```



```
<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-kvm</emulator>
  <!-- guest IPL disk -->
  <disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none"
      io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae0000000000021d7"/>
    <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 19, “CPU management,” on page 153

Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks:

“Managing virtual CPUs” on page 134

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 244).

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

<code>vcpu</code> element:	<code><number-of-CPU></code>
----------------------------	------------------------------------

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.

<code>vcpu</code> <code>current</code> attribute:	<code><number></code>
---	-----------------------------

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 134).

```
<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 154.

Procedure

Use the `cputune` element to group CPU tuning elements.

You specify the CPU weight by using the `shares` element (see “`<shares>`” on page 232).

shares element: `<CPU-weight>`

Example

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

Configuring virtual memory

Configure the virtual server memory.

Related concepts:

Chapter 20, “Memory management,” on page 157

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 139

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 217).

memory element:	<i><memory-size></i>
memory unit attribute:	<i><memory-unit></i>

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 20, “Memory management,” on page 157.

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 233).

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

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 217)

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 Systems™ 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 203).

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:	<i><emulator-file></i>
-------------------	------------------------------

In KVM for IBM z Systems, `/usr/bin/qemu-kvm` is a shell script that will invoke `/usr/bin/qemu-system-s390x`. If you do not specify the emulator element, libvirt will generate `/usr/bin/qemu-kvm` in the emulator element of the libvirt-internal configuration.

Example:

```
<devices>
  <emulator>/usr/bin/qemu-kvm</emulator>
  ...
</devices>
```

Configuring persistent devices

The domain configuration-XML file specifies virtual devices for the virtual server that are persistent across virtual server reboots. Hotplug devices are configured in separate device configuration-XML files.

Before you begin

Ensure that the devices are prepared for the use of the virtual server.

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 161.

iothreads element:	<code><number-of-IOthreads></code>
--------------------	--

(see “<iothreads>” on page 210)

Example:

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

2. Specify a configuration-XML for each device. Chapter 10, “Configuring devices,” on page 73 describes how to specify a configuration-XML for a device.
3. For each persistent device, place the configuration-XML as child element of the devices element in the domain configuration-XML file. Please note that hotplug devices are configured in separate device configuration-XML files.

Example

```
<domain type="kvm">
  <iothreads>1</iothreads>
  ...
  <devices>
    ...
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iotread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae0000000000020d3"/>
      <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 193).

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 239).

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 214).

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 145

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 reports a positive status to the watchdog device at regular intervals. If the watchdog device does not receive a positive report within 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 247).

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 Systems 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 213).

Its child element cipher (see “<cipher>” on page 191) 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 216)

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 or 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 76
 - “Configuring virtual SCSI devices” on page 86
 - “Configuring virtual Ethernet devices” on page 96

2. To configure a persistent device, enter the device configuration as child element of the devices element in the domain configuration-XML file.
3. To configure a hotplug device, enter the device configuration in a separate device configuration-XML file.

Device configuration-XML

Configure a hotplug device with a device configuration-XML file.

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" iotread="1"/>
  <source dev="/dev/mapper/36005076305ffc1ae0000000000021d5"/>
  <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 26, “Selected libvirt XML elements,” on page 183

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 82
- “Configuring a volume as storage device” on page 84

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 161.

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 198)

- 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 201)

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

For persistent devices:

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 hotplug devices:

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 161.

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

target bus attribute:	<code>virtio</code>
-----------------------	---------------------

(see “`<target>` as child element of `<disk>`” on page 240)

- 2. Identify the device on the host.

Specify a device node of the device.

source dev attribute:	<code><device-node></code>
-----------------------	----------------------------------

(see “`<source>` as child element of `<disk>`” on page 234)

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 240)

- 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 186)

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" iotthread="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" iotthread="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" iotread="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" iotread="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 323 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 198)

- 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 201)

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 240)

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 234)

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 240)

- 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 186)

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 198)

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 201)

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 240)

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 234)

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 240)

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 186)

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 or 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 88
 - “Configuring a virtual SCSI-attached CD/DVD drive” on page 93

Example

See “Example of a multipathed SCSI tape and medium changer device configuration” on page 91.

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 194).

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 200):

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 161.

Example:

```

<domain>
  ...
  <iotreads>2</iotreads>
  ...
  <devices>
    <controller type="scsi" model="virtio-scsi" index="0">
      <driver iotread="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 186)

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 86), 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 FC-attached SCSI tape or medium changer devices as hotplug devices, that is, create a separate device configuration-XML file for each device. Attach the device only if necessary, and detach the device 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 207).

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 185 and “<address> as

child element of <source>” on page 188)

- 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 187)

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: $0 \leq \text{<target>} < 256$

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

$0 \leq \text{<unit>} < 256$

SCSI LUN := <unit>

$256 \leq \text{<unit>} \leq 16383$

SCSI LUN := $0x\text{<unit>} \vee 0x4000$

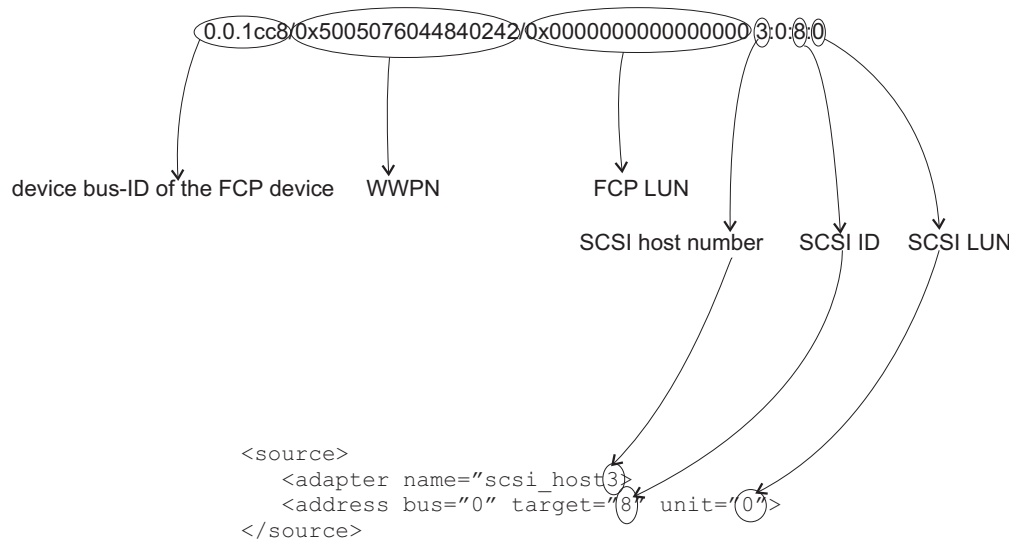
Tip: Choose a value between 0 and 255, because these values are identically mapped to the SCSI LUN on the virtual server.

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:



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 86), 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 198).

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 201).

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 240).

target bus attribute:	scsi
-----------------------	------

- d. Specify the virtual DVD as read-only using the `readonly` element (see “<readonly>” on page 229).

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 234). 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 240).

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 187)

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: $0 \leq \text{<target>} < 256$

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

$0 \leq \text{<unit>} < 256$
SCSI LUN := <unit>

$256 \leq \text{<unit>} \leq 16383$
SCSI LUN := $0x\text{<unit>} \vee 0x4000$

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 143

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 98

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 209). 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 215)

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 237)

3. Specify the model type (see “<model>” on page 220).

model type attribute:	virtio
-----------------------	--------

Example

- To configure bonded interface bond0:

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

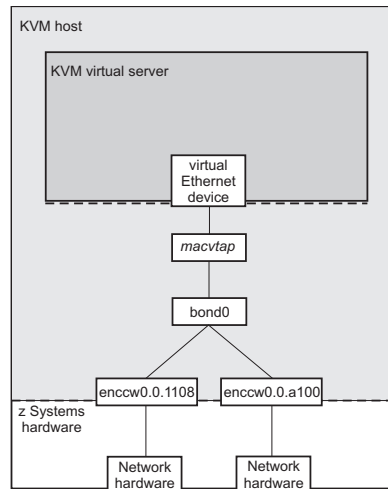


Figure 15. Direct interface type which configures a bonded interface

- To configure virtual LAN bond0.623:

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

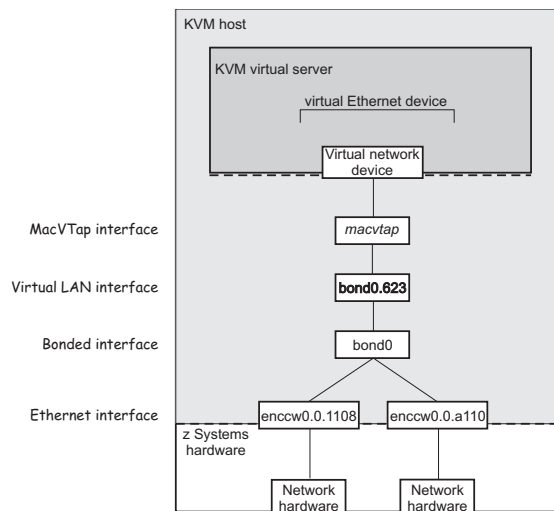


Figure 16. Direct interface type which configures a virtual LAN interface

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 209).

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 215)

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 237)

3. Specify the type.

virtualport type attribute:	openvswitch
-----------------------------	-------------

(see “<virtualport>” on page 245)

4. Specify the model type.

model type attribute:	virtio
-----------------------	--------

(see “<model>” on page 220)

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 230).

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 189). 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 227

Is the root element of a storage pool configuration-XML.

“<volume>” on page 246

Is the root element of a volume configuration-XML.

Part 4. Operation

Chapter 12. Creating, modifying, and deleting persistent virtual server definitions	105
Defining a virtual server	106
Modifying a virtual server definition	106
Undefined a virtual server.	107

Chapter 13. Managing the virtual server life cycle	109
Starting a virtual server	110
Terminating a virtual server	110
Suspending a virtual server.	112
Resuming a virtual server	112

Chapter 14. Monitoring virtual servers	115
Browsing virtual servers.	116
Displaying information about a virtual server	116
Displaying the current libvirt-internal configuration	118

Chapter 15. Migration	121
Definition of a virtual server on different hosts using the same configuration-XML	122
Live virtual server migration	123

Hypervisor release	123
Live migration setup	124
Preservation of the virtual server resources	124
Host environments	127
Phases of a live migration	128
Performing a live migration	129

Chapter 16. Managing system resources	133
Managing virtual CPUs	134
Modifying the number of virtual CPUs.	134
Modifying the virtual CPU weight	137
Managing virtual memory	139

Chapter 17. Managing devices	141
Attaching a device.	142
Detaching a device	143
Replacing a virtual DVD	143
Connecting to the console of a virtual server	145

Chapter 18. Managing storage pools.	147
Storage pool management commands	148
Volume management commands	149

Manage the operation of virtual servers using virsh commands.

Chapter 12. 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 106.
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 106.
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 107.

Related reference:

Chapter 27, “Selected virsh commands,” on page 249

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 257):

```
# 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 116) 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 118.
3. Start the virtual server as described in “Starting a virtual server” on page 110.
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 145.

Related reference:

Chapter 25, “Virtual server life cycle,” on page 177

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 271):

```
# 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 110 and “Starting a virtual server” on page 110).

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 311):

```
# virsh undefine <VS>
```

<VS> Is the name of the virtual server as specified in its domain configuration-XML file.

Chapter 13. 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 278) 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 106.

About this task

- “Starting a virtual server” on page 110
Start a defined virtual server.
- “Terminating a virtual server” on page 110
Properly shut down a virtual server, save a system image, or, if necessary, immediately terminate it.
- “Suspending a virtual server” on page 112
Pause a virtual server.
- “Resuming a virtual server” on page 112
Transfer a paused virtual server to the running state.

Related reference:

Chapter 25, “Virtual server life cycle,” on page 177

Display the state of a defined virtual server including the reason with the virsh **domstate --reason** command.

Chapter 27, “Selected virsh commands,” on page 249

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 25, “Virtual server life cycle,” on page 177 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 308):

```
# 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 25, “Virtual server life cycle,” on page 177 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 305	
To save a system image and terminate a virtual server properly:	“managedsave” on page 280	
To terminate a virtual server immediately:	“destroy” on page 258	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 25, “Virtual server life cycle,” on page 177 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 310):

```
# 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 25, “Virtual server life cycle,” on page 177 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 303):

```
# virsh resume <VS>
```

<VS> Is the name of the virtual server.

Chapter 14. 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 278) 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 106.

About this task

- “Browsing virtual servers” on page 116
View lists of all defined or of all running virtual servers.
- “Displaying information about a virtual server” on page 116
View information about a virtual server, its state, its devices, or scheduling properties.
- “Displaying the current libvirt-internal configuration” on page 118
The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, and is enhanced by libvirt-internal information and the dynamically attached devices.

Related reference:

Chapter 27, “Selected virsh commands,” on page 249

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 278):

```
# 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 266	
Current state	“domstate” on page 268	Display the reason of the current state by using the <code>--reason</code> option.
Scheduling information	“schedinfo” on page 304	
Number of virtual CPUs	“vcpucount” on page 312	
Virtual block devices	“domblkstat” on page 262	To retrieve the device name, use the <code>virsh domblklist</code> command.
Virtual Ethernet interfaces	“domifstat” on page 265	To retrieve the interface name, use the <code>virsh domiflist</code> command.
I/O threads	“iothreadinfo” on page 277	

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-36005076305ffc1ae0000000000023bc

# virsh domblkstat vserv1 /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc rd_req 17866
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc rd_bytes 180311040
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc wr_req 11896
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc wr_bytes 126107648
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc flush_operations 3884
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc rd_total_times 14496884715
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc wr_total_times 9834388979
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023bc 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, and is enhanced by libvirt-internal information and the dynamically attached devices.

Procedure

To display the current libvirt-internal configuration of a defined virtual server, use the virsh `dumpxml` command (see “`dumpxml`” on page 270):

```
# 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/36005076305fffc1ae00000000000020d3"/>
      <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 hotplug device:

dev1.xml

```
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
  <source dev="/dev/mapper/36005076305fffc1ae00000000000021d7"/>
  <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-kvmibm-1.1.2">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-kvm</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iotread="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" iotread="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="sc1p" 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 dynamically attached device.

Chapter 15. 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 122

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 123

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 KVM for IBM z Systems V1.1.0:

```
<type arch="s390x" machine="s390-ccw-kvmibm-1.1.0">hvm</type>
```

Libvirt-internal configuration for QEMU release 2.5.0:

```
<type arch="s390x" machine="s390-ccw-virtio-2.5">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

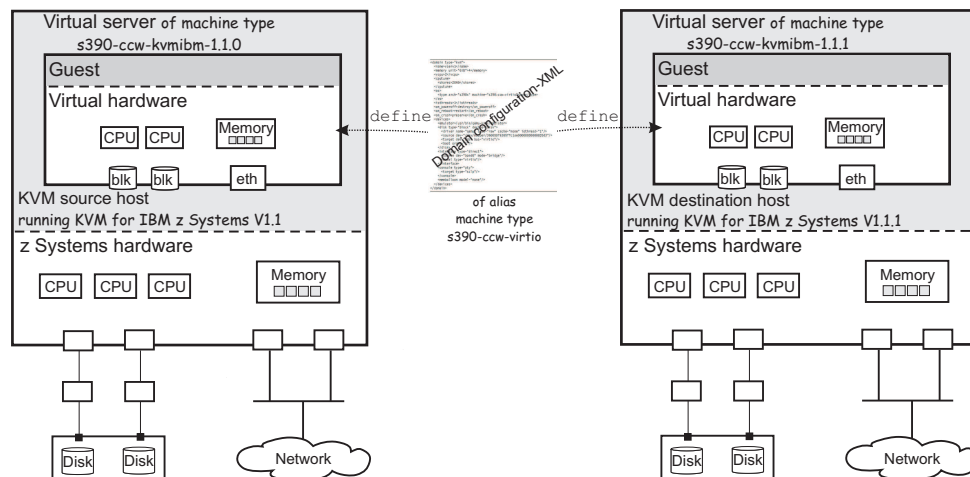


Figure 17. Defining virtual servers on different hosts

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*”.

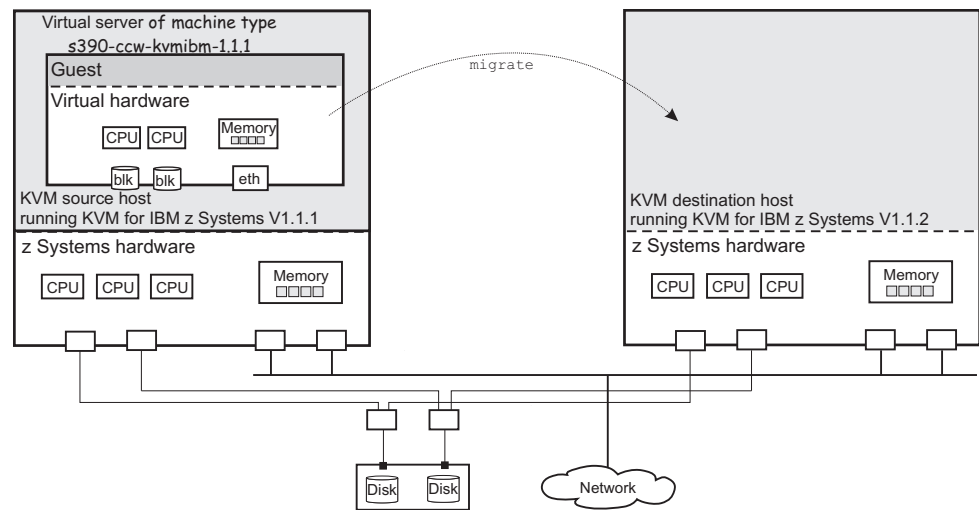
Hypervisor release

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.

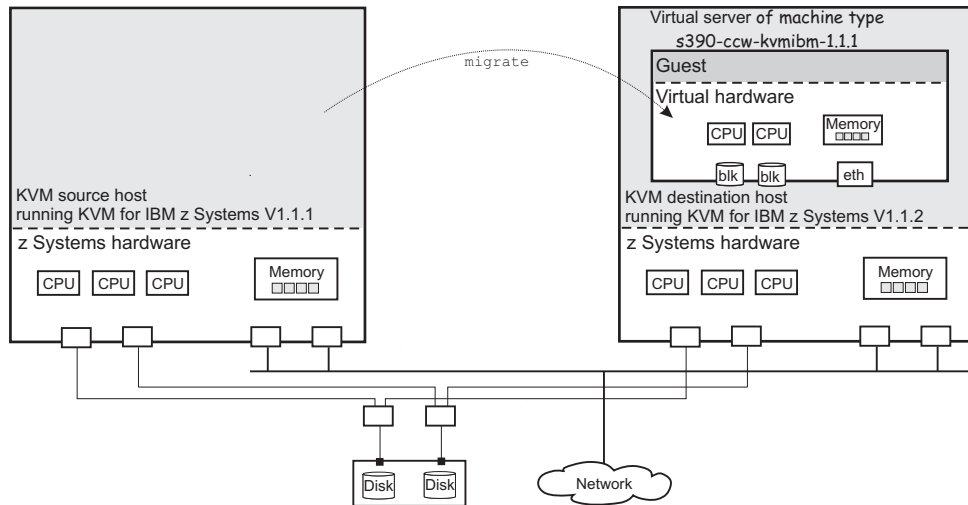
However, if you try to migrate a virtual server to a destination host with an earlier hypervisor release than the currently reflected machine type, you need to explicitly specify this earlier machine type in the virtual server definition before the migration.

Example:

1. Before the migration, the virtual server is running on the source host with hypervisor release KVM for IBM z Systems V1.1.1. The virtual server's machine type is `s390-ccw-kvmibm-1.1.1`.



2. After the migration, the virtual server is running on the destination host with hypervisor release KVM for IBM z Systems V1.1.2. The virtual server's machine type is still `s390-ccw-kvmibm-1.1.1`.



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.

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.

The destination host may also run on another z System, but do not perform a live migration from a z13™ to a zEC12 system. The migration might lead to unexpected results, because the guest might use processor-specific features that are not available on the destination host.

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 as hotplug devices. 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

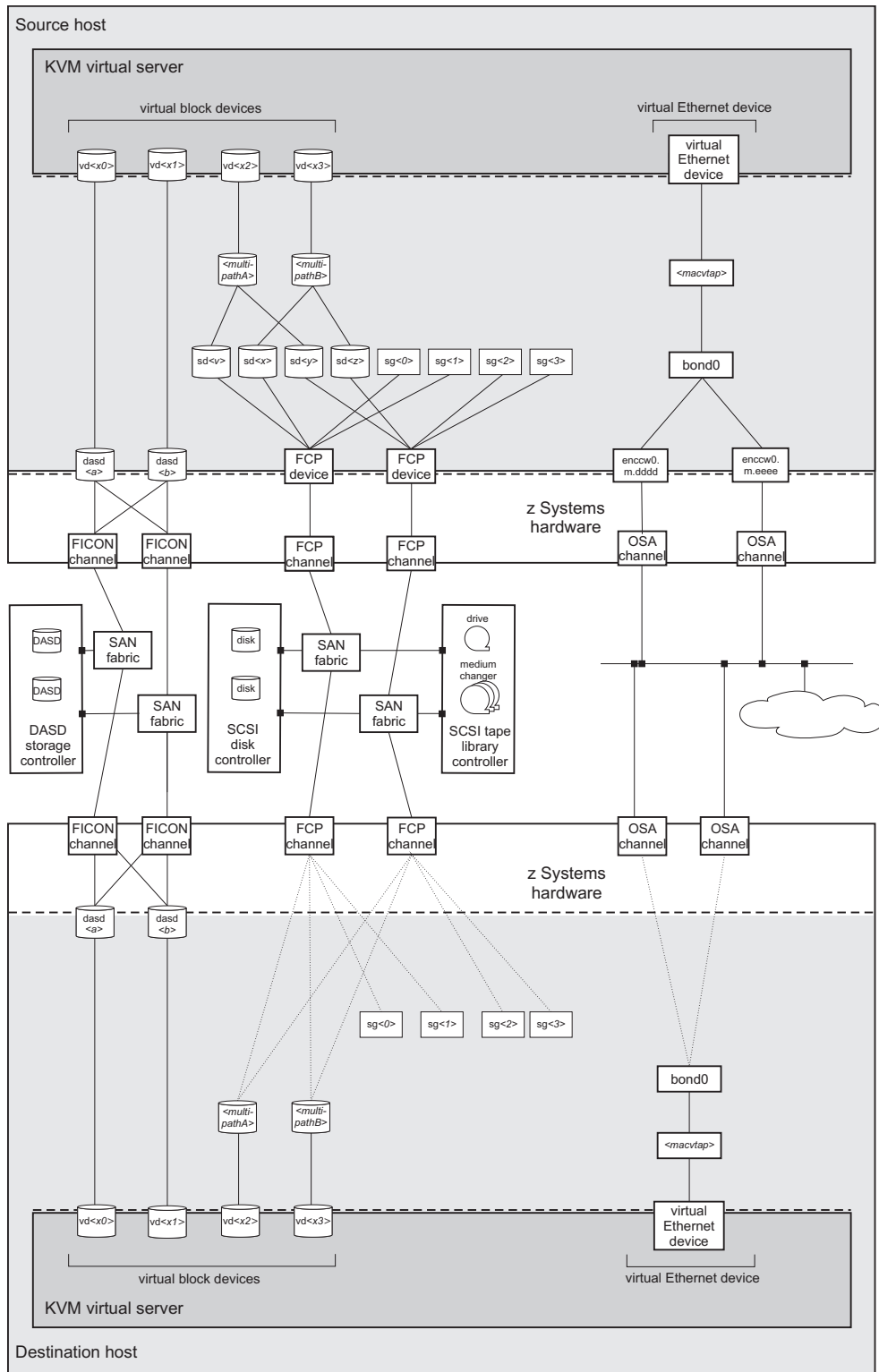


Figure 18. Example of a device setup on the source and destination hosts that allows the migration of the virtual server using these devices

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 287). 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 288):

```
# 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 286):

```
# 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 283):

```
# 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 27, “Selected virsh commands,” on page 249):

```
# 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:

<code>keepalive interval</code>	5 seconds
<code>keepalive count</code>	6

These defaults can be changed in `/etc/libvirt/libvirtd.conf`.

Example:

```
# virsh --keepalive-interval 10 migrate --live --persistent --undefinesource \
--timeout 1200 --verbose vserver1 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 file on 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 16. 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 278) 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 106.

About this task

- “Managing virtual CPUs” on page 134
Modify the portion of the run time that is assigned to the CPUs of a defined virtual server.
- “Managing virtual memory” on page 139
Restrict the amount of physical memory used by a virtual server.

Related reference:

Chapter 27, “Selected virsh commands,” on page 249

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 137 describes how to modify the portion of the run time that is assigned to the virtual server CPUs.

Related concepts:

Chapter 19, “CPU management,” on page 153

Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks:

“Configuring virtual CPUs” on page 60

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 306).

- 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-CPU>

Is the maximum number of available virtual CPUs for the virtual server after the next restart.

<number-of-CPU>

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
```

- 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 304):

```
# 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 61

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 282):

```
# 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 20, “Memory management,” on page 157

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 62

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 17. 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 278) 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 106.

About this task

- “Attaching a device” on page 142
Dynamically attach a hotplug device to a virtual server. If the virtual server is running, the device is hotplugged.
- “Detaching a device” on page 143
Dynamically detach a hotplug device from a virtual server. If the virtual server is running, the device is unplugged.
- “Replacing a virtual DVD” on page 143
Remove the currently provided ISO image, or provide a different one.
- “Connecting to the console of a virtual server” on page 145
Connect to the console of a virtual server.

Related reference:

Chapter 27, “Selected virsh commands,” on page 249

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

Dynamically attach a hotplug device to a virtual server. If the virtual server is running, the device is hotplugged.

Before you begin

- Ensure that the new device is not yet 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 **domblocklist** command to display a list of currently assigned block devices or the virsh **domiflist** command to display a list of currently assigned interface devices.
- Ensure that there is 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 hotplug device:

Add an I/O thread dedicated for the hotplug device by using the virsh **iothreadadd** command (see “iothreadadd” on page 273):

```
# virsh iothreadadd <VS> <IOthread-ID>
```

2. Attach the hotplug device using the virsh **attach-device** command (see “attach-device” on page 252).

This command attaches a hotplug device that remains available for the virtual server until the virtual server is terminated or you detach the device:

```
# virsh attach-device <VS> <device-configuration-XML-filename>
```

This command persistently attaches a hotplug device to a virtual server with the next virtual server restart:

```
# virsh attach-device <VS> <device-configuration-XML-filename> --config
```

<device-configuration-XML-filename>

Is the name of the device 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 which is used in the hotplug device configuration-XML.

<VS> Is the name of the virtual server as defined in the domain configuration-XML file.

Related concepts:

“I/O threads” on page 161

I/O threads are dedicated to perform I/O operations on virtual block devices.

Related tasks:

Chapter 10, “Configuring devices,” on page 73

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 118

The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, and is enhanced by

libvirt-internal information and the dynamically attached devices.

Detaching a device

Dynamically detach a hotplug device from a virtual server. If the virtual server is running, the device is unplugged.

Before you begin

Ensure that the device to be detached was dynamically attached to the virtual server.

To list the devices that are assigned to a virtual server, you can display the current libvirt-internal configuration. To see whether the device was dynamically attached to the virtual server, compare it to your copy of the domain configuration-XML file.

Procedure

1. Detach the device using the virsh **detach-device** command (see “detach-device” on page 259):

```
# virsh detach-device <VS> <device-configuration-XML-filename>
```

2. Optional: If you detach a virtual block device, you might want to remove the I/O thread which is dedicated for the hotplug 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 275):

```
# virsh iothreaddel <VS> <IOthread-ID>
```

<device-configuration-XML-filename>

Is the name of the device configuration-XML file.

<IOthread-ID>

Is the ID of the I/O thread to be deleted from the virtual server.

<VS> Is the name of the virtual server as defined in the domain configuration-XML file.

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 93).

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 254):

```
# 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 256):

```
# 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 308):

```
# 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 110

Use the `virsh start` command to start a shut off virtual server.

“Configuring the console” on page 67

Configure the console by using the console element.

Chapter 18. Managing storage pools

Before you begin

1. Configure one or more storage pools as described in Chapter 11, “Configuring storage pools,” on page 101.
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:

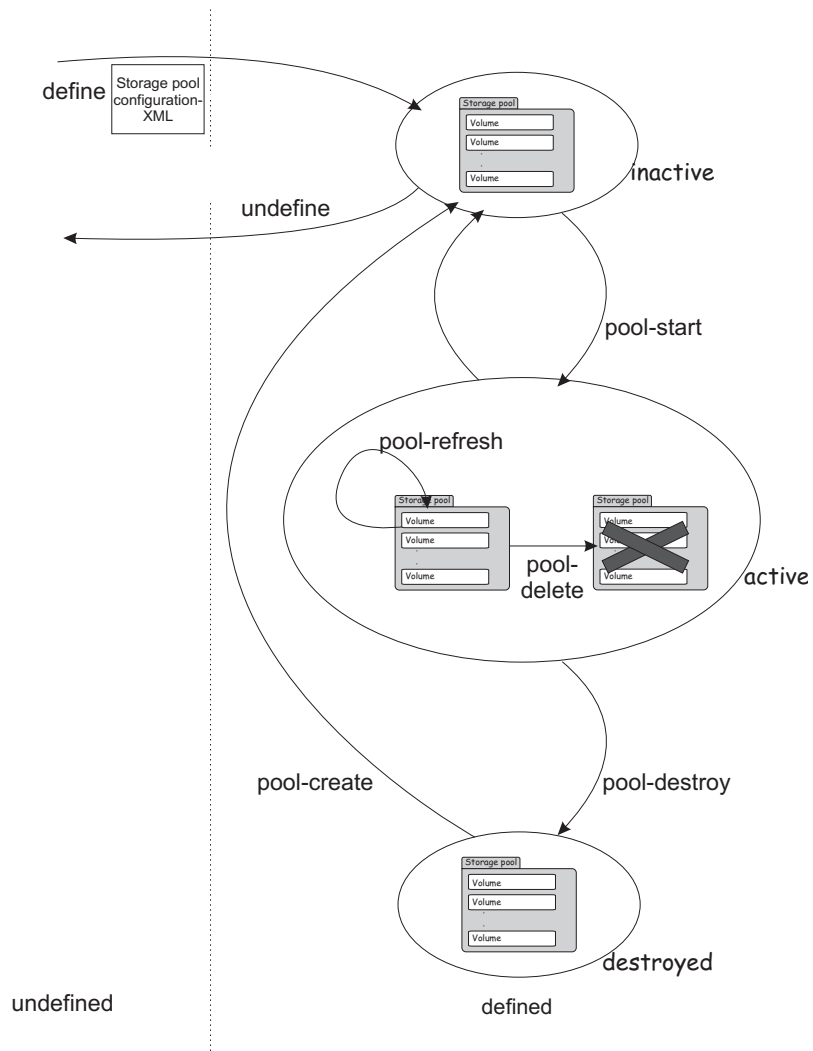


Figure 19. Storage pool state-transition diagram

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 149.

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 290
Edit the libvirt-internal configuration of a defined storage pool	“pool-edit” on page 294
Delete the persistent libvirt definition of a storage pool	“pool-undefine” on page 300

- 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 289
Start a defined inactive storage pool	“pool-start” on page 299
Update the volume list of a storage pool	“pool-refresh” on page 298
Shut down a storage pool. The pool can be recreated by using the virsh pool-create command	“pool-destroy” on page 292
Delete the volumes of a storage pool	“pool-delete” on page 291

- Monitoring storage pools:

Functionality	Command
View a list of all defined storage pools	“pool-list” on page 296
Display the current libvirt-internal configuration of a storage pool	“pool-dumpxml” on page 293
Display information about a defined storage pool	“pool-info” on page 295
Retrieve the name of a storage pool from its UUID	“pool-name” on page 297
Retrieve the UUID of a storage pool from its name	“pool-uuid” on page 301

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 313
Remove a volume from a storage pool	"vol-delete" on page 314

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 318
Display the current libvirt-internal configuration of a storage volume	"vol-dumpxml" on page 315
Display information about a defined volume	"vol-info" on page 316
Display the key of a volume from its name or path	"vol-key" on page 317
Display the name of a volume from its key or path	"vol-name" on page 319
Display the path of a volume from its name or key	"vol-path" on page 320
Display the storage pool name or UUID which hosts the volume	"vol-pool" on page 321

Part 5. Best practices and performance considerations

Chapter 19. CPU management	153	Chapter 21. Storage management.	161
Linux scheduling	153	I/O threads	161
CPU weight	154	Logical volume management	161

Chapter 20. Memory management.	157
---	-----

Avoid common pitfalls and tune the virtual server.

Chapter 19. CPU management

Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks:

“Configuring virtual CPUs” on page 60

Configure virtual CPUs for a virtual server.

“Managing virtual CPUs” on page 134

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.

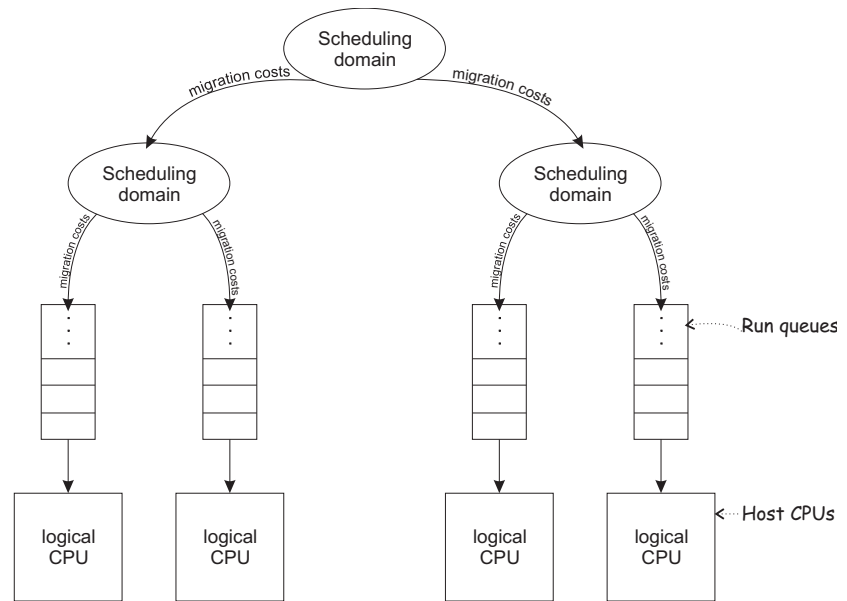


Figure 20. Linux scheduling

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.

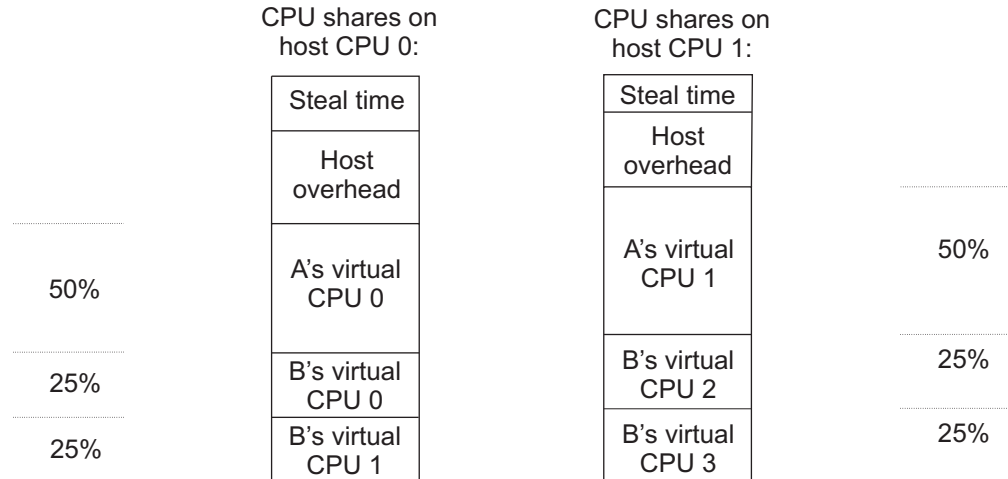
Example:

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 20. 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*.

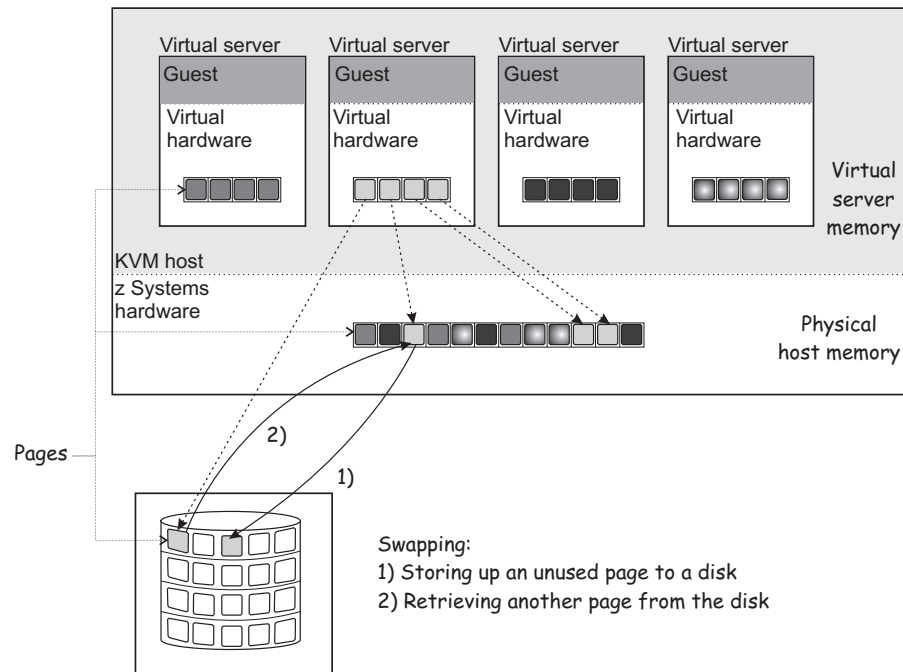


Figure 21. Swapping

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 z Systems 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.

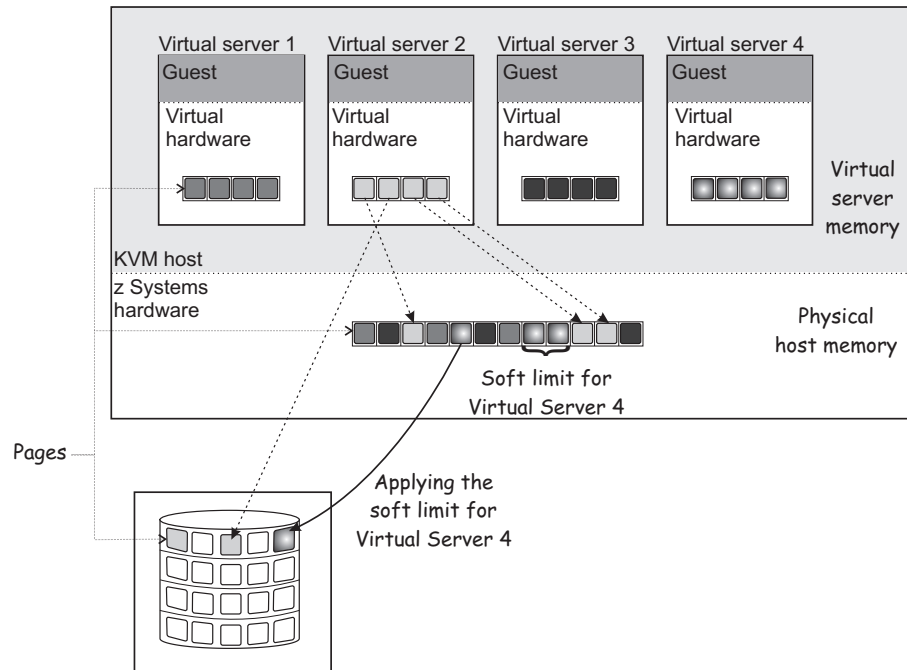


Figure 22. Applying the soft limit

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 62

Configure the virtual server memory.

“Managing virtual memory” on page 139

Specify a soft limit for the amount of physical host memory used by a virtual server.

Chapter 21. 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.

They are used for persistent devices as well as for hotplug 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 persistent devices” on page 66
- “Configuring a DASD or SCSI disk” on page 76
- “Configuring a virtual HBA” on page 86

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 142
- “Detaching a device” on page 143

Logical volume management

Consider these aspects when the virtual server utilizes logical volumes.

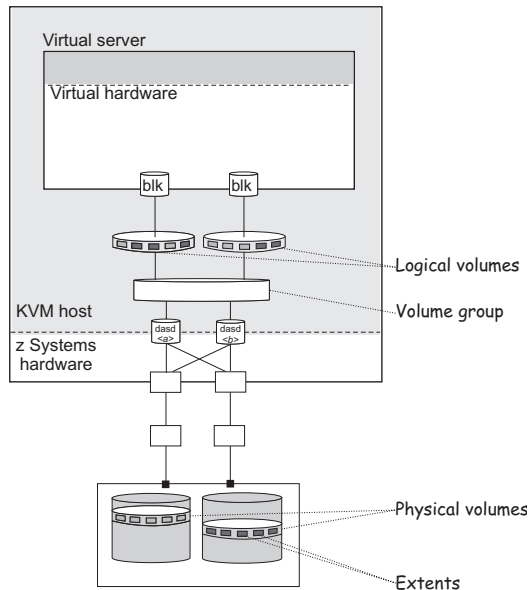
Path redundancy

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.

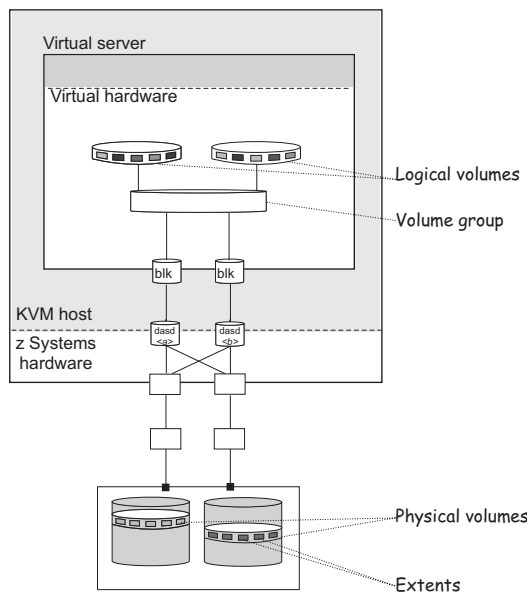
Data integrity

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-0x123412341234:\
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.

Related publications

KVM for IBM z Systems: System Administration, SC27-8237

Part 6. Diagnostics and troubleshooting

Chapter 22. Logging	167	Creating a virtual server dump on the host	169
Log messages	167	Creating a dump on the virtual server	169
Specifying the logging level of the libvirt log messages	167	Chapter 24. Collecting performance metrics	171
Chapter 23. Dumping	169		

Monitor and display information that helps to diagnose and solve problems.

Chapter 22. Logging

Adapt the logging facility to your needs.

Log messages

These logs are created.

libvirt log messages

You define where libvirt log messages are stored. By default they will be stored in the system journal.

`/var/log/libvirt/qemu/<VS>.log`

QEMU log file of the specified virtual server.

<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 23. 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 24. 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

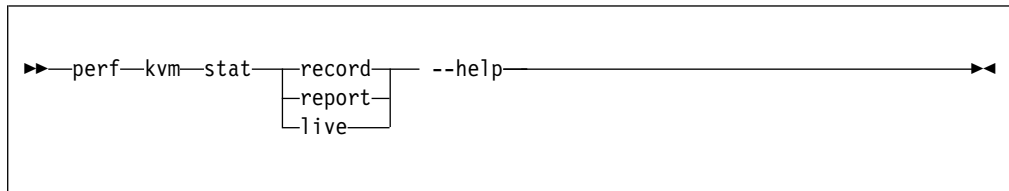
Analyze events for all VMs, all VCPUs:

          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 SSEK      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:



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 173 lists all diagnoses (DIAG) as supported by KVM on z Systems.

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 z Systems. A list of all Linux diagnoses is provided in *Device Drivers, Features, and Commands*, SC33-8411.

Part 7. Reference

Chapter 25. Virtual server life cycle	177	<source> as child element of <hostdev>	236
shut off	178	<source> as child element of <interface>	237
running	179	<source> as child element of <pool>	238
paused	180	<target> as child element of <console>	239
crashed	181	<target> as child element of <disk>	240
in shutdown.	182	<target> as child element of <pool>	241
		<target> as child element of <volume>	242
		<type>	243
Chapter 26. Selected libvirt XML elements.	183	<vcpu>	244
<adapter> as child element of <source>	185	<virtualport>	245
<address> as child element of <controller>, <disk>, <interface>, and <memballoon>	186	<volume>	246
<address> as child element of <hostdev> or <disk>	187	<watchdog>	247
<address> as child element of <source>	188		
<backend>	189	Chapter 27. Selected virsh commands	249
<boot>	190	attach-device	252
<cipher>	191	change-media	254
<cmdline>	192	console	256
<console>	193	define	257
<controller>	194	destroy	258
<cputune>	195	detach-device	259
<device>	196	dombklist	261
<devices>	197	dombkstat	262
<disk>	198	domiflist	264
<domain>	199	domifstat	265
<driver> as child element of <controller>	200	dominfo	266
<driver> as child element of <disk>	201	domjobabort	267
<emulator>	203	domstate	268
<format>	204	dump	269
<geometry>	205	dumpxml	270
<host>	206	edit	271
<hostdev>	207	inject-nmi	272
<initrd>	208	iothreadadd	273
<interface>	209	iothreaddel	275
<iothreads>	210	iothreadinfo	277
<kernel>	211	list	278
<key>	212	managedsave	280
<keywrap>	213	memtune	282
<log>	214	migrate	283
<mac>	215	migrate-getspeed	286
<memballoon>	216	migrate-setmaxdowntime	287
<memory>	217	migrate-setspeed	288
<memtune>	219	pool-autostart	289
<model>	220	pool-define	290
<name>	221	pool-delete	291
<on_crash>	222	pool-destroy	292
<on_reboot>	223	pool-dumpxml	293
<os>	224	pool-edit	294
<path> as child element of <pool><target>	225	pool-info	295
<path> as child element of <volume><target>	226	pool-list	296
<pool>	227	pool-name	297
<readonly>	229	pool-refresh	298
<rng>	230	pool-start	299
<shareable>	231	pool-undefine	300
<shares>	232	pool-uuid	301
<soft_limit>	233	reboot	302
<source> as child element of <disk>	234	resume	303

schedinfo	304		vol-list	318
shutdown	305		vol-name	319
setvcpus	306		vol-path	320
start	308		vol-pool	321
suspend	310			
undefine	311			
vcpucount	312			
vol-create	313			
vol-delete	314			
vol-dumpxml	315			
vol-info	316			
vol-key	317			
			Chapter 28. Selected QEMU commands.	323
			QEMU monitor commands	323
			Examples for the use of the qemu-img command	323
			Chapter 29. Hypervisor information for the	
			virtual server user	325

Get an overview of the virtual server states and the elements and commands that are specific to configure and operate a virtual server on z Systems. The virtual server user can retrieve information about the z Systems hardware and the LPAR on which the KVM host runs.

Chapter 25. Virtual server life cycle

Display the state of a defined virtual server including the reason with the virsh `domstate --reason` command.

Figure 23 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.

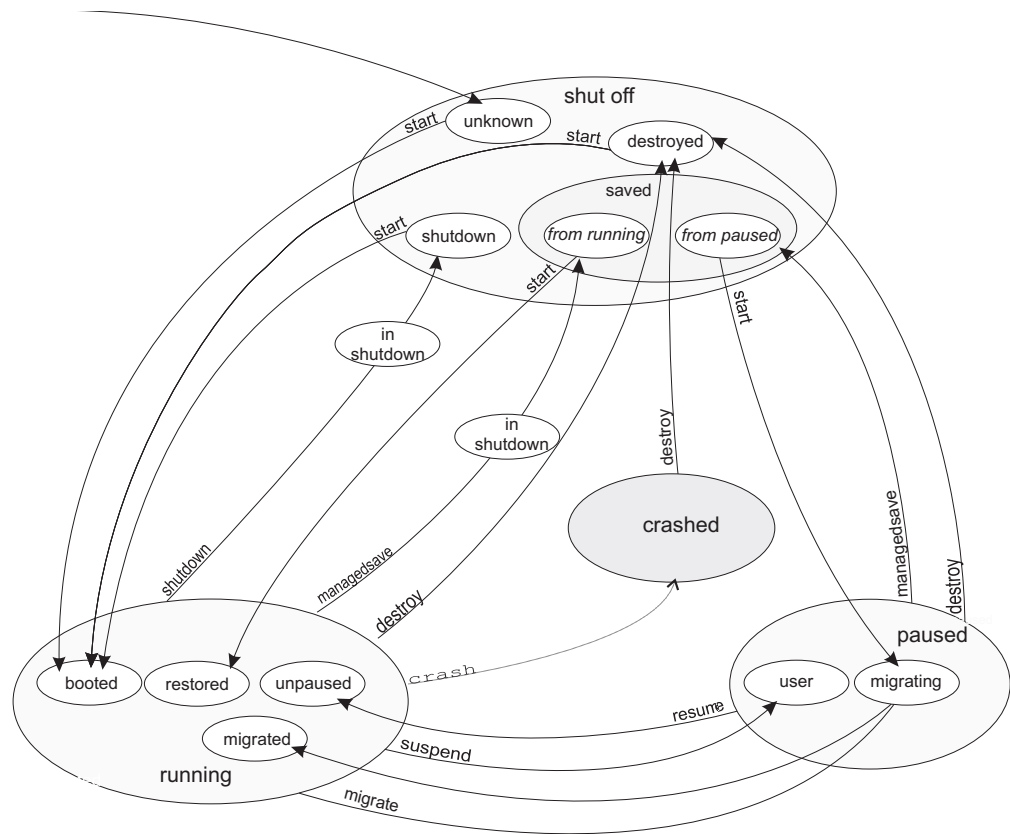


Figure 23. State-transition diagram of a virtual server including reasons

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 <code>/var/lib/libvirt/qemu/save/<VS>.save</code> 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 <i>from running</i>)	running (restored)
start	shut off (saved <i>from paused</i>)	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 <i>from running</i>)	running (booted)
start --force-boot	shut off (saved <i>from paused</i>)	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 <i>from running</i>)	paused (migrating)
start --paused	shut off (saved <i>from paused</i>)	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 <i>from running</i>)
managedsave --running	n/a	shut off (saved <i>from running</i>)
managedsave --paused	n/a	shut off (saved <i>from paused</i>)
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 <i>from paused</i>)
managedsave --running	n/a	shut off (saved <i>from running</i>)
managedsave --paused	n/a	shut off (saved <i>from paused</i>)
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

in shutdown

While the virtual server is shutting down, it traverses the “in shutdown” state.

Chapter 26. 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 185
- “<address> as child element of <controller>, <disk>, <interface>, and <memballoon>” on page 186
- “<address> as child element of <hostdev> or <disk>” on page 187
- “<address> as child element of <source>” on page 188
- “<backend>” on page 189
- “<boot>” on page 190
- “<cipher>” on page 191
- “<cmdline>” on page 192
- “<console>” on page 193
- “<controller>” on page 194
- “<cputune>” on page 195
- “<device>” on page 196
- “<devices>” on page 197
- “<disk>” on page 198
- “<domain>” on page 199
- “<driver> as child element of <controller>” on page 200
- “<driver> as child element of <disk>” on page 201
- “<emulator>” on page 203
- “<format>” on page 204
- “<geometry>” on page 205
- “<host>” on page 206
- “<hostdev>” on page 207
- “<initrd>” on page 208
- “<interface>” on page 209
- “<iothreads>” on page 210
- “<kernel>” on page 211
- “<key>” on page 212
- “<keywrap>” on page 213
- “<log>” on page 214
- “<mac>” on page 215
- “<memballoon>” on page 216
- “<memory>” on page 217
- “<memtune>” on page 219
- “<model>” on page 220
- “<name>” on page 221
- “<on_crash>” on page 222
- “<on_reboot>” on page 223
- “<os>” on page 224

- “<path> as child element of <pool><target>” on page 225
- “<pool>” on page 227
- “<readonly>” on page 229
- “<rng>” on page 230
- “<shareable>” on page 231
- “<shares>” on page 232
- “<soft_limit>” on page 233
- “<source> as child element of <disk>” on page 234
- “<source> as child element of <hostdev>” on page 236
- “<source> as child element of <interface>” on page 237
- “<source> as child element of <pool>” on page 238
- “<target> as child element of <console>” on page 239
- “<target> as child element of <disk>” on page 240
- “<target> as child element of <pool>” on page 241
- “<target> as child element of <volume>” on page 242
- “<type>” on page 243
- “<vcpu>” on page 244
- “<virtualport>” on page 245
- “<volume>” on page 246
- “<watchdog>” on page 247

<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 88

Parent elements

“<source> as child element of <hostdev>” on page 236.

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>

<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 76
- "Configuring an image file as storage device" on page 82

Parent elements

- "<controller>" on page 194
- "<disk>" on page 198
- "<interface>" on page 209
- "<memballoon>" on page 216

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 88
- “Configuring a virtual SCSI-attached CD/DVD drive” on page 93

Parent elements

- “<hostdev>” on page 207
- “<disk>” on page 198

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>

<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 88

Parent elements

“<source> as child element of <hostdev>” on page 236

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 100

Parent elements

“<rng>” on page 230

Child elements

None.

Example

```
<devices>
  ...
  <rng model="virtio">
    <backend model="random"/>/dev/random</backend>
  </rng>
  ...
</devices>
```

<boot>

<boot>

Specifies 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.

Usage

“Configuring the boot process” on page 53

Parent elements

“<disk>” on page 198

Child elements

None.

Example

```
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iotthread="1"/>
  <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
  <target dev="vdb" bus="virtio"/>
  <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xa30e"/>
  <boot order="1"/>
</disk>
```

<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 69

Parent elements

“<keywrap>” on page 213

Child elements

None.

Example

```
<domain type="kvm">
  ...
  <keywrap>
    <cipher name="aes" state="off"/>
  </keywrap>
  ...
</domain>
```

<cmdline>

<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 224

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 67

Parent elements

“<devices>” on page 197

Child elements

- “<log>” on page 214
- <protocol>
- “<target> as child element of <console>” on page 239

Example

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

<controller>

<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 88

Parent elements

“<devices>” on page 197

Child elements

- “<address> as child element of <controller>, <disk>, <interface>, and <memballoon>” on page 186
- “<driver> as child element of <controller>” on page 200

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>
```

<cputune>

Groups CPU tuning parameters.

Text content

None.

Selected attributes

None.

Usage

“Tuning virtual CPUs” on page 61

Parent elements

“<domain>” on page 199

Child elements

“<shares>” on page 232

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>

<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 101

Parent elements

“<source> as child element of <pool>” on page 238

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 73

Parent elements

“<domain>” on page 199

Child elements

- “<console>” on page 193
- “<controller>” on page 194
- “<disk>” on page 198
- “<emulator>” on page 203
- “<hostdev>” on page 207
- “<interface>” on page 209
- “<memballoon>” on page 216
- “<watchdog>” on page 247

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>
```

<disk>

<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 73
- “Configuring a virtual SCSI-attached CD/DVD drive” on page 93

Parent elements

“<devices>” on page 197

Child elements

- “<address> as child element of <controller>, <disk>, <interface>, and <memballoon>” on page 186
- <blockio>
- “<boot>” on page 190
- “<driver> as child element of <disk>” on page 201
- “<geometry>” on page 205
- “<readonly>” on page 229
- “<shareable>” on page 231
- “<source> as child element of <disk>” on page 234
- “<target> as child element of <disk>” on page 240

Example

```
<disk type="block" device="disk">  
  <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>  
  <source dev="/dev/mapper/36005076305fffc1ae0000000000021d5"/>  
  <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

None.

Attributes

type=kvm

Specifies the virtual server type.

Usage

“Domain configuration-XML” on page 51

Parent elements

None.

Child elements

- <clock>
- “<console>” on page 193
- “<controller>” on page 194
- “<cputune>” on page 195
- <currentMemory>
- “<devices>” on page 197
- “<iothreads>” on page 210
- <memory>
- <name>
- “<on_crash>” on page 222
- <on_poweroff>
- <on_reboot>
- <os>
- <uuid>
- “<vcpu>” on page 244

<driver> as child element of <controller>

<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 86

Parent elements

“<controller>” on page 194

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!

<driver> as child element of <disk>

Usage

- “Configuring a DASD or SCSI disk” on page 76
- “Configuring a virtual SCSI-attached CD/DVD drive” on page 93

Parent elements

“<disk>” on page 198

Child elements

None.

Example

```
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iotthread="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 65
- “Displaying the current libvirt-internal configuration” on page 118

Parent elements

“<devices>” on page 197

Child elements

None.

Example

```
<emulator>/usr/bin/qemu-kvm</emulator>
```

<format>

<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 101

Parent elements

“<target> as child element of <volume>” on page 242

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>
```

<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 76

Parent elements

“<disk>” on page 198

Child elements

None.

Example

```
<geometry cyls="16383" heads="16" secs="64" trans="lba"/>
```

<host>

<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 101

Parent elements

“<source> as child element of <pool>” on page 238

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 88

Parent elements

“<devices>” on page 197

Child elements

- “<address> as child element of <hostdev> or <disk>” on page 187
- “<readonly>” on page 229
- “<shareable>” on page 231
- “<source> as child element of <hostdev>” on page 236

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>

<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 224

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

Specifies the type of connection:

direct Creates a MacVTap interface.

bridge Attaches to a bridge, as for example implemented by a virtual switch.

trustGuestRxFilters = no | 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 96

Parent elements

"<devices>" on page 197

Child elements

- "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 186
- "<mac>" on page 215
- "<model>" on page 220
- "<source> as child element of <interface>" on page 237
- "<virtualport>" on page 245

Example

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

<iothreads>

<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 persistent devices” on page 66

Parent elements

“<domain>” on page 199

Child elements

None.

Example

```
<iothreads>3</iothreads>
```


<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 224

Child elements

None.

Example

```
<kernel>/boot/vmlinuz-3.9.3-60.x.20130605-s390xrhel</kernel>
```

<key>

<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 101

Parent elements

“<volume>” on page 246

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 69

Parent elements

“<domain>” on page 199

Child elements

“<cipher>” on page 191

Example

```
<domain type="kvm">
  ...
  <keywrap>
    <cipher name="aes" state="off"/>
  </keywrap>
  ...
</domain>
```

<log>

<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 67

Parent elements

"<console>" on page 193

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 96

Parent elements

“<interface>” on page 209

Child elements

None.

Example

```
<interface type='direct'>
  <mac address='02:10:10:f9:80:00' />
  <model type='virtio' />
</interface>
```

<memballoon>

<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 71

Parent elements

“<devices>” on page 197

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 62
- “Configuring the collection of QEMU core dumps” on page 64

Parent elements

“<domain>” on page 199

Child elements

None.

Example

This example:

- Configures 524,288 KB of virtual memory.

<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 20, “Memory management,” on page 157
 - “Tuning virtual memory” on page 62
- |

| **Parent elements**

| “<domain>” on page 199

| **Child elements**

| “<soft_limit>” on page 233

| **Example**

| This example ...

```
| <domain>
|   ...
|   <memtune>
|     <soft_limit unit="M">128</soft_limit>
|   </memtune>
| </domain>
```

`<model>`

`<model>`

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 96
- “Configuring a virtual switch” on page 98

Parent elements

“`<interface>`” on page 209

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>

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 199

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
```

<on_crash>

<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 199

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 302.

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 199

Child elements

None.

Example

```
<on_reboot>restart</on_reboot>
```

<os>

<OS>

Groups the operating system parameters.

Text content

None.

Selected attributes

None.

Usage

"Domain configuration-XML" on page 51

Parent elements

"<domain>" on page 199

Child elements

- "<type>" on page 243
- "<kernel>" on page 211
- "<initrd>" on page 208
- "<cmdline>" on page 192

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 CMM=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 101

Parent elements

“<target> as child element of <pool>” on page 241

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/fspool1"/>
  </source>
  <target>
    <path>/var/lib/libvirt/images/fspool01</path>
  </target>
</pool>
```

<path> as child element of <volume><target>

<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 101

Parent elements

“<target> as child element of <volume>” on page 242

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 101

Parent elements

None.

Child elements

- <name>
- "<source> as child element of <pool>" on page 238
- "<target> as child element of <pool>" on page 241

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>
```

<pool>

```
    <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 93

Parent elements

- “<disk>” on page 198
- “<hostdev>” on page 207

Child elements

None.

Example

```
<disk type="block" device="disk">  
  <driver name="qemu" type="raw" cache="none" io="native" iotthread="1"/>  
  <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>  
  <target dev="vdb" bus="virtio"/>  
  <readonly/>  
</disk>
```

<rng>

<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 100

Parent elements

“<devices>” on page 197

Child elements

“<backend>” on page 189

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 198
- “<hostdev>” 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"/>
    <shareable/>
  </hostdev>
</devices>
```

<shares>

<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 61
- “CPU weight” on page 154

Parent elements

“<cputune>” on page 195

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 20, “Memory management,” on page 157
- “Configuring virtual memory” on page 62

Parent elements

“<memtune>” on page 219

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 76
- "Configuring an image file as storage device" on page 82
- "Configuring a volume as storage device" on page 84
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 93

Parent elements

"<disk>" on page 198

See also:

- "<source> as child element of <interface>" on page 237

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-vo10"/>  
  <target dev="vdb" bus="virtio"/>  
  <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0009"/>  
</disk>
```

|
|
|
|
|
|
|

<source> as child element of <hostdev>

<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 88

Parent elements

“<hostdev>” on page 207

Child elements

- “<address> as child element of <source>” on page 188
- “<adapter> as child element of <source>” on page 185

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; 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.

Usage

“Configuring virtual Ethernet devices” on page 96

Parent elements

“<interface>” on page 209

Child elements

None.

Example

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

<source> as child element of <pool>

<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 101

Parent elements

“<pool>” on page 227

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 67

Parent elements

“<console>” on page 193

See also:

- “<target> as child element of <disk>” on page 240

Child elements

None.

Example

```
<console type="pty">  
  <target type="sc1p"/>  
</console>
```

<target> as child element of <disk>

<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 76
- “Configuring an image file as storage device” on page 82
- “Configuring a virtual SCSI-attached CD/DVD drive” on page 93

Parent elements

“<disk>” on page 198

See also: “<target> as child element of <console>” on page 239

Child elements

None.

Example

```
<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iotthread="1"/>
  <source dev="/dev/mapper/36005076305fffc1ae00000000000021d7"/>
  <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 101

Parent elements

“<pool>” on page 227

Child elements

- “<path> as child element of <pool><target>” on page 225
- <permissions>

Example

```
<pool type="dir">
  <name>directoryPool</name>
  <target>
    <path>/var/lib/libvirt/images</path>
  </target>
</pool>
```

<target> as child element of <volume>

| <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 101

| **Parent elements**

| “<volume>” on page 246

| **Child elements**

- | • “<path> as child element of <volume><target>” on page 226
- | • “<format>” on page 204

| **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 122

Parent elements

“<os>” on page 224

Child elements

None.

Example

```
<type arch="s390x" machine="s390-ccw-virtio">hvm</type>
```

<vcpu>

<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 60

Parent elements

“<domain>” on page 199

Child elements

None.

Example

```
<domain type="kvm">  
  <name>vserv1</name>  
  <memory>524288</memory>  
  <vcpu current="2">5</vcpu>  
  . . . .  
</domain>
```

<virtualport>

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 98

Parent elements

“<interface>” on page 209

Child elements

None.

Example

```
<interface>
  ...
  <virtualport type="openvswitch">
</interface>
```

<volume>

| <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 101

| **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 68

Parent elements

“<devices>” on page 197

Child elements

None.

<watchdog>

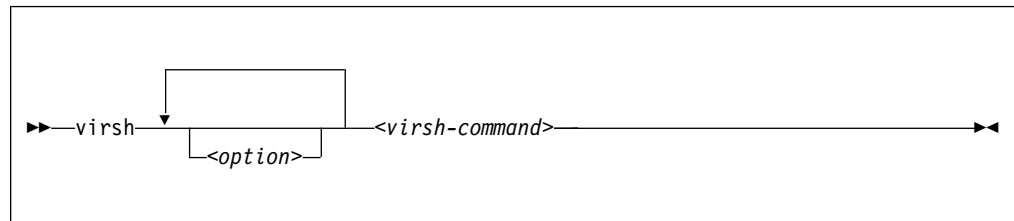
Example

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

Chapter 27. 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 252
- “change-media” on page 254

- “console” on page 256
- “define” on page 257
- “destroy” on page 258
- “detach-device” on page 259
- “domblklist” on page 261
- “domblkstat” on page 262
- “domiflist” on page 264
- “domifstat” on page 265
- “dominfo” on page 266
- “domjobabort” on page 267
- “domstate” on page 268
- “dump” on page 269
- “dumpxml” on page 270
- “edit” on page 271
- “inject-nmi” on page 272
- “iothreadadd” on page 273
- “iothreaddel” on page 275
- “iothreadinfo” on page 277
- “list” on page 278
- “managedsave” on page 280
- “memtune” on page 282
- “migrate” on page 283
- “migrate-getspeed” on page 286
- “migrate-setmaxdowntime” on page 287
- “migrate-setspeed” on page 288
- “pool-autostart” on page 289
- “pool-define” on page 290
- “pool-delete” on page 291
- “pool-destroy” on page 292
- “pool-dumpxml” on page 293
- “pool-edit” on page 294
- “pool-info” on page 295
- “pool-list” on page 296
- “pool-name” on page 297
- “pool-refresh” on page 298
- “pool-start” on page 299
- “pool-undefine” on page 300
- “pool-uuid” on page 301
- “reboot” on page 302
- “resume” on page 303
- “schedinfo” on page 304
- “shutdown” on page 305
- “setvcpus” on page 306
- “start” on page 308
- “suspend” on page 310

- “undefine” on page 311
- “vcpucount” on page 312
- “vol-create” on page 313
- “vol-delete” on page 314
- “vol-dumpxml” on page 315
- “vol-info” on page 316
- “vol-key” on page 317
- “vol-list” on page 318
- “vol-name” on page 319
- “vol-path” on page 320
- “vol-pool” on page 321

Example

This example displays the virsh online help of the virsh **migrate** command:

```
# virsh help migrate
```

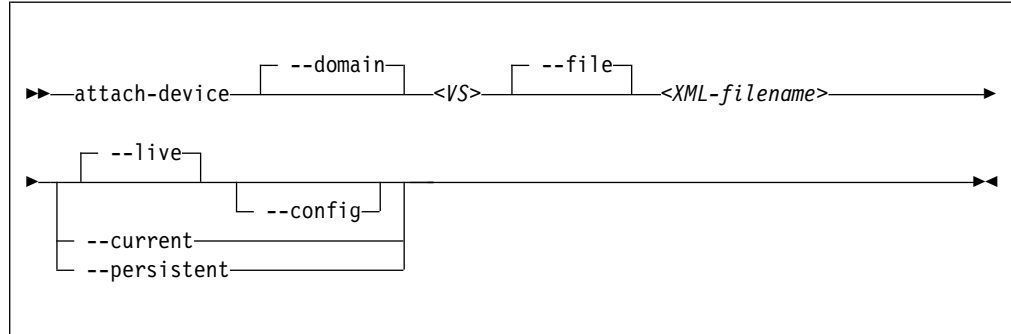
This example increases the keepalive interval of the connection to the host to 10 seconds during a live migration:

```
# 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



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 142

Example

This example attaches the devices that are defined in device configuration-XML file dev1.xml to the virtual server vserv1.

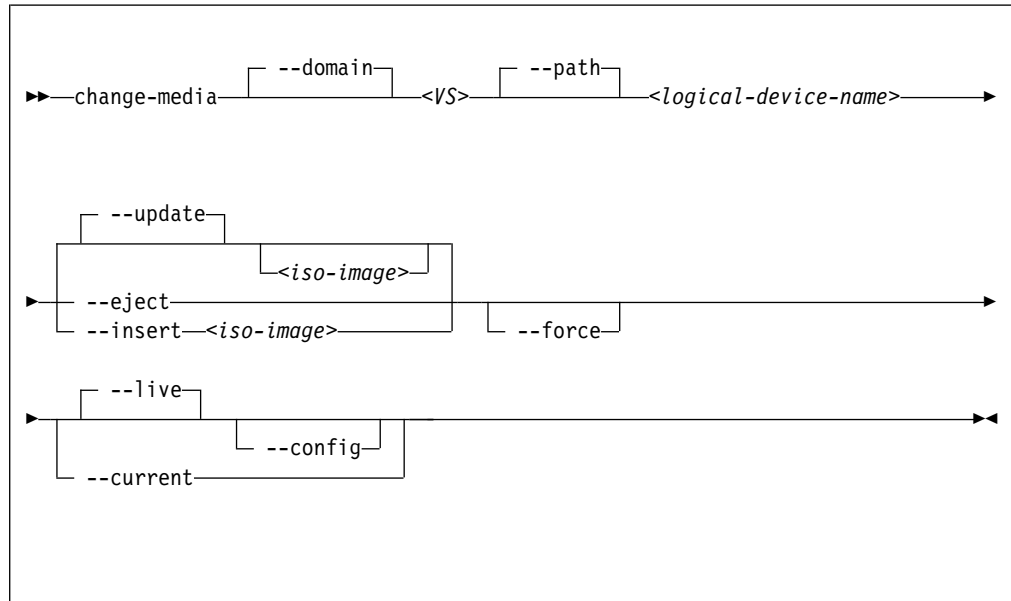
```
# virsh attach-device vserv1 dev1.xml
```

See also the example on page 119.

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 143

Example

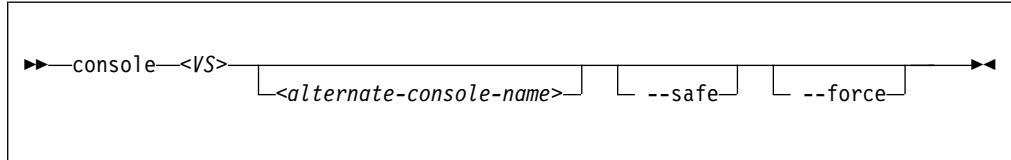
This command replaces the currently provided virtual DVD by a different one:

```
# 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 145

Example

This example connects to the console of virtual server vserv1.

```
# virsh console vserv1
```

define

Creates a persistent virtual server definition.

Syntax

```
▶▶ define <XML-filename> [---validate] ▶▶
```

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 106

Example

This example defines the virtual server, which is configured in domain configuration-XML file `vserv1.xml`.

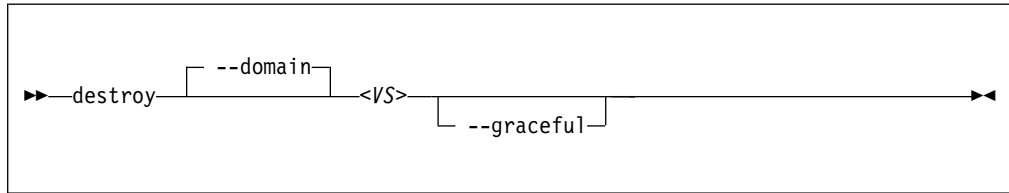
```
# virsh define vserv1.xml
```

destroy

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 110

Example

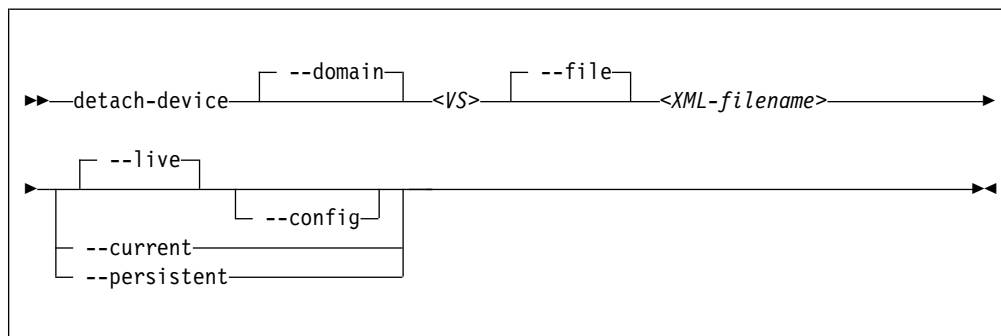
This example immediately terminates virtual server vserv1.

```
# virsh destroy vserv1
```


detach-device

Detaches a device from a defined virtual server.

Syntax



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.

detach-device

Usage

“Detaching a device” on page 143

Example

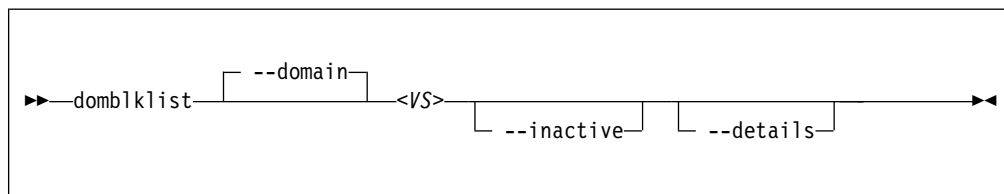
This example detaches the device that is defined in device configuration-XML file `vda.xml` from virtual server `vserv1`.

```
# virsh detach-device vserv1 vda.xml
```

dombklist

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 116

Example

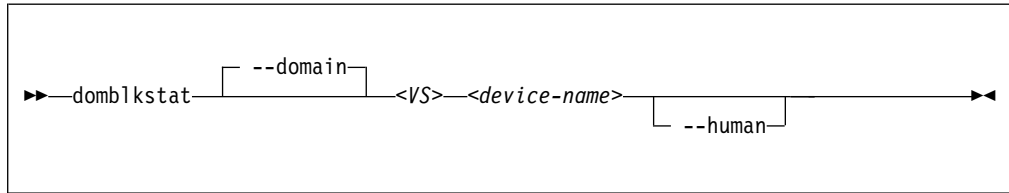
```

# virsh dombklist vserv1
Target    Source
-----
vda       /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023be
  
```

dombkstat

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 116

Example

Obtain the device names of the block devices of virtual server vserv1:

```

# virsh dombklist vserv1
Target      Source
-----
vda         /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae0000000000023be
  
```

Obtain information about the virtual block device vda:

```

# virsh dombkstat 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
  
```

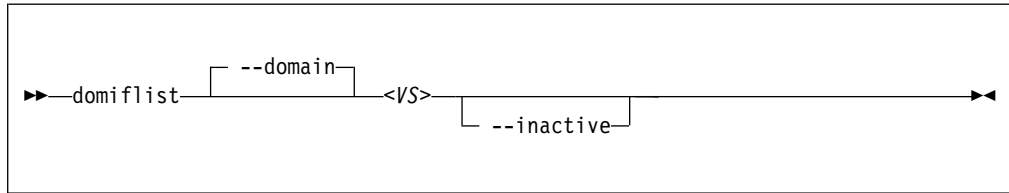
Alternatively, display written-out information:

```
# 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
```

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 116

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

```
▶▶—domifstat [ --domain ] <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 116

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 116

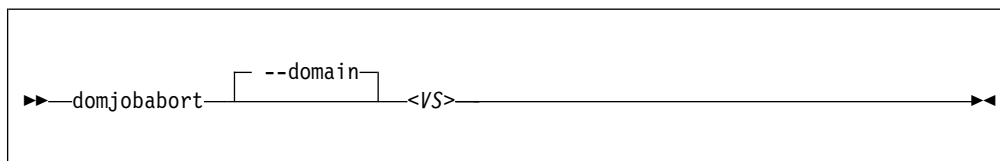
Example

```
# 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



Where:

<VS> Is the name, ID or UUID of the virtual server.

Selected options

None.

Usage

“Live virtual server migration” on page 123

Example

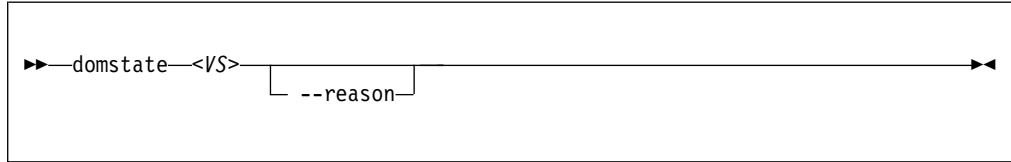
This example aborts the currently running dump request for vserv1.

```
# 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



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 116

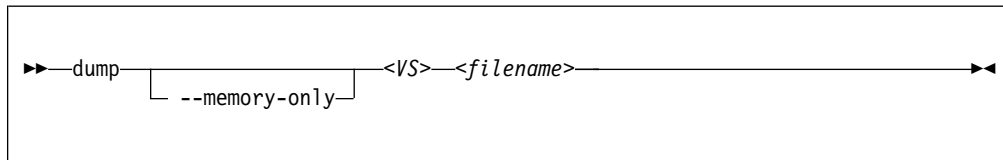
Example

```
# virsh domstate vserv1  
crashed  
# virsh domstate vserv1 --reason  
crashed (panicked)
```

dump

Creates a virtual server dump on the host.

Syntax



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 169

Example

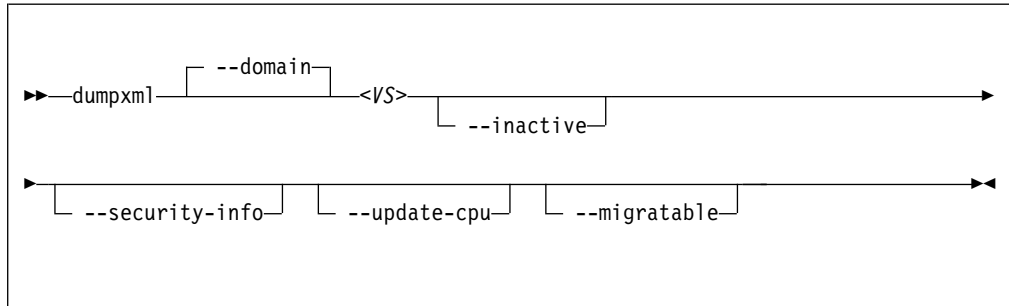
This example dumps the virtual server `vserv1` to the file `dumpfile.name`.

```
# 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 118

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



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 106

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 169

Example

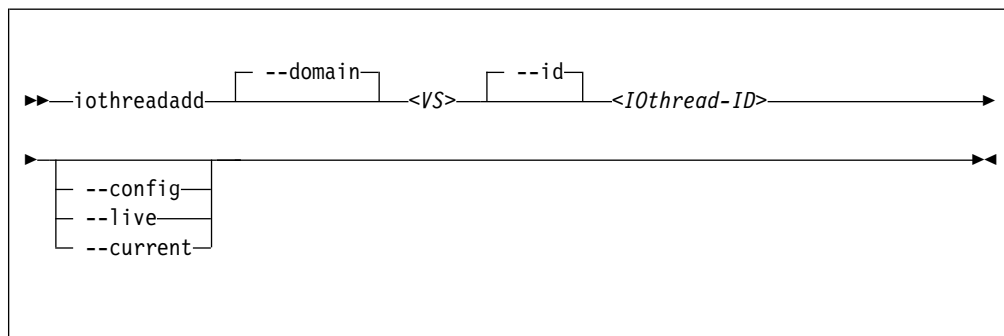
This example causes a restart interrupt for the virtual server `vserv1` including a core dump.

```
# virsh inject-nmi vserv1
```

iothreadadd

Provides an additional I/O thread for a virtual server.

Syntax



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 142

iothreadadd

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

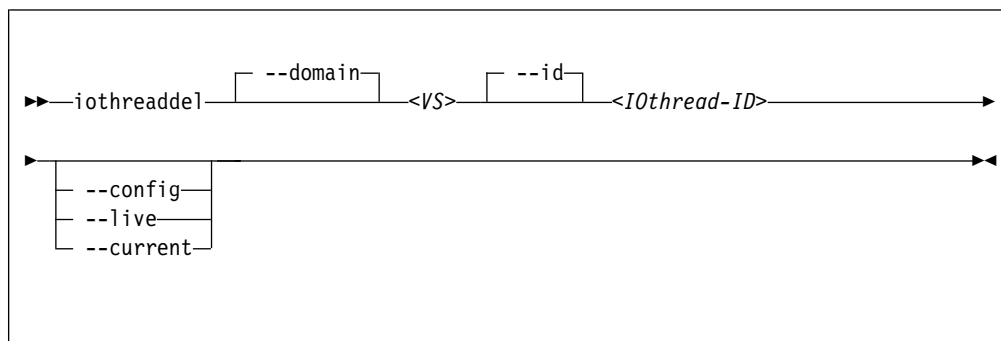
# virsh iothreadinfo vserv1
IOThread ID    CPU Affinity
-----
1              0-7
2              0-7
3              0-7
4              0-7
```


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



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 143

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

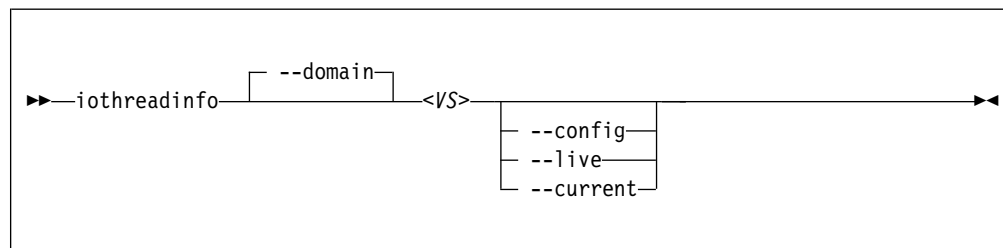
# virsh iothreaddel vserv1 3

# 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 116

Example

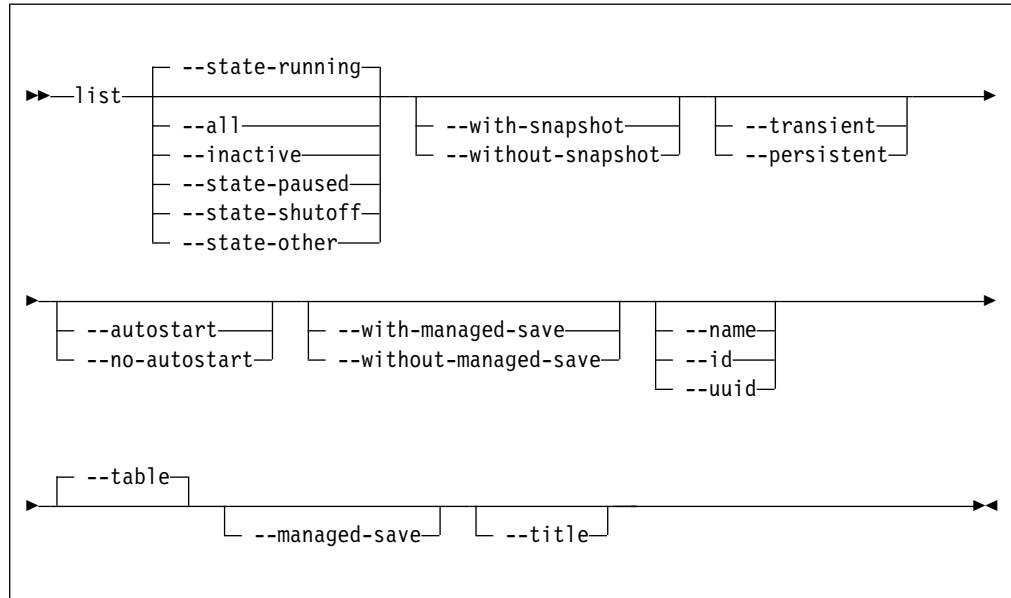
This example shows the **iothreadinfo** command for 8 virtual CPUs:

```

# 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 116

Example

This example lists all defined virtual servers.

```
# virsh list --all
```

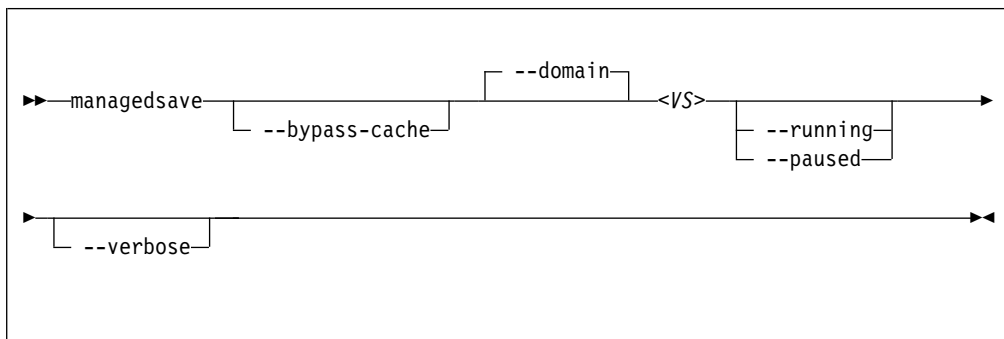
managedsave

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 <i>from running</i>)
managedsave	paused	shut off (saved <i>from paused</i>)
managedsave --running	running	shut off (saved <i>from running</i>)
managedsave --running	paused	shut off (saved <i>from running</i>)
managedsave --paused	running	shut off (saved <i>from paused</i>)
managedsave --paused	paused	shut off (saved <i>from paused</i>)

Usage

- “Terminating a virtual server” on page 110
- Chapter 25, “Virtual server life cycle,” on page 177

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

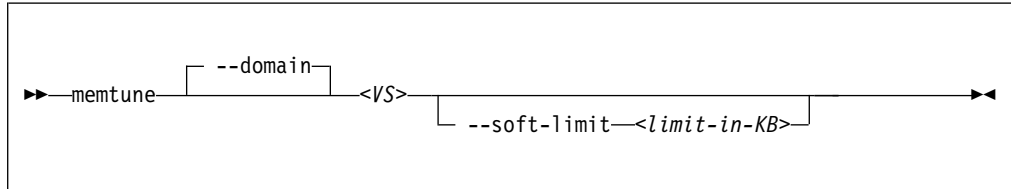
# virsh start vserv1
Domain vserv1 started

# 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 20, “Memory management,” on page 157
- “Managing virtual memory” on page 139

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
```

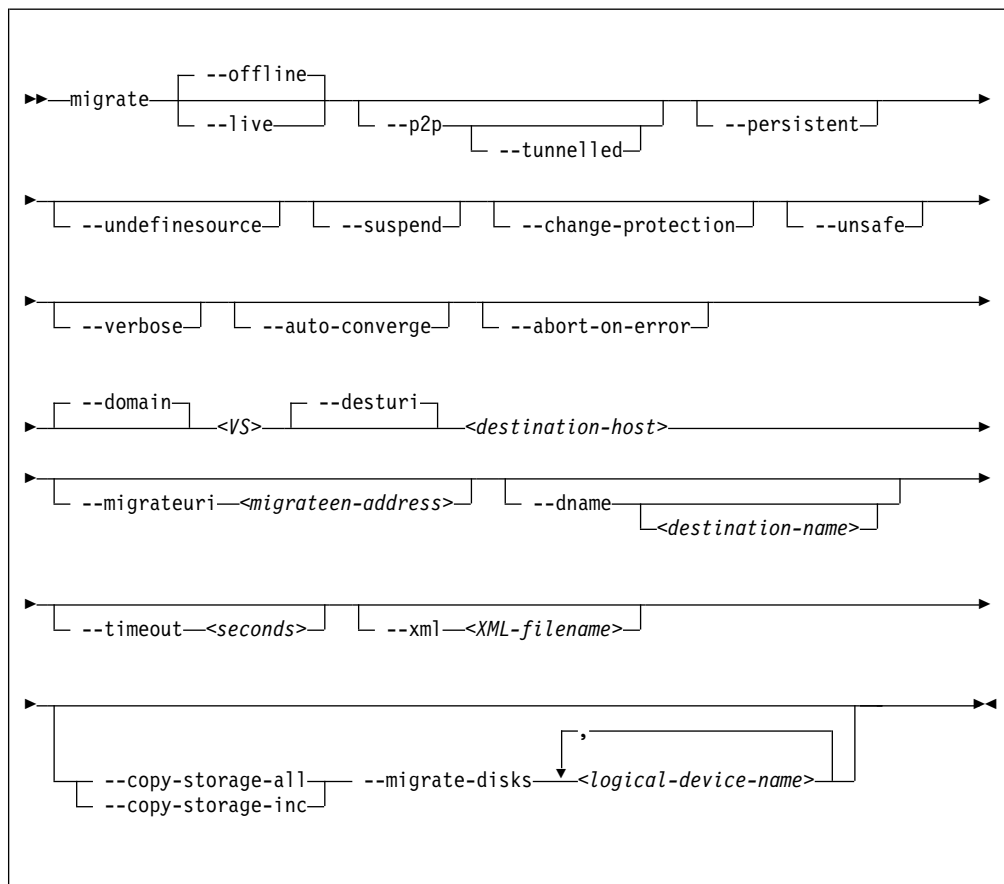
This example displays the memory tuning parameters of `vserv1`. Be sure not to modify the `hard_limit` and `swap_hard_limit` parameters.

```
# 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.

<migrateuri-address>

The host specific URI of the destination host.

<VS> Is the name, ID, or UUID of the virtual server.

migrate

<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 123

Example

This example migrates the virtual server `vserv1` to the host `zhost`.

```
# virsh migrate --auto-converge --timeout 300 vserv1 qemu+ssh://zhost/system
```

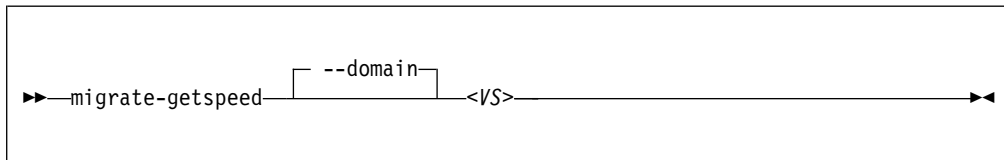
More information

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 123

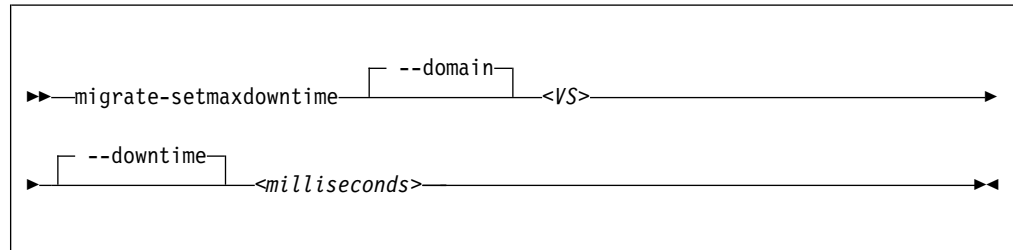
Example

```
# 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



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 123

Example

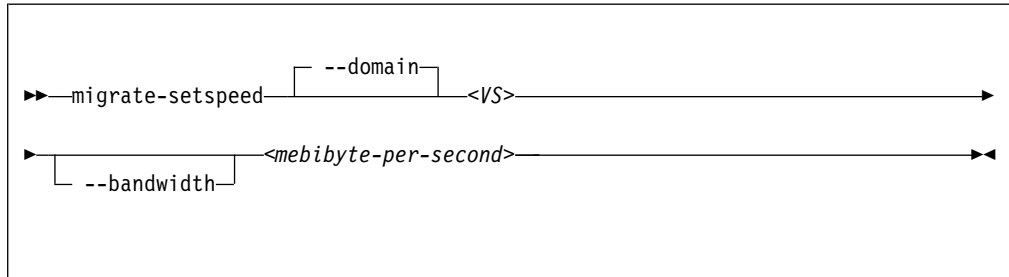
This example specifies a tolerable downtime of 100 milliseconds for the virtual server vserv1 in case it is migrated to another host.

```
# 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 123

Example

```
# virsh migrate-setspeed vserv1 --bandwidth 100
# virsh migrate-getspeed vserv1
100
```

pool-autostart

Enables or disables the automatic start of a storage pool when the libvirt daemon is started.

Syntax

```

▶▶ pool-autostart <pool-name> <pool-UUID> [--disable]

```

Where:

<pool-name>

Is the name of the storage pool.

<pool-UUID>

Is the UUID of the storage pool.

Selected options

--disable

Disables the automatic start of the storage pool when the libvirt daemon is started.

Usage

“Storage pool management commands” on page 148

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

```
▶▶ pool-define <XML-filename> ◀◀
```

Where:

<XML-filename>

Is the name of the storage pool configuration-XML file.

Selected options

None.

Usage

“Storage pool management commands” on page 148

Example

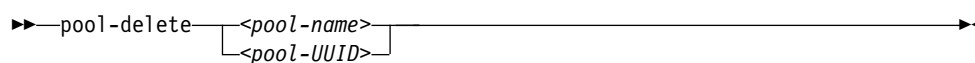
This example defines the storage pool that is configured by the storage pool configuration-XML file named pool1.xml.

```
# virsh pool_define pool1.xml
```


pool-delete

Deletes the volumes of a storage pool.

Syntax



```
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

None.

Usage

“Storage pool management commands” on page 148

Example

This example deletes the volumes of storage pool pool1.

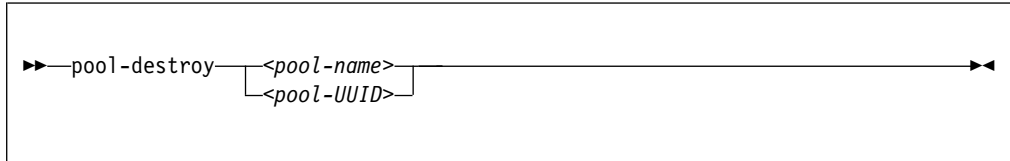
```
# virsh pool_destroy pool1
```

pool-destroy

Shut down a storage pool.

The pool can be recreated by using the virsh **pool-create** command.

Syntax



Where:

<pool-name>

Is the name of the storage pool.

<pool-UUID>

Is the UUID of the storage pool.

Selected options

None.

Usage

“Storage pool management commands” on page 148

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

```

▶▶ pool-dumpxml <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

None.

Usage

“Storage pool management commands” on page 148

Example

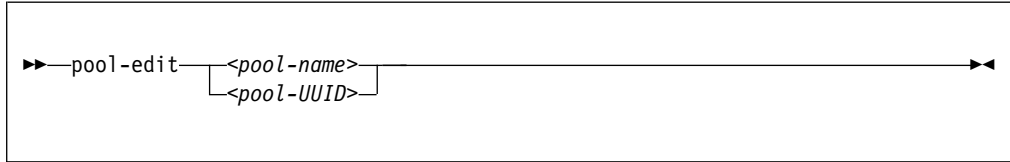
```

# 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

None.

Usage

“Storage pool management commands” on page 148

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

```
▶▶ pool-info <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

None.

Usage

“Storage pool management commands” on page 148

Example

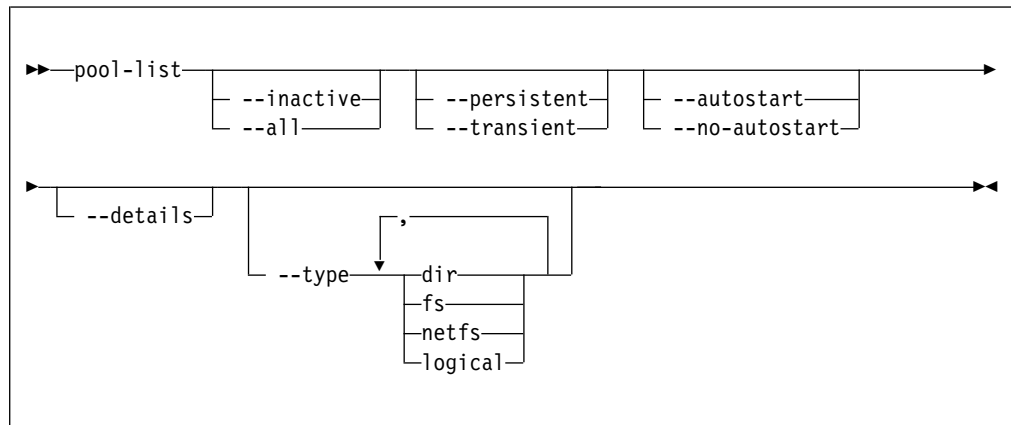
```
# virsh pool-info pool1
```

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

“Storage pool management commands” on page 148

Example

```
# virsh pool-list
```

pool-name

Displays the name of a storage pool specified by its UUID.

Syntax

```
▶▶ pool-name <pool-UUID> ◀◀
```

Where:

<pool-UUID>

Is the UUID of the storage pool.

Selected options

None.

Usage

“Storage pool management commands” on page 148

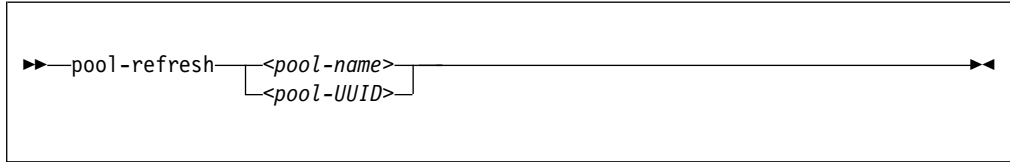
Example

```
# virsh pool-name pool1
```

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

None.

Usage

“Storage pool management commands” on page 148

Example

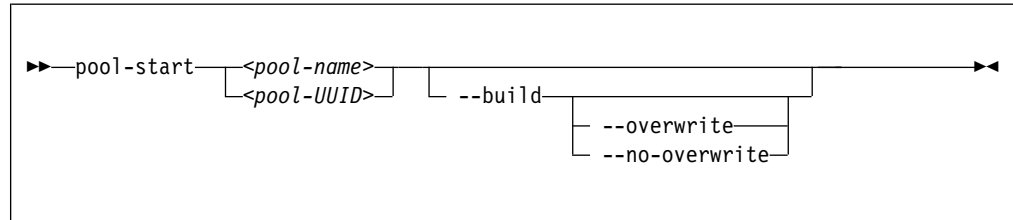
This example updates the list of volumes contained in storage pool pool1.

```
# 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

--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

“Storage pool management commands” on page 148

Example

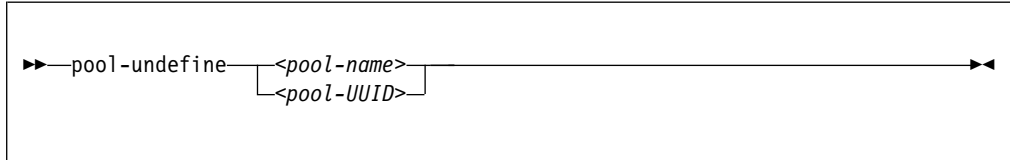
This example starts storage pool pool1.

```
# 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

None.

Usage

“Storage pool management commands” on page 148

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-uuid <pool-name> ◀◀
```

Where:

<pool-name>

Is the name of the storage pool.

Selected options

None.

Usage

“Storage pool management commands” on page 148

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 223).

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 112

Example

```
# virsh list
 Id   Name                               State
-----
 13   vserv1                             paused

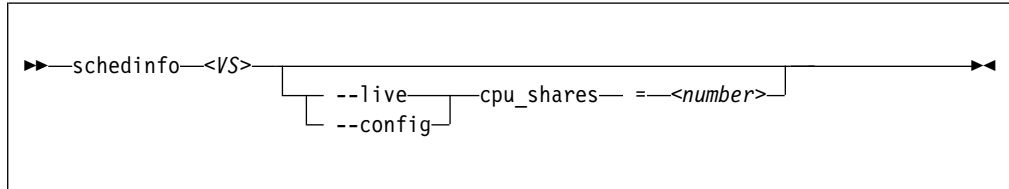
# virsh resume vserv1
Domain vserv1 resumed

# 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 137

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



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 110

Example

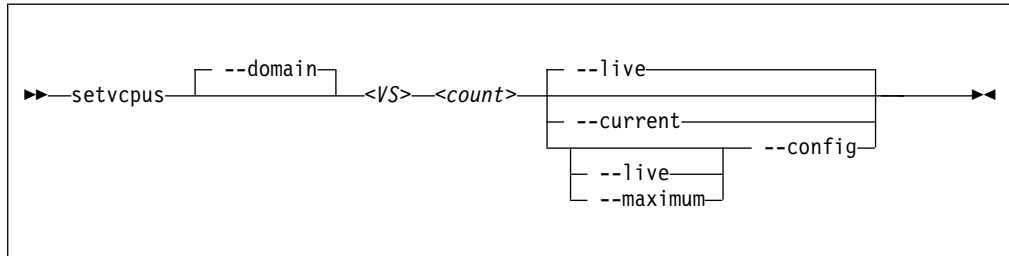
This example terminates virtual server `vserv1`.

```
# 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 134

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

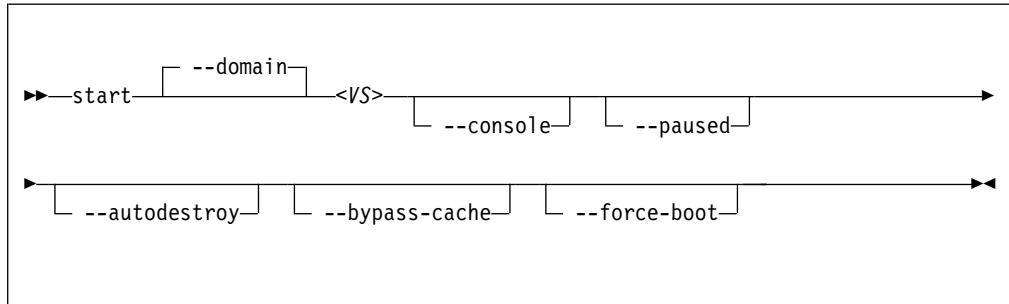
# virsh setvcpus vserv1 4 --live --config

# 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 <i>from running</i>)	running (restored)
start	shut off (saved <i>from paused</i>)	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 <i>from running</i>)	running (booted)
start --force-boot	shut off (saved <i>from paused</i>)	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 <i>from running</i>)	paused (migrating)
start --paused	shut off (saved <i>from paused</i>)	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 110
- “Connecting to the console of a virtual server” on page 145

Example

This example starts virtual server `vserv1` with initial console access.

```
# virsh start vserv1 --console
Domain vserv1 started
```

suspend

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 112

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
- “Undefineding a virtual server” on page 107

Example

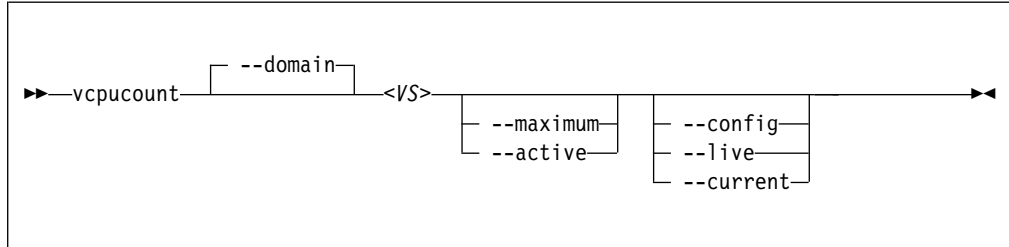
This example removes virtual server vserv1 from the libvirt definition.

```
# 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 134

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 149

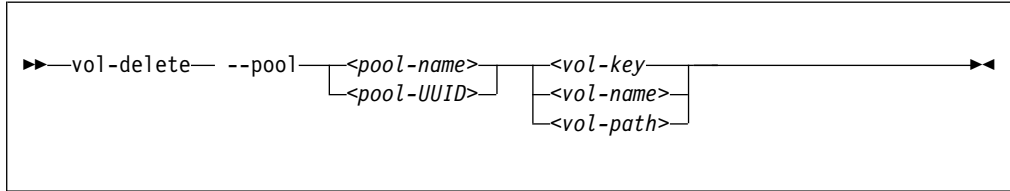
Example

```
# virsh vol-create pool1 vol1.xml
```

vol-delete

Remove a volume from a storage pool.

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 149

Example

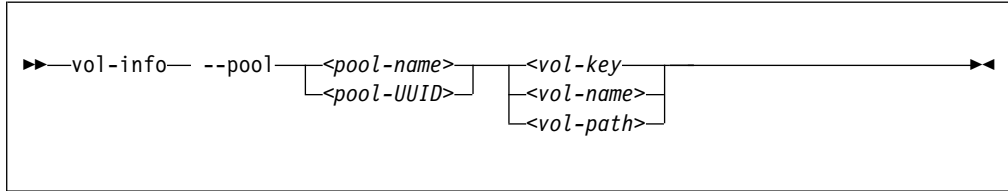
```
# virsh vol-delete --pool pool1 vol1
```


vol-info

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 149

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> | <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-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 149

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

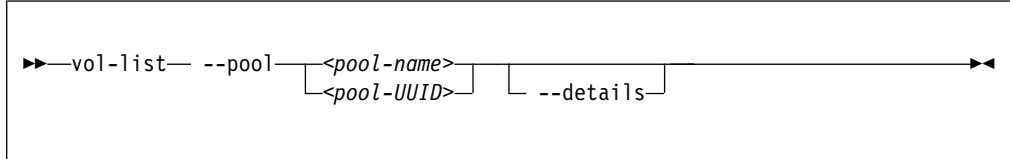
```

vol-list

Displays a list of defined storage pools.

By default, a list of active storage pools is displayed.

Syntax



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 149

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>|<pool-UUID>|<vol-key>|<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 149

Example

This example displays the volume name of vol1 as a volume of storage pool pool1.

```

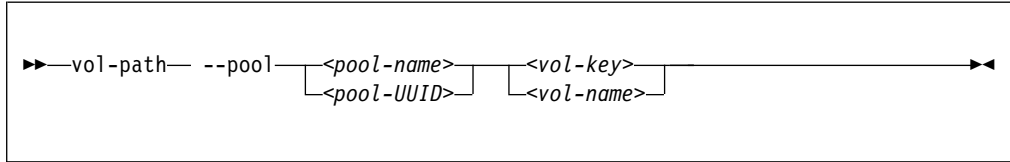
# virsh vol-name --pool pool1 /var/lib/libvirt/images/federico.img
vol1

```

vol-path

Displays the path of a volume from its name or key.

Syntax



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 149

Example

This example displays the volume key of vol1 as a volume of storage pool pool1.

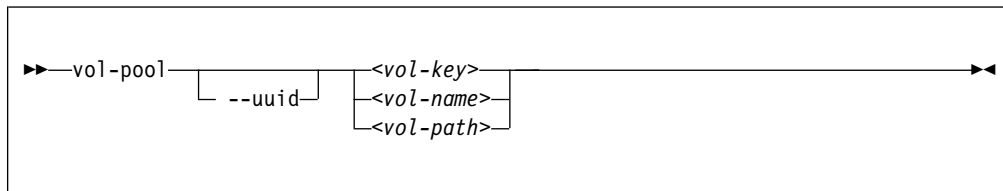
```
# virsh vol-path --pool pool1 vol1  
/var/lib/libvirt/images/federico.img
```

vol-pool

Displays the name or the UUID of the storage pool containing a given volume.

By default, the storage pool name is displayed.

Syntax



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 149

Example

```
# virsh vol-pool vol1
pool1
```

vol-pool

Chapter 28. 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.

info Displays information about 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 29. 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 z Systems hardware and the LPAR on which the KVM host runs.

The instruction provides applications with details about the maximum processing capacity that is available to the KVM guest. 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	<p>Header Flag Byte 1</p> <p>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.</p> <p>0x80 Global Performance Data unavailable.</p> <p>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.</p> <p>0x20 Virtualization stack is incomplete. This bit indicates one of two cases:</p> <ul style="list-style-type: none"> • 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. <p>0x10 Execution environment is not within a logical partition.</p>
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

Length	Data Type	Offset (dec)	Name	Contents
2	Unsigned Binary Integer	24	INFGOFF1	Offset to Guest Section1 mapped by INFOGST
2	Unsigned Binary Integer	26	INFGLEN1	Length of Guest Section1
2	Unsigned Binary Integer	28	INFHOFF2	Offset to Hypervisor Section2 mapped by INFOHYP
2	Unsigned Binary Integer	30	INFHLEN2	Length of Hypervisor Section2
2	Unsigned Binary Integer	32	INFGOFF2	Offset to Guest Section2 mapped by INFOGST
2	Unsigned Binary Integer	34	INFGLEN2	Length of Guest Section2
2	Unsigned Binary Integer	36	INFHOFF3	Offset to Hypervisor Section3 mapped by INFOHYP
2	Unsigned Binary Integer	38	INFHLEN3	Length of Hypervisor Section3
2	Unsigned Binary Integer	40	INFGOFF3	Offset to Guest Section3 mapped by INFOGST
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, INFMDPCPS, INFMSIFL, and INFMDIFL contain valid counts. The validity bit may be off when: <ul style="list-style-type: none"> • 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

Length	Data Type	Offset (dec)	Name	Contents
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	INFMDPCS	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	INFNAME	Machine Name
4	EBCDIC	20	INFMTYPE	Type
16	EBCDIC	24	INFMMANU	Manufacturer
16	EBCDIC	40	INFMSIQ	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: INFPNUM and INFPNAM. 0x08 This bit being on indicates that INFPNGM, INFPNGCP, and INFPNGIF are valid.
1	Bitstring	3	INFPVAL2	Partition Validity Byte 2 reserved for IBM use

Length	Data Type	Offset (dec)	Name	Contents
2	Unsigned Binary Integer	4	INFPPNUM	Logical partition number
2	Unsigned Binary Integer	6	INFPCSPS	Number of shared logical CPs configured for this partition. Count of cores when MT is enabled.
2	Unsigned Binary Integer	8	INFPCDPS	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	EBCDIC	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.
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	INFPAIIF	Partition absolute capped capacity for IFLs, a scaled number where X'00010000' represents one core. Zero if not capped.
8	EBCDIC	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 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

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Adobe is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Index

Special characters

- active option
 - of the vcpucount virsh command 312
- all option
 - of the list virsh command 107, 116, 278
- auto-converge option
 - of the migrate virsh command 128, 129
- autodestroy option
 - of the start virsh command 308
- autostart option
 - of the list virsh command 278
- bandwidth option
 - of the migrate-setspeed virsh command 129, 288
- build option
 - of the pool-start virsh command 299
- bypass-cache option
 - of the managedsave virsh command 280
 - of the start virsh command 308
- change-protection option
 - of the migrate virsh command 283
- compressed option
 - of the migrate virsh command 283
- config option
 - of the attach-device virsh command 142, 252
 - of the change-media virsh command 254
 - of the detach-device virsh command 259
 - of the iothreadadd virsh command 273
 - of the iothreaddel virsh command 275
 - of the iothreadinfo virsh command 277
 - of the schedinfo virsh command 137, 304
 - of the setvcpus virsh command 306
 - of the vcpucount virsh command 312
- console option
 - of the start virsh command 110, 145, 308
- copy-storage-all option
 - of the migrate virsh command 129, 283
- copy-storage-inc option
 - of the migrate virsh command 129, 283
- current option
 - of the change-media virsh command 254
 - of the detach-device virsh command 259
 - of the iothreadadd virsh command 273
 - of the iothreaddel virsh command 275
 - of the iothreadinfo virsh command 277
 - of the setvcpus virsh command 306
 - of the vcpucount virsh command 312
- direct option
 - of the migrate virsh command 283
- disable option
 - of the pool-autostart virsh command 289
- domain option
 - of the attach-device virsh command 252
 - of the change-media virsh command 254
 - of the destroy virsh command 258
 - of the detach-device virsh command 259
 - of the dumpxml virsh command 270
 - of the shutdown virsh command 305
 - of the start virsh command 308
- eject option
 - of the change-media virsh command 143, 254
- file option
 - of the attach-device virsh command 252
 - of the detach-device virsh command 259
- force option
 - of the change-media virsh command 254
 - of the console virsh command 256
- force-boot option
 - of the start virsh command 110, 308
- graceful option
 - of the destroy virsh command 110, 258
- id option
 - of the iothreadadd virsh command 273
 - of the iothreaddel virsh command 275
 - of the list virsh command 278
- inactive option
 - of the dumpxml virsh command 270
 - of the list virsh command 278
- insert option
 - of the change-media virsh command 143, 254
- keepalive-count option
 - of the virsh command 129
- keepalive-interval option
 - of the virsh command 129
- live option
 - of the change-media virsh command 254
 - of the detach-device virsh command 259
 - of the iothreadadd virsh command 273
 - of the iothreaddel virsh command 275
 - of the iothreadinfo virsh command 277
 - of the migrate virsh command 283
 - of the schedinfo virsh command 137, 304
 - of the setvcpus virsh command 306
 - of the vcpucount virsh command 312
- managed-save option
 - of the list virsh command 278
- maximum option
 - of the setvcpus virsh command 306
 - of the vcpucount virsh command 312
- memory-only option
 - of the dump virsh command 169, 269
- migratable option
 - of the dumpxml virsh command 270
- migrate-disks option
 - of the migrate virsh command 129, 283
- mode option
 - of the shutdown virsh command 305
- name option
 - of the list virsh command 278
- no-autostart option
 - of the list virsh command 278
- no-overwrite option
 - of the pool-start virsh command 299
- offline option
 - of the migrate virsh command 283
- overwrite option
 - of the pool-start virsh command 299
- p2p option
 - of the migrate virsh command 283
- path option
 - of the change-media virsh command 254

- paused option
 - of the managedsave virsh command 280
 - of the start virsh command 308
- persistent option
 - of the migrate virsh command 283
- persistent option
 - of the detach-device virsh command 259
 - of the list virsh command 278
 - of the migrate virsh command 283
- reason option
 - of the domstate virsh command 116, 268
- running option
 - of the managedsave virsh command 280
- safe option
 - of the console virsh command 256
- security-info option
 - of the dumpxml virsh command 270
- soft-limit option
 - of the memtune virsh command 282
- state-other option
 - of the list virsh command 278
- state-paused option
 - of the list virsh command 278
- state-running option
 - of the list virsh command 278
- state-shutoff option
 - of the list virsh command 278
- suspend option
 - of the migrate virsh command 283
- table option
 - of the list virsh command 278
 - of the migrate virsh command 283
- timeout option
 - of the migrate virsh command 128, 129, 283
- title option
 - of the list virsh command 278
- transient option
 - of the list virsh command 278
 - of the migrate virsh command 283
- tunnelled option
 - of the migrate virsh command 283
- undefinesource option
 - of the migrate virsh command 283
- unsafe option
 - of the migrate virsh command 283
- update option
 - of the change-media virsh command 254
- update-cpu option
 - of the dumpxml virsh command 270
- uuid option
 - of the list virsh command 278
- verbose option
 - of the managedsave virsh command 280
 - of the migrate virsh command 283
- with-managed-save option
 - of the list virsh command 278
- with-snapshot option
 - of the list virsh command 278
- without-managed-save option
 - of the list virsh command 278
- without-snapshot option
 - of the list virsh command 278
- xml option
 - of the migrate virsh command 283

/etc/hosts 106
 /etc/libvirt/libvirtd.conf 129, 167
 /etc/libvirt/qemu.conf 127

/etc/lvm/lvm.conf 161
 /etc/ssh/sshd_config 127
 /etc/zfcp.conf 29
 /var/log/libvirtd/qemu/<VS>.log 167
 /var/log/messages 167
 \$EDITOR environment variable 106
 \$VISUAL environment variable 106

Numerics

0xfe value

- of the address cssid attribute 86

 3DEA 69
 3DES 69

A

accessibility 333
 action attribute

- of the watchdog element 247

 activating

- bonded interfaces 40
- network interfaces 38

 active bridge port 43
 adapter element 88, 185

- name attribute 88, 185

 adding

- CPUs 134
- I/O threads 142

 address attribute

- of the mac element 96, 98, 215
- of the target element 239

 address element 74, 76, 82, 84, 86, 88, 93, 186, 187, 188

- bus attribute 88, 93, 187, 188
- controller attribute 88, 93, 187
- cssid attribute 76, 82, 84, 86, 186
 - 0xfe value 86
- devno attribute 76, 82, 84, 86, 186
- ssid attribute 76, 82, 84, 86, 186
- target attribute 88, 93, 187, 188
- type attribute 76, 82, 84, 86, 88, 186, 187
 - ccw value 76, 82, 84, 86, 186
 - drive value 93
 - scsi value 88
- unit attribute 88, 93, 187, 188

 Advanced Encryption Standard 69
 AES 69
 aes value

- of the cipher name attribute 69, 191

 alias device 27
 alias machine type 122, 123
 append attribute

- of the log element 67, 214

 arch attribute

- of the target element 243
- of the type element 51

 attach-device command

- config option 142

 attach-device virsh command 142, 252
 attaching

- devices 142

 auto value

- of the controller model attribute 194
- of the geometry trans attribute 205

B

- backend element 189
 - model attribute 189
- balloon device 71
- base device 27
- block device 9, 14, 73
 - attaching 142
 - detaching 143
 - displaying 116
 - hotplugging 142
 - unplugging 143
- block device driver 4
- block value
 - of the disk type attribute 76, 198
- bonded interface 21, 40
 - activating 40
 - configuring 96
 - preparing 40
 - setting up 40
- bonded network interface 40
- bonding
 - mode 40
 - parameter 40
- boot element 53, 190
 - order attribute 53, 190
- boot process
 - configuring 53
- bootable kernel 53, 55
- booted reason
 - of the running state 179
- booting
 - from a DASD 53
 - from a kernel image file 55
 - from a SCSI disk 53
 - from an ISO image 54
- bridge attribute
 - of the source element 98
- bridge port 43
 - active 43
 - primary 43
- bridge port role
 - enabling 43
- bridge value
 - of the interface type attribute 98, 209
 - of the source mode attribute 96
- bridge_state sysfs attribute 43
- browsing
 - virtual servers 116
- bus attribute
 - of the address element 88, 93, 187, 188
 - of the target element 76, 82, 84, 93, 240
- by-ID device node 9, 14
- by-path device node 9, 14, 27, 76
- by-uuid device node 9, 14

C

- cache attribute
 - of the driver element 76, 82, 84, 93, 201
- canceling
 - a live migration 129
- CCW device 73
- CCW group device 38, 43
- ccw value
 - of the address type attribute 76, 82, 84, 86, 186
- CD-ROM 73

- cdrom value
 - of the disk device attribute 93, 198
- Central Processor Assist for Cryptographic Functions 69
- cgroups 154
- change-media command
 - eject option 143
 - insert option 143
- change-media virsh command 143, 254
- channel bonding 40
- channel bonding module 40
- channel path 9, 14, 73
- channel path type 73
- channel subsystem 9, 14, 73
- channel subsystem-ID 9, 14, 73, 76, 82, 84
- chccwdev command 27, 29, 33
- chcpu command 134
- cipher element 69, 191
 - name attribute 69, 191
 - aes value 69, 191
 - dea value 69, 191
 - state attribute 69, 191
 - off value 69, 191
 - on value 69, 191
- clear key 69
- cmdline element 51, 55, 192
- collecting
 - performance metrics 171
- command
 - chccwdev 29, 33
 - chcpu 134
 - crash 169
 - fdisk 29
 - ip 37, 38
 - ls 76
 - lscss 33, 76
 - lsqeth 38, 43
 - lsscsi 29
 - lszfcp 29, 33
 - modprobe 40
 - multipath 29, 76
 - 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 171
 - pvscan 161
 - QEMU monitor info 323
 - qemu-kvm 122, 123
 - start openvswitch 43
 - status openvswitch 43
 - virsh attach-device 142, 252
 - virsh change-media 143, 254
 - virsh console 145, 256
 - virsh define 106, 257
 - virsh destroy 110, 258
 - virsh detach-device 143, 259
 - virsh dombklist 116, 261
 - virsh domblkstat 116, 262
 - virsh domiflist 116, 264
 - virsh domifstat 116, 265
 - virsh dominfo 116, 266
 - virsh domjobabort 129, 267
 - virsh domstate 116, 268
 - virsh dump 169, 269
 - virsh dumpxml 86, 118, 270
 - virsh edit 106, 271

- command *(continued)*
 - virsh inject-nmi 272
 - virsh iothreadadd 273
 - virsh iothreaddel 275
 - virsh iothreadinfo 277
 - virsh list 107, 116, 141, 145, 278
 - virsh managedsave 110, 280
 - virsh memtune 282
 - virsh migrate 129, 283
 - virsh migrate-getspeed 129, 286
 - virsh migrate-setmaxdowntime 129
 - virsh migrate-setspeed 129, 288
 - virsh pool-autostart 289
 - virsh pool-define 290
 - virsh pool-delete 291
 - virsh pool-destroy 292
 - virsh pool-dumpxml 293
 - virsh pool-edit 294
 - virsh pool-info 295
 - virsh pool-list 148, 296
 - virsh pool-name 297
 - virsh pool-refresh 298
 - virsh pool-start 299
 - virsh pool-undefine 300
 - virsh pool-uuid 301
 - virsh reboot 302
 - virsh resume 112, 303
 - virsh schedinfo 116, 137, 304
 - virsh setvcpus 134, 306
 - virsh shutdown 110, 305
 - virsh start 110, 145, 308
 - virsh suspend 112, 310
 - virsh undefine 107, 311
 - virsh vcpuaccount 116, 134, 312
 - virsh vol-create 313
 - virsh vol-delete 314
 - virsh vol-dumpxml 315
 - virsh vol-info 316
 - virsh vol-key 317
 - virsh vol-list 149, 318
 - virsh vol-name 319
 - virsh vol-path 320
 - virsh vol-pool 321
 - zipl 53, 55
 - znetconf 37, 38, 43, 96
- concurrency 127
- concurrent connections 127
- CONFIG_HAVE_PERF_EVENTS 171
- CONFIG_PERF_EVENTS 171
- CONFIG_TRACEPOINTS 171
- configuration
 - libvirt-internal 118
 - of devices 4
 - of virtual servers 4
- configuration file
 - libvirt 167
 - of the OpenSSH SSH daemon configuration file 127
- configuration topology 9, 14
- configuration-XML file 9, 14, 73
 - of a device 4, 9, 14, 74
 - of a domain 4, 9, 14, 51
 - of a storage pool 101
 - of a volume 101
- configuring
 - an ISO image as IPL device 54
 - bonded interfaces 96
 - boot devices 49, 53
- configuring *(continued)*
 - boot process 53, 54, 55
 - consoles 67
 - CPUs 49, 60
 - DASDs 76
 - devices 4, 73
 - Ethernet interfaces 96
 - FC-attached SCSI tape devices 88
 - I/O threads 66
 - image files as storage devices 82
 - logical volumes 161
 - memory 49, 62
 - multipath device mapper support 29
 - network devices 49
 - network interfaces 96
 - operating systems 49
 - persistent devices 66
 - physical volumes 161
 - protected key encryption 69
 - random number generators 100
 - removable ISO images 93
 - SCSI disks 76
 - SCSI medium changer devices 86, 88
 - SCSI tapes 86, 88
 - storage devices 49
 - storage pools 101
 - user space 49, 65
 - virtual CPUs 60
 - virtual Host Bus Adapters 86
 - virtual memory 62
 - virtual servers 4
 - virtual switches 96, 98
 - volumes as storage devices 84
 - watchdog devices 68
- connecting
 - to the console of a virtual server 145
- connections 127
- console 67, 145
 - configuring 67
 - connecting 145
- console element 67, 193
 - type attribute 67, 193
 - pty value 67, 193
- console log file 67, 167
- console output
 - logging 67
- console virsh command 145, 256
- control unit model 73
- controller attribute
 - of the address element 88, 93, 187
- controller element 74, 86, 194
 - index attribute 86, 194
 - model attribute 86, 194
 - virtio-scsi value 86
 - ports attribute 194
 - type attribute 86, 194
 - scsi value 86
 - vectors attribute 194
- core dump
 - configurable 64
- CPACF 69
- CPU
 - adding 134
 - configuring 60
 - management 153
 - modifying the number 134
 - pinning 153, 154

- CPU migration 153
- CPU shares 137, 154
- CPU time 137, 154
- CPU weight 61, 137, 154
 - modifying 137
- cpu_shares parameter
 - of the schedinfo virsh command 137, 304
- cputune element 51, 61, 195
- crash command 169
- crashed state 177, 181
- creating
 - file systems 29
 - network interfaces 38
 - partitions 29
 - persistent virtual server definitions 105
 - uplink ports 43
 - virtual server dumps 169
 - virtual servers 4
 - virtual switches 43
- cryptographic key 69
- cssid attribute
 - of the address element 76, 82, 84, 86, 186
- current attribute
 - of the vcpu element 244
- current live 134
- cyls attribute
 - of the geometry element 205

D

- DASD 76
 - configuring 76
 - preparing 27
 - setting up 27
 - virtualization 9, 14
- DASD configuration
 - example 79
- dasdfmt command 27
- Data Encryption Algorithm 69
- Data Encryption Standard 69
- DEA 69
- dea value
 - of the cipher name attribute 69, 191
- deadlock prevention 127
- default value
 - of the driver cache attribute 201
- define virsh command 106, 257
- defined virtual server 4
- defining
 - virtual servers 4, 106
- definition 4, 122, 123
 - creating 106
 - deleting 107
 - modifying 106
- deleting
 - virtual server definitions 105
- DES 69
- destination port 127
- destroy text content
 - of the on_reboot element 223
- destroy virsh command 110, 258
 - graceful option 110, 258
- destroyed reason
 - of the shut off state 178
- destroying
 - virtual servers 110
- detach-device virsh command 143, 259
- detaching
 - devices 143
- dev attribute
 - of the source element 76, 82, 84, 96, 234, 237
 - of the target element 76, 82, 84, 93, 240
- device
 - attaching 142
 - configuring 73
 - detaching 143
 - hotplugging 142
 - managing 141
 - unplugging 143
- device attribute
 - of the disk element 76, 82, 84, 93, 198
- device bus-ID 9, 14, 37
 - of a virtual block device 9, 14, 76, 82, 84
 - of an FCP device 9, 14, 29
- device configuration topology 9, 14
- device configuration-XML 4
- device configuration-XML file 4, 73, 74
 - child elements 74
 - root element 74
- device driver
 - lin_tape 17, 33
 - MacVTap 21
 - SCSI generic 17
 - watchdog 171
- device element 196
 - path attribute 196
- device mapper-created device node 9, 14, 29, 76
- device name 9, 14
 - logical 76, 82, 84
 - standard 9, 14
- device node 9, 14, 76
 - device mapper 9, 14
 - device mapper-created 29, 76
 - standard 9, 14, 76
 - udev-created 9, 14, 76
- device number 9, 14
 - of a virtual block device 9, 14, 76, 82, 84
 - of an FCP device 9, 14
- device type 9, 14, 73
- device-mapper multipathing 29
- devices element 51, 197
- devno attribute
 - of the address element 76, 82, 84, 86, 186
- DIAG event 171
- diagnose 171
- direct connection 96
- direct MacVTap connection 40
- direct value
 - of the interface type attribute 96, 209
- directsync value
 - of the driver cache attribute 201
- dirty pages 128
- disabling
 - protected key encryption 69
- disk 73
- disk element 74, 76, 82, 84, 93, 198
 - device attribute 76, 82, 84, 93, 198
 - cdrom value 93, 198
 - disk value 76, 82, 84, 198
 - type attribute 76, 82, 84, 93, 198
 - block value 76, 198
 - file value 82, 84, 93, 198
 - disk migration 124, 129

- disk value
 - of the disk device attribute 76, 82, 84, 198
- displaying
 - block devices 116
 - information about a virtual server 116
 - network interfaces 116
 - performance metrics 171
 - scheduling information 116
 - states 116
 - the libvirt-internal configuration 118
- domain 3, 4
- domain configuration-XML 4, 9, 14
- domain configuration-XML file 4, 49, 51, 73
 - child elements 51
 - root element 51
- domain element 199
 - type attribute 51, 199
- domblklist virsh command 116, 261
- domblkstat virsh command 116, 262
- domiflist virsh command 116, 264
- domifstat virsh command 116, 265
- dominfo virsh command 116, 266
- domjobabort virsh command 129, 267
- domstate virsh command 116, 268
 - reason option 116, 268
- drive value
 - of the address type attribute 93
- driver element 74, 76, 82, 84, 93, 200, 201
 - cache attribute 76, 82, 84, 93, 201
 - default value 201
 - directsync value 201
 - none value 82, 84, 93, 201
 - unsafe value 201
 - writeback value 201
 - writethrough value 82, 84, 201
 - error_policy attribute 201
 - enospace value 201
 - ignore value 201
 - report value 201
 - stop value 201
 - event_idx attribute 201
 - off value 201
 - on value 201
 - io attribute 76, 82, 84, 93, 201
 - native value 82, 84, 93, 201
 - of the driver element 82, 84
 - threads value 201
 - ioeventfd attribute 201
 - off value 201
 - on value 201
 - iothread attribute 76, 200
 - name attribute 76, 82, 84, 93, 201
 - qemu value 93, 201
 - error_policy attribute 201
 - ignore value 201
 - report value 201
 - stop value 201
 - type attribute 76, 82, 84, 93, 201
 - qcow2 value 82, 84
 - raw value 82, 84, 93, 201
- driver value
 - of the error_policy attribute 201
 - of the rerror_policy attribute 201
- dump 169
 - configurable 64
- dump file 169
- dump location 169

- dump virsh command 169, 269
 - memory-only option 169, 269
- dumpCore attribute
 - of the memory element 64
- dumping
 - virtual servers 169
- dumpxml virsh command 86, 118, 270
- DVD 73

E

- edit virsh command 106, 271
- editing
 - libvirt-internal configurations 106
 - persistent virtual server definitions 106
- emulator element 65, 203
- enabling
 - bridge port roles 43
- encryption 69
- enospace value
 - of the driver error_policy attribute 201
- environment variable
 - \$EDITOR 106
 - \$VISUAL 106
- error_policy attribute
 - of the driver element 201
- Ethernet interface
 - configuring 96
 - preparing 37
- event_idx attribute
 - of the driver element 201
- example
 - of a DASD configuration 79
 - of a multipathed SCSI device configuration 91
 - of a SCSI disk configuration 80
 - of an initial installation 56

F

- failover redundancy 29
- FC-attached SCSI medium changer device 17
- FC-attached SCSI tape device 17
 - configuring 88
 - preparing 33
- FCP device 9, 14, 29
- FCP LUN 9, 14, 29
- fdasd command 27
- fdisk command 27, 29
- file 73
- file attribute
 - of the log element 67, 214
 - of the source attribute 93
 - of the source element 82, 84, 234
- file value
 - of the disk type attribute 82, 84, 93, 198
- filtered value
 - of the hostdev sgio attribute 207
- firewall configuration 127
- format element 204
 - type attribute 204
- full isolation 21

G

- geometry element 205
 - cyls attribute 205

geometry element (*continued*)
 heads attribute 205
 secs attribute 205
 trans attribute 205
GRE tunnel 43
guest x, 3, 4
 relocation
 See live migration

H

hardware information 325
HBA 73
 configuring 86
heads attribute
 of the geometry element 205
high reliability 9, 14, 17, 21
host x, 4, 40
Host Bus Adapter 73
 configuring 86
host device 88
host element 206
 name attribute 206
host network
 OSA-Express device 40
hostdev element 74, 88, 207
 mode attribute 88, 207
 subsystem value 88
 rawio attribute 207
 sgio attribute 207
 type attribute 88, 207
 scsi value 88
hotplug device 4, 74, 88, 142, 143
hotplugging
 devices 73, 142
hvm text content
 of the type element 243
hypervisor
 release 122, 123
hypervisor information for the virtual server user 325
hypervisor release 127

I

I/O operations
 improving performance 66, 161
I/O thread 76, 161
 adding 142
 configuring 66
 providing 66
 removing 143
ignore value
 of the driver error_policy attribute 201
 of the driver rerror_policy attribute 201
image file 73
 qcow2 82, 84
 raw 82, 84
improving
 the performance of I/O operations 66, 161
index attribute
 of the controller element 86, 194
info QEMU command 323
initial installation
 example 56
Initial Program Load 53, 110
initial ramdisk 53, 55, 56

initrd element 51, 55, 208
inject-nmi virsh command 169, 272
installation DVD 54
installation file 55
installing a guest
 from a kernel image file 55
 from an ISO image 54
interface
 bonded 40
 MacVTap 40
 virtual LAN 40
 VLAN 40
interface element 74, 96, 98, 209
 trustGuestRxFilters attribute 96, 209
 no value 209
 yes value 209
 type attribute 96, 98, 209
 bridge value 98, 209
 direct value 96, 209
interface name 37
io attribute
 of the driver element 76, 93, 201
IOCDs 27, 37, 38
ioeventfd attribute
 of the driver element 201
iothread attribute
 of the driver element 76, 200
iothreadadd virsh command 273
 --config option 273
 --current option 273
 --id option 273
 --live option 273
iothreaddel virsh command 275
 --config option 275
 --current option 275
 --id option 275
 --live option 275
iothreadinfo virsh command 277
 --config option 277
 --current option 277
 --live option 277
iothreads element 51, 66, 76, 210
IP address 37
ip command 37, 38
IPL 53, 110
ISO image 17, 56, 93, 143
 as IPL device 54
 removable 93, 143

K

kdump 169
keepalive interval
 of the virsh command 129
kernel element 51, 53, 55, 211
kernel image file 53, 55, 56
kernel parameters 53, 55
 specifying 55
key element 212
keywrap element 69, 213
KVM 6
 KVM guest x, 3, 4
 KVM host x
 kvm kernel module 6
 kvm value
 of the domain type attribute 51, 199
KVM virtual server x, 3, 4

kvm_s390_sie_enter tracepoint event 171

L

layer 2 mode 38

lba value

of the geometry trans attribute 205

libvirt 4, 6, 33

libvirt configuration file 167

libvirt daemon 109, 141

starting 109, 141

libvirt XML attribute

adapter name 88

address bus 88, 93, 187, 188

address controller 88, 93, 187

address cssid 76, 82, 84, 86, 186

address devno 76, 82, 84, 86, 186

address ssid 76, 82, 84, 86, 186

address target 88, 93, 187, 188

address type 76, 82, 84, 86, 88, 93, 186, 187

address unit 88, 93, 187, 188

backend model 189

boot order 53, 190

cipher name 69, 191

cipher state 69, 191

console type 67, 193

controller index 86, 194

controller model 86, 194

controller ports 194

controller type 86, 194

controller vectors 194

device path 196

disk device 76, 82, 84, 93, 198

disk type 76, 82, 84, 93, 198

domain type 51, 199

driver cache 76, 82, 84, 93, 201

driver error_policy 201

driver event_idx 201

driver io 76, 82, 84, 93, 201

driver ioeventfd 201

driver iothread 76, 200

driver name 76, 82, 84, 93, 201

driver rerror_policy 201

driver type 76, 82, 84, 93, 201

format type 204

geometry cyls 205

geometry heads 205

geometry secs 205

geometry trans 205

host name 206

hostdev mode 88, 207

hostdev rawio 207

hostdev sgio 207

hostdev type 88, 207

interface trustGuestRxFilters 209

interface type 96, 98, 209

log append 67, 214

log file 67, 214

mac address 96, 98

memballoon model 71, 216

memory dumpCore 64

memory unit 62

model type 98

pool type 227

source bridge 98

source dev 76, 96, 234

source file 82, 84, 93, 234

libvirt XML attribute (*continued*)

source mode 96

source pool 234

source startupPolicy 234

source volume 234

target address 239

target bus 76, 82, 84, 93, 240

target dev 76, 82, 84, 93, 240

target port 239

target type 67, 239

type arch 51, 243

type machine 51, 243

vcpu current 244

virtualport type 98, 245

volume type 246

watchdog action 247

watchdog model 247

libvirt XML element

adapter 88

address 74, 76, 82, 84, 86, 88, 93, 186, 187, 188

backend 189

boot 53, 190

cipher 69, 191

cmdline 51, 55, 192

console 67, 193

controller 74, 86, 194

cpuname 51, 61, 195

device 196

devices 51, 197

disk 74, 76, 82, 84, 93, 198

domain 199

driver 74, 76, 82, 84, 93, 200, 201

emulator 65, 203

format 204

geometry 205

host 206

hostdev 74, 88, 207

initrd 51, 55, 208

interface 74, 96, 98, 209

iothreads 51, 66, 76, 210

kernel 51, 55, 211

key 212

keywrap 69, 213

log 67, 214

mac 74, 96, 98, 215

memballoon 71, 216

memory 51, 62, 64, 217

memtune 62, 219

model 74, 98, 220

name 51, 221

on_crash 51, 222

on_poweroff 51

on_reboot 51, 223

os 51, 55, 224

path 225, 226

pool 227

readonly 93, 229

rng 230

shareable 231

shares 51, 61, 232

soft_limit 62, 233

source 74, 76, 82, 84, 93, 96, 98, 234, 238

target 67, 74, 76, 82, 84, 93, 239, 240, 241, 242

type 51, 243

vcpu 51, 60, 244

virtualport 98, 245

volume 246

- libvirt XML element (*continued*)
 - watchdog 247
- libvirt-internal configuration 105, 106
 - displaying 118
- libvirtd log messages 167
- lin_tape device driver 17, 33
- Linux scheduling 153
- list virsh command 107, 116, 141, 145, 278
 - all option 107, 116
- live migration 76, 88, 122, 123
 - cancellation 129
 - concurrency 127
 - concurrent connections 127
 - connections 127
 - deadlock prevention 127
 - destination port 127
 - firewall configuration 127
 - host environments 127
 - host setup 127
 - hypervisor release 127
 - image files 124
 - live phase 128
 - maximum downtime 128
 - MaxStartups parameter 127
 - memory intensive workload 128
 - memory intensive workloads 129
 - messages 283
 - migration port range 127
 - migration_port_max parameter 127
 - non-tunneled 127
 - performance considerations 127
 - phases 128
 - port range 127
 - prerequisites 124
 - preserving virtual server resources 124
 - process 129
 - setup 124
 - stopped phase 128
 - storage keys 128
 - verification 129
 - virtual server CPUs 124
 - virtual server memory 124
 - virtual server network 124
 - virtual server storage 124
- live phase of a migration 128
- live subcommand
 - or perf kvm stat 171
- live virtual server migration
 - See* live migration
- load balancing 29
- log element 67, 214
 - append attribute 214
 - off value 214
 - on value 214
 - file attribute 214
- log file
 - for console output 67
- log messages 167
- logging
 - console output 67
- logging level 167
- logical device name 76, 82, 84
- logical volume
 - configuration 161
 - management 161
- Logical Volume Manager 76, 161
- ls command 76

- lscss command 33, 76
- lsdasd command 27
- lsqeth command 38, 43
- lsscsi command 29
- lsscss command 29
- lszfcp command 29, 33
- LUN 33
- LVM 76, 161

M

- MAC address 21, 43, 98
- mac element 74, 96, 98, 215
 - address attribute 96, 98, 215
- machine attribute
 - of the target element 243
 - of the type element 51, 122, 123
- machine type
 - alias value 122, 123
- machine type of the virtual server 127
- MacVTap
 - direct connection 40
 - kernel modules 40
 - network device driver 21
- MacVTap interface 21, 96
 - preparing 40
 - setting up 40
- makedumpfile command 169
- managedsave state 110, 177
- managedsave virsh command 110, 280
 - bypass-cache option 280
 - paused option 280
 - running option 280
 - verbose option 280
- managing
 - devices 141
 - storage pools 148
 - system resources 133
 - virtual CPUs 134
 - virtual memory 139
 - virtual servers 109
 - volumes 149
- mandatory value
 - of the source startupPolicy attribute 234
- mapping a virtio block device to a host device 9, 14
- master bonded interface 40
- maximum downtime 128
- maximum number of available virtual CPUs 134
- MaxStartups parameter 127
- memballoon element 71, 216
 - model attribute 71, 216
 - none value 71, 216
- memory 62
 - configuring 62
- memory balloon device 71, 73
- memory element 51, 62, 64, 217
 - dumpCore attribute 64, 217
 - unit attribute 62, 217
- memory intensive workloads 128, 129
- memtune element 62
- memtune virsh command 282
 - soft-limit option 282
- migrate virsh command 129, 283
 - auto-converge option 129
 - timeout option 129
- migrate-getspeed virsh command 129, 286
- migrate-setmaxdowntime virsh command 129

- migrate-setspeed virsh command 129, 288
 - bandwidth option 129, 288
- migrated reason
 - of the running state 179
- migrating
 - CPUs 153
 - image files 124
 - running virtual servers 123, 129
 - storage 9, 14
 - virtual servers 121
- migrating reason
 - of the paused state 180
- migration 153
 - of a running virtual server to another host 76, 88
 - See live migration
 - of the storage server 9, 14, 76
 - of virtual disks
 - See disk migration
 - preparing virtual servers for 9, 14, 29
 - to a different hypervisor release 122, 123
- migration costs 153
- migration port range 127
- migration_port_max parameter 127
- mode attribute 207
 - of the hostdev element 88, 207
 - of the source element 96, 237
 - subsystem value 207
- model attribute
 - of the backend element 189
 - of the controller element 86, 194
 - of the memballoon element 71, 216
 - of the rng element 230
 - of the watchdog element 247
- model element 74, 96, 98, 220
 - type attribute 96, 98, 220
 - virtio value 96, 98
- modifying
 - persistent virtual server definitions 106
 - the CPU weight 137
 - the number of virtual CPUs 134
 - virtual server definitions 105
- modprobe command 40
- monitoring
 - virtual servers 115
- multipath command 29, 76
- multipath device mapper support 9, 14, 29
 - configuring 29
 - preparing 29
- multipathed SCSI device configuration
 - example 91

N

- N_Port ID virtualization 29
- name attribute
 - of the adapter element 88, 185
 - of the cipher element 69, 191
 - of the driver element 76, 82, 84, 93, 201
 - of the host element 206
- name element 51, 221
- name property
 - of virtual servers 4, 49
- native value
 - of the driver io attribute 76, 82, 84, 93, 201
- NDP 209
- Neighbor Discovery Protocol 209
- network device 21, 38, 73

- network device (*continued*)
 - attaching 142
 - detaching 143
 - hotplugging 142
 - preparing 37
 - unplugging 143
- network device driver 4
- network file system 124
- network interface
 - activating 38
 - configuring 96
 - creating 38
 - displaying 116
 - preparing 38
 - setting up 37, 38
- network isolation 21
- NFS
 - See network file system
- NIC 98
- no value
 - of the hostdev rawio attribute 207
 - of the interface trustGuestRxFilters attribute 209
- node 3
- non-tunneled migration 127
- none value
 - of the driver cache attribute 76, 82, 84, 93, 201
 - of the geometry trans attribute 205
 - of the memballoon model attribute 71, 216
- NPV 29
- number of virtual CPUs 134

O

- off value
 - of the cipher state attribute 69, 191
 - of the driver event_idx attribute 201
 - of the driver ioeventfd attribute 201
 - of the log append attribute 67, 214
 - of the memory dumpCore attribute 64
- on value
 - of the cipher state attribute 69, 191
 - of the driver event_idx attribute 201
 - of the driver ioeventfd attribute 201
 - of the log append attribute 67, 214
- on_crash element 51, 222
- on_poweroff element 51
- on_reboot element 51, 223
- Open vSwitch 21, 43
 - package 43
- openvswitch command
 - start 43
 - status 43
- openvswitch value
 - of the virtualport type attribute 98, 245
- optional value
 - of the source startupPolicy attribute 234
- order attribute
 - of the boot element 53, 190
- os element 51, 55, 224
- 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

ovs-vsctl command (*continued*)
show 43

P

para-virtualized device driver 4
parameter file 55
partition 9, 14
path attribute
 of the device element 196
path element 225, 226
path redundancy
 of DASDs 9, 14
 of network devices 21
 of physical volumes 161
 of SCSI disks 9, 14
 of SCSI medium changer devices 17
 of SCSI tape devices 17
paused state 4, 177
 migrating reason 180
 user reason 180
perf kvm stat command
 live subcommand 171
 record subcommand 171
 report subcommand 171
perf tool 171
perf.data.guest 171
performance considerations 127
performance metrics
 collecting 171
 displaying 171
 recording 171
performing
 a live migration 129
persistent device
 configuring 66
physical volume
 configuration 161
 filter 161
pool attribute
 of the source element 234
pool element 227
 type attribute 227
pool-autostart virsh command 289
pool-define virsh command 290
pool-delete virsh command 291
pool-destroy virsh command 292
pool-dumpxml virsh command 293
pool-edit virsh command 294
pool-info virsh command 295
pool-list virsh command 148, 296
pool-name virsh command 297
pool-refresh virsh command 298
pool-start virsh command 299
pool-undefine virsh command 300
pool-uuid virsh command 301
port attribute
 of the target element 239
port range for migration 127
ports attribute
 of the controller element 194
preparing
 bonded interfaces 40
 DASDs 27
 devices for virtual servers 4
 host devices 27
 MacVTap interfaces 40

preparing (*continued*)
 multipath device mapper support 29
 network devices 37
 network interfaces 40
 physical volumes 161
 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, 222
 of the on_reboot element 223
preserve value
 of the on_crash element 51
primary bridge port 43
property
 name 49
protected key 69
 management operations 69
providing
 I/O threads 66, 142
 ISO images 143
pty value
 of the console type attribute 67, 193
pvscan command 161

Q

qcow2 image file 82, 84
qcow2 value
 of the driver type attribute 82, 84
 of the format type attribute 204
QEMU 6, 9, 14, 33
QEMU command
 info 323
 qemu-img create 82, 84
QEMU core dump
 configuring 64
qemu value
 of the driver name attribute 76, 82, 84, 93, 201
qemu-kvm command 122, 123
qemu-system-s390x user space process 65
qethconf 43

R

ramdisk 53, 55
random number generator 73
 configuring 100
random value
 of the backend model attribute 189
raw image file 82, 84
raw value
 of the driver type attribute 76, 82, 84, 93, 201
 of the format type attribute 204
rawio attribute 207
 no value 207
 of the hostdev element 207
 yes value 207
readonly element 93, 229
reboot virsh command 302

- record subcommand
 - or perf kvm stat 171
- recording
 - performance metrics 171
- redundant paths 9, 14, 17, 21
- relocating
 - virtual servers
 - See live migration
- relocation
 - See live migration
- removable ISO image
 - configuring 93
 - removing 143
 - replacing 143
- removing
 - I/O threads 143
 - ISO images 143
- replacing
 - ISO images 143
- report subcommand
 - or perf kvm stat 171
- report value
 - of the driver error_policy attribute 201
 - of the driver rerror_policy attribute 201
- requisite value
 - of the source startupPolicy attribute 234
- error_policy attribute
 - of the driver element 201
- restart text content
 - of the on_reboot element 223
- restored reason
 - of the running state 179
- resume virsh command 112, 303
- resuming
 - virtual servers 4, 112
- retrieving hardware information
 - as a virtual server user 325
- RNG device 73, 100
- rng element 230
 - model attribute 230
- root file system 53, 55
- root path 53, 55
- run queue 153
- running state 4, 177
 - booted reason 179
 - migrated reason 179
 - restored reason 179
 - unpaused reason 179

S

- s390-ccw-virtio value
 - of the type machine attribute 51, 122, 123, 243
- s390x value
 - of the type arch attribute 51, 243
- SAM 88
- SAN fabric 9, 14
- saved reason
 - of the shut off state 178
- saved system image 110
- saving
 - system images 110
- schedinfo virsh command 116, 137, 304
 - config option 137, 304
 - live option 137, 304
 - cpu_shares attribute 137, 304
- scheduling domain 153

- scheduling information
 - displaying 116
- sclp value
 - of the target type attribute 67, 239
- SCSI Architecture Model 88
- SCSI device driver 4
- SCSI device name 17, 33, 88
- SCSI disk 9, 14, 76
 - configuring 76
 - preparing 29
 - setting up 29
- SCSI disk configuration
 - example 80
- SCSI generic device driver 17
- SCSI host 88
- SCSI Host Bus Adapter 73
- SCSI host number 17, 88
- SCSI ID 17, 88
- SCSI identifier 9, 14
- SCSI LUN 17, 88
- SCSI medium changer 17
 - configuring 86, 88
 - preparing 33
 - setting up 33
- SCSI stack 33
- SCSI stack address 17
- SCSI tape 17
 - configuring 86, 88
 - preparing 33
 - setting up 33
- scsi value
 - of the address type attribute 88, 187
 - of the controller type attribute 86, 194
 - of the hostdev type attribute 88, 207
 - of the target bus attribute 93
- secs attribute
 - of the geometry element 205
- Security-Enhanced Linux 43
- SELinux 43
- serial device 73
- setting up
 - bonded interfaces 40
 - host devices 27
 - MacVTap interfaces 40
 - network interfaces 37, 40
 - the host for live migration 127
 - virtual LAN interfaces 40
 - virtual switches 43
 - VLAN interfaces 40
- setvcpus command
 - setvcpus 306
- setvcpus virsh command 134, 306
- sgio attribute 207
 - filtered value 207
 - of the hostdev element 207
 - unfiltered value 207
- shareable element 231
- shares element 51, 61, 232
- shut off state 4, 107, 177
 - destroyed reason 178
 - saved reason 178
 - shutdown reason 178
 - unknown reason 178
- shutdown reason
 - of the shut off state 178
- shutdown virsh command 110, 305

- shutting down
 - virtual servers 4, 110
- shutting down state 177
- slave device 40
- soft_limit element 62, 233
 - unit attribute 233
- source element 74, 76, 82, 84, 93, 96, 98, 234, 236, 237, 238
 - bridge attribute 98
 - dev attribute 76, 82, 84, 96, 234, 237
 - file attribute 82, 84, 93, 234
 - mode attribute 96, 237
 - bridge value 96
 - pool attribute 234
 - startupPolicy attribute 234
 - mandatory value 234
 - optional value 234
 - requisite value 234
 - volume attribute 234
- SSH daemon 127
 - configuration file 127
- ssid attribute
 - of the address element 76, 82, 84, 86, 186
- standard device name 9, 14
 - of a SCSI medium changer 17
 - of a SCSI tape 17
- standard device node 9, 14, 76
- standard interface name 37
- start openvswitch command 43
- start virsh command 110, 145, 308
 - console option 110, 145
 - force-boot option 110
- starting
 - libvirt daemon 109, 141
 - virtual servers 4, 110
- startupPolicy attribute
 - of the source element 234
- state 4, 177
 - crashed 177, 181
 - displaying 116
 - managesave 110, 177
 - paused 4, 177, 180
 - running 4, 177, 179
 - shut off 4, 107, 177, 178
 - shutting down 177
- state attribute
 - of the cipher element 69, 191
- state-transition diagram 177
 - simplified 4
- status openvswitch command 43
- STHYI instruction 325
- stop value
 - of the driver error_policy attribute 201
 - of the driver rerror_policy attribute 201
- stopped phase of a migration 128
- storage controller 9, 14
 - port 9, 14
- storage keys 128
- storage migration 76, 161
- storage pool
 - configuring 101
 - managing 148
- storage pool configuration-XML file 101
- Store Hypervisor Information instruction 325
- subchannel set-ID 9, 14, 76, 82, 84
- subsystem value
 - of the hostdev mode attribute 88, 207
- suspend virsh command 112, 310

- suspending
 - virtual servers 4, 112
- symmetric encryption 69
- sysfs attribute
 - bridge_state 43
- system image 280
 - saved 110
 - saving 110
- system journal 167
- system resources
 - configuring 49
 - managing 133

T

- tape 73
- target attribute
 - of the address element 88, 93, 187, 188
- target element 67, 74, 76, 82, 84, 93, 239, 240, 241, 242
 - address attribute 239
 - bus attribute 76, 82, 84, 240
 - scsi value 93
 - virtio value 76, 82, 84, 240
 - dev attribute 76, 82, 84, 93, 240
 - port attribute 239
 - type attribute 67, 239
 - scslp value 67, 239
 - virtio value 67, 239
- TDEA 69
- TDES 69
- terminating
 - virtual servers 4, 110
- threads value
 - of the driver io attribute 201
- topology 9, 14
- trans attribute
 - of the geometry element 205
- Triple DEA 69
- Triple DES 69
- trustGuestRxFilters attribute
 - of the interface element 96, 209
- tuning
 - virtual CPUs 61
 - virtual memory 62
- type
 - of the virtual channel path 73
- type attribute 207
 - of the address element 76, 82, 84, 86, 88, 93, 186, 187
 - of the console element 67, 193
 - of the controller element 86, 194
 - of the disk element 76, 82, 84, 93, 198
 - of the domain element 51, 199
 - of the driver element 76, 82, 84, 93, 201
 - of the format element 204
 - of the hostdev element 88, 207
 - of the interface element 96, 98, 209
 - of the model element 96, 98, 220
 - of the pool element 227
 - of the target element 67, 239
 - of the virtualport element 98, 245
 - of the volume element 246
 - scsi value 187, 194, 207
 - virtio-serial value 194
- type element 51, 243
 - arch attribute 243
 - machine attribute 243

U

- udev-created by-path device node 76
- udev-created device node 9, 14, 76
- UID 9, 14
- undefine virsh command 107, 311
- undefined virtual server 4
- undefining
 - virtual servers 4, 107
- unfiltered value
 - of the hostdev sgio attribute 207
- unique ID 9, 14
- unit 33
- unit attribute
 - of the address element 88, 93, 187, 188
 - of the memory element 62
 - of the soft_limit element 233
- universally unique identifier 9, 14
- unknown reason
 - of the shut off state 178
- unpaused reason
 - of the running state 179
- unplugging
 - devices 143
- unsafe value
 - of the driver cache attribute 201
- uplink port
 - creating 43
- user reason
 - of the paused state 180
- user space 49
 - configuring 65
- user space process 200, 201
- user-friendly name 29
- UUID 9, 14

V

- vcpu element 51, 60, 244
 - current attribute 244
- vcpucount virsh command 116, 134, 312
 - active option 312
 - config option 312
 - current option 312
 - live option 312
 - maximum option 312
- vectors attribute
 - of the controller element 194
- verifying
 - a live migration 129
- virsh command 4
 - attach-device 142, 252
 - change-media 143, 254
 - console 145, 256
 - define 106, 257
 - destroy 110, 258
 - detach-device 143, 259
 - dombklist 116, 261
 - dombkstat 116, 262
 - domiflist 116, 264
 - domifstat 116, 265
 - dominfo 116, 266
 - domjobabort 129, 267
 - domstate 116, 268
 - dump 169, 269
 - dumpxml 86, 118, 270
 - edit 106, 271

- virsh command (*continued*)
 - inject-nmi 169, 272
 - iothreadadd 273
 - iothreaddel 275
 - iothreadinfo 277
 - list 107, 116, 141, 145, 278
 - makedumpfile 169
 - managedsave 110, 280
 - memtune 282
 - migrate 129
 - migrate-getspeed 129, 286
 - migrate-setmaxdowntime 129
 - migrate-setspeed 129, 288
 - pool-autostart 289
 - pool-define 290
 - pool-delete 291
 - pool-destroy 292
 - pool-dumpxml 293
 - pool-edit 294
 - pool-info 295
 - pool-list 148, 296
 - pool-name 297
 - pool-refresh 298
 - pool-start 299
 - pool-undefine 300
 - pool-uuid 301
 - reboot 302
 - resume 112, 303
 - schedinfo 116, 137, 304
 - setvcpus 134
 - shutdown 110, 305
 - start 110, 145, 308
 - suspend 112, 310
 - undefine 107, 311
 - vcpucount 116, 134, 312
 - vol-create 313
 - vol-delete 314
 - vol-dumpxml 315
 - vol-info 316
 - vol-key 317
 - vol-list 149, 318
 - vol-name 319
 - vol-path 320
 - vol-pool 321
- virsh command option
 - all 278
 - autodestroy 308
 - autostart 278
 - build 299
 - bypass-cache 280, 308
 - config 137, 252, 254, 259, 304
 - console 308
 - current 254, 259
 - disable 289
 - domain 252, 254, 258, 259, 270, 305, 308
 - eject 254
 - file 252, 259
 - force 254, 256
 - force-boot 308
 - graceful 258
 - id 278
 - inactive 270, 278
 - insert 254
 - live 137, 254, 259, 304
 - managed-save 278
 - memory-only 269, 282
 - migratable 270

virsh command option (*continued*)

- mode 305
- name 278
- no-autostart 278
- no-overwrite 299
- overwrite 299
- path 254
- paused 280, 308
- persistent 259, 278
- reason 116, 268
- running 280
- safe 256
- security-info 270
- state-other 278
- state-paused 278
- state-running 278
- state-shutoff 278
- table 278
- title 278
- transient 278
- update 254
- update-cpu 270
- uuid 278
- verbose 280
- with-managed-save 278
- with-snapshot 278
- without-managed-save 278
- without-snapshot 278

virsh command-line interface 6

virtio 4

- block device 9, 14, 76, 82, 84
- block device driver 4
- device driver 4
- network device driver 4

virtio value

- of the model type attribute 96, 98
- of the target bus attribute 76, 82, 84, 240
- of the target type attribute 67, 239

virtio-block device 73

virtio-net device 73

virtio-scsi device 73

virtio-scsi value

- of the controller model attribute 86, 194

virtio-serial value

- of the controller type attribute 194

virtual block device 9, 14, 76

- attaching 142
- detaching 143
- device configuration-XML 74
- hotplugging 142
- unplugging 143

virtual channel 76, 82, 84

virtual channel path 9, 14, 73

virtual channel path type 73

virtual channel subsystem 73

virtual channel subsystem-ID 73

virtual control unit model 73

virtual CPU 153

- configuring 60
- configuring the number 60
- Linux management of 153
- managing 134
- modifying the weight 137
- tuning 61

virtual CPUs

- actual number 134
- current config 134

virtual CPUs (*continued*)

- maximum config 134
- maximum live 134
- maximum number 134

virtual DVD 17, 93, 143

virtual DVD drive 17, 143

virtual Ethernet device 21

- attaching 142
- detaching 143
- device configuration-XML 74
- hotplugging 142
- unplugging 143

virtual Ethernet interface

- preparing 37

virtual HBA 73

- attaching 142
- configuring 86
- detaching 143
- device configuration-XML 74
- hotplugging 142
- unplugging 143

virtual Host Bus Adapter

- configuring 86
- device configuration-XML 74

virtual LAN interface 21, 40

virtual machine relocation

- See* live migration

virtual memory 62

- configuring 62
- managing 139
- tuning 62

virtual SCSI device 17, 86

- attaching 142
- detaching 143
- device configuration-XML 74
- hotplugging 142
- unplugging 143

virtual SCSI-attached CD/DVD drive 93

virtual server x, 3, 4

- browsing 116
- configuration 9, 14
- configuring 4
- crashed 169
- defining 4, 106
- destroying 110
- devices 49
- displaying block devices 116
- displaying information 116
- displaying network interfaces 116
- displaying scheduling information 116
- displaying the state 116
- dumping 169
- kernel panic 169
- managing 109
- migrating 121, 123
- monitoring 115
- name 4, 49, 51
- persistent definition
 - creating 105
 - defining 105
 - deleting 105
 - editing 106
 - modifying 105, 106
- preparing 4
- properties 49
- relocation
 - See* live migration

- virtual server *(continued)*
 - resuming 4, 112
 - shutting down 4, 110
 - starting 4, 110
 - state 4, 177
 - crashed 177, 181
 - paused 4, 177, 180
 - running 4, 177, 179
 - shut off 4, 177, 178
 - shutting down 177
 - stopping 4
 - suspending 4, 112
 - system resources 49
 - terminating 110
 - undefining 4, 107
- virtual switch 21
 - configuring 96, 98
 - 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 98, 245
 - type attribute 98, 245
 - openvswitch value 98, 245
- VLAN ID 40
- VLAN interface 40
- vol-create virsh command 313
- vol-delete virsh command 314
- vol-dumpxml virsh command 315
- vol-info virsh command 316
- vol-key virsh command 317
- vol-list virsh command 149, 318
- vol-name virsh command 319
- vol-path virsh command 320
- vol-pool virsh command 321
- volume
 - managing 149
- volume attribute
 - of the source element 234
- volume configuration-XML file 101
- volume element 246
 - type attribute 246
- VXLAN tunnel 43

W

- watchdog device
 - configuring 68
- watchdog device driver 171
- watchdog element 247
 - action attribute 247
 - model attribute 247
- watchdog timer 68
- weight-fraction 154
- workload
 - memory intensive 128, 129
- worldwide port name 9, 14
- wrapping key 69
- writeback value
 - of the driver cache attribute 201

- writethrough value
 - of the driver cache attribute 82, 84, 201
- WWPN 9, 14, 29

X

- XML format 6

Y

- yes value
 - of the hostdev rawio attribute 207
 - of the interface trustGuestRxFilters attribute 209

Z

- zipl
 - command 53, 55
 - configuration file 53, 55
- znetconf command 37, 38, 43, 96

Readers' Comments — We'd Like to Hear from You

Linux on z Systems and LinuxONE
KVM Virtual Server Management
March 2017

Publication No. SC34-2752-03

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



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



SC34-2752-03

