Linux on Z and LinuxONE

KVM Virtual Server Management
October 2021

IBM

Note

Before using this document, be sure to read the information in "Notices" on page 451.

Contents

Summary of changes

This revision reflects changes for the October 2021 edition.

Updates for the October 2021 edition

The October 2021 edition (SC34-2752-07) contains changes compared to the previous SC34-2752-06 edition.

New information

- You can now use **virsh** commands to manage VFIO mediated devices, see "Managing mediated devices for DASD with libvirt" on page 64.
- You can now use the launchSecurity element of the domain configuration-XML to prepare virtual servers for guests in IBM® Secure Execution mode, see Chapter 16, "Configuring for IBM Secure Execution for Linux," on page 141.
- A new topic summarizes how you can optimize the performance of your virtual servers, see Chapter 30, "Performance hints and tips summary," on page 211.

Changed Information

- None.

This revision also includes maintenance and editorial changes.

Deleted Information

- None.

Updates for the December 2020 edition

The December 2020 edition (SC34-2752-06) contains changes compared to the previous SC34-2752-05 edition.

New information

- You can now use PCIe-attached NVMe devices on the host to back virtio block devices on KVM virtual servers, see "NVMe devices" on page 16.
- You can set up KVM virtual servers for guests in IBM Secure Execution mode, see Chapter 5, "IBM Secure Execution for Linux," on page 27.
- The **virsh hypervisor-cpu-compare** and **virsh hypervisor-cpu-baseline** commands help you to investigate CPU model support of KVM hosts, see "IBM Z hardware CPU model" on page 164.
- With **virt-xml**, you can now start a KVM virtual server from a temporary boot device without editing the domain configuration-XML, see Chapter 26, "Booting from a temporary boot device," on page 195.

Changed Information

- None.

This revision also includes maintenance and editorial changes.

Deleted Information

- None.

Updates for the November 2019 edition

The November 2019 edition (SC34-2752-05) contains changes compared to the previous SC34-2752-04 edition.

New information

- HiperSockets and OSA devices that connect a virtual switch to a network can now be configured with VNIC characteristics, see "Preparing a virtual switch" on page 49.

- You can now configure VFIO pass-through devices for PCI, DASD, and for cryptographic adapter resources, see Chapter 11, "Preparing VFIO pass-through devices," on page 55.

- VFIO pass-through DASD are supported as boot devices, see "Configuring the boot process" on page 77.

- You can now configure virtual server memory to be backed by huge pages, see "Configuring huge pages" on page 91.

- A new libvirt device type provides virtual graphics cards for remote access from workstations to guest applications that require user interaction through a graphical user interface, see "Configuring a virtual graphics card" on page 113.

- The **virt-install** command helps you to set up virtual servers, see Chapter 25, "Fast path to a running guest - virt-install," on page 193.

- For specific causes of a crash, the QEMU log file of a virtual server now contains diagnostic information about the crash, see Chapter 33, "Finding virtual server crash information," on page 221.

Changed Information

- None.

This revision also includes maintenance and editorial changes.

Deleted Information

- None.

About this document

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

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

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

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

KVM users familiar with other platforms should be aware that:

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

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

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

How this document is organized

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

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

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

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

Part five provides performance-related information for KVM virtual servers and describes tools that can help you to efficiently manage KVM virtual servers.

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

Conventions and assumptions used in this publication

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

Authority

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

Persistent configuration

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

Terminology

This document uses the following terminology:

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

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

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

Highlighting

This publication uses the following highlighting styles:

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

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

Where to get more information

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

Kernel based virtual machine (KVM)

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

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 and LinuxONE

You can find publications for Linux on Z and LinuxONE on IBM Documentation.

These publications are available on IBM Documentation at
ibm.com/docs/en/linux-on-systems?topic=linuxone-library-overview

- *Device Drivers, Features, and Commands*
- *Using the Dump Tools*
- *KVM Virtual Server Management*, SC34-2752
- *Configuring Crypto Express Adapters for KVM Guests*, SC34-7717
- *How to use FC-attached SCSI devices with Linux on z Systems®*, SC33-8413
- *Introducing IBM Secure Execution for Linux*, SC34-7721
- *openCryptoki - An Open Source Implementation of PKCS #11*, SC34-7730
- *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
- *Pervasive Encryption for Data Volumes*, SC34-2782
- *Enterprise Key Management for Pervasive Encryption of Data Volumes*, SC34-7740
- *How to set an AES master key*, SC34-7712
- *Troubleshooting*, SC34-2612
- *Kernel Messages*, SC34-2599
- *How to Improve Performance with PAV*, SC33-8414
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596

Part 1. General concepts

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

Chapter 1. Overview

Set up, configure, and manage the operation of virtual servers.
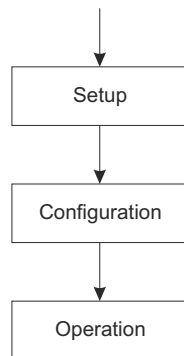
Figure 1. Virtual server administrator's tasks

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

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

The *KVM host* is the Linux instance that runs the KVM virtual servers and manages their resources. In the libvirt documentation, a host is also called a *node*.
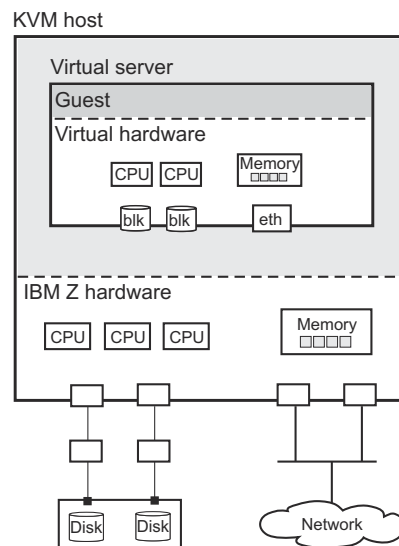
Figure 2. KVM host with a virtual server including a guest operating system

Virtual server management tasks

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

1. Device setup

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

   For a detailed description of this task, see Part 2, "Device setup," on page 29.

2. Virtual server and device configuration

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

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

   For a detailed description of this task, see Part 3, "Configuration," on page 71.

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 on page 4).

      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.
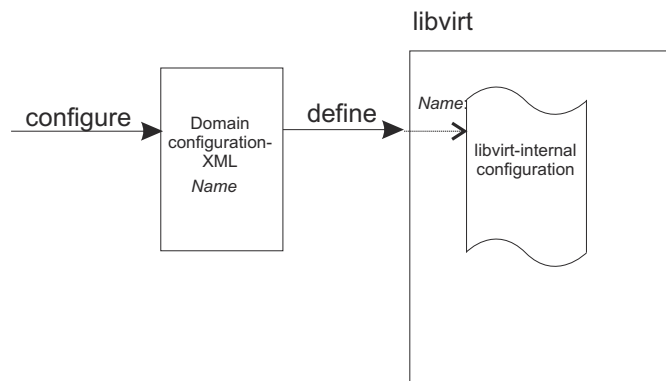
Figure 3. Creating a persistent virtual server definition

   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 on page 5).

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

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

For a detailed description of these tasks, see Part 4, "Operation," on page 145.
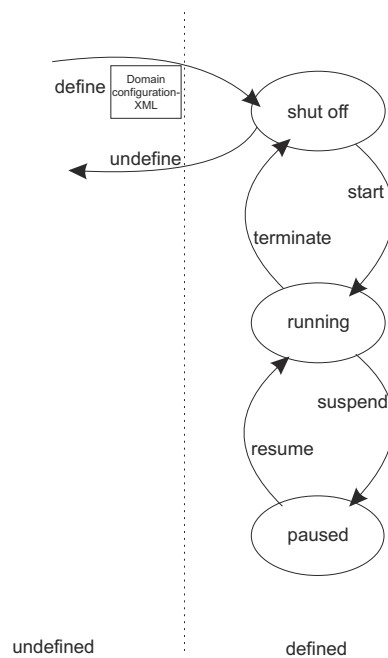
Figure 4. Simplified state-transition diagram of a virtual server

Related concepts

"Fast path to a running guest - virt-install" on page 193
With a single **virt-install** command, configure and define a virtual server, and install and run a guest.

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.

Device
setup
on the host
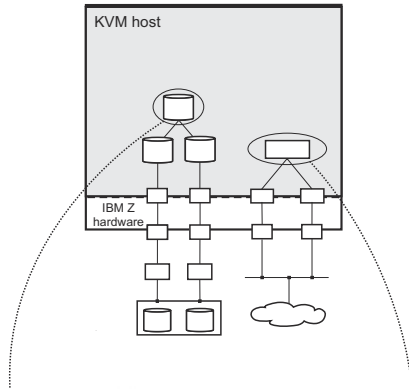
Configuration
using XML format
including the prepared
device and network
interface names

Operation
using virsh:
Create a persistent
virtual server
definition
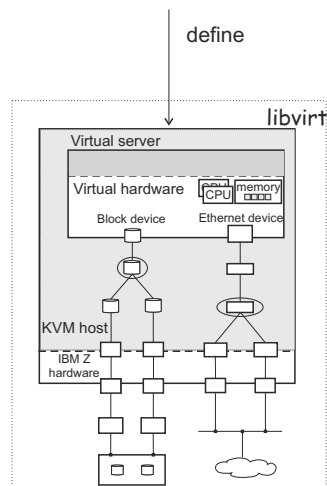
Operation
using virsh:
Manage the
virtual server
life cycle
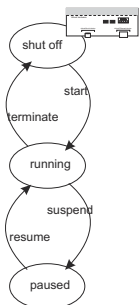
Figure 5. Virtual server administrator tasks using XML format and the virsh command-line interface

Device virtualization techniques

KVM on Z offers two major techniques to virtualize devices for its guests: virtio and VFIO.

Virtio devices

On Z, the virtio framework virtualizes devices as virtio CCW devices. KVM guests access virtio CCW devices through a virtual z/Architecture channel subsystem.

Virtio CCW devices can be paravirtualized host devices that resemble generic devices, as do paravirtualized devices on other architectures. Many of the characteristics of paravirtualized host devices are hidden from the guests.

The virtio CCW devices also include devices that are provided by QEMU and that are not based on physical host devices.

A special type of virtio CCW devices are virtual SCSI Host Bus Adapters (HBAs), which provide virtual SCSI devices to KVM guests. Virtual SCSI devices include SCSI pass-through devices, which are based on SCSI devices on the host.

VFIO pass-through devices

The VFIO framework can give guests direct access to specific host devices.

On the host, these VFIO pass-through devices are set up to be controlled by device-specific VFIO device drivers instead of their default device drivers. Depending on the device type, VFIO pass-through devices might require a VFIO mediated device that is based on host resources.

Virtual SCSI devices

Virtual SCSI host bus adapters (HBAs) enable guests to access virtual SCSI devices within virtual servers. Virtual SCSI HBAs are virtio devices that are provided by the KVM hypervisor and do not map to physical FCP devices on the host.

Virtual SCSI devices, in contrast, can map to SCSI devices or other resources on the host. Figure 6 on page 9 illustrates two common mappings for virtual SCSI devices:

- A LUN for a SCSI-attached tape drive on the host.
- A DVD ISO file on the host file system.
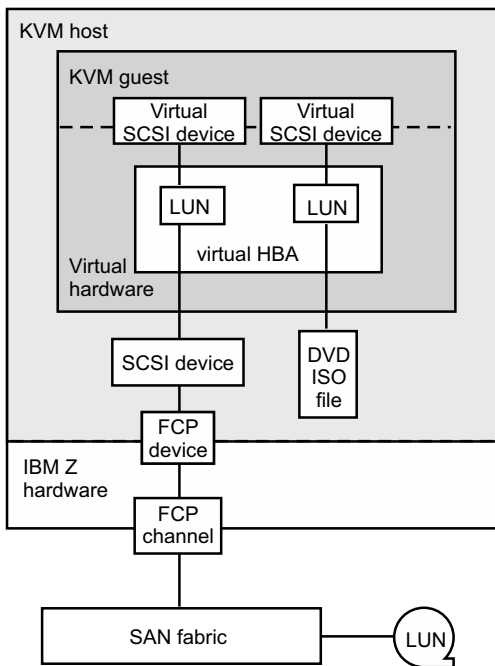
Figure 6. Examples of host backing for virtual SCSI devices

In this publication, a virtual SCSI device on the guest that is backed by a SCSI device on the host is called a SCSI pass-through device. Do not confuse SCSI pass-through with VFIO pass-through. With SCSI pass-through, the SCSI device on the host and the virtual SCSI device on the guest are different devices.

Configure virtual SCSI devices only if you need to address the device through SCSI-specific interfaces. In particular, configure SCSI disks as virtual block devices rather than virtual SCSI devices.

Related concepts
"SCSI tapes and medium changers as virtual SCSI devices" on page 19
FC-attached SCSI tape and medium changer devices are virtualized as virtio SCSI devices.

Related tasks
"Preparing SCSI tape and medium changer devices" on page 37
Consider these aspects when setting up FC-attached SCSI tapes and SCSI medium changers for the use of a virtual server.

"Preparing SCSI disks" on page 33
Consider these aspects when setting up FC-attached SCSI disks for the use of a virtual server.

"Configuring virtual SCSI devices" on page 115
Configure SCSI tape devices, SCSI medium changer devices, and DVD drives as virtual SCSI devices for a virtual server.

Chapter 2. Virtual block devices

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

Related publications

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

DASDs and SCSI disks

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

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

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

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

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

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

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

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

    See Chapter 6, "Preparing DASDs," on page 31 and Chapter 7, "Preparing SCSI disks," on page 33.

2. Configure the DASDs and FC-attached SCSI disks as virtual block devices.

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

    See Chapter 13, "Configuring devices," on page 101 and "Configuring virtual block devices" on page 103.

DASD and SCSI disk configuration topology

shows how multipathed DASD and SCSI disks are configured as virtual block devices.
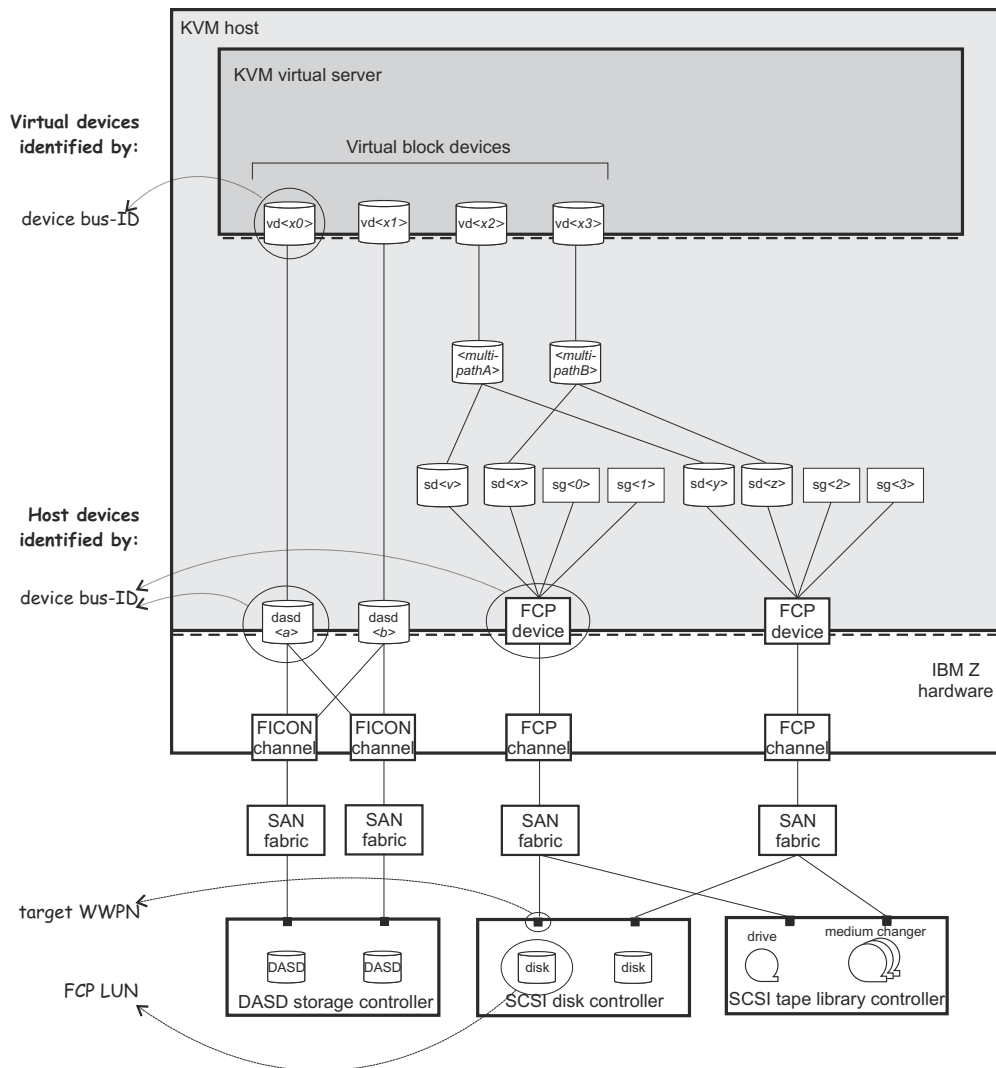
Figure 7. Multipathed DASD and SCSI disks configured as virtual block devices

Disk device identification

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

Device bus-ID and device number of an FCP device

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

```
0.m.dddd
```

Where:

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

Example:	
0.0.1700	device bus-ID of the FCP device.
1700	device number of the FCP device.

Device bus-ID and device number of a DASD

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

```
0.m.dddd
```

Example:

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

Unique ID (UID) of a DASD

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

Example:

```
IBM.75000000010671.5600.00
```

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

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

```
0.m.dddd
```

Where:

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

Example:

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

Standard device name

Standard device names are of the form:

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

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

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

Example:

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

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

Standard device node

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

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

Example:

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

udev-created device node

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

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

Example for DASDs on the host:

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

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

where:

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

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

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

where:

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

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

where:

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

Example for SCSI devices on the host:

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

where:

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

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

> where:
>
> ```
> scsi-36005076303ffc56200000000000010ea
> ```
> is the SCSI identifier.

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

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

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

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

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

Virtual server example:

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

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

> where:
>
> | | |
> |---|---|
> | `0.0.1a12` | is the device bus-ID. |
> | `part1` | denotes the first partition of the device. |

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

Example:

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

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

> where
>
> | | |
> |---|---|
> | `p1` | denotes the first partition of the device. |

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

NVMe devices

PCIe-attached NVMe devices are available for IBM LinuxONE as of LinuxONE II. NVMe devices can be virtualized as virtio block devices.

To provide NVMe devices as virtual block devices to a virtual server:

1. Set up the NVMe devices, see Chapter 9, "Preparing NVMe devices," on page 41.

   Provide device nodes that persist across host reboots.

   Standard device nodes of the form `/dev/nvme<i>n<n>` do not persist across host reboots. Persistent device nodes, typically, include information that identifies the physical resources for the device. Depending on your host setup, udev might create suitable nodes that include the PCI function address of the NVMe device, for example `/dev/disk/by-path/pci-<function_address>-nvme-<n>`.

   For live migration of a virtual server, device nodes must be such that they addresses equivalent resources on both the source and destination host. Migration of the virtio block device is then possible through *disk migration*, see step "3.c" on page 171 in "Performing a live migration" on page 170.

2. Configure the NVMe devices as virtual block devices.

   Configure devices that are to be defined with the virtual server in its domain configuration-XML file. Alternatively, you can define devices in a separate device configuration-XML file. Such devices can be attached to an already defined virtual server.

   See "Configuring virtual block devices" on page 103 and "Device configuration-XML" on page 133.

The following graphic illustrates PCIe-attached NVMe devices on the host and the corresponding block devices on the guest.
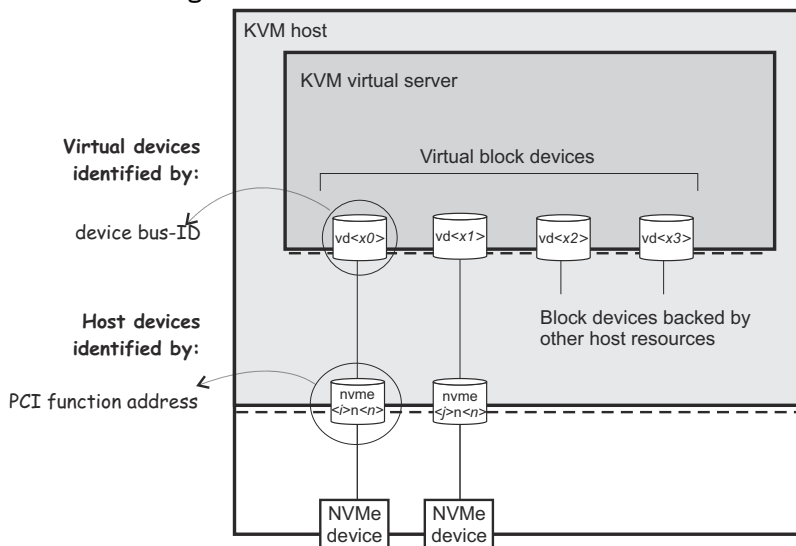
Figure 8. NVMe devices configured as virtual block devices

Image files and logical volumes

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

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

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

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

   See Chapter 13, "Configuring devices," on page 101 and "Configuring an image file as storage device" on page 109.

Storage pools

Alternatively, you can configure *storage pools*, leaving the resource management of step "1" on page 17 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 9 on page 17 shows a storage pool backed by the image files of a directory:
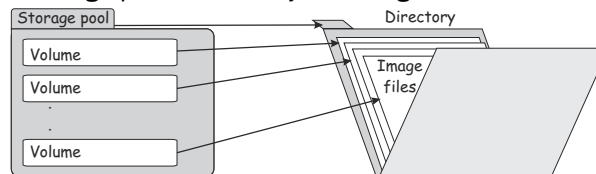
Figure 9. Storage pool backed by the image files of a directory

Figure 10 on page 17 shows a storage pool backed by the logical volumes of a volume group:
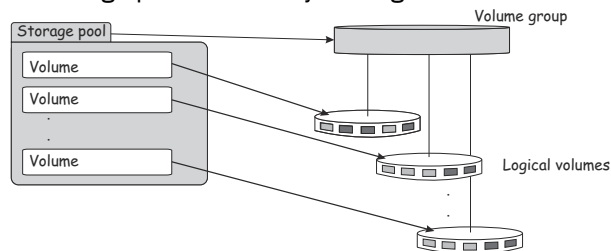
Figure 10. Storage pool backed by the logical volumes of a volume group

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

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

   See Chapter 14, "Configuring storage pools," on page 137.
4. Configure volumes as virtual storage devices for the virtual server.

See "Configuring a volume as storage device" on page 111.

Figure 11 on page 18 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:
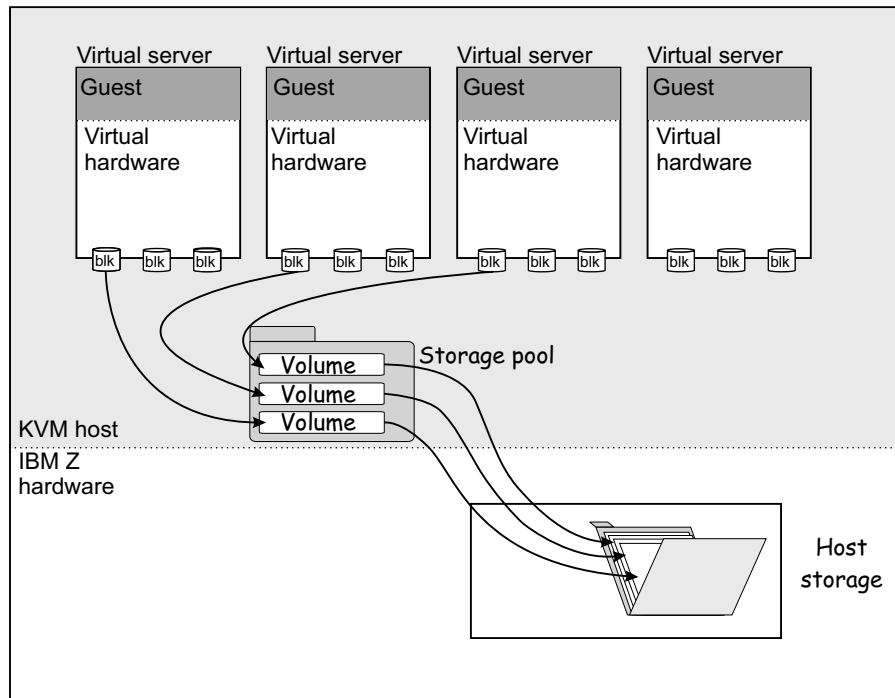
Figure 11. Storage pool volumes configured as virtual block devices

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 23, "Managing storage pools ," on page 187.

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 12 on page 20 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 8, "Preparing SCSI tape and medium changer devices," on page 37.

2. Configure the SCSI tape or medium changer device in separate device configuration-XML files.

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

   See Chapter 13, "Configuring devices," on page 101 and "Configuring virtual SCSI devices" on page 115.

Virtual SCSI device configuration topology

Figure 12 on page 20 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.
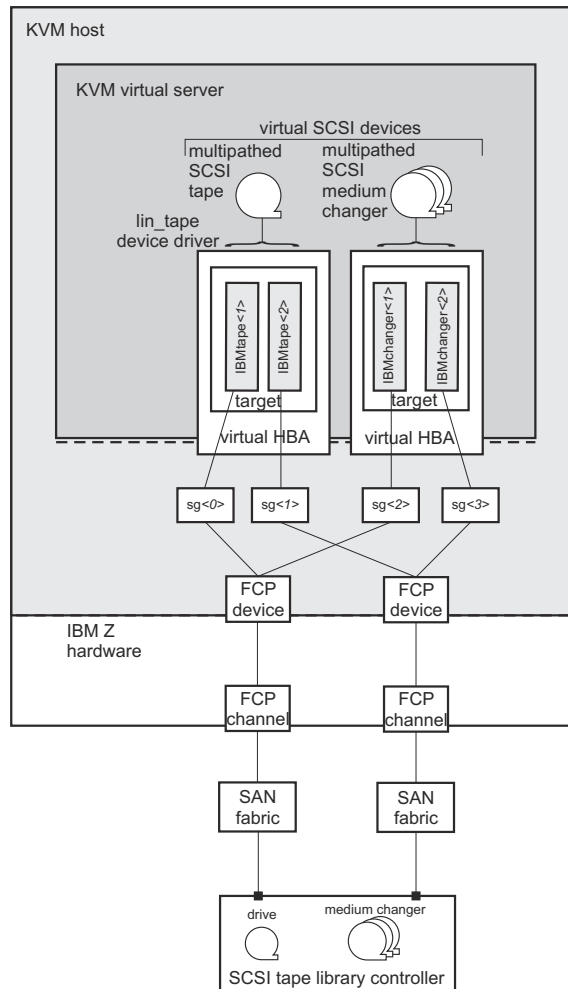
Figure 12. Multipathed SCSI tapes and SCSI medium changer devices configured as virtual SCSI devices

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

SCSI device identification

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

Standard device name

Standard device names are of the form:

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

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

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

SCSI device name

SCSI device names are of the form:

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

Where:

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

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

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

Example:	`0:0:1:7`

Related publication

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

Related concepts
"Virtual SCSI devices" on page 9

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 13 on page 23:
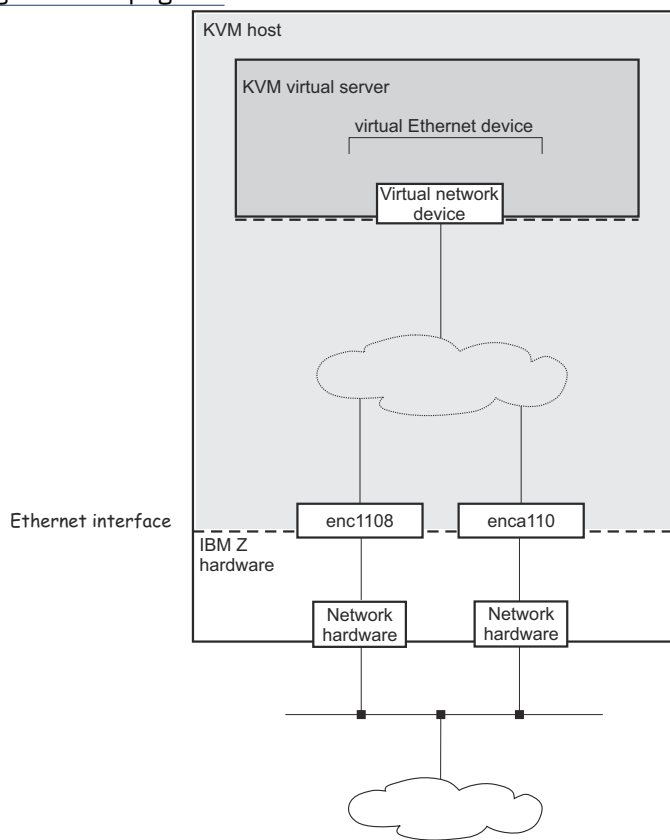
Figure 13. Highly reliable virtual network device configuration

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.
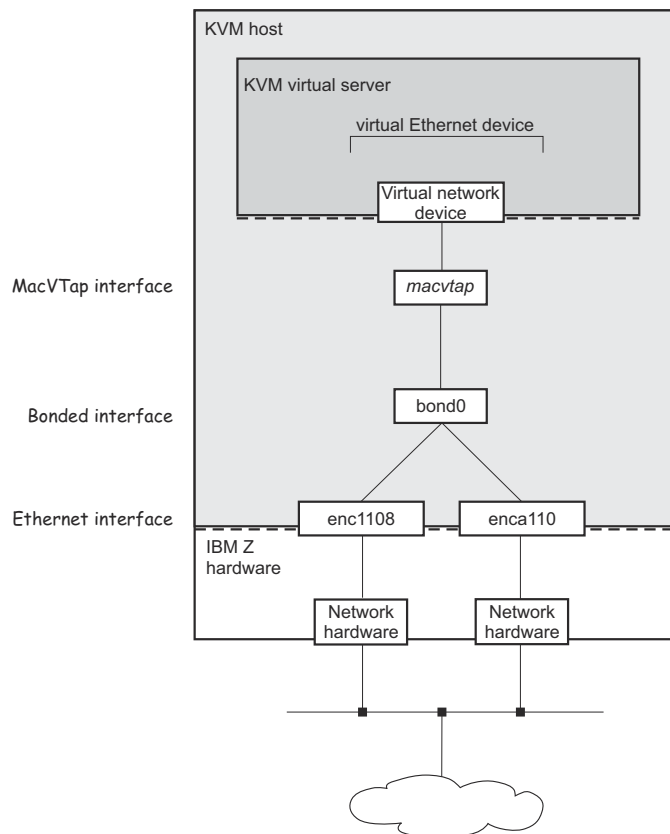
Figure 14. Configuration using a direct MacVTap connection

When you configure a virtual Ethernet device, you associate it with a network interface name on the host in the configuration-XML. In Figure 14 on page 24, 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 13 on page 23 can be realized as follows:
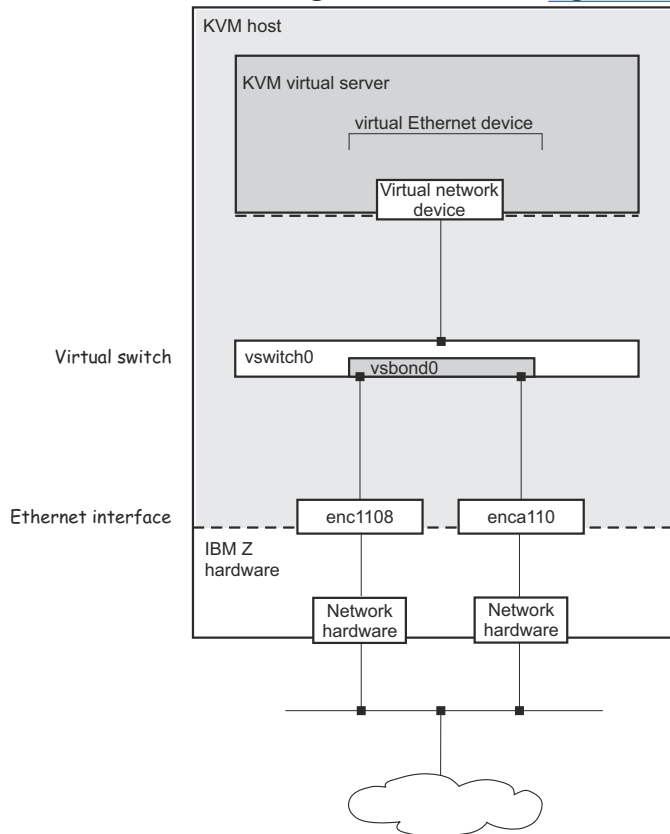
Figure 15. Configuration using a virtual switch

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

Related publications

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

Related tasks
"Preparing network devices" on page 43
Consider these aspects when setting up network interfaces for the use of virtual servers.

"Configuring virtual Ethernet devices" on page 124
Configure network interfaces, such as Ethernet interfaces, bonded interfaces, virtual LANs, or virtual switches as virtual Ethernet devices for a virtual server.

Chapter 5. IBM Secure Execution for Linux

IBM Secure Execution for Linux is a z/Architecture security technology that is introduced with IBM z15 and LinuxONE III.

It protects data of workloads that run in a KVM guest from being inspected or modified by the server environment. For more information, see *Introducing IBM Secure Execution for Linux*, SC34-7721.

The IBM Secure Execution for Linux feature must be enabled on your IBM Z or IBM LinuxONE hardware (see *IBM Dynamic Partition Manager (DPM) Guide*, SB10-7170).

Host setup

- The KVM host must run in logical partition (LPAR) mode. On DPM-enabled systems, the host must run directly in a partition.
- The KVM host distribution must support IBM Secure Execution for Linux. This support became available with kernel 5.7.
- The kernel parameters for the KVM host must include `prot_virt=1`.

KVM hosts that successfully start with support for IBM Secure Execution for Linux issue a kernel message like this: `prot_virt: Reserving <amount>MB as ultravisor base storage`.

Tip: Issue the **virt-host-validate** command on the host. The command output includes a line that starts with

```
QEMU: Checking for secure guest support    :
```

An OK after the colon confirms that you can run guests in IBM Secure Execution mode on this host. Otherwise, the colon is followed with information about unfulfilled requirements.

IBM Secure Execution for Linux does not automatically protect data that your workload writes to persistent storage. Depending on your requirements, you might have to set up encrypted devices to back your virtual block devices and storage pools.

Virtual server configuration

The virtual server must configure all virtio devices to use a bounce buffer in the guest, and must not include items that are incompatible with IBM Secure Execution for Linux, see Chapter 16, "Configuring for IBM Secure Execution for Linux," on page 141.

Guest preparation

Linux instances that are to run in IBM Secure Execution mode must be prepared as described in *Introducing IBM Secure Execution for Linux*, SC34-7721.

Guest migration

KVM guests that are prepared for IBM Secure Execution for Linux are configured to run only on specific Z or LinuxONE hardware systems.

Offline migration of a virtual server to another KVM host is supported if the following conditions are fulfilled:

- The target host supports guests in IBM Secure Execution mode.
- The target host runs on the same hardware system or on a hardware system for which the KVM guest has also been configured.

You cannot perform live migration of a KVM guest in IBM Secure Execution mode.

Constraints that result from memory and state protection

IBM Secure Execution for Linux is designed to protect the guest memory and state from the hypervisor.

As a result, you intentionally cannot perform the following actions:

- Host-initiated dumps
- Save and restore with the **virsh save** and **virsh restore** command.

Part 2. Device setup

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

Chapter 6. Preparing DASDs

After some preparation steps on the host, ECKD DASDs can be used as virtio block devices on virtual server.

Before you begin

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

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

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

About this task

The following publication describes how to configure, prepare, and work with DASDs:

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

Procedure

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

For a persistent setup, see your host administration documentation (see also).

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 by DASDs.

What to do next

Configure the DASDs as described in "Configuring a DASD, SCSI, or NVMe disk" on page 103.

Related concepts

"Virtual block devices" on page 11

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

Chapter 7. Preparing SCSI disks

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

Before you begin

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

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

   According to your product or distribution mechanism:

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

   See your host administration documentation to find out how to prepare multipath support.

2. Provide either of the following information:

   - The device bus-IDs of the FCP devices, target WWPNs, and the FCP LUNs of the SCSI disk.
   - The device mapper-created device node of the SCSI disk.

About this task

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

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

Procedure

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

For a persistent setup, see your host administration documentation (see also).

1. Linux senses the available FCP devices.

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

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

2. Set the FCP device online.

   You can use the **chccwdev** command to set an FCP device online or offline.

3. Configure the SCSI disks on the host.

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

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

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

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

Example

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

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

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

1. Display the available FCP devices.

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

2. Set the FCP device online.

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

3. Configure the SCSI disk on the host.

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

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

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

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

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

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

   ```
   /dev/mapper/36005076305ffc1ae00000000000023bc
   ```

What to do next

Configure the SCSI disks as described in "Configuring a DASD, SCSI, or NVMe disk" on page 103.
Related concepts
"Virtual SCSI devices" on page 9

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

Chapter 8. Preparing SCSI tape and medium changer devices

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

Before you begin

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

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

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

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

About this task

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

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

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

Procedure

The following steps describe a SCSI tape or medium changer setup on the host that does not persist across host reboots.

For a persistent setup, see your host administration documentation (see also).

a) 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.

b) 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.

c) 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.

d) Obtain the following information to be able to configure the SCSI tape or medium changer device:

- The SCSI host number that corresponds to the FCP device
- The SCSI ID of the target port
- The SCSI LUN

You obtain this information by issuing:

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

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

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

Example

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

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

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

1. Display the available FCP devices:

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

2. Bring the FCP device online:

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

3. Register the SCSI tape device on the host:

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

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

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

where:

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

What to do next

Configure the SCSI tape and medium changer devices as described in "Configuring a SCSI tape or medium changer device" on page 117.

Related concepts

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

Chapter 9. Preparing NVMe devices

After some preparation steps on the host, NVMe devices can be used as virtio block devices on virtual servers.

Before you begin

• You need to know the function address of the NVMe device.

Procedure

The following steps describe a setup on the host that does not persist across host reboots. For a persistent setup, see your host administration documentation.

1. Ensure that the NVMe device is connected to the host LPAR and online by listing your PCI devices with **lspci**.

   If the device is not listed in the command output, perform the following steps:

   a. Confirm that, in the hardware configuration, the device is assigned to the LPAR.

   b. Obtain the slot specification for the device.

   c. Write 1 to the power attribute of the slot representation in sysfs.

2. Obtain a persistent device node for the NVMe device.

   Use this node to configure the device to the virtual server. Perform all actions on the device from the guest. In particular, do not proceed to format or partition the device from the host.

Example

This example assumes that an NVMe device at slot 00000017 is offline, but assigned to the LPAR with function address 1003:00:00.0.

1. Set the NVMe device online.

   ```
   # echo 1 > /sys/bus/pci/slots/00000017/power
   1
   ```

2. The **lspci** output now includes the device.

   ```
   # lspci
   ...
   1003:00:00.0 Non-Volatile memory controller: ...
   ...
   ```

3. Find a device node. There might be multiple nodes for this device. This example uses a udev-created node that includes the function address.

   ```
   # ls /dev/**
   ...
   /dev/disk/by-path/pci-1003:00:00.0-nvme-1
   ```

What to do next

Configure the NVMe devices as described in "Configuring a DASD, SCSI, or NVMe disk" on page 103.

Related concepts

"Virtual block devices" on page 11

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

Chapter 10. Preparing network devices

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

About this task

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

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

Procedure

1. Create network interfaces as described in "Creating a network interface" on page 44.
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 46.

   b) To configure a virtual switch, perform the steps described in "Preparing a virtual switch" on page 49.

      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 124.

Related concepts

"Network devices as virtual Ethernet devices" on page 23

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

Creating a network interface

Create a network interface for a network device.

Before you begin

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

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

```
# lsqeth -p
```

About this task

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

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

Procedure

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

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

   ```
   # znetconf -u
   ```

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

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

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

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

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

   ```
   # znetconf -c
   ```

3. Activate the network interfaces.

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

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

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

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

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

Example

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

1. Determine the available network devices.

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

2. Configure the network devices and set them online.

```
# znetconf -a 0.0.8050 -o layer2=1 -o buffer_count=128
Scanning for network devices...
Successfully configured device 0.0.8050 (enc8050)

# 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 enc8050        online
...
```

3. Activate the network interfaces.

```
# ip link show enc8050
32: enc8050: <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 enc8050 up
```

4. Increase the transmit queue length.

```
# ip link set enc8050 qlen 2500
# ip link show enc8050
32: enc8050: <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 46
- or "Preparing a virtual switch" on page 49

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" on page 46.

2. Optional: Create a virtual LAN (VLAN) interface.

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

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

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

   Example:

   Create a virtual LAN interface with VLAN ID 623.

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

Preparing a bonded interface

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

Before you begin

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

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

About this task

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

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

Procedure

1. Define the bonded interface.

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

2. Set the bonding parameters for the desired bonding mode.

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

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 enc8050 master bond1
# ip link set enc1108 master bond1
# ip link show enc8050
5: enc8050: <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 enc1108
6: enc1108: <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: enc8050
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: enc8050
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: enc1108
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 124
Configure network interfaces, such as Ethernet interfaces, bonded interfaces, virtual LANs, through a
direct MacVTap interface.

Preparing a virtual switch

Virtual switches can be based on OSA-Express or HiperSockets network devices.

Before you begin

You need the Open vSwitch package. See your distribution documentation to find out whether this package is included in the distribution or how to obtain it.

Procedure

1. Configure the network devices that are to be used by the virtual switch to receive all frames with unknown MAC addresses. Use one of the following options:

   - Set VNIC characteristics for the network device
   - Configure the network device as a bridge port

   VNIC characteristics and bridge ports are mutually exclusive. You cannot configure a network as a bridge port and also set VNIC characteristics.

   See your Z hardware documentation about support for VNIC characteristics and for bridge port configurations. Support might differ for HiperSockets and OSA devices.

   - To configure a network device with VNIC characteristics enable flooding and learning:

     Use the **lszdev** command to show the current settings.

     ```
     # lszdev <ccwgroup> --info
     ```

     Example: This example shows a typical configuration for a bridge-like behavior of the device.

     ```
     # lszdev 0.0.5b11 --info
     ...
     ATTRIBUTE              ACTIVE     PERSISTENT
     ...
     vnicc/bridge_invisible  "0"        -
     vnicc/flooding          "1"        -
     vnicc/learning          "1"        -
     vnicc/learning_timeout  "600"      -
     vnicc/mcast_flooding    "1"        -
     vnicc/rx_bcast          "1"        -
     vnicc/takeover_learning "1"        -
     vnicc/takeover_setvmac  "1"        -
     ```

     If necessary, change the VNIC characteristic settings with the **chzdev** command.

     If the attribute values show "n/a", your device is already configured as a bridge port, or your Z hardware does not support VNIC characteristics for your device type, HiperSockets or OSA. See the section that follows about checking if a device is configured as a bridge port.

   - To check whether a network device is an active bridge port, read the `bridge_state` attribute of the qeth group device, for example with the **lszdev** command:

     ```
     # lszdev <ccwgroup> --columns ATTR:bridge_state
     ```

     Example:

     ```
     # lszdev 0.0.1108 --columns ATTR:bridge_state
     active/-
     ```

     Unless the output already shows "active", use the **chzdev** command to enable the bridge-port role:

     ```
     # chzdev qeth <device-bus-ID> layer2=1 bridge_role=primary
     ```

For more information about bridge ports or VNIC characteristics, see *Device Drivers, Features, and Commands*, SC33-8411.

2. Ensure that an Open vSwitch package is installed and running.

   Issue **systemctl status openvswitch** to find out whether Open vSwitch is running. If necessary, issue **systemctl start openvswitch** to start it.

   For more information about Open vSwitch commands, see openvswitch.org/support/dist-docs.

3. 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.

4. Create an uplink port.

   • If your network setup offers connections through two different OSA network-devices, you can configure for increased availability by using bonded ports. Use the **ovs-vsctl add-bond** command to create a bonded port.

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

   • Use the **ovs-vsctl add-port** command to create a regular switch port. HiperSockets connections do not rely on adapter hardware and are typically configured with a single port.

   ```
   # ovs-vsctl add-port <vswitch> <interface>
   ```

Related tasks
"Configuring a virtual switch" on page 126
Configure virtual switches as virtual Ethernet devices.

Example: virtual switch with HiperSockets devices and VNIC characteristics

Configure HiperSockets based virtual switches with VNIC characteristics.

About this task

This example creates virtual switch, vs_hs0. Because HiperSockets do not use adapter hardware that might constitute a single point of failure, a single uplink interface is sufficient.
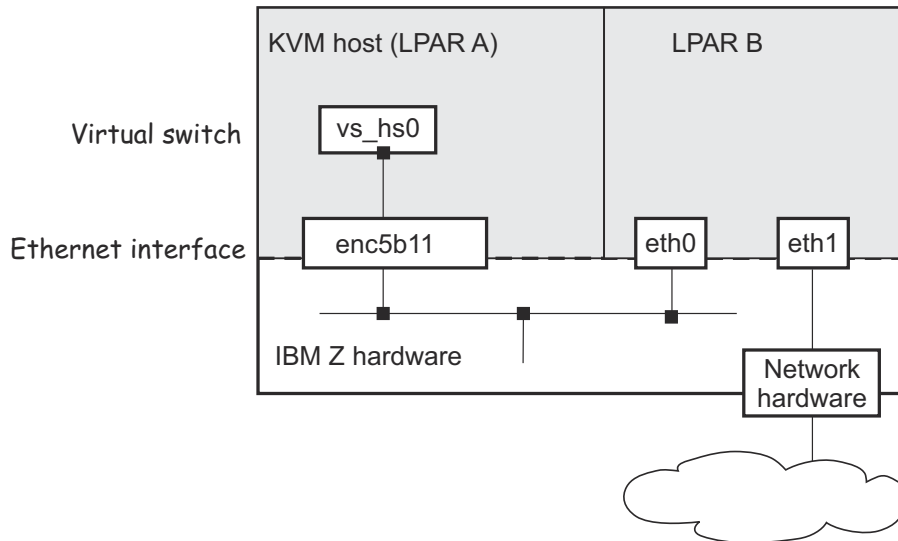
Figure 16. Virtual switch with HiperSockets

Figure 16 on page 51 shows the virtual switch with just one port which connects to the HiperSockets interface. When virtual server connect to the switch, libvirt automatically creates the required switch ports.

In the example, the KVM host in LPAR A uses the HiperSockets connection to communicate with LPAR B, which connects to an external network. An availability setup for the connection from LPAR B to the external network is outside the scope of the KVM host configuration.

Procedure

1. Verify that the VNIC characteristics of the HiperSockets devices include flooding and learning.

```
# lszdev 0.0.5b11 --info
...
ATTRIBUTE                ACTIVE     PERSISTENT
...
vnicc/bridge_invisible   "0"           -
vnicc/flooding           "0"           -
vnicc/learning           "0"           -
vnicc/learning_timeout   "0"           -
vnicc/mcast_flooding     "0"           -
vnicc/rx_bcast           "1"           -
vnicc/takeover_learning  "0"           -
vnicc/takeover_setvmac   "0"           -
```

Configure the device, if necessary.

```
# chzdev 0.0.5b11 vnicc/flooding=1 vnicc/learning=1 vnicc/learning_timeout=600 \
vnicc/mcast_flooding=1 vnicc/takeover_learning=1 vnicc/takeover_setvmac=1
# lszdev 0.0.5b11 --info
...
ATTRIBUTE                ACTIVE     PERSISTENT
...
vnicc/bridge_invisible   "0"           "0"
vnicc/flooding           "1"           "1"
vnicc/learning           "1"           "1"
vnicc/learning_timeout   "600"         "600"
vnicc/mcast_flooding     "1"           "1"
vnicc/rx_bcast           "1"           "1"
vnicc/takeover_learning  "1"           "1"
vnicc/takeover_setvmac   "1"           "1"
```

2. Create the virtual switch.

```
# ovs-vsctl add-br vs_hs0
# ovs-vsctl show
...
    Bridge "vs_hs0"
        Port "vs_hs0"
            Interface "vs_hs0"
                type: internal
    ovs_version: ...
```

3. Create an uplink port.

```
# ovs-vsctl add-port vs_hs0 enc5b11
# ovs-vsctl show
...
    Bridge "vs_hs0"
        Port "enc5b11"
            Interface "enc5b11"
        Port "vs_hs0"
            Interface "vs_hs0"
                type: internal
...
```

Example: virtual switch with OSA devices as bridge port

If possible, use bonded ports for OSA-based virtual switches to configure for increased availability.

About this task

This example creates a virtual switch, vs_osa0, that groups the network interfaces enc1108 and enca112 to a bonded interface, vsbond0:
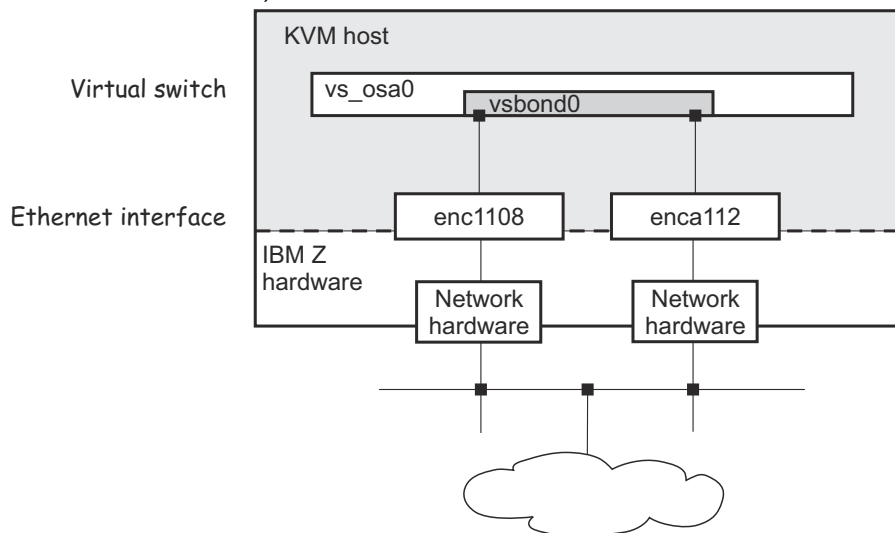
Figure 17. Virtual switch with a bonded interface

Figure 17 on page 52 shows the virtual switch with just two ports that connect to the bonded interfaces. When virtual servers connect to the switch, libvirt automatically creates the required switch ports.

Procedure

1. Verify that the OSA network-devices are configured as bridge ports.

```
# lszdev 0.0.1108,0.0.a112 --columns NAMES,ATTR:bridge_state
ID              ATTR:bridge_state
enc1108    active
enca112    active
```

2. Create the virtual switch.

```
# ovs-vsctl add-br vs_osa0
# ovs-vsctl show
3935bfec-241e-4610-a555-9e6f60987f87
    Bridge "vs_osa0"
        Port "vs_osa0"
            Interface "vs_osa0"
                type: internal
    ovs_version: ...
```

3. Create an uplink port.

```
# ovs-vsctl add-bond vs_osa0 vsbond0 enc1108 enca112
# ovs-vsctl show
...
    Bridge "vs_osa0"
        Port "vsbond0"
            Interface "enc1108"
            Interface "enca112"
        Port "vs_osa0"
            Interface "vs_osa0"
                type: internal
...
```

Chapter 11. Preparing VFIO pass-through devices

On the host, the host resources of a VFIO pass-through device must be controlled by a VFIO device driver. For pass-through DASD and pass-through cryptographic adapter resources, you must also create and configure a VFIO mediated device.

Preparing PCI pass-through devices

To make a PCIe device eligible as a VFIO pass-through device, you must bring it under control of the `vfio_pci` device driver.

Before you begin

PCI devices can be configured for automatic management by libvirt, as described in "Configuring pass-through PCI devices" on page 131. This management includes a dynamic host preparation. Perform the steps that follow only for devices that are not managed by libvirt.

Procedure

1. Free the intended pass-through PCI devices from the applicable PCI device driver by writing their function address to /sys/bus/pci/drivers/*<pci_device_driver>*/unbind.

   ```
   # echo <function_address> > /sys/bus/pci/drivers/<pci_device_driver>/unbind
   ```

   In the path, *<pci_device_driver>* is the name of the device driver that handles the particular PCI device, for example `mlx4_core` for 10 GbE RoCE Express devices.

   Tip: Issue **lspci -v** to find out which device driver controls the device.

2. Configure the vfio-pci device driver.

   You must ensure that the vfio-pci device driver is operational and configured to control the intended PCI card types. You need to specify the applicable vendor code and device code to configure a specific card type.

   Tip: Issue **lspci -n** to display the vendor and device codes for your PCI devices in the format *<vendor_code>*:*<device_code>*.

   - Configure the card types through sysfs.

     a. Unless it is compiled into the kernel, load the vfio-pci device driver.

        ```
        # modprobe vfio_pci
        ```

     b. Set the PCI card types to be controlled by the vfio-pci device driver by writing the vendor and device code to /sys/bus/pci/drivers/vfio-pci/new_id.

        ```
        # echo <vendor_code> <device_code> > /sys/bus/pci/drivers/vfio-pci/new_id
        ```

        Separate the vendor code and the device code with a blank. Repeat the command to specify multiple card types.

   - If the vfio-pci device driver is compiled as a separate module, you can configure the card types with the `ids=` module parameter.

     ```
     # modprobe vfio_pci ids=<vendor_code>:<device_code>
     ```

     You can specify multiple card types as a comma-separated value.

- If the vfio-pci device driver is compiled into the kernel, you can configure the card types with the `vfio_pci.ids=` module parameter.

  ```
  vfio_pci.ids=<vendor_code>:<device_code>
  ```

  You can specify multiple card types as a comma-separated value.

Example

To make a 10 GbE RoCE Express device with function address `0001:000:000:0` eligible for KVM guests:

```
# lspci -n
0001:000:000:0 15b3:1003
# echo 0001:000:000:0 > /sys/bus/pci/drivers/mlx4_core/unbind
# modprobe vfio_pci ids=15b3:1003
```

What to do next

You can now configure a virtual PCI device as a pass-through device that is based on the device at function address `0001:000:000:0`. For more information, see .

Preparing DASD pass-through devices

To make a DASD device eligible as a VFIO pass-through device, you must bring its subchannel under control of the `vfio_ccw` device driver and you must create a mediated device for the DASD.

Procedure

1. Load the `vfio_ccw` device driver.

   ```
   # modprobe vfio_ccw
   ```

2. Find out the subchannel bus-ID of your DASD.

   ```
   # lscss -d <device_bus_id>
   ```

   Example:

   ```
   # lscss -d 0.0.3000
   Device   Subchan. DevType CU Type Use PIM PAM POM CHPIDs
   ----------------------------------------------------------------
   0.0.3000 0.0.0004 3390/0A 3990/E9 YES C0  C0  FF  34400000 00000000
   ```

3. Free the intended DASD pass-through devices from the DASD device driver.

   ```
   # echo <device_bus_id> > /sys/bus/ccw/drivers/dasd-eckd/unbind
   ```

4. Free the subchannel of the intended DASD pass-through devices from the subchannel device driver.

   ```
   # echo <subchannel_bus_id> > /sys/bus/css/devices/<subchannel_bus_id>/driver/unbind
   ```

5. Hand control of the subchannel to the `vfio_ccw` device driver.

   ```
   # echo <subchannel_bus_id> > /sys/bus/css/drivers/vfio_ccw/bind
   ```

6. Obtain a UUID as an identifier for the mediated device.
   Example:

```
# uuidgen
18e124fb-b2fc-47f6-a407-f256b6c49767
```

7. Create a mediated device.

To create a persistent mediated device, use the **virsh nodedev-define** command and a node-device XML file.

For a transient mediated device, use the **virsh nodedev-create** command and a node-device XML file, or use general Linux commands.

- Follow these steps to create a persistent mediated device from a description in node-device XML format.

    a. Use the following template for your XML file:

    ```
    <device>
        <parent>SUBCHANNELSPEC</parent>
        <capability type="mdev">
            <type id="vfio_ccw-io"/>
            <uuid>UUIDSPEC</uuid>
        </capability>
    </device>
    ```

    b. Replace *SUBCHANNELSPEC* with a specification for your subchannel. The specification must consist of a prefix `css_` followed by a string that corresponds to the subchannel bus-ID with underscore characters (_) instead of dots (.). For example, for subchannel bus-ID `0.0.0004`, specify `css_0_0_0004`.

    c. Replace *UUIDSPEC* with the UUID that you obtained in step "6" on page 56.

    d. Create the mediated device by issuing a **virsh nodedev-define** command with the node-device XML file as a command argument.

    In libvirt, the mediated device is represented with a prefix, mdev_, which is followed by a string that corresponds to the UUID with underscore characters (_) instead of hyphens (-).

    e. Activate the mediated device with a **virsh nodedev-start** command.

    Example: This example uses a node-device XML file my_dasd_mdev.xml to create a mediated device. With a UUID 18e124fb-b2fc-47f6-a407-f256b6c49767, the resulting device in libvirt is mdev_18e124fb_b2fc_47f6_a407_f256b6c49767.

    ```
    # modprobe vfio_ccw
    # lscss -d 0.0.3000
    Device   Subchan. DevType CU Type Use PIM PAM POM CHPIDs
    ----------------------------------------------------------------
    0.0.3000 0.0.0004 3390/0A 3990/E9 YES C0  C0  FF  34400000 00000000
    # echo 0.0.3000 > /sys/bus/ccw/drivers/dasd-eckd/unbind
    # echo 0.0.0004 > /sys/bus/css/devices/0.0.0004/driver/unbind
    # echo 0.0.0004 > /sys/bus/css/drivers/vfio_ccw/bind
    # cat my_dasd_mdev.xml
    <device>
        <parent>css_0_0_0004</parent>
        <capability type="mdev">
            <type id="vfio_ccw-io"/>
            <uuid>18e124fb-b2fc-47f6-a407-f256b6c49767</uuid>
        </capability>
    </device>
    # virsh nodedev-define my_dasd_mdev.xml
    Node device 'mdev_18e124fb_b2fc_47f6_a407_f256b6c49767' defined from my_dasd_mdev.xml
    # virsh nodedev-start mdev_18e124fb_b2fc_47f6_a407_f256b6c49767
    Device mdev_18e124fb_b2fc_47f6_a407_f256b6c49767 started
    ```

    For more information about managing mediated devices with **virsh** commands and about creating a transient mediated device by using the **virsh nodedev-create** command, see "Managing mediated devices with libvirt" on page 62.

- As an alternative to using **virsh** commands, create a transient mediated device by using a general Linux command.

```
# echo <uuid> > /sys/bus/css/devices/<subchannel_bus_id>/mdev_supported_types/vfio_ccw-io/create
```

In the command, *<uuid>* is the UUID you obtained in step "6" on page 56.

Example: To make an ECKD DASD with bus ID `0.0.3000` eligible for KVM guests:

```
# modprobe vfio_ccw
# lscss -d 0.0.3000
Device   Subchan. DevType CU Type Use PIM PAM POM CHPIDs
----------------------------------------------------------------
0.0.3000 0.0.0004 3390/0A 3990/E9 YES C0  C0  FF  34400000 00000000
# echo 0.0.3000 > /sys/bus/ccw/drivers/dasd-eckd/unbind
# echo 0.0.0004 > /sys/bus/css/devices/0.0.0004/driver/unbind
# echo 0.0.0004 > /sys/bus/css/drivers/vfio_ccw/bind
# uuidgen
18e124fb-b2fc-47f6-a407-f256b6c49767
# echo 18e124fb-b2fc-47f6-a407-f256b6c49767 > \
/sys/bus/css/devices/0.0.0004/mdev_supported_types/vfio_ccw-io/create
```

What to do next

You can now use the mediated device to configure a virtual ECKD DASD as a pass-through device that is based on the DASD with bus ID `0.0.3000` on the host. See "Configuring a pass-through DASD" on page 130.

Preparing pass-through devices for cryptographic adapter resources

To make an AP queue eligible as a VFIO pass-through device, you must bring it under control of the `vfio_ap` device driver and you must create and configure a mediated device for it.

Free AP queues for use by KVM guests

By default, the `zcrypt` device driver controls all AP queues that are available to a KVM host and so makes them unavailable to guests.

About this task

In a common setup, the KVM host acts as a broker of AP queues for its guests without using AP queues itself. You use 2 bit masks in sysfs, `/sys/bus/ap/apmask` and `/sys/bus/ap/aqmask`, to manage host control of AP queues.

This procedure describes a crude, fast-path approach for freeing all queues for use by guests. For information about handling AP queues more selectively, see the section about the generic cryptographic device driver in *Device Drivers, Features, and Commands*.

Procedure

1. Load the `vfio-ap` device driver.

   ```
   # modprobe vfio_ap
   ```

2. Free all adapters by specifying the following command:

   ```
   # echo 0x0 > /sys/bus/ap/apmask
   ```

3. Free all domains by specifying the following command:

   ```
   # echo 0x0 > /sys/bus/ap/aqmask
   ```

4. Optional: Issue **lszcrypt -V** to confirm your settings.

With the verbose option, the **lszcrypt** command shows the AP queues that were freed from `zcrypt` control as being controlled by the `vfio-ap` device driver. The output of **lszcrypt** without the verbose option omits AP queues that are not controlled by `zcrypt`. The adapters themselves always remain under control of the applicable `zcrypt` submodule, `cex4card` in the example.

Example:

```
# lszcrypt -V
CARD.DOMAIN TYPE    MODE         STATUS  REQUESTS  PENDING HWTYPE QDEPTH FUNCTIONS  DRIVER
-------------------------------------------------------------------------------------------
00          CEX6A   Accelerator  online         0        0     12     08 -MC-A-NF-  cex4card
00.0001     CEX6A   Accelerator  online         0        0     12     08 -MC-A-NF-  vfio_ap
00.0002     CEX6A   Accelerator  online         0        0     12     08 -MC-A-NF-  vfio_ap
00.0004     CEX6A   Accelerator  online         0        0     12     08 -MC-A-NF-  vfio_ap
00.001b     CEX6A   Accelerator  online         0        0     12     08 -MC-A-NF-  vfio_ap
01          CEX6C   CCA-Coproc   online        31        0     12     08 S--D--NF-  cex4card
01.0001     CEX6C   CCA-Coproc   online        10        0     12     08 S--D--NF-  vfio_ap
01.0002     CEX6C   CCA-Coproc   online         7        0     12     08 S--D--NF-  vfio_ap
01.0004     CEX6C   CCA-Coproc   online         9        0     12     08 S--D--NF-  vfio_ap
01.001b     CEX6C   CCA-Coproc   online         5        0     12     08 S--D--NF-  vfio_ap
0a          CEX6P   EP11-Coproc  online         0        0     12     08 -----XNF-  cex4card
0a.0001     CEX6P   EP11-Coproc  online         0        0     12     08 -----XNF-  vfio_ap
0a.0002     CEX6P   EP11-Coproc  online         0        0     12     08 -----XNF-  vfio_ap
0a.0004     CEX6P   EP11-Coproc  online         0        0     12     08 -----XNF-  vfio_ap
0a.001b     CEX6P   EP11-Coproc  online         0        0     12     08 -----XNF-  vfio_ap
```

If the `vfio-ap` device driver has not been loaded, the DRIVER column in the verbose output shows `-no-driver-` instead of `vfio_ap`.

Important: The mask specifications in sysfs do not persist across reboots.

What to do next

Optional: If the Linux instance of your KVM host needs to use AP queues, assign these AP queues to the host before you proceed to assign any queues to guests. Follows these steps to assign an AP queue to the KVM host:

1. Write the numerical value of the AP queue's adapter ID, prefixed with a plus sign (+), to /sys/bus/ap/apmask. Repeat this step to assign AP queues from multiple adapters to the KVM host.

2. Write the numerical value of the AP queue's domain ID, prefixed with a plus sign (+), to /sys/bus/ap/aqmask. Repeat this step to assign AP queues from multiple domains to the KVM host.

For example, to assign AP queue 00.001b to the KVM host, issue:

```
# echo +0x0 > /sys/bus/ap/apmask
# echo +0x1b > /sys/bus/ap/aqmask
```

AP queues that are not assigned to the host can now be configured for KVM guests.

For more information about setting the adapter and domain masks, see *Device Drivers, Features, and Commands*.

Creating a mediated device with AP queues

KVM guests access AP queues through an AP Virtual Function I/O (VFIO) mediated device. The configuration of the mediated device defines the AP configuration of the KVM guest to which it is assigned.

About this task

In the steps that follow, a mediated device is first created, then adapters and domains are configured for the device. The adapter and domain specifications define a matrix of AP queues. After the mediated device is included in a KVM virtual server configuration, these AP queues become available to the guest that runs in the virtual server.

The examples in the steps that follow assume that 11 AP queues are available. These AP queues correspond to intersections of three adapters with IDs 00, 01, and 0a and four domains with IDs 0000, 0001, 0002, and 001b, omitting 00.001b.

According to the example in "Free AP queues for use by KVM guests" on page 58, 00.001b are already assigned to the KVM host and is no longer available to guests.

In the example, AP queues 01.0002 and 0a.0002 are to be assigned to a mediated device. The following figure illustrates how this assignment results from a specification of two adapters and a domain.

Figure 18. Matrix of AP queues available to KVM guests, omitting a queue that is used by the host

Procedure

1. Generate a UUID as an identifier for the mediated device.

   Example:

   ```
   # uuidgen
   4b0518fd-9237-493f-93c8-c5597f8006a3
   ```

2. Create a mediated device.

   To create a persistent mediated device, use the **virsh nodedev-define** command and a node-device XML file.

   For a transient mediated device, use the **virsh nodedev-create** command and a node-device XML file, or use general Linux commands.

   - Follow these steps to create a persistent mediated device from a description in node-device XML format.

     a. Use the following template for your node-device XML file:

     ```
     <device>
         <parent>ap_matrix</parent>
         <capability type="mdev">
             <type id="vfio_ap-passthrough"/>
             <uuid>UUIDSPEC</uuid>
         </capability>
     </device>
     ```

     b. Replace *UUIDSPEC* with the UUID that you obtained in step "1" on page 60.

     c. Configure adapters by specifying attr elements as child elements of the capability element.

attr name attribute:	assign_adapter
attr value attribute:	*<adapter_id>*

     For *<adapter_id>*, specify the adapter ID as two hexadecimal digits with prefix 0x.

     d. Configure domains by specifying attr elements as child elements of the capability element.

attr name attribute:	assign_domain

attr value attribute:		<domain_id>

For *<domain_id>*, specify the domain ID as four hexadecimal digits with prefix 0x.

e. Create the mediated device by issuing a **virsh nodedev-define** command with the node-device XML file as a command argument.

In libvirt, the mediated device is represented with a prefix, mdev_, which is followed by a string that corresponds to the UUID with underscore characters (_) instead of hyphens (-).

f. Activate the mediated device with a **virsh nodedev-start** command.

Example: This example uses a device configuration-XML file my_ap_mdev.xml to create a mediated device.

With a UUID 4b0518fd-9237-493f-93c8-c5597f8006a3, the command results in a device mdev_4b0518fd_9237_493f_93c8_c5597f8006a3 in libvirt and in a directory /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3 that represents the device in sysfs.

```
# cat my_ap_mdev.xml
<device>
    <parent>ap_matrix</parent>
    <capability type="mdev">
        <type id="vfio_ap-passthrough"/>
        <attr name="assign_adapter" value="0x01"/>
        <attr name="assign_adapter" value="0x0a"/>
        <attr name="assign_domain" value="0x0002"/>
        <uuid>4b0518fd-9237-493f-93c8-c5597f8006a3</uuid>
    </capability>
</device>
# virsh nodedev-define my_ap_mdev.xml
Node device 'mdev_4b0518fd_9237_493f_93c8_c5597f8006a3' defined from my_ap_mdev.xml
# virsh nodedev-start mdev_4b0518fd_9237_493f_93c8_c5597f8006a3
Device mdev_4b0518fd_9237_493f_93c8_c5597f8006a3 started
```

For more information about managing mediated devices with **virsh** commands and about creating a transient mediated device by using the **virsh nodedev-create** command, see "Managing mediated devices with libvirt" on page 62.

- As an alternative to using **virsh** commands, follow these steps to create a transient mediated device by using general Linux commands.

  a. Create the device by writing the UUID of step "1" on page 60 to /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create

  Example:

  ```
  # echo 4b0518fd-9237-493f-93c8-c5597f8006a3 \
  > /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/create
  ```

  This command creates a mediated device that is represented by a sysfs directory /sys/devices/vfio_ap/matrix/*<uuid>*, where *<uuid>* is the UUID that was used to create the device.

  b. Assign an adapter to the mediated device by writing the adapter ID, as two hexadecimal digits with a 0x prefix, to the device's assign_adapter sysfs attribute. Repeat this step to assign multiple adapters.

  Example: To assign adapters 01 and 0a:

  ```
  # echo 0x01 > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/assign_adapter
  # echo 0x0a > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/assign_adapter
  ```

  c. Assign a domain to the mediated device by writing the domain ID, as four hexadecimal digits with a 0x prefix, to /sys/devices/vfio_ap/matrix/*<device_id>*/assign_domain. Repeat this step to assign multiple domains.

  Example: To assign domain 0002:

```
# echo 0x0002 > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/assign_domain
```

d. For each domain that you assigned in the previous step, assign a control domain, so you can manage your domains from the guest that uses the mediated device.

Other than for z/VM® guests, usage domains on KVM guests are not automatically also control domains.

Example: To assign domain 0002 as a control domain:

```
# echo 0x0002 > /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/assign_control_domain
```

3. Optional: Read the `matrix` attribute of the mediated device to confirm that the assignment of adapters and domains resulted in the intended AP queue assignment.

```
# cat /sys/devices/vfio_ap/matrix/4b0518fd-9237-493f-93c8-c5597f8006a3/matrix
01.0002
0a.0002
```

What to do next
You can repeat this procedure to create multiple mediated devices, but you must not assign a specific AP queue to multiple mediated devices. You can use the attributes of the mediated device to investigate and control the device's properties.

```
ls -1 /sys/devices/vfio_ap/matrix/<device_id>
assign_adapter
assign_control_domain
assign_domain
control_domains
driver
iommu_group
matrix
mdev_type
power
remove
subsystem
uevent
unassign_adapter
unassign_control_domain
unassign_domain
```

In particular, you can write to the `assign_*` and `unassign_*` attributes to modify the mediated device, and you can use the `remove` attribute to remove the mediated device. For more details about these attributes, see the VFIO section in *Device Drivers, Features, and Commands*.

Important: Modifications of mediated devices through sysfs affect only the active device. For persistent mediated devices, such modifications do not affect the device definition in libvirt and they do not persist across device activation cycles and guest reboots.

You can now use the mediated device to configure the AP queues for a KVM guest. See .

Managing mediated devices with libvirt

Use libvirt commands to manage the lifecycle of VFIO mediated devices.

nodedev-create
    to create a transient VFIO mediated device and start it.

nodedev-define
    to define a persistent VFIO mediated device.

nodedev-start
    to activate an inactive persistent VFIO mediated device.

nodedev-list
to list mediated devices and other host resources that can be detected by libvirt.

nodedev-dumpxml
to display the properties, in node-device XML format, of a host resource that can be detected by libvirt.

nodedev-destroy
to deactivate a persistent VFIO mediated device or to completely remove a transient VFIO mediated device.

nodedev-undefine
to remove the definition of a persistent VFIO mediated device from libvirt.
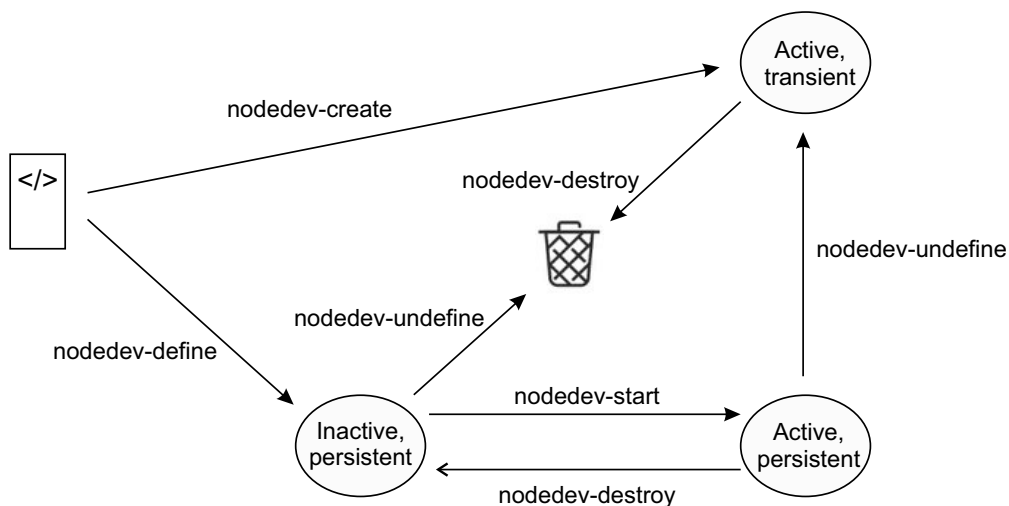
Figure 19. Lifecycle of VFIO mediated devices and virsh nodedev commands

As illustrated in the lifecycle of a mediated device begins with a device description in node-device XML format. You can use templates to write a node-device XML file.

If you already have a mediated device, for example a transient device that you have created with general Linux commands, you can obtain a node-device XML file as the output of the **nodedev-dumpxml** command.

You can create a persistent or a transient mediated device from the node-device XML file:

Persistent VFIO mediated device
Processing the node-device XML file with the **virsh nodedev-define** command creates an inactive persistent mediated device within libvirt. Before you can add the device to a virtual server, you must activate it with a **virsh nodedev-start** command.

The node-device XML for a persistent device must specify a UUID.

Applying the **virsh nodedev-destroy** command to an active persistent mediated device deactivates it. It can then be activated again, for example after a host reboot.

An inactive persistent device ceases to exist when you apply the **virsh nodedev-undefine** command to it. If you apply **virsh nodedev-undefine** to an active persistent device, it continues to exist as an active transient device.

Transient VFIO mediated device
Processing the node-device XML file with the **virsh nodedev-create** command creates a transient mediated device. Transient mediated devices are always active and can be added to a virtual server.

The node-device XML for a persistent device can but need not specify a UUID for the device. If omitted, the **virsh nodedev-create** command generates a UUID.

A transient mediated device ceases to exist when you apply the **virsh nodedev-destroy** command to it.

You can use mediated devices as hotplug devices or you can define them in a domain configuration-XML file of a virtual server. Mediated devices that are defined in a domain configuration-XML file require a stable UUID and the device must be active before the virtual server is started.

Managing mediated devices for DASD with libvirt

Use **virsh** commands to manage the lifecycle of mediated devices for DASD VFIO pass-through devices.

Before you begin
To make DASDs eligible for VFIO pass-through, you must free them from control of the DASD device driver. You must also bring the corresponding subchannel under control of the vfio_ccw-io device driver, see "Preparing DASD pass-through devices" on page 56.

Procedure
1. Confirm that the DASD for which you want to create a mediated device is eligible for VFIO pass-through, by issuing the following command:

```
# virsh nodedev-list --all --cap mdev_types
css_0_0_0072
```

If the libvirt representation of the subchannel of interest is not listed in the output, you must make it eligible for VFIO pass-through, see "Preparing DASD pass-through devices" on page 56.

Assuming that you want to create a mediated device for the DASD at subchannel 0.0.0072, confirm that no mediated device exists, by issuing the following command:

```
# virsh nodedev-list --tree
computer
  |
...
  +- css_0_0_0071
  |    |
  |    +- ccw_0_0_1006
  |        |
  |        +- block_dasdf_IBM_750000000DHVL1_0001_06
  |
  +- css_0_0_0072
  |
  |
...
```

The tree view verifies that no mediated device exists for subchannel 0.0.0072 by showing the subchannel as a leaf device.

2. Use the following template to create a file with a node-device XML description of the mediated device.

```
<device>
    <parent>SUBCHANNELSPEC</parent>
    <capability type="mdev">
        <type id="vfio_ccw-io"/>
    </capability>
</device>
```

Replace *SUBCHANNELSPEC* with a specification for your subchannel. The specification must consist of a prefix css_ followed by a string that corresponds to the subchannel bus-ID with underscore characters (_) instead of dots (.).

For example, for subchannel bus ID 0.0.0072, specify css_0_0_0072.

```
<device>
    <parent>css_0_0_0072</parent>
    <capability type="mdev">
        <type id="vfio_ccw-io"/>
    </capability>
</device>
```

3. As a child element of the capabilities element, add a uuid element that specifies the UUID that you want to use for the mediated device.

   You can use the **uuidgen** command to obtain a UUID. For transient mediated devices, you can omit this specification. A UUID is then generated for you with the command that creates the transient mediated device.

```
<device>
    <parent>css_0_0_0072</parent>
    <capability type="mdev">
        <type id="vfio_ccw-io"/>
        <uuid>7b36c4c2-b280-4ea7-8f40-77b192bf6fec</uuid>
    </capability>
</device>
```

4. Define the persistent mediated device by issuing a **virsh nodedev-define** command.

   The following example assumes that the node-device XML file of the previous step is stored at ~/ccwmdevs/css_72.xml:

```
# virsh nodedev-define ~/ccwmdevs/css_72.xml
Node device 'mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec' defined from ~/ccwmdevs/css_72.xml
```

   For transient devices, use the **virsh nodedev-create** command instead. This command creates and activates a mediated device and the activation step that follows for persistent devices does not apply.

   The mediated device of the example has a UUID 7b36c4c2-b280-4ea7-8f40-77b192bf6fec. Within libvirt, the device is represented as mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec.

   The subchannel listing now shows the mediated device:

```
# lscss --vfio
MDEV                                  Subchan.  PIM PAM POM  CHPIDs
-------------------------------------------------------------------------
7b36c4c2-b280-4ea7-8f40-77b192bf6fec  0.0.0072  c0  c0  ff   05020000 00000000
```

   You can also list the device with the **virsh nodedev-list** command.

```
# virsh nodedev-list --cap mdev
mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec
```

   The tree view of the **virsh nodedev-list** command now shows the mediated device as a direct child of the subchannel. Because the device is not yet activated, you need the --all option to include inactive devices in the command output.

```
# virsh nodedev-list --tree --all
computer
  |
...
  +- css_0_0_0071
  |   |
  |   +- ccw_0_0_1006
  |       |
  |       +- block_dasdf_IBM_750000000DHVL1_0001_06
  |
  +- css_0_0_0072
  |   |
  |   +- mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec
...
```

   For subchannel 0.0.0071, the sample output also shows a DASD that is controlled by the DASD device driver. In contrast to the mediated device, this DASD is not listed as a direct child of its subchannel but as a child of an intervening CCW device.

   Use the **virsh nodedev-dumpxml** command to display the properties of the mediated device in node-device XML format.

```
# virsh nodedev-dumpxml mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec
<device>
    <name>mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec</name>
    <path>/sys/devices/css0/0.0.0072/7b36c4c2-b280-4ea7-8f40-77b192bf6fec</path>
    <parent>css_0_0_0072</parent>
    <driver>
        <name>vfio_mdev</name>
    </driver>
    <capability type="mdev">
        <type id="vfio_ccw-io"/>
        <uuid>7b36c4c2-b280-4ea7-8f40-77b192bf6fec</uuid>
        <iommuGroup number="1"/>
    </capability>
</device>
```

5. For persistent mediated devices: Activate the mediated device by issuing a **virsh nodedev-start** command.

```
# virsh nodedev-start mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec
Device mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec started
```

What to do next

You can now use the mediated device to configure a virtual ECKD DASD as a pass-through device that is based on the DASD at subchannel 0.0.0072 on the host. See .

When you no longer need a mediated device, you can stop it with the **virsh nodedev-destroy** command.

```
# virsh nodedev-destroy mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec
Destroyed node device 'mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec'
```

A transient device ceases to exist as a result of the **virsh nodedev-destroy** command. A persistent device is deactivated, and you must use the **virsh nodedev-undefine** command to remove the inactive device from libvirt.

```
# virsh nodedev-undefine mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec
Undefined node device 'mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec'
```

Managing VFIO AP mediated devices with libvirt

Use libvirt commands to manage the lifecycle of mediated devices for VFIO pass-through of cryptographic resources.

Before you begin

Cryptographic adapter resources are managed as AP queues. To make an AP queue eligible for VFIO pass-through, you must bring it under control of the vfio_ap device driver, see .

Procedure

1. List the available cryptographic resources by issuing the following command:

```
# lszcrypt -V
card.domain type   mode           status ... functions driver
-------------------------------------------------------------
08           cex5c cca-coproc     online ... s--d--nf- cex4card
08.0001      cex5c cca-coproc     -      ... s--d--nf- vfio_ap
08.0002      cex5c cca-coproc     -      ... s--d--nf- vfio_ap
09           cex5c cca-coproc     online ... s--d--nf- cex4card
09.0001      cex5c cca-coproc     -      ... s--d--nf- vfio_ap
09.0002      cex5c cca-coproc     -      ... s--d--nf- vfio_ap
```

The AP queues with driver `vfio_ap` are eligible for a mediated device. In the sample output, these AP queues are `08.0001`, `08.0002`, `09.0001`, and `09.0002`, as shown in the `card.domain` column. These AP queues correspond to a matrix of adapters 08 and 09 with domains 0001 and 0002.

If the AP queues of interest are not controlled by `vfio_ap`, you have to free them from control of the crypto device driver, see "Free AP queues for use by KVM guests" on page 58.

You can also use the **virsh nodedev-list** command to list the adapters and queues.

```
# virsh nodedev-list --cap ap_card
ap_card08
ap_card09
# virsh nodedev-list --cap ap_queue
ap_08_0001
ap_08_0002
ap_09_0001
ap_09_0002
```

Use the **virsh nodedev-dumpxml** command to confirm that the AP queues are controlled by the `vfio_ap` device driver.

```
# virsh nodedev-dumpxml ap_08_0001
<device>
    <name>ap_08_0001</name>
    <path>/sys/devices/ap/card08/08.0001</path>
    <parent>ap_card08</parent>
    <driver>
        <name>vfio_ap</name>
    </driver>
    <capability type="ap_queue">
        <ap-adapter>0x08</ap-adapter>
        <ap-domain>0x0001</ap-domain>
    </capability>
</device>
```

2. Use the following template to create a file with a node-device XML description of the mediated device.

```
<device>
    <parent>ap_matrix</parent>
    <capability type="mdev">
        <type id="vfio_ap-passthrough"/>
    </capability>
</device>
```

As child elements of the capabilities element, add attr elements for adapters and domains to specify a subset of the matrix of available AP queues. The values are adapter IDs and domain IDs in hexadecimal notation with 0x prefixes.

For example, for the matrix of domain 0002 on both of the available adapters 08 and 09, add three attr elements, one for each adapter and one for the domain.

```
<device>
    <parent>ap_matrix</parent>
    <capability type="mdev">
        <type id="vfio_ap-passthrough"/>
        <attr name="assign_adapter" value="0x08"/>
        <attr name="assign_adapter" value="0x09"/>
        <attr name="assign_domain" value="0x0002"/>
    </capability>
</device>
```

3. As a child element of the capabilities element, add a uuid element that specifies the UUID that you want to use for the mediated device.

You can use the **uuidgen** command to obtain a UUID. For transient mediated devices, you can omit this specification. A UUID is then generated for you with the command that creates the transient mediated device.

For example, the following specification configures a stable UUID `bfccf00d-21f1-448c-9979-b7341129d985` for the mediated device.

```
<device>
    <parent>ap_matrix</parent>
    <capability type="mdev">
        <type id="vfio_ap-passthrough"/>
        <attr name="assign_adapter" value="0x08"/>
        <attr name="assign_adapter" value="0x09"/>
        <attr name="assign_domain" value="0x0002"/>
        <uuid>bfccf00d-21f1-448c-9979-b7341129d985</uuid>
    </capability>
</device>
```

4. Define the persistent mediated device by issuing a **virsh nodedev-define** command.

   The following example assumes that the node-device XML file of the previous step is stored at ~/apmatrices/08-09_0002.xml.

```
# virsh nodedev-define ~/apmatrices/08-09_0002.xml
Node device mdev_bfccf00d_21f1_448c_9979_b7341129d985 defined from ~/apmatrices/08-09_0002.xml
```

   For transient devices, use the **virsh nodedev-create** command instead. This command creates and activates a mediated device and the activation step that follows for persistent devices does not apply.

   The mediated device of the example has a UUID `bfccf00d-21f1-448c-9979-b7341129d985`. Within libvirt, the device is represented as mdev_bfccf00d_21f1_448c_9979_b7341129d985.

   You can list the device with the **virsh nodedev-list** command.

```
# virsh nodedev-list --cap mdev
mdev_bfccf00d_21f1_448c_9979_b7341129d985
```

   The tree view of the **virsh nodedev-list** command shows that the mediated device corresponds to a matrix of AP queues. Because the device is not yet activated, you need the --all option to include inactive devices in the command output.

```
$ virsh nodedev-list --tree --all
computer
  |
  +- ap_card08
  |
  | +- ap_08_0001
  | +- ap_08_0002
  |
  +- ap_card09
  |   |
  |   +- ap_09_0001
  |   +- ap_09_0002
  |
  +- ap_matrix
  |   |
  |   +- mdev_bfccf00d_21f1_448c_9979_b7341129d985
  ...
```

   Use the **virsh nodedev-dumpxml** command to display the properties of the mediated device in node-device XML format.

```
# virsh nodedev-dumpxml mdev_bfccf00d_21f1_448c_9979_b7341129d985
<device>
    <name>mdev_bfccf00d_21f1_448c_9979_b7341129d985</name>
    <path>/sys/devices/vfio_ap/matrix/bfccf00d-21f1-448c-9979-b7341129d985</path>
    <parent>ap_matrix</parent>
    <driver>
        <name>vfio_mdev</name>
    </driver>
    <capability type="mdev">
        <type id="vfio_ap-passthrough"/>
        <iommuGroup number="4"/>
    </capability>
</device>
```

The path element contains the sysfs path of the mediated device. Read the matrix attribute to display the matrix of AP queues.

```
# cat /sys/devices/vfio_ap/matrix/bfccf00d-21f1-448c-9979-b7341129d985/matrix
08.0002
09.0002
```

5. For persistent mediated devices: Activate the mediated device by issuing a **virsh nodedev-start** command.

```
# virsh nodedev-start mdev_bfccf00d_21f1_448c_9979_b7341129d985
Device mdev_bfccf00d_21f1_448c_9979_b7341129d985 started
```

What to do next

You can now use the mediated device to configure the AP queues for a KVM guest. See "Configuring cryptographic adapter resources" on page 133.

When you no longer need a mediated device, you can stop it with the **virsh nodedev-destroy** command.

```
# virsh nodedev-destroy mdev_bfccf00d_21f1_448c_9979_b7341129d985
Destroyed node device 'mdev_bfccf00d_21f1_448c_9979_b7341129d985'
```

A transient device ceases to exist as a result of the **virsh nodedev-destroy** command. A persistent device is deactivated, and you must use the **virsh nodedev-undefine** command to remove the inactive device from libvirt.

```
# virsh nodedev-undefine mdev_bfccf00d_21f1_448c_9979_b7341129d985
Undefined node device 'mdev_bfccf00d_21f1_448c_9979_b7341129d985'
```

Part 3. Configuration

Create configuration-XML files to configure virtual servers, devices, virtual networks, and storage pools.

Chapter 12. 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.

Before you begin

Tip: With a single `virt-install` command, you can configure and define a virtual server, and install and boot a guest, see Chapter 25, "Fast path to a running guest - virt-install," on page 193.

Note: Virtual servers for guests in IBM Secure Execution mode have specific configuration requirements, see Chapter 16, "Configuring for IBM Secure Execution for Linux," on page 141.

Procedure

1. Create a domain configuration-XML file.

   See "Domain configuration-XML" on page 75.
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 77.
   b) Configure virtual CPUs.

      See "Configuring virtual CPUs" on page 85.
   c) Configure memory.

      See "Configuring virtual memory" on page 89.
   d) Optional: Configure the collection of QEMU core dumps.

      See "Configuring the collection of QEMU core dumps" on page 92.
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 93.
   b) Configure persistent devices.

      See "Configuring devices with the virtual server" on page 94.
   c) Configure the console device.

      See "Configuring the console" on page 95.
   d) Optional: Configure a watchdog device.

      See "Configuring a watchdog device" on page 96.
   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 97.
   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 99.
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 148.

Domain configuration-XML

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

Root element

domain
Specify kvm as the domain type.

domain type attribute:	kvm

Selected child elements

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

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

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

cputune
Groups the CPU tuning parameters:

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

os
Groups the operating system parameters:

> **type**
> Specifies the machine type.
>
> **kernel**
> Optionally specifies the kernel image file on the host.
>
> **initrd**
> Optionally specifies the initial ramdisk on the host.
>
> **cmdline**
> Optionally specifies command-line arguments.

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

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

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

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

on_crash element:	preserve

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

launchSecurity
Prepares the virtual server for guests in IBM Secure Execution mode.

Example

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

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

Related reference

"Selected libvirt XML elements" on page 235
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 virtio block device as IPL device" on page 77.

- 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 79 or "Configuring a kernel image file as IPL device" on page 80.

You can use the **chreipl** command on the guest to configure alternative reboot devices.

You can use the **virt-xml** command to start the virtual server and boot the guest from an alternative boot device, see Chapter 26, "Booting from a temporary boot device," on page 195.

Configuring a virtio block device as IPL device

Boot a guest from a virtio block device that is configured as a disk device.

Before you begin

Prepare your device with a root file system and a bootable kernel as described in the section for your device type.

Procedure

1. Configure the disk with the root file system as a persistent device, see "Configuring virtual block devices" on page 103.

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 247).

boot order attribute:	*\<number\>*

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

   If the specified device has a boot menu configuration, you can use the loadparm attribute of the boot element to specify a particular menu entry to be booted.

boot loadparm attribute:	*\<selection\>*

3. For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

Example

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

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

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

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

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

Configuring a VFIO DASD as IPL device

Boot a guest from a VFIO pass-through DASD.

Before you begin
Prepare your device with a root file system and a bootable kernel.

Procedure

1. Configure the DASD with the root file system as a persistent device, see "Configuring a pass-through DASD" on page 130.
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 247).

boot order attribute:	*<number>*

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

If the specified device has a boot menu configuration, you can use the loadparm attribute of the boot element to specify a particular menu entry to be booted.

boot loadparm attribute:	*<selection>*

Example

The following domain configuration-XML configures a virtual server V3, which is booted from a VFIO pass-through DASD 00a1:

```
<domain type="kvm">
    <name>V3</name>
    ...
    <devices>
        ...
        <hostdev mode="subsystem" type="mdev" model="vfio-ccw">
            <source>
                <address uuid="90c6c135-ad44-41d0-b1b7-bae47de48627"/>
            </source>
            <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00a1"/>
            <boot order="1"/>
        </hostdev>
        ...
    </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 122.

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

2. Per default, the guest is booted from the first specified disk device in the current libvirt-internal configuration. To avoid possible errors, explicitly specify the boot device with the boot element in the disk device definition (see "<boot>" on page 247).

boot order attribute:	*<number>*

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

3. For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

Example

1. Specify the ISO image.

   Configure the ISO image as a virtual DVD:

```
<devices>
    ...
    <controller type="scsi" model="virtio-scsi" index="4"/>
```

```
    <disk type="file" device="cdrom">
        <driver name="qemu" type="raw" io="native" cache="none"/>
        <source file="/var/lib/libvirt/images/LinuxDVD1.iso"/>
        <target dev="sda" bus="scsi"/>
        <address type="drive" controller="4" bus="0" target="0" unit="0"/>
        <readonly/>
        <boot order="1"/>
    </disk>
    ...
 </devices>
```

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

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

2. Provide a disk for the guest installation:

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

Configuring a kernel image file as IPL device

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

Procedure

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

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

   a) Specify the fully qualified path to the initial ramdisk on the host with the initrd element, which is a child of the os element (see "<initrd>" on page 270).

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 274).

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 249).

   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.

3. For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

Kernel images for IBM Secure Execution for Linux, typically, include all data that is required for booting, so no separate initrd and cmdline are present.

Example

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

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

2. Provide a disk for the guest installation:

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

What to do next

When configuring a different boot device with the boot element, for example the disk for the guest installation, you must remove the os element. The boot and os elements are mutually exclusive.

Configuring a network IPL device

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

Before you begin

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

Procedure

1. Configure an interface to a virtual network, to an Open vSwitch, or for a direct MacVTap connection (see "<interface>" on page 272).

interface type attribute:	network | bridge | direct

2. Use the source element as a child of the interface element, to specify the network or bridge that provides the connection to the network boot server (see "<source> as child element of <interface>" on page 299).

   • For a virtual network:

source network attribute:	*<network-name>*

   • For an Open vSwitch or a direct MacVTap connection:

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

3. Specify `virtio` as the interface type with model element, which is a child of the interface element (see "<model> as a child element of <interface>" on page 286).

model type attribute:	virtio

4. Specify the boot device with the boot element, which is a child of the interface element (see "<boot>" on page 247).

boot order attribute:	*<number>*

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

5. Specify the device type as CCW and a device bus-ID with the address element as a child of the interface element (see "<address> as child element of <interface>" on page 242).

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

6. For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

Example

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

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

Example of an initial installation

The guest installation process depends on your product or distribution.

Procedure

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

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

   You can either mount the ISO image containing the installation files during the guest installation process, copy the required files to the host file system, or connect to an FTP server.

2. Create a domain configuration-XML file.

   a) If you intend to boot from an ISO image, the domain configuration-XML file should contain:
   - The fully qualified path and filename of the ISO image.
   - A persistent device configuration for the device that will contain the bootable installed guest.

   Example:

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

         <!-- IPL device -->
         <controller type="scsi" model="virtio-scsi" index="4"/>
         <disk type="file" device="cdrom">
           <driver name="qemu" type="raw" io="native" cache="none"/>
   ```

```
            <source file="/var/lib/libvirt/images/LinuxDVD1.iso"/>
            <target dev="sda" bus="scsi"/>
            <address type="drive" controller="4" bus="0" target="0" unit="0"/>
            <readonly/>
            <boot order="1"/>
        </disk>

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

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

b) If you intend to boot from a kernel image file and an initial ramdisk, the domain configuration-XML file should contain:

- The fully qualified path and filename of the kernel image.
- The fully qualified path and filename of the initial ramdisk.
- The kernel command-line parameters.
- A persistent device configuration for the device that will contain the bootable installed guest.

Example:

```
<domain>
    ...
    <os>

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

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

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

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

3. Start the virtual server for the initial installation.

4. Install the guest as described in your distribution documentation.

5. When a bootable guest is installed, modify the domain configuration-XML using **virsh edit** to boot from the IPL disk containing the boot record.

   a) In case you installed the guest using the ISO image:

   Example:

```
<domain>
    ...
    <os>
        ...
    </os>
```

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

        <!-- IPL device -->
        <controller type="scsi" model="virtio-scsi" index="4"/>
        <disk type="file" device="cdrom">
            <driver name="qemu" type="raw" io="native" cache="none"/>
            <source file="/var/lib/libvirt/images/LinuxDVD1.iso"/>
            <target dev="sda" bus="scsi"/>
            <address type="drive" controller="4" bus="0" target="0" unit="0"/>
            <readonly/>
        </disk>

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

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

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

Example:

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

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

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

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

Configuring virtual CPUs

Configure virtual CPUs for a virtual server.

Related concepts

"CPU management" on page 199
Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks

"Managing virtual CPUs" on page 176
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 303).

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

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

   Note: It is not useful to configure more virtual CPUs than available host CPUs.

2. To configure the actual number of virtual CPUs that are available for the virtual server when it is started, specify the current attribute. The value of the current attribute is limited by the maximum number of available virtual CPUs.

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

vcpu current attribute:	*<number>*

Example

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

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

This example configures a maximum of 5 available virtual CPUs for the virtual server. When the virtual server is started, only 2 virtual CPUs are available. You can modify the number of virtual CPUs that are available for the running virtual server using the virsh **setvcpus** command (see "Modifying the number of virtual CPUs" on page 176).

```
<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 200.

Procedure

Use the cputune element to group CPU tuning elements.

- You specify the CPU weight by using the shares element (see "<shares>" on page 294).

shares element:	*<CPU-weight>*

Example

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

Configuring the CPU model

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

About this task

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

- Disable a CPU feature that causes problems for a particular application.
- Keep the option for a live migration to an earlier hardware model that does not support all CPU features of the current hardware (see "IBM Z hardware CPU model" on page 164).
- Keep the option for a live migration to a KVM host with an earlier QEMU version that does not support all CPU features of the current version.

Procedure

- To define a CPU model with a specific set of hardware features, specify:

  1. Declare that a specific CPU model is to be configured.

cpu mode attribute:	custom
cpu match attribute:	exact

     (see "<cpu>" on page 252

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

model element:	*<cpu_model>*

     (see "<model> as a child element of <cpu>" on page 285)

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

     Example: This example identifies z15 as an eligible CPU model.

     ```
     <model usable="yes">z15</model>
     ```

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

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

     <mainframe_model>.<n>
     : Specifies the default CPU features for the *<n>*th major hardware release of a mainframe model. This default is the subset of features that are supported by the QEMU version of the KVM host. For example, z14.2 specifies the QEMU supported CPU features of IBM z14 hardware with its first major update in October 2018 (informally also known as GA2).

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

3. Optionally, use one or more <feature> elements as child elements of the <cpu> element. With each <feature> element, you can add or remove an individual CPU feature from the CPU model of the previous step (see "<feature>" on page 264).

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

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

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

cpu mode attribute:	host-model

When the virtual server is started, libvirt expands the CPU model to an explicit specification in the libvirt-internal configuration. This explicit specification makes guest migration to a suitable alternative KVM host an option.

- To use the same CPU model as the KVM host, specify:

cpu mode attribute:	host-passthrough

(see "<cpu>" on page 252)

Other than for host-model, libvirt does not interpret the host's CPU model, and the CPU definition is not expanded in the libvirt-internal configuration. The lack of an explicit CPU specification has the following consequences:

– After a migration to a more advanced KVM host, the virtual server can use all CPU features that the new host offers.

– Live migration for a virtual server with this specification is highly risky because libvirt cannot assess the compatibility of the CPU model of the source and destination KVM host.

Example

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

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

- To require the QEMU supported CPU features of a z15 mainframe, but without the `iep` feature:

```
<cpu mode="custom">
  <model>z15</model>
  <feature policy="disable" name="iep">
</cpu>
```

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

Configuring virtual memory

Configure the virtual server memory.

Related concepts

"Memory management" on page 203

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 180

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 281).

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

Example

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

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

Tuning virtual memory

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

About this task

For more information about memory tuning, see Chapter 28, "Memory management," on page 203.

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 295).

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

Example

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

```
        </memtune>
        ...
</domain>
```

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

Configuring huge pages

Configure the virtual server to use huge pages.

Procedure

Use the memoryBacking element with a nested hugepages element. The memoryBacking element is a child of the domain element. Do not specify any attributes.

See "Using huge pages" on page 203 for prerequisites and restrictions.

Example

```
<domain type="kvm">
    <name>vserv1</name>
    ...
    <memoryBacking>
        <hugepages/>
    </memoryBacking>
    ...
</domain>
```

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 281)

Example

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

Configuring the user space

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

Procedure

The optional emulator element contains path and file name of the user space process (see "<emulator>" on page 263).

The emulator element is a child of the devices element. If you do not specify it, libvirt automatically inserts the user space configuration to the libvirt-internal configuration when you define it.

emulator element:	*<emulator-file>*

Example:

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

Configuring devices with the virtual server

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

Before you begin

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

Procedure

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

   For more information about I/O threads, see "I/O threads" on page 207.

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

   (see "<iothreads>" on page 273)

   Example:

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

2. Specify a configuration-XML for each device.

   Chapter 13, "Configuring devices," on page 101 describes how to specify a configuration-XML for a device.

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

Example

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

Configuring the console

Configure the console by using the console element.

Procedure

1. You configure the host representation of the console by using the console type attribute (see "<console>" on page 250).

   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 300).

   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 278). 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 185
Open a console when you start a virtual server, or connect to the console of a running virtual server.

Configuring a watchdog device

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

About this task

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

Procedure

Use the watchdog element as child of the devices element to configure a watchdog device (see).

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

Example

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

Disabling protected key encryption

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

Before you begin

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

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

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

About this task

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

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

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

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

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

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

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

Procedure

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 280)

Example

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

Chapter 13. Configuring devices

A device configuration maps host devices and resources to device representations on a virtual server.

About this task

You can configure devices as part of a domain configuration-XML file. Some devices can also be configured in a separate device configuration-XML file. Separately defined devices can be attached to an already defined virtual server, see "Device configuration-XML" on page 133.

Depending on the virtualization technique, virtual devices can be VFIO or SCSI pass-through devices, or they can be virtio devices, see "Device virtualization techniques" on page 8.

VFIO pass-through DASDs and virtio devices are accessed through a virtual channel subsystem, see "CCW device specifications for virtio devices" on page 102.

Configuring virtio devices

On Z, the virtio framework virtualizes devices as virtio CCW devices. KVM guests access virtio CCW devices through a virtual z/Architecture channel subsystem.

CCW device specifications for virtio devices

On z/Architecture, virtual servers access virtio devices as virtual channel subsystem (CCW) devices. Virtual CCW devices have common specifications in the device configuration.

Virtual servers access virtual CCW devices through a virtual channel subsystem, with channel subsystem-ID `0x00`. In device configurations, this channel subsystem is specified through the reserved channel subsystem-ID `0xfe`.

Virtual CCW devices are all of channel path type `0x32` and virtual control unit type `0x3832`. Device types for virtio devices are specified as control unit models as listed in .

Table 1. Device types and models in the virtual channel subsystem

Device	Device type
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	0x02
• Serial devices For z/Architecture, this device type is not well suited for a console device. The preferred option is an SCLP based console.	0x03
• Random number generators (RNGs) Do not configure a virtual random number generator for a virtual server, unless the host is equipped with a hardware random number generator, such as the secure IBM CCA coprocessor of a Crypto Express adapter.	0x04
• Balloon devices This device can be suppressed in the configuration of the virtual server	0x05
• SCSI Host Bus Adapter (virtio-scsi)	0x08
• Virtual graphic card, GPU (virtio-gpu)	0x10
• Virtual human input interface (virtio-input)	0x12
• Device for communication with the host (virtio-vsock)	0x13

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, SCSI, or NVMe disk" on page 103
- "Configuring an image file as storage device" on page 109
- "Configuring a volume as storage device" on page 111

Configuring a DASD, SCSI, or NVMe disk

You can configure DASDs, SCSI disks, and NVMe devices as virtio block devices in the configuration-XML.

Before you begin

Make sure that

- DASDs are prepared as described in Chapter 6, "Preparing DASDs," on page 31.
- SCSI disks are prepared as described in Chapter 7, "Preparing SCSI disks," on page 33.
- Chapter 9, "Preparing NVMe devices," on page 41

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 207.

About this task

You specify DASDs, FC-attached SCSI disks, and NVMe devices through device nodes. 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 255)

    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 259)

    <IOthread-ID> assigns an I/O thread for I/O operations on the device.

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

For devices that are defined in separate device configuration-XML files:
Specify an existing I/O thread or use the **virsh iothreadadd** command to create a new one 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 assigned 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 207.

c) For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

d) Specify virtio as the virtual server disk device type.

target bus attribute:	virtio

(see "<target> as child element of <disk>" on page 301)

2. Identify the device on the host.

Specify a device node of the device.

source dev attribute:	*<device-node>*

(see "<source> as child element of <disk>" on page 296)

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.

For NVMe devices:

Use udev-created persistent device nodes.

For live migration of a virtual server, device nodes must be such that they addresses equivalent resources on both the source and destination host. Migration of the virtio block device is then possible through *disk migration*, see step "3.c" on page 171 in "Performing a live migration" on page 170.

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 301)

   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>, and <memballoon>" on page 239)

      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.

Example of a DASD configuration

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

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

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

Define the devices:

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

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

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

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

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

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

Example of a SCSI disk configuration

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

```
# multipathd -k'show topology'
36005076305ffc1ae00000000000021df dm-3 IBM     ,2107900
size=30G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=0 status=active
  |- 1:0:7:1088372769 sdm   8:192  active ready running
  |- 1:0:3:1088372769 sdn   8:208  active ready running
  |- 1:0:5:1088372769 sdo   8:224  active ready running
  |- 1:0:4:1088372769 sdl   8:176  active ready running
  |- 0:0:3:1088372769 sdbd  67:112 active ready running
  |- 0:0:4:1088372769 sdax  67:16  active ready running
  |- 0:0:8:1088372769 sdbj  67:208 active ready running
  `- 0:0:6:1088372769 sdbp  68:48  active ready running

...
36005076305ffc1ae00000000000021d5 dm-0 IBM     ,2107900
size=30G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='service-time 0' prio=0 status=active
  |- 1:0:4:1087717409 sdg   8:96   active ready running
  |- 1:0:7:1087717409 sdq   65:0   active ready running
  |- 1:0:5:1087717409 sdi   8:128  active ready running
  |- 1:0:3:1087717409 sdf   8:80   active ready running
  |- 0:0:4:1087717409 sdaw  67:0   active ready running
  |- 0:0:3:1087717409 sdbc  67:96  active ready running
  |- 0:0:6:1087717409 sdbo  68:32  active ready running
  `- 0:0:8:1087717409 sdbi  67:192 active ready running
```

Define the devices:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021df"/>
    <target dev="vda" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x1a10"/>
</disk>
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x1a12"/>
</disk>
```

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

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

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

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

Example of an NVMe device configuration

To see the function addresses of the available NVMe devices on the host, enter:

```
# lspci
...
1003:00:00.0 Non-Volatile memory controller: ...
...
```

The function addresses in the example is 1003:00:00.0.

If the **lspci** command does not list the device, it has not been assigned to the LPAR in the hardware configuration, or you need to set it online through its representation in /sys/bus/pci/slots.

Find a suitable device node, for example, a udev-created node that includes the function address:

```
# ls /dev/**
...
/dev/disk/by-path/pci-1003:00:00.0-nvme-1
```

Define the device:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/disk/by-path/pci-1003:00:00.0-nvme-1"/>
    <target dev="vdn" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe001"/>
</disk>
```

The guest on 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 nodes and devices does not persist across reboots, and the node assigned by the guest need not match the specification for dev= on the target element.

The guest on the virtual server sees the standard device node /dev/vdn and udev-created device nodes . To see the mapping between the standard device nodes and the udev-created by-path device nodes, enter:

```
[root@guest:] # ls /dev/disk/by-path -l
...
lrwxrwxrwx 1 root root  9 Oct 20 15:20 ccw-0.0.e001 -> ../../vdh...
```

The guest always sees 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.e001 0.0.0000  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 "QEMU image command" on page 441 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 255)

   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 259)

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

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

   c) For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

   d) Specify virtio as the virtual server disk device type.

target bus attribute:	virtio

   (see "<target> as child element of <disk>" on page 301)

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 296)

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 301)

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>, and <memballoon>" on page 239)

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 77
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 255)

   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 259)

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

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

   c) For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

   d) Specify virtio as the virtual server disk device type.

target bus attribute:	virtio

   (see "<target> as child element of <disk>" on page 301)

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 296)

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 301)

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>, and <memballoon>" on page 239)

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 77
Specify the device that contains a root file system, or a prepared kernel image file.

Configuring a virtual graphics card

Virtual graphics cards support remote access from workstations to guest applications that require user interaction through a graphical user interface.

About this task

Some workstation setups do not provide ready access to graphical applications on KVM guests on Z. Virtual graphics processing unit (GPU) devices act as virtual graphics cards that provide remote frame buffers for workstation clients. Workstation clients can use Virtual Network Computing (VNC) to access these frame buffers. Along with the GPU, you need to configure one or more input devices, like a keyboard or a mouse.

Procedure

1. Use the graphics element as child of the devices element to configure a GPU device.

graphics type attribute:	vnc
graphics autoport attribute:	yes
graphics port attribute:	*<port_number>*

   Attributes autoport and port are mutually exclusive. Use autoport to automatically assign a free port number. Use port to explicitly assign a port number, for example, if you must adhere to a specific network topology.

   (see "<graphics>" on page 266)

2. Optional: Use the listen element as child of the graphics element to specify where on the virtual server the GPU should listen to clients.

listen type attribute:	address
listen address attribute:	*<IP_address>*

   (see "<listen>" on page 277)

   <IP_address> specifies the IP address or host name on the virtual server at which the GPU listens to clients. The specification 0.0.0.0 configures all of the virtual server's IP addresses. This is the default.

3. Use the input element as child of the devices element to configure the workstation keyboard as an input device (see "<input>" on page 271).

input type attribute:	keyboard
input bus attribute:	virtio

4. Use the input element as child of the devices element to configure the workstation mouse as an input device.

input type attribute:	mouse
input bus attribute:	virtio

Example

```
<domain type="kvm">
   ...
   <devices>
      ...
      <graphics type="vnc" autoport="yes"
         <listen type="address" address="0.0.0.0"/>
      </graphics>
      <input type="keyboard" bus="virtio"/>
      <input type="mouse" bus="virtio"/>
      ...
   </devices>
   ...
</domain>
```

Configuring virtual SCSI devices

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

Before you begin

SCSI disks can be configured as pass-through devices (see "Virtual SCSI devices" on page 9), but the preferred configuration is as virtual block devices (see "Configuring virtual block devices" on page 103).

Procedure

1. Configure a virtual HBA.

   See "Configuring a virtual HBA" on page 115

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

   See one of the following:

   - "Configuring a SCSI tape or medium changer device" on page 117
   - "Configuring a virtual SCSI-attached CD/DVD drive" on page 122

Example

See "Example of a multipathed SCSI tape and medium changer device configuration" on page 120.

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 251).

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. For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

3. Optional: To attain good performance, assign an I/O thread for 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 258):

driver iothread attribute:	*<IOthread-ID>*

   <IOthread-ID> specifies the I/O thread to be used for I/O operations on all virtual SCSI devices that are 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.

See also "I/O threads" on page 207.

Example:

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

In this example, the I/O thread with ID 2 is assigned for I/O operations on the device.

4. 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>, and <memballoon>" on page 239)

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 8, "Preparing SCSI tape and medium changer devices," on page 37:

- 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 115), or
- Connect to a new virtual HBA which will be automatically configured for you.

About this task

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

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

Procedure

1. Configure the SCSI tape or medium changer device using the hostdev element (see "<hostdev>" on page 267).

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 238 and "<address> as child element of <source>" on page 244)

3. 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 240)

Where

<controller-index>

specifies the virtual HBA to which the SCSI device is connected.

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

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

<target>

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

<unit>

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

0 ≤ *<unit>* < 256

SCSI LUN := *<unit>*

256 ≤ *<unit>* ≤ 16383

SCSI LUN := 0x*<unit>* ∨ 0x4000

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

Example

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

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

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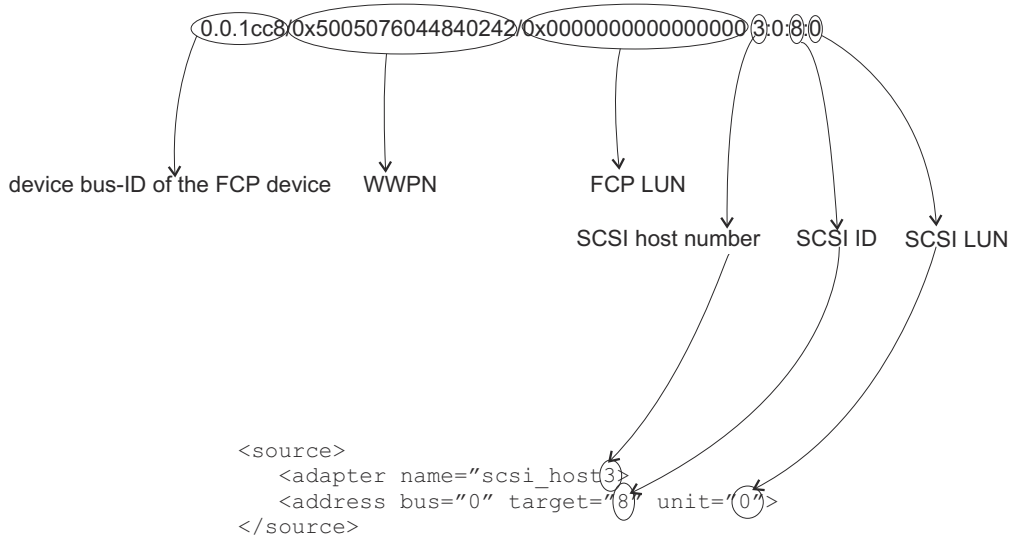
Assign a SCSI device name to the virtual SCSI device on the virtual server. The controller attribute of the address element refers to the index attribute of the controller element.

- 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="drive" 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 .

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="drive" 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="drive" 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="drive" 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="drive" 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 115), 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 255).

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 259).

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 301).

target bus attribute:	scsi

   d) Specify the virtual DVD as read-only using the readonly element (see "<readonly>" on page 291).

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 296). 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 301).

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 240)

Where

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

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

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

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

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

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

256 ≤ \<unit\> ≤ 16383
SCSI LUN := 0x*\<unit\>* v 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 184
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 10, "Preparing network devices," on page 43.

Procedure

- To configure a MacVTap interface, follow the steps described in "Configuring a MacVTap interface" on page 124.
- To configure a virtual switch, follow the steps described in "Configuring a virtual switch" on page 126

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 272).

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 279)

   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 299)

3. Specify the model type (see "<model> as a child element of <interface>" on page 286).

model type attribute:	virtio

4. For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

Example

- To configure bonded interface bond0:

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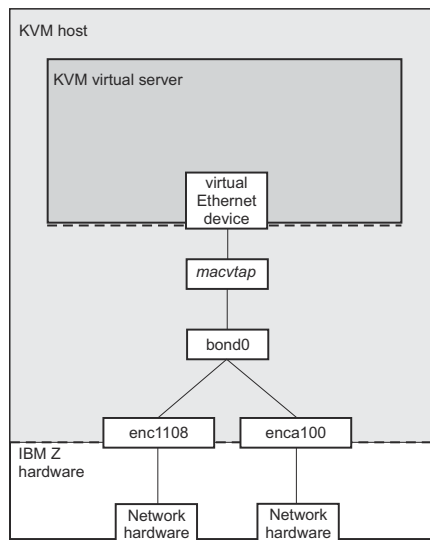

Figure 20. Direct interface type which configures a bonded interface

- To configure virtual LAN bond0.623:

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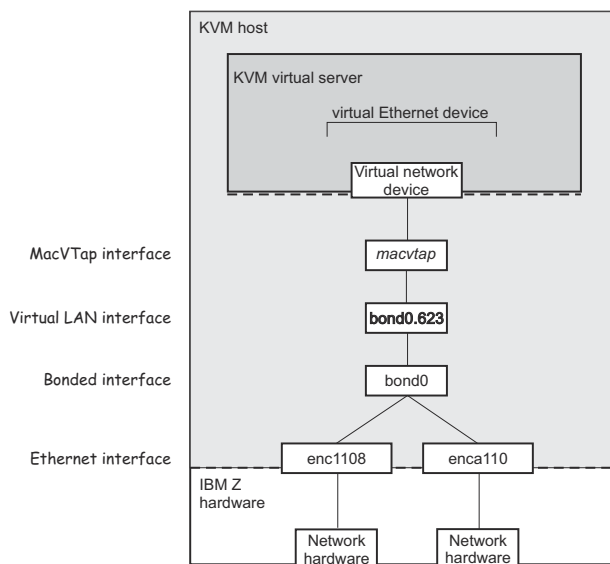

Figure 21. Direct interface type which configures a virtual LAN interface

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 272).

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 279)

2. Specify the virtual switch that you created before as described in "Preparing a virtual switch" on page 49.

source bridge attribute:	<vswitch>

   (see "<source> as child element of <interface>" on page 299)

3. Specify the type.

virtualport type attribute:	openvswitch

   (see "<virtualport> as a child element of <interface>" on page 304)

4. Specify the model type.

model type attribute:	virtio

   (see "<model> as a child element of <interface>" on page 286)

5. For guests that are to run in IBM Secure Execution mode and cannot use the launchSecurity element in the virtual server configuration, ensure that the device uses the guest's bounce buffer, see "Preparing the virtual server" on page 141.

Example

Display the available virtual switches:

```
# ovs-vsctl show
...
    Bridge "vswitch0"
        Port "vsbond0"
            Interface "enc1108"
            Interface "enca112"
        Port "vswitch0"
            Interface "vswitch0"
                type: internal
...
```

Configure the virtual switch which is shown in Figure 15 on page 25:

```
<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 "enc1108"
           Interface "enca112"
       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 backend element as child of the rng element to specify the device node of the input character device (see "<backend>" on page 246).

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>
```

Configuring VFIO devices

The VFIO framework can give guests direct access to specific host devices.

On the host, these VFIO pass-through devices are set up to be controlled by device-specific VFIO device drivers instead of their default device drivers. Depending on the device type, VFIO pass-through devices might require a VFIO mediated device that is based on host resources.

Configuring a pass-through DASD

In the configuration-XML, specify pass-through DASDs as CCW devices that are based on VFIO mediated devices on the host.

Before you begin

On the host, the vfio_ccw device driver must control the DASD, and a VFIO mediated device must be configured for it (see "Preparing DASD pass-through devices" on page 56).

You use the UUID of the mediated device to configure a pass-through DASD.

Procedure

1. Configure the device as a VFIO mediated device with the hostdev element (see "<hostdev>" on page 267).

hostdev mode attribute:	subsystem
hostdev type attribute:	mdev
hostdev model attribute:	vfio-ccw

2. Identify the device on the host with the address element as a child of the source element.

address uuid attribute:	*<uuid>*

   (see "<source> as child element of <hostdev>" on page 298 and "<address> as child element of <source>" on page 244)

3. Identify the DASD on the virtual server with the address element as a child of the hostdev element.

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

Example

```
<hostdev mode="subsystem" type="mdev" model="vfio-ccw">
   <source>
      <address uuid="90c6c135-ad44-41d0-b1b7-bae47de48627"/>
   </source>
   <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00a1"/>
</hostdev>
```

In the example, the UUID of the mediated device on the host is 90c6c135-ad44-41d0-b1b7-bae47de48627. On the guest, the bus ID of the DASD is 0.0.00a1.

Configuring pass-through PCI devices

Specify the PCI device as a VFIO device in the configuration-XML.

Before you begin

On the host, the PCI function must be controlled by the vfio-pci device driver before it can be used by a KVM guest. You can configure the device such that libvirt automatically effects this host setup while the device is claimed by a guest. Alternatively, you can perform the host setup as described in "Preparing PCI pass-through devices" on page 55.

Procedure

1. Configure the PCI device with the hostdev element (see "<hostdev>" on page 267).

hostdev mode attribute:	subsystem
hostdev type attribute:	pci
hostdev managed attribute:	yes|no

   The managed attribute is optional. If it is set to "yes", libvirt handles the host setup for the device for you.

   You can configure PCI networking devices with an interface element of type hostdev instead of using a hostdev element. Use this alternative if you must adhere to a particular network topology with specific values for MAC addresses or virtual ports. Apart from the mac and virtualport elements, the interface and hostdev elements have the same child elements (see "<interface>" on page 272.

2. Configure the PCI device as a VFIO device with the driver element (see "<driver>" on page 257).

driver name attribute:	vfio

3. Identify the device on the host with the address element as child of the source element (see "<source> as child element of <hostdev>" on page 298 and "<address> as child element of <source>" on page 244).

address domain attribute:	0x*<domain>*
address bus attribute:	0x*<bus>*
address slot attribute:	0x*<slot>*
address function attribute:	0x*<function>*

   Set the values of the domain, bus, slot, and function attributes such as to match the PCI function addresses of the form *<domain>.<bus>.<slot>.<function>* on the host.

   Tip: Issue `lspci` to list your PCIe devices with their function addresses.

   Example: To identify a PCI device with function address 0002:00:00.0, specify:

   ```
   <address domain="0x0002" bus="0x00" slot="0x00" function="0x0"/>
   ```

4. Optional: Identify the device on the guest with the address element as child of the source element. If you omit this specification, libvirt generates a valid, unique device address for you.

address type attribute:	pci

   QEMU needs valid settings for the PCI domain, bus, slot, and function, although these settings are not visible on the guest. Omit these specifications to make libvirt generate valid values for you. You can see the generated values in the expanded libvirt-internal configuration of the virtual server.

   (see "<address> as child element of <hostdev> or <disk>" on page 240)

5. Optional: Complement the device identification on the guest with identifiers that are specific to z/
   Architecture. These specifications are visible in the guest and help you to identify the device.

zpci uid attribute:	*\<uid>*
zpci fid attribute:	*\<fid>*

The uid attribute has a similar role to a domain on the host. Valid specifications for *\<uid>* are 4-digit
hexadecimal values in the range 0x0000 - 0xffff. Valid specifications for *\<fid>* are 8-digit hexadecimal
values in the range 0x00000000 - 0xffffffff.

(see "<zpci>" on page 306)

Example

```
<hostdev mode="subsystem" type="pci" managed="yes">
  <driver name="vfio"/>
  <source>
    <address domain="0x0002" bus="0x00" slot="0x00" function="0x0"/>
  </source>
  <address type="pci">
    <zpci uid="0x0001" fid="0x00000000"/>
  </address>
</hostdev>
```

Configuring cryptographic adapter resources

In the configuration-XML, specify a VFIO mediated device as the host device.

Before you begin

The host must relinquish some of its cryptographic resources and assign them to a VFIO mediated device (see "Preparing pass-through devices for cryptographic adapter resources" on page 58).

About this task

Linux on Z accesses cryptographic adapters through the zcrypt device driver and a generic device node. The cryptographic resources that are available through the device node depend on the configuration of the real or virtual hardware.

For a KVM guest, a subset of the host's cryptographic resources can be assigned to a VFIO mediated device, which is then passed through to the guest. VFIO mediated devices are identified by a UUID.

On the guest, cryptographic adapter resources are accessed through the generic device node, as usual. No guest device needs to be specified for these resources.

Procedure

1. Configure the device as a VFIO mediated device that uses the hostdev element (see "<hostdev>" on page 267).

hostdev mode attribute:	subsystem
hostdev type attribute:	mdev
hostdev model attribute:	vfio-ap

2. Identify the device on the host with the address element as child of the source element (see "<source> as child element of <hostdev>" on page 298 and "<address> as child element of <source>" on page 244).

address uuid attribute:	*<uuid>*

Example

```
<hostdev mode="subsystem" type="mdev" model="vfio-ap">
   <source>
     <address uuid="99e714ec-8eee-40fd-a26e-80ff3b1a2564"/>
   </source>
</hostdev>
```

In the example, the UUID of the mediated device on the host is 99e714ec-8eee-40fd-a26e-80ff3b1a2564.

Device configuration-XML

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

The element that delimits a device configuration within the domain configuration-XML file varies by device type. For example, this element can be hostdev, disk, controller, or interface. In the domain configuration-XML file, all these elements are specified as children of the devices element. In separate device configuration-XML files these elements are the root elements.

<disk>: Virtual block device
  Use the disk element to configure virtual block devices. On the host, the source could be a DASD, a SCSI disk, a logical volume, or a file in the host file system.

Root element
   disk

Selected child elements
   driver, source, target, address

Example: SCSI disk

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

Example: NVMe device

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/disk/by-path/pci-1003:00:00.0-nvme-1"/>
    <target dev="vdc" bus="virtio"/>
</disk>
```

Related information:
   "Configuring virtual block devices" on page 103

<hostdev>: VFIO or SCSI pass-through devices
   Use the hostdev element to configure host resources as pass-through devices. SCSI disks can be
   configured as pass-through devices, but the preferred configuration is as virtual block devices. Do not
   configure VFIO cryptographic adapter resources as hotplug devices.

Root element
   hostdev

Selected child elements
   source, address

Example: VFIO pass-through DASD

```
<hostdev mode="subsystem" type="mdev" model="vfio-ccw">
    <source>
        <address uuid="90c6c135-ad44-41d0-b1b7-bae47de48627"/>
    </source>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00a1"/>
</hostdev>
```

Example: pass-through device for cryptographic adapter resources

```
<hostdev mode="subsystem" type="mdev" model="vfio-ap">
    <source>
        <address uuid="99e714ec-8eee-40fd-a26e-80ff3b1a2564"/>
    </source>
</hostdev>
```

Example: pass-through SCSI

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

Related information:

   "Configuring a pass-through DASD" on page 130
   "Configuring pass-through PCI devices" on page 131
   "Configuring virtual SCSI devices" on page 115

<controller>: Virtual host bus adapter
   Use the controller element to configure a virtual host bus adapter for pass-through SCSI devices.

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>
```

Related information:

"Configuring virtual SCSI devices" on page 115

<interface>: Virtual Ethernet devices

Root element

interface

Selected child elements

mac, source, model

Example

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

Related information:

"Configuring virtual Ethernet devices" on page 124

Chapter 14. 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 331
Is the root element of a storage pool configuration-XML.

"<volume>" on page 340
Is the root element of a volume configuration-XML.

Chapter 15. Configuring virtual networks

Use the network configuration-XML to configure virtual networks that connect KVM virtual servers among themselves and to an external network.

KVM hosts on IBM Z support networks with three types of Linux bridges. All types make a communication setup addressable as a network or bridge.

- Bridge with network address translation (NAT)
- Open vSwitch bridge
- Bridge with IP routing

Each bridge type has a different forwarding mode as specified with the <forward> element. Omitting the <forward> element results in a virtual network among the virtual servers, without a connection to a physical network.

Bridge with network address translation (NAT)

With network address translation, traffic of all virtual servers to the physical network is routed through the host's routing stack and uses the host's public IP address. This type of network supports outbound traffic only.

Forwarding mode
    nat

Example

```
<network>
  <name>net0</name>
  <uuid>fec14861-35f0-4fd8-852b-5b70fdc112e3</uuid>
  <forward mode="nat">
    <nat>
      <port start="1024" end="65535"/>
    </nat>
  </forward>
  <bridge name="virbr0" stp="on" delay="0"/>
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

Open vSwitch bridge

With an Open vSwitch bridge, the switch implements a subnet. The <bridge> element must reference an already existing Open vSwitch (see).

Forwarding mode
    bridge

Example

```
<network>
  <name>ovs</name>
  <uuid>58681f9f-20e1-4673-97a0-5c819660db3e</uuid>
  <forward mode="bridge"/>
  <bridge name="ovs-br0"/>
  <virtualport type="openvswitch"/>
</network>
```

Bridge with IP routing

Bridges with IP routing link to a virtual IP subnet on the host. Traffic to and from virtual servers that are connected to that subnet are then handled by the IP protocol.

Forwarding mode
　　route
Example

```
<network>
  <name>net1</name>
  <uuid>34fc97f4-86c5-4d65-887a-cc8b33d2a260</uuid>
  <forward mode="route"/>
  <bridge name="iedn" stp="off" delay="0"/>
  <mac address="f6:2b:85:a9:bf:d9"/>
  <ip address="198.51.100.1" netmask="255.255.255.0">
  </ip>
</network>
```

Related reference

"<bridge>" on page 308
Configures the bridge device that is used to set up the virtual network.

"<dhcp>" on page 309
Configures DHCP services for the virtual network.

"<forward>" on page 310
Configures the forwarding mode for the bridge that connects the virtual network to a physical LAN.
Omitting this tag results in an isolated network that can connect guests.

"<ip>" on page 311
Configures IP addresses for the virtual network.

"<name> as a child element of <network>" on page 312
Assigns a short name to a virtual network.

"<network>" on page 313
Is the root element of a network configuration-XML.

Chapter 16. Configuring for IBM Secure Execution for Linux

To support guests in IBM Secure Execution mode, the configuration of a virtual server must be compatible with IBM Secure Execution for Linux.

In particular, memory access by virtio devices must be regulated through IOMMU. To prevent IOMMU bypass, guests that are set up for IBM Secure Execution mode provide a bounce buffer that all virtio devices of the virtual server must use. For information about configuring the bounce buffer within the guest, see *Introducing IBM Secure Execution for Linux*, SC34-7721.

- The preferred method for enabling virtio devices to use the bounce buffer is to use a generic specification for the virtual server, see "Preparing the virtual server" on page 141.
- The fallback method is to enable each virtio device separately, see "Enable each device separately to use the bounce buffer" on page 141.

For configuration items that can lead to malfunctioning devices or prevent the guest from running in IBM Secure Execution mode, see "Omit items that conflict with IBM Secure Execution for Linux" on page 143.

Related concepts
"IBM Secure Execution for Linux" on page 27

Preparing the virtual server

Add the launchSecurity element with type s390-pv to the domain configuration-XML of your virtual server to set defaults that simplify configuring the virtual server for IBM Secure Execution for Linux.

```
<domain type="kvm">
    ...
    <launchSecurity type="s390-pv"/>
    ...
</domain>
```

For example, this setting makes the required bounce buffer for virtio devices the default and you do not have to specify it explicitly for each device, see "Enable each device separately to use the bounce buffer" on page 141. This setting also leads to warning messages if the CPU model of the virtual server does not include all features that are required by IBM Secure Execution for Linux.

To confirm that this setting is available in your environment, look for the following line in the output of the **virsh domcapabilities** command:

```
<s390-pv supported="yes">
```

Alternatively, you can use the **virt-host-validate** command and look for the following output line:

```
QEMU: Checking for secure guest support                    : PASS
```

The **virt-host-validate** command generally checks the host requirements for IBM Secure Execution for Linux, see Chapter 5, "IBM Secure Execution for Linux," on page 27.

Enable each device separately to use the bounce buffer

You can enable each virtio device separately to use the bounce buffer.

Before you begin: Preferably, enable all virtio devices of your virtual server by default, see "Preparing the virtual server" on page 141. Enable individual virtio devices to use the bounce buffer only as a fallback method.

Important: With the bounce buffer as a default, do not override this setting by configuring `iommu="off"` for any virtio device.

If you must enable individual virtio devices to use the bounce buffer, do not overlook virtio SCSI devices, serial devices, or hotplug devices that are configured in separate device configuration-XML files.

Tip: Scan the libvirt-internal configuration for address elements that have a type, cssid, ssid, and devno attribute, where the type attribute has a value `ccw`, as in the following example:

```
<address type="ccw" cssid="0xfe" ssid="0x1" devno="0xe714"/>
```

To enable the bounce buffer for an individual device, specify the `iommu="on"` attribute for the driver element that is nested within the element that represents the virtio device. You might have to add the driver element.

Example for disk devices

For disk devices, add the `iommu="on"` attribute to the existing driver element.

```
<domain>
    ...
    <devices>
        ...
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native"
                    iothread="1" iommu="on"/>
            ...
        </disk>
        ...
    </devices>
    ...
</domain>
```

Example for network interfaces

Network interface devices might not have a driver element. If necessary, add one with the `iommu="on"` attribute.

```
<domain>
    ...
    <devices>
        ...
        <interface type="direct">
            <source dev="bond0" mode="bridge"/>
            <model type="virtio"/>
            <driver iommu="on"/>
        </interface>
        ...
    </devices>
    ...
</domain>
```

Example for SCSI devices

Individual SCSI device configurations reference a SCSI controller element. You enable the bounce buffer for the controller element to cover all associated SCSI devices.

Specify the `iommu="on"` attribute for a driver element that is nested in the controller element. You might have to split the controller element into a start and end tag to insert the driver element.

```
<domain>
    ...
    <devices>
        ...
        <controller type="scsi" model="virtio-scsi" index="0">
            <driver iommu="on"/>
        </controller>
        ...
        <disk type="file" device="cdrom">
```

```
            ...
            <address type="drive" controller="0" bus="0" target="0" unit="0"/>
            ...
        </disk>
        ...
    </devices>
    ...
</domain>
```

Example for serial devices

```
<controller type="virtio-serial" index="0">
    <driver iommu="on"/>
</controller>
```

Virtio hotplug devices must also use the bounce buffer. Always use device configuration-XML files to enable such devices and use **virsh attach-device** to attach the device to a running virtual server. Attaching devices with **virsh attach-disk** is likely to crash the virtual server.

Omit items that conflict with IBM Secure Execution for Linux

Do not configure the virtual server with devices and settings that are incompatible with IBM Secure Execution for Linux.

Omit VFIO pass-through devices

The domain configuration-XML must not configure any VFIO pass-through devices such as: VFIO DASD, VFIO PCI devices, or cryptographic resources that are set up and configured as VFIO devices.

Tip: Scan the domain configuration-XML for "vfio". In particular, there must be no driver elements with name="vfio" and no hostdev elements with model="vfio-ccw".

Do not back your guest memory with huge pages

The domain configuration-XML must not include the hugepages element.

Part 4. Operation

Manage the operation of virtual servers using virsh commands.

Chapter 17. 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 148.

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 148.

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 149.

Related reference
"Selected virsh commands" on page 341
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 350):

  ```
  # 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 156) 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 158.

3. Start the virtual server as described in "Starting a virtual server" on page 152.

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 185.

Related reference
"Virtual server life cycle" on page 229
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 364):

  ```
  # 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 152 and "Starting a virtual server" on page 152).

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 393):

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

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

Chapter 18. 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 373) 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 148.

About this task

- "Starting a virtual server" on page 152

  Start a defined virtual server.

- "Terminating a virtual server" on page 152

  Properly shut down a virtual server, save a system image, or, if necessary, immediately terminate it.

- "Suspending a virtual server" on page 154

  Pause a virtual server.

- "Resuming a virtual server" on page 154

  Transfer a paused virtual server to the running state.

Related reference
"Virtual server life cycle" on page 229
Display the state of a defined virtual server including the reason with the virsh **domstate --reason** command.

"Selected virsh commands" on page 341
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" on page 152).

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 35, "Virtual server life cycle," on page 229 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 390):

```
# 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 35, "Virtual server life cycle," on page 229 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 387	

Description	Command	Comments
To save a system image and terminate a virtual server properly:	"managedsave" on page 375	
To terminate a virtual server immediately:	"destroy" on page 351	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 35, "Virtual server life cycle," on page 229 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 392):

```
# 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 35, "Virtual server life cycle," on page 229 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 385):

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

<VS>
> Is the name of the virtual server.

Chapter 19. 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 373) 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 148.

About this task

- "Browsing virtual servers" on page 156

   View lists of all defined or of all running virtual servers.

- "Displaying information about a virtual server" on page 156

   View information about a virtual server, its state, its devices, or scheduling properties.

- "Displaying the current libvirt-internal configuration" on page 158

   The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, complemented with libvirt-internal information, and modified as devices are attached or detached.

Related reference
"Selected virsh commands" on page 341
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 373):

```
# 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 359	
Current state	"domstate" on page 361	Display the reason of the current state by using the - -reason option.
Scheduling information	"schedinfo" on page 386	
Number of virtual CPUs	"vcpucount" on page 394	
Virtual block devices	"domblkstat" on page 355	To retrieve the device name, use the virsh **domblklist** command.
Virtual Ethernet interfaces	"domifstat" on page 358	To retrieve the interface name, use the virsh **domiflist** command.
I/O threads	"iothreadinfo" on page 372	

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-36005076305ffc1ae00000000000023bc

# virsh domblkstat vserv1 /dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc rd_req 17866
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc rd_bytes 180311040
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc wr_req 11896
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc wr_bytes 126107648
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc flush_operations 3884
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc rd_total_times 14496884715
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc wr_total_times 9834388979
/dev/disk/by-id/dm-uuid-mpath-36005076305ffc1ae00000000000023bc flush_total_times 755568088
```

• View information about the virtual Ethernet interfaces:

```
# virsh domiflist vserv1
Interface  Type       Source    Model      MAC
-------------------------------------------------------
vnet0      network    iedn      virtio     02:17:12:01:ff:01

# virsh domifstat vserv1 vnet0
vnet0 rx_bytes 2377970
vnet0 rx_packets 55653
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 831453
vnet0 tx_packets 18690
vnet0 tx_errs 0
vnet0 tx_drop 0
```

- View information about the I/O threads of a virtual server with 8 virtual CPUs:

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

Displaying the current libvirt-internal configuration

The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, complemented with libvirt-internal information, and modified as devices are attached or detached.

Procedure

- To display the current libvirt-internal configuration of a defined virtual server, use the virsh **dumpxml** command (see):

```
# virsh dumpxml <VS>
```

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

Example

Domain configuration-XML file vserv1.xml configures virtual server vserv1:

vserv1.xml

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

  <os>
    <type arch="s390x" machine="s390-ccw-virtio">hvm</type>
  </os>
  <iothreads>2</iothreads>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>preserve</on_crash>
  <devices>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
    </disk>
    <interface type="direct">
      <source dev="bond0" mode="bridge"/>
```

```
        <model type="virtio"/>
      </interface>
      <console type="pty">
        <target type="sclp"/>
      </console>
      <memballoon model="none"/>
    </devices>
  </domain>
```

Device configuration-XML file `dev1.xml` configures a separate device:

dev1.xml

```
<disk type="block" device="disk">
   <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
   <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
   <target dev="vdb" bus="virtio"/>
</disk>
```

You can define and start the virtual server and then attach the configured device with the commands:

```
# virsh define vserv1.xml
# virsh start vserv1 --console
# virsh attach-device vserv1 dev1.xml
```

The virsh **dumpxml** command displays the current libvirt-internal configuration, as for example:

```
# virsh dumpxml vserv1
<domain type="kvm">
  <name>quickstart1</name>
  <uuid>4a461da8-0253-4989-b267-bd4db02bfac4</uuid>
  <memory unit="KiB">4194304</memory>
  <currentMemory unit="KiB">4194304</currentMemory>
  <vcpu placement="static">2</vcpu>
  <iothreads>2</iothreads>
  <os>
    <type arch="s390x" machine="s390-ccw-virtio-4.2">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-system-s390x</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0000"/>
    </disk>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
      <target dev="vdb" bus="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0002"/>
    </disk>
    <interface type="direct">
      <mac address="52:54:00:6a:0b:53"/>
      <source dev="bond0" mode="bridge"/>
      <model type="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0001"/>
    </interface>
    <console type="pty">
      <target type="sclp" port="0"/>
    </console>
    <memballoon model="none"/>
  </devices>
</domain>
```

libvirt added a number of XML elements to the current representation of the virtual server configuration. They are shown in **bold**:

• A UUID for the virtual server.

• The current machine type, which depends on the host setup and distribution.

- For devices, the emulator, MAC addresses and address elements.

Chapter 20. 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 162

  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 163

  To provide high availability, for example during a hypervisor upgrade, you can migrate a running virtual server to a different host. You can decide whether or not the virtual server will be removed from the source host after the migration, so you can use live migration as a deployment vehicle, too.

  "Live" migration also provides means to migrate virtual servers offline. This option might be interesting for you if you want to deploy virtual servers and benefit from the migration automatism.

Migration to a different hypervisor release

The hypervisor release is defined by the installed QEMU release, by the hypervisor product or by your distribution on the host.

The virtual server's machine type determines which hypervisor release runs the virtual server on the host.

Be sure to configure the machine type with the alias value "s390-ccw-virtio" in the domain configuration-XML unless you intend to migrate the virtual server to a destination host with an earlier hypervisor release.

Definition of a virtual server on different hosts using the same configuration-XML

Deploy virtual servers by copying their domain configuration-XML files to different hosts and defining them afterwards.

When you define a virtual server using the alias machine type, libvirt replaces the alias machine type by the machine type which reflects the current hypervisor release of the host running the virtual server. The libvirt-internal configuration reflects the installed hypervisor release.

Example:

Domain configuration-XML using the alias machine type:
```
<type arch="s390x" machine="s390-ccw-virtio">hvm</type>
```

Libvirt-internal configuration for QEMU release 5.0:
```
<type arch="s390x" machine="s390-ccw-virtio-5.0">hvm</type>
```

Depending on your distribution, there may be additional machine types. The following command displays the available machine types:

```
# qemu-kvm --machine help
```
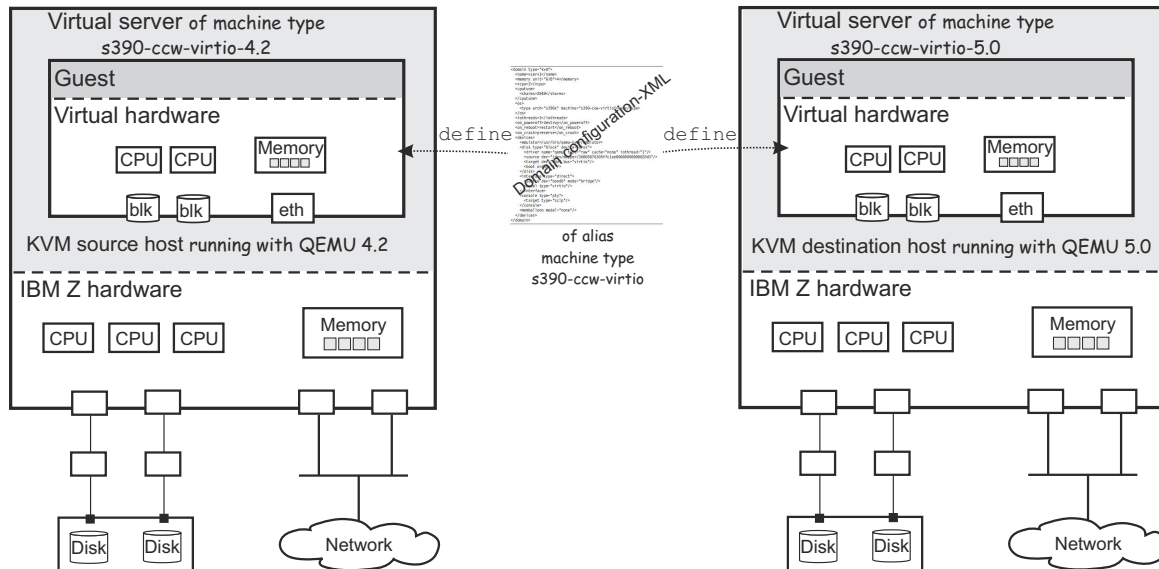

Figure 22. Defining virtual servers on different hosts

shows that creating virtual servers from the same domain configuration-XML file on different hosts results in different machine types.

Live virtual server migration

Migrate a running virtual server from one host to another without affecting the virtual server. The literature also uses the terms "virtual server, virtual machine, or guest *relocation*".

Hypervisor release

A live virtual server migration preserves the machine type of the virtual server.

The libvirt-internal configuration is not changed, that is, the machine type still reflects the hypervisor release of the source host. Newer hypervisor releases are compatible with earlier versions.

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

Example:

1. Before the migration, the virtual server is running on the source host with a hypervisor release based on QEMU 4.2. The virtual server's machine type is s390-ccw-virtio-4.2.
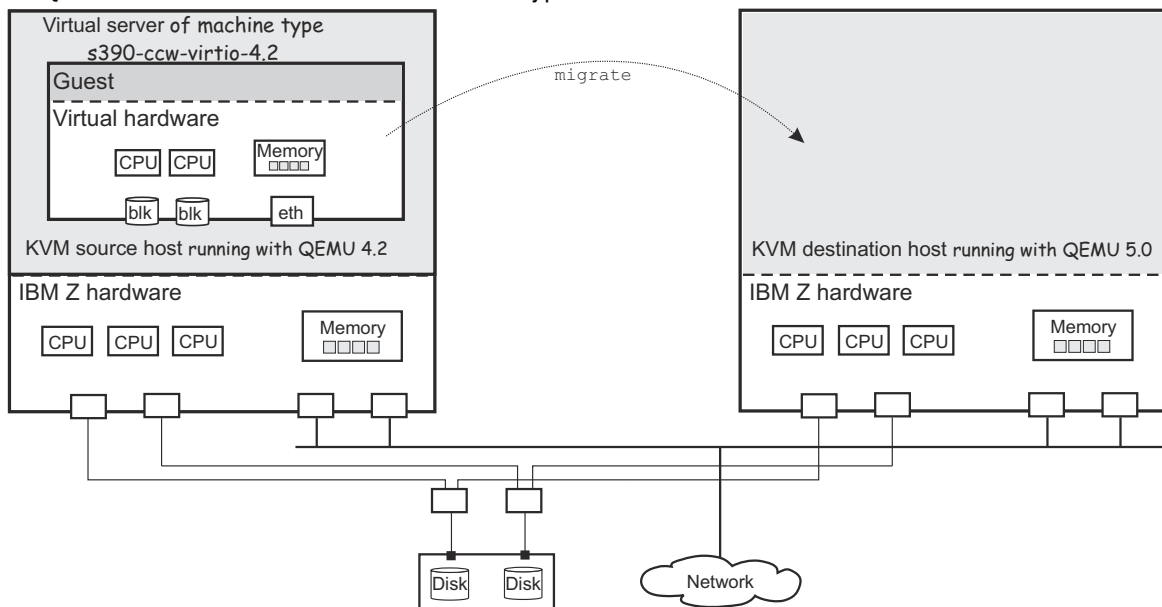
2. After the migration, the virtual server is running on the destination host with a hypervisor release based on QEMU 5.0. The virtual server's machine type is still s390-ccw-virtio-4.2.
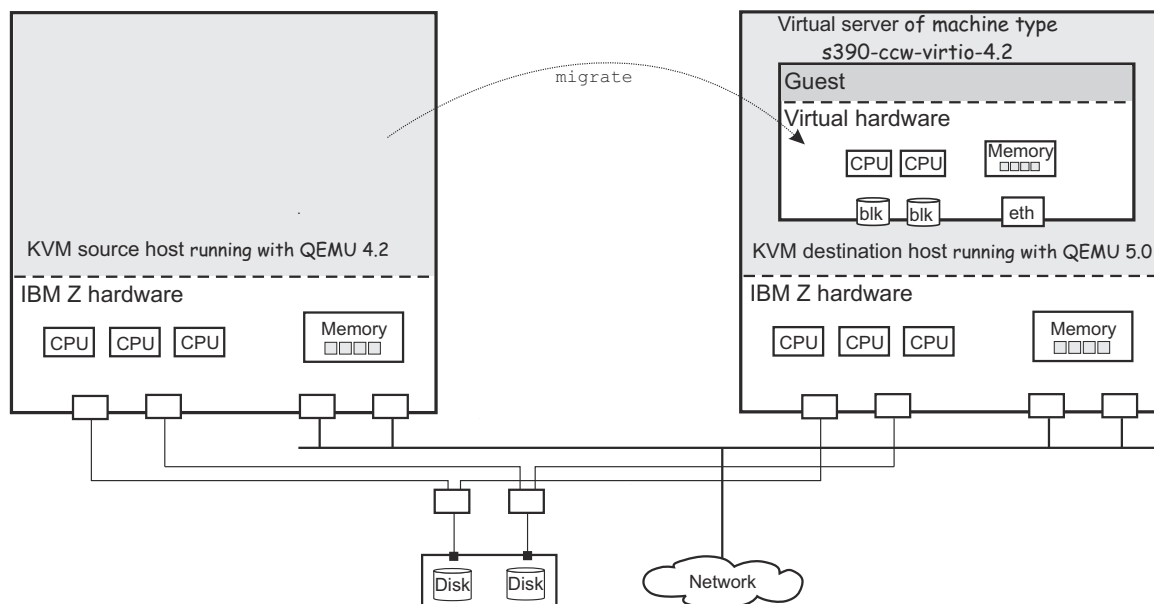
The virtual server runs on the earlier hypervisor release and does not exploit the features of the current release.

As long as you do not change the machine type to the new release, a migration of this virtual server back to its original source host will succeed.

IBM Z hardware CPU model

The destination host must offer the same or a later CPU model than the CPU model that is used on the original host.

You can perform virtual server live migrations across IBM Z hardware of the same model and upgrade level. You can also migrate to a later hardware model, for example from IBM z14 to IBM z15 hardware.

Migration to a prior hardware model is possible only if the virtual server on the newer hardware is restricted to CPU features that are also available on the older destination hardware. By default, a virtual server uses the latest CPU model of the hardware. Use the <cpu> element in the domain XML to configure a specific backlevel CPU model (see "Configuring the CPU model" on page 87).

After a live migration to a later hardware model, the virtual server keeps running with the CPU model of the original hardware. This behavior preserves the option for a live migration back to the original hardware. To use new CPU features on the destination hardware, stop the virtual server, modify the domain configuration-XML, and then restart the virtual server.

Regardless of the destination hardware, live guest migration is possible only to KVM hosts with CPU model support. Live migration is not possible for guests in IBM Secure Execution mode.

If you cannot perform live guest migration, migrate by shutting down the virtual server and then starting it on the destination hardware (see "Definition of a virtual server on different hosts using the same configuration-XML" on page 162).

Example

The following figure illustrates the rules for a live migration. A virtual server on z14 hardware runs a guest operating system with the CPU features of z13 with upgrade level 2.
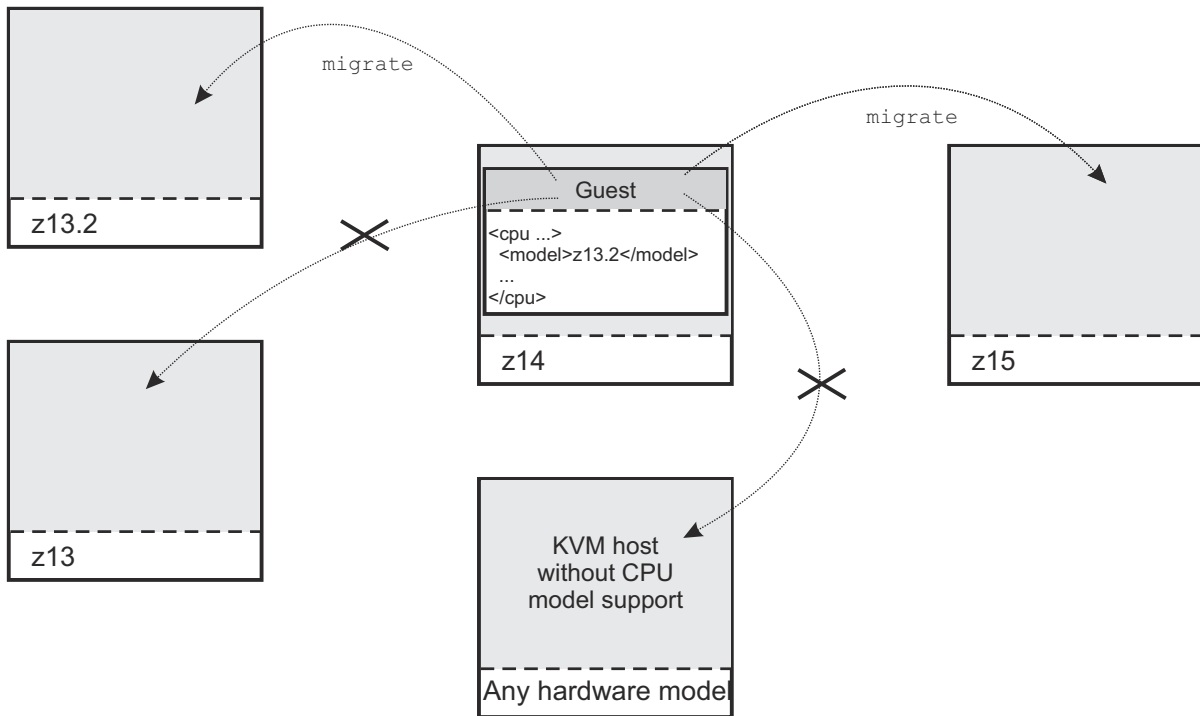
- Guest live migration is possible to z15 hardware.
- Guest live migration is possible to z13 hardware with upgrade level 2.
- Guest live migration is not possible to z13 hardware with less than upgrade level 2.

Confirming the CPU model of a destination host

Use the **virsh hypervisor-cpu-compare** command to confirm that the CPU model of a particular destination host satisfies the CPU requirements of a virtual server.

As command input, use an XML document that describes the CPU requirements of the virtual server that you want to migrate. For example, use the output of the **domcapabilities** command.

Run the **hypervisor-cpu-compare** command on the destination host.

For more information about the **hypervisor-cpu-compare** command, see "hypervisor-cpu-compare" on page 367.

Establishing a baseline CPU model

Use the **virsh hypervisor-cpu-baseline** command to establish a CPU model that is supported across a set of KVM hosts. The CPU requirements for any virtual server that uses this baseline model are then met by all hosts within the set.

As command input, use an XML document that contains a concatenation of CPU model descriptions, one for each host in the set. Descriptions must adhere to the syntax of the <cpu> element of a domain configuration-XML. To obtain a valid CPU description for a particular KVM hypervisor, issue the **domcapabilities** command on that hypervisor. Concatenate the output of the **domcapabilities** command from all KVM hypervisors for which you want to find a baseline CPU model. Optionally, you can reduce the information for each hypervisor to the CPU snippet.

Important: Issue the **hypervisor-cpu-baseline** command on the host with the most advanced CPU model within the set. The command might fail if the input XML file contains a CPU description that is ahead of the CPU model of the host that runs the command.

For more information about the **hypervisor-cpu-baseline** command, see "hypervisor-cpu-baseline" on page 365.

Live migration setup

To perform a live migration, the source and destination hosts must be connected and must have access to the same or equivalent system resources, the same storage devices and networks.

Preservation of the virtual server resources

Prepare a migration carefully to preserve the resources of the virtual server.

System resources
Provide access to the same or equivalent system resources, such as memory and CPUs, on both hosts.

Storage
Storage devices that are configured for the virtual server must be accessible from the destination host.

DASDs:

- Make sure that DASDs are configured using udev-created device nodes.
- If the DASDs are configured using the device bus-ID (by-path device node), make sure that you use identical device numbers in the IOCDS of both hosts.
- Make sure that there is a migration process for setting both the base devices and the alias devices online on the destination host.

SCSI disks:

- Make sure that SCSI disks are configured using device mapper-created device nodes.

Image files residing on a network file system (NFS):

- Make sure that both hosts have a shared access to the image files.

  If Security-Enhanced Linux (SELinux) is enabled on the destination host, using the following command can provide access to the NFS:

  ```
  # setsebool -P virt_use_nfs 1
  ```

  Please note that depending on the NFS configuration the image files could be accessible by other virtual servers.

Virtual block devices that are backed by host resources:
Use the *disk migration* options to migrate virtual block devices that are backed by local resources of the source host. Image files in the host file system and PCIe-attached NVMe devices are examples of such local resources.

Live migration has the following requirements for disk migration for each resource:

- The destination host must provide an equivalent local resource. The local resources on the source and destination host must be addressed through the same configuration.
- The virtual block devices must not be configured to use an I/O thread.
- The resource must be writable. For example, to be eligible for disk migration, an image file must not be configured as a virtual DVD.

For details about disk migration, see "3.c" on page 171.

SCSI tapes or medium changer devices:

- When you migrate a virtual server that uses a configured virtual SCSI device, be aware that the SCSI device name, which is used to specify the source device, might change on the destination host.

  Tip: Make sure that SCSI tapes or medium changer devices are configured in separate device configuration-XML files. Detach them before you perform a migration. After the migration, reconfigure the devices before you reattach them.

"Disk device identification" on page 12 and "SCSI device identification" on page 20 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
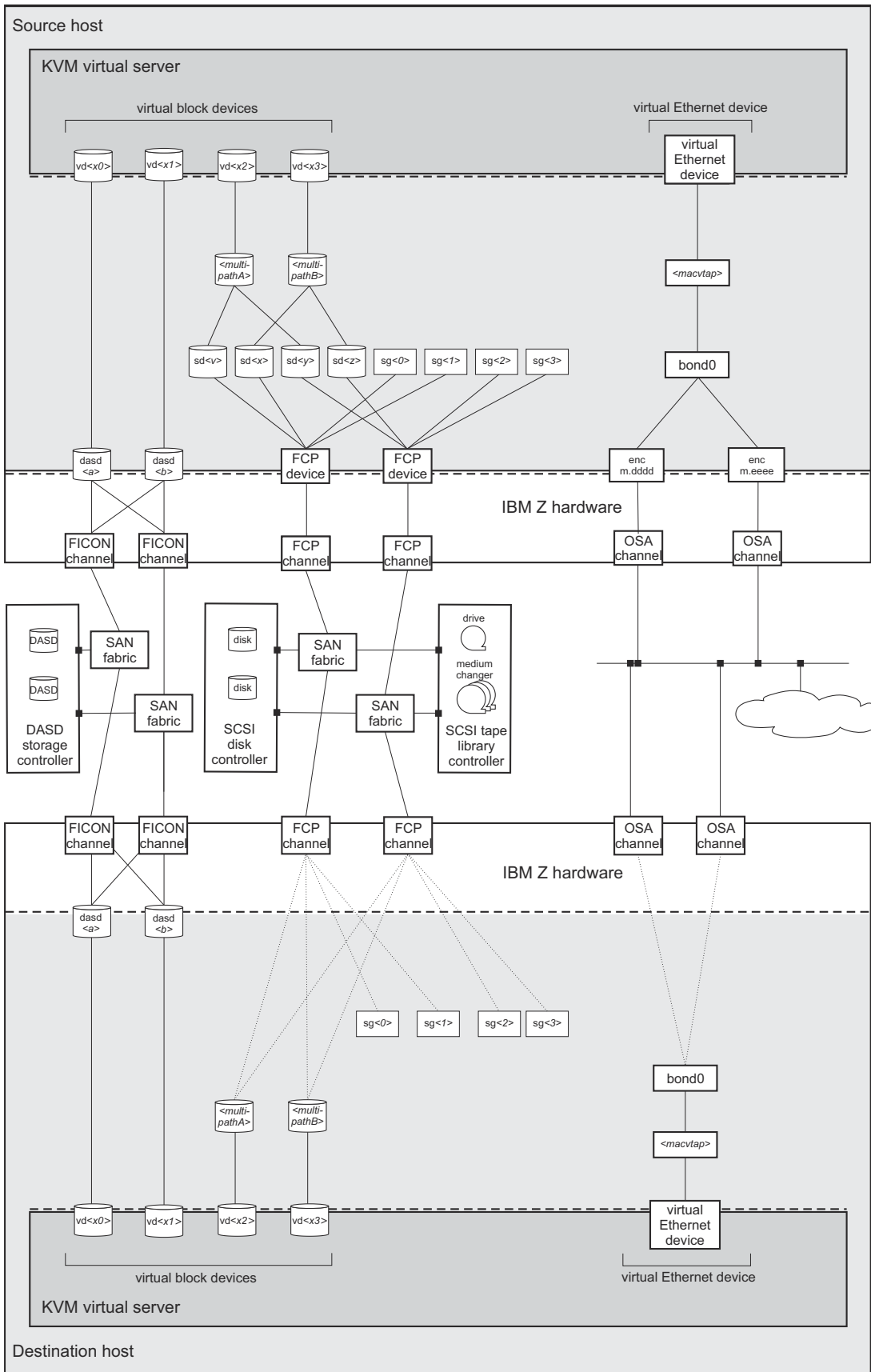
Figure 23. Example of a device setup on the source and destination hosts that allows the migration of the virtual server using these devices

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" on page 169).

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 382). 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 383):

   ```
   # 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 381):

```
# 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 378):

```
# 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 37, "Selected virsh commands," on page 341):

```
# virsh --keepalive-interval <interval-in-seconds>
```

The use of the virsh `--keepalive-interval` and `--keepalive-count` options preserves the communication connection between the host that initiates the migration and the libvirtd service on the source host during time-consuming processes.

Use the keepalive options if:

- The virtual server is running a memory intensive workload, so that it might need to be suspended to complete the migration.
- You make use of an increased timeout interval.

Defaults:

keepalive interval	5 seconds
keepalive count	6

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

Example:

```
# virsh --keepalive-interval 10 migrate --live --persistent --undefinesource \
--timeout 1200 --verbose vserv1 qemu+ssh://kvmhost/system
```

This example increases the keepalive interval of the connection to the host to 10 seconds.

c) Optional: Perform *disk migration* for any virtual block devices that are backed by local resources on the source host.

Such local host resource can be, for example, image files in the host file system or PCIe-attached NVMe devices.

Specify the option `--copy-storage-all` or `--copy-storage-inc` in combination with the option `--migrate-disks` to copy image files or file systems on NVMe 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, 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>
```

i) Create a qcow2 image on the destination host:

```
[root@destination]# qemu-img create -f qcow2 \
/var/lib/libvirt/images/vdd.qcow2 1G
```

ii) 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 21. 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 373) 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 148.

About this task

- "Managing virtual CPUs" on page 176

  Modify the portion of the run time that is assigned to the CPUs of a defined virtual server.

- "Managing virtual memory" on page 180

  Restrict the amount of physical memory used by a virtual server.

  Related reference
  "Selected virsh commands" on page 341
  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" on page 176

  describes how to modify the number of virtual CPUs of a running virtual server.

- "Modifying the virtual CPU weight" on page 179

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

Related concepts
"CPU management" on page 199
Virtual CPUs are realized as threads within the host, and scheduled by the process scheduler.

Related tasks
"Configuring virtual CPUs" on page 85
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 388).

- Modify **maximum config**:

    To modify the maximum number of available virtual CPUs with the next virtual server restart, use the `--maximum` and the `--config` options:

    ```
    # virsh setvcpus <VS> <max-number-of-CPUs> --maximum --config
    ```

    This modification takes effect after the termination of the virtual server and a subsequent restart. Please note that a virtual server reboot does not modify the libvirt-internal configuration.

- Modify **current config:**

    To increase or reduce the number of virtual CPUs with the next virtual server restart, use the `--config` option:

    ```
    # virsh setvcpus <VS> <number-of-CPUs> --config
    ```

    The virtual CPUs are not removed until the next virtual server reboot. Until then, the virtual server user might set the corresponding number of virtual CPUs offline.

- Modify **current live:**

    To increase the number of virtual CPUs of a running or paused virtual server, use the `--live` option:

    ```
    # virsh setvcpus <VS> <number-of-CPUs> --live
    ```

    The virtual server user has to bring the additional virtual CPUs online.

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

    <max-number-of-CPUs>
        Is the maximum number of available virtual CPUs for the virtual server after the next restart.

    <number-of-CPUs>
        Is the number of virtual CPUs assigned to the virtual server.

Example

- Change the maximum number of available virtual CPUs with the next virtual server restart.

```
# virsh vcpucount vserv1
maximum     config          5
maximum     live            5
current     config          4
current     live            4

# virsh setvcpus vserv1 6 --maximum --config

# virsh vcpucount vserv1
maximum     config          6
maximum     live            5
current     config          4
current     live            4
```

- You cannot remove virtual CPUs from a running virtual server.

    1. This example removes two virtual CPUs from the virtual server vserv1 with the next virtual server restart:

    ```
    # virsh vcpucount vserv1
    maximum     config          5
    maximum     live            5
    current     config          4
    current     live            4

    # virsh setvcpus vserv1 2 --config

    # virsh vcpucount vserv1
    maximum     config          5
    maximum     live            5
    current     config          2
    current     live            4
    ```

    2. To set the CPUs offline until the next virtual server restart, the virtual server user might set the virtual CPUs offline:

    ```
    [root@guest:] # chcpu -d 2
    CPU 2 disabled
    [root@guest:] # chcpu -d 3
    CPU 3 disabled
    ```

- Add virtual CPUs to a running virtual server.

    1. This example adds a virtual CPU to the virtual server vserv1:

    ```
    # virsh vcpucount vserv1
    maximum     config          5
    maximum     live            5
    current     config          3
    current     live            3

    # virsh setvcpus vserv1 4 --live

    # virsh vcpucount vserv1
    maximum     config          5
    maximum     live            5
    current     config          3
    current     live            4
    ```

    2. To set the additional CPU online, the virtual server user might enter:

    ```
    [root@guest:] # chcpu -e 3
    CPU 3 enabled
    ```

Modifying the virtual CPU weight

Modify the share of run time that is assigned to a virtual server.

About this task

The available CPU time is shared between the running virtual servers. Each virtual server receives the share that is configured with the shares element, or the default value.

To display the current CPU weight of a virtual server, enter:

```
# virsh schedinfo <VS>
```

You can modify this share for a running virtual server or persistently across virtual server restarts.

Procedure

- To modify the current CPU weight of a running virtual server, use the virsh **schedinfo** command with the --live option (see "schedinfo" on page 386):

```
# 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 86
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 377):

```
# 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

"Memory management" on page 203
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 89
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 22. 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 373) 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 148.

About this task

- "Attaching a device" on page 182

  Attach a device to a virtual server. If the virtual server is running, you can hotplug the device.
- "Detaching a device" on page 183

  Detach a device from a virtual server. If the virtual server is running, you can unplug the device.
- "Replacing a virtual DVD" on page 184

  Remove the currently provided ISO image, or provide a different one.
- "Connecting to the console of a virtual server" on page 185

  Connect to the console of a virtual server.

Related reference
"Selected virsh commands" on page 341
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Attaching a device

You can hotplug devices to a running virtual server, add devices to the persistent virtual server configuration, or both.

Before you begin

- Ensure that the new device is not already assigned to the virtual server.

  To list the devices that are assigned to a virtual server, you can

  – Display the current libvirt-internal configuration.

  – Use the virsh **domblklist** command to display a list of currently assigned block devices or the virsh **domiflist** command to display a list of currently assigned interface devices.

- You need a device configuration-XML file for the device.

Procedure

1. Optional:

   If you attach a virtual block device, and the current libvirt-internal configuration does not provide an I/O thread for the device:

   Add an I/O thread dedicated to the device by using the virsh **iothreadadd** command (see "iothreadadd" on page 370):

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

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

   <IOthread-ID>
   : Is the ID of the I/O thread to be added to the virtual server. Be sure that the I/O thread ID matches the I/O thread ID in the device configuration-XML (see "Configuring a DASD, SCSI, or NVMe disk" on page 103).

2. Attach the device using the virsh **attach-device** command (see "attach-device" on page 345).

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

   <device-configuration-XML-filename>
   : Is the name of the device configuration-XML file.

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

   <scope>
   : Specifies the scope of the command:

     --live
     : Hotplugs the device to a running virtual server. This configuration change does not persist across stopping and starting the virtual server.

     --config
     : Adds the device to the persistent virtual server configuration. The device becomes available when the virtual server is next started. This configuration change persists across stopping and starting the virtual server.

     --persistent
     : Adds the device to the persistent virtual server configuration and hotplugs it if the virtual server is running. This configuration change persists across stopping and starting the virtual server. This option is equivalent to specifying both **--live** and **--config**.

Related concepts

"I/O threads" on page 207
To attain good performance of I/O operations, configure each virtual block device to use an I/O thread.

Related tasks

"Configuring devices" on page 101
A device configuration maps host devices and resources to device representations on a virtual server.

"Displaying the current libvirt-internal configuration" on page 158
The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, complemented with libvirt-internal information, and modified as devices are attached or detached.

Detaching a device

You can unplug devices from a running virtual server, remove devices from the persistent virtual server configuration, or both.

Before you begin

You need a device configuration-XML file to detach a device from a virtual server. If the device has previously been attached to the virtual server, use the device configuration-XML file that was used to attach the device.

Procedure

1. Detach the device using the virsh **detach-device** command (see "detach-device" on page 352):

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

   <device-configuration-XML-filename>
   > Is the name of the device configuration-XML file.

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

   <scope>
   > Specifies the scope of the command:

   > **--live**
   >> Unplugs the device from a running virtual server. This configuration change does not persist across stopping and starting the virtual server.

   > **--config**
   >> Removes the device from the persistent virtual server configuration. The device becomes unavailable when the virtual server is next started. This configuration change persists across stopping and starting the virtual server.

   > **--persistent**
   >> Removes the device from the persistent virtual server configuration and unplugs it if the virtual server is running. This configuration change persists across stopping and starting the virtual server. This option is equivalent to specifying both **--live** and **--config**.

2. Optional: If removing a virtual block device leaves an unused I/O thread, remove the thread along with the device.

   The virsh **iothreadinfo** command displays the I/O threads that are available for a virtual server.

   Use the virsh **iothreaddel** command to remove an I/O thread (see "iothreaddel" on page 371):

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

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

> **<IOthread-ID>**
> Is the ID of the I/O thread to be deleted from the virtual server.

Replacing a virtual DVD

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

Before you begin

Make sure that the virtual DVD drive is configured as a virtual SCSI device (see "Configuring a virtual SCSI-attached CD/DVD drive" on page 122).

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 347):

   ```
   # 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 349):

```
# 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 390):

```
# 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 152
Use the virsh **start** command to start a shut off virtual server.

"Configuring the console" on page 95
Configure the console by using the console element.

Chapter 23. Managing storage pools

Before you begin

1. Configure one or more storage pools as described in <u>Chapter 14, "Configuring storage pools," on page 137</u>.
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".

<u>Figure 24 on page 187</u> shows the state-transition diagram of a storage pool:
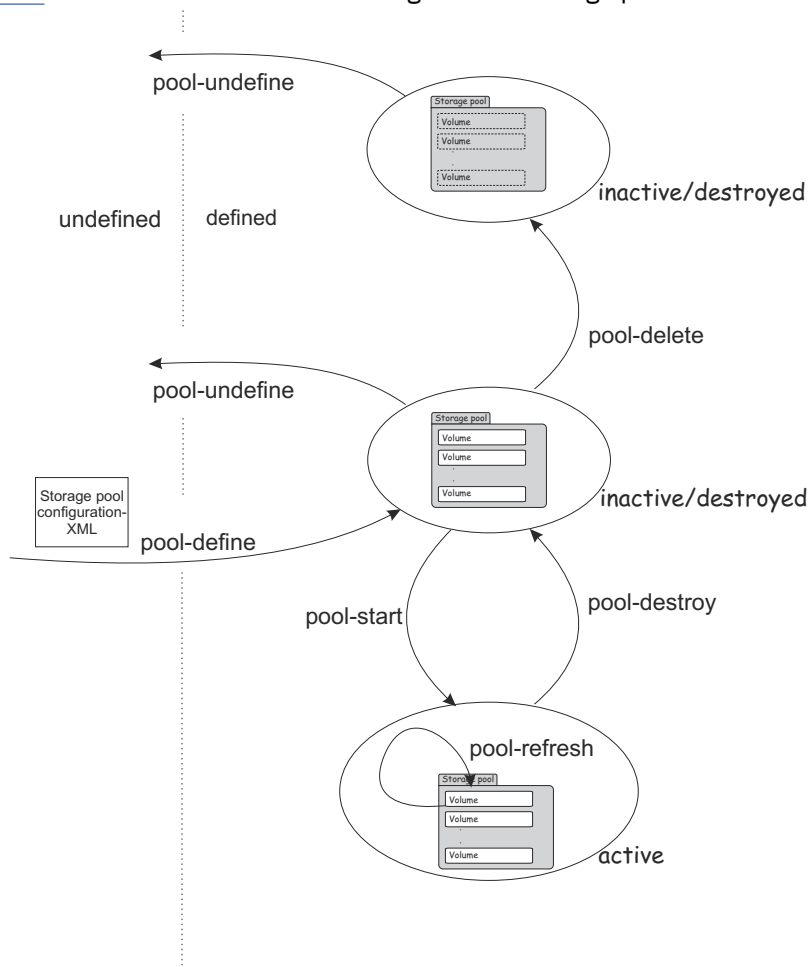
Figure 24. Storage pool state-transition diagram

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 <u>"Storage pool management commands" on page 188</u>.

There are also virsh commands to manage the volumes of a storage pool. Use the commands described in "Volume management commands" on page 189.

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 418
Edit the libvirt-internal configuration of a defined storage pool	"pool-edit" on page 422
Delete the persistent libvirt definition of a storage pool	"pool-undefine" on page 428

- 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 417
Start a defined inactive storage pool	"pool-start" on page 427
Update the volume list of a storage pool	"pool-refresh" on page 426
Shut down a storage pool - the pool can be restarted by using the virsh **pool-start** command	"pool-destroy" on page 420
Delete the volumes of a storage pool ⚠️ **Attention:** This command is intended for expert users. Depending on the pool type, the results range from no effect to loss of data. In particular, data is lost when a zfs or LVM group pool is deleted.	"pool-delete" on page 419

- Monitoring storage pools:

Functionality	Command
View a list of all defined storage pools	"pool-list" on page 424
Display the current libvirt-internal configuration of a storage pool	"pool-dumpxml" on page 421
Display information about a defined storage pool	"pool-info" on page 423
Retrieve the name of a storage pool from its UUID	"pool-name" on page 425
Retrieve the UUID of a storage pool from its name	"pool-uuid" on page 429

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 431
Remove a volume from a storage pool	"vol-delete" on page 432

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 436
Display the current libvirt-internal configuration of a storage volume	"vol-dumpxml" on page 433
Display information about a defined volume	"vol-info" on page 434
Display the key of a volume from its name or path	"vol-key" on page 435
Display the name of a volume from its key or path	"vol-name" on page 437
Display the path of a volume from its name or key	"vol-path" on page 438
Display the storage pool name or UUID which hosts the volume	"vol-pool" on page 439

Chapter 24. Managing virtual networks

Use `virsh` commands to manage virtual networks.

Before you begin

KVM hosts provide a preconfigured virtual network named `default`. You can configure more virtual networks as described in Chapter 15, "Configuring virtual networks," on page 139.

About this task

Virtual networks that are defined to libvirt can be in one of the states "inactive" or "active".

The following figure shows the state-transition diagram of a virtual network:
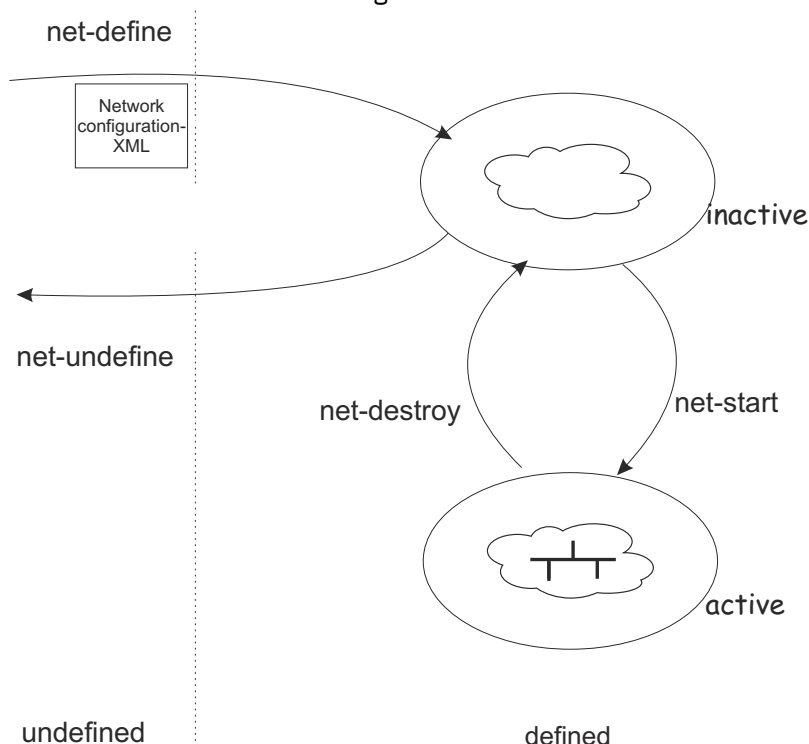
Figure 25. Virtual network state-transition diagram

Procedure

- Creating, modifying, and deleting a persistent network definition:

Task	Command
Create a persistent definition of a virtual network configuration.	"net-define" on page 397
Edit the libvirt-internal configuration of a defined virtual network.	"net-edit" on page 400
Delete the persistent libvirt definition of a virtual network.	"net-undefine" on page 405

- Managing the virtual network lifecycle:

Task	Command
Enable or disable the automatic activation of a virtual network when the libvirt daemon is started.	"net-autostart" on page 396

Task	Command
Activate a defined, inactive virtual network.	"net-start" on page 404
Deactivate a virtual network.	"net-destroy" on page 398

- Monitoring networks:

Task	Command
View a list of all defined virtual networks.	"net-list" on page 402
Display the current configuration of a virtual network.	"net-dumpxml" on page 399
Display information about a defined virtual network.	"net-info" on page 401
Retrieve the name of a virtual network from its UUID.	"net-name" on page 403
Retrieve the UUID of a virtual network from its name.	"net-uuid" on page 406

Chapter 25. Fast path to a running guest - virt-install

With a single **virt-install** command, configure and define a virtual server, and install and run a guest.

Before you begin:

- You need a source for your guest installation. The source can be an ISO file in the host file system, or a Web location that provides the installation files.
- Ensure that **virt-install** is available. Depending on your distribution, the package name for this tool might be "virtinst" or "virt-install".
- Use **virt-xml** to run a transient guest without defining a virtual server that persists beyond the guest shutdown.

The sections that follow show basic examples for running a **virt-install** command. See the man page for more options.

Installing from an ISO file

Issue a command of this form:

```
# virt-install --name=<vs_name> --disk size=<disk_size_in_GB> --memory=<vs_memory_size_in_MB> \
--cdrom <filepath_to_iso>
```

In the command:

<vs_name>
 is the name of the virtual server to be created.

<disk_size_in_GB>
 is the size, in GB, of the virtual disk on which the guest is to be installed. The disk is backed by an image file in the file system of the host.

<vs_memory_size_in_MB>
 is the size, in MB, of the memory of the virtual server to be created.

<filepath_to_iso>
 is the file path to the ISO file in the host file system. During the installation, the file is accessed through a virtual DVD drive.

The **virt-install** tool defines the virtual server. It then starts the virtual server, boots the specified ISO image with the installer, and displays the installation dialog on the console. Follow the instructions to complete the installation.

When the installation is completed the guest is rebooted from the virtual disk where it is installed. You do not need to edit the domain configuration-XML file. **virt-install** makes the required post-installation changes for you.

Example:

```
# virt-install --name=a_server --disk size=10 --memory=1000 \
--cdrom /var/lib/libvirt/images/linux-s390x.iso
```

Installing from a network location

Issue a command of this form:

```
# virt-install --name=<vs_name> --disk size=<disk_size_in_GB> --memory=<vs_memory_size_in_MB> \
--location <url>
```

In the command *<url>* specifies a download site and protocol. See the **virt-install** man page for examples of download sites for some of the major distributions.

The other variables have the same meaning as in "Installing from an ISO file" on page 193.

Example:

```
# virt-install --name=b_server --disk size=10 --memory=1000 \
--location https://download.fedoraproject.org/pub/fedora-secondary/releases/28/Server/s390x/os/
```

The **virt-install** tool defines the virtual server. It then starts the virtual server, boots the installer, and displays the installation dialog on the console.

Tip: With **--location**, you can optionally use the **--extra-args** parameter to specify a distribution-specific configuration file for an unattended installation. Examples of such configuration files are kickstart, autoyast, or a preseed file. You can also use the **--initrd-inject** parameter to dynamically add a configuration file to the initial RAM disk.

Example:

```
# virt-install --name=b_server --disk size=10 --memory=1000 \
--location https://download.fedoraproject.org/pub/fedora-secondary/releases/28/Server/s390x/os/ \
--extra-args inst.ks=file:///f28.ks --initrd-inject=f28.ks
```

When the installation is completed the guest is rebooted from the virtual disk where it is installed. You do not need to edit the domain configuration-XML file. **virt-install** makes the required post-installation changes for you.

Customizing your virtual server

Edit the domain configuration-XML file to customize the virtual server according to your needs.

Chapter 26. Booting from a temporary boot device

You might want to start a virtual server with a guest that boots from a device other than the configured boot device, for example for maintenance or in a recovery situation. With **virt-xml**, you can override the boot device that is configured in the libvirt-internal configuration and start the virtual server with a temporary boot device.

Procedure

1. Optional: Display the current virtual server configuration in XML format.

   Example:

   ```
   # virsh dumpxml vs002
   <domain>
       <name>vs002</name>
       ...
       <devices>
           ...
           <!-- DVD -->
           <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/LinuxDVD1.iso"/>
               <target dev="sda" bus="scsi"/>
               <address type="drive" controller="4" bus="0" target="0" unit="0"/>
               <readonly/>
           </disk>
           ...
           <!-- 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>
   ```

   The sample XML snippet configures two disks to the guest.

   vda
   includes the boot element with attribute `order="1"`, which configures this disk as the boot device.

   sda
   a DVD image file that is configured as another disk, which is to be the temporary boot device.

   The two devices need not be disks but can be any two bootable devices. The virt-xml command in the step that follows changes accordingly.

2. Start the virtual server with a **virt-xml** command that includes the `--no-define` and `--start` options and that overrides the boot device.

   The options for overriding the boot device depend on the device configurations.

   Example:

   For the example of the previous step, issue the following command to start virtual server `vs002` with sda as the temporary boot device:

   ```
   # virt-xml vs002 --edit target="sda" --disk="boot_order=1" --no-define --start
   ```

   The parameters in the command have the following meaning:

   --disk="boot_order=1"
   states that the temporary boot device is configured with a disk element.

195

target="sda"
    among the configured disk devices, identifies `sda` as the temporary boot device.

--no-define --start
    applies the changes to a transient copy of the libvirt-internal configuration and starts the virtual server with that transient copy. The persistent configuration remains unchanged.

You always need the `--no-define --start` parameters. The specifications for identifying the temporary boot device vary according to the boot device configuration.

Results
The virtual server starts and boots the guest from the temporary boot device. A subsequent regular start of the virtual server boots the guest from the persistently configured boot device.

Part 5. Best practices and performance considerations

Avoid common pitfalls and tune the virtual server.

Chapter 27. CPU management

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

Related tasks

"Configuring virtual CPUs" on page 85
Configure virtual CPUs for a virtual server.

"Managing virtual CPUs" on page 176
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.
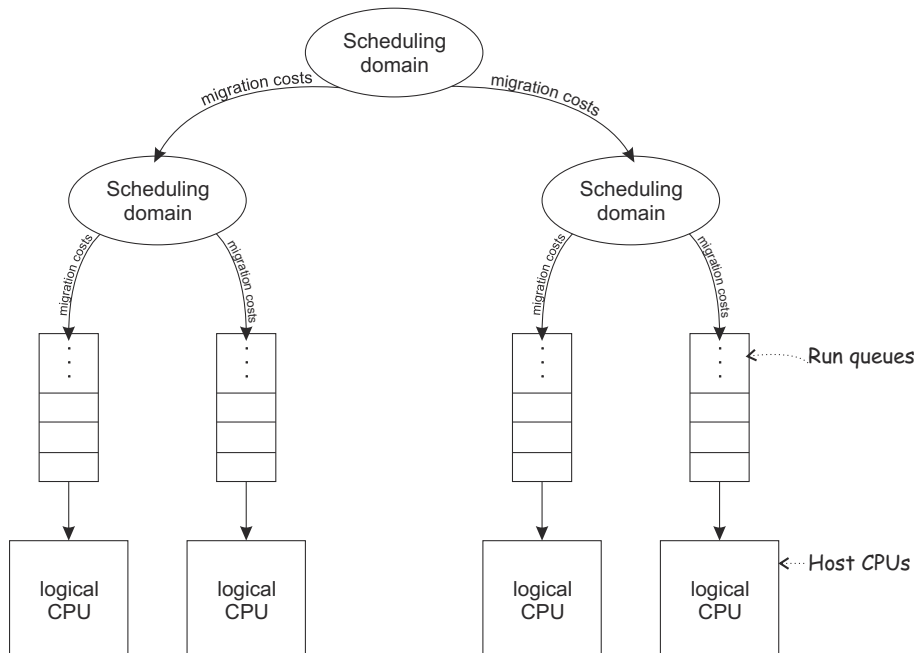
Figure 26. Linux scheduling

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:

CPU shares on host CPU 0:

Steal time
Host overhead
A's virtual CPU 0
B's virtual CPU 0
B's virtual CPU 1

50%

25%

25%

CPU shares on host CPU 1:

Steal time
Host overhead
A's virtual CPU 1
B's virtual CPU 2
B's virtual CPU 3

50%

25%

25%

Chapter 28. 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*.
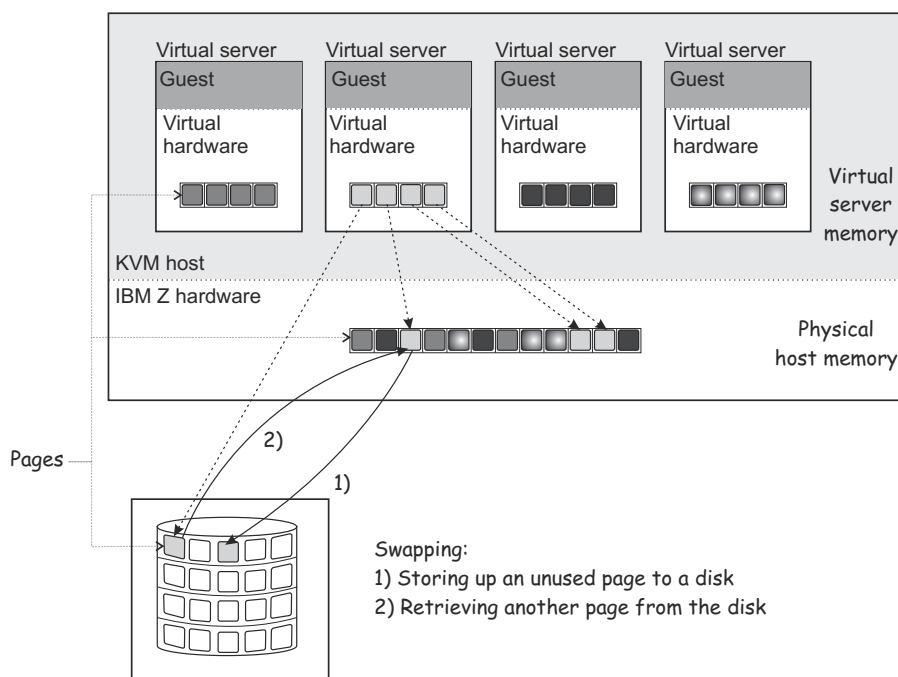
Figure 27. Swapping

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 host may be considerably less than the defined amount. There are multiple techniques allowing the host to efficiently deal with memory overcommitment:

Using huge pages

You can back the virtual memory of a virtual server with 1 MB huge pages of host memory. Depending on your workload, huge pages can result in performance gains.

Prerequisites on the host

- The host must be configured with enough 1 MB pages to satisfy the needs of its guests.
- The kvm module must be loaded with the `hpage` parameter to support guest-configurations with huge-page memory backing.

For more information, see the large pages (huge pages in recent editions) section in *Device Drivers, Features, and Commands*.

Restrictions for guests

- Transparent huge pages (THP) are not supported.
- The huge page size must be 1 MB.
- Memory overcommitment is not possible.
- Huge pages cannot be freed.
- Collaborative memory management is disabled on guests that use huge pages.
- KVM guests that use huge pages cannot be KVM hosts for higher-level guests.
- You cannot use huge pages for KVM guests that are to run in IBM Secure Execution mode.

For details about configuring huge pages for a guest, see "Configuring huge pages" on page 91.

Collaborative memory management

A guest operating system can mark memory pages as *unused* or *volatile* with the IBM Z Collaborative Memory Management Assist (CMMA) facility. This allows the host to avoid unnecessary disk swapping because unused pages can simply be discarded. Current Linux operating systems make use of CMMA. The subset of the CMMA facility as used by Linux is enabled in KVM, therefore transparently ensuring efficient physical host memory usage, while still allowing the virtual server to use all of the defined virtual memory if needed.

Ballooning

KVM implements a virtual memory balloon device that serves the purpose of controlling the physical host memory usage of a virtual server. With the balloon device, the host can request that the guest gives up memory. This could be done to re-balance the resource allocations between virtual servers to adapt to changing resource needs.

Whether and to which extent the guest honors the request depends on a few factors not controlled by the host, such as, whether or not a balloon device driver is installed in the guest, or whether there's enough memory that can be freed.

Unlike for CMMA, the memory given up by the balloon device is removed from the virtual server and cannot be reclaimed by the guest. As this can cause adverse effects and even lead to program or operating system failures due to low memory conditions, it should only be used in well-understood situations. By default, you should disable the balloon device by configuring `<memballoon model="none"/>`.

Memory tuning

Another way to control virtual server memory usage is by means of the Linux cgroups memory controller. By specifying a soft limit the amount of physical host memory used by a virtual server can be restricted once the host is under high memory pressure, that is, the host is experiencing high swapping activity. Again, this would typically be done to re-balance resource allocations between virtual servers.

Since the virtual server memory available to the guest is not modified, applying a soft limit is transparent, except for the performance penalty caused by swapping. If swapping becomes excessive, time-critical processes may be affected, causing program failures. Therefore the soft limit should be applied carefully as well.
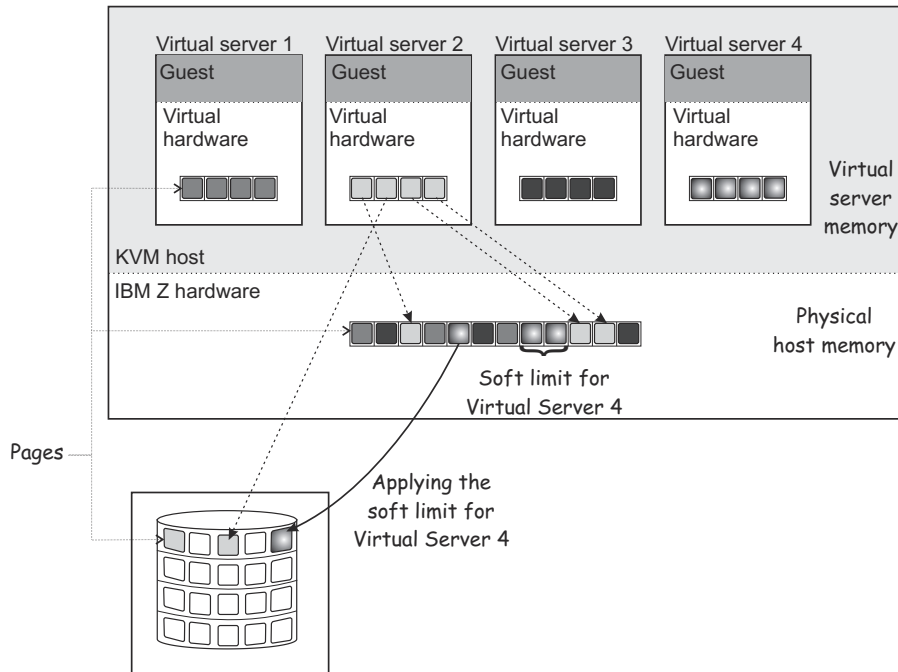
Figure 28. Applying the soft limit

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 89
Configure the virtual server memory.

"Managing virtual memory" on page 180
Specify a soft limit for the amount of physical host memory used by a virtual server.

Chapter 29. Storage management

Consider these aspects when setting up and configuring the virtual server storage.

I/O threads

To attain good performance of I/O operations, configure each virtual block device to use an I/O thread.

I/O threads are intended to enhance the performance but they consume processing and memory resources. Defining an excessive number of I/O threads can be counterproductive. Do not configure more I/O threads than available host CPUs. Do not configure more I/O threads than virtual block I/O devices that will be available for the virtual server.

You can configure multiple devices to use the same thread. Which mapping of threads to devices yields best results depends on the available resources and on the workload.

You can configure I/O threads in the domain configuration-XML of a virtual server. For more information, see:

- "Configuring devices with the virtual server" on page 94
- "Configuring a DASD, SCSI, or NVMe disk" on page 103
- "Configuring a virtual HBA" on page 115

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 182
- "Detaching a device" on page 183

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 11, 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.
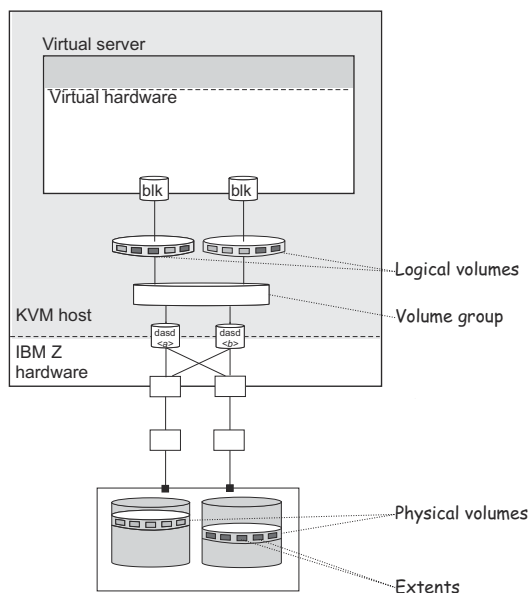
- On the virtual server:

  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.
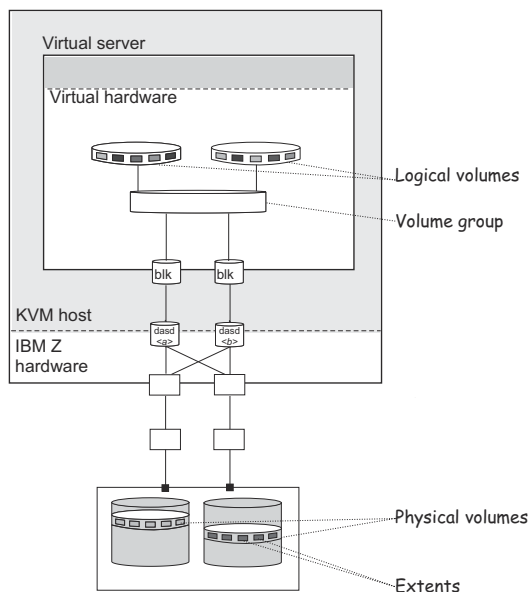
  To prohibit a logical volume management for physical volumes that are managed on the virtual server, provide an explicit allowlist in `/etc/lvm/lvm.conf` which explicitly contains all disk block devices to be managed on the host, or a blocklist that contains all physical volumes that are to be managed on the virtual server.

  The filter section in the device settings allows to specify an allowlist using the prefix "a", and to specify a blocklist using the prefix "r".

Example

This allowlist in `/etc/lvm/lvm.conf` filters the physical volumes which are to be managed on the host. The last line (`"r|.*|"`) denotes that all other physical volumes that are not listed here are not to be managed on the host.

```
devices
        {filter = [ "a|/dev/mapper/36005076305ffc1ae00000000000021d5p1|",
                    "a|/dev/mapper/36005076305ffc1ae00000000000021d7p1|",
                    "a|/dev/disk/by-path/ccw-0.0.1607-part1|",
                    "r|.*|" ]
        }
```

The following physical volumes are to be managed on the host:

- `/dev/mapper/36005076305ffc1ae00000000000021d5p1`

- `/dev/mapper/36005076305ffc1ae00000000000021d7p1`

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

You can verify that SCSI disks are referenced correctly by issuing the following **pvscan** command:

```
# pvscan -vvv 2>&1 | fgrep '/dev/sd'
 ...
    /dev/sda: Added to device cache
    /dev/block/8:0: Aliased to /dev/sda in device cache
    /dev/disk/by-path/ccw-0.0.50c0-zfcp-0x1234123412341234:\
      0x0001000000000000: Aliased to /dev/sda in device cache
    ...
    /dev/sda: Skipping (regex)
```

The output must contain the string "Skipping (regex)" for each SCSI disk standard device name which is configured for the virtual server.

Chapter 30. Performance hints and tips summary

Tune the performance of your virtual server.

How best to tune a KVM virtual server environment strongly depends on the workloads of the virtual servers and on the available resources. The same measure that enhances performance in one environment can have adverse effects in another. Finding the best balance for a particular setting can be a challenge and often involves experimentation. Use the following suggestions for your consideration.

Use multiple queues for your virtio network interfaces

With multiple virtual CPUs, you can transfer packages in parallel if you provide multiple queues for incoming and outgoing packets. Use the queues attribute of the driver element to configure multiple queues. Specify an integer of at least 2 but not exceeding the number of virtual CPUs of the virtual server.

The following specification configures two input and output queues for a network interface:

```
<interface type="direct">
      <source network="net01"/>
      <model type="virtio"/>
      <driver ... queues="2"/>
</interface>
```

Multiple queues are designed to provide enhanced performance for a network interface, but they also use memory and CPU resources. Start with defining two queues for busy interfaces. Then, try just one queue for interfaces with less traffic or more than two queues for busy interfaces.

Use I/O threads for your virtual block devices

Two configuration steps are required to make virtual block devices use I/O threads.

1. You must configure one or more I/O threads for the virtual server.
2. You must configure each virtual block device to use one of these I/O threads.

The following example uses the specification `<iothreads>3</iothreads>` to configure three I/O threads, with consecutive decimal thread IDs 1, 2, and 3. The `iothread="2"` specification for the driver element of the disk device then configures the device to use the I/O thread with ID 2.

```
<domain>
    <iothreads>3</iothreads>
    ...
    <devices>
        ...
        <disk type="block" device="disk">
            <driver ... iothread="2"/>
            ...
        </disk>
        ...
    </devices>
    ...
</domain>
```

Threads can increase the performance of I/O operations for disk devices, but they also use memory and CPU resources. You can configure multiple devices to use the same thread. Which mapping of threads to devices yields best results depends on the available resources and the workload.

Start with a small number of I/O threads. Often, a single I/O thread for all disk devices is sufficient. Do not configure more threads than the number of virtual CPUs, and do not configure idle threads.

For a running virtual server, you can use the **virsh iothreadadd** command to add I/O threads with specific thread IDs.

See also .

Exclude the memory balloon device

Unless you need a dynamic memory size, do not define a memory balloon device and ensure that libvirt does not create one for you. Include the following specification as a child of the devices element in your domain configuration-XML.

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

For more information about the balloon device, see "Ballooning" on page 204.

Disable the cpuset cgroup controller

Note: This setting applies only to KVM hosts with cgroups version 1.

To enable CPU hotplug on the host, disable the cgroup controller.

1. Open /etc/libvirt/qemu.conf with an editor of your choice.
2. Find the cgroup_controllers line.
3. Duplicate this line and from the copy remove the leading number sign (#).
4. Remove cpuset from the list of values, such that the line reads:

```
cgroup_controllers = [ "cpu", "devices", "memory", "blkio", "cpuacct" ]
```

Restart the libvirtd daemon to make the setting take effect.

1. Stop all virtual servers.
2. Issue the following command:

```
# systemctl restart libvirtd
```

3. Start the virtual servers.

This setting persists across host reboots.

Avoid virtual SCSI devices

Configure virtual SCSI devices only if you need to address the device through SCSI-specific interfaces. In particular, configure disk space as virtual block devices rather than virtual SCSI devices, regardless of the backing on the host.

In the examples that follow you might need SCSI specifics.

- A LUN for a SCSI-attached tape drive on the host.
- A DVD ISO file on the host file system that is mounted on a virtual DVD drive.

Configure disk caching

Configure disk devices such that caching is done by the guest and not by the host.

Ensure that the driver element of the disk device includes the specifications cache="none" and io="native".

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    ...
</disk>
```

Back your virtual servers with huge pages

Unless you need host paging or IBM Secure Execution for Linux, back the memory of your virtual servers with huge pages on the host.

Multiple configuration steps are required to use huge pages:

1. Enable KVM to use huge pages for virtual servers. For example, include the following line in /etc/
modprobe.conf/kvm.conf on the host:

```
options kvm hpage=1
```

   To make this setting take effect on a running host, you must temporarily stop all virtual servers and
reload the kvm module.

2. On the host, reserve huge pages by issuing a command like this:

```
# sysctl vm.nr_hugepages=30000
```

   The number of pages depends on the requirements of your virtual servers.

3. Include the following specification in the domain configuration-XML of your virtual servers:

```
<domain type="kvm">
    ...
    <memoryBacking>
        <hugepages/>
    </memoryBacking>
    ...
</domain>
```

For more information about backing virtual server memory with huge pages, see "Using huge pages" on
page 203.

Configure a suitable number of virtual CPUs

Virtual CPUs use host resources. An excessive number of virtual CPUs can adversely affect performance.

For each virtual server, configure just enough virtual CPUs to adequately handle peak loads. Do not
configure more virtual CPUs than the number of available host CPUs.

Accommodate growing processing requirements or unusual peaks by configuring virtual CPUs that can
be dynamically added to the virtual servers. For example, the following specification in the domain
configuration-XML of a virtual server starts the virtual server with 4 virtual CPUs that can then be
dynamically increased to up to 8 virtual CPUs.

```
<domain type="kvm">
    ...
    <vcpu current="4">8</vcpu>
    ...
</domain>
```

Tune the polling period for idle virtual CPUs

When a virtual CPU becomes idle, KVM polls for wakeup conditions for the virtual CPU before ceding
the host resource. The time interval after which polling stops is specified, in nanoseconds, in sysfs
at /sys/module/kvm/parameters/halt_poll_ns. During the specified time, 50000 ns by default,
polling reduces the wakeup latency for the virtual CPU at the expense of resource usage. Depending on
the workload, a longer or shorter time for polling can be advantageous.

To optimize for low CPU consumption, write a small value to /sys/module/kvm/parameters/
halt_poll_ns, or write 0 to disable polling.

```
# echo 0 > /sys/module/kvm/parameters/halt_poll_ns
```

To optimize for low latency, for example for transactional workloads, write a large value to /sys/
module/kvm/parameters/halt_poll_ns, for example:

```
# echo 80000 > /sys/module/kvm/parameters/halt_poll_ns
```

Add the following line to `/etc/modprobe.d/kvm.conf` or change an existing line to make it persistent:

```
options kvm halt_poll_ns=<polling_halt>
```

where *<polling_halt>* is the period, in nanoseconds, after which polling stops.

To make this setting take effect on a running host, you must temporarily stop all virtual servers and reload the kvm module.

Regardless of this setting, KVM disables idle virtual CPUs if contention occurs within KVM or at the LPAR level.

See also the information about `halt_poll_ns` in *KVM Network Performance - Best Practices and Tuning Recommendations* [PDF | HTML].

Tune the CPU migration algorithm of the host scheduler

Important: Do not change the scheduler settings unless you are an expert who understands the implications. Do not apply changes to production systems without testing them and confirming that they have the intended effect.

The `kernel.sched_migration_cost_ns` parameter specifies a time interval in nanoseconds. After the last execution of a task, the CPU cache is considered to have useful content until this interval expires. Increasing this interval results in fewer task migrations. The default value is 500000 ns.

If the CPU idle time is higher than expected when there are runnable processes, try reducing this interval. If tasks bounce between CPUs too often, try increasing it.

To dynamically set the interval to 60000 ns, enter the following command:

```
# sysctl kernel.sched_migration_cost_ns=60000
```

Use the following entry in `/etc/sysctl.conf`, to persistently change the value to 60000 ns:

```
kernel.sched_migration_cost_ns=60000
```

Part 6. Diagnostics and troubleshooting

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

Chapter 31. Logging

Adapt the logging facility to your needs.

Log messages

These logs are created.

libvirt log messages
> By default, libvirt log messages are stored in the system journal. You can specify a different location in the libvirt configuration file at `/etc/libvirt/libvirtd.conf`. For more information, see libvirt.org/logging.html.

QEMU log file of a virtual server
> `/var/log/libvirt/qemu/<VS>.log`, where *<VS>* is the name of the virtual server.

Console log file
> If the log element is specified in the console configuration, the log file attribute indicates the console log file.

> **Example:**

> The following console configuration specifies the console log file `/var/log/libvirt/qemu/vserv-cons0.log`:

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

Specifying the logging level of the libvirt log messages

Specify the level of logging information that is displayed in the libvirt log messages file.

About this task
For further information, see: libvirt.org/logging.html

Procedure
1. In the libvirt configuration file `/etc/libvirt/libvirtd.conf`, specify:

   ```
   log_level = <n>
   ```

   Where *<n>* is the logging level:

   4
   > Displays errors.

   3
   > Is the default logging level, which logs errors and warnings.

   2
   > Provides more information than logging level 3.

   1
   > Is the most verbose logging level.

2. Restart the libvirt daemon to enable the changes.

```
# systemctl restart libvirtd.service
```

Chapter 32. 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*. Typically, you configure kdump though distribution-specific tools to write a filtered and compressed dump file to persistent storage and reboot the production kernel.

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.

Chapter 33. Finding virtual server crash information

For specific causes of a virtual server crash, libvirt writes crash information to the QEMU log file of the virtual server.

The following reasons for a crash trigger a message:

- A disabled wait state
- Wrong interrupt
- Program check loop

The default location for the QEMU log of a virtual server is `/var/log/libvirt/qemu/<VS>`, where *<VS>* is the name of the virtual server see also "Log messages" on page 217).

A message provides information about the PSW and states the reason of the crash.

Example:

```
s390: psw-mask='0xXXXXXXXXXXXXXXXX', psw-addr='0xXXXXXXXXXXXXXXXX',
crash reason: disabled wait
```

Chapter 34. Collecting performance metrics

You can monitor virtual server machine code instructions.

Before you begin

- You need a kernel that is built with the common code options CONFIG_TRACEPOINTS, CONFIG_HAVE_PERF_EVENTS, and CONFIG_PERF_EVENTS.
- Confirm that the **perf** tool is installed, for example, by issuing the **perf list** command.

```
# 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

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).
  - The **report** subcommand displays the performance metrics that were previously recorded with the **record** subcommand.
  - Save a copy of perf.data.guest before you collect new statistics. A new **record** might overwrite this file.

- The **live** subcommand displays the current statistics without saving them.

  The **perf** tool displays events until you terminate it by pressing Control and C (Ctrl+C).

Example

```
# ./perf kvm stat record -a
^C[ perf record: Woken up 7 times to write data ]
[ perf record: Captured and wrote 13.808 MB perf.data.guest (~603264 samples) ]

# ./perf kvm stat report


Analyze events for all VMs, all VCPUs:

                          VM-EXIT   Samples  Samples%   Time%   Min Time      Max Time      Avg time
                Host interruption     14999    35.39%    0.39%    0.45us     1734.88us      0.82us ( +-  19.59% )
         DIAG (0x44) time slice end    13036    30.76%    0.57%    1.06us     1776.08us      1.39us ( +-   9.81% )
     DIAG (0x500) KVM virtio functions  13011    30.70%    1.90%    1.15us     2144.75us      4.65us ( +-   5.08% )
                       0xE5 TPROT        512     1.21%    0.01%    0.79us        2.18us      0.83us ( +-   0.42% )
                       0xB2 TSCH         406     0.96%    0.19%    7.35us      109.43us     14.95us ( +-   2.97% )
                       0xB2 SERVC        117     0.28%    0.15%   10.97us      339.00us     40.46us ( +-   9.17% )
                 External request       113     0.27%    0.01%    0.75us        2.58us      1.56us ( +-   1.55% )
                       0xB2 STSCH         57     0.13%    0.02%    7.30us       26.40us      9.47us ( +-   5.99% )
                       Wait state         40     0.09%   96.48% 3334.30us   464600.00us  76655.28us ( +-  32.97% )
                       0xB2 MSCH          14     0.03%    0.00%    7.22us        9.19us      7.74us ( +-   2.13% )
                       0xB2 SSCH          14     0.03%    0.01%    8.67us       35.41us     16.16us ( +-  16.38% )
                       0xB2 CHSC          10     0.02%    0.00%    7.51us       22.90us     11.06us ( +-  15.20% )
                       I/O request         8     0.02%    0.00%    1.37us        1.97us      1.55us ( +-   5.77% )
                       0xB2 STPX           8     0.02%    0.00%    1.04us        7.10us      1.98us ( +-  37.25% )
                       0xB2 STSI           7     0.02%    0.00%    1.65us       62.09us     22.26us ( +-  41.95% )
                       0xB2 STIDP          4     0.01%    0.00%    1.12us        3.62us      2.62us ( +-  21.07% )
                 SIGP set architecture     3     0.01%    0.00%    1.05us        2.68us      1.60us ( +-  33.74% )
                       0xB2 STAP           3     0.01%    0.00%    1.05us        7.61us      3.39us ( +-  62.25% )
                       0xB2 STFL           3     0.01%    0.00%    1.78us        3.88us      2.84us ( +-  21.31% )
      DIAG (0x204) logical-cpu utilization  2     0.00%    0.00%    4.58us       39.48us     22.03us ( +-  79.19% )
             DIAG (0x308) ipl functions     2     0.00%    0.01%   19.34us      329.25us    174.30us ( +-  88.90% )
      DIAG (0x9c) time slice end directed   1     0.00%    0.00%    1.09us        1.09us      1.09us ( +-   0.00% )
                       0xB2 SPX            1     0.00%    0.00%    4.58us        4.58us      4.58us ( +-   0.00% )
                       0xB2 SETR           1     0.00%    0.00%   56.97us       56.97us     56.97us ( +-   0.00% )
                       0xB2 SSKE           1     0.00%    0.25% 7957.94us     7957.94us   7957.94us ( +-   0.00% )
                       0xB2 STCRW          1     0.00%    0.00%   11.24us       11.24us     11.24us ( +-   0.00% )
      DIAG (0x258) page-reference services  1     0.00%    0.00%    4.87us        4.87us      4.87us ( +-   0.00% )
                       0xB9 ESSA           1     0.00%    0.00%    8.72us        8.72us      8.72us ( +-   0.00% )
                       0xEB LCTLG          1     0.00%    0.00%    9.27us        9.27us      9.27us ( +-   0.00% )

Total Samples:42377, Total events handled time:3178166.35us.
```

What to do next

For more information about the **perf** subcommand **kvm stat**, see the man page or issue the full
subcommand with the --help option:

```
▶▶─ perf ── kvm ── stat ──┬── record ──┬── --help ─▶◀
                          ├── report ──┤
                          └── live ────┘
```
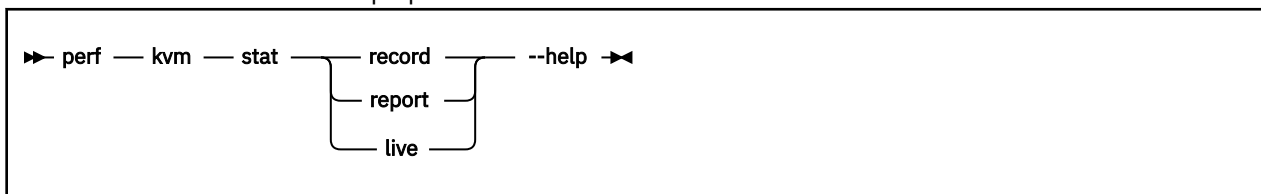
With the collected statistics, you can watch the virtual server behavior and time consumption, and then
analyze the recorded events to identify possible performance issues.

- 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.
- DIAGNOSE (DIAG) instructions send virtual server requests to the hypervisor. Read the contents
  of /sys/kernel/debug/diag_stat to display a list of which diagnose instruction are called how
  frequently by your Linux instance:

```
# cat /sys/kernel/debug/diag_stat
```

The output might include instructions that are called, but not supported by KVM on IBM Z.
lists the supported instructions.

Table 2. Supported Linux DIAGNOSE instructions		
Number	**Description**	**Linux use**
0x010	Release pages	CMM
0x044	Voluntary time-slice end	In the kernel for spinlock and udelay
0x09c	Voluntary time slice yield	Spinlock
0x258	Page-reference services	In the kernel, for pfault
0x288	Virtual server time bomb	The watchdog device driver
0x308	Re-ipl	Re-ipl and dump code
0x318	CP Name and Version Codes	Identifies itself as Linux to the hypervisor
0x500	Virtio functions	Operate virtio-ccw devices

Part 7. Reference

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

Chapter 35. Virtual server life cycle

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

Figure 29 on page 229 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.
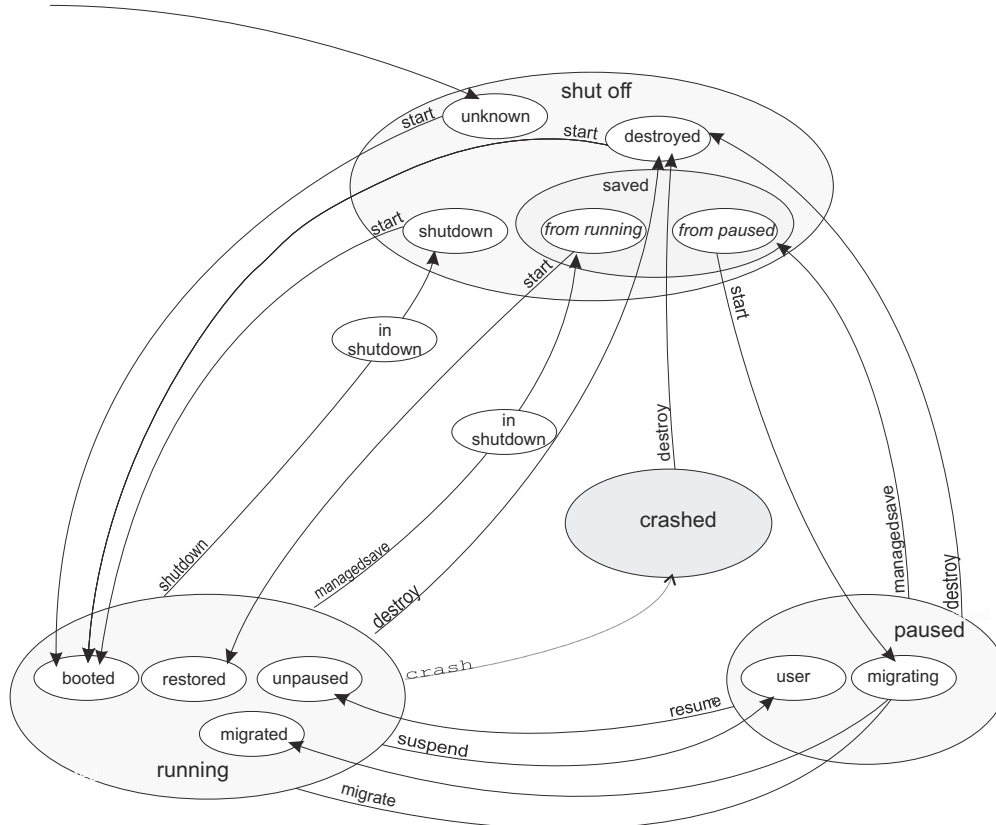
Figure 29. State-transition diagram of a virtual server including reasons

shut off

The virtual server is defined to libvirt and has not yet been started, or it was terminated.

Reasons

unknown	The virtual server is defined to the host.
saved	The system image of the virtual server is saved in the file `/var/lib/libvirt/qemu/save/<VS>.save` and can be restored.
	The system image contains state information about the virtual server. Depending on this state, the virtual server is started in the state running or paused.
shutdown	The virtual server was properly terminated. The virtual server's resources were released.
destroyed	The virtual server was immediately terminated. The virtual server's resources were released.

Commands

Command	From state (reason)	To state (reason)
start	shut off (unknown)	running (booted)
start	shut off (saved *from running*)	running (restored)
start	shut off (saved *from paused*)	paused (migrating)
start	shut off (shutdown)	running (booted)
start	shut off (destroyed)	running (booted)
start --force-boot	shut off (unknown)	running (booted)
start --force-boot	shut off (saved *from running*)	running (booted)
start --force-boot	shut off (saved *from paused*)	paused (user)
start --force-boot	shut off (shutdown)	running (booted)
start --force-boot	shut off (destroyed)	running (booted)
start --paused	shut off (unknown)	paused (user)
start --paused	shut off (saved *from running*)	paused (migrating)
start --paused	shut off (saved *from paused*)	paused (migrating)
start --paused	shut off (shutdown)	paused (user)
start --paused	shut off (destroyed)	paused (user)

running

The virtual server was started.

Reasons

booted   The virtual server was started from scratch.

migrated   The virtual server was restarted on the destination host after the stopped phase of a live migration.

restored   The virtual server was started at the state indicated by the stored system image.

unpaused   The virtual server was resumed from the paused state.

Commands

Command	Transition state	To state (reason)
destroy	n/a	shut off (destroyed)
managedsave	n/a	shut off (saved *from running*)
managedsave --running	n/a	shut off (saved *from running*)
managedsave --paused	n/a	shut off (saved *from paused*)
migrate	paused (migrating)	running (migrated)
migrate --suspend	paused (migrating)	paused (user)
shutdown	in shutdown	shut off (shutdown)
suspend	n/a	paused (user)

paused

The virtual server has been suspended.

Reasons

user	The virtual server was suspended with the virsh **suspend** command.
migrating	The virtual server's system image is saved and the virtual server is halted - either because it is being migrated, or because it is started from a saved shut off state.

Commands

Command	Transition state	To state (reason)
`destroy`	n/a	shut off (destroyed)
`managedsave`	n/a	shut off (saved *from paused*)
`managedsave --running`	n/a	shut off (saved *from running*)
`managedsave --paused`	n/a	shut off (saved *from paused*)
`resume`	n/a	running (unpaused)
`shutdown`	in shutdown	shut off (shutdown)

crashed

The virtual server crashed and is not prepared for a reboot.

You can create memory dumps of the virtual server.

Then, you can terminate the virtual server and restart it.

For testing purposes, you can crash a virtual server with the virsh **inject-nmi** command.

Commands

Command	To state (reason)
destroy	shut off (destroyed)

in shutdown

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

Chapter 36. Selected libvirt XML elements

These libvirt XML elements might be useful for you. You find the complete libvirt XML reference at libvirt.org.

Some of the described elements can occur in different contexts. You will find multiple, context-dependent descriptions for elements that have different meanings in different contexts. For example, the device element can occur within a storage pool configuration-XML, or it can be the root element of a node-device XML.

Domain configuration-XML

The domain configuration-XML describes KVM virtual servers. For related virsh commands, see "Domain management virsh commands" on page 343.

`<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 117

Parent elements
"<source> as child element of <hostdev>" on page 298.

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="drive" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
   ...
</devices>
```

\<address\> as child element of \<controller\>, \<disk\>, 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, SCSI, or NVMe disk" on page 103
- "Configuring an image file as storage device" on page 109

Parent elements

- "\<controller\>" on page 251
- "\<disk\>" on page 255
- "\<memballoon\>" on page 280

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 pass-through device on the guest. Such devices are connected to the virtual server through a controller or through the VFIO framework.

Text content
None.

Selected attributes

The set of relevant attributes depends on the device type: `drive`, `ccw`, or `pci`.

type="drive"
Specifies a SCSI device. The parent element can be hostdev or disk.

 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 in the range 0 - 255.

 unit
 Specifies the unit number (LUN) of the virtual SCSI device.

type="ccw"
Specifies a device that is attached to the virtual channel subsystem, for example a VFIO pass-through DASD. The parent element is hostdev.

 cssid
 Specifies the channel subsystem number of the device. Must be `0xfe`. In the device address on the guest, this value resolves to `0`.

 ssid
 Specifies the subchannel set of the device. Valid values are in the range `0x0` - `0x3`.

 devno
 Specifies the device number of the device. Must be a unique value in the range `0x0000` - `0xffff`.

type="pci"
Specifies a VFIO pass-through PCI device. The parent element is hostdev. You can but need not specify valid values for the domain, bus, slot, and function of the PCI address. If you omit these specification, libvirt generates valid values for you.

 domain
 Specifies the virtual PCI domain number. Must be `0x0000`.

 bus
 Specifies the virtual PCI bus. Must be `0x00`.

 slot
 Specifies the virtual PCI slot. Valid values are in the range `0x01` - `0x1f`.

 function
 Specifies the virtual PCI function. Must be `0x0`.

Usage

- "Configuring a SCSI tape or medium changer device" on page 117
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 122
- "Configuring a pass-through DASD" on page 130

- "Configuring pass-through PCI devices" on page 131

Parent elements

- "<hostdev>" on page 267
- "<disk>" on page 255

Child elements

"<zpci>" on page 306

Examples

```
<devices>
    ...
    <hostdev mode="subsystem" type="mdev" model="vfio-ccw">
        <source>
            <address uuid="90c6c135-ad44-41d0-b1b7-bae47de48627"/>
        </source>
        <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x00a1"/>
    </hostdev>
    ...
    <hostdev mode="subsystem" type="pci">
        <driver name="vfio"/>
        <source>
          <address domain="0x0002" bus="0x00" slot="0x00" function="0x0"/>
        </source>
        <address type="pci">
          <zpci uid="0x0001" fid="0x00000000"/>
        </address>
    </hostdev>
    ...
    <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="drive" 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 <interface>

Specifies the address of a network device on the virtual server.

Text content
None.

Selected attributes

The set of relevant attributes depends on the type: `ccw` or `pci`

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".

type="pci"

Specifies a VFIO pass-through network PCI device. You can but need not specify valid values for the domain, bus, slot, and function of the PCI address. If you omit these specification, libvirt generates valid values for you.

domain

Specifies the virtual PCI domain number. Must be `0x0000`.

bus

Specifies the virtual PCI bus. Must be `0x00`.

slot

Specifies the virtual PCI slot. Valid values are in the range `0x01` - `0x1f`.

function

Specifies the virtual PCI function. Must be `0x0`.

Usage

- "Configuring pass-through PCI devices" on page 131
- "Configuring a network IPL device" on page 81

Parent elements

"<interface>" on page 272

Child elements

"<zpci>" on page 306

Example

```
<devices>
    ...
    <interface mode="subsystem" type="pci">
        ...
        <address type="pci">
            <zpci uid="0x0001" fid="0x00000000"/>
        </address>
```

```
      ...
    </interface>
    ...
</devices>
```

\<address> as child element of \<source>

Specifies a device address from the host point of view.

Text content
None.

Selected attributes
The set of relevant attributes depends on the type of the host device: `scsi`, `mdev`, or `pci`. The device type is specified as the type attribute of the ancestor hostdev element.

type="scsi"
> Specifies a SCSI device.
>
> **bus="0"**
> > For a SCSI device, the value is zero.
>
> **target**
> > Specifies the SCSI ID.
>
> **unit**
> > Specifies the SCSI LUN.

type="mdev"
> Specifies a VFIO mediated device, for example, a pass-through device for cryptographic adapter resources.
>
> **uuid**
> > Specifies the UUID of the mediated device on the host.

type="pci"
> Specifies a pass-through PCI device. On Z, PCI devices are identified by a function address of the form: *\<domain>:\<bus>:\<slot>.\<function>*. All specifications must match the values of the host device as displayed with **lspci -D**.
>
> **domain**
> > Specifies the PCI domain number.
>
> **bus**
> > Specifies the PCI bus.
>
> **slot**
> > Specifies the PCI slot.
>
> **function**
> > Specifies the PCI function.

Usage
- "Configuring a SCSI tape or medium changer device" on page 117
- "Configuring cryptographic adapter resources" on page 133
- "Configuring a pass-through DASD" on page 130
- "Configuring pass-through PCI devices" on page 131

Parent elements
"\<source> as child element of \<hostdev>" on page 298

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="drive" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
   ...
   <hostdev mode="subsystem" type="mdev" model="vfio-ap">
      <source>
         <address uuid="99e714ec-8eee-40fd-a26e-80ff3b1a2564"/>
      </source>
   </hostdev>
   ...
   <hostdev mode="subsystem" type="pci" managed="yes">
      <driver name="vfio"/>
      <source>
         <address domain="0x0002" bus="0x00" slot="0x00" function="0x0"/>
      </source>
      <address type="pci">
         <zpci uid="0x0001" fid="0x00000000"/>
      </address>
   </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 128

Parent elements

"\<rng>" on page 292

Child elements

None.

Example

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

\<boot\>

Indicates that the virtual block device is bootable.

Text content

None.

Selected attributes

order=*number*

Specifies the order in which a device is considered as boot device during the boot sequence.

loadparm=*number*

For IPL devices with a boot menu configuration: Specifies the boot menu entry. If this parameter is omitted, the default entry is booted.

Usage

"Configuring the boot process" on page 77

Parent elements

- "\<disk\>" on page 255
- "\<hostdev\>" on page 267

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x1" devno="0xa30e"/>
    <boot order="1" loadparm="2"/>
</disk>
```

<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=<u>on</u> | 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 97

Parent elements
"<keywrap>" on page 275

Child elements
None.

Example

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

<cmdline>

Specifies arguments to be passed to the kernel (or installer) at boot time.

Text content
Command line arguments using the same syntax as if they were specified in the command line.

Selected attributes
None.

Usage
"Configuring a kernel image file as IPL device" on page 80

Parent elements
"<os>" on page 290

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 95

Parent elements

"<devices>" on page 254

Child elements

- "<log>" on page 278
- <protocol>
- "<target> as child element of <console>" on page 300

Example

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

<controller>

Specifies a device controller for a virtual server.

Text content
None.

Selected attributes

type=scsi | virtio-serial
  Specifies the type of controller.

index
  This decimal integer specifies the controller index, which is referenced by the attached host device.

  To reference a controller, use the controller attribute of the address element as child of the hostdev element.

scsi type-specific attributes:

  model=virtio-scsi
    Optional; specifies the model of the controller.

Usage
"Configuring a SCSI tape or medium changer device" on page 117

Parent elements
"<devices>" on page 254

Child elements

• "<address> as child element of <controller>, <disk>, and <memballoon>" on page 239
• "<driver> as child element of <controller>" on page 258

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="drive" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
</devices>
```

<cpu>

Specifies the features of the virtual CPUs of a virtual server.

Text content
None.

Selected attributes

match=exact
> The virtual CPU provided to the virtual server must match the specification. The virtual server can be started only if the specified CPU model is supported. This is the default.

mode= <u>custom</u> | host-model | host-passthrough

> **custom**
> > The <cpu> element and its nested elements define the CPU to be presented to the virtual server. This mode ensures that a persistent virtual server uses the same CPU model on any KVM host. The virtual server can be started only if the KVM host supports the specified CPU model. This is the default.

> **host-model**
> > The CPU definition is derived from the KVM host CPU. On a running virtual server, the model is expanded to an explicit specification in the libvirt-internal configuration.

> **host-passthrough**
> > The CPU model of the KVM host is used. On a running virtual server, the model remains host-passthrough. It is not expanded to an explicit specification in the libvirt-internal configuration.

Usage
"Configuring the CPU model" on page 87

Parent elements
"<domain>" on page 256

Child elements

- "<model> as a child element of <cpu>" on page 285
- "<feature>" on page 264

Example

This example sets the CPU model to the default for z14:

```
<cpu mode="custom">
   <model>z14</model>
</cpu>
```

<cputune>

Groups CPU tuning parameters.

Text content

None.

Selected attributes

None.

Usage

"Tuning virtual CPUs" on page 86

Parent elements

"<domain>" on page 256

Child elements

"<shares>" on page 294

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>
```

<devices>

Specifies the virtual network and block devices of the virtual server.

Text content

None.

Selected attributes

None.

Usage

Chapter 13, "Configuring devices," on page 101

Parent elements

"<domain>" on page 256

Child elements

- "<console>" on page 250
- "<controller>" on page 251
- "<disk>" on page 255
- "<emulator>" on page 263
- "<hostdev>" on page 267
- "<interface>" on page 272
- "<memballoon>" on page 280
- "<watchdog>" on page 305

Example

```
<devices>
    <interface type="direct">
        <source dev="enc1108" 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>

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=<u>disk</u> | cdrom
> Optional; Indicates how the virtual block device is to be presented to the virtual server.

Usage

- Chapter 13, "Configuring devices," on page 101
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 122

Parent elements

"<devices>" on page 254

Child elements

- "<address> as child element of <controller>, <disk>, and <memballoon>" on page 239
- <blockio>
- "<boot>" on page 247
- "<driver> as child element of <disk>" on page 259
- "<geometry>" on page 265
- "<readonly>" on page 291
- "<shareable>" on page 293
- "<source> as child element of <disk>" on page 296
- "<target> as child element of <disk>" on page 301

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0009"/>
</disk>
```

\<domain\>

Is the root element of a domain configuration-XML.

Text content

None.

Selected attributes

type="kvm"
 Specifies the virtual server type.

Usage

"Domain configuration-XML" on page 75

Parent elements

None.

Child elements

- \<clock\>
- "\<console\>" on page 250
- "\<controller\>" on page 251
- "\<cputune\>" on page 253
- \<currentMemory\>
- "\<devices\>" on page 254
- "\<iothreads\>" on page 273
- "\<launchSecurity\>" on page 276
- \<memory\>
- "\<memoryBacking\>" on page 283
- \<name\>
- "\<on_crash\>" on page 288
- \<on_poweroff\>
- \<on_reboot\>
- \<os\>
- \<uuid\>
- "\<vcpu\>" on page 303

<driver>

Specifies details that are related to the user space process used to implement the block device.

Parent elements

Several elements can be a parent of the driver element. The following sections provide details for <driver> in the context of the most relevant parents for z/Architecture:

- <driver> as a child element of <controller>
- <driver> as a child element of <disk>
- <driver> as a child element of <hostdev>
- <driver> as a child element of <interface>

<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.

iommu="on"
> For virtio devices on guests that are to run in IBM Secure Execution mode: with this attribute, you enable the device to use the guest's bounce buffer. Use this attribute only if the launchSecurity element at the virtual server level is not an option.
>
> With the preferred launchSecurity element, you enable all virtio devices to use the bounce buffer by default, see "Preparing the virtual server" on page 141.

Usage
- "Configuring a virtual HBA" on page 115
- Chapter 16, "Configuring for IBM Secure Execution for Linux," on page 141

Parent elements
"<controller>" on page 251

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=<u>report</u> | 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!

iommu="on"
　　For virtio devices on guests that are to run in IBM Secure Execution mode: with this attribute, you enable the device to use the guest's bounce buffer. Use this attribute only if the launchSecurity element at the virtual server level is not an option.

　　With the preferred launchSecurity element, you enable all virtio devices to use the bounce buffer by default, see "Preparing the virtual server" on page 141.

Usage

- "Configuring a DASD, SCSI, or NVMe disk" on page 103
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 122

- Chapter 16, "Configuring for IBM Secure Execution for Linux," on page 141

Parent elements
"<disk>" on page 255

Child elements
None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xd501"/>
</disk>
```

<driver> as child element of <hostdev>

Defines a PCI device as a VFIO pass-through device.

Text content

None.

Selected attributes

name="vfio"
　　Specifies the virtualization technique as VFIO.

Usage

• "Configuring pass-through PCI devices" on page 131

Parent elements

• "<hostdev>" on page 267

Child elements

None.

Example

```
<hostdev mode="subsystem" type="pci">
  <driver name="vfio"/>
  <source>
    <address domain="0x0002" bus="0x00" slot="0x00" function="0x0"/>
  </source>
  <address type="pci">
    <zpci uid="0x0001" fid="0x00000000"/>
  </address>
</hostdev>
```

<driver> as child element of <interface>

Defines a PCI network device as a VFIO pass-through device or configures a virtio network device to use the guest's bounce buffer.

Text content
None.

Selected attributes

iommu="on"
> For virtio devices on guests that are to run in IBM Secure Execution mode: with this attribute, you enable the device to use the guest's bounce buffer. Use this attribute only if the launchSecurity element at the virtual server level is not an option.
>
> With the preferred launchSecurity element, you enable all virtio devices to use the bounce buffer by default, see "Preparing the virtual server" on page 141.

queues=<*n*>
> For virtio interfaces, the number of queues, that the interface uses for parallel processing of incoming and outgoing packets. Specify an integer in the range 2 to the number of available virtual CPUs.

name="vfio"
> Specifies the virtualization technique as VFIO.

Usage

- "Configuring pass-through PCI devices" on page 131
- Chapter 16, "Configuring for IBM Secure Execution for Linux," on page 141
- Enhancing the performance of virtio network interfaces

Parent element

- "<interface>" on page 272

Child elements
None.

Examples

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

```
<interface mode="subsystem" type="hostdev">
   <driver name="vfio"/>
   <source>
      <address domain="0x0004" bus="0x00" slot="0x00" function="0x0"/>
   </source>
   <address type="pci">
      <zpci uid="0x0002" fid="0x00000011"/>
   </address>
</intrface>
```

```
<interface type="direct">
      <source network="net01"/>
      <model type="virtio"/>
      <driver queues="2"/>
</interface>
```

<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 93
- "Displaying the current libvirt-internal configuration" on page 158

Parent elements

"<devices>" on page 254

Child elements

None.

Example

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

<feature>

Adds or removes a CPU feature in a CPU model specification.

Text content

None.

Selected attributes

name
>   Specifies one of the features as shown in the output of the **qemu-system-s390x -cpu help** command.

policy= require | disable

>   **require**
>   >   Adds the feature to the CPU definition specified with the name attribute.

>   **disable**
>   >   Removes the feature from the CPU definition specified with the name attribute.

Usage

"Configuring the CPU model" on page 87

Parent elements

"<cpu>" on page 252

Child elements

None.

Example

This example uses the default for CPU model z14 as a starting point, but removes the iep feature:

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

<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, SCSI, or NVMe disk" on page 103

Parent elements

"<disk>" on page 255

Child elements

None.

Example

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

<graphics>

Specifies a graphics processing unit (GPU) device that provides remote frame buffers for workstation clients.

Text content

None.

Selected attributes

type= "vnc"
Supports workstation clients that use Virtual Network Computing (VNC).

autoport="yes"
Optional and mutually exclusive with port; automatically allocates the TCP port to use.

port=*<port_number>*
Optional and mutually exclusive with autoport; specifies the TCP port to use.

Usage

"Configuring a virtual graphics card" on page 113

Parent elements

"<devices>" on page 254

Child elements

"<listen>" on page 277

Example

```
<domain type="kvm">
    ...
    <devices>
        ...
        <graphics type="vnc" autoport="yes">
            <listen type="address" address="0.0.0.0"/>
        </graphics>
        <input type="keyboard" bus="virtio"/>
        <input type="mouse" bus="virtio"/>
        ...
    </devices>
    ...
</domain>
```

\<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. The virtualization of the pass-through device can be SCSI-based or it can be based on the VFIO framework (see "Device virtualization techniques" on page 8).

Text content
None.

Selected attributes

mode="subsystem"
Specifies the pass-through mode. Other relevant tributes depend on the specified type.

    type="scsi"
    Specifies SCSI as the type of device that is assigned to a virtual server.

        rawio=<u>no</u>| 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=<u>filtered</u> | unfiltered
        Indicates whether the kernel filters unprivileged SG_IO commands for the device.

    type="pci"
    Specifies PCI as the type of device that is assigned to a virtual server.

        managed=yes | <u>no</u>
        For managed="yes", automatically handles the host setup of the PCI device. When the virtual server is started or the device is hotplugged to a running virtual server, the device is detached from the host and assigned to the virtual server. When the virtual server is stopped or detaches the device, it is returned to the host.

    type="mdev"
    Specifies a VFIO mediated device as the type of device that is assigned to a virtual server.

        model=vfio-ccw | vfio-ap
        Specifies the device driver that controls the host resource that the mediated device represents. For a pass-through DASD, the value is `vfio-ccw`, and for pass-through cryptographic adapter resources the value is `vfio-ap`.

Usage

- "Configuring a SCSI tape or medium changer device" on page 117
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 122
- "Configuring a pass-through DASD" on page 130
- "Configuring pass-through PCI devices" on page 131

Parent elements
"\<devices\>" on page 254

Child elements

- "\<address\> as child element of \<hostdev\> or \<disk\>" on page 240
- "\<boot\>" on page 247
- "\<readonly\>" on page 291

- "<shareable>" on page 293
- "<source> as child element of <hostdev>" on page 298

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="drive" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
</devices>
```

<hugepages>

Sets 1 MB huge pages of the KVM host memory as the backing for the virtual memory of the virtual server.

Text content
None.

Selected attributes
None.

Usage
"Configuring huge pages" on page 91

Parent elements
"<memoryBacking>" on page 283

Child elements

None.

Example

```
<domain type="kvm">
    ...
    <memoryBacking>
        <hugepages/>
    </memoryBacking>
    ...
</domain>
```

\<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 80

Parent elements

"\<os\>" on page 290

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>
```

<input>

Enables remote input devices for graphical application.

Text content

None.

Selected attributes

type= keyboard | mouse

> **keyboard**
>> Enables the keyboard of a remote workstation as an input device for graphical application on a KVM guest.

> **mouse**
>> Enables the mouse of a remote workstation as an input device for graphical application on a KVM guest.

bus="virtio"
> Specifies the device type on the virtual server. Specify "virtio".

Usage

"Configuring a virtual graphics card" on page 113

Parent elements

"<devices>" on page 254

Child elements

None.

Example

```
<input type="keyboard" bus="virtio"/>
```

<interface>

Specifies a virtual Ethernet device for a virtual server.

Text content
None.

Selected attributes

type = direct | bridge | network | hostdev

Specifies the type of connection:

direct
Creates a MacVTap interface.

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

network
Attaches to a virtual network as configured with a network configuration-XML.

hostdev
Configures a VFIO PCI pass-through network device.

trustGuestRxFilters = <u>no</u> | yes
Only valid if type = "direct".

Set this attribute to "yes" to allow the virtual server to change its MAC address. As a consequence, the virtual server can join multicast groups. The ability to join multicast groups is a prerequisite for the IPv6 Neighbor Discovery Protocol (NDP).

Setting trustGuestRxFilters to "yes" has security implications, because it allows the virtual server to change its MAC address and thus to receive all frames delivered to this address.

Usage
- "Configuring virtual Ethernet devices" on page 124
- "Configuring pass-through PCI devices" on page 131

Parent elements
"<devices>" on page 254

Child elements
- "<address> as child element of <interface>" on page 242
- "<mac>" on page 279
- "<model> as a child element of <interface>" on page 286
- "<source> as child element of <interface>" on page 299
- "<virtualport> as a child element of <interface>" on page 304
- "<driver> as child element of <interface>" on page 262

Example

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

<iothreads>

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

The use of I/O threads improves the performance of I/O operations of the virtual server. If this element is not specified, no I/O threads are provided.

Text content

Natural number specifying the number of threads.

Selected attributes

None.

Usage

"Configuring devices with the virtual server" on page 94

Parent elements

"<domain>" on page 256

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 80

Parent elements

"<os>" on page 290

Child elements

None.

Example

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

<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 97

Parent elements

"<domain>" on page 256

Child elements

"<cipher>" on page 248

Example

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

<launchSecurity>

Configures architecture-specific security settings for virtual servers.

Text content

None.

Selected attributes

type="s390-pv"
> Prepares the virtual server for a guest in IBM Secure Execution mode.

Usage

Chapter 16, "Configuring for IBM Secure Execution for Linux," on page 141

Parent elements

"<domain>" on page 256

Child elements

None.

Example

```
<domain type="kvm">
    ...
    <launchSecurity type="s390-pv"/>
    ...
</domain>
```

<listen>

Specifies where on the virtual server a graphics processing unit (GPU) listens to workstation clients.

Text content

None.

Selected attributes

type= "address"
   Sets address as the type of object to listen to.

address=*<IP_address>*
   Specifies a host name or IP address. The specification 0.0.0.0 sets all IP addresses of the virtual server.

Usage

"Configuring a virtual graphics card" on page 113

Parent elements

"<graphics>" on page 266

Child elements

None.

Example

```
...
<graphics type="vnc" autoport="yes"
   <listen type="address" address="0.0.0.0"/>
</graphics>
...
```

<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=<u>off</u> | on
>   Specifies whether the information in the file is preserved (append="on") or overwritten (append="off")
>   on a virtual server restart.

Usage

Parent elements

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 124

Parent elements

"<interface>" on page 272

Child elements

None.

Example

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

<memballoon>

Specifies memory balloon devices.

Text content

None.

Selected attributes

model=none
> Suppresses the automatic creation of a default memory balloon device.

Usage

"Suppressing the automatic configuration of a default memory balloon device" on page 99

Parent elements

"<devices>" on page 254

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 | <u>KiB</u> | 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 89
- "Configuring the collection of QEMU core dumps" on page 92

Parent elements

"<domain>" on page 256

Child elements

None.

Example

This example:

domain configuration-XML

- Configures 524,288 KB of virtual memory.
- Excludes the virtual memory from QEMU core dumps.

```
<memory dumpCore="off" unit="KB">524288</memory>
```

<memoryBacking>

Configures the KVM host memory to be used as to back the virtual memory of the virtual server.

Text content
None.

Selected attributes
None.

Usage
"Configuring huge pages" on page 91

Parent elements
"<domain>" on page 256

Child elements
"<hugepages>" on page 269

Example

```
<domain type="kvm">
    ...
    <memoryBacking>
        <hugepages/>
    </memoryBacking>
    ...
</domain>
```

<memtune>

Groups memory tuning parameters.

Text content

None.

Selected attributes

None.

Usage

- Chapter 28, "Memory management," on page 203
- "Tuning virtual memory" on page 89

Parent elements

"<domain>" on page 256

Child elements

"<soft_limit>" on page 295

Example

This example ...

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

\<model> as a child element of \<cpu>

Specifies a CPU model.

Text content

CPU model as shown in the output of the **virsh domcapabilities** command. Eligible values are specified with \<model> tags that have the attribute useable="yes".

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

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

Selected attributes

None.

Usage

"Configuring the CPU model" on page 87

Parent elements

"\<cpu>" on page 252

Child elements

None.

Example

This example sets the CPU model to the default for z14:

```
<cpu mode="custom">
  <model>z14</model>
</cpu>
```

<model> as a child element of <interface>

Specifies the interface model type.

Text content

None.

Selected attributes

type=virtio
Specifies the interface model type virtio.

Usage

- "Configuring a MacVTap interface" on page 124
- "Configuring a virtual switch" on page 126

Parent elements

"<interface>" on page 272

Child elements

None.

Example

This example configures a virtio interface:

```
<interface type="direct">
    <source dev="enca100" mode="bridge"/>
    <model type="virtio"/>
</interface>
```

\<name\> as a child element of \<domain\>

Assigns a unique name to the virtual server.

Text content
Unique alphanumeric name for the virtual server.

Selected attributes
None.

Usage
"Domain configuration-XML" on page 75

Parent elements
"\<domain\>" on page 256

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\>

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.

coredump-destroy
> Write a core dump to `/var/lib/libvirt/qemu/dump`, then stop the virtual server and release all resources.

coredump-restart
> Write a core dump to `/var/lib/libvirt/qemu/dump`, then restart the virtual server.

Selected attributes

None.

Usage

Parent elements

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 384.

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 75

Parent elements

"<domain>" on page 256

Child elements

None.

Example

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

<os>

Groups the operating system parameters.

Text content
None.

Selected attributes
None.

Usage
"Domain configuration-XML" on page 75

Parent elements
"<domain>" on page 256

Child elements

- "<type> as child element of <os>" on page 302
- "<kernel>" on page 274
- "<initrd>" on page 270
- "<cmdline>" on page 249

Example

```
<os>
    <type arch="s390x" machine="s390-ccw-virtio">hvm</type>
    <initrd>/boot/initramfs-3.9.3-60.x.20130605-s390xrhel.img</initrd>
    <kernel>/boot/vmlinuz-3.9.3-60.x.20130605-s390xrhel</kernel>
    <cmdline>rd.md=0 rd.lvm=0 LANG=en_US.UTF-8
            KEYTABLE=us SYSFONT=latarcyrheb-sun16 rd.luks=0
            root=/dev/disk/by-path/ccw-0.0.e714-part1
            rd.dm=0 selinux=0 CMMA=on
            crashkernel=128M plymouth.enable=0
    </cmdline>
</os>
```

<readonly>

Indicates that a device is readonly.

Text content

None.

Selected attributes

None.

Usage

"Configuring a virtual SCSI-attached CD/DVD drive" on page 122

Parent elements

- "<disk>" on page 255
- "<hostdev>" on page 267

Child elements

None.

Example

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

<rng>

Specifies a random number generator.

Text content

None.

Selected attributes

model=virtio
Specifies the random number generator device type.

Usage

"Configuring a random number generator" on page 128

Parent elements

"<devices>" on page 254

Child elements

"<backend>" on page 246

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 255
- "<hostdev>" on page 267

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="drive" controller="0" bus="0" target="0" unit="0"/>
        <shareable/>
    </hostdev>
</devices>
```

\<shares\>

Specifies the initial CPU weight.

The CPU shares of a virtual server are calculated from the CPU weight of all virtual servers running on the host. For example, a virtual server that is configured with value 2048 gets twice as much CPU time as a virtual server that is configured with value 1024.

Text content

Natural number specifying the CPU weight.

- Valid values are in the natural numbers between 2 and 262144.
- The default value is 1024.

Selected attributes

None.

Usage

- "Tuning virtual CPUs" on page 86
- "CPU weight" on page 200

Parent elements

"\<cputune\>" on page 253

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 28, "Memory management," on page 203
- "Configuring virtual memory" on page 89

Parent elements
"<memtune>" on page 284

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=<u>mandatory</u> | 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, SCSI, or NVMe disk" on page 103
- "Configuring an image file as storage device" on page 109
- "Configuring a volume as storage device" on page 111
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 122

Parent elements
"<disk>" on page 255

See also:

- "<source> as child element of <interface>" on page 299

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 an NVMe device as virtual block device:

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/disk/by-path/pci-1003:00:00.0-nvme-1"/>
    <target dev="vdc" bus="virtio"/>
</disk>
```

- This example configures a file as virtual block device:

```
<disk type="file" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native"/>
    <source file="/var/lib/libvirt/images/disk.img"/>
    <target dev="vda1" bus="virtio"/>
</disk>
```

- This example configures a storage pool as virtual block device:

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

<source> as child element of <hostdev>

Specifies the host view of a pass-through device configuration.

Text content
None.

Selected attributes
None.

Usage

- "Configuring pass-through PCI devices" on page 131
- "Configuring a SCSI tape or medium changer device" on page 117

Parent elements

- "<hostdev>" on page 267

Child elements

- "<address> as child element of <source>" on page 244
- "<adapter> as child element of <source>" on page 238

Example

```
<devices>
    ...
    <hostdev mode="subsystem" type="scsi">
        <source>
            <adapter name="scsi_host0"/>
            <address bus="0" target="0" unit="0"/>
        </source>
        <address type="drive" controller="0" bus="0" target="0" unit="0"/>
    </hostdev>
    ...
    <hostdev mode="subsystem" type="pci" managed="yes">
        <driver name="vfio"/>
        <source>
            <address domain="0x0002" bus="0x00" slot="0x00" function="0x0"/>
        </source>
        <address type="pci">
            <zpci uid="0x0001" fid="0x00000000"/>
        </address>
    </hostdev>
    ...
</devices>
```

\<source\> as child element of \<interface\>

Specifies the host view of a network interface configuration.

Text content
None.

Selected attributes

dev
> Specifies the network interface.

mode=bridge | vepa
> Optional and mutually exclusive with network; indicates whether packets are delivered to the target device or to the external bridge.

> **bridge**
>> If packets have a destination on the host from which they originated, they are delivered directly to the target. For direct delivery, both origin and destination devices need to be in bridge mode. If either the origin or destination is in vepa mode, a VEPA-capable bridge is required.

> **vepa**
>> All packets are sent to the external bridge. If packets have a destination on the host from which they originated, the VEPA-capable bridge will return the packets to the host.

network
> Optional and mutually exclusive with mode; specifies the name of a virtual network.

Usage
- "Configuring virtual Ethernet devices" on page 124
- "Configuring pass-through PCI devices" on page 131

Parent elements
"\<interface\>" on page 272

Child elements
"\<address\> as child element of \<interface\>" on page 242

Example

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

<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 95

Parent elements
"<console>" on page 250

See also:

- "<target> as child element of <disk>" on page 301

Child elements
None.

Example

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

\<target\> as child element of \<disk\>

Specifies the virtual server view of a device that is provided from the host.

Text content
None.

Selected attributes

dev

Unique name for the device of the form vd*\<x\>*, where *\<x\>* can be one or more letters.

If no address element is specified, the order in which device bus-IDs are assigned to virtio block devices is determined by the order of the target dev attributes.

bus=virtio

Specifies the device type on the virtual server. Specify "virtio".

Usage

- "Configuring a DASD, SCSI, or NVMe disk" on page 103
- "Configuring an image file as storage device" on page 109
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 122

Parent elements
"\<disk\>" on page 255

Child elements
None.

Example

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

<type> as child element of <os>

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 75
- "Definition of a virtual server on different hosts using the same configuration-XML" on page 162

Parent elements

"<os>" on page 290

Child elements

None.

Example

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

<vcpu>

Specifies the number of virtual CPUs for a virtual server.

Text content
Natural number specifying the maximum number of available virtual CPUs.

Selected attributes

current
> Optional; specifies the number of virtual CPUs available at startup.
>
> The value of the current attribute is limited by the maximum number of available virtual CPUs. If you do not specify the current attribute, the maximum number of virtual CPUs is available at startup.

Usage
"Configuring virtual CPUs" on page 85

Parent elements
"<domain>" on page 256

Child elements
None.

Example

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

<virtualport> as a child element of <interface>

Specifies the type of a virtual switch.

Text content

None.

Selected attributes

type=openvswitch
    Specifies the type of the virtual switch.

Usage

• "Configuring a virtual switch" on page 126

Parent elements

"<interface>" on page 272

Child elements

None.

Example

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

\<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=<u>reset</u> | 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

Parent elements

Child elements
None.

Example

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

<zpci>

complements the address of a PCI pass-through device on the guest with identifiers that are specific to the z/Architecture and that persist across reboots.

Text content

None.

Selected attributes

uid

Specifies a user-defined ID that has a similar role to a domain on the host, and that persists across reboots. Valid values are in the range `0x0001` - `0xffff`.

fid

Specifies a PCI function identifier that persists across reboots. Valid values are in the range `0x00000000` - `0xffffffff`. According to best practices, specify values that are supported by your Z hardware.

Usage

"Configuring pass-through PCI devices" on page 131

Parent element

"<address> as child element of <hostdev> or <disk>" on page 240

Child elements

None.

Examples

```
<devices>
   ...
   <hostdev mode="subsystem" type="pci" managed="yes">
      <driver name="vfio"/>
      <source>
        <address domain="0x0002" bus="0x00" slot="0x00" function="0x0"/>
      </source>
      <address type="pci">
        <zpci uid="0x0001" fid="0x00000000"/>
      </address>
   </hostdev>
   ...
</devices>
```
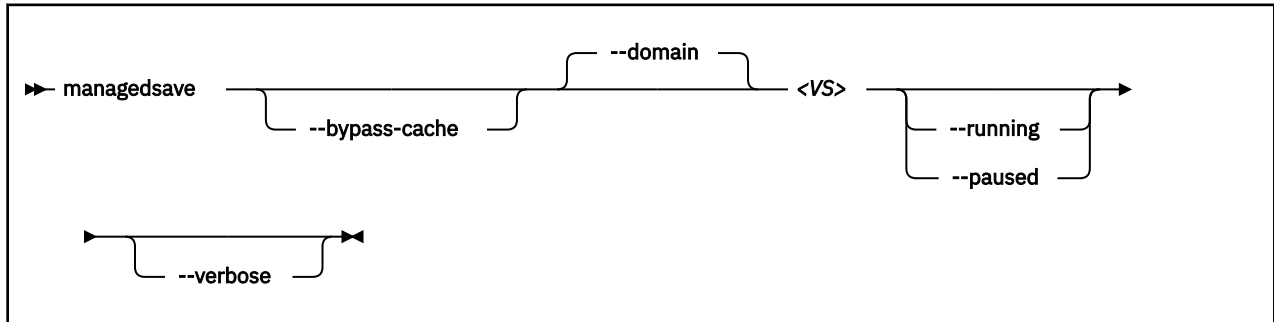
Network configuration-XML

The network configuration-XML describes virtual networks that connect KVM virtual servers among themselves and to external networks. For related virsh commands, see "Network management virsh commands" on page 395.

- "<bridge>" on page 308
- "<dhcp>" on page 309
- "<forward>" on page 310
- "<ip>" on page 311
- "<name> as a child element of <network>" on page 312
- "<network>" on page 313
- "<virtualport> as a child element of <network>" on page 314

<bridge>

Configures the bridge device that is used to set up the virtual network.

Text content

None.

Selected attributes

name
Specifies a name for the bridge.

Usage

Chapter 15, "Configuring virtual networks," on page 139

Parent elements

"<network>" on page 313

Child elements

None.

Example

```
<network>
  ...
  <bridge name="virbr2"/>
  ...
</network>
```

\<dhcp>

Configures DHCP services for the virtual network.

Text content

None.

Selected attributes

None.

Usage

Chapter 15, "Configuring virtual networks," on page 139

Parent elements

"\<ip>" on page 311

Child elements

\<range>

Example

```
<network>
  ...
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

<forward>

Configures the forwarding mode for the bridge that connects the virtual network to a physical LAN. Omitting this tag results in an isolated network that can connect guests.

Text content
None.

Selected attributes

mode=nat | bridge | route
Specifies the method of forwarding for the LAN connection.

nat
Configures a bridge with network address translation (NAT). All guest traffic to the physical network is routed through the host's routing stack and uses the host's public IP address. This mode supports only outbound traffic.

bridge
Configures a bridge based on an already configured Open vSwitch (see "Preparing a virtual switch" on page 49).

route
Configures a bridge with IP routing. Bridges with IP routing link to a virtual IP subnet on the host. Traffic to and from virtual servers that are connected to that subnet are then handled by the IP protocol.

Usage
Chapter 15, "Configuring virtual networks," on page 139

Parent elements
"<network>" on page 313

Child elements
<nat>

Example

```
<network>
  <name>net0</name>
  <uuid>fec14861-35f0-4fd8-852b-5b70fdc112e3</uuid>
  <forward mode="nat">
    <nat>
      <port start="1024" end="65535"/>
    </nat>
  </forward>
  <bridge name="virbr0" stp="on" delay="0"/>
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

<ip>

Configures IP addresses for the virtual network.

Text content

None.

Selected attributes

address
>   Sets the IP address for the bridge device. The value must be a valid IPv4 address.

netmask
>   Specifies a subnet mask for the virtual network.

Usage

Chapter 15, "Configuring virtual networks," on page 139

Parent elements

"<network>" on page 313

Child elements

"<dhcp>" on page 309

Example

```
<network>
  ...
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

<name> as a child element of <network>

Assigns a short name to a virtual network.

Text content

Alphanumeric name for the virtual network. The name must be unique for the scope of the KVM host.

Selected attributes

None.

Usage

Chapter 15, "Configuring virtual networks," on page 139

Parent elements

"<network>" on page 313

Child elements

None.

Example

```
<network>
  <name>net0</name>
  ...
</network>
```

<network>

Is the root element of a network configuration-XML.

Text content

None.

Selected attributes

None.

Usage

Chapter 15, "Configuring virtual networks," on page 139

Parent elements

None.

Child elements

- "<bridge>" on page 308
- "<forward>" on page 310
- "<ip>" on page 311
- "<name> as a child element of <network>" on page 312
- <uuid>
- "<virtualport> as a child element of <network>" on page 314

Example

```
<network>
  <name>ovs</name>
  <forward mode="bridge"/>
  <bridge name="ovs-br0"/>
  <virtualport type="openvswitch"/>
</network>
```

<virtualport> as a child element of <network>

Identifies the bridge of a virtual network as an Open vSwitch.

Text content
None.

Selected attributes

type=openvswitch
> In combination with forwarding mode `bridge` and a <bridge> element, identifies the bridge as an Open vSwitch.

Usage
Chapter 15, "Configuring virtual networks," on page 139

Parent elements
"<network>" on page 313

Child elements
None.

Example

```
<network>
  <name>ovs</name>
  <forward mode="bridge"/>
  <bridge name="ovs-br0"/>
  <virtualport type="openvswitch"/>
</network>
```

Node-device XML

The node-device XML describes resources on the KVM host. A node-device XML file can describe an existing resource as detected by libvirt, or it can provide the specifications for creating a VFIO mediated device. For related virsh commands, see .

<ap-adapter>

Specifies a cryptographic adapter.

Text content

The adapter ID as a two-digit hexadecimal number with a 0x prefix.

Selected attributes

None.

Usage

"Managing mediated devices with libvirt" on page 62

Parent elements

"<capability>" on page 319

Child elements

None.

Example

```
<device>
    <name>ap_08_0001</name>
    ...
    <capability type="ap_queue">
        <ap-adapter>0x08</ap-adapter>
        <ap-domain>0x0001</ap-domain>
    </capability>
</device>
```

\<ap-domain>

Specifies a cryptographic domain.

Text content

The domain ID as a four-digit hexadecimal number with a `0x` prefix.

Selected attributes

None.

Usage

"Managing mediated devices with libvirt" on page 62

Parent elements

"\<capability>" on page 319

Child elements

None.

Example

```
<device>
    <name>ap_08_0001</name>
    ...
    <capability type="ap_queue">
        <ap-adapter>0x08</ap-adapter>
        <ap-domain>0x0001</ap-domain>
    </capability>
</device>
```

<attr>

Specifies an attribute of a VFIO mediated device.

Text content

None.

Selected attributes

name
    The name of the attribute.
value
    The value of the attribute.

Usage

"Managing mediated devices with libvirt" on page 62

Parent elements

"<capability>" on page 319

Child elements

None.

Example

```
<device>
    <parent>ap_matrix</parent>
    <capability type="mdev">
        <type id="vfio_ap-passthrough"/>
        <attr name="assign_adapter" value="0x09"/>
        <attr name="assign_domain" value="0x0001"/>
    </capability>
</device>
```

\<capability>

Categorizes host devices according to the resources they can provide to a virtual server and describes these resources.

Text content

None.

Selected attributes

type=ap_card | ap_queue | ap_matrix | ccw | css | mdev | mdev_types
   Categorizes the host device. The child elements that are used to describe the host resources depend on the type.

   ap_card
      Categorizes the host device as a cryptographic adapter.

   ap_queue
      Categorizes the host device as an AP queue, which corresponds to a domain on a cryptographic adapter.

   ap_matrix
      Categorizes the host device as a matrix of AP queues.

   ccw
      Categorizes the host device as a z/Architecture CCW device.

   css
      Categorizes the host device as a subchannel of the z/Architecture channel subsystem (CSS).

   mdev
      Categorizes the host device as a VFIO mediated device.

   mdev_types
      Categorizes the host device as a potential parent of a VFIO mediated device in the libvirt hierarchy of host devices.

   The values for the type attribute can be used for the `--cap` parameter of the **nodedev-list** command to filter the command output by resource type.

Usage

"Managing mediated devices with libvirt" on page 62

Parent elements

• "\<device> as a root element" on page 322
• \<capability>

Child elements

Depend on the capability type.

ap_card

   • "\<ap-adapter>" on page 316

ap_queue

   • "\<ap-adapter>" on page 316
   • "\<ap-domain>" on page 317

ap_matrix

   • \<capability>

> **Note:** <capability> as a child element of <capability> always is of type mdev_types.

ccw

- <cssid>
- <ssid>
- <devno>

css

- <cssid>
- <ssid>
- <devno>
- <capability>

Note: <capability> as a child element of <capability> always is of type mdev_types.

mdev

- "<attr>" on page 318
- <iommuGroup>
- <type>
- "<uuid>" on page 327

mdev_types

- <type>

Examples

This example shows the capabilities of a CSS subchannel.

```
<device>
    <name>css_0_0_0071</name>
    ...
    <capability type="css">
        <cssid>0x0</cssid>
        <ssid>0x0</ssid>
        <devno>0x0071</devno>
    </capability>
    ...
</device>
```

This example shows the capabilities of a VFIO mediated device.

```
<device>
    <name>mdev_96c03c6a_a109_44fe_816a_ba371542164b</name>
    ...
    <capability type="mdev">
        <type id='vfio_ccw-io'/>
        <iommuGroup number='1'/>
    </capability>
</device>
```

This example shows the capabilities of the ap_matrix host device. In the libvirty host-device hierarchy, this device is the common parent for AP mediated devices. It describes common properties for AP mediated devices.

```
<device>
    <name>ap_matrix</name>
    ...
    <capability type="ap_matrix">
        <capability type="mdev_types'">
            <type id="vfio_ap-passthrough">
                <name>VFIO AP Passthrough Device</name>
                <deviceAPI>vfio-ap</deviceAPI>
                <availableInstances>65534</availableInstances>
            </type>
        </capability>
    </capability>
</device>
```

<device> as a root element

Is the root element of a node-device XML.

Text content
None.

Selected attributes
None.

Usage

Parent elements
None.

Child elements

-
- <devnode>
- <driver>
-
-
-

Example

```
<device>
  <name>mdev_96c03c6a_a109_44fe_816a_ba371542164b</name>
  <path>/sys/devices/css0/0.0.0072/96c03c6a-a109-44fe-816a-ba371542164b</path>
  <parent>css_0_0_0072</parent>
  <driver>
    <name>vfio_mdev</name>
  </driver>
  <capability type="mdev">
    <type id="vfio_ccw-io"/>
    <uuid>96c03c6a-a109-44fe-816a-ba371542164b</uuid>
    <iommuGroup number="1"/>
  </capability>
</device>
```

\<name\> as a child element of \<device\>

Specifies the name, in libvirt, of a host device.

Text content

The name that libvirt uses to identify the host device.

Selected attributes

None.

Usage

"Managing mediated devices with libvirt" on page 62

Parent elements

"\<device\> as a root element" on page 322

Child elements

None.

Example

```
<device>
  <name>mdev_96c03c6a_a109_44fe_816a_ba371542164b</name>
  ...
</device>
```

<parent>

In the hierarchy of host devices as detected by libvirt, specifies the name of the parent of a host device.

Text content

The name that libvirt uses to identify the parent of the host device. For host devices that do not have a parent, the value is "computer".

Selected attributes

None.

Usage

"Managing mediated devices with libvirt" on page 62

Parent elements

"<device> as a root element" on page 322

Child elements

None.

Example

```
<device>
  ...
  <parent>css_0_0_0072</parent>
  ...
</device>
```

\<path\> as a child element of \<device\>

Shows the path to the sysfs representation of a host device.

Text content

Read-only value that shows the path to the sysfs representation of the host device.

Selected attributes
None.

Usage
"Managing mediated devices with libvirt" on page 62

Parent elements
"\<device\> as a root element" on page 322

Child elements

None.

Example

```
<device>
  <name>mdev_96c03c6a_a109_44fe_816a_ba371542164b</name>
  <path>/sys/devices/css0/0.0.0072/96c03c6a-a109-44fe-816a-ba371542164b</path>
  ...
</device>
```

\<type\> as child element of \<capability\>

Specifies the category of a VFIO mediated device.

Text content

None.

Selected attributes

id
> The type of a VFIO mediated device. For a KVM host on Z, the type can be `vfio_ccw-io` for a path-through DASD, or it can be `vfio_ap-passthrough` for cryptographic adapter resources.

Usage
"Managing mediated devices with libvirt" on page 62

Parent elements
"\<capability\>" on page 319

Child elements

None.

Example

```
<device>
  <name>mdev_96c03c6a_a109_44fe_816a_ba371542164b</name>
  <path>/sys/devices/css0/0.0.0072/96c03c6a-a109-44fe-816a-ba371542164b</path>
  <parent>css_0_0_0072</parent>
  <driver>
    <name>vfio_mdev</name>
  </driver>
  <capability type="mdev">
    <type id="vfio_ccw-io"/>
    <uuid>96c03c6a-a109-44fe-816a-ba371542164b</uuid>
    <iommuGroup number="1"/>
  </capability>
</device>
```

<uuid>

In the node-device XML context, identifies a VFIO mediated device.

Text content

A universally unique identifier (UUID).

Selected attributes

None.

Usage

"Managing mediated devices with libvirt" on page 62

Parent elements

"<capability>" on page 319

Child elements

None.

Example

```
<device>
  <name>mdev_96c03c6a_a109_44fe_816a_ba371542164b</name>
  <path>/sys/devices/css0/0.0.0072/96c03c6a-a109-44fe-816a-ba371542164b</path>
  <parent>css_0_0_0072</parent>
  <driver>
    <name>vfio_mdev</name>
  </driver>
  <capability type="mdev">
    <type id="vfio_ccw-io"/>
    <uuid>96c03c6a-a109-44fe-816a-ba371542164b</uuid>
    <iommuGroup number="1"/>
  </capability>
</device>
```

Storage pool configuration-XML

The storage pool configuration-XML describes storage pools, which consist of a set of similar volumes. The volume configuration-XML describes the volumes of a storage pool. For related virsh commands, see "Storage pool management virsh commands" on page 416.

- "<host>" on page 329
- "<path> as child element of <pool><target>" on page 330
- "<pool>" on page 331
- "<source> as child element of <pool>" on page 333
- "<target> as child element of <pool>" on page 334

<host>

Specifies the host that stores the network file system backing a storage pool.

Text content

None.

Selected attributes

name
    Host name or IP address of the host.

Usage

Chapter 14, "Configuring storage pools," on page 137

Parent elements

"<source> as child element of <pool>" on page 333

Child elements

None.

Example

```
<pool type="netfs">
    <name>nfspool01</name>
    <source>
        <format type="nfs"/>
        <host name="sandbox.example.com"/>
        <dir path="/srv/nfsexport"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/nfspool01</path>
    </target>
</pool>
```

\<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 14, "Configuring storage pools," on page 137

Parent elements

"\<target\> as child element of \<pool\>" on page 334

Child elements

None.

Example

This example specifies a directory backing a storage pool of type directory:

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

This example specifies an FC-attached SCSI disk backing a storage pool of type file system:

```
<pool type="fs">
    <name>fspool01</name>
    <source>
        <device path="/dev/s356001/fspool"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/fspool01</path>
    </target>
</pool>
```

<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 14, "Configuring storage pools," on page 137

Parent elements

None.

Child elements

- <name>
- "<source> as child element of <pool>" on page 333
- "<target> as child element of <pool>" on page 334

Example

- This example configures a storage pool backed by a directory as shown in Figure 9 on page 17:

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

- This example configures a storage pool backed by a file system:

```
<pool type="fs">
    <name>fspool01</name>
    <source>
        <device path="/dev/s356001/fspool"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/fspool01</path>
    </target>
</pool>
```

- This example configures a storage pool backed by a network file system:

```
<pool type="netfs">
    <name>nfspool01</name>
    <source>
        <format type="nfs"/>
        <host name="sandbox.example.com"/>
        <dir 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 :

```
<pool type="logical">
    <name>lvPool01</name>
    <source>
        <name>lvpool01</name>
        <format type="lvm2"/>
    </source>
</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 14, "Configuring storage pools," on page 137

Parent elements
"<pool>" on page 331

Child elements

- <format>
- <host>
- <dir>

Example

```
<pool type="netfs">
    <name>nfspool01</name>
    <source>
        <format type="nfs"/>
        <host name="sandbox.example.com"/>
        <dir path="/srv/nfsexport"/>
    </source>
    <target>
        <path>/var/lib/libvirt/images/nfspool01</path>
    </target>
</pool>
```

<target> as child element of <pool>

Specifies the directory backing a storage pool.

Text content

None.

Selected attributes

None.

Usage

Chapter 14, "Configuring storage pools," on page 137

Parent elements

"<pool>" on page 331

Child elements

- "<path> as child element of <pool><target>" on page 330
- <permissions>

Example

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

Volume configuration-XML

The volume configuration-XML describes storage pool volumes. For related virsh commands, see "Volume management virsh commands" on page 430.

\<format\>

Specifies the image file format backing the storage pool volume.

Text content

None.

Selected attributes

type=raw | qcow2

Usage

Chapter 14, "Configuring storage pools," on page 137

Parent elements

"\<target\> as child element of \<volume\>" on page 339

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>
```

<key>

Identifies the volume.

Text content

Read-only value that identifies the volume within libvirt.

Selected attributes

None.

Usage

Chapter 14, "Configuring storage pools," on page 137

Parent elements

"<volume>" on page 340

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>
```

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

Shows the fully qualified path of the image file.

Text content

Read-only value that shows the fully qualified path of the image file on the host.

Selected attributes

None.

Usage

Chapter 14, "Configuring storage pools," on page 137

Parent elements

"\<target\> as child element of \<volume\>" on page 339

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>
```

<target> as child element of <volume>

Specifies the image file backing a volume.

Text content

None.

Selected attributes

None.

Usage

Parent elements

Child elements

-
-

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>
```

<volume>

Is the root element of a volume configuration-XML.

Text content
None.

Selected attributes
type=file

Usage
Chapter 14, "Configuring storage pools," on page 137

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>
```

Chapter 37. 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
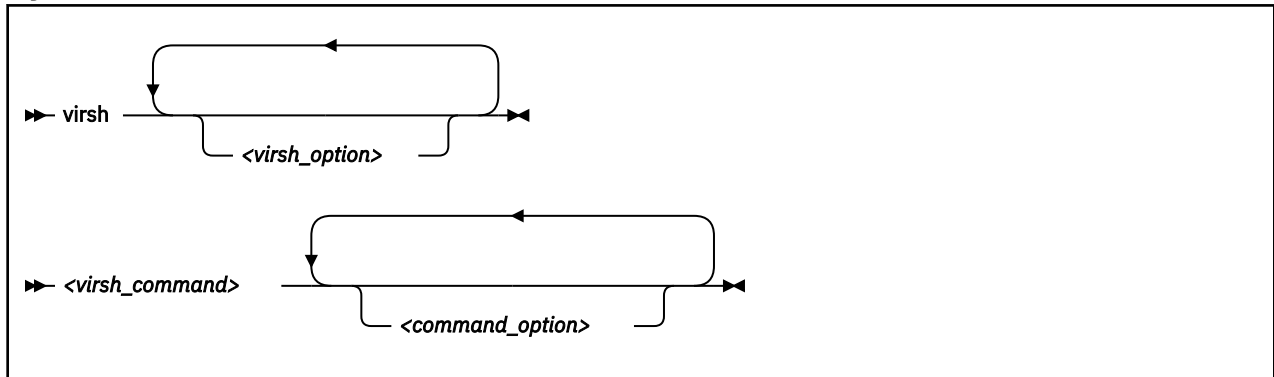
Where:

<virsh_option>
Is a generic **virsh** option.

<virsh_command>
Is a virsh command.

For a complete list of the virsh commands, see libvirt.org/virshcmdref.html.

<command_option>
Is an option that is specific to the specified virsh command.

Selected generic virsh 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.

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
```

Commands grouped by context

Domain management virsh commands

Use the domain management virsh commands to manage KVM virtual servers. For related XML elements, see "Domain configuration-XML" on page 236.

Other virsh commands

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 182

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 .

change-media

Removes a currently provided ISO image from a virtual SCSI-attached CD/DVD drive, or provides a different ISO image.

Syntax
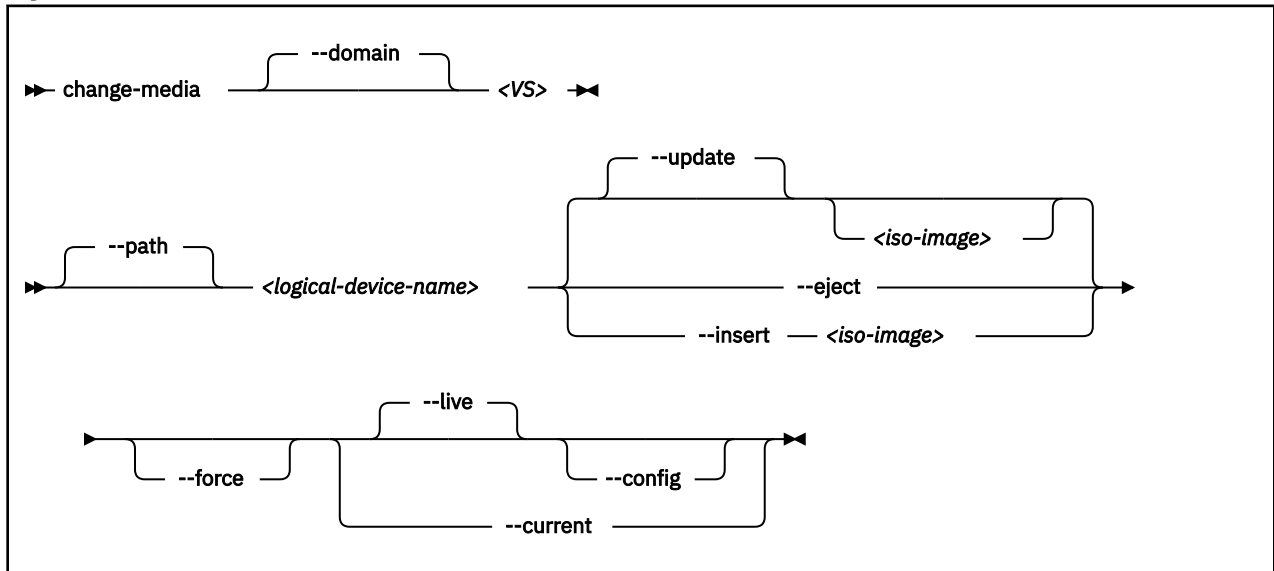
Where:

<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

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

```
►►── console ── <VS> ──┬────────────────────────┬──┬──────────┬──┬───────────┬──►◄
                       └── <alternate-console-name> ──┘  └── --safe ──┘  └── --force ──┘
```
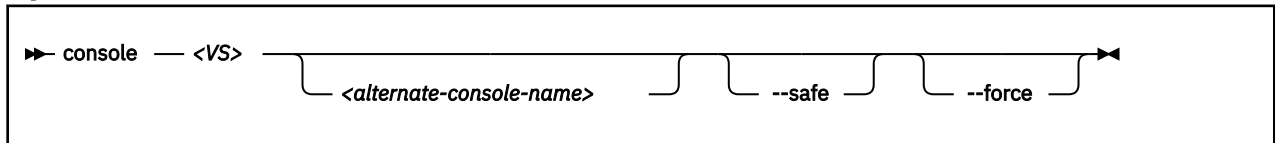
Where:

<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 185

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 ──┘
```
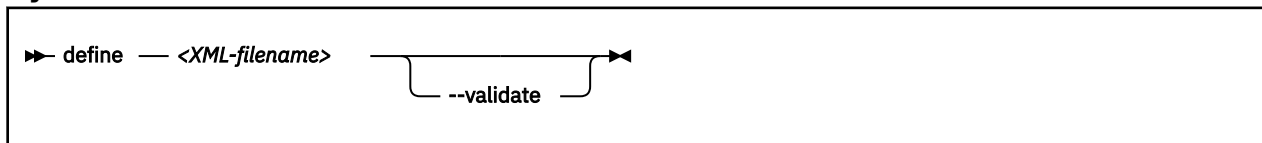
Where:

\<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 148

Example

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

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

destroy

Immediately terminates a virtual server and releases any used resources.

Syntax
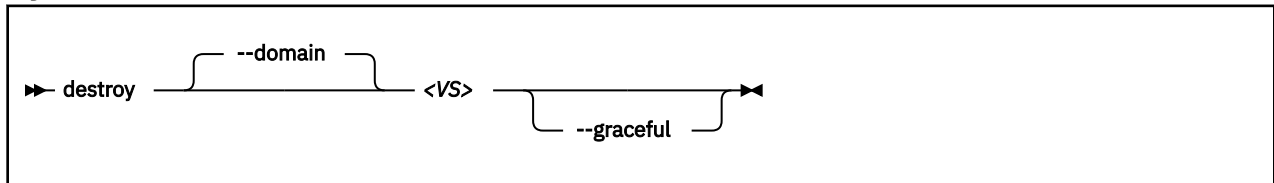
Where:

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

Selected options

--domain
Specifies the virtual server.

--graceful
Tries to properly terminate the virtual server, and only if it is not responding in a reasonable amount of time, it is forcefully terminated.

Virtual server state transitions

From State	To State (reason)
running	shut off (destroyed)
paused	shut off (destroyed)
crashed	shut off (destroyed)

Usage

Example

This example immediately terminates virtual server vserv1.

```
# virsh destroy vserv1
```

detach-device

Detaches a device from a defined virtual server.
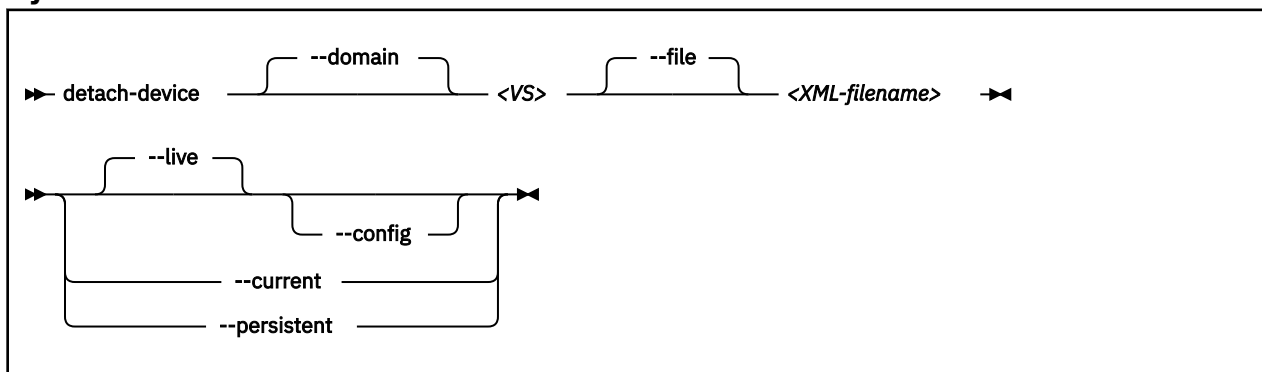
Syntax

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.

Usage
"Detaching a device" on page 183

Example

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

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

domblklist

Displays information about the virtual block devices of a virtual server.

Syntax

Where:

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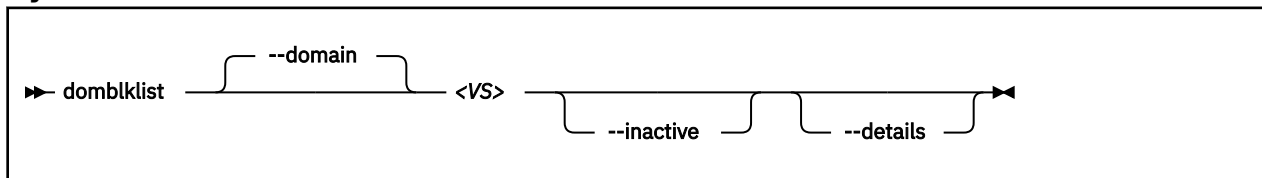
Selected options

--details
Display details, such as device type and value.

--domain
Specifies the virtual server.

--inactive
Lists the block devices that will be used with the next virtual server reboot.

Usage

Example

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

domblkstat

Displays status information about a virtual block device.

Syntax

```
►►─ domblkstat ─┬─────────────┬─ <VS> ── <device-name> ─┬─────────────┬─►◄
                │  ┌─ --domain ─┐                         │             │
                └──┴───────────┴┘                         └─ --human ───┘
```
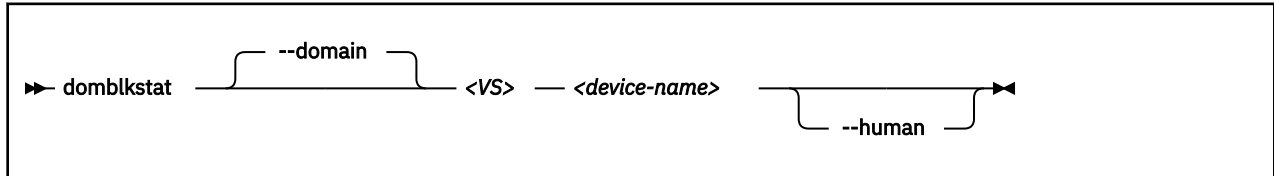
Where:

<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

Example

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

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

Obtain information about the virtual block device vda:

```
#  virsh domblkstat vserv1 vda
vda rd_req 20359
vda rd_bytes 235967488
vda wr_req 4134
vda wr_bytes 52682752
vda flush_operations 1330
vda rd_total_times 49294200385
vda wr_total_times 4403369039
vda flush_total_times 256032781
```

Alternatively, display written-out information:

```
# virsh domblkstat vserv vda --human
Device: vda
 number of read operations:      20359
 number of bytes read:           235967488
 number of write operations:     4348
 number of bytes written:        54353920
 number of flush operations:     1372
 total duration of reads (ns):   49294200385
 total duration of writes (ns):  4626108064
 total duration of flushes (ns): 265417103
```

domcapabilities

Prints an XML document that describes the hypervisor capabilities.

Syntax

```
►►─ domcapabilities ─┬──────────────────────────────┬─►◄
                     └─ --machine ── <machine> ─┘
```
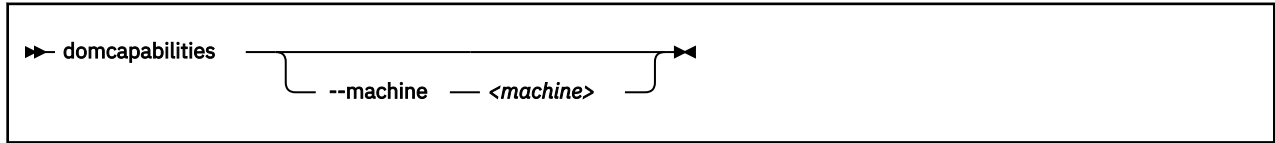
Selected options

<machine>

Is a supported hypervisor version as listed by the **qemu-kvm --machine help** command. If this option is omitted, the XML document for the default hypervisor version is displayed.

Usage

"Configuring the CPU model" on page 87

Example

```
# virsh domcapabilities --machine s390-ccw-virtio-2.8
<domainCapabilities>
  <path>/usr/bin/qemu-system-s390x</path>
  <domain>kvm</domain>
  <machine>s390-ccw-virtio-2.8</machine>
  <arch>s390x</arch>
  <vcpu max='248'/>
  ...
  <cpu>
    <mode name='host-passthrough' supported='yes'/>
    <mode name='host-model' supported='yes'>
      <model fallback='forbid'>z13s-base</model>
      <feature policy='require' name='aefsi'/>
      <feature policy='require' name='msa5'/>
      <feature policy='require' name='msa4'/>
      ...
    </mode>
    <mode name='custom' supported='yes'>
      <model usable='yes'>z10EC-base</model>
      <model usable='yes'>z9EC-base</model>
      <model usable='yes'>z196.2-base</model>
      ...
    </mode>
  </cpu>
  ...
</domainCapabilities>
```

domiflist

Displays network interface information for a running virtual server.
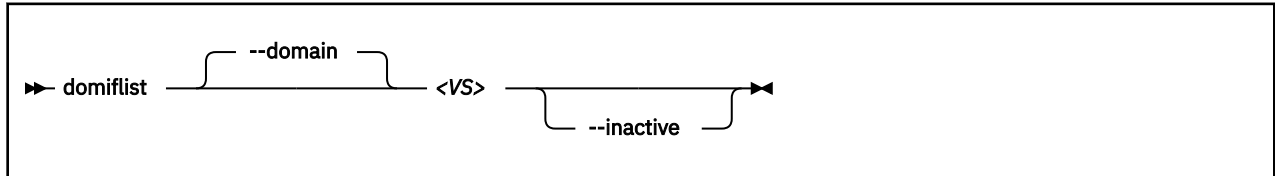
Syntax

```
►►─ domiflist ──┬──────────────┬── <VS> ──┬──────────────┬─►◄
                └── --domain ──┘          └── --inactive ─┘
```

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 156

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 ----┬──────────────┬--- <VS> --- <interface> ---><
                    └--- --domain --┘
```
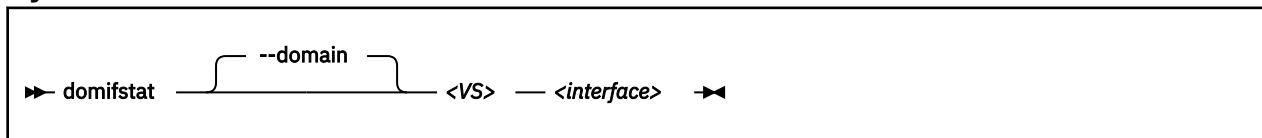
Where:

<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

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
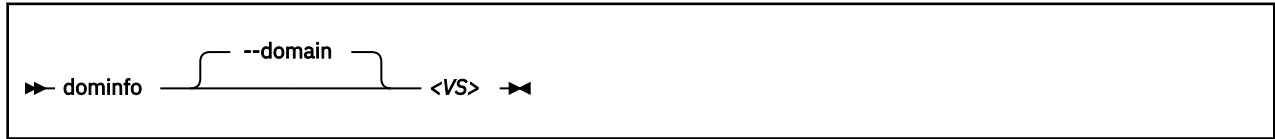
Where:

<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 156

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

```
►►─ domjobabort ──┬──────────┬── <VS> ─►◄
                  └─ --domain ─┘
```

Where:

<VS>

Is the name, ID or UUID of the virtual server.

Selected options

None.

Usage

"Live virtual server migration" on page 163

Example

This example aborts the currently running dump request for vserv1.

```
# virsh dump vserv1 vserv1.txt
error: Failed to core dump domain vserv1 to vserv1.txt
error: operation aborted: domain core dump job: canceled by client

# virsh domjobabort vserv1
```

domstate

Displays the state of a virtual server.

Syntax

```
►►— domstate —— <VS> ─────────────────►◄
                    └── --reason ──┘
```

Where:

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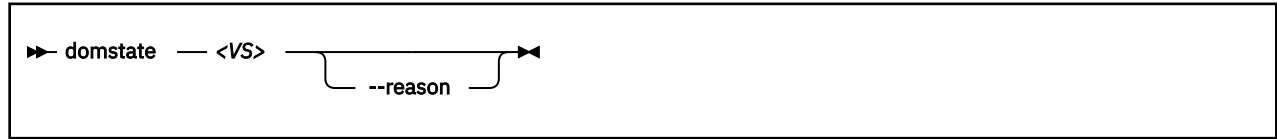
Selected options

--reason
Displays information about the reason why the virtual server entered the current state.

Usage

"Displaying information about a virtual server" on page 156

Example

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

dump

Creates a virtual server dump on the host.

Syntax

```
►►─ dump ─┬─────────────────┬─ <VS> ── <filename> ─►◄
          └─ --memory-only ─┘
```

Where:

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

<filename>
    Is the name of the target dump file.

Selected options

--memory-only
    Issues ELF dumps, which can be inspected by using the **crash** command.

Usage

"Creating a virtual server dump on the host" on page 219

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.
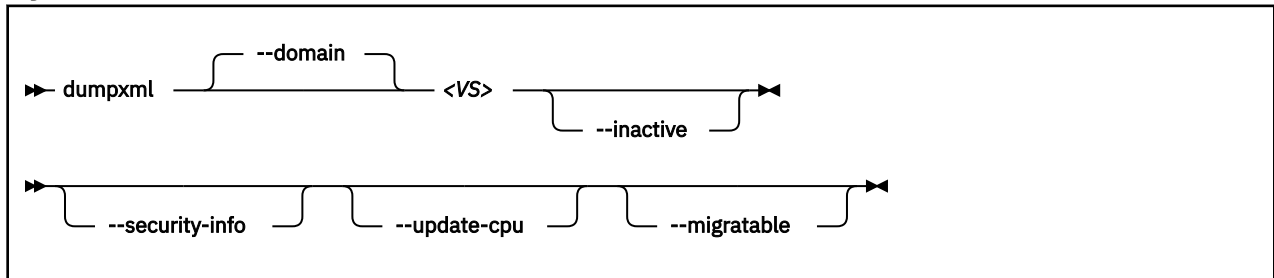
Selected options

--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 158

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

```
►►─ edit ─┬─────────────┬─ <VS> ─►◄
          └── --domain ──┘
```

Where:

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

Selected options

--domain
Specifies the virtual server.

Usage

Example

This example edits the libvirt-internal configuration of virtual server vserv1.

```
# virsh edit vserv1
```

hypervisor-cpu-baseline

Derives a baseline CPU model with features that are a subset of all CPU descriptions in a specified XML file.

Syntax
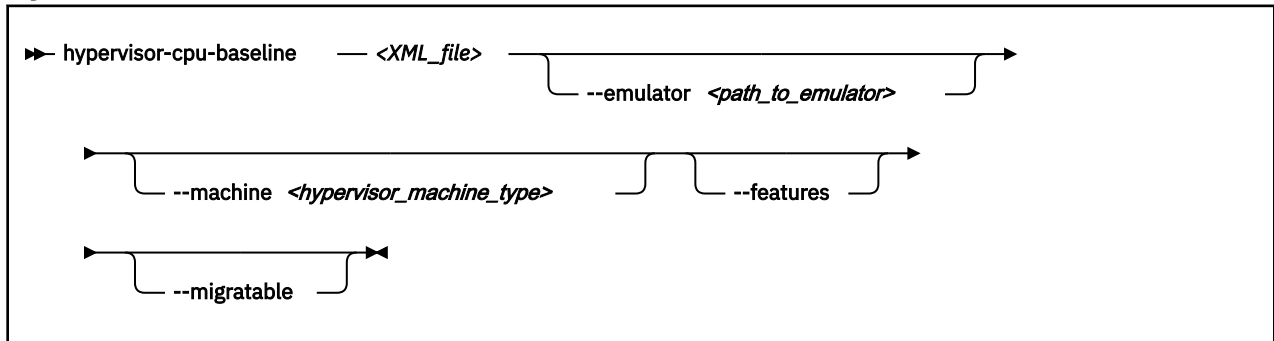
Where:

<XML_file>
Specifies a file that describes the CPU capabilities of one or more KVM hypervisors.

Descriptions must adhere to the syntax of the <cpu> element of a domain configuration-XML. To obtain a valid CPU description for a particular KVM hypervisor, issue the **domcapabilities** command on that hypervisor.

Concatenate the output of the **domcapabilities** command from all KVM hypervisors for which you want to find a baseline CPU model. Optionally, you can reduce the information for each hypervisor to the CPU snippet.

Selected options

<path_to_emulator>
Specifies the path to the emulator, for example, `/usr/bin/qemu-system-s390x`.

Omit this specification to make libvirt automatically use the correct path to the currently used emulator. Specify a value only if you are an expert user and need to experiment with a specific path.

<hypervisor_machine_type>
Specifies the hypervisor machine type, for example, `s390-ccw-virtio-5.0`.

Reduce the supported CPU features to an earlier hypervisor version by specifying the machine type of this version rather than the current hypervisor version. The current version is the default.

--features
Print a verbose description that explicitly specifies all features of the baseline CPU. Omitting this option prints features only if they divert from the CPU model.

--migratable
Omits features that might block migration. For example, a z14 baseline CPU might be reduced to z14-base to mitigate QEMU dependencies.

Usage

Example

```
# virsh hypervisor-cpu-baseline cpu.xml
<cpu>
    <model>z13-base</model>
    <feature policy='require' name='aen'/>
    <feature policy='require' name='aefsi'/>
    <feature policy='require' name='msa5'/>
    ...
    <feature policy='require' name='cmm'/>
</cpu>
```

hypervisor-cpu-compare

Compares a CPU description in an XML file with the CPU model that the hypervisor can provide on the host.

Syntax
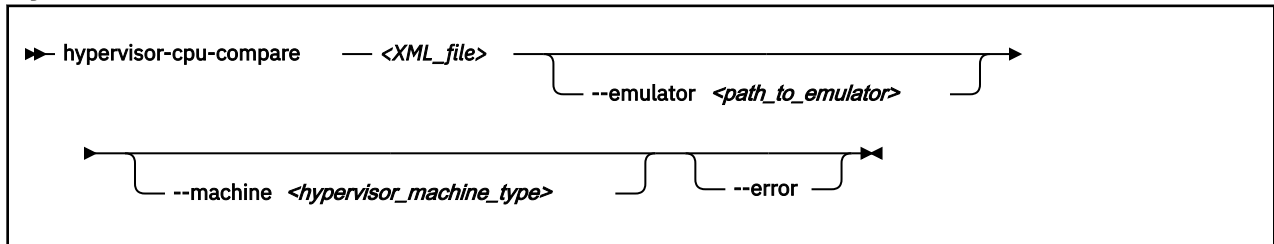
Where:

<XML_file>
Specifies an XML document that contains a CPU description. If the document contains multiple descriptions, only the first description is evaluated.

Descriptions must adhere to the syntax of the <cpu> element of a domain configuration-XML. For example, use the output of the **domcapabilities** command to obtain a valid CPU description for a particular virtual server.

Selected options

<path_to_emulator>
Specifies the path to the emulator, for example, `/usr/bin/qemu-system-s390x`.

Omit this specification to make libvirt automatically use the correct path to the currently used emulator. Specify a value only if you are an expert user and need to experiment with a specific path.

<hypervisor_machine_type>
Specifies the hypervisor machine type, for example, `s390-ccw-virtio-5.0`.

You can compare the XML document with an earlier hypervisor version by specifying the machine type of this version rather than the current hypervisor version. The current version is the default.

--error
Reports an error if the CPU description in the XML file is not compatible with the hypervisor. Also provides details about the incompatibility.

Command output

The output states how the CPU description of the XML document compares to the CPU that the hypervisor provides.

The host satisfies the CPU requirements as expressed in the XML description if it provides a matching CPU model or a superset of the required features.

The host falls short of the CPU requirements if the command output states that the XML description is incompatible with the CPU that the hypervisor provides. If the XML document represents the CPU definition of a virtual server, the virtual server cannot run on this host.

Usage

"Confirming the CPU model of a destination host" on page 165

Example

```
# virsh hypervisor-cpu-compare cpu.xml
...
CPU described in cpu.xml is identical to the CPU provided by hypervisor
on the host
```

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 219

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
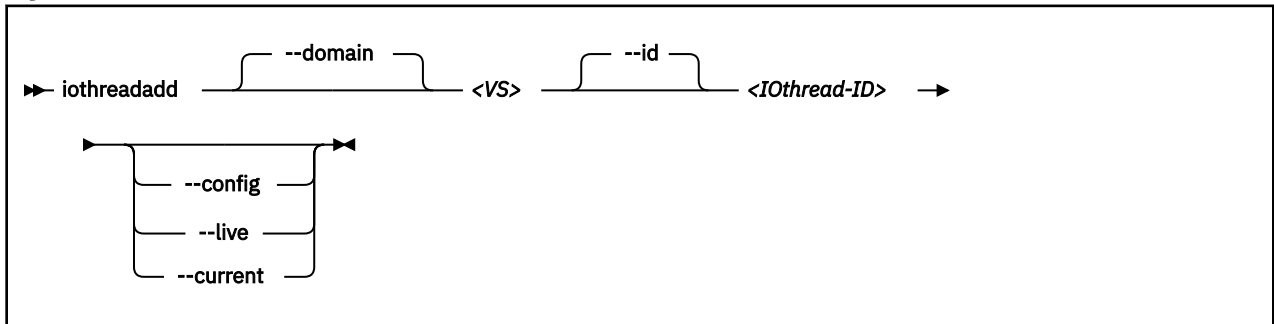
Where:

<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 182

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
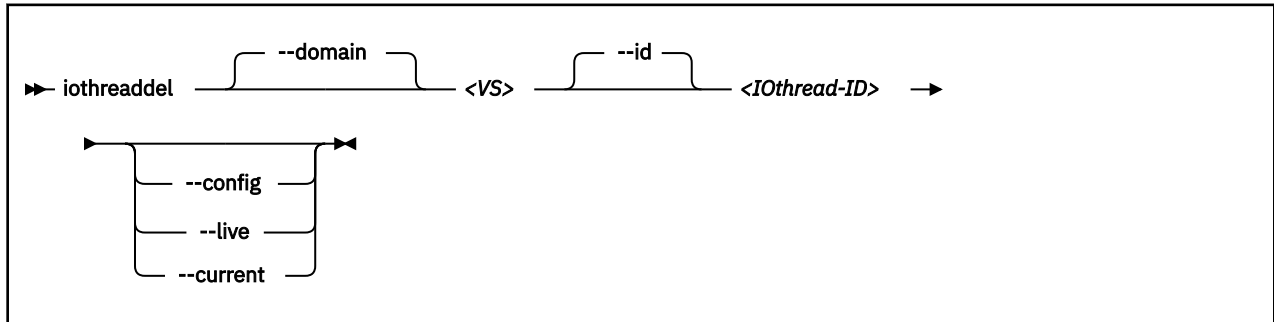
Where:

<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 183

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.
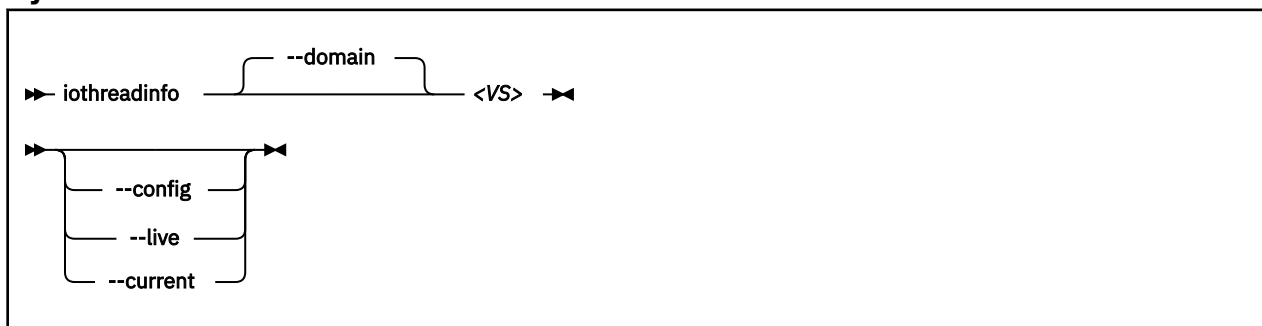
Syntax

```
▶▶─ iothreadinfo ──┬──────────────┬── <VS> ─▶◀
                   └── --domain ──┘

▶▶─┬──────────────┬─▶◀
   ├── --config ──┤
   ├── --live ────┤
   └── --current ─┘
```

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 156

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
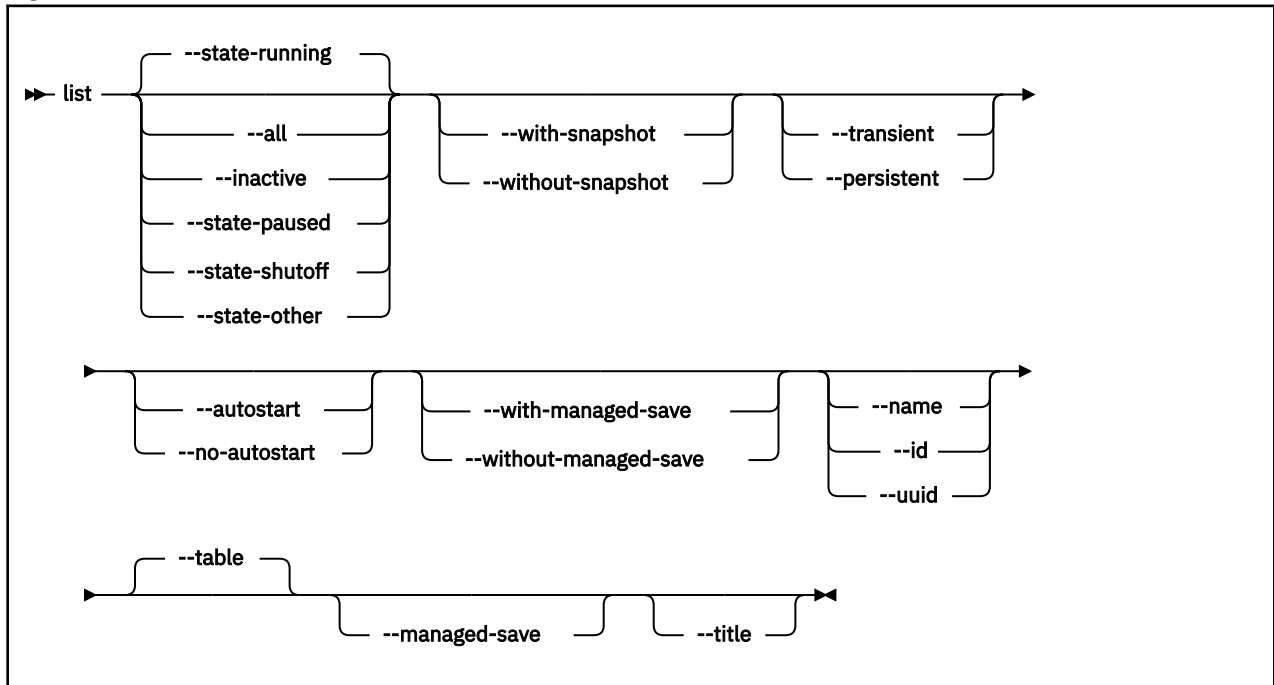
Selected options

--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

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

```
►►── managedsave ──┬──────────────────┬──┬─ --domain ─┬── <VS> ──┬───────────────┬──►
                   └── --bypass-cache ─┘  └────────────┘          ├── --running ──┤
                                                                  └── --paused ───┘

    ┌──────────────────────┐
►───┴──┬──────────────┬───►◄
       └── --verbose ──┘
```

Where:

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

Selected options

--bypass-cache
    Writes virtual server data directly to the disk bypassing the file system cache. This sacrifices write speed for data integrity by getting the data written to the disk faster.

--running
    When you restart the virtual server, it will be running.

--paused
    When you restart the virtual server, it will be paused.

--verbose
    Displays the progress of the save operation.

Virtual server state transitions

Command option	From state	To state (reason)
managedsave	running	shut off (saved *from running*)
managedsave	paused	shut off (saved *from paused*)
managedsave --running	running	shut off (saved *from running*)
managedsave --running	paused	shut off (saved *from running*)
managedsave --paused	running	shut off (saved *from paused*)
managedsave --paused	paused	shut off (saved *from paused*)

Usage

- "Terminating a virtual server" on page 152
- Chapter 35, "Virtual server life cycle," on page 229

Domain management

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

```
►►─ memtune ─┬─────────────┬─ <VS> ─┬──────────────────────────────┬─►◄
             └── --domain ──┘        └─ --soft-limit ── <limit-in-KB> ─┘
```
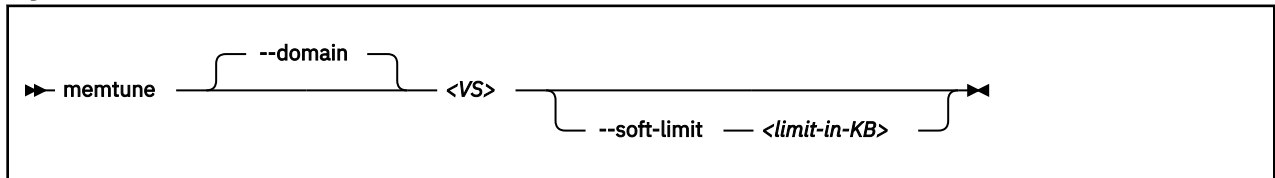
Where:

<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.

> ⚠️ **Warning:** Do not use the options `--hard-limit` and `--swap_hard_limit`. Their use might lead to a virtual server crash.

Usage

- Chapter 28, "Memory management," on page 203
- "Managing virtual memory" on page 180

Example

This example allows the host to limit the physical host memory usage of vserv1 memory to 256 MB in case the host is under memory pressure:

```
# virsh memtune vserv1 --soft-limit 256000
```

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

```
# virsh memtune vserv1
hard_limit     : unlimited
soft_limit     : 256000
swap_hard_limit: unlimited
```

migrate

Migrates a virtual server to a different host.

Syntax
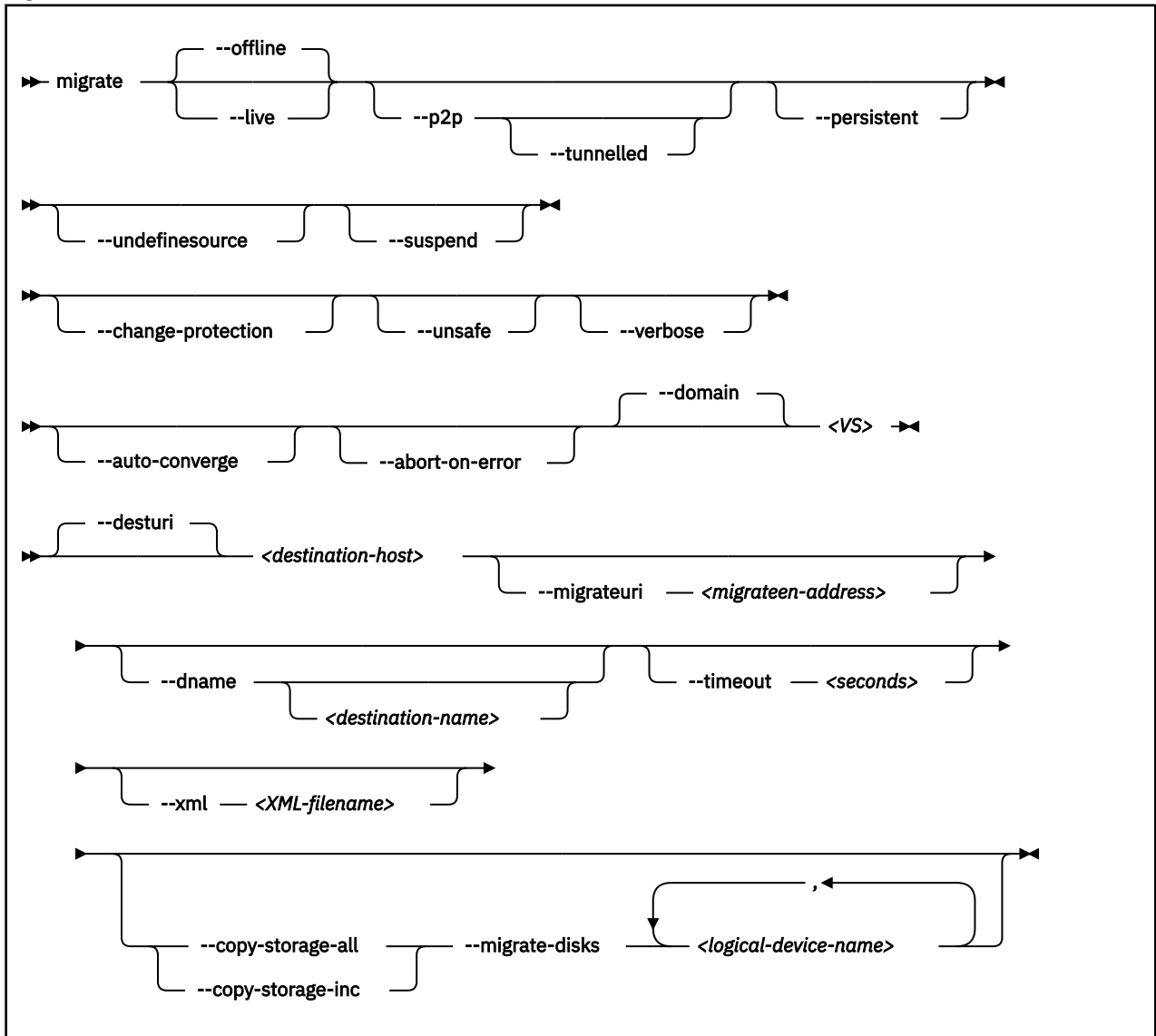
where

<destination-host>

The libvirt connection URI of the destination host.

   Normal migration:

   Specify the address of the destination host as seen from the virtual server.

   Peer to-peer migration:

   Specify the address of the destination host as seen from the source host.

<destination-name>

Is the new name of the virtual server on the destination host.

<logical-device-name>

The logical device name of the virtual block device.

<migrateen-address>

The host specific URI of the destination host.

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

<XML-filename>
The domain configuration-XML for the source virtual server.

Selected options

--abort-on-error
Causes an abort on soft errors during migration.

--auto-converge
Forces auto convergence during live migration.

--change-protection
Prevents any configuration changes to the virtual server until the migration ends

--copy-storage-all
Copies image files that back up virtual block devices to the destination. Make sure that an image file with the same path and filename exists on the destination host before you issue the virsh `migrate` command. The regarding virtual block devices are specified by the `--migrate-disks` option.

--copy-storage-inc
Incrementally copies non-readonly image files that back up virtual block devices to the destination. Make sure that an image file with the same path and filename exists on the destination host before you issue the virsh `migrate` command. The regarding virtual block devices are specified by the `--migrate-disks` option.

--dname
Specifies that the virtual server is renamed during migration (if supported).

--domain
Specifies the virtual server.

--live
Specifies the migration of a running or a paused virtual server.

--migrate-disks
Copies the files which back up the specified virtual block devices to the destination host. Use the `--copy-storage-all` or the `--copy-storage-inc` option in conjunction with this option. The regarding files must be writable. Please note that virtual DVDs are read-only disks. If in doubt, check your domain configuration-XML. If the disk device attribute of a disk element is configured as cdrom, or contains a readonly element, then the disk cannot be migrated.

--migrateuri
Specifies the host specific URI of the destination host.

If not specified, libvirt automatically processes the host specific URI from the libvirt connection URI. In some cases, it is useful to specify a destination network interface or port manually.

--offline
Specifies the migration of the virtual server in "shut off" state. A copy of the libvirt-internal configuration of the virtual server on the source host is defined on the destination host.

If you specify this option, specify the `--persistent` option, too.

--persistent
Specifies to persistent the virtual server on the destination system.

--p2p
Specifies peer-to-peer migration:

libvirt establishes a connection from the source to the destination host and controls the migration process. The migration continues even if virsh crashes or loses the connection.

Without the `--p2p` option, virsh handles the communication between the source and the destination host.

--suspend
>   Specifies that the virtual server will not be restarted on the destination system.

--timeout *seconds*
>   The number of seconds allowed before the virtual server is suspended while live migration continues.

--tunnelled
>   Specifies a tunneled migration:

>   libvirt pipes the migration data through the libvirtd communication socket. Thus, no extra ports are required to be opened on the destination host. This simplifies the networking setup required for migration.

>   The tunneled migration has a slight performance impact, because the data is copied between the libvirt daemons of the source host and the destination host.

>   Nevertheless, also in a tunneled migration, disk migration requires one extra destination port per disk.

--undefinesource
>   Specifies to undefine the virtual server on the source system.

--unsafe
>   Forces a migration even if it may cause data loss or corruption on the virtual server.

--verbose
>   Displays messages which indicate the migration progress.

Usage

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

```
►►── migrate-getspeed ──┬─────────────┬── <VS> ──►◄
                        └── --domain ──┘
```

Where:

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

Selected options
None.

Usage
"Live virtual server migration" on page 163

Example

```
# virsh migrate-getspeed vserv1
8796093022207
```

migrate-setmaxdowntime

Specifies a tolerable downtime for the virtual server during the migration, which is used to estimate the point in time when to suspend it.

Syntax

```
                                  --domain
►► migrate-setmaxdowntime  ┌──────────────┐  <VS>  ►◄
                           └──────────────┘


        --downtime
►► ┌──────────────┐  <milliseconds>  ►◄
   └──────────────┘
```
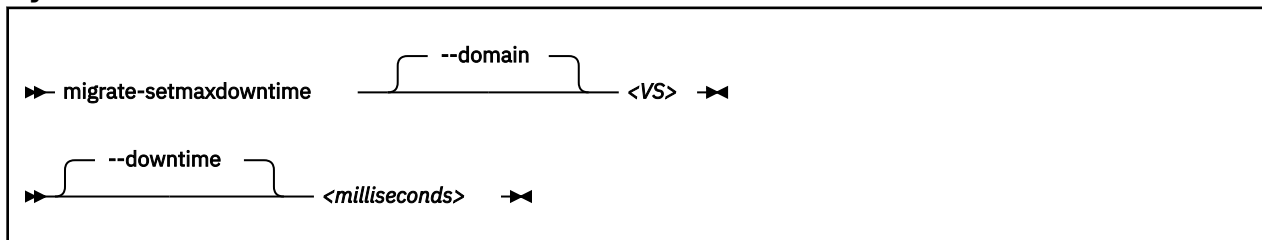
where

<milliseconds>
    Is the tolerable downtime of the virtual server during migration in milliseconds.

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

Selected options

None.

Usage

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 163

Example

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

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 289.

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 154

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

```
►►─ schedinfo ─── <VS> ─►◄

►►─┬──────────────────────┬─ cpu_shares ─── = ─── <number> ─┬─►◄
   ├─── --live ───┤
   └─── --config ─┘
```
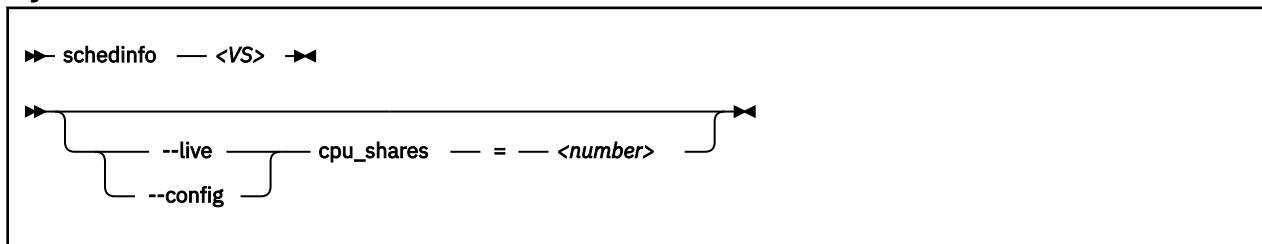
Where:

<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

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

```
►►─ shutdown ──┬─────────────┬── <VS> ─►◄
               └── --domain ──┘
```

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 152

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
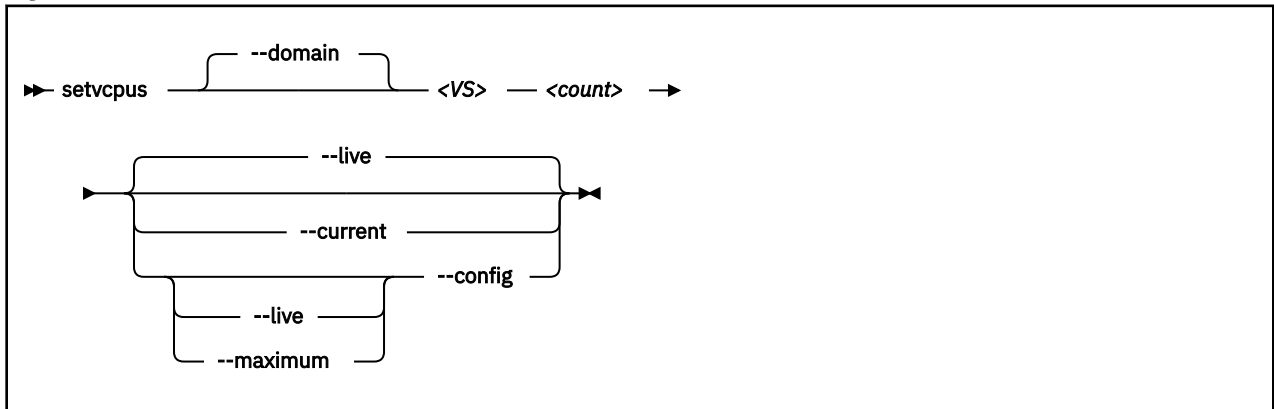
Where:

<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 176

Example

This example persistently adds a virtual CPU to the running virtual server vserv1:

```
# virsh vcpucount vserv1
maximum       config         5
maximum       live           5
current       config         3
current       live           3

# virsh setvcpus vserv1 4 --live --config

# virsh vcpucount vserv1
maximum       config         5
maximum       live           5
current       config         4
current       live           4
```

start

Starts a defined virtual server that is shut off or crashed.

Syntax

Where:

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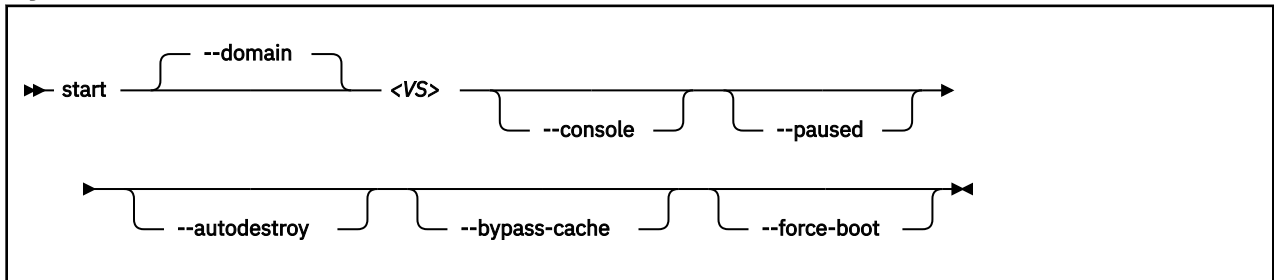
Selected options

--autodestroy
    Destroys the virtual server when virsh disconnects from libvirt.

--bypass-cache
    Does not load the virtual server from the cache.

--console
    Connects to a configured pty console.

--domain
    Specifies the virtual server.

--force-boot
    Any saved system image is discarded before booting.

--paused
    Suspends the virtual server as soon as it is started.

Virtual server state transitions

Command option	From state (reason)	To state (reason)
start	shut off (unknown)	running (booted)
start	shut off (saved *from running*)	running (restored)
start	shut off (saved *from paused*)	paused (migrating)
start	shut off (shutdown)	running (booted)
start	shut off (destroyed)	running (booted)
start	crashed	running (booted)
start --force-boot	shut off (unknown)	running (booted)
start --force-boot	shut off (saved *from running*)	running (booted)
start --force-boot	shut off (saved *from paused*)	paused (user)
start --force-boot	shut off (shutdown)	running (booted)
start --force-boot	shut off (destroyed)	running (booted)

Command option	From state (reason)	To state (reason)
start --paused	shut off (unknown)	paused (user)
start --paused	shut off (saved *from running*)	paused (migrating)
start --paused	shut off (saved *from paused*)	paused (migrating)
start --paused	shut off (shutdown)	paused (user)
start --paused	shut off (destroyed)	paused (user)

Usage

Example

This example starts virtual server vserv1 with initial console access.

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

suspend

Transfers a virtual server from the running to the paused state.

Syntax

```
►►─ suspend ──── <VS> ─►◄
```

Where:

<VS>

Is the name, ID, or UUID of the virtual server.

Selected options

None.

Virtual server state transition

From State	To State (reason)
running	paused (user)

Usage

"Suspending a virtual server" on page 154

Example

This example suspends virtual server vserv1.

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

# virsh suspend vserv1
Domain vserv1 suspended

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

undefine

Deletes a virtual server from libvirt.

Purpose

Syntax

►►── undefine ──── *<VS>* ──►◄

Where:

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

Selected options

None.

Usage

- Chapter 1, "Overview," on page 3
- "Undefining a virtual server" on page 149

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.
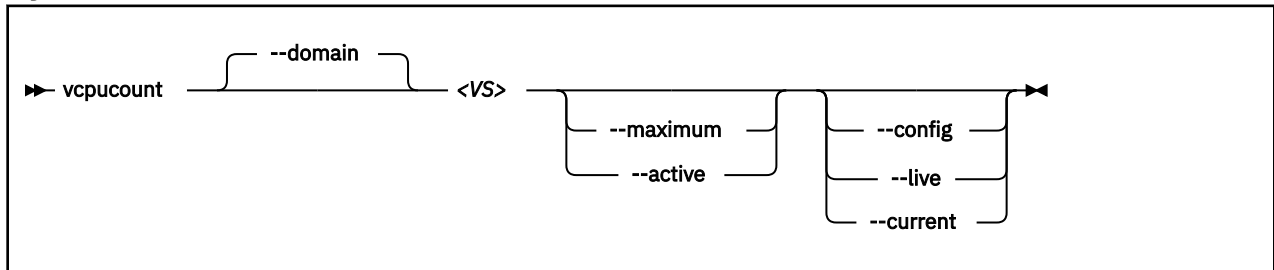
Selected options

--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

Example

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        3
current      live          3
```

Network management virsh commands

Use the network management virsh commands to manage virtual networks that connect KVM virtual servers among themselves and to external networks. For related XML elements, see "Network configuration-XML" on page 307.

- "net-autostart" on page 396
- "net-define" on page 397
- "net-destroy" on page 398
- "net-dumpxml" on page 399
- "net-edit" on page 400
- "net-info" on page 401
- "net-list" on page 402
- "net-name" on page 403
- "net-start" on page 404
- "net-undefine" on page 405
- "net-uuid" on page 406

Other virsh commands

- "Domain management virsh commands" on page 343
- "Node-device management virsh commands" on page 407
- "Storage pool management virsh commands" on page 416
- "Volume management virsh commands" on page 430

net-autostart

Enables or disables the automatic start of a virtual network when the libvirt daemon is started.
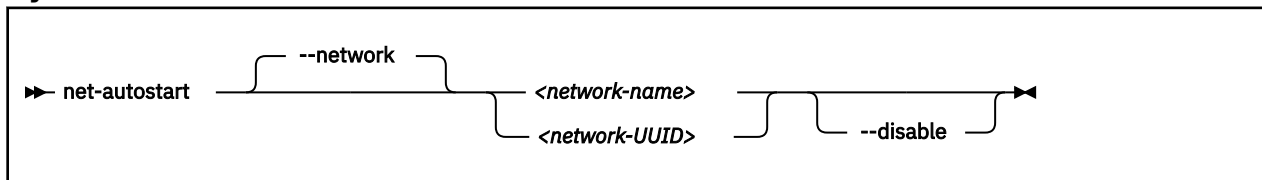
Syntax

```
►►— net-autostart ─┬─    --network    ─┬─ <network-name> ─┬────────────┬─►◄
                   │                    │                  │            │
                   └────────────────────┤ <network-UUID> ├┴─ --disable ┘
```

Where:

<network-name>
    Is the name of the virtual network.

<network-UUID>
    Is the UUID of the virtual network.

Selected options

--network
    Specifies the virtual network.

--disable
    Disables the automatic start of the virtual network when the libvirt daemon is started.

Usage

Chapter 24, "Managing virtual networks ," on page 191

Example

This example configures the automatic start of virtual network net0 when the libvirt daemon is started.

```
# virsh net-autostart net0
```

net-define

Creates a persistent definition of a virtual network.

Syntax

```
►►─ net-define ──┬─── --file ───┬── <XML-filename> ──►◄
                 └──────────────┘
```

Where:

<XML-filename>
    Is the name of the network configuration-XML file.

Selected options

--file
    specifies the network configuration-XML file.

Usage

Example

This example defines the virtual network that is configured by the `net0.xml` network configuration-XML file.

```
# virsh net-define net0.xml
```

net-destroy

Deactivates an active virtual network.
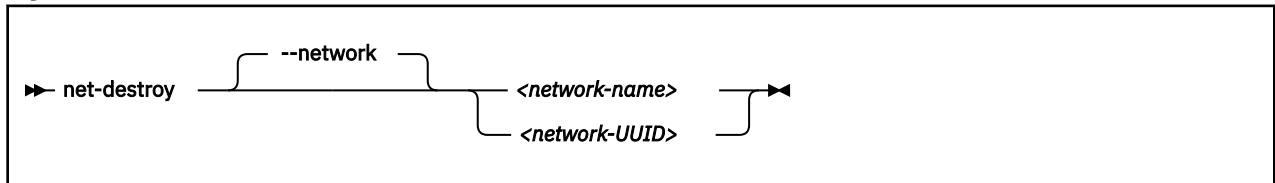
Syntax

```
►►─ net-destroy ─┬────────────────┬─┬─ <network-name> ─┬─►◄
                 └─── --network ───┘ └─ <network-UUID> ─┘
```

Where:

<network-name>
Is the name of the virtual network.

<network-UUID>
Is the UUID of the virtual network.

Selected options

--network
Specifies the virtual network.

Usage

Chapter 24, "Managing virtual networks ," on page 191

Example

This example shuts down the virtual network with name net0.

```
# virsh net-destroy net0
```

net-dumpxml

Displays the current configuration of a virtual network.
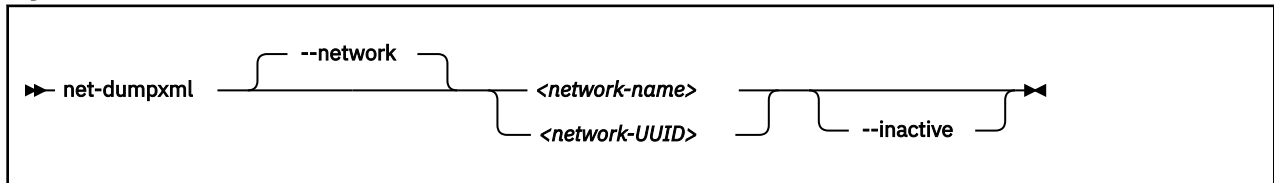
Syntax

```
►►─ net-dumpxml ─┬─────── --network ───────┬─── <network-name> ──┬──────────────────┬─►◄
                 │                          │                     │                  │
                 └──────────────────────────┤   <network-UUID>   └──── --inactive ───┘
```

Where:

<network-name>
　　Is the name of the virtual network.

<network-UUID>
　　Is the UUID of the virtual network.

Selected options

--network
　　Specifies the virtual network.

--inactive
　　Displays the network XML without the automatic expansions in the libvirt-internal representation.

Usage

Example

```
# virsh net-dumpxml net0
<network>
  <name>net0</name>
  <uuid>fec14861-35f0-4fd8-852b-5b70fdc112e3</uuid>
  <forward mode="nat">
    <nat>
      <port start="1024" end="65535"/>
    </nat>
  </forward>
  <bridge name="virbr0" stp="on" delay="0"/>
  <mac address="aa:25:9e:d9:55:13"/>
  <ip address="192.0.2.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.0.2.2" end="192.0.2.254"/>
    </dhcp>
  </ip>
</network>
```

net-edit

Edits the configuration of a defined virtual network. When the update is saved, both the libvirt-internal configuration and the Network configuration-XML are updated.

Syntax
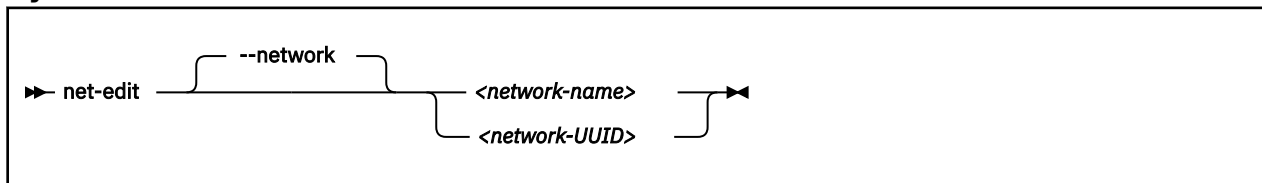
Where:

<network-name>
Is the name of the virtual network.

<network-UUID>
Is the UUID of the virtual network.

Selected options

--network
Specifies the virtual network.

Usage

Example

This example edits the configuration of net0.

```
# virsh net-edit net0
```

net-info

Displays information about a defined virtual network.
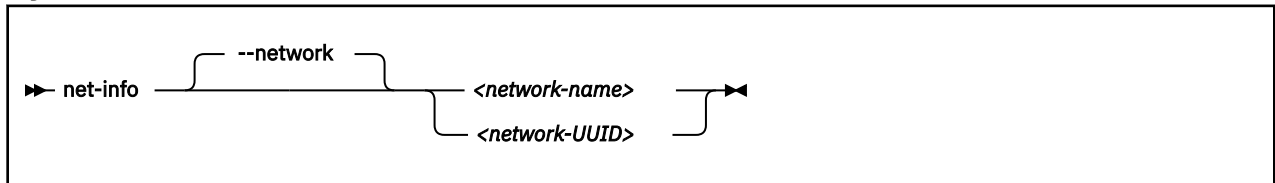
Syntax

```
►► net-info ──┬── --network ──┬──┬── <network-name> ──┬── ►◄
              └───────────────┘  └── <network-UUID> ──┘
```

Where:

<network-name>
    Is the name of the virtual network.

<network-UUID>
    Is the UUID of the virtual network.

Selected options

--network
    Specifies the virtual network.

Usage

Chapter 24, "Managing virtual networks ," on page 191

Example

```
# virsh net-info net0
Name:           net0
UUID:           fec14861-35f0-4fd8-852b-5b70fdc112e3
Active:         yes
Persistent:     yes
Autostart:      yes
Bridge:         virbr0
```

net-list

Displays a list of defined virtual networks.

By default, a list of active virtual networks is displayed.

Syntax

```
►►─ net-list ─┬──────────────┬─┬─────────────────┬─┬─ --table ─┬─►◄
              ├─ --inactive ─┤ ├─ --autostart ───┤ ├─ --name ──┤
              └─ --all ──────┘ └─ --no-autostart ┘ └─ --uuid ──┘
```
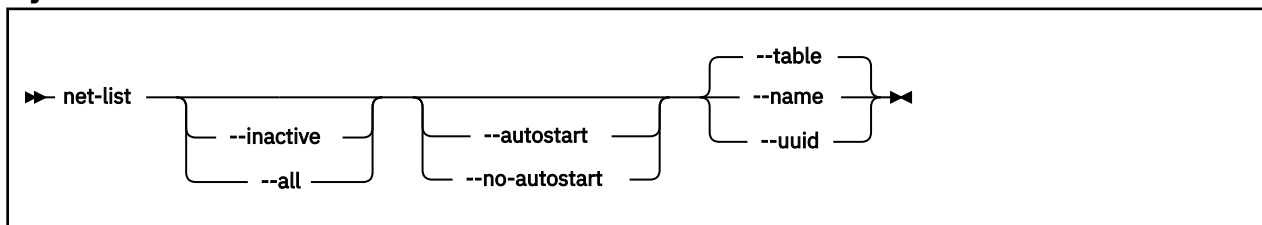
Selected options

--all
  Displays active and inactive virtual networks.

--autostart
  Displays only virtual networks that start automatically when the libvirt daemon is started.

--inactive
  Displays only inactive virtual networks.

--name
  Lists the network names instead of displaying a table of virtual networks.

--no-autostart
  Displays only virtual networks that do not start automatically when the libvirt daemon is started.

--table
  Displays the virtual network information in table format.

--uuid
  Lists the virtual network UUIDs instead of displaying a table of virtual networks.

Usage
Chapter 24, "Managing virtual networks ," on page 191

Example

```
# virsh net-list
 Name                 State     Autostart    Persistent
---------------------------------------------------------
 default              active    yes          yes
 net0                 active    no           yes
```

net-name

Displays the name of a virtual network that is specified with its UUID.

Syntax
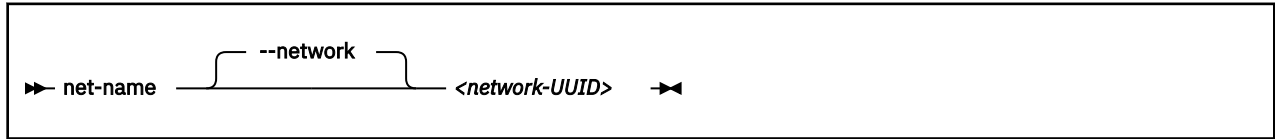
```
►► net-name ──┌──--network──┐── <network-UUID> ►◄
```

Where:

<network-UUID>
 Is the UUID of the virtual network.

Selected options

--network
 Specifies the virtual network.

Usage

Example

```
# virsh net-name fec14861-35f0-4fd8-852b-5b70fdc112e3
net0
```

net-start

Activates a defined, inactive virtual network.
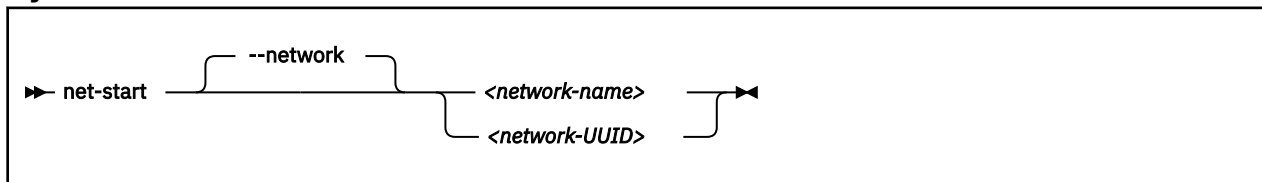
Syntax

```
►►─ net-start ──┬─────────────────┬── <network-name> ──┬──►◄
                │    --network     │                    │
                └─────────────────┴── <network-UUID> ──┘
```

Where:

<network-name>
Is the name of the virtual network.

<network-UUID>
Is the UUID of the virtual network.

Selected options

--network
Specifies the virtual network.

Usage

Example

This example starts the virtual network with the name net0.

```
# virsh net-start net0
```

net-undefine

Deletes the persistent libvirt definition of a virtual network.
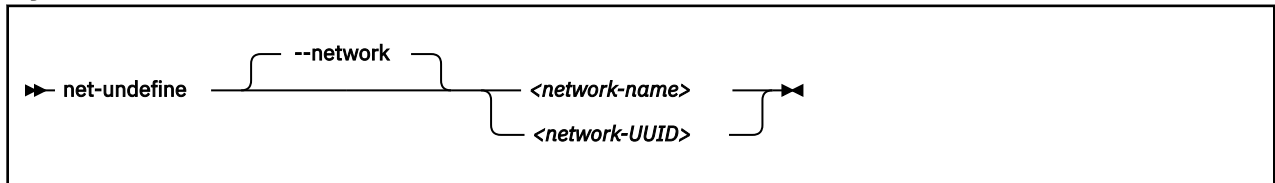
Syntax

```
►►── net-undefine ──┬──── --network ────┬──── <network-name> ──┬──►◄
                    │                   │                      │
                    └───────────────────┴──── <network-UUID> ──┘
```

Where:

<network-name>
  Is the name of the virtual network.

<network-UUID>
  Is the UUID of the virtual network.

Selected options

--network
  Specifies the virtual network.

Usage

Chapter 24, "Managing virtual networks ," on page 191

Example

This example removes the virtual network with name net0 from the libvirt definition.

```
# virsh net-undefine net0
```

net-uuid

Displays the UUID of a virtual network that is specified with its name.

Syntax
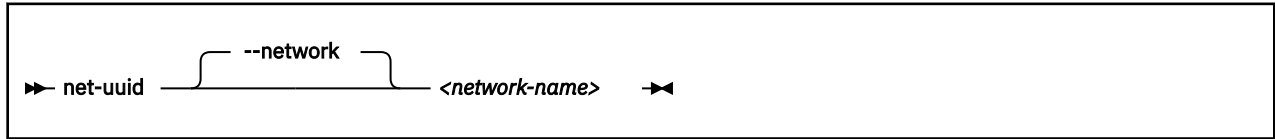
```
►►── net-uuid ──┬──────────────┬── <network-name> ──►◄
                └── --network ──┘
```

Where:

<network-name>
Is the name of the virtual network.

Selected options

--network
Specifies the virtual network.

Usage

Example

```
# virsh net-uuid net0
fec14861-35f0-4fd8-852b-5b70fdc112e3
```

Node-device management virsh commands

Use the node-device management virsh commands to manage VFIO mediated devices and other resources on the KVM host. For related XML elements, see "Node-device XML" on page 315.

Other virsh commands

nodedev-create

Creates active transient VFIO mediated devices from node-device XML files.

Syntax

```
►►─ nodedev-create ──── <mdev_host_device_xml_file> ─────►◄
```

Selected options

<mdev_host_device_xml_file>
A specification, in node-device XML format, for the mediated device to be created.

Usage

- "Preparing DASD pass-through devices" on page 56
- "Creating a mediated device with AP queues" on page 59
- "Managing mediated devices with libvirt" on page 62

Example

This example uses a node-device XML file, mdev_vfiocss72.xml, to create a mediated device mdev_96c03c6a_a109_44fe_816a_ba371542164b.

```
# cat mdev_vfiocss72.xml
<device>
    <parent>css_0_0_0072</parent>
    <capability type="mdev">
        <type id="vfio_ccw-io"/>
        <uuid>96c03c6a-a109-44fe-816a-ba371542164b</uuid>
    </capability>
</device>
# virsh nodedev-create mdev_vfiocss72.xml
Node device mdev_96c03c6a_a109_44fe_816a_ba371542164b created from mdev_vfiocss72.xml
```

nodedev-define

Defines persistent VFIO mediated devices to libvirt by using specifications from node-device XML files.

Syntax

```
►►─ nodedev-define ──── <mdev_host_device_xml_file>      ►◄
```

Selected options

<mdev_host_device_xml_file>
    A specification, in node-device XML format, for the mediated device to be defined.

Usage

- "Preparing DASD pass-through devices" on page 56
- "Creating a mediated device with AP queues" on page 59
- "Managing mediated devices with libvirt" on page 62

Example

This example uses a node-device XML file, `my_dasd_mdev.xml`, to define a persistent mediated device `mdev_18e124fb_b2fc_47f6_a407_f256b6c49767` to libvirt.

```
# virsh nodedev-define my_dasd_mdev.xml
Node device 'mdev_18e124fb_b2fc_47f6_a407_f256b6c49767' defined from my_dasd_mdev.xml
```

nodedev-destroy

Deactivates a VFIO mediated device. Transient mediated devices cease to exist. Persistent mediated devices can be activated again with a **nodedev-start** command.

Syntax

```
►►─ nodedev-destroy ──── <mdev_nodedev_name> ───►◄
```

Selected options

<mdev_nodedev_name>
Specifies a mediated device as listed with the **virsh nodedev-list --cap mdev** command.

Usage

Example

This example destroys a mediated device mdev_96c03c6a_a109_44fe_816a_ba371542164b.

```
# virsh nodedev-destroy mdev_96c03c6a_a109_44fe_816a_ba371542164b
Destroyed node device 'mdev_96c03c6a_a109_44fe_816a_ba371542164b'
```

nodedev-dumpxml

Displays the properties, in node-device XML format, of a host resource that is represented in sysfs on the KVM host and that can be detected by libvirt.

Syntax

```
►►─ nodedev-dumpxml ──── <nodedev_name> ──►◄
```

Selected options

<nodedev_name>

Specifies a host resource as listed with the **virsh nodedev-list** command.

Usage

- "Managing mediated devices for DASD with libvirt" on page 64

Examples

This example displays the properties of a mediated device
`mdev_96c03c6a_a109_44fe_816a_ba371542164b`.

```
$ virsh nodedev-dumpxml mdev_96c03c6a_a109_44fe_816a_ba371542164b
<device>
  <name>mdev_96c03c6a_a109_44fe_816a_ba371542164b</name>
  <path>/sys/devices/css0/0.0.0072/96c03c6a-a109-44fe-816a-ba371542164b</path>
  <parent>css_0_0_0072</parent>
  <driver>
    <name>vfio_mdev</name>
  </driver>
  <capability type="mdev">
    <type id="vfio_ccw-io"/>
    <uuid>96c03c6a-a109-44fe-816a-ba371542164b</uuid>
    <iommuGroup number="1"/>
  </capability>
</device>
```

This example displays the properties of subchannel 0.0.0072.

```
# virsh nodedev-dumpxml css_0_0_0072
<device>
  <name>css_0_0_0072</name>
  <path>/sys/devices/css0/0.0.0072</path>
  <parent>computer</parent>
  <driver>
    <name>vfio_ccw</name>
  </driver>
  <capability type="css">
    <cssid>0x0</cssid>
    <ssid>0x0</ssid>
    <devno>0x0072</devno>
    <capability type="mdev_types">
      <type id="vfio_ccw-io">
        <name>I/O subchannel (Non-QDIO)</name>
        <deviceAPI>vfio-ccw</deviceAPI>
        <availableInstances>1</availableInstances>
      </type>
    </capability>
  </capability>
</device>
```

nodedev-list

Lists resources that are represented in sysfs on the KVM host and that can be detected by libvirt.

Syntax
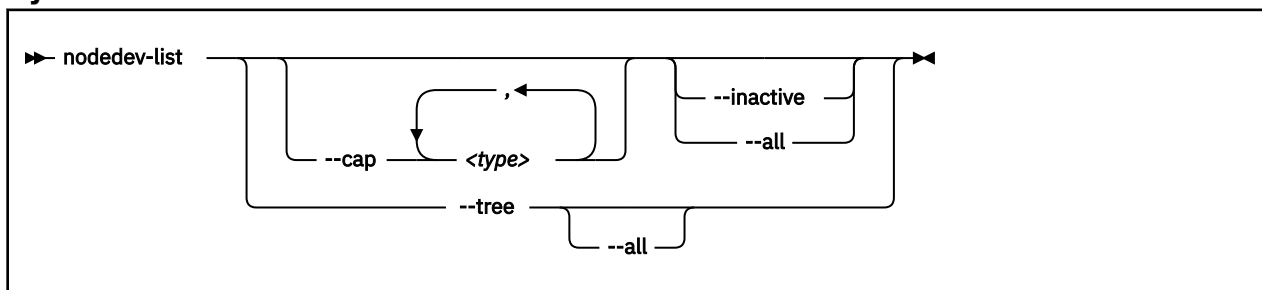
Selected options

--cap *<type>*
Restricts the command output to one or more capability types. Capability types group resources that can be used for similar purposes in guest configurations. The following specifications are examples of valid types:

mdev
Lists the available mediated devices.

mdev_types
Lists potential parent devices for mediated devices.

ap_card
Lists the available cryptographic adapters.

ap_queue
Lists the available domains on cryptographic adapters.

ap_matrix
Lists VFIO mediated devices that represent AP configurations. Such mediated devices can be assigned to virtual servers as pass-through devices. Only one such AP configuration can be assigned to a specific virtual server.

storage
Lists the available storage devices, for example, DASD or SCSI block devices.

--inactive
Lists inactive devices. By default, only active devices are listed.

--all
Lists active and inactive devices. By default, only active devices are listed.

--tree
Shows the hierarchical relationship of the host resources in a tree view.

Usage

• "Managing mediated devices with libvirt" on page 62

Example

This example shows a tree view of all host resources that are detected by libvirt.

```
$ virsh nodedev-list --tree
computer
  |
  +- ap_card08
  |
  | +- ap_08_0001
  | +- ap_08_0002
  |
  +- ap_card09
  |   |
  |   +- ap_09_0001
  |   +- ap_09_0002
  |
  +- ap_matrix
  |   |
  |   +- mdev_45b9ba40_b8aa_4b84_9cdd_bdeea3a6a365
  |   +- mdev_4e7855ca_933a_489d_a3a4_d3aec6e0de69
  ...
  +- css_0_0_006b
  |   |
  |   +- ccw_0_0_1000
  |       |
  |       +- block_dasdg_IBM_750000000DHVL1_0001_00
  |
  +- css_0_0_0072
  |   |
  |   +- mdev_96c03c6a_a109_44fe_816a_ba371542164b
  ...
```

This example lists all cryptographic adapters and mediated devices.

```
$ virsh nodedev-list --cap ap_card,mdev
ap_card08
ap_card09
mdev_45b9ba40_b8aa_4b84_9cdd_bdeea3a6a365
mdev_4e7855ca_933a_489d_a3a4_d3aec6e0de69
...
mdev_96c03c6a_a109_44fe_816a_ba371542164b
...
```

nodedev-start

Activates an inactive persistent VFIO mediated device.

Syntax

```
▶▶─ nodedev-start ──── <mdev_nodedev_name>   ──▶◀
```

Selected options

<mdev_nodedev_name>
An inactive persistent VFIO mediated device as listed by the **virsh nodedev-list --cap mdev --inactive** command.

Usage

- "Preparing DASD pass-through devices" on page 56
- "Creating a mediated device with AP queues" on page 59
- "Managing mediated devices with libvirt" on page 62

Example

This example starts a mediated device mdev_18e124fb_b2fc_47f6_a407_f256b6c49767.

```
# virsh nodedev-start mdev_18e124fb_b2fc_47f6_a407_f256b6c49767
Device mdev_18e124fb_b2fc_47f6_a407_f256b6c49767 started
```

nodedev-undefine

Removes the definition of a persistent VFIO mediated device from libvirt.

Syntax

```
▶▶─ nodedev-undefine  ─── <mdev_nodedev_name>     ─▶◀
```

Selected options

<mdev_nodedev_name>
    A persistent VFIO mediated device as listed by the **`virsh nodedev-list --cap mdev`** command.

Usage

- "Preparing DASD pass-through devices" on page 56
- "Creating a mediated device with AP queues" on page 59
- "Managing mediated devices with libvirt" on page 62

Example

This example removes the definition of a mediated device
mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec from libvirt.

```
# virsh nodedev-undefine mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec
Undefined node device 'mdev_7b36c4c2_b280_4ea7_8f40_77b192bf6fec'
```

Storage pool management virsh commands

Use the storage pool management virsh commands to manage storage pools, which consist of a set of similar volumes. Manage the volumes of a storage pool with the volume management virsh commands. For related XML elements, see "Storage pool configuration-XML" on page 328.

Other virsh commands

pool-autostart

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

Syntax

Where:

<pool-name>
    Is the name of the storage pool.

<pool-UUID>
    Is the UUID of the storage pool.

Selected options

--pool
    Specifies the storage pool.

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

Usage
Chapter 23, "Managing storage pools ," on page 187

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

Where:

<XML-filename>
    Is the name of the storage pool configuration-XML file.

Selected options

--file
    Specifies the storage pool configuration-XML file.

Usage

Example

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

```
# virsh pool-define pool1.xml
```

pool-delete

Deletes the volumes of a storage pool.

Syntax

⚠️ **Attention:** This command is intended for expert users. Depending on the pool type, the results range from no effect to loss of data. In particular, data is lost when a zfs or LVM group pool is deleted.

```
            --pool
►►─ pool-delete ──┬──────────┬──┬─ <pool-name> ─┬──►◄
                               └─ <pool-UUID> ─┘
```

Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.
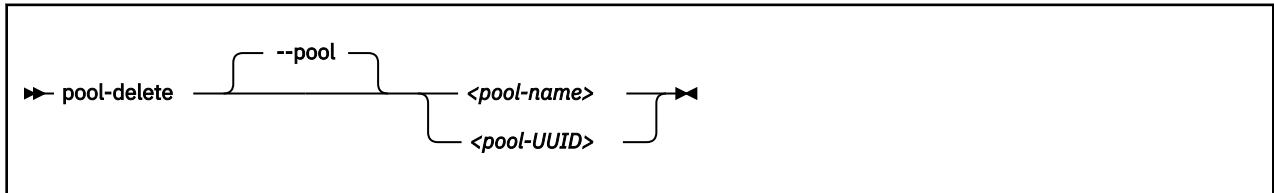
Selected options

--pool
Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools ," on page 187

Example

This example deletes the volumes of storage pool `pool1`.

```
# virsh pool-delete pool1
```

pool-destroy

Shut down a storage pool.

The pool can be restarted by using the virsh **pool-start** command.

Syntax
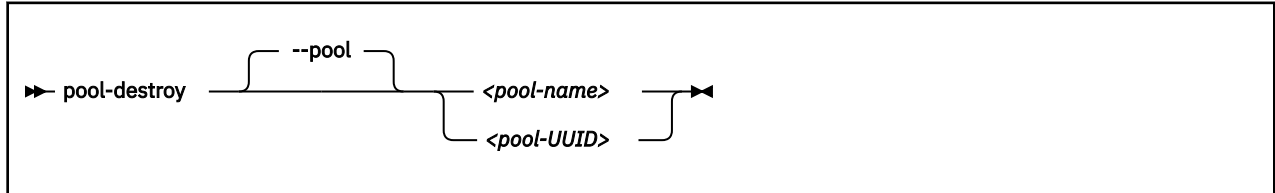
Where:

<pool-name>
    Is the name of the storage pool.

<pool-UUID>
    Is the UUID of the storage pool.

Selected options

--pool
    Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools ," on page 187

Example

This example shuts down storage pool pool1.

```
# virsh pool-destroy pool1
```

pool-dumpxml

Displays the current libvirt-internal configuration of a storage pool.
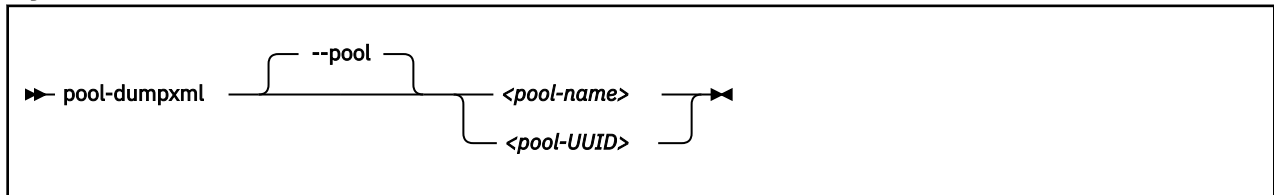
Syntax

Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool
Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools," on page 187

Example

```
# virsh pool-dumpxml pool1 default
<pool type="dir">
  <name>default</name>
  <uuid>09382b31-03ac-6726-45be-dfcaaf7b01cc</uuid>
  <capacity unit="bytes">243524067328</capacity>
  <allocation unit="bytes">109275693056</allocation>
  <available unit="bytes">134248374272</available>
  <source>
  </source>
  <target>
    <path>/var/lib/libvirt/images</path>
    <permissions>
      <mode>0711</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
  </target>
</pool>
```

pool-edit

Edits the libvirt-internal configuration of a defined storage pool.
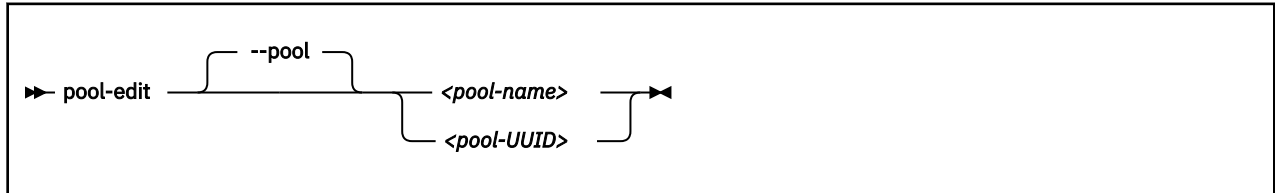
Syntax

Where:

<pool-name>
  Is the name of the storage pool.

<pool-UUID>
  Is the UUID of the storage pool.

Selected options

--pool
  Specifies the storage pool.

Usage

Example

This example edits the libvirt-internal configuration of pool1.xml.

```
# virsh pool-edit pool1
```

pool-info

Displays information about a defined storage pool.
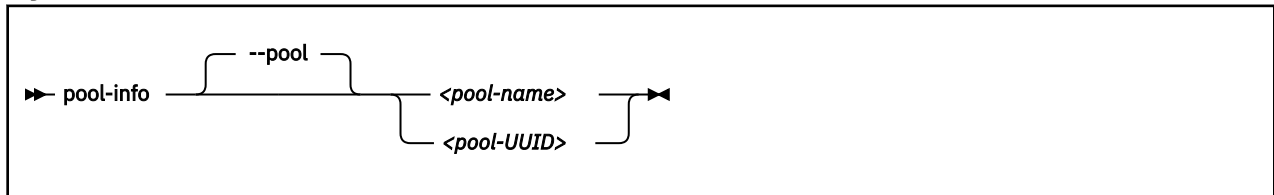
Syntax

Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool
Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools ," on page 187

Example

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

pool-list

Displays a list of defined storage pools.

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

Syntax
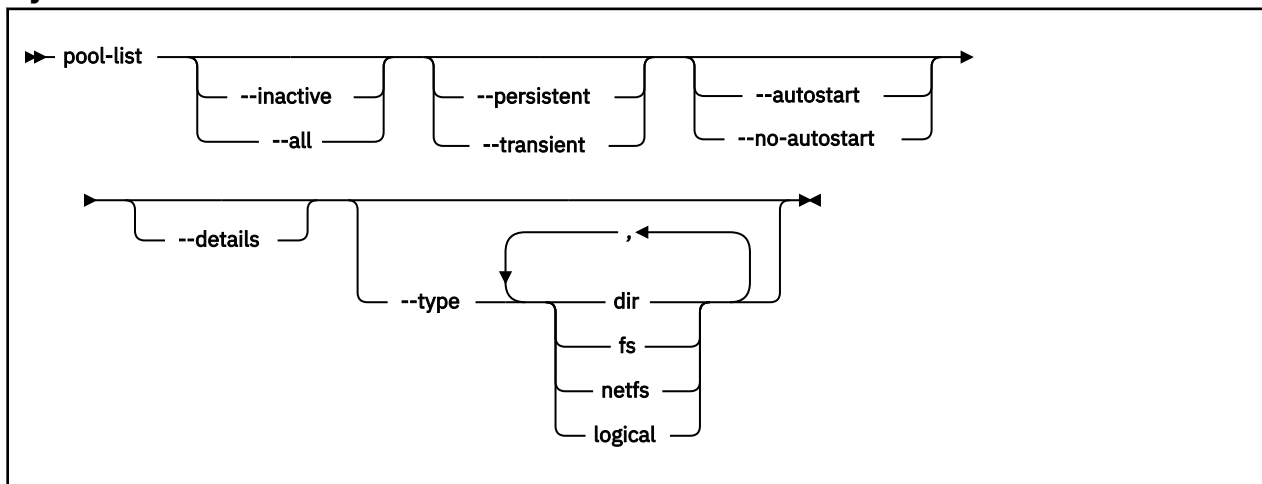
Selected options

--all
Displays all defined storage pools.

--autostart
Displays all storage pools that start automatically when the libvirt daemon is started.

--details
Displays pool persistence and capacity related information.

--inactive
Displays all inactive storage pools.

--no-autostart
Displays all storage pools that do not start automatically when the libvirt daemon is started.

--persistent
Displays all persistent storage pools.

--transient
Displays all transient storage pools.

--type
Displays all storage pools of the specified types.

Usage

Chapter 23, "Managing storage pools ," on page 187

Example

```
# virsh pool-list
```

pool-name

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

Syntax

Where:

<pool-UUID>
    Is the UUID of the storage pool.

Selected options

--pool
    Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools ," on page 187

Example

```
# virsh pool-name bc403958-a355-4d3c-9d5d-872c16b205ca
```

pool-refresh

Updates the volume list of a storage pool.

Syntax
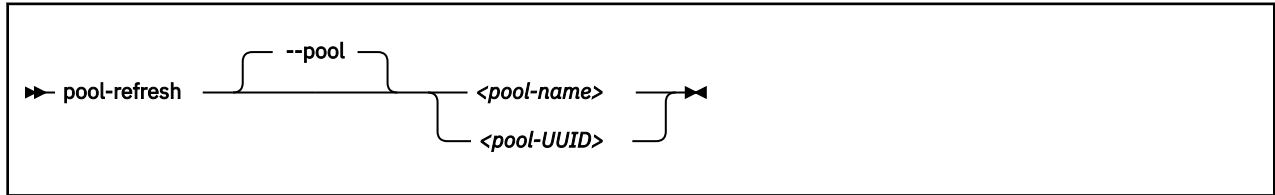
Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool
Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools ," on page 187

Example

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

```
# virsh pool-refresh pool1
```

pool-start

Starts a defined inactive storage pool.

Syntax
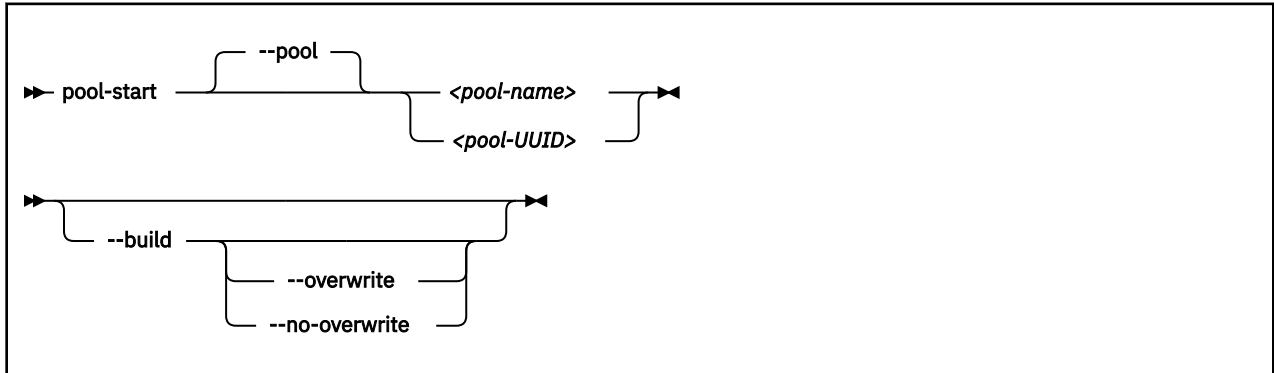
Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool
Specifies the storage pool.

--build
Creates the directory or the file system or creates the label for a disk (depending on the storage pool type) before starting the storage pool.

--overwrite
Only valid for storage pools of type dir, fs, or netfs.

Creates the directory, the file system or the label, overwriting existing ones.

--no-overwrite
Only valid for storage pools of type dir, fs, or netfs.

Creates the directory or the file system only if it does not exist. Returns an error if it does exist.

Usage
Chapter 23, "Managing storage pools," on page 187

Example

This example starts storage pool `pool1`.

```
# virsh pool-start pool1
```

pool-undefine

Deletes the persistent libvirt definition of a storage pool.
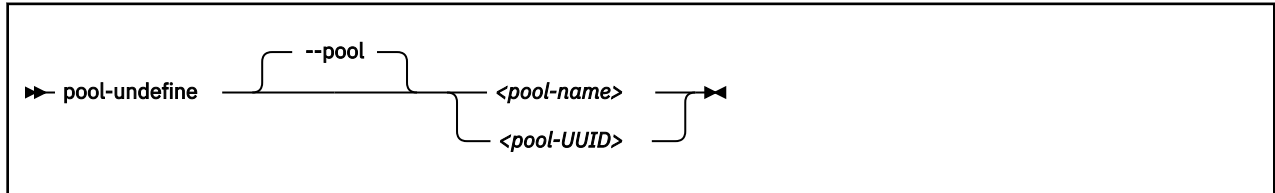
Syntax

Where:

<pool-name>
Is the name of the storage pool.

<pool-UUID>
Is the UUID of the storage pool.

Selected options

--pool
Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools ," on page 187

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
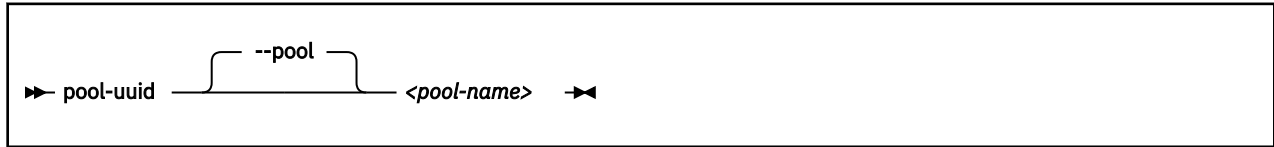
```
►►─ pool-uuid ──┬──── --pool ────┬── <pool-name> ─►◄
```

Where:

<pool-name>
    Is the name of the storage pool.

Selected options

--pool
    Specifies the storage pool.

Usage

Chapter 23, "Managing storage pools ," on page 187

Example

```
# virsh pool-uuid pool1
```

Volume management virsh commands

Use the volume management virsh commands to manage storage pool volumes. For related XML elements, see "Volume configuration-XML" on page 335.

Other virsh commands

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

Example

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

vol-delete

Remove a volume from a storage pool.

Syntax
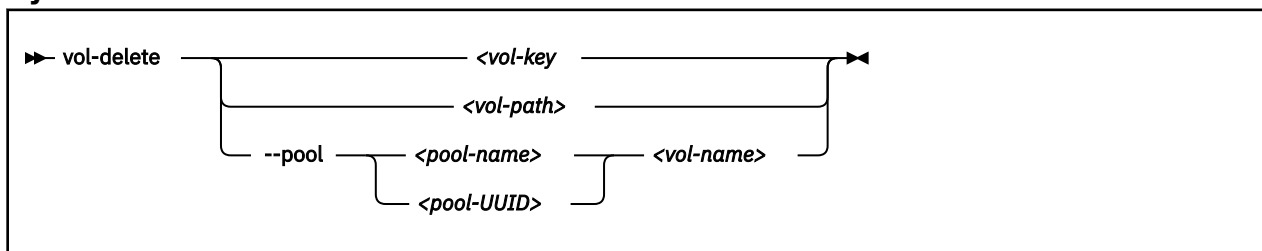
```
►►── vol-delete ─┬────────────── <vol-key ──────────┬─►◄
                 │              <vol-path>           │
                 └── --pool ─┬─ <pool-name> ─┬─ <vol-name> ─┘
                            └─ <pool-UUID> ─┘
```

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

Example

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

vol-dumpxml

Displays the current libvirt-internal configuration of a storage volume.

Syntax

```
►►─ vol-dumpxml ──┬──────────────┬──────────┬── <vol-key ─────────────┬──►◄
                  │              │          └── <vol-path> ───────────┤
                  └─ --pool ──┬─ <pool-name> ─┬── <vol-name> ─────────┘
                             └─ <pool-UUID> ─┘
```

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 189

Example

```
# virsh vol-dumpxml --pool default federico.img
<volume type="file">
  <name>federico.img</name>
  <key>/var/lib/libvirt/images/federico.img</key>
  <source>
  </source>
  <capacity unit="bytes">12582912000</capacity>
  <allocation unit="bytes">2370707456</allocation>
  <target>
    <path>/var/lib/libvirt/images/federico.img</path>
    <format type="qcow2"/>
    <permissions>
      <mode>0600</mode>
      <owner>0</owner>
      <group>0</group>
    </permissions>
    <timestamps>
      <atime>1481535271.342162944</atime>
      <mtime>1481292068.444109102</mtime>
      <ctime>1481292068.916109091</ctime>
    </timestamps>
  </target>
</volume>
```

vol-info

Displays information about a defined volume.

Syntax
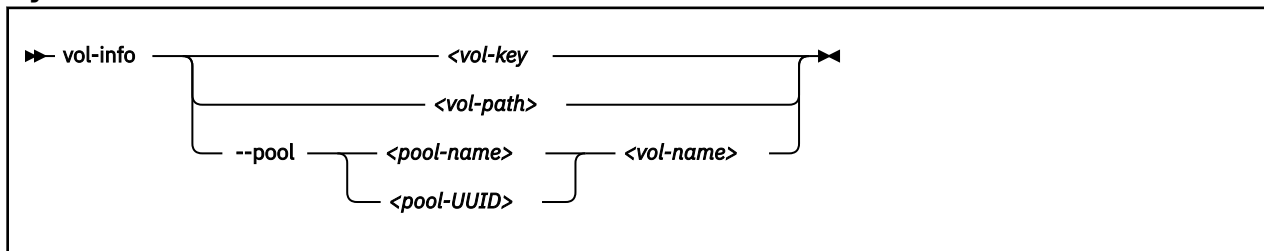
```
►►─ vol-info ─┬─────────────── <vol-key ───────────────┬─►◄
              │                 <vol-path>             │
              └─ --pool ─┬─ <pool-name> ─┬─ <vol-name> ─┘
                         └─ <pool-UUID> ─┘
```

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 189

Example

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

vol-key

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

Syntax
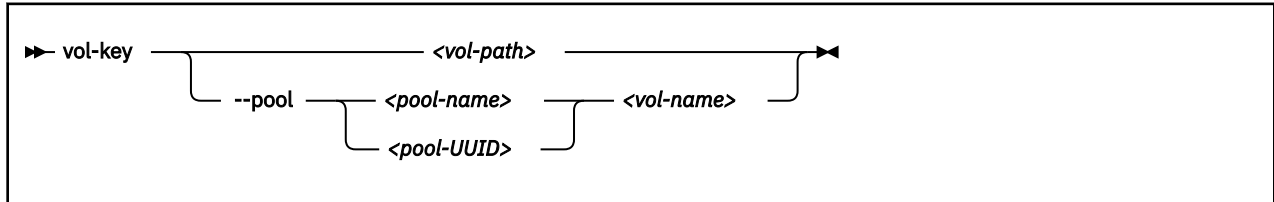
Where:

<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 189

Example

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

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

vol-list

Displays a list of defined storage pools.

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

Syntax

```
►►── vol-list ── --pool ──┬── <pool-name> ──┬──┬──────────────┬──►◄
                          └── <pool-UUID> ──┘  └── --details ──┘
```

Where:

<pool-name>
>   Is the name of the storage pool.

<pool-UUID>
>   Is the UUID of the storage pool.
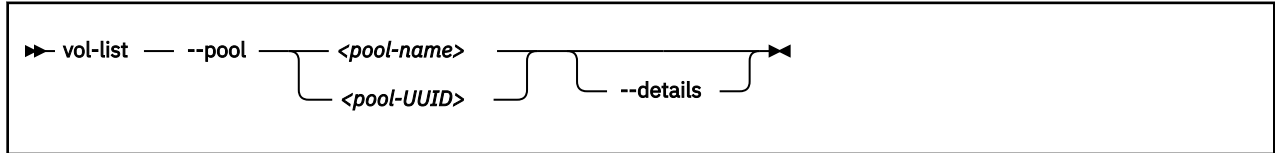
Selected options

--details
>   Displays volume type and capacity related information.

--pool
>   Specifies the storage pool.

Usage

"Volume management commands" on page 189

Example

```
# virsh vol-list --pool pool1
```

vol-name

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

Syntax
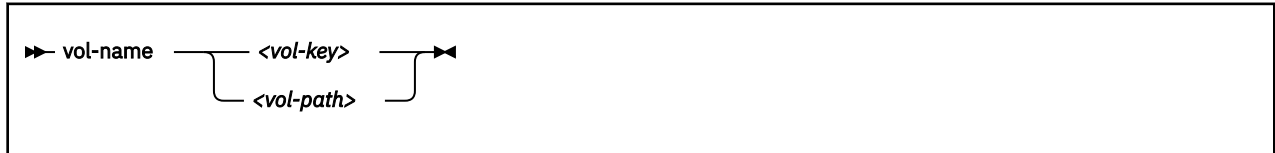
```
►►─ vol-name ─┬─ <vol-key> ──┬─►◄
              └─ <vol-path> ─┘
```

Where:

<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 189

Example

This example displays the volume name of a volume with a path `/var/lib/libvirt/images/federico.img`.

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

vol-path

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

Syntax
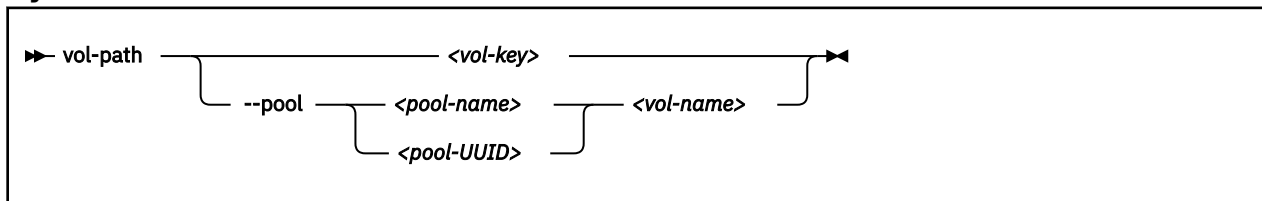
Where:

<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

Example

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

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

vol-pool

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

By default, the strorage pool name is displayed.

Syntax

```
►►— vol-pool ──┬──────────┬──┬── <vol-key> ──┬──►◄
               └── --uuid ──┘  └── <vol-path> ──┘
```

Where:

<vol-key>
    Is the key of the volume.

<vol-path>
    Is the path of the volume.
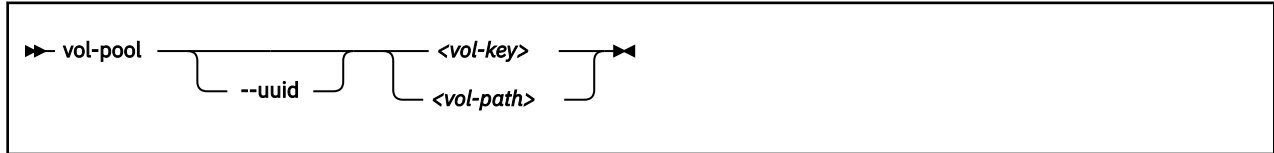
Selected options

--uuid
    Returns the storage pool UUID.

Usage

"Volume management commands" on page 189

Example

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

Chapter 38. 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.

QEMU image command

You can use the qemu-img command to manage disk images.

- This example creates a qcow2 image with a maximum size of 10 GB:

```
# 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 10 GB:

```
# 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 39. Hypervisor information for the virtual server user

The virtual server user can use the emulated Store Hypervisor Information (STHYI) instruction to retrieve information about the IBM Z hardware and the LPAR on which the KVM host runs.

The information includes:

- The CPU count, by type (CP or IFL)
- Limitations for shared CPUs
- CEC and LPAR identifiers

KVM guests use the `qclib` and the GCC inline assembly to run the emulated instruction. For an example, see `arch/s390/kvm/sthyi.c` in the Linux source tree.

The emulated STHYI instruction provides information through a response buffer with three data sections:

- The header section, at the beginning of the response buffer, which identifies the locations and length of the sections that follow.
- The machine section.
- The partition section.

Header section

Length	Data Type	Offset (dec)	Name	Contents
1	Bitstring	0	INFHFLG1	Header Flag Byte 1 These flag settings indicate the environment that the instruction was executed in and may influence the value of the validity bits. The validity bits, and not these flags, should be used to determine if a field is valid. **0x80** Global Performance Data unavailable. **0x40** One or more hypervisor levels below this level does not support the STHYI instruction. When this flag is set the value of INFGPDU is not meaningful because the state of the Global Performance Data setting cannot be determined. **0x20** Virtualization stack is incomplete. This bit indicates one of two cases: • One or more hypervisor levels does not support the STHYI instruction. For this case, INFSTHYI will also be set. • There were more than three levels of guest/hypervisor information to report. **0x10** Execution environment is not within a logical partition.
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

Length	Data Type	Offset (dec)	Name	Contents
2	Unsigned Binary Integer	18	INFPLEN	Length of Partition Section
2	Unsigned Binary Integer	20	INFHOFF1	Offset to Hypervisor Section1 mapped by INF0HYP
2	Unsigned Binary Integer	22	INFHLEN1	Length of Hypervisor Section1
2	Unsigned Binary Integer	24	INFGOFF1	Offset to Guest Section1 mapped by INF0GST
2	Unsigned Binary Integer	26	INFGLEN1	Length of Guest Section1
2	Unsigned Binary Integer	28	INFHOFF2	Offset to Hypervisor Section2 mapped by INF0HYP
2	Unsigned Binary Integer	30	INFHLEN2	Length of Hypervisor Section2
2	Unsigned Binary Integer	32	INFGOFF2	Offset to Guest Section2 mapped by INF0GST
2	Unsigned Binary Integer	34	INFGLEN2	Length of Guest Section2
2	Unsigned Binary Integer	36	INFHOFF3	Offset to Hypervisor Section3 mapped by INF0HYP
2	Unsigned Binary Integer	38	INFHLEN3	Length of Hypervisor Section3
2	Unsigned Binary Integer	40	INFGOFF3	Offset to Guest Section3 mapped by INF0GST
2	Unsigned Binary Integer	42	INFGLEN3	Length of Guest Section3
4		44		Reserved for future IBM use

Format machine section

Length	Data Type	Offset (dec)	Name	Contents
1	Bitstring	0	INFMFLG1	Machine Flag Byte 1 reserved for IBM use
1	Bitstring	1	INFMFLG2	Machine Flag Byte 2 reserved for IBM use

Lengt h	Data Type	Offset (dec)	Name	Contents

1	Bitstring	2	INFMVAL1	Machine Validity Byte 1
				
				0x80
				    Processor Count Validity. When this bit is on, it indicates that INFMSCPS, INFMDCPS, INFMSIFL, and INFMDIFL contain valid counts. The validity bit may be off when:
				
				• STHYI support is not available on a lower level hypervisor, or
				
				• Global Performance Data is not enabled.
				
				0x40
				    Machine ID Validity. This bit being on indicates that a SYSIB 1.1.1 was obtained from STSI and information reported in the following fields is valid: INFMTYPE, INFMMANU, INFMSEQ, and INFMPMAN.
				
				0x20
				    Machine Name Validity. This bit being on indicates that the INFMNAME field is valid.

1	Bitstring	3	INFMVAL2	Machine Validity Byte 2 reserved for IBM use
2	Unsigned Binary Integer	4	INFMSCPS	Number of shared CPs configured in the machine or in the physical partition if the system is physically partitioned
2	Unsigned Binary Integer	6	INFMDCPS	Number of dedicated CPs configured in this machine or in the physical partition if the system is physically partitioned
2	Unsigned Binary Integer	8	INFMSIFL	Number of shared IFLs configured in this machine or in the physical partition if the system is physically partitioned.
2	Unsigned Binary Integer	10	INFMDIFL	Number of dedicated IFLs configured in this machine or in the physical partition if the system is physically partitioned.
8	EBCDIC	12	INFMNAME	Machine Name
4	EBCDIC	20	INFMTYPE	Type
16	EBCDIC	24	INFMMANU	Manufacturer
16	EBCDIC	40	INFMSEQ	Sequence Code
4	EBCDIC	56	INFMPMAN	Plant of Manufacture
4		60		Reserved for future IBM use

Format partition section

Length	Data Type	Offset (dec)	Name	Contents
1	Bitstring	0	INFPFLG1	Partition Flag Byte 1 **0x80** Multithreading (MT) is enabled.
1	Bitstring	1	INFPFLG2	Partition Flag Byte 2 reserved for IBM use

1	Bitstring	2	INFPVAL1	Partition Validity Byte 1
				
				0x80
				    This bit being on indicates that INFPSCPS, INFPDCPS, INFPSIFL, and INFPDIFL contain valid counts.
				
				0x40
				    This bit being on indicates that INFPWBCP and INFPWBIF are valid
				
				0x20
				    This bit being on indicates that INFPABCP and INFPABIF are valid.
				
				0x10
				    This bit being on indicates that a SYSIB 2.2.2 was obtained from STSI and information reported in the following fields is valid: INFPPNUM and INFPPNAM.
				
				0x08
				    This bit being on indicates that INFPLGNM, INFPLGCP, and INFPLGIF are valid.

1	Bitstring	3	INFPVAL2	Partition Validity Byte 2 reserved for IBM use
2	Unsigned Binary Integer	4	INFPPNUM	Logical partition number
2	Unsigned Binary Integer	6	INFPSCPS	Number of shared logical CPs configured for this partition. Count of cores when MT is enabled.
2	Unsigned Binary Integer	8	INFPDCPS	Number of dedicated logical CPs configured for this partition. Count of cores when MT is enabled.
2	Unsigned Binary Integer	10	INFPSIFL	Number of shared logical IFLs configured for this partition. Count of cores when MT is enabled.
2	Unsigned Binary Integer	12	INFPDIFL	Number of dedicated logical IFLs configured for this partition. Count of cores when MT is enabled.
2		14		Reserved for future IBM use
8	EBCIDIC	16	INFPPNAM	Logical partition name
4	Unsigned Binary Integer	24	INFPWBCP	Partition weight-based capped capacity for CPs, a scaled number where X'00010000' represents one core. Zero if not capped.

Length	Data Type	Offset (dec)	Name	Contents
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	INFPABIF	Partition absolute capped capacity for IFLs, a scaled number where X'00010000' represents one core. Zero if not capped.
8	EBCIDIC	40	INFPLGNM	LPAR group name. Binary zeros when the partition is not in an LPAR group. EBCDIC and padded with blanks on the right when in a group. The group name is reported only when there is a group cap on CP or IFL CPU types and the partition has the capped CPU type.
4	Unsigned Binary Integer	48	INFPLGCP	LPAR group absolute capacity value for CP CPU type when nonzero. This field will be nonzero only when INFPLGNM is nonzero and a cap is defined for the LPAR group for the CP CPU type. When nonzero, contains a scaled number where X'00010000' represents one core.
4	Unsigned Binary Integer	52	INFPLGIF	LPAR group absolute capacity value for IFL CPU type when nonzero. This field will be nonzero only when INFPLGNM is nonzero and a cap is defined for the LPAR group for the IFL CPU type. When nonzero, contains a scaled number where X'00010000' represents one core.

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 and LinuxONE publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication 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.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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.

Index

auto value *(continued)*
  of the controller model attribute 251
  of the geometry trans attribute 265

B

backend element
  model attribute 246
balloon device 99
base device 31
block device
  attaching 182
  detaching 183
  displaying 156
  hotplugging 182
  unplugging 183
block device driver 4
block value
  of the disk type attribute 103, 255
bonded interface
  activating 46
  configuring 124
  preparing 46
  setting up 46
bonded network interface 46
bonding
  mode 46
  parameter 46
boot element
  loadparm attribute 247
  order attribute 77, 247
boot menu selection 247
boot process
  configuring 77
bootable kernel 77, 80
booted reason
  of the running state 231
booting
  from a DASD 77
  from a kernel image file 80
  from a SCSI disk 77
  from a VFIO DASD 78
  from an ISO image 79
  from temporary device 195
  over the network 81
bounce buffer 141
bridge attribute
  of the source element 126
bridge element
  name attribute 308
bridge port
  active 49
  primary 49
bridge port role
  enabling 49
bridge value
  of the forward mode attribute 310
  of the interface type attribute 126, 272
  of the source mode attribute 124
bridge_state sysfs attribute 49
browsing
  virtual servers 156
bus attribute
  of the address element 117, 122, 240, 244

bus attribute *(continued)*
  of the target element 103, 109, 111, 122, 301
by-ID device node 11, 17
by-path device node 11, 17, 31, 41, 103
by-uuid device node 11, 17

C

cache attribute
  of the driver element 103, 109, 111, 122, 259
canceling
  a live migration 170
capability element 319
CCW device 102
CCW group device 44, 49
ccw value
  of the address type attribute 103, 109, 111, 115, 239
  of the capability type attribute 319
CD-ROM 102
cdrom value
  of the disk device attribute 122, 255
Central Processor Assist for Cryptographic Functions 97
cgroups 200
change-media command
  --eject option 184
  --insert option 184
change-media virsh command 184, 347
channel bonding 46
channel bonding module 46
channel path 11, 17, 102
channel path type 102
channel subsystem 11, 17, 102
channel subsystem-ID 11, 17, 102, 103, 109, 111
chccwdev command 31, 33, 37
chcpu command 176
cipher element
  name attribute
    aes value 97, 248
    dea value 97, 248
  state attribute
    off value 97, 248
    on value 97, 248
clear key 97
cmdline element 75, 80, 249
collecting
  performance metrics 223
command
  chccwdev 33, 37
  chcpu 176
  crash 219
  fdisk 33
  ip 43, 44
  ls 103
  lscss 37, 103
  lsqeth 44, 49
  lsscsi 33
  lszfcp 33, 37
  modprobe 46
  multipath 33, 103
  ovs-vsctl add-bond 49
  ovs-vsctl add-br 49
  ovs-vsctl add-port 49
  ovs-vsctl del-br 49
  ovs-vsctl show 49

libvirt XML element *(continued)*

address 103, 109, 111, 115, 117, 122, 133, 239, 240, 244
ap-adapter 316
ap-domain 317
attr 318
backend 246
boot 77, 247
bridge 308
capability 319
cipher 97, 248
cmdline 75, 80, 249
console 95, 250
controller 115, 133, 251
cpu 87, 252
cputune 75, 86, 253
device as root 322
devices 75, 254
dhcp 309
disk 103, 109, 111, 122, 133, 255
domain 256
driver 103, 109, 111, 122, 133, 258, 259
driver, child of hostdev 261
driver, child of interface 262
emulator 93, 263
feature 87, 264
format 336
forward 310
geometry 265
graphics 266
host 329
hostdev 117, 133, 267
hugepages 91, 269
initrd 75, 80, 270
input 271
interface 124, 126, 133, 272
iothreads 75, 94, 103, 273
ip 311
kernel 75, 80, 274
key 337
keywrap 97, 275
launchSecurity 276
listen 277
log 95, 278
mac 124, 126, 133, 279
memballoon 99, 280
memory 75, 89, 92, 281
memoryBacking 91, 283
memtune 89, 284
model 126, 133, 285
model as a child element of cpu 87
model as a child element of interface 286
name 75, 287, 312, 323
network 313
on_crash 75, 288
on_poweroff 75
on_reboot 75, 289
os 75, 80, 290
parent 324
path 325, 330, 338
pool 331
readonly 122, 291
rng 292
shareable 293

libvirt XML element *(continued)*

shares 75, 86, 294
soft_limit 89, 295
source 103, 109, 111, 122, 124, 126, 133, 296, 333
target 95, 103, 109, 111, 122, 133, 300, 301, 334, 339
type 75, 302
type child of capability 326
uuid 327
vcpu 75, 85, 303
virtualport 126, 304, 314
volume 340
watchdog 305
zpci 306
libvirt-internal configuration
displaying 158
libvirtd log messages 217
lin_tape device driver 19, 37
Linux scheduling 199
list virsh command
--all option 149, 156
listen element
address attribute 277
type attribute 277
live migration
cancellation 170
concurrency 169
concurrent connections 169
connections 169
deadlock prevention 169
destination port 169
firewall configuration 169
hardware dependencies 164
host environments 169
host setup 169
hypervisor release 169
image files 166
live phase 170
maximum downtime 170
MaxStartups parameter 169
memory intensive workload 170
memory intensive workloads 170
messages 378
migration port range 169
migration_port_max parameter 169
non-tunneled 169
performance considerations 169
phases 170
port range 169
prerequisites 166
preserving virtual server resources 166
process 170
setup 166
stopped phase 170
storage keys 170
verification 170
virtual server CPUs 166
virtual server memory 166
virtual server network 166
virtual server storage 166
live phase of a migration 170
live subcommand
or perf kvm stat 223
live virtual server migration, *See* live migration
load balancing 33

symmetric encryption 97
sysfs attribute
    bridge_state 49
system image
    saved 152
    saving 152
system journal 217
system resources
    configuring 73
    managing 175

T

tape 102
target attribute
    of the address element 117, 122, 240, 244
target element
    address attribute 300
    bus attribute
        scsi value 122
        virtio value 103, 109, 111, 301
    dev attribute 103, 109, 111, 122, 301
    port attribute 300
    type attribute
        sclp value 95, 300
        virtio value 95, 300
TDEA 97
TDES 97
terminating
    virtual servers 4, 152
threads value
    of the driver io attribute 259
topology 11, 17
trans attribute
    of the geometry element 265
Triple DEA 97
Triple DES 97
trustGuestRxFilters attribute
    of the interface element 124, 272
tuning
    virtual CPUs 86
    virtual memory 89
type
    of the virtual channel path 102
type attribute
    drive value 240
    of the address element 103, 109, 111, 115, 117, 122, 239, 240
    of the capability element 319
    of the console element 95, 250
    of the controller element 115, 251
    of the disk element 103, 109, 111, 122, 255
    of the domain element 75, 256
    of the driver element 103, 109, 111, 122, 259
    of the format element 336
    of the hostdev element 117, 267
    of the interface element 124, 126, 272
    of the model element 124, 126, 286
    of the pool element 331
    of the target element 95, 300
    of the virtualport element 126, 304, 314
    of the volume element 340
    scsi value 251, 267
    virtio-serial value 251

type element
    arch attribute 302
    machine attribute 302
type element as child of capability 326

U

udev-created by-path device node 103
udev-created device node 11, 17, 103
UID 11, 17
uid attribute
    of the address element 240
    of the zpci element 306
undefine virsh command 149, 393
undefined virtual server 4
undefining
    virtual servers 4, 149
unfiltered value
    of the hostdev sgio attribute 267
unique ID 11, 17
unit 37
unit attribute
    of the address element 117, 122, 240, 244
    of the memory element 89
    of the soft_limit element 295
universally unique identifier 11, 17
unknown reason
    of the shut off state 230
unpaused reason
    of the running state 231
unplugging
    devices 183
unsafe value
    of the driver cache attribute 259
uplink port
    creating 49
user reason
    of the paused state 232
user space
    configuring 93
user space process 258, 259
user-friendly name 33
UUID 11, 17
uuid attribute
    of the address element 244
uuid element 327

V

value attribute
    of the attr element 318
vcpu element
    current attribute 303
vcpucount virsh command
    --active option 394
    --config option 394
    --current option 394
    --live option 394
    --maximum option 394
vectors attribute
    of the controller element 251
verifying
    a live migration 170

IBM.