Linux on z Systems

KVM Virtual Server Management

March 2016

SC34-2752-01

Linux on z Systems

KVM Virtual Server Management

March 2016

SC34-2752-01

Contents

About this document

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

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

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

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

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

This document describes a selection of helpful libvirt XML elements and virsh commands that can be used to perform the documented administration tasks on z Systems. The described subset is not complete.

KVM users familiar with other platforms should be aware that:

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

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

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

How this document is organized

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

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

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

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

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

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

Conventions and assumptions used in this publication

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

Authority

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

Persistent configuration

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

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

Terminology

This document uses the following terminology:

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

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

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

Highlighting

This publication uses the following highlighting styles:

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

```
within a screen frame.
Prompts on the KVM host are shown as hash signs:
#
Prompts on the KVM virtual server are shown as hash signs 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 IBM® information about KVM, see www.ibm.com/systems/virtualization/kvm.

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

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

libvirt virtualization API

libvirt provides the management API on the host.

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

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

QEMU

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

For QEMU documentation, see wiki.qemu.org.

Other publications

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

Other publications for Linux on z Systems

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

These publications are available on IBM Knowledge Center at

www.ibm.com/support/knowledgecenter/linuxonibm/liaaf/lnz_r_lib.html

- *Device Drivers, Features, and Commands* (distribution-specific editions)
- *Using the Dump Tools* (distribution-specific editions)
- *KVM Virtual Server Management*, SC34-2752
- *KVM Virtual Server Quick Start*, SC34-2753
- *Installing SUSE Linux Enterprise Server 12 as a KVM Guest*, SC34-2755

- *Device Drivers, Features, and Commands for Linux as a KVM Guest*, SC34-2754
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *libica Programmer's Reference*, SC34-2602
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Linux on z Systems Troubleshooting*, SC34-2612
- *Linux Health Checker User's Guide*, SC34-2609
- *Kernel Messages*, SC34-2599
- *How to Set up a Terminal Server Environment on z/VM®*, SC34-2596

These publications are available on developerWorks at

www.ibm.com/developerworks/linux/linux390/documentation_dev.html
- *Device Drivers, Features, and Commands*, SC33-8411
- *Using the Dump Tools*, SC33-8412
- *KVM Virtual Server Management*, SC34-2752
- *KVM Virtual Server Quick Start*, SC34-2753
- *Installing SUSE Linux Enterprise Server 12 as a KVM Guest*, SC34-2755
- *Device Drivers, Features, and Commands for Linux as a KVM Guest*, SC34-2754
- *How to Improve Performance with PAV*, SC33-8414
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *How to use Execute-in-Place Technology with Linux on z/VM*, SC34-2594
- *How to Set up a Terminal Server Environment on z/VM*, SC34-2596
- *Kernel Messages*, SC34-2599
- *libica Programmer's Reference*, SC34-2602
- *Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide*, SC33-8294
- *Exploiting Enterprise PKCS #11 using openCryptoki*, SC34-2713
- *Linux on z Systems Troubleshooting*, SC34-2612
- *Linux Health Checker User's Guide*, SC34-2609

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

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

Part 1. General concepts

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

Chapter 1. Overview

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

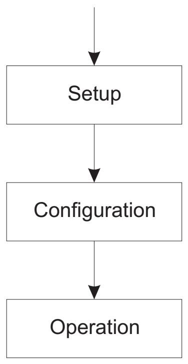


Figure 1. Virtual server administrator's tasks

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

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

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

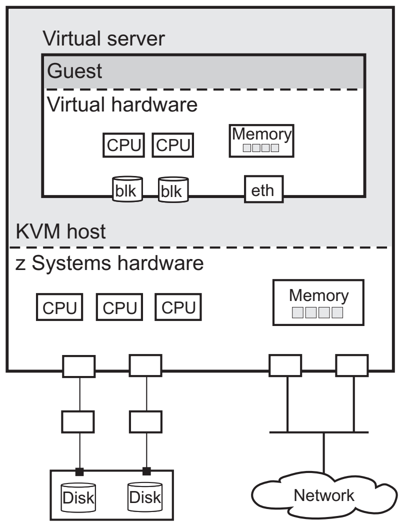


Figure 2. KVM host with a virtual server including a guest operating system

Virtual server management tasks

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

1. Device setup

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

   For a detailed description of this task, see Part 2, “Device setup,” on page 27.

2. Virtual server and device configuration

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

   You can also configure *hotplug devices* by using *device configuration-XMLs*.

   For a detailed description of this task, see Part 3, “Configuration,” on page 49.

3. Virtual server and device operation

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

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

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

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

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

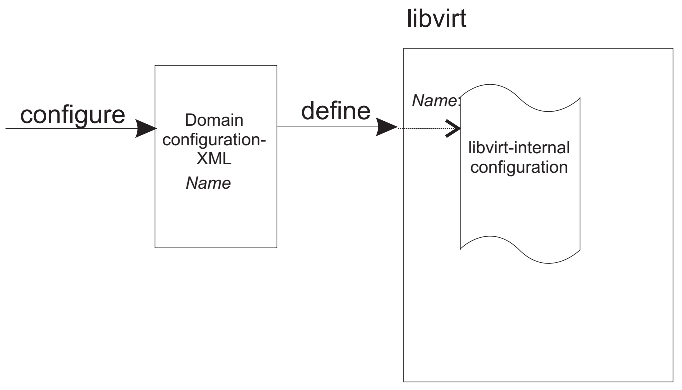


Figure 3. Creating a persistent virtual server definition

   b. Now you can manage the *operation* of the virtual server. This consists of:

      - Life cycle management:

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

        You can issue virsh commands to start, terminate, suspend, or resume a virtual server (see Figure 4 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 dynamically attach devices to or detach devices from a defined virtual server. If the virtual server is running, the devices are hotplugged or unplugged.

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

For a detailed description of these tasks, see Part 4, “Operation,” on page 93.

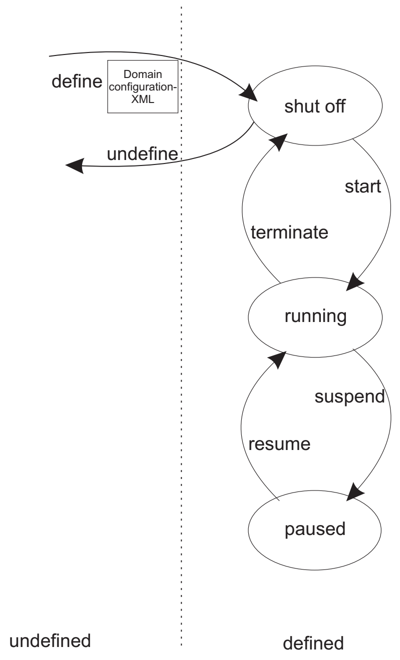


Figure 4. Simplified state-transition diagram of a virtual server

Virtualization components

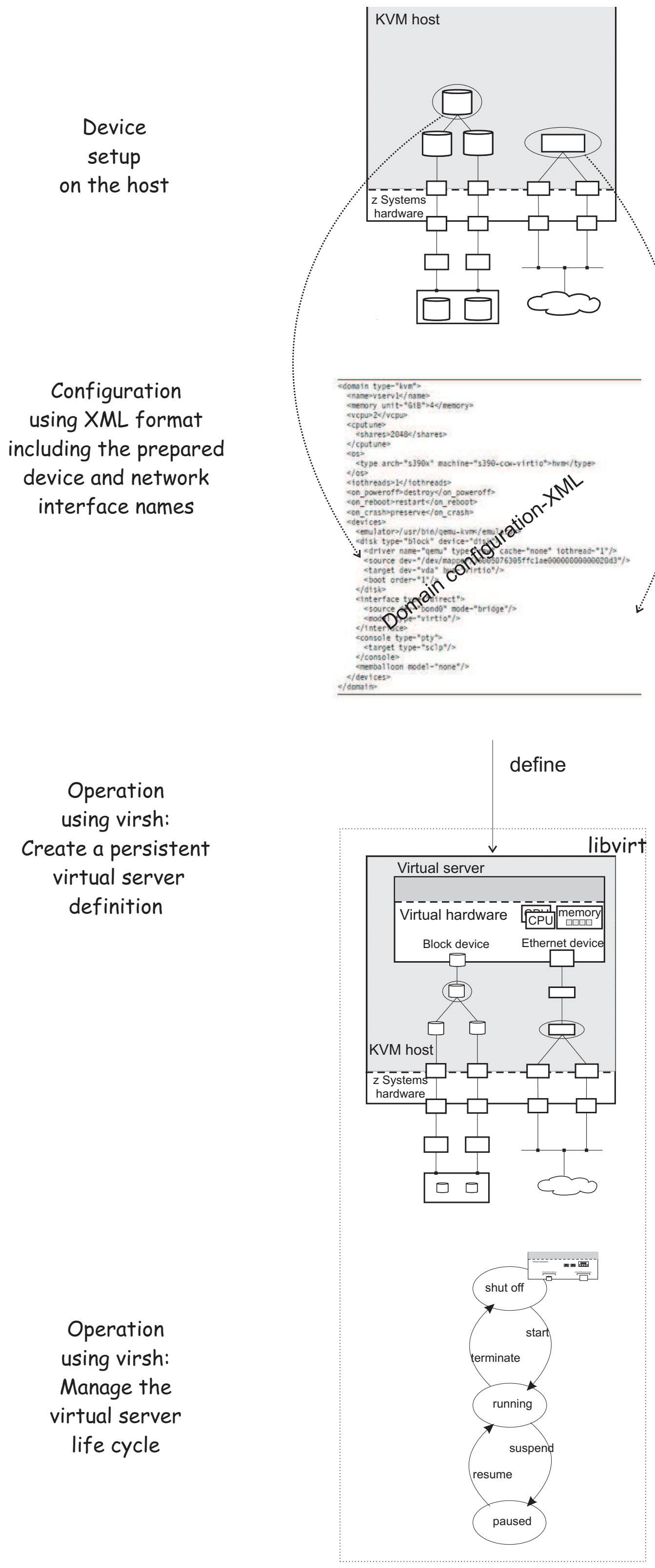
The virtual server management as described in this document is based on the following virtualization components.

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

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

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

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



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

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

Chapter 2. DASDs and SCSI disks as virtual block devices

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

On the host, you manage various types of disk devices and their configuration topology. Path redundancy in the setup of FC-attached SCSI disks guarantees high availability of the devices. Analogous, multipathing is implemented in the z Systems hardware of DASDs.

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

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

   In particular, prepare multipathing to guarantee high availability, because virtual block devices cannot be multipathed on the virtual server.

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

   See Chapter 6, “Preparing DASDs,” on page 29 and Chapter 7, “Preparing SCSI disks,” on page 31.
2. Configure DASDs, FC-attached SCSI disks, and files as virtual block devices.

   You configure devices that are persistent for a virtual server in its domain configuration-XML file and hotplug devices in a separate device configuration-XML file.

   See Chapter 11, “Configuring devices,” on page 69 and “Configuring virtual block devices” on page 72.

Virtual block device configuration topology

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

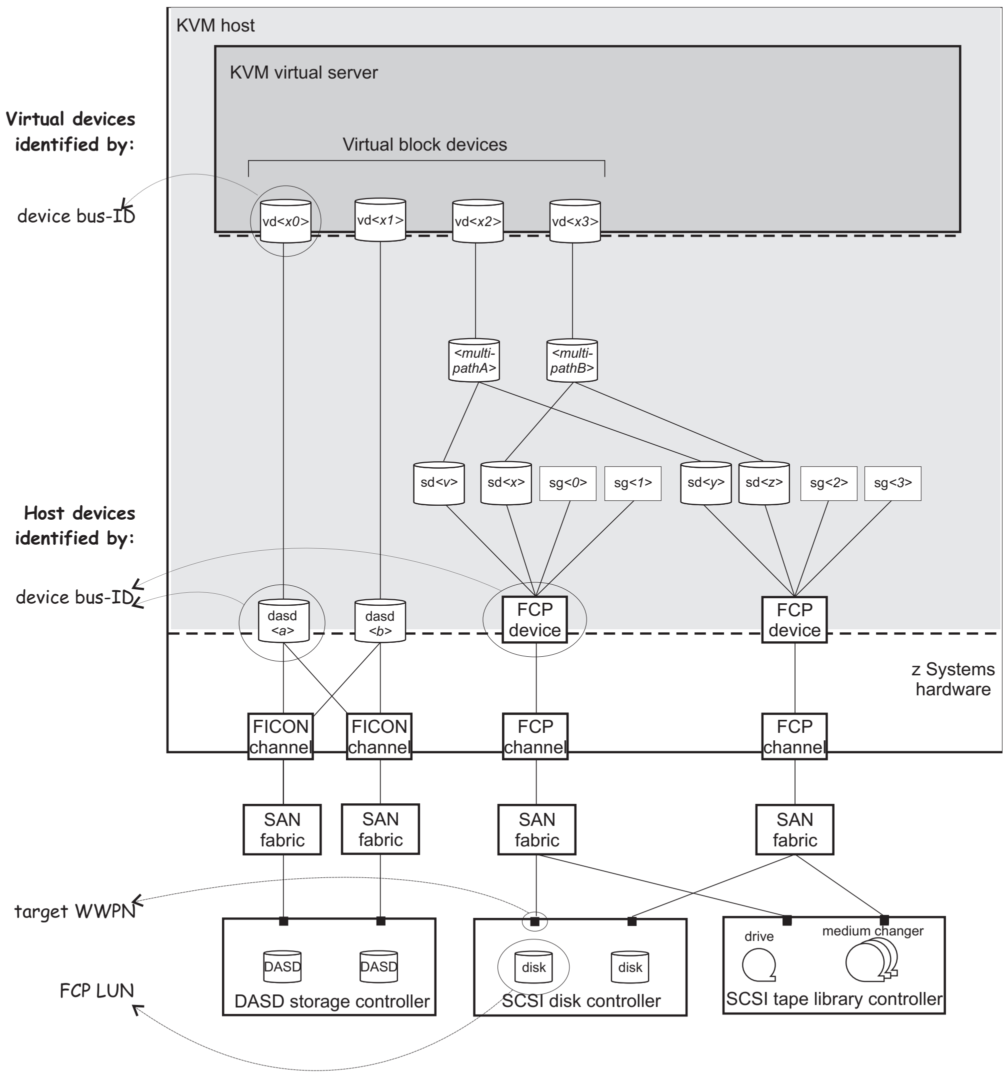


Figure 6. Multipathed DASD and SCSI disks configured as virtual block devices

Disk device identification

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

Device bus-ID and device number of an FCP device

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

```
0.m.dddd
```

Where:

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

Example:

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

`1700`	device number of the FCP device.

Device bus-ID and device number of a DASD

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

`0.m.dddd`

Example:

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

Unique ID (UID) of a DASD

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

Example:

`IBM.75000000010671.5600.00`

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

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

`0.m.dddd`

Where:

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

Example:

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

Standard device name

Standard device names are of the form:

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

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

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

Example:

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

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

Standard device node

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

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

Example:

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

udev-created device node

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

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

Example for DASDs on the host:

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

where:

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

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

where:

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

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

where:

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

Example for SCSI devices on the host:

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

where:

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

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

where:

`scsi-36005076303ffc56200000000000010ea`	is the SCSI identifier.

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

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

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

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

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

Virtual server example:

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

where:

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

Device mapper-created device node

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

Example:

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

where

`p1`	denotes the first partition of the device.

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

Related publications

- *Device Drivers, Features, and Commands*, SC33-8411
- *How to use FC-attached SCSI devices with Linux on z Systems*, SC33-8413
- *KVM for IBM z Systems: System Administration*, SC27-8237

Chapter 3. SCSI tapes and SCSI 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 7 on page 16 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 35.
2. Configure the SCSI tape or medium changer device as hotplug device.

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

   See Chapter 11, "Configuring devices," on page 69 and "Configuring virtual SCSI devices" on page 80.

Virtual SCSI device configuration topology

Figure 7 on page 16 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.

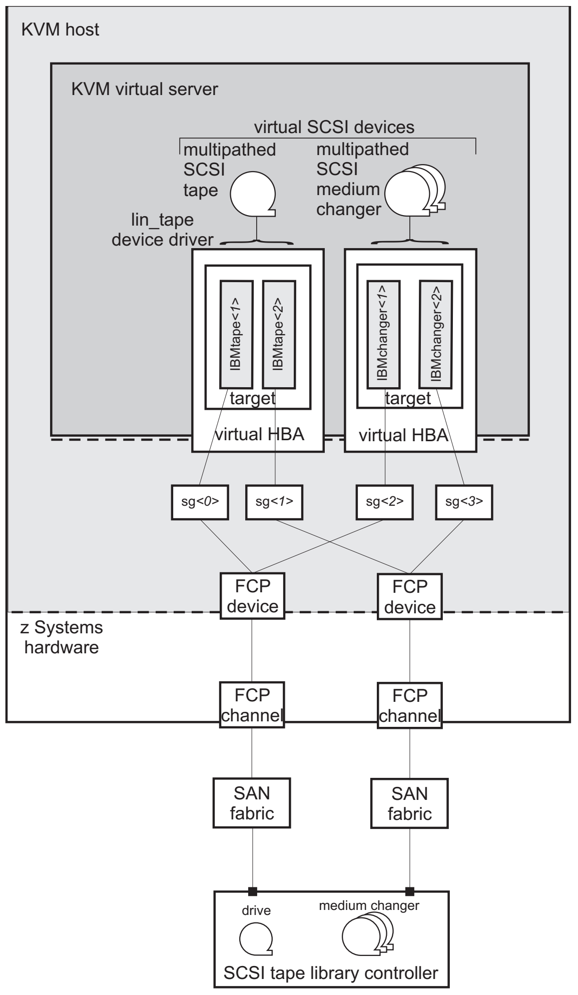


Figure 7. Multipathed SCSI tapes and SCSI medium changer devices configured as virtual SCSI devices

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 publications

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

Chapter 4. Network devices as virtual Ethernet devices

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

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

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

Whatever configuration you choose, be sure to provide high reliability through path redundancy as shown in Figure 8:

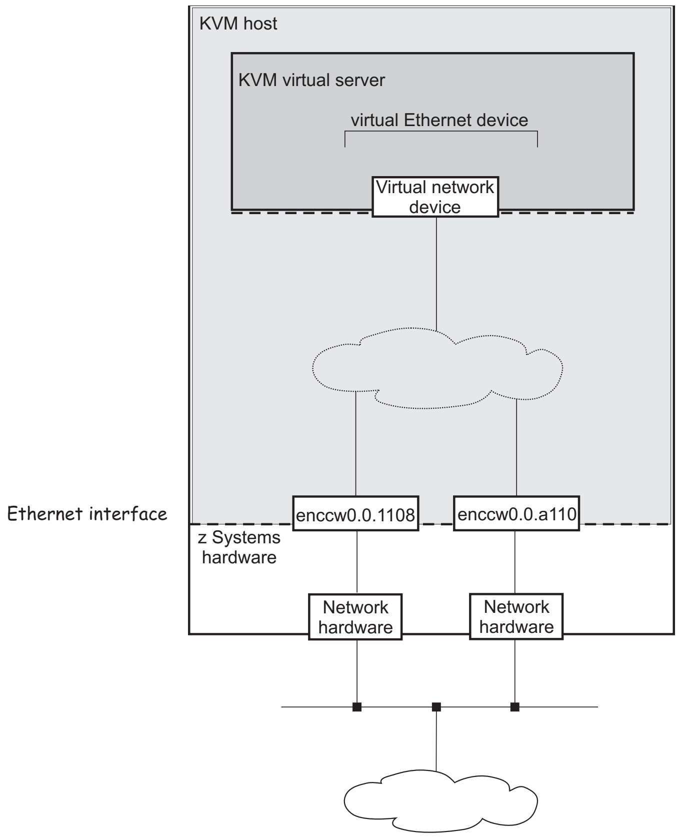


Figure 8. Highly reliable virtual network device configuration

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.

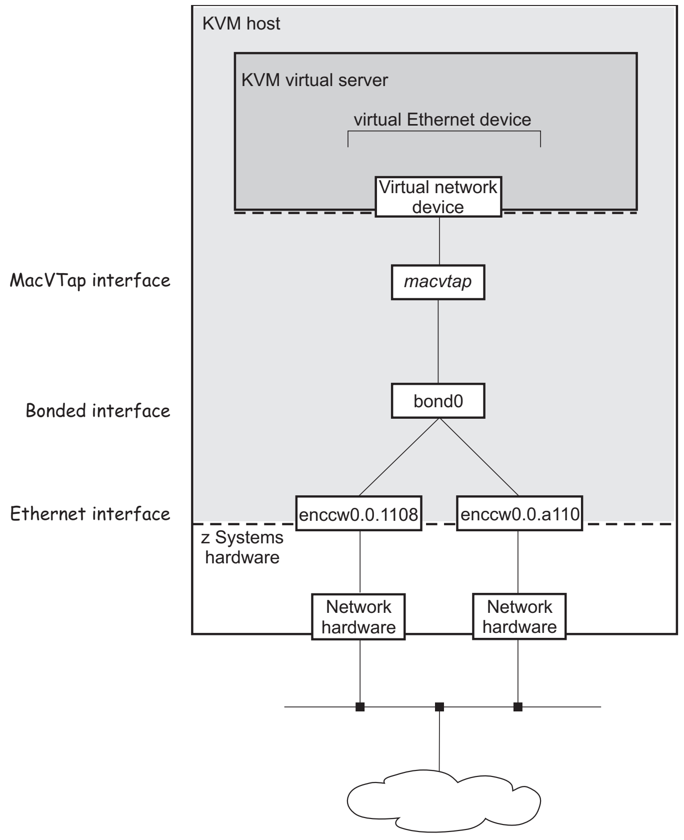


Figure 9. Configuration using a direct MacVTap connection

When you configure a virtual Ethernet device, you associate it with a network interface name on the host in the configuration-XML. In Figure 9, 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 8 on page 19 can be realized as follows:

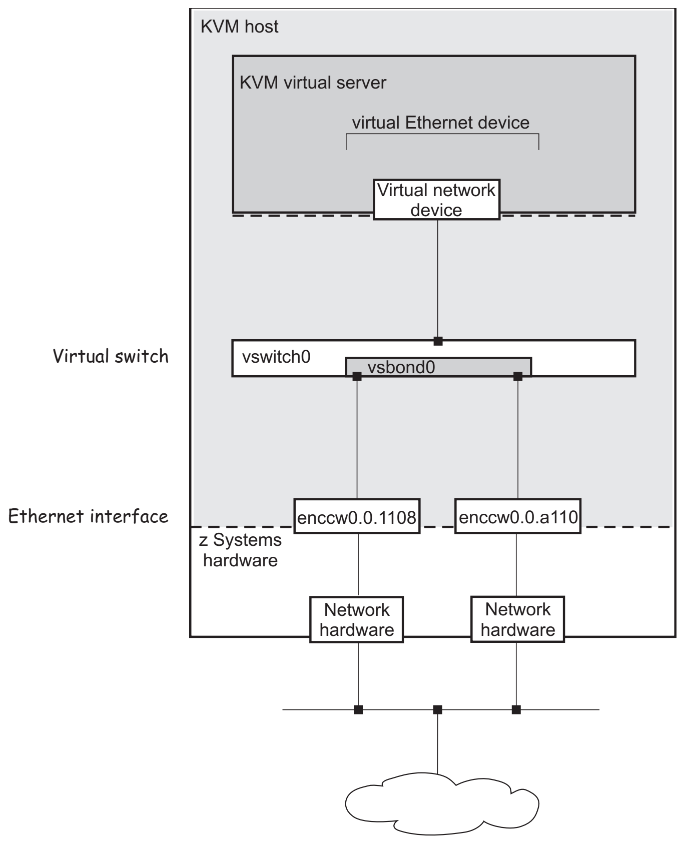


Figure 10. Configuration using a virtual switch

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

Related publications

- *Device Drivers, Features, and Commands*, SC33-8411
- *KVM for IBM z Systems: System Administration*, SC27-8237
- Libvirt networking documentation at wiki.libvirt.org/page/Networking

Related tasks:

Chapter 9, "Preparing network devices," on page 39
Consider these aspects when setting up network interfaces for the use of virtual servers.

"Configuring virtual Ethernet devices" on page 89
Configure network interfaces, such as Ethernet interfaces, bonded interfaces,

virtual LANs, or virtual switches as virtual Ethernet devices for a virtual server.

Chapter 5. Migration to a different hypervisor release

The virtual server's machine type determines the hypervisor release which runs the virtual server on the host.

The hypervisor release is defined by the installed QEMU release, by the hypervisor product or by your distribution.

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.

Virtual server definition

When you define a virtual server, libvirt replaces the alias machine type by the machine type which reflects the current hypervisor release of the host running the virtual server. In the libvirt-internal configuration, the machine attribute of the type element then reflects the installed hypervisor release.

Example:

Domain configuration-XML using the alias machine type:
```
<type arch="s390x" machine="s390-ccw-virtio">hvm</type>
```

Libvirt-internal configuration for KVM for IBM z Systems™ V1.1.0:
```
<type arch="s390x" machine="s390-ccw-virtio-kvmibm-1.1.0">hvm</type>
```

Libvirt-internal configuration for KVM for IBM z Systems V1.1.1:
```
<type arch="s390x" machine="s390-ccw-virtio-kvmibm-1.1.1">hvm</type>
```

Libvirt-internal configuration for QEMU release 2.5.0:
```
<type arch="s390x" machine="s390-ccw-virtio-2.5">hvm</type>
```

Depending on your distribution, there may be additional machine types. The following command displays the available machine types:

```
# qemu-kvm --machine help
```

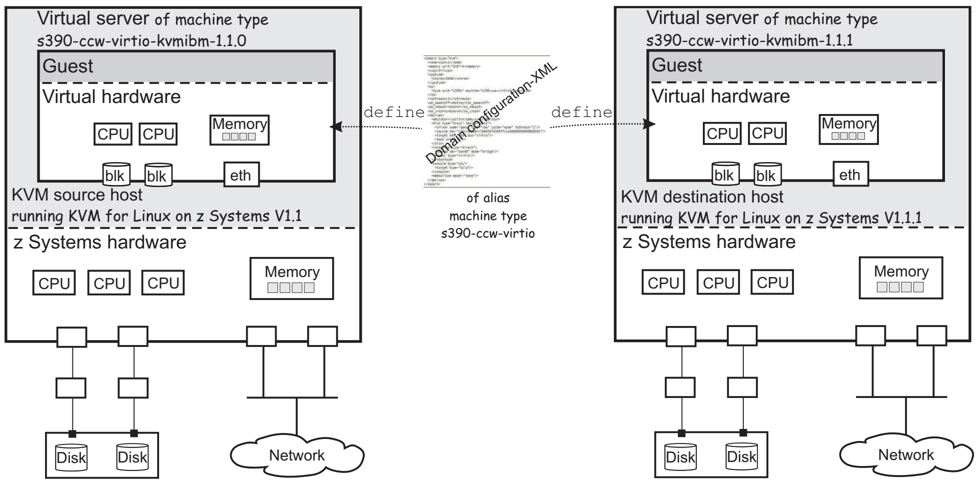

Virtual server of machine type s390-ccw-virtio-kvmibm-1.1.0
Guest
Virtual hardware
CPU
CPU
Memory
blk
blk
eth
KVM source host running KVM for Linux on z Systems V1.1
z Systems hardware
Memory
Disk
Disk
Network
define
Domain configuration-XML
define
of alias machine type s390-ccw-virtio
Virtual server of machine type s390-ccw-virtio-kvmibm-1.1.1
KVM destination host running KVM for Linux on z Systems V1.1.1

Figure 11. Defining virtual servers on different hosts

Figure 11 shows that creating virtual servers from the same domain configuration-XML file on different hosts results in different machine types.

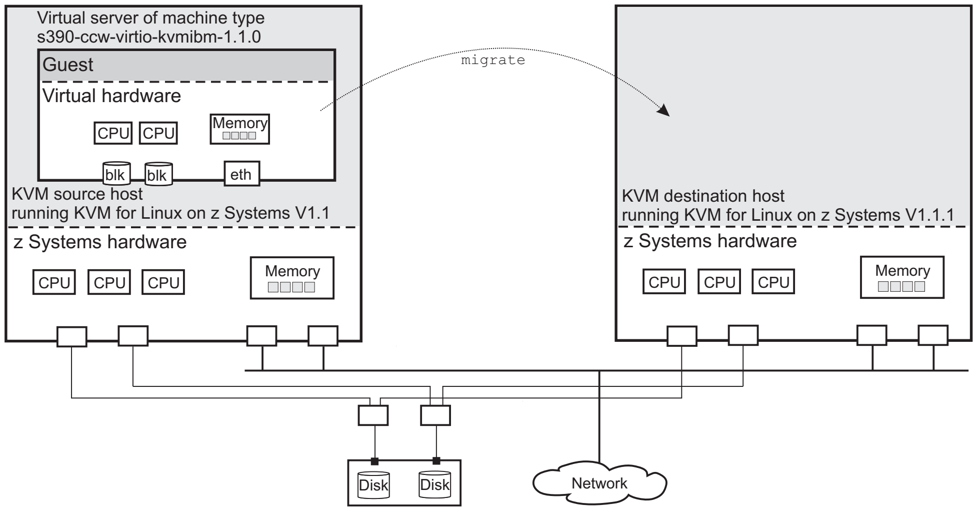
Live virtual server migration

A live virtual server migration preserves the machine type of the virtual server. The libvirt-internal configuration is not changed, that is, the machine type still reflects the hypervisor release of the source host. Newer hypervisor releases are compatible with earlier versions.

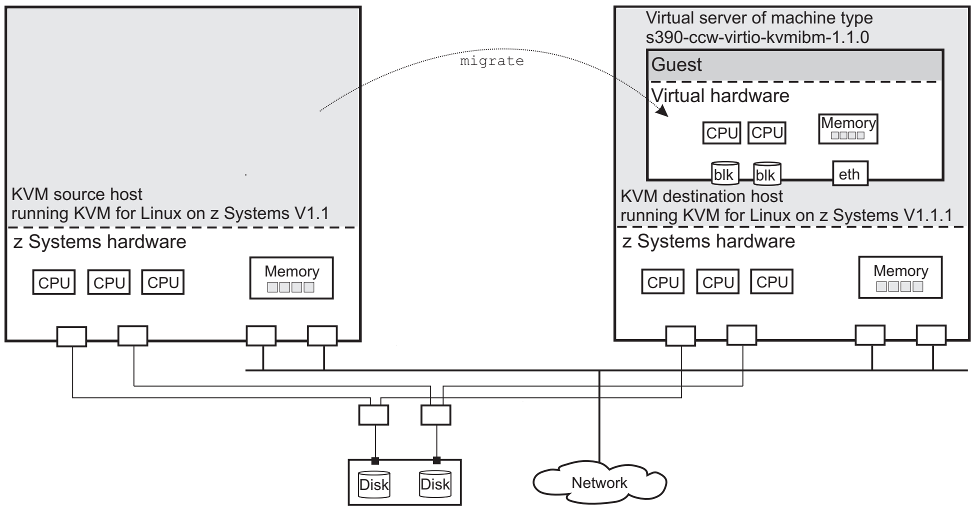
However, if you try to migrate a virtual server to a destination host with an earlier hypervisor release than the currently reflected machine type, you need to explicitly specify this earlier machine type in the virtual server definition before the migration.

Example:

1. Before the migration, the virtual server is running on the source host with hypervisor release KVM for Linux on z Systems V1.1.0. The virtual server's machine type is s390-ccw-virtio-kvmibm-1.1.0.


Virtual server of machine type s390-ccw-virtio-kvmibm-1.1.0
Guest
Virtual hardware
CPU
CPU
Memory
blk
blk
eth
KVM source host running KVM for Linux on z Systems V1.1
z Systems hardware
Memory
migrate
KVM destination host running KVM for Linux on z Systems V1.1.1
Disk
Disk
Network

2. After the migration, the virtual server is running on the destination host with hypervisor release KVM for Linux on z Systems V1.1.1. The virtual server's machine type is still s390-ccw-virtio-kvmibm-1.1.0.



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.

Related tasks:

Chapter 15, "Performing a live migration," on page 109
Migrate a running virtual server from one host to another without affecting the virtual server.

"Defining a virtual server" on page 96
Create a persistent definition of a virtual server configuration.

"Modifying a virtual server definition" on page 96
Edit the libvirt-internal configuration of a defined virtual server.

"Displaying the current libvirt-internal configuration" on page 106
The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, and is enhanced by libvirt-internal information and the dynamically attached devices.

Related reference:

"Domain configuration-XML" on page 53
Configure a virtual server with a domain configuration-XML file.

"<type>" on page 194
Specifies the machine type.

Part 2. Device setup

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

Chapter 6. Preparing DASDs

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

Before you begin

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

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

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

About this task

The following publications describe in detail how to configure, prepare, and work with DASDs:

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

Procedure

The following steps describe a dynamic DASD setup on the host.
To set up DASDs persistently across host reboots, refer to your host administration documentation (see also "Persistent configuration" on page viii).

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

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

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

Example

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

   For example, for device `0.0.7500`, issue:

```
# chccwdev —e 0.0.7500
```

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

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

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

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

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

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

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

What to do next

Configure the DASDs as described in “Configuring a DASD or SCSI disk” on page 72.

Related concepts:

Chapter 2, “DASDs and SCSI disks as virtual block devices,” on page 9
DASDs and FC-attached SCSI disks are 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
- *KVM for IBM z Systems: System Administration*, SC27-8237

Procedure

The following steps describe a dynamic SCSI disk setup on the host.
If you want to set up a SCSI disk persistently across host reboots, refer to your host administration documentation (see also “Persistent configuration” on page viii).

1. Linux senses the available FCP devices.

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

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

2. Set the FCP device online.

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

3. Configure the SCSI disks on the host.

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

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

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

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

Example

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

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

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

1. Display the available FCP devices.

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

2. Set the FCP device online.

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

3. Configure the SCSI disk on the host.

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

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

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

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

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

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

`/dev/mapper/36005076305ffc1ae00000000000023bc`

What to do next

Configure the SCSI disks as described in “Configuring a DASD or SCSI disk” on page 72.

Related concepts:

Chapter 2, “DASDs and SCSI disks as virtual block devices,” on page 9
DASDs and FC-attached SCSI disks are 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
- *KVM for IBM z Systems: System Administration*, SC27-8237

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

Procedure

The following steps describe a dynamic SCSI tape or medium changer setup on the host.
If you want to set up a SCSI tape or medium changer persistently across host reboots, refer to your host administration documentation (see also “Persistent configuration” on page viii).

1. Linux senses the available FCP devices.

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

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

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

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

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

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

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

   You obtain this information by issuing:

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

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

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

Example

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

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

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

1. Display the available FCP devices:

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

2. Bring the FCP device online:

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

3. Register the SCSI tape device on the host:

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

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

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

   where:

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

SCSI LUN	`0`

What to do next

Configure the SCSI tape and medium changer devices as described in "Configuring a SCSI tape or medium changer device" on page 81.

Related concepts:

Chapter 3, "SCSI tapes and SCSI medium changers as virtual SCSI devices," on page 15
FC-attached SCSI tape and medium changer devices are virtualized as virtio SCSI devices.

Chapter 9. Preparing network devices

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

About this task

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

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

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

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

Procedure

1. Create network interfaces as described in “Creating a network interface” on page 40.
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 42.
   b. To configure a virtual switch, perform the steps described in “Preparing a virtual switch” on page 45.

      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 89.

Related concepts:

Chapter 4, “Network devices as virtual Ethernet devices,” on page 19
Virtualize network devices as virtual Ethernet devices by configuring direct MacVTap connections or virtual switches.

Creating a network interface

Create a network interface for a network device.

Before you begin

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

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

```
# lsqeth -p
```

About this task

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

For a description of the necessary steps, refer to your host administration documentation (see also “Persistent configuration” on page viii).

Procedure

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

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

   ```
   # znetconf -u
   ```

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

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

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

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

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

   ```
   # znetconf -c
   ```

3. Activate the network interfaces.

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

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

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

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

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

Example

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

1. Determine the available network devices.

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

2. Configure the network devices and set them online.

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

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

3. Activate the network interfaces.

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

# ip link set enccw0.0.8050 up
```

4. Increase the transmit queue length.

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

What to do next

Prepare the configuration-specific setup as described in:

- "Preparing a network interface for a direct MacVTap connection" on page 42
- or "Preparing a virtual switch" on page 45

Preparing a network interface for a direct MacVTap connection

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

Before you begin

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

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

Procedure

1. Create a bonded interface to provide high availability.

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

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

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

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

   Example:

   Create a virtual LAN interface with VLAN ID 623.

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

Preparing a bonded interface

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

Before you begin

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

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

About this task

The following steps describe a dynamic bonded interface setup on the host. If this bonded interface is expected to persist over subsequent reboots, you need to configure it persistently.

For a description of the necessary steps, refer to your host administration documentation (see also “Persistent configuration” on page viii).

Procedure

1. Define the bonded interface.

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

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

Example

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

1. Add a new master bonded interface:

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

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

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

3. Add slave interfaces to the bonded interface:

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

4. Activate the interface:

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

To verify the bonding settings, issue:

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

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

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

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

Related tasks:

"Configuring a MacVTap interface" on page 89
Configure network interfaces, such as Ethernet interfaces, bonded interfaces, virtual LANs, through a direct MacVTap interface.

Preparing a virtual switch

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

Before you begin

Make sure that:

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

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

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

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

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

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

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

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

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

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

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

About this task

Further information:

- *KVM for IBM z Systems: System Administration*, SC27-8237
- Open vSwitch command reference: openvswitch.org/support/dist-docs

Procedure

1. Create a virtual switch.

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

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

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

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

2. Create an uplink port.

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

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

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

Example

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

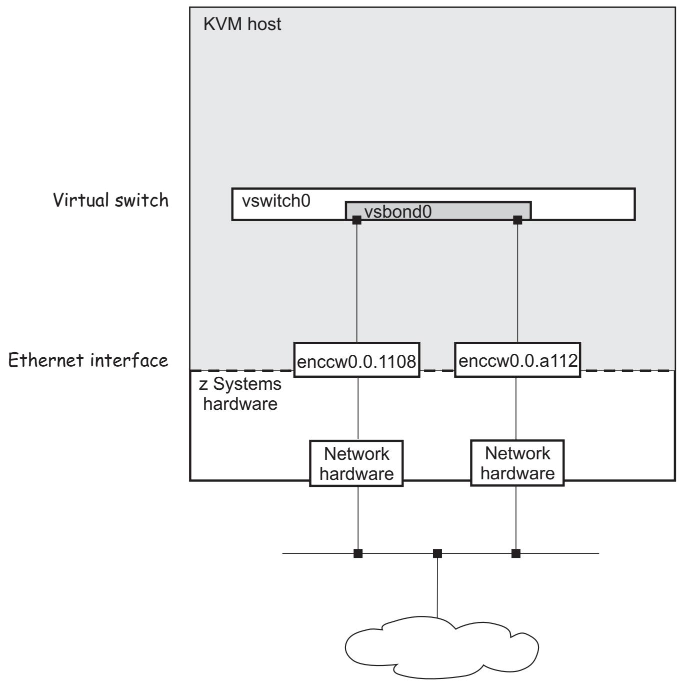


Figure 12. Virtual switch with a bonded interface

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

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

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

1. Create a virtual switch:

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

2. Create an uplink port:

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

Related tasks:

"Configuring a virtual switch" on page 91
Configure virtual switches as virtual Ethernet devices.

Part 3. Configuration

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

Chapter 10. Configuring a virtual server

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

Procedure

1. Create a domain configuration-XML file.

   See "Domain configuration-XML" on page 53.
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 55.

   b. Configure virtual CPUs.

      See "Configuring virtual CPUs" on page 60.

   c. Configure memory.

      See "Configuring virtual memory" on page 61.

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

      See "Configuring the collection of QEMU core dumps" on page 61.
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 62.

   b. Configure persistent devices.

      See "Configuring persistent devices" on page 63.

   c. Configure the console device.

      See "Configuring the console" on page 64.

   d. Optional: Configure a watchdog device.

      See "Configuring a watchdog device" on page 65.

   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 66.

   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 68.
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 96.

Domain configuration-XML

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

Root element

domain
Specify:

domain type attribute:	kvm

Selected child elements

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

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

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

cputune
Groups the CPU tuning parameters:

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

os Groups the operating system parameters:

| **type** Specifies the machine type.

kernel Optionally specifies the kernel image file on the host.

initrd Optionally specifies the initial ramdisk on the host.

cmdline
Optionally specifies command-line arguments.

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

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

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

on_crash
Configures the behavior of the virtual server when it crashes. Specify the preserve value.

on_crash element:	preserve

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

Example

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

Related reference:

Chapter 24, “Selected libvirt XML elements,” on page 153
These libvirt XML elements might be useful for you. You find the complete libvirt XML reference at libvirt.org.

Configuring the boot process

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

Before you begin

Ensure that there is a way to boot a guest.

About this task

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

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

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

  For a description, see "Configuring a kernel image file as IPL device" on page 56.

  For a description of the guest installation process, see:
  - *KVM for IBM z Systems: System Administration*, SC27-8237
  - *Installing SUSE Linux Enterprise Server 12 as a KVM Guest*, SC34-2755

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

Configuring a DASD or SCSI disk as IPL device

Boot a guest from a configured disk device.

Before you begin

Prepare a DASD or a SCSI disk, which contains a root file system with a bootable kernel as described in Chapter 6, "Preparing DASDs," on page 29 or Chapter 7, "Preparing SCSI disks," on page 31.

Procedure

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

   See "Configuring persistent devices" on page 63 and "Configuring a DASD or SCSI disk" on page 72.
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 158).

boot order attribute:	<*number*>

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

Example

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

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

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

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

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 172).

initrd element:	*<initial-ramdisk>*

b. Specify the fully qualified kernel image file in the kernel element, which is a child of the os element (see "<kernel>" on page 175).

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 160).

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

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

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

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

Example

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

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

2. Provide a disk for the guest installation:

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

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

   Example:

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

        <initrd>initial-ramdisk</initrd>
        <kernel>kernel-image</kernel>
        <cmdline>command-line-parameters</cmdline>
        ...
    </os>
    ...
    <devices>
        <emulator>/usr/bin/qemu-kvm</emulator>
        <!-- guest installation device -->

        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none"
                    io="native" iothread="1"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
            <target dev="vda" bus="virtio"/>
            <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
        </disk>
        <console type="pty">
            <target type="sclp"/>
        </console>
    </devices>
</domain>
```

3. Start the virtual server for the initial installation.
4. Install the guest as described in your administration documentation or in *Installing SUSE Linux Enterprise Server 12 as a KVM Guest*, SC34-2755.

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

   Example:

```
<domain>
    ...
    <os>
        ...
    </os>
    ...
    <devices>
        <emulator>/usr/bin/qemu-kvm</emulator>
        <!-- guest IPL disk -->

        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none"
                    io="native" iothread="1"/>
            <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
            <target dev="vda" bus="virtio"/>
            <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xe716"/>
        </disk>

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

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

Configuring virtual CPUs

Configure virtual CPUs for a virtual server.

Related concepts:

Chapter 18, “CPU management,” on page 127
Virtual CPUs are realized as threads, and scheduled by the process scheduler.

Related tasks:

“Managing virtual CPUs” on page 116
Modify 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

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 195).

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

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

Example

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

The virtual server has 5 virtual CPUs, which are all available at startup.

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.

Procedure

Use the cputune element to group CPU tuning elements.

You specify the CPU weight by using the shares element (see “<shares>” on page 188).

shares element:	*<CPU-weight>*

Example

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

Configuring virtual memory

Configure the virtual 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 179).

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.

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 179)

Example

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

Configuring the user space

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

Procedure

The optional emulator element contains path and file name of the user space process (see "<emulator>" on page 169).
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>*

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

Example:

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

Configuring persistent devices

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

Before you begin

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

Procedure

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

   For more information about I/O threads, see “I/O threads” on page 131.

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

   (see “<iothreads>” on page 174)

   Example:

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

2. Specify a configuration-XML for each device.

   Chapter 11, “Configuring devices,” on page 69 describes how to specify a configuration-XML for a device.

3. For each persistent device, place the configuration-XML as child element of the devices element in the domain configuration-XML file.

   Please note that hotplug devices are configured in separate device configuration-XML files.

Example

```
<domain type="kvm">
    <iothreads>1</iothreads>
    ...
    <devices>
        ...
        <disk type="block" device="disk">
            <driver name="qemu" type="raw" cache="none" io="native" 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 161).

   To configure a pty console, enter:

console type attribute:	pty

   Example:

   ```
   <console type="pty">
       ...
   </console>
   ```

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 192).

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

target type attribute:	sclp

   Example:

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

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

Related tasks:

"Connecting to the console of a virtual server" on page 123
Open a console when you start a virtual server, or connect to the console of a running virtual server.

Configuring a watchdog device

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

About this task

When the guest is loading the watchdog module, it provides the new device node `/dev/watchdog` for the watchdog device. The watchdog timer is started when the watchdog device is opened by the guest watchdog application. The watchdog application reports a positive status to the watchdog device at regular intervals. If the watchdog device does not receive a positive report within a specified time, the watchdog timer elapses, and QEMU assumes that the guest is in an error state. QEMU then triggers a predefined action against the guest. For example, the virtual server might be terminated and rebooted, or a dump might be initiated.

Procedure

Use the watchdog element as child of the devices element to configure a watchdog device (see "<watchdog>" on page 197).

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

Example

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

Disabling protected key encryption

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

Before you begin

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

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

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

About this task

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

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

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

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

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

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

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

Procedure

You configure the generation of wrapping keys by using the keywrap element (see "<keywrap>" on page 176).
Its child element cipher (see "<cipher>" on page 159) enables or disables the generation of a wrapping key and the use of the respective protected key

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

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

cipher name attribute:	aes | des
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="des" 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 178)

Example

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

Chapter 11. Configuring devices

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

About this task

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

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

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

The virtual server sees the following predefined values:

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

Procedure

1. Configure the device as described in:
   - "Configuring virtual block devices" on page 72
   - "Configuring virtual SCSI devices" on page 80
   - "Configuring virtual Ethernet devices" on page 89

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

Device configuration-XML

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

Virtual block device

Root element
disk

Selected child elements
driver, source, target, address

Example

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

Virtual SCSI device

Root element
hostdev

Selected child elements
source, address

Example

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

Virtual Host Bus Adapter

Root element
controller

Selected child elements
address

Example

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

Virtual Ethernet device

Root element
interface

Selected child elements
mac, source, model

Example

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

Related reference:

Chapter 24, “Selected libvirt XML elements,” on page 153
These libvirt XML elements might be useful for you. You find the complete libvirt XML reference at libvirt.org.

Configuring virtual block devices

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

About this task

- "Configuring a DASD or SCSI disk"
- "Configuring a file as storage device" on page 78

Configuring a DASD or SCSI disk

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

Before you begin

Make sure that

- DASDs are prepared as described in Chapter 6, "Preparing DASDs," on page 29.
- SCSI disks are prepared as described in Chapter 7, "Preparing SCSI disks," on page 31.

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 131.

About this task

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

Procedure

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

disk type attribute:	block
disk device attribute:	disk

   (see "<disk>" on page 165)

   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 167)

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

   For persistent devices:
   Specify a value between 1 and the number of I/O threads configured by the iothreads element in the domain

configuration-XML file. To improve performance, be sure that there is an I/O thread dedicated for this device.

For hotplug devices:
Specify the ID of the I/O thread that is created when the device is attached.

Example:

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

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

For more information about I/O threads, see "I/O threads" on page 131.

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

target bus attribute:	virtio

(see "<target> as child element of <disk>" on page 193)

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 189)

Note: You should be aware that the selection of the specified device node determines whether or not you will be able to:

- Perform a live migration of the virtual server accessing the device.
- Migrate the storage to another storage server or another storage controller.

For DASDs:
Use udev-created device nodes.

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

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

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

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

3. Identify the device on the virtual server.

a. Specify a unique logical device name.

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

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

(see "<target> as child element of <disk>" on page 193)

b. Optional: Specify a unique device number.

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

```
fe.n.dddd
```

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

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

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

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

(see "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 155)

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

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

Assign device numbers depending on your policy, such as:

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

Related concepts:

Chapter 2, "DASDs and SCSI disks as virtual block devices," on page 9
DASDs and FC-attached SCSI disks are virtualized as virtio block devices.

Example of a DASD configuration

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

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

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

Define the devices:

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

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

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

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

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

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

Example of a SCSI disk configuration

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

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

Define the devices:

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

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

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

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

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

Configuring a file as storage device

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

Before you begin

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

Use the QEMU command **`qemu-img create`** to create a qcow2 image file. See "Examples for the use of the qemu-img command" on page 247 for examples.

Procedure

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

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

   (see "<disk>" on page 165)

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

	raw file:	qcow2 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 167)

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

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

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

target bus attribute:	virtio

   (see "<target> as child element of <disk>" on page 193)

2. Identify the file on the host.

   Specify the file name.

source file attribute:	*<file-name>*

   (see "<source> as child element of <disk>" on page 189)

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 193)

b. Optional: Specify a device number.

   You specify a device bus-ID of the form

   ```
   fe.n.dddd
   ```

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

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

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

   (see “<address> as child element of <controller>, <disk>, <interface>, and <memballoon>” on page 155)

   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

```
<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 55
Specify the device that contains a root file system, or a prepared kernel image file.

Configuring virtual SCSI devices

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

About this task

- "Configuring virtual SCSI devices"
- "Configuring a SCSI tape or medium changer device" on page 81
- | "Configuring a virtual SCSI-attached CD/DVD drive" on page 86
- "Example of a multipathed SCSI tape and medium changer device configuration" on page 84

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 162).

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

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

   Example:

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

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

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

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

   (see "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 155)

   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 35:

- 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 80), or
- Connect to a new virtual HBA which will be automatically configured for you.

About this task

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

Tip: Configure FC-attached SCSI tape or medium changer devices as hotplug devices, that is, create a separate device configuration-XML file for each device. Attach the device only if necessary, and detach the device before you migrate the virtual server, or set one of the devices in the configuration path offline.

Procedure

1. Configure the SCSI tape or medium changer device using the hostdev element (see "<hostdev>" on page 171).

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 154 and "<address> as child element of <source>" on page 157)

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

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

(see "<address> as child element of <hostdev> or <disk>" on page 156)

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

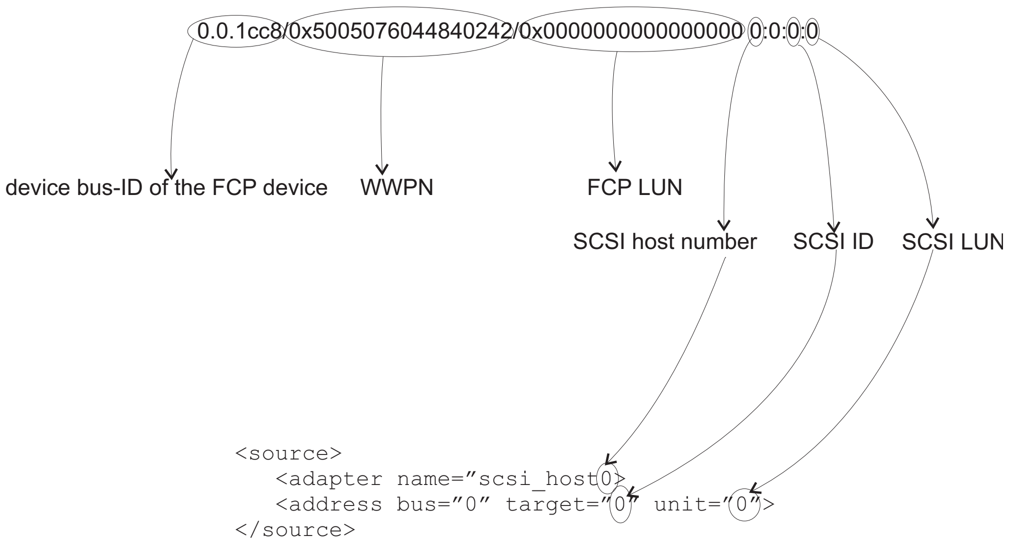
Tip: Choose a value between 0 and 255, because these values are identically mapped to the SCSI LUN on the virtual server.

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 0:0:0:0
```

where:


0.0.1cc8/0x5005076044840242/0x0000000000000000 0:0:0:0
device bus-ID of the FCP device
WWPN
FCP LUN
SCSI host number
SCSI ID
SCSI LUN
<source>
<adapter name=”scsi_host0>
<address bus=”0” target=”0” unit=”0”>
</source>

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

- Domain configuration-XML file:

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

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

- Device configuration-XML file:

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

Example of a multipathed SCSI tape and medium changer device configuration

Provide one virtual SCSI device for each configuration path.

About this task

This example provides a configuration for the topology as shown in Figure 7 on page 16.

Procedure

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

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

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

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

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

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

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

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

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

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

Configuring a virtual SCSI-attached CD/DVD drive

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

Before you begin

You need a virtual HBA to connect to.

- Either use a configured virtual HBA (see "Configuring a virtual HBA" on page 80), 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.

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 165).

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 167).

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 193).

target bus attribute:	scsi

   d. Specify the virtual DVD as read-only using the readonly element (see "<readonly>" on page 186).
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 189). 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 193).

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 156)

Where

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

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

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

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

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

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

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

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

Example

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

Related tasks:

"Replacing a virtual DVD" on page 121
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 9, "Preparing network devices," on page 39.

Procedure

- To configure a MacVTap interface, follow the steps described in "Configuring a MacVTap interface."
- To configure a virtual switch, follow the steps described in "Configuring a virtual switch" on page 91

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 173).
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 177)

   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 191)
3. Specify the model type (see "<model>" on page 181).

model type attribute:	virtio

Example

- To configure bonded interface bond0:

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

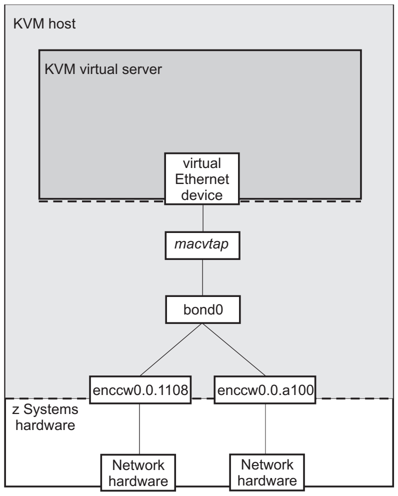


Figure 13. Direct interface type which configures a bonded interface

- To configure virtual LAN bond0.623:

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

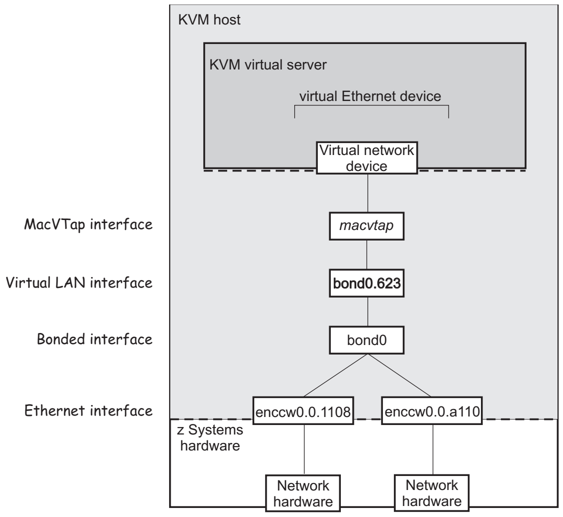


Figure 14. Direct interface type which configures a virtual LAN interface

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 173).

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 177)
2. Specify the virtual switch that you created before as described in “Preparing a virtual switch” on page 45.

source bridge attribute:	*<vswitch>*

   (see “<source> as child element of <interface>” on page 191)
3. Specify the type.

virtualport type attribute:	openvswitch

   (see “<virtualport>” on page 196)
4. Specify the model type.

model type attribute:	virtio

   (see “<model>” on page 181)

Example

Display the available virtual switches:

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

Configure the virtual switch which is shown in Figure 10 on page 21:

```
<interface type="bridge">
      <source bridge="vswitch0"/>
      <virtualport type="openvswitch"/>
      <model type="virtio"/>
</interface>
```

After the creation and the start of the virtual server, the virtual switch is displayed as follows:

```
# ovs-vsctl show
...
   Bridge "vswitch0"
        Port "vnet0"
            Interface "vnet0"
        Port "vsbond0"
            Interface "enccw0.0.1108"
            Interface "enccw0.0.a112"
        Port "vswitch0"
            Interface "vswitch0"
                type: internal
...
```

Part 4. Operation

Manage the operation of virtual servers using virsh commands.

Chapter 12. Creating, modifying, and deleting persistent virtual server definitions

Pass a virtual server configuration to libvirt, modify the libvirt-internal configuration, or delete it.

Before you begin

- Ensure that the libvirt daemon is running on the host:

```
# systemctl status libvirtd
libvirtd.service - Virtualization daemon
Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled)
Active: active (running) since Thu 2015-04-16 10:55:29 CEST; 2 months 3 days ago
Docs: man:libvirtd(8)
http://libvirt.org
Main PID: 5615 (libvirtd)
CGroup: /system.slice/libvirtd.service
├─5615 /usr/sbin/libvirtd
├─6750 /sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro ...
└─6751 /sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro ...
```

  If the libvirt daemon is not running, enter:

```
# systemctl start libvirtd.service
```

- Ensure that a domain configuration-XML file, which configures the virtual server, is created.

About this task

1. To create a persistent virtual server definition, you pass its domain configuration-XML file to libvirt. From the domain configuration-XML file, libvirt creates a libvirt-internal configuration, which may differ from the domain configuration-XML. For example, libvirt generates a UUID or MAC addresses for virtual Ethernet devices, if they are not specified.

   See "Defining a virtual server" on page 96.
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 96.
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 97.

Related reference:

Chapter 25, "Selected virsh commands," on page 199
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 206):

```
# 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 104) 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 106.
3. Start the virtual server as described in "Starting a virtual server" on page 100.
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 123.

Related reference:
Chapter 23, "Virtual server life cycle," on page 147
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 220):

```
# 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 100 and "Starting a virtual server" on page 100).

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 244):

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

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

Chapter 13. Managing the virtual server life cycle

Use libvirt commands to start, terminate, suspend, or resume a defined virtual server.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 227) 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 96.

About this task

- "Starting a virtual server" on page 100

  Start a defined virtual server.
- "Terminating a virtual server" on page 100

  Properly shut down a virtual server, save a system image, or, if necessary, immediately terminate it.
- "Suspending a virtual server" on page 102

  Pause a virtual server.
- "Resuming a virtual server" on page 102

  Transfer a paused virtual server to the running state.

Related reference:

Chapter 23, "Virtual server life cycle," on page 147
Display the state of a defined virtual server including the reason with the virsh **domstate --reason** command.

Chapter 25, "Selected virsh commands," on page 199
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Starting a virtual server

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

About this task

When you start a virtual server, usually, an Initial Program Load (IPL) is performed, for example to boot the guest. But if there is a saved system image for the virtual server, the guest is restored from this system image. It depends on the command that terminated a virtual server whether the system image was saved or not (see "Terminating a virtual server").

The "saved shut off" state indicates the availability of a saved system image. To display the state and the reason of a virtual server, enter the command:

```
# virsh domstate <VS> --reason
shut off (saved)
```

where <*VS*> is the name of the virtual server.

Refer to Chapter 23, "Virtual server life cycle," on page 147 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 241):

```
# 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 23, "Virtual server life cycle," on page 147 to see the effect of the virsh commands to terminate a virtual server depending on its state.

Procedure

- In most cases, you use the virsh **shutdown** command to properly terminate a virtual server (see "shutdown" on page 240).

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>
```

- Save the system image of a running or a paused virtual server and terminate it thereafter with the virsh **managedsave** command (see "managedsave" on page 229).

  ```
  # virsh managedsave <VS>
  ```

  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 (see "destroy" on page 207).

  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
  ```

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

Example

- To properly shut down virtual server vserv1, issue:

  ```
  # virsh shutdown vserv1
  Domain vserv1 is being shutdown
  ```

- To save the system image of virtual server vserv2 and properly shut it down, issue:

  ```
  # virsh managedsave vserv2
  Domain vserv2 state saved by libvirt
  ```

- To force a shutdown of virtual server vserv3, issue:

  ```
  # virsh destroy vserv3
  Domain vserv3 destroyed
  ```

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 23, “Virtual server life cycle,” on page 147 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 243):

```
# 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 23, “Virtual server life cycle,” on page 147 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 238):

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

<VS> Is the name of the virtual server.

Chapter 14. Monitoring virtual servers

Use libvirt commands to display information about a defined virtual server.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 227) 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 96.

About this task

- "Browsing virtual servers" on page 104

  View lists of all defined or of all running virtual servers.
- "Displaying information about a virtual server" on page 104

  View information about a virtual server, its state, its devices, or scheduling properties.
- "Displaying the current libvirt-internal configuration" on page 106

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

Related reference:

Chapter 25, "Selected virsh commands," on page 199
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 227):

```
# 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 215	
Current state	"domstate" on page 217	Display the reason of the current state by using the `--reason` option.
Scheduling information	"schedinfo" on page 239	
Virtual block devices	"domblkstat" on page 211	To retrieve the device name, use the virsh **domblklist** command.
Virtual Ethernet interfaces	"domifstat" on page 214	To retrieve the interface name, use the virsh **domiflist** command.
I/O threads	"iothreadinfo" on page 226	

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
```

- 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, and is enhanced by libvirt-internal information and the dynamically attached devices.

Procedure

To display the domain configuration-XML of a defined virtual server, use the virsh **dumpxml** command (see "dumpxml" on page 219):

```
# 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 hotplug 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-kvmibm-1.1.1">hvm</type>
  </os>
  <clock offset="utc"/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>preserve</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000020d3"/>
      <target dev="vda" bus="virtio"/>
      <boot order="1"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0000"/>
    </disk>
    <disk type="block" device="disk">
      <driver name="qemu" type="raw" cache="none" io="native" iothread="2"/>
      <source dev="/dev/mapper/36005076305ffc1ae00000000000021d7"/>
      <target dev="vdb" bus="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0002"/>
    </disk>
    <interface type="direct">
      <mac address="52:54:00:6a:0b:53"/>
      <source dev="bond0" mode="bridge"/>
      <model type="virtio"/>
      <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x0001"/>
    </interface>
    <console type="pty">
      <target type="sclp" port="0"/>
    </console>
    <memballoon model="none"/>
  </devices>
</domain>
```

libvirt added a number of XML elements to the current representation of the virtual server configuration. They are shown in **bold** typeface: a UUID, the current machine type which depends on the host setup and might be of the form "s390-virtio-ccw-<*x.x*>" as well, the emulator, mac address and address elements, and the dynamically attached device.

Chapter 15. Performing a live migration

Migrate a running virtual server from one host to another without affecting the virtual server.

Before you begin

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. There are no restrictions on the location of the destination host; it can run on another z System as well.

Hypervisor release

Newer hypervisor releases are compatible with earlier ones. A migration keeps the libvirt-internal configuration. Especially the machine type of the virtual server is not changed and still reflects the hypervisor release of the source host.

If you intend to migrate to a destination host with an earlier hypervisor release than the one of the source host, modify the machine type of the virtual server to the value that reflects the hypervisor release of the destination host (see also Chapter 5, “Migration to a different hypervisor release,” on page 23).

Prepare the migration of a virtual server to another host carefully:

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.

For 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.

For SCSI disks:

- Make sure that SCSI disks are configured using device mapper-created device nodes.

For 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.

For SCSI tapes or medium changer devices:

- When you migrate a virtual server that uses a configured virtual SCSI device, be aware that the SCSI device name, which is used to specify the source device, might change on the destination host.

  Tip: Make sure that SCSI tapes or medium changer devices are configured as hotplug devices. Detach them before you perform a migration. After the migration, reconfigure the devices before you reattach them.

"Disk device identification" on page 10 and "SCSI device identification" on page 16 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.

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.
- For non-SSH migrations, make sure that the firewall configuration of the involved systems allows access to all required network resources.

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.

Example:

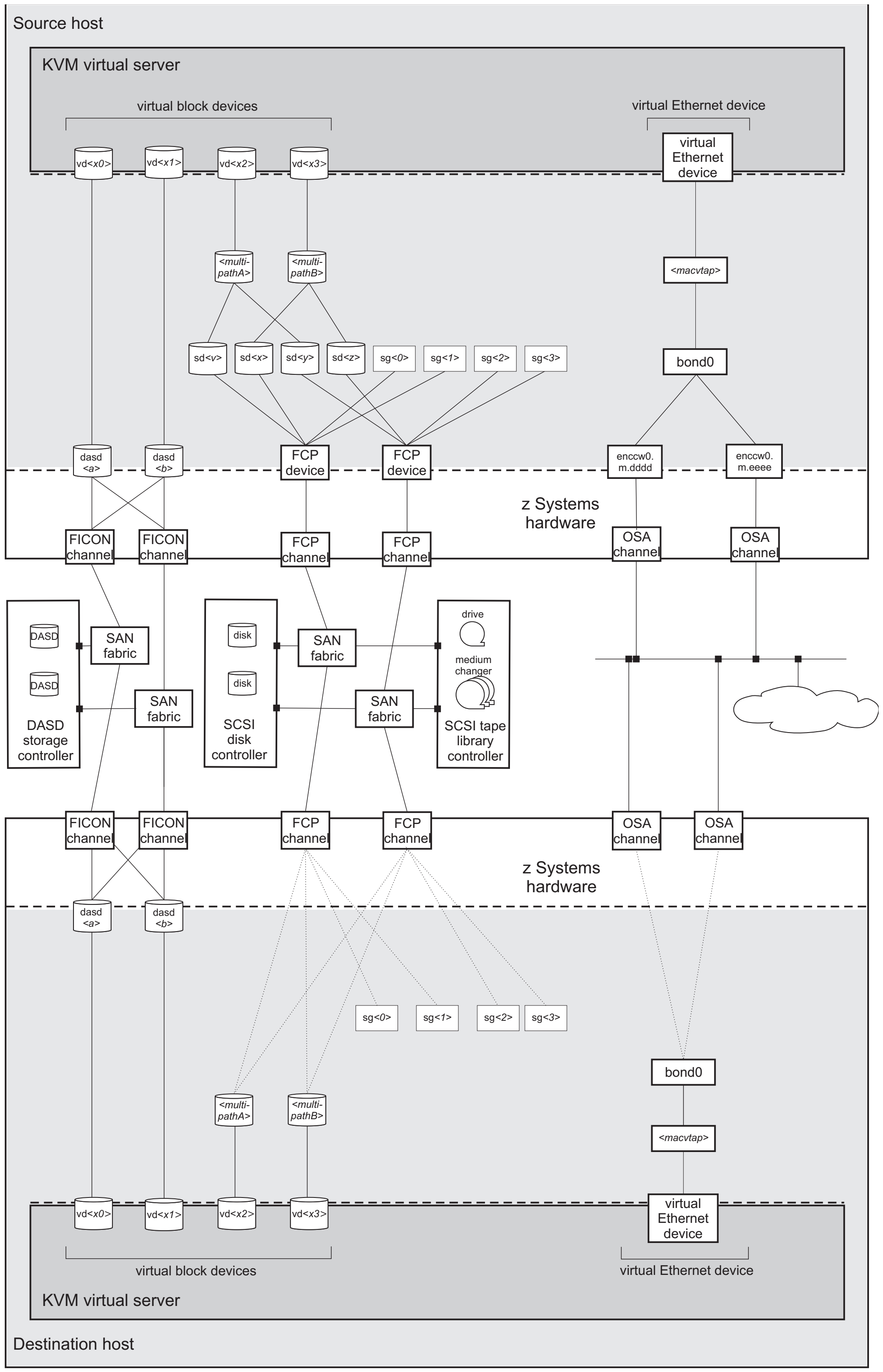


Figure 15. Example of a device setup on the source and destination hosts that allows the migration of the virtual server using these devices

About this task

In the literature, you also find the term virtual server *relocation* for what we call a live virtual server migration.

The migration of a virtual server from a source to a destination host consists of two phases:

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.

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 235). 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 236):

   ```
   # 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 234):

```
# 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 231):

```
# 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 to the host during a time-consuming migration process, increase the virsh keepalive interval (see Chapter 25, "Selected virsh commands," on page 199):

```
# virsh --keepalive-interval <interval-in-seconds>
```

The use of the virsh `--keepalive-interval` and `--keepalive-count` options preserves the communication connection between the host and the virtual server 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.

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 vserv1 to the destination host zhost. The virtual server will be transient on zhost, that is, after vserv1 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 vserv1 qemu+ssh://zhost/system
  ```

- This example starts a live migration of vserv2 to the destination host zhost. After a successful migration, vserv2 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 \
  vserv2 qemu+ssh://zhost/system
  ```

What to do next

- You can verify whether the migration completed successfully by looking for a running status of the virtual server on the destination, for example by using the virsh **list** command:

  ```
  # virsh list

  Id Name                State
  ----------------------------------
  10 kvm1               running
  ```

- You can cancel an ongoing migration operation by using the virsh **domjobabort** command:

  ```
  # virsh domjobabort <VS>
  ```

Chapter 16. Managing system resources

Use libvirt commands to manage the system resources of a defined virtual server, such as virtual CPUs.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 227) 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 96.

About this task

- "Managing virtual CPUs" on page 116

  Modify the portion of the run time that is assigned to the CPUs of a defined virtual server.

Related reference:

Chapter 25, "Selected virsh commands," on page 199
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 portion of the run time that is assigned to the virtual CPUs of a defined virtual server.

About this task

- "Modifying the virtual CPU weight"

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

Related concepts:

Chapter 18, "CPU management," on page 127
Virtual CPUs are realized as threads, and scheduled by the process scheduler.

Related tasks:

"Configuring virtual CPUs" on page 60
Configure virtual CPUs for a virtual server.

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 239):

```
# 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 60
Regardless of the number of its virtual CPUs, the CPU weight determines the shares of CPU time which is dedicated to a virtual server.

Chapter 17. Managing devices

Add, remove, or access devices of a running virtual server.

Before you begin

- Ensure that the libvirt daemon is running on the host.
- Use the virsh **list** command (see "list" on page 227) 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 96.

About this task

- "Attaching a device" on page 120

  Dynamically attach a hotplug device to a virtual server. If the virtual server is running, the device is hotplugged.
- "Detaching a device" on page 121

  Dynamically detach a hotplug device from a virtual server. If the virtual server is running, the device is unplugged.
- "Replacing a virtual DVD" on page 121

  Remove the currently provided ISO image, or provide a different one.
- "Connecting to the console of a virtual server" on page 123

  Connect to the console of a virtual server.

Related reference:

Chapter 25, "Selected virsh commands," on page 199
These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Attaching a device

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

Before you begin

- Ensure that the new device is not yet assigned to the virtual server.

  To list the devices that are assigned to a virtual server, you can
  - Display the current libvirt-internal configuration.
  - Use the virsh **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.
- Ensure that there is a device configuration-XML file for the device.

Procedure

1. Optional: If you attach a virtual block device, and the current libvirt-internal configuration does not provide an I/O thread for the hotplug device:

   Add an I/O thread dedicated for the hotplug device by using the virsh **iothreadadd** command (see “iothreadadd” on page 222):

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

2. Attach the hotplug device using the virsh **attach-device** command (see “attach-device” on page 201).

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

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

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

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

   <device-configuration-XML-filename>
   Is the name of the device configuration-XML file.

   <IOthread-ID>
   Is the ID of the I/O thread to be added to the virtual server. Be sure that the I/O thread ID matches the I/O thread ID which is used in the hotplug device configuration-XML.

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

Related concepts:

“I/O threads” on page 131
I/O threads are dedicated to perform I/O operations on virtual block devices.

Related tasks:

Chapter 11, “Configuring devices,” on page 69
When you configure storage and network devices, you specify the physical hardware on which the resources are based.

“Displaying the current libvirt-internal configuration” on page 106
The current libvirt-internal configuration is based on the domain configuration-XML file of the defined virtual server, and is enhanced by

libvirt-internal information and the dynamically attached devices.

Detaching a device

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

Before you begin

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

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

Procedure

1. Detach the device using the virsh **detach-device** command (see "detach-device" on page 208):

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

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

   The virsh **iothreadinfo** command displays the I/O threads that are available for a virtual server.

   Use the virsh **iothreaddel** command to remove an I/O thread (see "iothreaddel" on page 224):

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

   <device-configuration-XML-filename>
   Is the name of the device configuration-XML file.

   <IOthread-ID>
   Is the ID of the I/O thread to be deleted from the virtual server.

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

Replacing a virtual DVD

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

Before you begin

Make sure that the virtual DVD drive is configured as a virtual SCSI device (see "Configuring a virtual SCSI-attached CD/DVD drive" on page 86).

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 203):

   ```
   # 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 205):

```
# 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 241):

```
# 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 100
Use the virsh **start** command to start a shut off virtual server.

"Configuring the console" on page 64
Configure the console by using the console element.

Part 5. Best practices and performance considerations

Avoid common pitfalls and tune the virtual server.

Chapter 18. CPU management

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

Related tasks:

"Configuring virtual CPUs" on page 60
Configure virtual CPUs for a virtual server.

"Managing virtual CPUs" on page 116
Modify 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 that are processed 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.

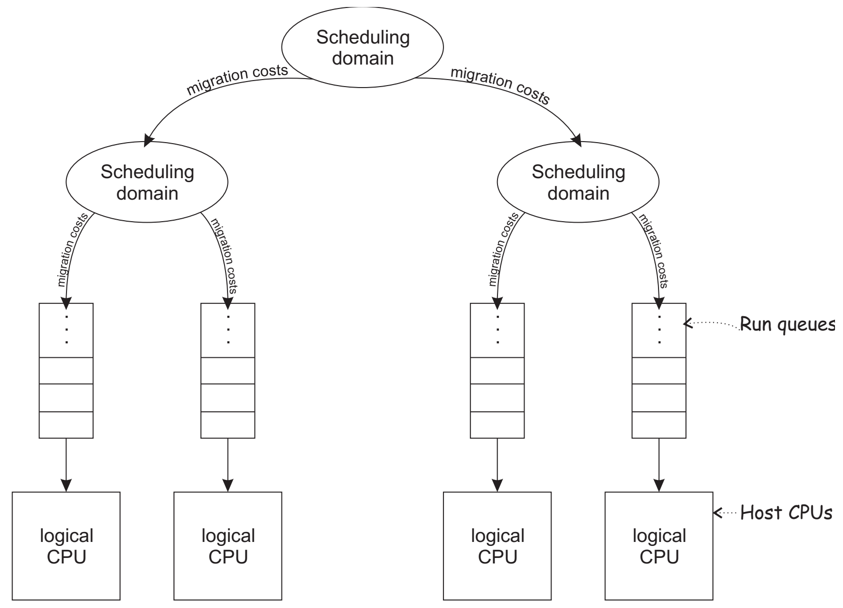


Figure 16. Linux scheduling

libvirt provides means to assign virtual CPUs to groups of host CPUs in order to minimize migration costs. This process is called *CPU pinning*. CPU pinning forces the Linux scheduler to migrate virtual CPUs only between those host CPUs of the specified group. Likewise, the execution of the user space process or I/O threads can be assigned to groups of host CPUs.

Attention: Do not use CPU pinning, because a successful CPU pinning depends on a variety of factors which can change over time:

- CPU pinning can lead to the opposite effect of what was desired when the circumstances for which it was designed change. This may occur, for example, when the host reboots, the workload on the host changes, or the virtual servers are modified.
- Deactivating operating CPUs and activating standby CPUs (CPU hotplug) on the host may lead to a situation where host CPUs are no longer available for the execution of virtual server threads after their reactivation.

CPU weight

The host CPU time which is available for the execution of the virtual CPUs depends on the system utilization.

The available CPU time is divided up between the virtual servers running on the host.

The Linux scheduler and the Linux kernel feature cgroups allocate the upper limit of *CPU time shares* (or simply: *CPU shares*) which a virtual server is allowed to use based on the CPU weight of all virtual servers running on the host.

You can configure the CPU weight of a virtual server, and you can modify it during operation.

The CPU shares of a virtual server are calculated by forming the virtual server's weight-fraction.

Example:

Virtual server	CPU weight	Weight-sum	Weight-fraction	CPU shares
A	1024	3072	1024/3072	1/3
B	2048	3072	2048/3072	2/3

The number of virtual CPUs does not affect the CPU shares of a virtual server.

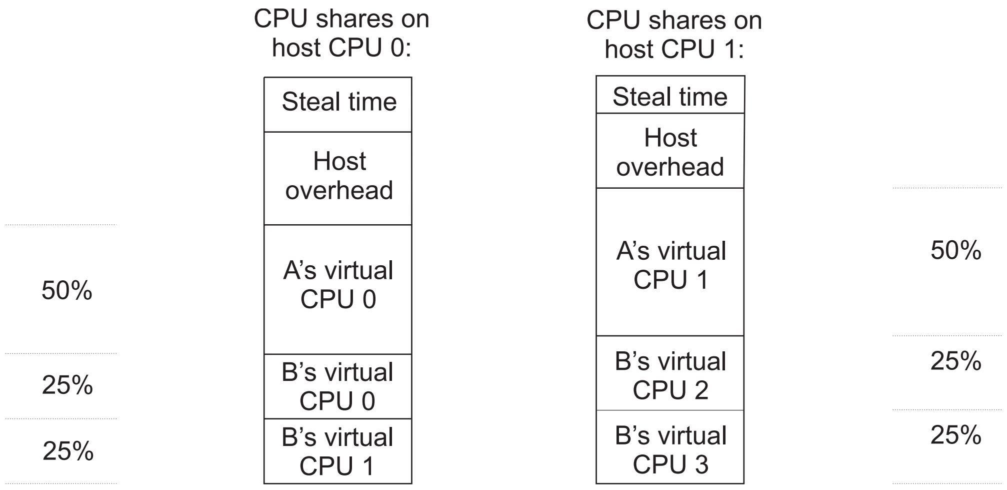
Example:

Virtual server	CPU weight	Number of virtual CPUs
A	1024	2
B	1024	4

The CPU shares are the same for both virtual servers:

Virtual server	CPU weight	Weight-sum	Weight-fraction	CPU shares
A	1024	2048	1024/2048	1/2
B	1024	2048	1024/2048	1/2

The CPU shares of each virtual server are spread across its virtual CPUs, such as:



Chapter 19. Storage management

Consider these aspects when setting up and configuring the virtual server storage.

I/O threads

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

They are used for persistent devices as well as for hotplug devices.

For a good performance of I/O operations, provide one I/O thread for each virtual block device that is used by a virtual server. Estimate no more than one or two I/O threads per host CPU and no more I/O threads than virtual block I/O devices that will be available for the virtual server. Too many I/O threads will reduce system performance by increasing the system overhead.

You can configure I/O threads in the domain configuration-XML of a virtual server. For more information, see:
- "Configuring persistent devices" on page 63
- "Configuring a DASD or SCSI disk" on page 72

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 120
- "Detaching a device" on page 121

Logical volume management

Consider these aspects when the virtual server utilizes logical volumes.

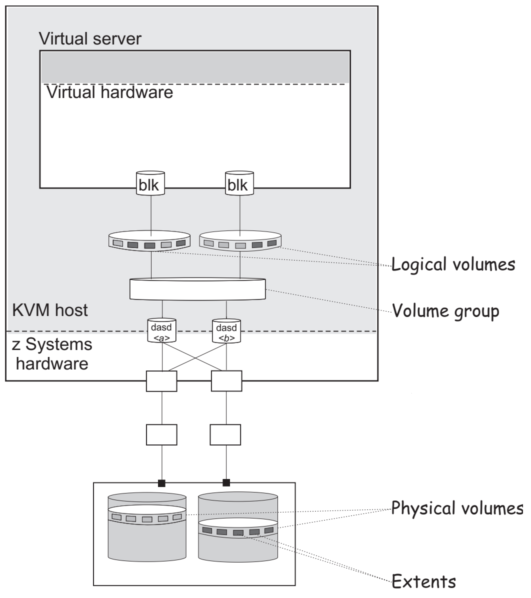
Path redundancy

As discussed in Chapter 2, "DASDs and SCSI disks as virtual block devices," on page 9, it is important to ensure that you provide path redundancy for all physical volumes. Especially, all LVM physical volumes on SCSI disks have to be assembled from device mapper-created device nodes.

Data integrity

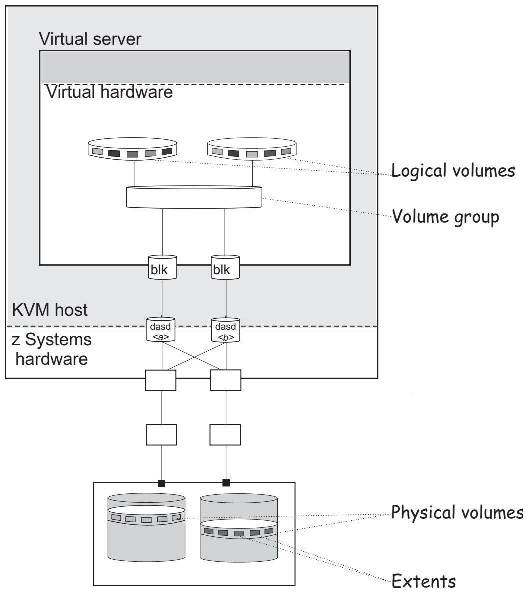
There are two ways to manage logical volumes:
- On the host:

  This example shows multipathed DASDs. The logical volumes that are managed on the host are configured as virtual block devices.



- On the virtual server:

  When you configure physical volumes as virtual block devices, the logical volumes are managed on the virtual server. In this case you need to prohibit a logical volume management of the configured physical volumes on the host. Else, the host might detect the physical volumes and try to manage them on the host, too. Storing host metadata on the physical volumes might cause a loss of virtual server data.



To prohibit a logical volume management for physical volumes that are managed on the virtual server, provide an explicit whitelist in `/etc/lvm/lvm.conf` which explicitly contains all disk block devices to be managed on the host, or a blacklist that contains all physical volumes that are to be managed on the virtual server.

The filter section in the device settings allows to specify a whitelist using the prefix "a", and to specify a blacklist using the prefix "r".

Example

This whitelist in `/etc/lvm/lvm.conf` filters the physical volumes which are to be managed on the host. The last line ("`r|.*|`") denotes that all other physical volumes that are not listed here are not to be managed on the host.

```
devices
       {filter = [ "a|/dev/mapper/36005076305ffc1ae00000000000021d5p1|",
                   "a|/dev/mapper/36005076305ffc1ae00000000000021d7p1|",
                   "a|/dev/disk/by-path/ccw-0.0.1607-part1|",
                   "r|.*|" ]
       }
```

The following physical volumes are to be managed on the host:
- `/dev/mapper/36005076305ffc1ae00000000000021d5p1`
- `/dev/mapper/36005076305ffc1ae00000000000021d7p1`
- `/dev/disk/by-path/ccw-0.0.1607-part1`

You can verify that SCSI disks are referenced correctly by issuing the following **`pvscan`** command:

```
# pvscan -vvv 2>&1 | fgrep '/dev/sd'
...
   /dev/sda: Added to device cache
   /dev/block/8:0: Aliased to /dev/sda in device cache
   /dev/disk/by-path/ccw-0.0.50c0-zfcp-0x1234123412341234:\
     0x0001000000000000: Aliased to /dev/sda in device cache
   ...
   /dev/sda: Skipping (regex)
```

The output must contain the string "Skipping (regex)" for each SCSI disk standard device name which is configured for the virtual server.

Related publications

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

Part 6. Diagnostics and troubleshooting

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

Chapter 20. Logging

Adapt the logging facility to your needs.

Log messages

These logs are created.

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

`/var/log/libvirt/qemu/<`*`VS`*`>.log`
: QEMU log file of the specified virtual server.

: <*VS*> is the name of the virtual server.

Specifying the logging level

Specify the level of logging information that is displayed in the log 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 21. Dumping

Create dumps of a crashed virtual server on the host or on the virtual server.

Creating a virtual server dump on the host

When the virtual server is crashed, you can create a dump on the host.

Procedure

Create a dump of the crashed virtual server using the virsh **dump** command with the `--memory-only` option:

```
# virsh dump --memory-only <VS> <dumpfile>
```

<dumpfile>
Is the name of the dump file. If no fully qualified path to the dump file is specified, it is written to the current working directory of the user who issues the virsh **dump** command.

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

Results

The dump is written to the file *<dumpfile>*.

What to do next

To inspect the dump, enter the command:

```
# crash <dumpfile> <kernel-image-filename>
```

<kernel-image-filename>
Is the name of the kernel image file of the guest running on the dumped virtual server.

Creating a dump on the virtual server

When a virtual server is crashed, you can provide a dump for the virtual server user.

Before you begin

Ensure that kdump is installed and enabled on the virtual server.

If kdump is not enabled on the virtual server, the following procedure causes only a restart of the virtual server.

| For more information about kdump, see *Using the Dump Tools*, SC33-8412.

Procedure

- In case of a virtual server kernel panic, a dump is automatically created.
- In case of a non-responding virtual server, you can trigger a restart interrupt.

  The interrupt handling of a restart interrupt depends on the PSW restart configuration and ends up in a dump.

  To trigger a restart interrupt, use the virsh **inject-nmi** command:

  ```
  # virsh inject-nmi <VS>
  ```

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

Results

The virtual server creates a dump and then restarts in kdump mode.

What to do next

To verify your action, you might want to see the dump on the virtual server:

1. Log in to the virtual server as root.
2. Use the **makedumpfile** command to create a dump file from the `vmcore` file:

   ```
   [root@guest:] # makedumpfile -c <vmcore> <dumpfile>
   ```

3. To inspect the dump, enter:

   ```
   [root@guest:] # crash <dumpfile> <kernel-image-filename>
   ```

   The **`crash`** command is available with the kernel-debuginfo package.

   <dumpfile>
   Is the fully qualified path and file name of the dump file.

   <kernel-image-filename>
   Is the name of the kernel image file of the guest running on the dumped virtual server.

   <vmcore>
   Is the fully qualified path and file name of the `vmcore` file of the guest.

Chapter 22. Collecting performance metrics

You can monitor virtual server machine code instructions.

Before you begin

- Make sure that your kernel is built using the common source options `CONFIG_TRACEPOINTS`, `CONFIG_HAVE_PERF_EVENTS`, and `CONFIG_PERF_EVENTS`.
- Make sure that the **perf** tool is installed.

  You can check this by issuing:

  ```
  # perf list
  ...
  kvm:kvm_s390_sie_enter                              [Tracepoint event]
  ...
  ```

  If the command returns a list of supported events, such as the tracepoint event `kvm_s390_sie_enter`, the tool is installed.

Procedure

You collect, record, and display performance metrics with the **perf kvm stat** command.

- The **record** subcommand records performance metrics and stores them in the file `perf.data.guest`.
  - The **perf** tool records events until you terminate it by pressing Control and c (Ctrl+c).
  - To display the recorded data, use the **report** subcommand.
  - It is recommended to save `perf.data.guest` before you collect new statistics, because a new record may overwrite this file.
- The **live** subcommand displays the current statistics without saving them.

  The **perf** tool displays events until you terminate it by pressing Control and c (Ctrl+c).

Example

```
# ./perf kvm stat record -a
^C[ perf record: Woken up 7 times to write data ]
[ perf record: Captured and wrote 13.808 MB perf.data.guest (~603264 samples) ]

# ./perf kvm stat report


Analyze events for all VMs, all VCPUs:

                                 VM-EXIT    Samples  Samples%     Time%    Min Time    Max Time        Avg time

                       Host interruption      14999    35.39%     0.39%      0.45us   1734.88us      0.82us ( +-  19.59% )
              DIAG (0x44) time slice end      13036    30.76%     0.57%      1.06us   1776.08us      1.39us ( +-   9.81% )
       DIAG (0x500) KVM virtio functions      13011    30.70%     1.90%      1.15us   2144.75us      4.65us ( +-   5.08% )
                              0xE5 TPROT        512     1.21%     0.01%      0.79us      2.18us      0.83us ( +-   0.42% )
                               0xB2 TSCH        406     0.96%     0.19%      7.35us    109.43us     14.95us ( +-   2.97% )
                              0xB2 SERVC        117     0.28%     0.15%     10.97us    339.00us     40.46us ( +-   9.17% )
                        External request        113     0.27%     0.01%      0.75us      2.58us      1.56us ( +-   1.55% )
                              0xB2 STSCH         57     0.13%     0.02%      7.30us     26.40us      9.47us ( +-   5.99% )
                              Wait state         40     0.09%    96.48%   3334.30us 464600.00us  76655.28us ( +-  32.97% )
                               0xB2 MSCH         14     0.03%     0.00%      7.22us      9.19us      7.74us ( +-   2.13% )
                               0xB2 SSCH         14     0.03%     0.01%      8.67us     35.41us     16.16us ( +-  16.38% )
                               0xB2 CHSC         10     0.02%     0.00%      7.51us     22.90us     11.06us ( +-  15.20% )
                             I/O request          8     0.02%     0.00%      1.37us      1.97us      1.55us ( +-   5.77% )
                               0xB2 STPX          8     0.02%     0.00%      1.04us      7.10us      1.98us ( +-  37.25% )
                               0xB2 STSI          7     0.02%     0.00%      1.65us     62.09us     22.26us ( +-  41.95% )
                              0xB2 STIDP          4     0.01%     0.00%      1.12us      3.62us      2.62us ( +-  21.07% )
                   SIGP set architecture          3     0.01%     0.00%      1.05us      2.68us      1.60us ( +-  33.74% )
                               0xB2 STAP          3     0.01%     0.00%      1.05us      7.61us      3.39us ( +-  62.25% )
                               0xB2 STFL          3     0.01%     0.00%      1.78us      3.88us      2.84us ( +-  21.31% )
    DIAG (0x204) logical-cpu utilization          2     0.00%     0.00%      4.58us     39.48us     22.03us ( +-  79.19% )
              DIAG (0x308) ipl functions          2     0.00%     0.01%     19.34us    329.25us    174.30us ( +-  88.90% )
     DIAG (0x9c) time slice end directed          1     0.00%     0.00%      1.09us      1.09us      1.09us ( +-   0.00% )
                                0xB2 SPX          1     0.00%     0.00%      4.58us      4.58us      4.58us ( +-   0.00% )
                               0xB2 SETR          1     0.00%     0.00%     56.97us     56.97us     56.97us ( +-   0.00% )
                               0xB2 SSKE          1     0.00%     0.25%   7957.94us   7957.94us   7957.94us ( +-   0.00% )
                              0xB2 STCRW          1     0.00%     0.00%     11.24us     11.24us     11.24us ( +-   0.00% )
    DIAG (0x258) page-reference services          1     0.00%     0.00%      4.87us      4.87us      4.87us ( +-   0.00% )
                               0xB9 ESSA          1     0.00%     0.00%      8.72us      8.72us      8.72us ( +-   0.00% )
                              0xEB LCTLG          1     0.00%     0.00%      9.27us      9.27us      9.27us ( +-   0.00% )

Total Samples:42377, Total events handled time:3178166.35us.
```

What to do next

For more information about the **perf** subcommand **kvm stat**, see the man page or issue the full subcommand with the `--help` option:

```
►►──perf──kvm──stat──┬─record─┬── --help─────────────────────────────────►◄
                     ├─report─┤
                     └─live───┘
```

With the collected statistics, you can watch the virtual server behavior and time consumption and then analyze the recorded events. So you may find hints for possible sources of error.

- You can find a description of the general instructions in the *z/Architecture® Principles of Operation*, SA22-7832, for example:

Mnemonic	Instruction	Opcode
TPROT	TEST PROTECTION	E501
TSCH	TEST SUBCHANNEL	B235

- Signal-processor orders (SIGP) are also described in the *z/Architecture Principles of Operation*, SA22-7832.
- Table 1 on page 143 lists all diagnoses (DIAG) as supported by KVM on z Systems.

Table 1. Supported Linux diagnoses

Number	Description	Linux use	Required/ Optional
0x010	Release pages	CMM	Required
0x044	Voluntary time-slice end	In the kernel for spinlock and udelay	Required
0x09c	Voluntary time slice yield	Spinlock	Optional
0x258	Page-reference services	In the kernel, for pfault	Optional
0x288	Virtual server time bomb	The watchdog device driver	Required
0x308	Re-ipl	Re-ipl and dump code	Required
0x500	Virtio functions	Operate virtio-ccw devices	Required

Required means that a function is not available without the diagnose; optional means that the function is available but there might be a performance impact.

You may also find other DIAG events on your list, but those are not supported by KVM on z Systems. A list of all Linux diagnoses is provided in *Device Drivers, Features, and Commands*, SC33-8411.

Part 7. Reference

Virtual server states, a selection of elements to configure a virtual server, and a selection of commands to operate a virtual server.

Chapter 23. Virtual server life cycle

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

Figure 17 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.

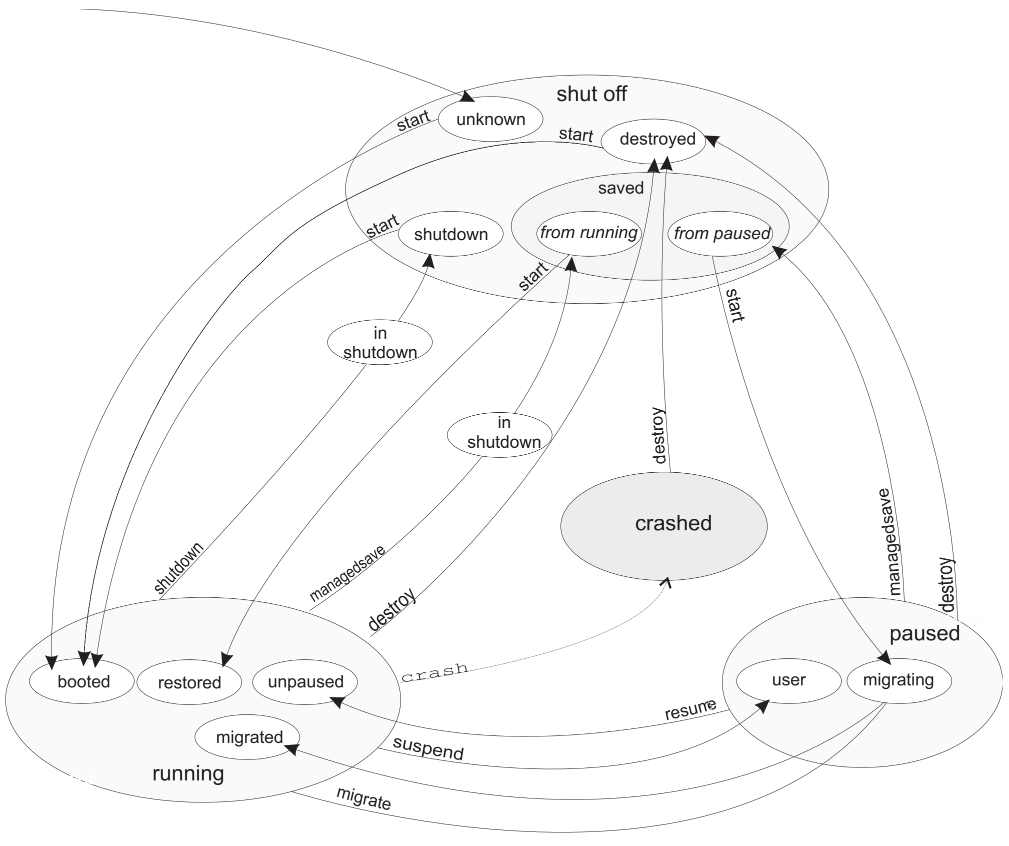


Figure 17. State-transition diagram of a virtual server including reasons

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 24. Selected libvirt XML elements

These libvirt XML elements might be useful for you. You find the complete libvirt XML reference at libvirt.org.

- <adapter>
- <address>
- <boot>
- <cipher>
- <cmdline>
- <console>
- <controller>
- <cputune>
- <devices>
- <disk>
- <domain>
- <driver>
- <emulator>
- <geometry>
- <hostdev>
- <initrd>
- <interface>
- <iothreads>
- <kernel>
- <keywrap>
- <mac>
- <memballoon>
- <memory>
- <name>
- <on_crash>
- <on_reboot>
- <os>
- <readonly>
- <shareable>
- <shares>
- <source>
- <target>
- <type>
- <vcpu>
- <virtualport>
- <watchdog>

<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 81

Parent elements

"<source> as child element of <hostdev>" on page 190.

Child elements

None.

Example

```
<devices>
   ...
   <controller type="scsi" model="virtio-scsi" index="0"/>
   <hostdev mode="subsystem" type="scsi">
      <source>
         <adapter name="scsi_host0"/>
         <address bus="0" target="0" unit="0"/>
      </source>
      <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
   ...
</devices>
```

<address> as child element of <controller>, <disk>, <interface>, and <memballoon>

Specifies the address of a device on the virtual server.

Text content

None.

Selected attributes

type=ccw
Specifies a virtio CCW device, such as a block device or a network device.

You can specify the device bus-ID with the address attributes cssid, ssid, and devno.

cssid Specifies the channel subsystem number of the virtual device. Must be "0xfe".

ssid Specifies the subchannel set of the virtual device. Valid values are between "0x0" and "0x3".

devno Specifies the device number of the virtio device. Must be a unique value between "0x0000" and "0xffff".

Usage

- "Configuring a DASD or SCSI disk" on page 72
- "Configuring a file as storage device" on page 78

Parent elements

- "<controller>" on page 162
- "<disk>" on page 165
- "<interface>" on page 173
- "<memballoon>" on page 178

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vda" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0x1108"/>
</disk>
```

<address> as child element of <hostdev> or <disk>

Specifies the address of a device, which is connected to the virtual server through a controller.

Text content

None.

Selected attributes

type=scsi
Specifies a SCSI device.

controller
Specifies the virtual controller of the virtual device. Enter the index attribute value of the respective controller element.

bus Specifies the virtual SCSI bus of the virtual device.

target Specifies the virtual SCSI target of the virtual device. This value can be between 0 and 255.

unit Specifies the unit number (LUN) of the virtual SCSI device.

Usage

- "Configuring a SCSI tape or medium changer device" on page 81
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 86

Parent elements

- "<hostdev>" on page 171
- "<disk>" on page 165

Child elements

None.

Example

```
<devices>
    ...
    <controller type="scsi" model="virtio-scsi" index="0"/>
    <hostdev mode="subsystem" type="scsi">
        <source>
            <adapter name="scsi_host0"/>
            <address bus="0" target="0" unit="0"/>
        </source>
        <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
    </hostdev>
    ...
    <controller type="scsi" model="virtio-scsi" index="1"/>
    <disk type="file" device="cdrom">
        <driver name="qemu" type="raw" io="native" cache="none"/>
        <source file="/var/lib/libvirt/images/cd.iso"/>
        <target dev="vda" bus="scsi"/>
        <address type="drive" controller="1" bus="0" target="0" unit="0"/>
        <readonly/>
    </disk>
    ...
</devices>
```

<address> as child element of <source>

Specifies a device address from the host point of view.

Text content

None.

Selected attributes

bus=0 For a SCSI device the value is zero.

target Specifies the SCSI ID.

unit Specifies the SCSI LUN.

Usage

"Configuring a SCSI tape or medium changer device" on page 81

Parent elements

"<source> as child element of <hostdev>" on page 190

Child elements

None.

Example

```
<devices>
   ...
   <controller type="scsi" model="virtio-scsi" index="0"/>
   <hostdev mode="subsystem" type="scsi">
      <source>
         <adapter name="scsi_host0"/>
         <address bus="0" target="0" unit="0"/>
      </source>
      <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
   </hostdev>
   ...
</devices>
```

<boot>

Specifies that the virtual block device is bootable.

Text content

None.

Selected attributes

order=*number*
: Specifies the order in which a device is considered as boot device during the boot sequence.

Usage

"Configuring the boot process" on page 55

Parent elements

"<disk>" on page 165

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"/>
    <boot order="1"/>
</disk>
```

<cipher>

Configures the generation of an AES or DEA/TDEA wrapping key and the use of the respective protected key management operations on the virtual server.

Text content

None.

Selected attributes

name=aes | dea
Specifies the AES or DEA/TDEA wrapping key.

state=on | off

on Enables wrapping key generation.

The respective protected key management operations are available on the virtual server.

off Disables wrapping key generation.

The respective protected key management operations are not available on the virtual server.

Usage

"Disabling protected key encryption" on page 66

Parent elements

"<keywrap>" on page 176

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 56

Parent elements

"<os>" on page 185

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 64

Parent elements

"<devices>" on page 164

Child elements

- <protocol>
- "<target> as child element of <console>" on page 192

Example

```
<console type="pty">
   <target type="sclp" port="0"/>
</console>
```

<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 81

Parent elements

"<devices>" on page 164

Child elements

None.

Example

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

<cputune>

Groups CPU tuning parameters.

Text content

None.

Selected attributes

None.

Usage

"Tuning virtual CPUs" on page 60

Parent elements

"<domain>" on page 166

Child elements

"<shares>" on page 188

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 11, “Configuring devices,” on page 69

Parent elements

“<domain>” on page 166

Child elements

- “<console>” on page 161
- “<controller>” on page 162
- “<disk>” on page 165
- “<emulator>” on page 169
- “<hostdev>” on page 171
- “<interface>” on page 173
- “<memballoon>” on page 178
- “<watchdog>” on page 197

Example

```
<devices>
    <interface type="direct">
        <source dev="enccw0.0.1108" mode="bridge"/>
        <model type="virtio"/>
    </interface>

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

<disk>

Specifies a virtual block device, such as a SCSI device, or a file.

Text content

None.

Selected attributes

type=block | file
Specifies the underlying disk source.

device=disk | cdrom
Optional; Indicates how the virtual block device is to be presented to the virtual server.

Usage

- Chapter 11, "Configuring devices," on page 69
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 86

Parent elements

"<devices>" on page 164

Child elements

- "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 155
- <blockio>
- "<boot>" on page 158
- "<driver> as child element of <disk>" on page 167
- "<geometry>" on page 170
- "<readonly>" on page 186
- "<shareable>" on page 187
- "<source> as child element of <disk>" on page 189
- "<target> as child element of <disk>" on page 193

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

None.

Attributes

type=kvm
Specifies the virtual server type.

Usage

"Domain configuration-XML" on page 53

Parent elements

None.

Child elements

- <clock>
- "<console>" on page 161
- "<controller>" on page 162
- "<cputune>" on page 163
- <currentMemory>
- "<devices>" on page 164
- "<iothreads>" on page 174
- <memory>
- <name>
- "<on_crash>" on page 183
- <on_poweroff>
- <on_reboot>
- <os>
- <uuid>
- "<vcpu>" on page 195

<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!

<driver> as child element of <disk>

Usage

- "Configuring a DASD or SCSI disk" on page 72
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 86

Parent elements

"<disk>" on page 165

Child elements

None.

Example

```
<disk type="block" device="disk">
    <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
    <source dev="/dev/mapper/36005076305ffc1ae00000000000021d5"/>
    <target dev="vdb" bus="virtio"/>
    <address type="ccw" cssid="0xfe" ssid="0x0" devno="0xd501"/>
</disk>
```

<emulator>

Specifies the user space process.

Text content

Fully qualified path and file name of the user space process.

Selected attributes

None.

Usage

- "Configuring the user space" on page 62
- "Displaying the current libvirt-internal configuration" on page 106

Parent elements

"<devices>" on page 164

Child elements

None.

Example

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

<geometry>

Overrides the geometry settings of DASDs or FC-attached SCSI disks.

Text content

None.

Selected attributes

cyls	Specifies the number of cylinders.
heads	Specifies the number of heads.
secs	Specifies the number of sectors per track.

Usage

"Configuring a DASD or SCSI disk" on page 72

Parent elements

"<disk>" on page 165

Child elements

None.

Example

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

<hostdev>

Passes host-attached devices to a virtual server.

Ensure that the device that is passed through to the virtual server is not in use by the host.

Text content

None.

Selected attributes

mode=subsystem
Specifies the pass-through mode.

type=scsi
Specifies the type of device that is assigned to a virtual server.

rawio=no | yes
Indicates whether the device needs raw I/O capability. If any device in a device configuration-XML file is specified in raw I/O mode, this capability is enabled for all such devices of the virtual server.

sgio=filtered | unfiltered
Indicates whether the kernel will filter unprivileged SG_IO commands for the device.

Usage

“Configuring a SCSI tape or medium changer device” on page 81

Parent elements

“<devices>” on page 164

Child elements

- “<address> as child element of <hostdev> or <disk>” on page 156
- “<readonly>” on page 186
- “<shareable>” on page 187
- “<source> as child element of <hostdev>” on page 190

Example

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

<initrd>

Specifies the fully qualified path of the ramdisk image in the host operating system.

Text content

Fully qualified path and file name of the initial ramdisk.

Selected attributes

None.

Usage

"Configuring a kernel image file as IPL device" on page 56

Parent elements

"<os>" on page 185

Child elements

None.

Example

```
<os>
    <type arch='s390x' machine='s390-virtio'>hvm</type>
    <kernel>/boot/vmlinuz-3.1.0-7.fc16.s390x</kernel>
    <initrd>/boot/initramfs-3.1.0-7.fc16.s390x.img</initrd>
    <cmdline>printk.time=1</cmdline>
</os>
```

<interface>

Specifies a virtual Ethernet device for a virtual server.

Text content

None.

Selected attributes

type = direct | bridge

Specifies the type of connection:

direct Creates a MacVTap interface.

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

trustGuestRxFilters = no | yes

| Only valid if type = "direct".

| Set this attribute to "yes" to allow the virtual server to change its MAC
| address. As a consequence, the virtual server can join multicast groups.

| 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 89

Parent elements

"<devices>" on page 164

Child elements

- "<address> as child element of <controller>, <disk>, <interface>, and <memballoon>" on page 155
- "<mac>" on page 177
- "<model>" on page 181
- "<source> as child element of <interface>" on page 191
- "<virtualport>" on page 196

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 persistent devices" on page 63

Parent elements

"<domain>" on page 166

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 56

Parent elements

"<os>" on page 185

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 66

Parent elements

"<domain>" on page 166

Child elements

"<cipher>" on page 159

Example

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

<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 89

Parent elements

"<interface>" on page 173

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 68

Parent elements

"<devices>" on page 164

Child elements

None.

Example

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

<memory>

Specifies the amount of memory allocated for a virtual server at boot time and configures the collection of QEMU core dumps.

Text content

Natural number specifying the amount of memory. The unit is specified with the unit attribute.

Selected attributes

dumpCore=on | off
Specifies whether the memory of a virtual server is included in a generated core dump.

on	Specifies that the virtual server memory is included.
off	Specifies that the virtual server memory is excluded.

unit=b | KB | k | KiB | MB | M | MiB | GB | G | GiB | TB | T | TiB
Specifies the units of memory used:

b	bytes
KB	kilobytes (1,000 bytes)
k or KiB	kibibytes (1024 bytes), the default
MB	megabytes (1,000,000 bytes)
M or MiB	mebibytes (1,048,576 bytes)
GB	gigabytes (1,000,000,000 bytes)
G or GiB	gibibytes (1,073,741,824 bytes)
TB	terabytes (1,000,000,000,000 bytes)
T or TiB	tebibytes (1,099,511,627,776 bytes

Usage

- "Configuring virtual memory" on page 61
- "Configuring the collection of QEMU core dumps" on page 61

Parent elements

"<domain>" on page 166

Child elements

None.

Example

This example:
- Configures 524,288 KB of virtual memory.

- Excludes the virtual memory from QEMU core dumps.

```
<memory dumpCore="off" unit="KB">524288</memory>
```

<model>

Specifies the interface model type.

Text content

None.

Selected attributes

type=virtio
Specifies the interface model type virtio.

Usage

- “Configuring a MacVTap interface” on page 89
- “Configuring a virtual switch” on page 91

Parent elements

“<interface>” on page 173

Child elements

None.

Example

This example configures a virtio interface:

```
<interface type="direct">
    <source dev="enccw0.0.a100" mode="bridge"/>
    <model type="virtio"/>
</interface>
```

<name>

Assigns a unique name to the virtual server.

Text content

Unique alphanumeric name for the virtual server.

Selected attributes

None.

Usage

"Domain configuration-XML" on page 53

Parent elements

"<domain>" on page 166

Child elements

None.

Example

```
<name>Virtual server 25</name>
```

<on_crash>

Configures the behavior of the virtual server in the crashed state.

Set to preserve to ensure that virtual server crashes are detected.

Text content

preserve
Preserves the crashed state.

Selected attributes

None.

Usage

“Domain configuration-XML” on page 53

Parent elements

“<domain>” on page 166

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 237.

Text content

restart Terminates the virtual server using the **`shutdown`** command and then starts it again.

destroy
Terminates the virtual server using the **`destroy`** command and then starts it again.

Selected attributes

None.

Usage

"Domain configuration-XML" on page 53

Parent elements

"<domain>" on page 166

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 53

Parent elements

"<domain>" on page 166

Child elements

- "<type>" on page 194
- "<kernel>" on page 175
- "<initrd>" on page 172
- "<cmdline>" on page 160

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 86

Parent elements

- "<disk>" on page 165
- "<hostdev>" on page 171

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>
```

<shareable>

Indicates that a device can be shared between various virtual servers.

Text content

None.

Selected attributes

None.

Parent elements

- “<disk>” on page 165
- “<hostdev>” on page 171

Child elements

None.

Example

```
<devices>
    <controller type="scsi" model="virtio-scsi" index="0"/>
    <hostdev mode="subsystem" type="scsi">
        <source>
            <adapter name="scsi_host0"/>
            <address bus="0" target="0" unit="0"/>
        </source>
        <address type="scsi" controller="0" bus="0" target="0" unit="0"/>
        <shareable/>
    </hostdev>
</devices>
```

<shares>

Specifies the initial CPU weight.

The CPU shares of a virtual server are calculated from the CPU weight of all virtual servers running on the host. For example, a virtual server that is configured with value 2048 gets twice as much CPU time as a virtual server that is configured with value 1024.

Text content

Natural number specifying the CPU weight.
- Valid values are in the natural numbers between 2 and 262144.
- The default value is 1024.

Selected attributes

None.

Usage

"Tuning virtual CPUs" on page 60

Parent elements

"<cputune>" on page 163

Child elements

None.

Example

```
<cputune>
   <shares>2048</shares>
</cputune>
```

<source> as child element of <disk>

Specifies the host view of a device configuration.

Text content

None.

Selected attributes

file
Must be specified for disk type="file". Specifies the fully qualified host file name.

dev
Must be specified for disk type="block". Specifies a host device node of the block device.

startupPolicy=mandatory | requisite | optional
For disk type file that represents a CD or diskette, you may define a policy what to do with the disk if the source file is not accessible:

mandatory
fail if missing for any reason

requisite
fail if missing on boot up, drop if missing on migrate/restore/revert

optional
drop if missing at any start attempt

Usage

- "Configuring a DASD or SCSI disk" on page 72
- "Configuring a file as storage device" on page 78
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 86

Parent elements

"<disk>" on page 165

See also:

- "<source> as child element of <interface>" on page 191

Child elements

<seclabel>

Examples

```
<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>
```

```
<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>
```

<source> as child element of <hostdev>

Specifies the host view of a host device configuration.

Text content

None.

Selected attributes

None.

Usage

"Configuring a SCSI tape or medium changer device" on page 81

Parent elements

"<hostdev>" on page 171

Child elements

- "<address> as child element of <source>" on page 157
- "<adapter> as child element of <source>" on page 154

Example

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

<source> as child element of <interface>

Specifies the host view of a network interface configuration.

Text content

None.

Selected attributes

dev Specifies the network interface.

mode=bridge | vepa
Optional; indicates whether packets are delivered to the target device or to the external bridge.

bridge If packets have a destination on the host from which they originated, they are delivered directly to the target. For direct delivery, both origin and destination devices need to be in bridge mode. If either the origin or destination is in vepa mode, a VEPA-capable bridge is required.

vepa All packets are sent to the external bridge. If packets have a destination on the host from which they originated, the VEPA-capable bridge will return the packets to the host.

Usage

"Configuring virtual Ethernet devices" on page 89

Parent elements

"<interface>" on page 173

Child elements

None.

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 64

Parent elements

"<console>" on page 161

See also:
- "<target> as child element of <disk>" on page 193

Child elements

None.

Example

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

<target> as child element of <disk>

Specifies the virtual server view of a device that is provided from the host.

Text content

None.

Selected attributes

dev
Unique name for the device of the form vd<*x*>, where <*x*> can be one or more letters.

If no address element is specified, the order in which device bus-IDs are assigned to virtio block devices is determined by the order of the target dev attributes.

bus=virtio
Specifies the device type on the virtual server. Specify "virtio".

Usage

- "Configuring a DASD or SCSI disk" on page 72
- "Configuring a file as storage device" on page 78
- "Configuring a virtual SCSI-attached CD/DVD drive" on page 86

Parent elements

"<disk>" on page 165

See also: "<target> as child element of <console>" on page 192

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>

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 53
- Chapter 5, "Migration to a different hypervisor release," on page 23

Parent elements

"<os>" on page 185

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 number of available virtual CPUs.

Selected attributes

current
Not supported.

Usage

"Configuring virtual CPUs" on page 60

Parent elements

"<domain>" on page 166

Child elements

None.

Example

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

<virtualport>

Specifies the type of a virtual switch.

Text content

None.

Selected attributes

type=openvswitch
Specifies the type of the virtual switch.

Usage

- "Configuring a virtual switch" on page 91

Parent elements

"<interface>" on page 173

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=reset | poweroff | pause | dump | inject-nmi | none | shutdown
Optional; specifies an action that is automatically performed when the watchdog timer expires:

reset Default; immediately terminates the virtual server and restarts it afterwards.

poweroff
Immediately terminates the virtual server.

pause Suspends the virtual server.

dump Creates a virtual server dump on the host.

inject-nmi
Causes a restart interrupt for the virtual server including a dump on the virtual server, if it is configured respectively.

none Does not perform any command.

shutdown
Tries to properly shut down the virtual server.

Since the usage of this action assumes that the virtual server is not responding, it is unlikely that the virtual server will respond to the shutdown command. It is recommended not to use this action.

Usage

"Configuring a watchdog device" on page 65

Parent elements

"<devices>" on page 164

Child elements

None.

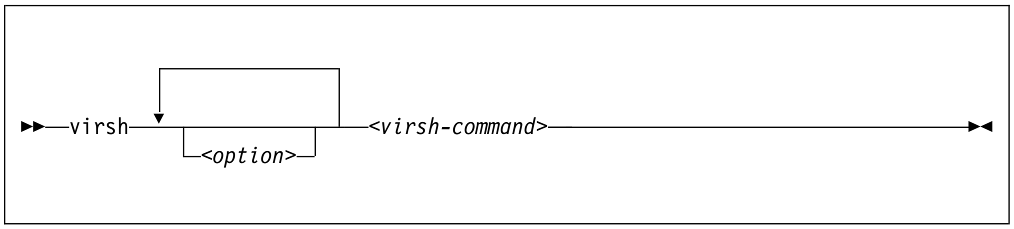
Example

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

Chapter 25. Selected virsh commands

These virsh commands might be useful for you. They are described with a subset of options that are valuable in this context.

Syntax



Where:

<option>
: Is a command option.

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

<virsh-command>
: Is a virsh command.

For a complete list of the virsh commands, see libvirt.org/virshcmdref.html.

<XML-filename>
: Is the name of the XML file, which defines the device to be attached to the running virtual server.

Selected options

--help Displays the virsh online help.

--keepalive-interval ***<interval-in-seconds>***
: Sets an interval for sending keepalive messages to the virtual server to confirm the connection between the host and the virtual server. If the virtual server does not answer for a number of times which is defined by the `--keepalive-count` option, the host closes the connection. Setting the interval to 0 disables this mechanism. The default is 5 seconds.

--keepalive-count ***<keepalive-count>***
: Sets the number of times keepalive message can be sent without getting an answer from the virtual server without closing the connection. If the keepalive interval is set to 0, this option has no effect. The default is 6.

--version
: Displays the installed libvirt version.

Selected virsh commands

These virsh commands are listed in the following chapters:

- “attach-device” on page 201
- “change-media” on page 203

- "console" on page 205
- "define" on page 206
- "destroy" on page 207
- "detach-device" on page 208
- "domblklist" on page 210
- "domblkstat" on page 211
- "domiflist" on page 213
- "domifstat" on page 214
- "dominfo" on page 215
- "domjobabort" on page 216
- "domstate" on page 217
- "dump" on page 218
- "dumpxml" on page 219
- "edit" on page 220
- "inject-nmi" on page 221
- "iothreadadd" on page 222
- "iothreaddel" on page 224
- "iothreadinfo" on page 226
- "list" on page 227
- "managedsave" on page 229
- "migrate" on page 231
- "migrate-getspeed" on page 234
- "migrate-setmaxdowntime" on page 235
- "migrate-setspeed" on page 236
- "reboot" on page 237
- "resume" on page 238
- "schedinfo" on page 239
- "shutdown" on page 240
- "start" on page 241
- "suspend" on page 243
- "undefine" on page 244
- "vcpucount" on page 245

Example

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

```
# virsh help migrate
```

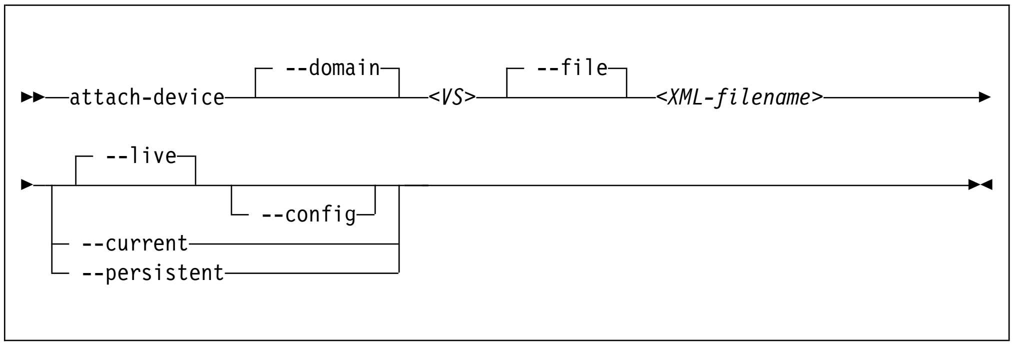
This example increases the keepalive interval of the connection to the host to 10 seconds during a live migration:

```
# virsh --keepalive-interval 10 migrate --live --persistent --undefinesource \
--timeout 1200 --verbose vserv1 qemu+ssh://kvmhost/system
```

attach-device

Attaches a device to a defined virtual server.

Syntax



Where:

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

<*XML-filename*>
Is the name of the XML file, which defines the device to be attached to the running virtual server.

Selected options

--config
Persistently attaches the device to the virtual server with the next restart.

--current
Depending on the virtual server state:

running, paused
Attaches the device to the virtual server until it is detached or the virtual server is terminated.

shut off
Persistently attaches the device to the virtual server with the next restart.

--domain
Specifies the virtual server.

--file Specifies the device configuration-XML file.

--live Attaches the device to the running virtual server until it is detached or the virtual server is terminated.

--persistent
Depending on the virtual server state:

running, paused
Attaches the device to the virtual server.

The device remains persistently attached across restarts.

shut off
Persistently attaches the device to the virtual server with the next restart.

Usage

"Attaching a device" on page 120

Example

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

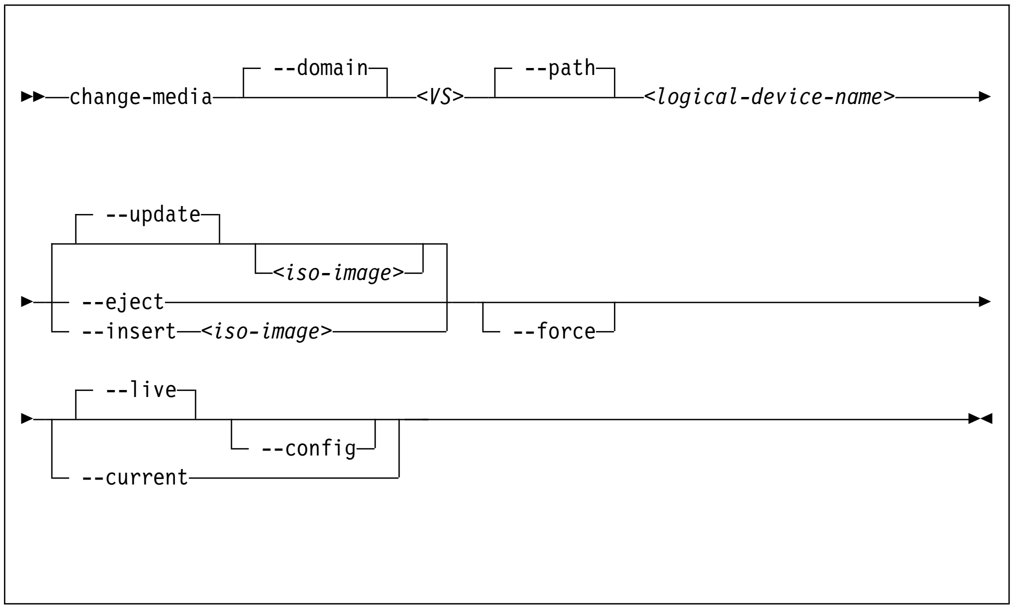
```
# virsh attach-device vserv1 dev1.xml
```

See also the example on page 107.

change-media

Removes a currently provided ISO image from a virtual SCSI-attached CD/DVD drive, or provides a different ISO image.

Syntax



Where:

<logical-device-name>
: Identifies the virtual SCSI-attached CD/DVD drive as specified with the target dev attribute in the domain configuration-XML file.

<iso-image>
: Is the fully qualified path to the ISO image on the host.

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

Selected options

--config
: Persistently adds or removes the ISO image with the next virtual server restart.

--current
: Depending on the virtual server state:

  running, paused
  : Adds or removes the ISO image until the virtual server is terminated.

  shut off
  : Persistently removes the ISO image from the virtual server or provides a different one with the next restart.

--domain
: Specifies the virtual server.

--eject Removes the currently provided ISO image from the virtual SCSI-attached CD/DVD drive.

--force Forces the guest to release the file system residing on the virtual DVD, even if it is currently in use.

--insert
Provides a different ISO image for the virtual server.

--live Removes an ISO image from the running virtual server or provides an ISO image for a running virtual server until the virtual server is terminated.

--path Specifies the virtual SCSI-attached CD/DVD drive.

--update

If no ISO image is specified:
Removes the currently provided ISO image, just like the `--eject` option.

If an ISO image is specified:
Provides the specified ISO image. In case the current disk image has not been removed before, it is replaced by the new one.

Usage

"Replacing a virtual DVD" on page 121

Example

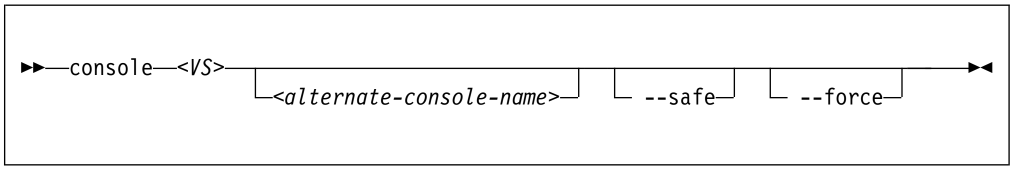
This command replaces the currently provided virtual DVD by a different one:

```
# virsh change-media vserv1 vdc -update /var/lib/libvirt/images/cd2.iso
Successfully inserted media.
```

console

Displays the console of a virtual server.

Syntax



Where:

<alternate-console-name>
: Is the device alias name of an alternative console that is configured for the virtual server.

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

Selected options

--force Disconnects any session in a case the connection is disrupted.

--safe Only connects to the console if the host ensures exclusive access to the console.

Usage

"Connecting to the console of a virtual server" on page 123

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 96

Example

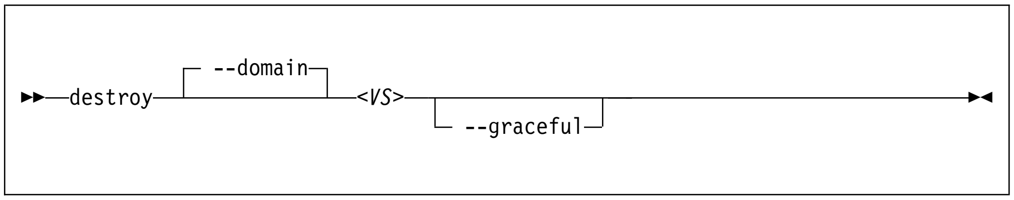
This example defines the virtual server, which is configured in domain configuration-XML file `vserv1.xml`.

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

destroy

Immediately terminates a virtual server and releases any used resources.

Syntax



Where:

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

Selected options

--domain
Specifies the virtual server.

--graceful
Tries to properly terminate the virtual server, and only if it is not responding in a reasonable amount of time, it is forcefully terminated.

Virtual server state transitions

From State	To State (reason)
running	shut off (destroyed)
paused	shut off (destroyed)
crashed	shut off (destroyed)

Usage

"Terminating a virtual server" on page 100

Example

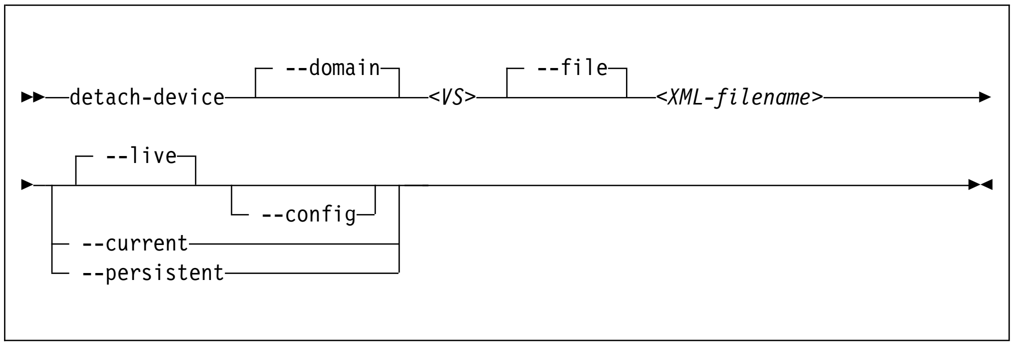
This example immediately terminates virtual server vserv1.

```
# 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 121

Example

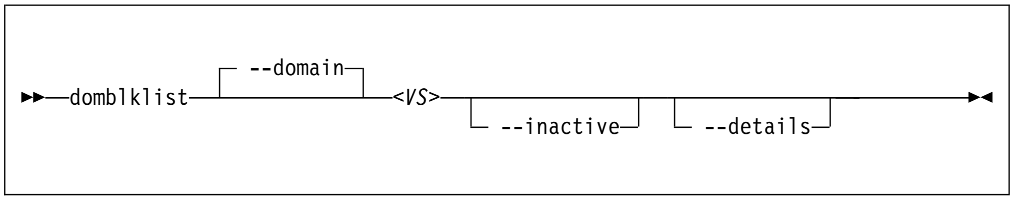
This example detaches the device that is defined in device configuration-XML file `vda.xml` from virtual server vserv1.

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

domblklist

Displays information about the virtual block devices of a virtual server.

Syntax



Where:

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

Selected options

--details
Display details, such as device type and value.

--domain
Specifies the virtual server.

--inactive
Lists the block devices that will be used with the next virtual server reboot.

Usage

"Displaying information about a virtual server" on page 104

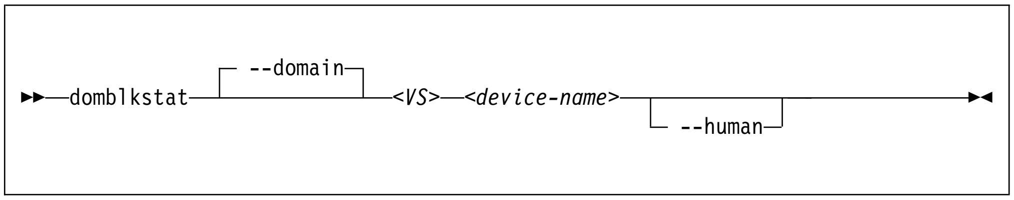
Example

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

domblkstat

Displays status information about a virtual block device.

Syntax



Where:

<device-name>
Is the name of the virtual block device.

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

Selected options

--domain
Specifies the virtual server.

--human
Replaces abbreviations by written-out information.

Usage

"Displaying information about a virtual server" on page 104

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:

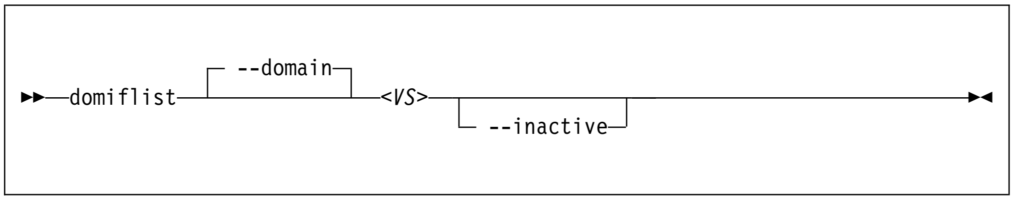
domblkstat

```
# virsh domblkstat vserv vda --human
Device: vda
 number of read operations:      20359
 number of bytes read:           235967488
 number of write operations:     4348
 number of bytes written:        54353920
 number of flush operations:     1372
 total duration of reads (ns):   49294200385
 total duration of writes (ns):  4626108064
 total duration of flushes (ns): 265417103
```

domiflist

Displays network interface information for a running virtual server.

Syntax



Where:

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

Selected options

--domain
Specifies the virtual server.

--inactive
Lists the interfaces that will be used with the next virtual server reboot.

Usage

"Displaying information about a virtual server" on page 104

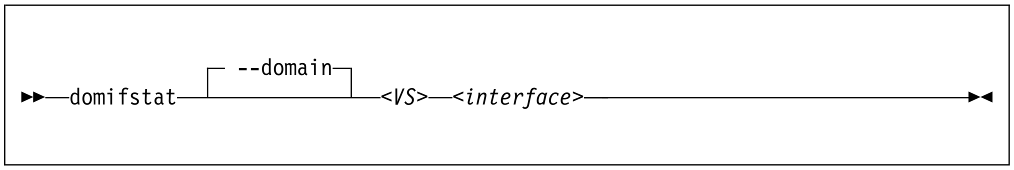
Example

```
#  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



Where:

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

<interface>
Is the name of the network interface as specified as target dev attribute in the configuration-XML file.

Selected options

--domain
Specifies the virtual server.

Usage

"Displaying information about a virtual server" on page 104

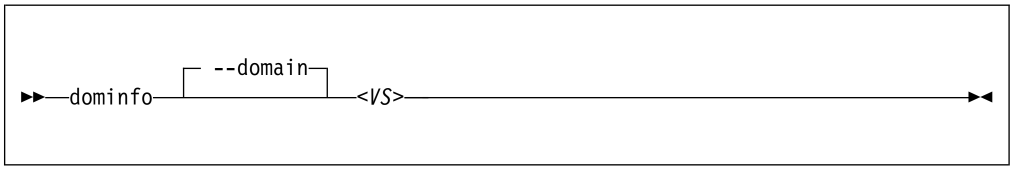
Example

```
# 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 104

Example

```
# virsh dominfo e20
Id:             55
Name:           e20
UUID:           65d6cee0-ca0a-d0c1-efc7-faacb8631497
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       1.2s
Max memory:     4194304 KiB
Used memory:    4194304 KiB
Persistent:     yes
Autostart:      enable
Managed save:   no
Security model: none
Security DOI:   0
```

domjobabort

Aborts the currently running virsh command related to the specified virtual server.

Syntax

```
                    ┌─ --domain─┐
►►──domjobabort─────┴───────────┴──<VS>──────────────────────────────────►◄
```

Where:

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

Selected options

None.

Usage

Chapter 15, "Performing a live migration," on page 109

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 104

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 139

Example

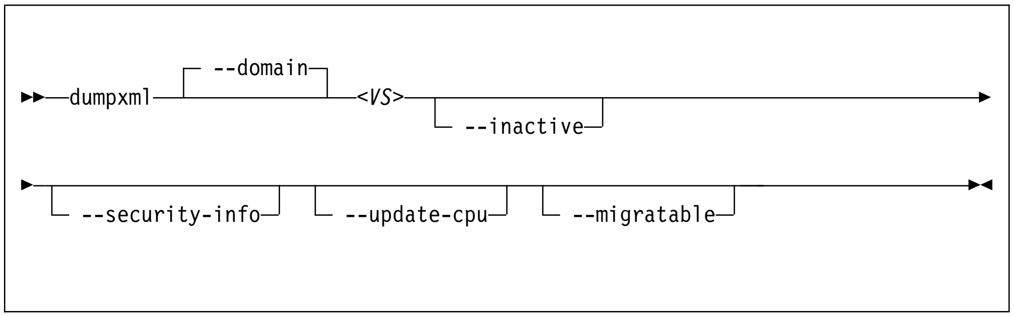
This example dumps the virtual server vserv1 to the file `dumpfile.name`.

```
# 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 106

Example

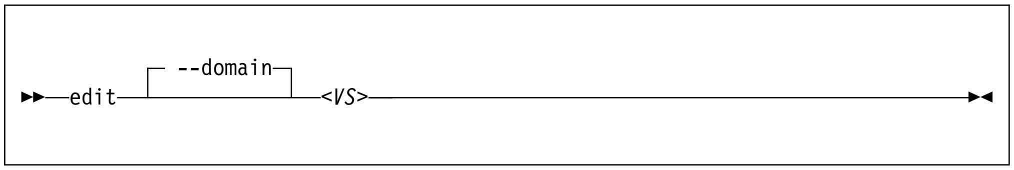
This example displays the current domain configuration-XML of virtual server vserv1.

```
# virsh dumpxml vserv1
```

edit

Edits the libvirt-internal configuration of a virtual server.

Syntax



Where:

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

Selected options

--domain
Specifies the virtual server.

Usage

"Modifying a virtual server definition" on page 96

Example

This example edits the libvirt-internal configuration of virtual server vserv1.

```
# virsh edit vserv1
```

inject-nmi

Causes a restart interrupt for a virtual server including a dump on the virtual server, if it is configured respectively.

The dump is displayed in the virtual server file `/proc/vmcore`.

Syntax

```
►►──inject-nmi──<VS>──────────────────────────────────────────────────────►◄
```

Where:

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

Selected options

None.

Usage

"Creating a dump on the virtual server" on page 139

Example

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

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

iothreadadd

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

Syntax

```
                  ┌ --domain┐       ┌ --id┐
►►─iothreadadd────┴─────────┴─<VS>──┴─────┴─<IOthread-ID>─────────────►

►─┬───────────┬───────────────────────────────────────────────────────►◄
  ├ --config──┤
  ├ --live────┤
  └ --current─┘
```

Where:

<IOthread-ID>
: Is the ID of the I/O thread to be added to the virtual server. The I/O thread ID must be beyond the range of available I/O threads.

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

Selected options

--config
: Affects the virtual server the next time it is restarted.

--current
: Affects the current virtual server.

--domain
: Specifies the virtual server.

--id Specifies the ID of the I/O thread that will be added to the I/O threads of the virtual server.

--live Affects the current virtual server only if it is running.

Usage

"Attaching a device" on page 120

Example

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

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

# virsh iothreadadd vserv1 4

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

iothreaddel

Removes an I/O thread from a virtual server.

If the specified I/O thread is assigned to a virtual block device that belongs to the current configuration of the virtual server, it is not removed.

Syntax

```
                  ┌─ --domain─┐          ┌─ --id─┐
►►──iothreaddel───┴───────────┴──<VS>────┴───────┴──<IOthread-ID>──────────────►

►──┬─────────────┬─────────────────────────────────────────────────────────────►◄
   ├─ --config───┤
   ├─ --live─────┤
   └─ --current──┘
```

Where:

<IOthread-ID>
: Is the ID of the I/O thread to be deleted from the virtual server.

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

Selected options

--config
: Affects the virtual server the next time it is restarted.

--current
: Affects the current virtual server.

--domain
: Specifies the virtual server.

--id Specifies the ID of the I/O thread that will be removed from the I/O threads of the virtual server.

--live Affects the current virtual server only if it is running.

Usage

"Detaching a device" on page 121

Example

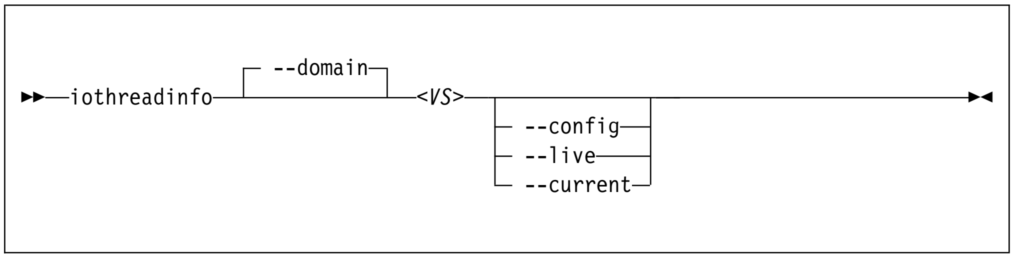
This example shows the **iothreadinfo** command for 8 virtual CPUs:

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

# virsh iothreaddel vserv1 3

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

iothreadinfo

Displays information about the I/O threads of a virtual server.

Syntax



Where:

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

Selected options

--config
Affects the virtual server the next time it is restarted.

--current
Affects the current virtual server.

--domain
Specifies the virtual server.

--live Affects the current virtual server only if it is running.

Usage

"Displaying information about a virtual server" on page 104

Example

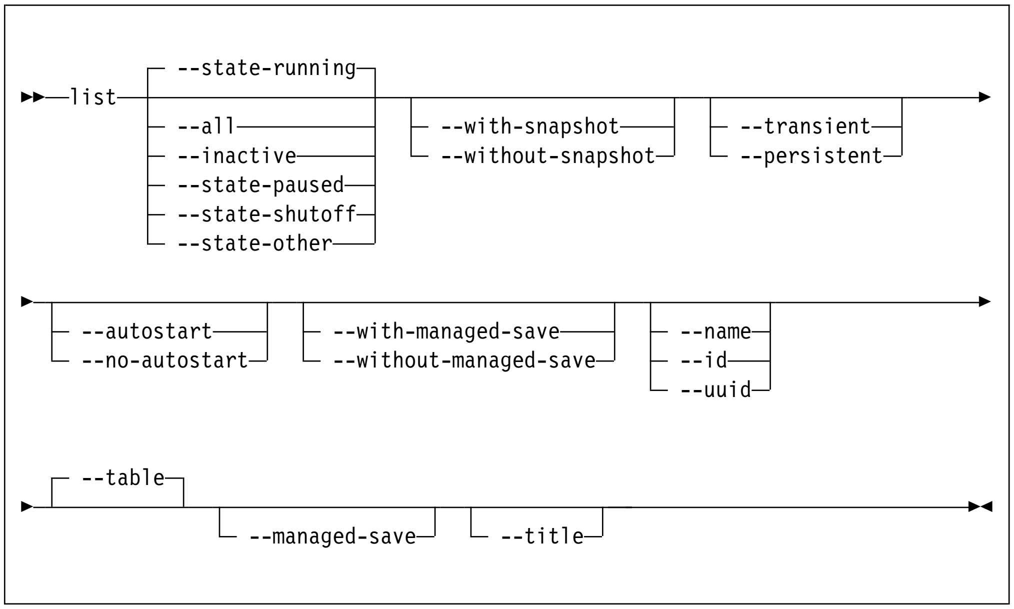
This example shows the **iothreadinfo** command for 8 virtual CPUs:

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

list

Browses defined virtual servers.

Syntax



Selected options

--all Lists all defined virtual servers.

--autostart
Lists all defined virtual servers with autostart enabled.

--inactive
Lists all defined virtual servers that are not running.

--managed-save
Only when **--table** is specified.

--name
Lists only virtual server names.

--no-autostart
Lists only virtual servers with disabled autostart option.

--persistent
Lists persistent virtual servers.

--state-other
Lists virtual servers in state “shutting down”.

--state-paused
Lists virtual servers in state “paused”.

--state-running
Lists virtual servers in state “running”.

--state-shutoff
Lists virtual servers in state “shut off”.

--table Displays the listing as a table.

--title Displays only a short virtual server description.

--transient
Lists transient virtual servers.

--uuid Lists only UUIDs.

--with-managed-save
Lists virtual servers with managed save state.

--with-snapshot
Lists virtual servers with existing snapshot.

--without-managed-save
Lists virtual servers without managed save state.

--without-snapshot
Lists virtual servers without existing snapshot.

Usage

"Browsing virtual servers" on page 104

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

```
                                            ┌─ --domain─┐
►►──managedsave──┬────────────────────┬─────┴───────────┴──<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 filesystem 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 100
- Chapter 23, "Virtual server life cycle," on page 147

Example

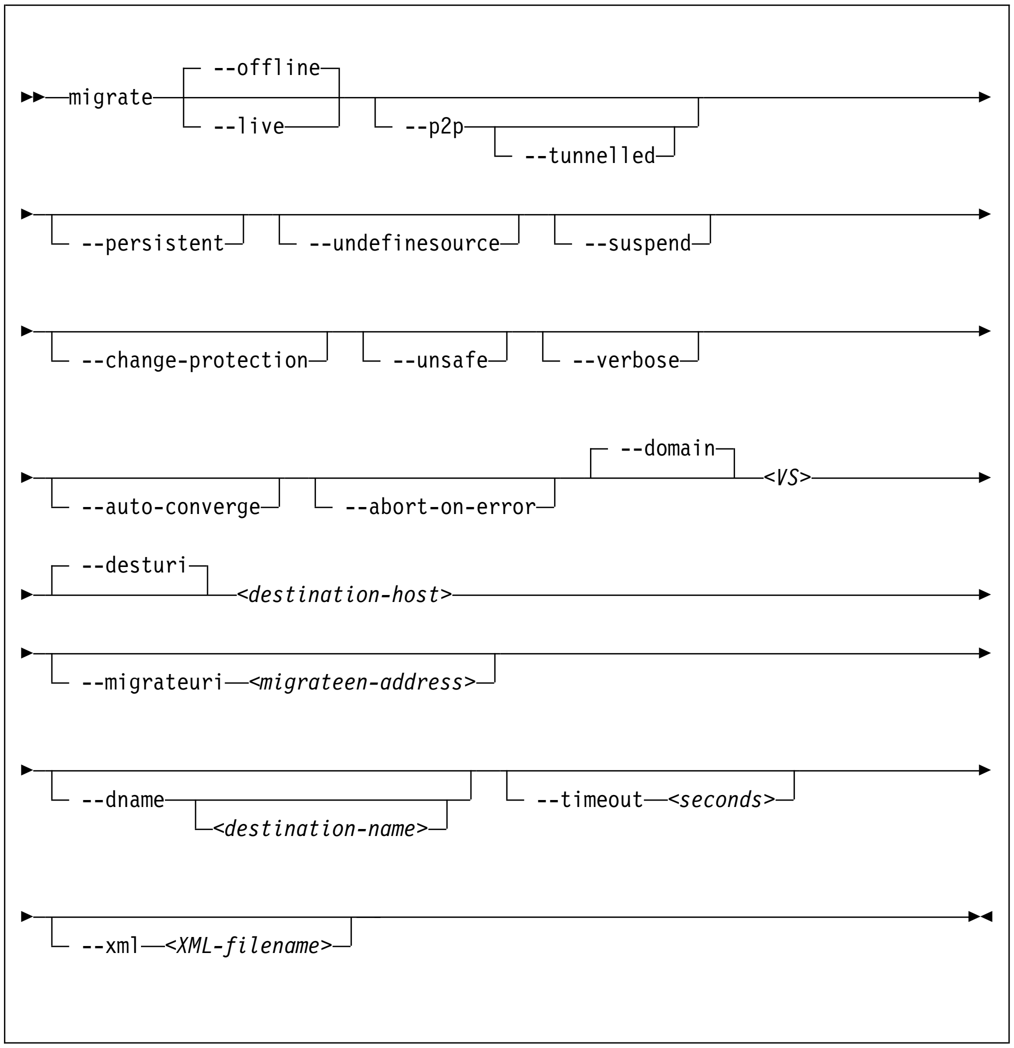
```
# virsh managedsave vserv1 --running
Domain vserv1 state saved by libvirt

# 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
```

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.

<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

--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.

--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 tunnelled 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.

--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

Chapter 15, "Performing a live migration," on page 109

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

```
                          ┌─ --domain─┐
►►──migrate-getspeed──────┴───────────┴──<VS>──────────────────────────────►◄
```

Where:

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

Selected options

None.

Usage

Chapter 15, “Performing a live migration,” on page 109

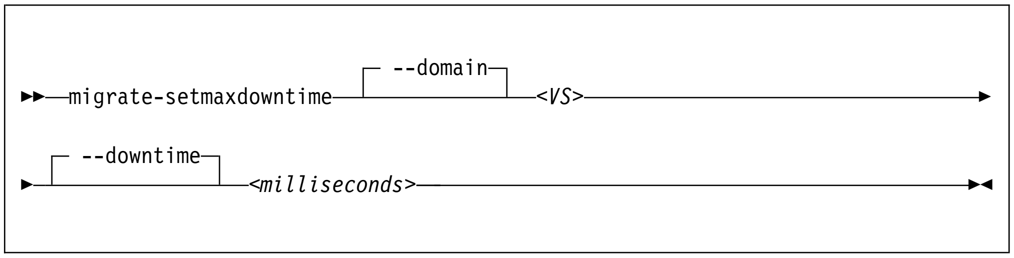
Example

```
# virsh migrate-getspeed vserv1
8796093022207
```

migrate-setmaxdowntime

Specifies a tolerable downtime for the virtual server during the migration, which is used to estimate the point in time when to suspend it.

Syntax



where

<milliseconds>
Is the tolerable downtime of the virtual server during migration in milliseconds.

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

Selected options

None.

Usage

Chapter 15, "Performing a live migration," on page 109

Example

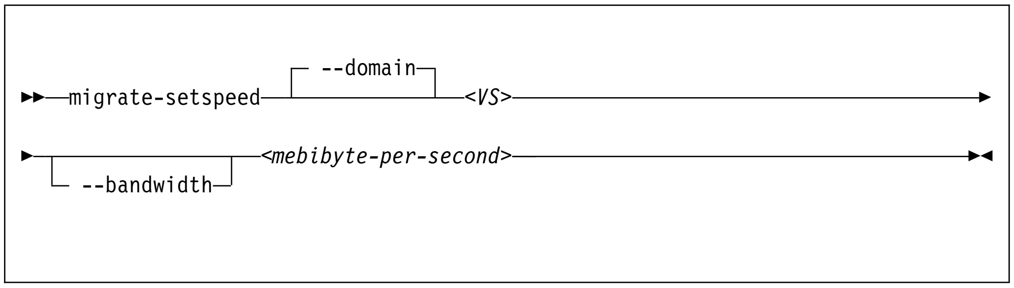
This example specifies a tolerable downtime of 100 milliseconds for the virtual server vserv1 in case it is migrated to another host.

```
# 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

Chapter 15, "Performing a live migration," on page 109

Example

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

reboot

Reboots a virtual server.

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 184.

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 102

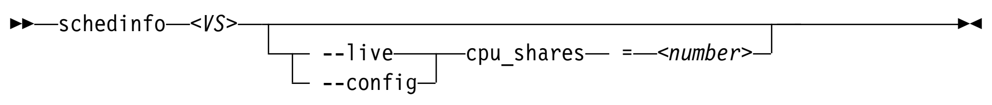
Example

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

# virsh resume vserv1
Domain vserv1 resumed

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

schedinfo

Displays scheduling information about a virtual server, and can modify the portion of CPU time that is assigned to it.

Syntax



Where:

<number>
Specifies the CPU weight.

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

Selected options

--live Specifies the modification of the current CPU weight of the running virtual server.

--config
Specifies the modification of the virtual server's CPU weight after the next restart.

Usage

"Modifying the virtual CPU weight" on page 116

Examples

This example sets the CPU weight of the running virtual server vserv1 to 2048.

```
# virsh schedinfo vserv1 --live cpu_shares=2048
```

This example modifies the domain configuration-XML, which will be effective from the next restart.

```
# virsh schedinfo vserv1 --config cpu_shares=2048
```

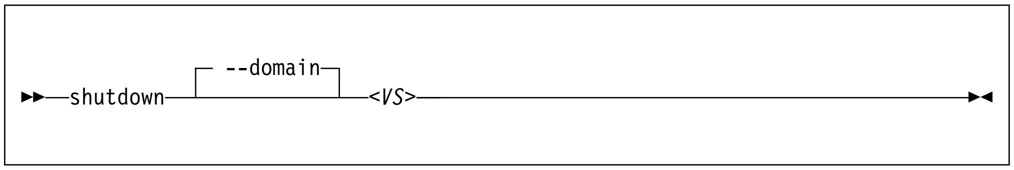
This example displays scheduling information about the virtual server vserv1.

```
# virsh schedinfo vserv1
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
```

shutdown

Properly shuts down a running virtual server.

Syntax



Where:

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

Selected options

--domain
Specifies the virtual server.

Virtual server state transitions

From State	To State (reason)
running	shut off (shutdown)

Usage

- Chapter 1, "Overview," on page 3
- "Terminating a virtual server" on page 100

Example

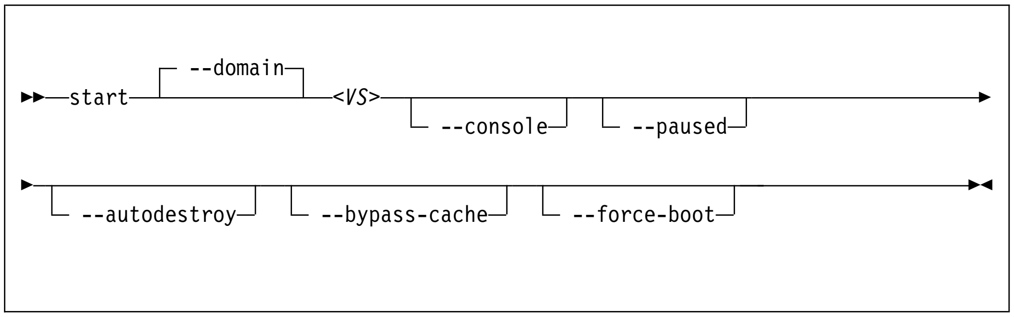
This example terminates virtual server vserv1.

```
# virsh shutdown vserv1
Domain vserv1 is being shutdown
```

start

Starts a defined virtual server that is shut off or crashed.

Syntax



```
            ┌─ --domain─┐
►►──start───┴───────────┴──<VS>──┬───────────┬──┬──────────┬──────────────────►
                                 └─ --console─┘  └─ --paused─┘

►──┬───────────────┬──┬────────────────┬──┬──────────────┬────────────────────►◄
   └─ --autodestroy─┘  └─ --bypass-cache─┘  └─ --force-boot─┘
```

Where:

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

Selected options

--autodestroy
Destroys the virtual server when virsh disconnects from libvirt.

--bypass-cache
Does not load the virtual server from the cache.

--console
Connects to a configured pty console.

--domain
Specifies the virtual server.

--force-boot
Any saved system image is discarded before booting.

--paused
Suspends the virtual server as soon as it is started.

Virtual server state transitions

Command option	From state (reason)	To state (reason)
start	shut off (unknown)	running (booted)
start	shut off (saved *from running*)	running (restored)
start	shut off (saved *from paused*)	paused (migrating)
start	shut off (shutdown)	running (booted)
start	shut off (destroyed)	running (booted)
start	crashed	running (booted)
start --force-boot	shut off (unknown)	running (booted)
start --force-boot	shut off (saved *from running*)	running (booted)
start --force-boot	shut off (saved *from paused*)	paused (user)
start --force-boot	shut off (shutdown)	running (booted)
start --force-boot	shut off (destroyed)	running (booted)

Command option	From state (reason)	To state (reason)
start --paused	shut off (unknown)	paused (user)
start --paused	shut off (saved *from running*)	paused (migrating)
start --paused	shut off (saved *from paused*)	paused (migrating)
start --paused	shut off (shutdown)	paused (user)
start --paused	shut off (destroyed)	paused (user)

Usage

- Chapter 1, “Overview,” on page 3
- “Starting a virtual server” on page 100
- “Connecting to the console of a virtual server” on page 123

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 102

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 97

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

```
                 ┌─ --domain─┐
►►──vcpucount────┴───────────┴──<VS>──┬─────────────┬──┬────────────┬──────────►◄
                                      ├─ --maximum──┤  ├─ --config──┤
                                      └─ --active───┘  ├─ --live────┤
                                                       └─ --current─┘
```

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.

Example

```
# virsh vcpucount vserv1
maximum      config        5
maximum      live          5
current      config        3
current      live          3
```

Chapter 26. Selected QEMU commands

QEMU monitor commands

Do not use the QEMU monitor commands, because their use can change the state of a virtual server, might disturb the correct operation of libvirt and lead to inconsistent states or even a crash of the virtual server.

info Displays information about the virtual server.

Examples for the use of the qemu-img command

- This example creates a QCOW2 image with a maximum size of 10GB:

```
# qemu-img create -f qcow2 /var/lib/libvirt/images/disk1.img 10G
Formatting '/var/lib/libvirt/images/disk1.img', fmt=qcow2
size=10737418240 encryption=off cluster_size=65536
lazy_refcounts=off
Format specific information:
compat: 1.1
lazy refcounts: false
refcount bits: 16
corrupt: false
```

- This example displays attributes of a QCOW2 image:

```
# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: qcow2
virtual size: 10G (10737418240 bytes)
disk size: 136K
cluster_size: 65536
```

- This example increases the size of a QCOW2 image:

```
# qemu-img resize /var/lib/libvirt/images/disk1.img 20G
Image resized.

# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 140K
cluster_size: 65536
```

- This example creates a RAW image with a maximum size of 10GB:

```
# qemu-img create -f raw /var/lib/libvirt/images/disk1.img 10G
Formatting '/var/lib/libvirt/images/disk1.img', fmt=raw
size=10737418240
```

- This example displays attributes of a RAW image:

```
# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: raw
virtual size: 10G (10737418240 bytes)
disk size: 0
```

- This example increases the size of a RAW image:

```
# qemu-img resize -f raw /var/lib/libvirt/images/disk1.img 20G
Image resized.

# qemu-img info /var/lib/libvirt/images/disk1.img
image: /var/lib/libvirt/images/disk1.img
file format: raw
virtual size: 20G (21474836480 bytes)
disk size: 0
```

Part 8. Appendixes

Accessibility

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Documentation accessibility

The Linux on z Systems publications are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF file and want to request a Web-based format for this publication, use the Readers' Comments form in the back of this publication, send an email to eservdoc@de.ibm.com, or write to:

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility at

`www.ibm.com/able`

Notices

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

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

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

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

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

Index

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

Linux on z Systems
KVM Virtual Server Management
March 2016

Publication No. SC34-2752-01

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
- Send your comments to the address on the reverse side of this form.
- Send your comments via email to: eservdoc@de.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-2752-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Research & Development GmbH
Information Development
Department 3282
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape **Please do not staple** Fold and Tape

SC34-2752-01

Cut or Fold
Along Line

SC34-2752-01