Managing Virtualization Platforms with `libvirt`

WHAT?

`libvirt` is a software toolkit that provides a consistent API for managing virtual machines across multiple hypervisor platforms such as KVM and QEMU.

WHY?

Use `libvirt` to eliminate the operational complexity of managing vendor-specific tools and to enable a unified management interface for diverse virtualization environments.

EFFORT?

It takes approximately 20 minutes to read this article and requires basic knowledge of Linux command-line operations and virtualization concepts.

GOAL

By the end of this guide, you will have a configured `libvirt` environment with functional storage pools and network bridges ready for VM guest deployment.

Publication Date: 05 Mar 2026

Contents

1 `libvirt` daemons

A `libvirt` deployment for accessing KVM requires one or more daemons to be installed and active on the host. `libvirt` provides two daemon deployment options: monolithic or modular daemons. `libvirt` has always provided the single monolithic daemon `libvirtd`. It includes the primary hypervisor drivers and all secondary drivers needed for managing storage, networking, node devices, etc. The monolithic `libvirtd` also provides secure remote access for external clients. Over time, `libvirt` added support for modular daemons, where each driver runs in its own daemon, allowing users to customize their `libvirt` deployment. Modular daemons are enabled by default, but a deployment can be switched to the traditional monolithic daemon by disabling the individual daemons and enabling `libvirtd`.

The modular daemon deployment is useful in scenarios where minimal `libvirt` support is needed. For example, if virtual machine storage and networking is not provided by `libvirt`, the `libvirt-daemon-driver-storage` and `libvirt-daemon-driver-network` packages are not required. Kubernetes is an example of an extreme case, where it handles all networking, storage, cgroups and namespace integration, etc. Only the `libvirt-daemon-driver-QEMU` package, providing `virtqemud`, needs to be installed. Modular daemons allow configuring a custom `libvirt` deployment containing only the components required for the use case.

1.1 Starting and stopping the modular daemons

The modular daemons are named after the driver which they are running, with the pattern "virt-*DRIVER*d". They are configured via the files `/etc/libvirt/virtDRIVERd.conf`. SUSE supports the `virtqemud` hypervisor daemons, along with all the secondary daemons:

- *virtnetworkd* - The virtual network management daemon which provides `libvirt`'s virtual network management APIs. For example, virtnetworkd can be used to create a NAT virtual network on the host for use by virtual machines.
- *virtnodedevd* - The host physical device management daemon which provides `libvirt`'s node device management APIs. For example, virtnodedevd can be used to detach a PCI device from the host for use by a virtual machine.
- *virtnwfilterd* - The host firewall management daemon which provides `libvirt`'s firewall management APIs. For example, virtnwfilterd can be used to configure network traffic filtering rules for virtual machines.

- *virtsecretd* - The host secret management daemon which provides `libvirt`'s secret management APIs. For example, virtsecretd can be used to store a key associated with a LUKs volume.
- *virtstoraged* - The host storage management daemon which provides `libvirt`'s storage management APIs. virtstoraged can be used to create storage pools and create volumes from those pools.
- *virtinterfaced* - The host NIC management daemon which provides `libvirt`'s host network interface management APIs. For example, virtinterfaced can be used to create a bonded network device on the host. SUSE discourages the use of `libvirt`'s interface management APIs in favor of default networking tools like wicked or NetworkManager. It is recommended to disable virtinterfaced.
- *virtproxyd* - A daemon to proxy connections between the traditional `libvirtd` sockets and the modular daemon sockets. With a modular `libvirt` deployment, virtproxyd allows remote clients to access the `libvirt` APIs similar to the monolithic `libvirtd`. It can also be used by local clients that connect to the monolithic `libvirtd` sockets.
- *virtlogd* - A daemon to manage logs from virtual machine consoles. virtlogd is also used by the monolithic `libvirtd`. The monolithic daemon and virtqemud `systemd` unit files require virtlogd, so it is not necessary to explicitly start virtlogd.
- *virtlockd* - A daemon to manage locks held against virtual machine resources such as disks. virtlockd is also used by the monolithic `libvirtd`. The monolithic daemon, virtqemud, `systemd` unit files require virtlockd, so it is not necessary to explicitly start virtlockd.

`virtlogd` and `virtlockd` are also used by the monolithic `libvirtd`. These daemons have always been separate from `libvirtd` for security reasons.

By default, the modular daemons listen for connections on the `/var/run/libvirt/virt`*`DRIVER`*`d-sock` and `/var/run/libvirt/virt`*`DRIVER`*`d-sock-ro` Unix Domain Sockets. The client library prefers these sockets over the traditional `/var/run/libvirt/libvirtd-sock`. The virtproxyd daemon is available for remote clients or local clients expecting the traditional `libvirtd` socket.

The `virtqemud` and `virtnetworkd`, `virtnodedevd`, `virtnwfilterd`, `virtstoraged` and `virtsecretd` are also enabled in the presets, ensuring the daemons are enabled and available when the corresponding packages are installed. Although enabled in presets for convenience, the modular daemons can also be managed with their `systemd` unit files:

- *virt`DRIVER`d.service* - The main unit file for launching the virt`DRIVER`d daemon. We recommend configuring the service to start on boot if VMs are also configured to start on host boot.
- *virt`DRIVER`d.socket* - The unit file corresponding to the main read-write UNIX socket `/var/run/libvirt/virtDRIVERd-sock`. We recommend starting this socket on boot by default.
- *virt`DRIVER`d-ro.socket* - The unit file corresponding to the main read-only UNIX socket `/var/run/libvirt/virtDRIVERd-sock-ro`. We recommend starting this socket on boot by default.
- *virt`DRIVER`d-admin.socket* - The unit file corresponding to the administrative UNIX socket `/var/run/libvirt/virtDRIVERd-admin-sock`. We recommend starting this socket on boot by default.

When `systemd` socket activation is used, several configuration settings in virt`DRIVER`d.conf are no longer honored. Instead, these settings must be controlled via the system unit files:

- *unix_sock_group* - UNIX socket group owner, controlled via the `SocketGroup` parameter in the `virtDRIVERd.socket` and `virtDRIVERd-ro.socket` unit files.
- *unix_sock_ro_perms* - Read-only UNIX socket permissions, controlled via the `SocketMode` parameter in the `virtDRIVERd-ro.socket` unit file.
- *unix_sock_rw_perms* - Read-write UNIX socket permissions, controlled via the `SocketMode` parameter in the `virtDRIVERd.socket` unit file.
- *unix_sock_admin_perms* - Admin UNIX socket permissions, controlled via the `SocketMode` parameter in the `virtDRIVERd-admin.socket` unit file.
- *unix_sock_dir* - Directory in which all UNIX sockets are created, independently controlled via the `ListenStream` parameter in any of the `virtDRIVERd.socket`, `virtDRIVERd-ro.socket` and `virtDRIVERd-admin.socket` unit files.

1.2 Starting and stopping the monolithic daemon

The monolithic daemon is known as `libvirtd` and is configured via `/etc/libvirt/libvirtd.conf`. `libvirtd` is managed with several `systemd` unit files:

- *libvirtd.service* - The main `systemd` unit file for launching `libvirtd`. We recommend configuring `libvirtd.service` to start on boot if VMs are also configured to start on host boot.
- *libvirtd.socket* - The unit file corresponding to the main read-write UNIX socket `/var/run/libvirt/libvirt-sock`. We recommend enabling this unit on boot.
- *libvirtd-ro.socket* - The unit file corresponding to the main read-only UNIX socket `/var/run/libvirt/libvirt-sock-ro`. We recommend enabling this unit on boot.
- *libvirtd-admin.socket* - The unit file corresponding to the administrative UNIX socket `/var/run/libvirt/libvirt-admin-sock`. We recommend enabling this unit on boot.
- *libvirtd-tcp.socket* - The unit file corresponding to the TCP 16509 port for non-TLS remote access. This unit should not be configured to start on boot until the administrator has configured a suitable authentication mechanism.
- *libvirtd-tls.socket* - The unit file corresponding to the TCP 16509 port for TLS remote access. This unit should not be configured to start on boot until the administrator has deployed x509 certificates and optionally configured a suitable authentication mechanism.

When `systemd` socket activation is used, certain configuration settings in `libvirtd.conf` are no longer honored. Instead, these settings must be controlled via the system unit files:

- *listen_tcp* - TCP socket usage is enabled by starting the `libvirtd-tcp.socket` unit file.
- *listen_tls* - TLS socket usage is enabled by starting the `libvirtd-tls.socket` unit file.
- *tcp_port* - Port for the non-TLS TCP socket, controlled via the `ListenStream` parameter in the `libvirtd-tcp.socket` unit file.
- *tls_port* - Port for the TLS TCP socket, controlled via the `ListenStream` parameter in the `libvirtd-tls.socket` unit file.
- *listen_addr* - IP address to listen on, independently controlled via the `ListenStream` parameter in the `libvirtd-tcp.socket` or `libvirtd-tls.socket` unit files.

- *unix_sock_group* - UNIX socket group owner, controlled via the `SocketGroup` parameter in the `libvirtd.socket` and `libvirtd-ro.socket` unit files.
- *unix_sock_ro_perms* - Read-only UNIX socket permissions, controlled via the `SocketMode` parameter in the `libvirtd-ro.socket` unit file.
- *unix_sock_rw_perms* - Read-write UNIX socket permissions, controlled via the `SocketMode` parameter in the `libvirtd.socket` unit file.
- *unix_sock_admin_perms* - Admin UNIX socket permissions, controlled via the `SocketMode` parameter in the `libvirtd-admin.socket` unit file.
- *unix_sock_dir* - Directory in which all UNIX sockets are created, independently controlled via the `ListenStream` parameter in any of the `libvirtd.socket`, `libvirtd-ro.socket` and `libvirtd-admin.socket` unit files.

1.3 Switching to the monolithic daemon

Several services need to be changed when switching from modular to the monolithic daemon. It is recommended to stop or evict any running virtual machines before switching between the daemon options.

1. Stop the modular daemons and their sockets. The following example disables the QEMU daemon for KVM and several secondary daemons.

```
for drv in qemu network nodedev nwfilter secret storage
do
 > sudo systemctl stop virt${drv}d.service
 > sudo systemctl stop virt${drv}d{,-ro,-admin}.socket
done
```

2. Disable future start of the modular daemons

```
for drv in qemu network nodedev nwfilter secret storage
do
 > sudo systemctl disable virt${drv}d.service
 > sudo systemctl disable virt${drv}d{,-ro,-admin}.socket
done
```

3. Enable the monolithic `libvirtd` service and sockets

```
> sudo systemctl enable libvirtd.service
> sudo systemctl enable libvirtd{,-ro,-admin}.socket
```

4. Start the monolithic `libvirtd` sockets

```
> sudo systemctl start libvirtd{,-ro,-admin}.socket
```

2 Preparing the VM Host Server

Before you can install guest virtual machines, you need to prepare the VM Host Server to provide the guests with the resources that they need for their operation. Specifically, you need to configure:

- *Networking* so that guests can make use of the network connection provided the host.
- A *storage pool* reachable from the host so that the guests can store their disk images.

2.1 Configuring networks

There are two common network configurations to provide a VM Guest with a network connection:

- A *network bridge*. This is the default and recommended way of providing the guests with network connection.
- A *virtual network* with forwarding enabled.

2.1.1 Network bridge

The network bridge configuration provides a Layer 2 switch for VM Guests, switching Layer 2 Ethernet packets between ports on the bridge based on MAC addresses associated with the ports. This gives the VM Guest Layer 2 access to the VM Host Server's network. This configuration is analogous to connecting the VM Guest's virtual Ethernet cable into a hub that is shared with the host and other VM Guests running on the host. The configuration is often referred to as *shared physical device*.

The network bridge configuration is the default configuration of SUSE Linux Enterprise Server when configured as a KVM hypervisor. It is the preferred configuration when you simply want to connect VM Guests to the VM Host Server's LAN.

2.1.1.1 Managing network bridges from the command line

This section includes procedures to add or remove network bridges using the command line.

2.1.1.1.1 Adding a network bridge

To add a new network bridge device on VM Host Server, follow these steps:

1. Log in as `root` on the VM Host Server where you want to create a new network bridge.
2. Choose a name for the new bridge—*`virbr_test`* in our example—and run

   ```
   # ip link add name VIRBR_TEST type bridge
   ```

3. Check if the bridge was created on VM Host Server:

   ```
   # bridge vlan
   [...]
   virbr_test  1 PVID Egress Untagged
   ```

   `virbr_test` is present, but is not associated with any physical network interface.
4. Bring the network bridge up and add a network interface to the bridge:

   ```
   # ip link set virbr_test up
   # ip link set eth1 master virbr_test
   ```

   Important: Network interface must be unused

   You can only assign a network interface that is not yet used by another network bridge.
5. Optionally, enable STP (see Spanning Tree Protocol (https://en.wikipedia.org/wiki/Spanning_Tree_Protocol)):

   ```
   # bridge link set dev virbr_test cost 4
   ```

2.1.1.1.2 Deleting a network bridge

To delete an existing network bridge device on VM Host Server from the command line, follow these steps:

1. Log in as `root` on the VM Host Server where you want to delete an existing network bridge.
2. List existing network bridges to identify the name of the bridge to remove:

```
# bridge vlan
[...]
virbr_test  1 PVID Egress Untagged
```

3. Delete the bridge:

```
# ip link delete dev virbr_test
```

2.1.1.2 Adding a network bridge with **nmcli**

This section includes procedures to add a network bridge with NetworkManager's command line tool **`nmcli`**.

1. List active network connections:

```
> sudo nmcli connection show --active
NAME                 UUID                                  TYPE      DEVICE
Ethernet connection 1  84ba4c22-0cfe-46b6-87bb-909be6cb1214  ethernet  eth0
```

2. Add a new bridge device named `br0` and verify its creation:

```
> sudo nmcli connection add type bridge ifname br0
Connection 'bridge-br0' (36e11b95-8d5d-4a8f-9ca3-ff4180eb89f7) \
successfully added.
> sudo nmcli connection show --active
NAME                 UUID                                  TYPE      DEVICE
bridge-br0           36e11b95-8d5d-4a8f-9ca3-ff4180eb89f7  bridge    br0
Ethernet connection 1  84ba4c22-0cfe-46b6-87bb-909be6cb1214  ethernet  eth0
```

3. Optionally, you can view the bridge settings:

```
> sudo nmcli -f bridge connection show bridge-br0
bridge.mac-address:                     --
bridge.stp:                             yes
bridge.priority:                        32768
bridge.forward-delay:                   15
bridge.hello-time:                      2
bridge.max-age:                         20
bridge.ageing-time:                     300
bridge.group-forward-mask:              0
bridge.multicast-snooping:              yes
bridge.vlan-filtering:                  no
bridge.vlan-default-pvid:               1
bridge.vlans:                           --
```

4. Link the bridge device to the physical Ethernet device `eth0`:

```
> sudo nmcli connection add type bridge-slave ifname eth0 master br0
```

5. Disable the `eth0` interface and enable the new bridge:

```
> sudo nmcli connection down "Ethernet connection 1"
> sudo nmcli connection up bridge-br0
Connection successfully activated (master waiting for slaves) \
(D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/9)
```

2.1.1.3 Using VLAN interfaces

Sometimes it is necessary to create a private connection either between two VM Host Servers or between VM Guest systems. For example, to migrate a VM Guest to hosts in a different network segment. Or to create a private bridge that only VM Guest systems may connect to (even when running on different VM Host Server systems). An easy way to build such connections is to set up VLAN networks.

VLAN interfaces are commonly set up on the VM Host Server. They either interconnect the different VM Host Server systems, or they may be set up as a physical interface to an otherwise virtual-only bridge. It is even possible to create a bridge with a VLAN as a physical interface that has no IP address in the VM Host Server. That way, the guest systems have no possibility to access the host over this network.

It is also possible to use the VLAN interface as a physical interface of a bridge. This makes it possible to connect several VM Host Server-only networks and allows live migration of VM Guest systems that are connected to such a network.

2.1.2 Virtual networks

`libvirt`-managed virtual networks are similar to bridged networks, but typically have no Layer 2 connection to the VM Host Server. Connectivity to the VM Host Server's physical network is accomplished with Layer 3 forwarding, which introduces additional packet processing on the VM Host Server as compared to a Layer 2 bridged network. Virtual networks also provide DHCP and DNS services for VM Guests. For more information on `libvirt` virtual networks, see the *Network XML format* documentation at https://libvirt.org/formatnetwork.html ↗.

A standard `libvirt` installation on SUSE Linux Enterprise Server already comes with a predefined virtual network named `default`. It provides DHCP and DNS services for the network, along with connectivity to the VM Host Server's physical network using the network address translation (NAT) forwarding mode. Although it is predefined, the `default` virtual network needs to be explicitly enabled by the administrator. For more information on the forwarding modes supported by `libvirt`, see the *Connectivity* section of the *Network XML format* documentation at https://libvirt.org/formatnetwork.html#elementsConnect ↗.

`libvirt`-managed virtual networks can be used to satisfy a wide range of use cases, but are commonly used on VM Host Servers that have a wireless connection or dynamic/sporadic network connectivity, such as laptops. Virtual networks are also useful when the VM Host Server's network has limited IP addresses, allowing forwarding of packets between the virtual network and the VM Host Server's network. However, most server use cases are better suited for the network bridge configuration, where VM Guests are connected to the VM Host Server's LAN.

Warning: Enabling forwarding mode

Enabling forwarding mode in a `libvirt` virtual network enables forwarding in the VM Host Server by setting `/proc/sys/net/ipv4/ip_forward` and `/proc/sys/net/ipv6/conf/all/forwarding` to 1, which turns the VM Host Server into a router. Restarting the VM Host Server's network may reset the values and disable forwarding. To avoid this behavior, explicitly enable forwarding in the VM Host Server by editing the `/etc/sysctl.conf` file and adding:

```
net.ipv4.ip_forward = 1
```

```
net.ipv6.conf.all.forwarding = 1
```

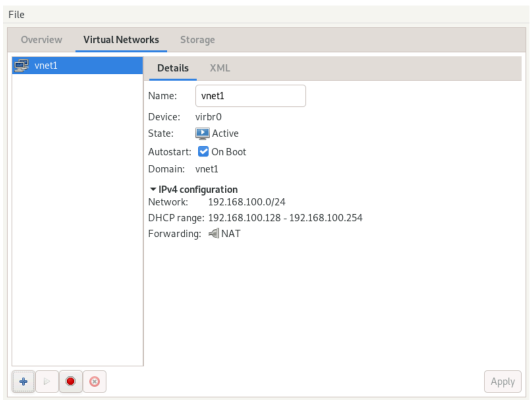
2.1.2.1 Managing virtual networks with Virtual Machine Manager

You can define, configure and operate virtual networks with Virtual Machine Manager.

2.1.2.1.1 Defining virtual networks

1. Start Virtual Machine Manager. In the list of available connections, right-click the name of the connection for which you need to configure the virtual network, and then select *Details*.

2. In the *Connection Details* window, click the *Virtual Networks* tab. You can see the list of all virtual networks available for the current connection. On the right, there are details of the selected virtual network.



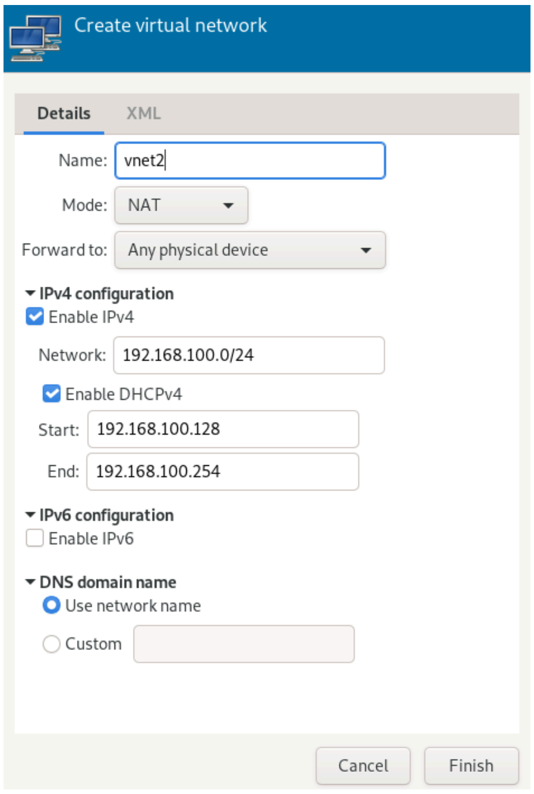
Name: vnet1
Device: virbr0
State: Active
Autostart: ✓ On Boot
Domain: vnet1

IPv4 configuration
Network: 192.168.100.0/24
DHCP range: 192.168.100.128 - 192.168.100.254
Forwarding: NAT

FIGURE 1: CONNECTION DETAILS

3. To add a new virtual network, click *Add*.

4. Specify a name for the new virtual network.


Create virtual network

Details | XML

Name: vnet2

Mode: NAT

Forward to: Any physical device

IPv4 configuration

[✓] Enable IPv4

Network: 192.168.100.0/24

[✓] Enable DHCPv4

Start: 192.168.100.128

End: 192.168.100.254

IPv6 configuration

[ ] Enable IPv6

DNS domain name

(●) Use network name

( ) Custom

FIGURE 2: CREATE VIRTUAL NETWORK

5. Specify the networking mode. For the *NAT* and *Routed* types, you can specify to which device to forward network communications. While *NAT* (network address translation) remaps the virtual network address space and allows sharing a single IP address, *Routed* forwards packets from the virtual network to the VM Host Server's physical network with no translation.

6. If you need IPv4 networking, activate *Enable IPv4* and specify the IPv4 network address. If you need a DHCP server, activate *Enable DHCPv4* and specify the assignable IP address range.

7. If you need IPv6 networking, activate *Enable IPv6* and specify the IPv6 network address. If you need a DHCP server, activate *Enable DHCPv6* and specify the assignable IP address range.

8. To specify a different domain name than the name of the virtual network, select *Custom* under *DNS domain name* and enter it here.

9. Click *Finish* to create the new virtual network. On the VM Host Server, a new virtual network bridge `virbrX` is available, which corresponds to the newly created virtual network. You can check with **`bridge link`**. `libvirt` automatically adds iptables rules to allow traffic to/from guests attached to the new *virbrX* device.

2.1.2.1.2 Starting virtual networks

To start a virtual network that is temporarily stopped, follow these steps:

1. Start Virtual Machine Manager. In the list of available connections, right-click the name of the connection for which you need to configure the virtual network, and then select *Details*.
2. In the *Connection Details* window, click the *Virtual Networks* tab. You can see the list of all virtual networks available for the current connection.
3. To start the virtual network, click *Start*.

2.1.2.1.3 Stopping virtual networks

To stop an active virtual network, follow these steps:

1. Start Virtual Machine Manager. In the list of available connections, right-click the name of the connection for which you need to configure the virtual network, and then select *Details*.
2. In the *Connection Details* window, click the *Virtual Networks* tab. You can see the list of all virtual networks available for the current connection.
3. Select the virtual network to be stopped, then click *Stop*.

2.1.2.1.4 Deleting virtual networks

To delete a virtual network from VM Host Server, follow these steps:

1. Start Virtual Machine Manager. In the list of available connections, right-click the name of the connection for which you need to configure the virtual network, and then select *Details*.
2. In the *Connection Details* window, click the *Virtual Networks* tab. You can see the list of all virtual networks available for the current connection.
3. Select the virtual network to be deleted, then click *Delete*.

2.1.2.1.5 Obtaining IP addresses with **nsswitch** for NAT networks (in KVM)

- On VM Host Server, install libvirt-nss, which provides NSS support for libvirt:

  ```
  > sudo zypper in libvirt-nss
  ```

- Add `libvirt` to `/etc/nsswitch.conf`:

  ```
  ...
  hosts:  files libvirt mdns_minimal [NOTFOUND=return] dns
  ...
  ```

- If NSCD is running, restart it:

  ```
  > sudo systemctl restart nscd
  ```

Now you can reach the guest system by name from the host.

The NSS module has limited functionality. It reads `/var/lib/libvirt/dnsmasq/*.status` files to find the host name and corresponding IP addresses in a JSON record describing each lease provided by **`dnsmasq`**. Host name translation can only be done on those VM Host Servers using a libvirt-managed bridged network backed by **`dnsmasq`**.

2.1.2.2 Managing virtual networks with **virsh**

You can manage `libvirt`-provided virtual networks with the **`virsh`** command line tool. To view all network related **`virsh`** commands, run

```
> sudo virsh help network
Networking (help keyword 'network'):
 net-autostart                  autostart a network
        net-create                     create a network from an XML file
        net-define                     define (but don't start) a network from an XML
 file
        net-destroy                    destroy (stop) a network
        net-dumpxml                    network information in XML
        net-edit                       edit XML configuration for a network
        net-event                      Network Events
        net-info                       network information
        net-list                       list networks
        net-name                       convert a network UUID to network name
        net-start                      start a (previously defined) inactive network
        net-undefine                   undefine an inactive network
        net-update                     update parts of an existing network's
 configuration
```

```
net-uuid                       convert a network name to network UUID
```

To view brief help information for a specific **`virsh`** command, run **`virsh help`** *`VIRSH_COMMAND`*:

```
> sudo virsh help net-create
  NAME
    net-create - create a network from an XML file

  SYNOPSIS
    net-create <file>

  DESCRIPTION
    Create a network.

  OPTIONS
    [--file] <string>  file containing an XML network description
```

2.1.2.2.1 Creating a network

To create a new *running* virtual network, run

```
> sudo virsh net-create VNET_DEFINITION.xml
```

The *`VNET_DEFINITION.xml`* XML file includes the definition of the virtual network that `libvirt` accepts.

To define a new virtual network without activating it, run

```
> sudo virsh net-define VNET_DEFINITION.xml
```

The following examples illustrate definitions of different types of virtual networks.

EXAMPLE 1: NAT-BASED NETWORK

The following configuration allows VM Guests outgoing connectivity if it is available on the VM Host Server. Without VM Host Server networking, it allows guests to talk directly to each other.

```
<network>
<name>vnet_nated</name>❶
<bridge name="virbr1"/>❷
 <forward mode="nat"/>❸
 <ip address="192.168.122.1" netmask="255.255.255.0">❹
  <dhcp>
   <range start="192.168.122.2" end="192.168.122.254"/>❺
   <host mac="52:54:00:c7:92:da" name="host1.testing.com" \
    ip="192.168.1.101"/>❻
   <host mac="52:54:00:c7:92:db" name="host2.testing.com" \
    ip="192.168.1.102"/>
   <host mac="52:54:00:c7:92:dc" name="host3.testing.com" \
    ip="192.168.1.103"/>
  </dhcp>
 </ip>
</network>
```

1. The name of the new virtual network.
2. The name of the bridge device used to construct the virtual network. When defining a new network with a <forward> mode of `"nat"` or `"route"` (or an isolated network with no <forward> element), `libvirt` automatically generates a unique name for the bridge device if none is given.
3. Inclusion of the <forward> element indicates that the virtual network is connected to the physical LAN. The `mode` attribute specifies the forwarding method. The most common modes are `"nat"` (Network Address Translation, the default), `"route"` (direct forwarding to the physical network, no address translation), and `"bridge"` (network bridge configured outside of `libvirt`). If the <forward> element is not specified, the virtual network is isolated from other networks. For a complete list of forwarding modes, see https://libvirt.org/formatnetwork.html#elementsConnect ↗.
4. The IP address and netmask for the network bridge.
5. Enable DHCP server for the virtual network, offering IP addresses ranging from the specified `start` and `end` attributes.
6. The optional <host> elements specify hosts that are given names and predefined IP addresses by the built-in DHCP server. Any IPv4 host element must specify the following: the MAC address of the host to be assigned a given name, the IP to be assigned to that host, and the name to be given to that host by the DHCP server. An IPv6 host element differs slightly from that for IPv4: there is no `mac` attribute since a MAC address has no defined meaning in IPv6. Instead, the `name` attribute is used to identify the host to be assigned the IPv6 address. For DHCPv6, the `name` is the plain name of the client host sent by the client to the server. This method of assigning a specific IP address can also be used instead of the `mac` attribute for IPv4.

EXAMPLE 2: ROUTED NETWORK

The following configuration routes traffic from the virtual network to the LAN without applying any NAT. The IP address range must be preconfigured in the routing tables of the router on the VM Host Server network.

```
<network>
```

```
 <name>vnet_routed</name>
 <bridge name="virbr1"/>
 <forward mode="route" dev="eth1"/>1
 <ip address="192.168.122.1" netmask="255.255.255.0">
  <dhcp>
   <range start="192.168.122.2" end="192.168.122.254"/>
  </dhcp>
 </ip>
</network>
```

1 The guest traffic may only go out via the `eth1` network device on the VM Host Server.

EXAMPLE 3: ISOLATED NETWORK

This configuration provides an isolated private network. The guests can talk to each other, and to VM Host Server, but cannot reach any other machines on the LAN, as the <forward> element is missing in the XML description.

```
<network>
 <name>vnet_isolated</name>
 <bridge name="virbr3"/>
 <ip address="192.168.152.1" netmask="255.255.255.0">
  <dhcp>
   <range start="192.168.152.2" end="192.168.152.254"/>
  </dhcp>
 </ip>
 </network>
```

EXAMPLE 4: USING AN EXISTING BRIDGE ON VM HOST SERVER

This configuration shows how to use an existing VM Host Server's network bridge `br0`. VM Guests are directly connected to the physical network. Their IP addresses are all on the subnet of the physical network, and there are no restrictions on incoming or outgoing connections.

```
<network>
        <name>host-bridge</name>
        <forward mode="bridge"/>
        <bridge name="br0"/>
</network>
```

2.1.2.2.2 Listing networks

To list all virtual networks available to `libvirt`, run:

```
> sudo virsh net-list --all
```

```
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 crowbar              active     yes           yes
 vnet_nated           active     yes           yes
 vnet_routed          active     yes           yes
 vnet_isolated        inactive   yes           yes
```

To list available domains, run:

```
> sudo virsh list
 Id    Name                           State
----------------------------------------------------
 1     nated_sles12sp3                running
 ...
```

To get a list of interfaces of a running domain, run `domifaddr DOMAIN`, or optionally specify the interface to limit the output to this interface. By default, it additionally outputs their IP and MAC addresses:

```
> sudo virsh domifaddr nated_sles12sp3 --interface vnet0 --source lease
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
 vnet0      52:54:00:9e:0d:2b    ipv6         fd00:dead:beef:55::140/64
 -          -                    ipv4         192.168.100.168/24
```

To print brief information of all virtual interfaces associated with the specified domain, run:

```
> sudo virsh domiflist nated_sles12sp3
Interface  Type       Source       Model       MAC
-----------------------------------------------------------
vnet0      network    vnet_nated   virtio      52:54:00:9e:0d:2b
```

2.1.2.2.3 Getting details about a network

To get detailed information about a network, run:

```
> sudo virsh net-info vnet_routed
Name:           vnet_routed
UUID:           756b48ff-d0c6-4c0a-804c-86c4c832a498
Active:         yes
Persistent:     yes
Autostart:      yes
Bridge:         virbr5
```

2.1.2.2.4 Starting a network

To start an inactive network that was already defined, find its name (or unique identifier, UUID) with:

```
> sudo virsh net-list --inactive
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 vnet_isolated        inactive   yes           yes
```

Then run:

```
> sudo virsh net-start vnet_isolated
Network vnet_isolated started
```

2.1.2.2.5 Stopping a network

To stop an active network, find its name (or unique identifier, UUID) with:

```
> sudo virsh net-list --inactive
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 vnet_isolated        active     yes           yes
```

Then run:

```
> sudo virsh net-destroy vnet_isolated
Network vnet_isolated destroyed
```

2.1.2.2.6 Removing a network

To remove the definition of an inactive network from VM Host Server permanently, run:

```
> sudo virsh net-undefine vnet_isolated
Network vnet_isolated has been undefined
```

2.2 Configuring a storage pool

When managing a VM Guest on the VM Host Server itself, you can access the complete file system of the VM Host Server to attach or create virtual hard disks or to attach existing images to the VM Guest. However, this is not possible when managing VM Guests from a remote host. For this reason, `libvirt` supports so called “Storage Pools”, which can be accessed from remote machines.

Tip: CD/DVD ISO images

To be able to access CD/DVD ISO images on the VM Host Server from remote clients, they also need to be placed in a storage pool.

`libvirt` knows two different types of storage: volumes and pools.

Storage volume

A storage volume is a storage device that can be assigned to a guest—a virtual disk or a CD/DVD/floppy image. Physically, it can be a block device, for example, a partition or a logical volume, or a file on the VM Host Server.

Storage pool

A storage pool is a storage resource on the VM Host Server that can be used for storing volumes, similar to network storage for a desktop machine. Physically it can be one of the following types:

File system directory (*dir*)

A directory for hosting image files. The files can be either one of the supported disk formats (raw or qcow2), or ISO images.

Physical disk device (*disk*)

Use a complete physical disk as storage. A partition is created for each volume that is added to the pool.

Pre-formatted block device (*fs*)

Specify a partition to be used in the same way as a file system directory pool (a directory for hosting image files). The only difference to using a file system directory is that `libvirt` takes care of mounting the device.

iSCSI target (iscsi)

Set up a pool on an iSCSI target. You need to have been logged in to the volume once before to use it with `libvirt`. Volume creation on iSCSI pools is not supported; instead, each existing Logical Unit Number (LUN) represents a volume. Each volume/LUN also needs a valid (empty) partition table or disk label before you can use it. If missing, use **`fdisk`** to add it:

```
> sudo fdisk -cu /dev/disk/by-path/ip-192.168.2.100:3260-iscsi-
iqn.2010-10.com.example:[...]-lun-2
Device contains neither a valid DOS partition table, nor Sun, SGI
or OSF disklabel
Building a new DOS disklabel with disk identifier 0xc15cdc4e.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

LVM volume group (logical)

Use an LVM volume group as a pool. You can either use a predefined volume group, or create a group by specifying the devices to use. Storage volumes are created as partitions on the volume.

Warning: Deleting the LVM-based pool

When the LVM-based pool is deleted in the Storage Manager, the volume group is deleted as well. This results in a non-recoverable loss of all data stored on the pool.

Multipath devices (*mpath*)

At the moment, multipathing support is limited to assigning existing devices to the guests. Volume creation or configuring multipathing from within `libvirt` is not supported.

Network exported directory (*netfs*)

Specify a network directory to be used in the same way as a file system directory pool (a directory for hosting image files). The only difference to using a file system directory is that `libvirt` takes care of mounting the directory. The supported protocol is NFS.

SCSI host adapter (*scsi*)

Use an SCSI host adapter in almost the same way as an iSCSI target. We recommend to use a device name from `/dev/disk/by-*` rather than `/dev/sdX`. The latter can change, for example, when adding or removing hard disks. Volume creation on iSCSI pools is not supported. Instead, each existing LUN (Logical Unit Number) represents a volume.

Warning: Security considerations

To avoid data loss or data corruption, do not attempt to use resources such as LVM volume groups, iSCSI targets, etc., that are also used to build storage pools on the VM Host Server. There is no need to connect to these resources from the VM Host Server or to mount them on the VM Host Server—`libvirt` takes care of this.

Do not mount partitions on the VM Host Server by label. Under certain circumstances it is possible that a partition is labeled from within a VM Guest with a name existing on the VM Host Server.

2.2.1 Managing storage with `virsh`

Managing storage from the command line is also possible by using **`virsh`**. However, creating storage pools is currently not supported by SUSE. Therefore, this section is restricted to documenting functions such as starting, stopping and deleting pools, and volume management.

A list of all **`virsh`** subcommands for managing pools and volumes is available by running **`virsh help pool`** and **`virsh help volume`**, respectively.

2.2.1.1 Listing pools and volumes

List all pools currently active by executing the following command. To also list inactive pools, add the option `--all`:

```
> virsh pool-list --details
```

Details about a specific pool can be obtained with the `pool-info` subcommand:

```
> virsh pool-info POOL
```

By default, volumes can only be listed per pool. To list all volumes from a pool, enter the following command.

```
> virsh vol-list --details POOL
```

At the moment **`virsh`** offers no tools to show whether a volume is used by a guest or not. The following procedure describes a way to list volumes from all pools that are currently used by a VM Guest.

PROCEDURE 1: LISTING ALL STORAGE VOLUMES CURRENTLY USED ON A VM HOST SERVER

1. Create an XSLT stylesheet by saving the following content to a file, for example, ~/libvirt/guest_storage_list.xsl:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:output method="text"/>
     <xsl:template match="text()"/>
     <xsl:strip-space elements="*"/>
     <xsl:template match="disk">
       <xsl:text>  </xsl:text>
       <xsl:value-of select="(source/@file|source/@dev|source/@dir)[1]"/>
       <xsl:text>&#10;</xsl:text>
     </xsl:template>
   </xsl:stylesheet>
   ```

2. Run the following commands in a shell. It is assumed that the guest's XML definitions are all stored in the default location (`/etc/libvirt/qemu`). **`xsltproc`** is provided by the package `libxslt`.

   ```
   SSHEET="$HOME/libvirt/guest_storage_list.xsl"
   ```

```
cd /etc/libvirt/qemu
for FILE in *.xml; do
  basename $FILE .xml
  xsltproc $SSHEET $FILE
done
```

2.2.1.2 Starting, stopping, and deleting pools

Use the **`virsh`** pool subcommands to start, stop or delete a pool. Replace *`POOL`* with the pool's name or its UUID in the following examples:

Stopping a pool

```
> virsh pool-destroy POOL
```

Note: A pool's state does not affect attached volumes

Volumes from a pool attached to VM Guests are always available, regardless of the pool's state (*Active* (stopped) or *Inactive* (started)). The state of the pool solely affects the ability to attach volumes to a VM Guest via remote management.

Deleting a pool

```
> virsh pool-delete POOL
```

Warning: Deleting storage pools

See *Warning: Deleting storage pools*

Starting a pool

```
> virsh pool-start POOL
```

Enable autostarting a pool

```
> virsh pool-autostart POOL
```

Only pools that are marked to autostart are automatically started if the VM Host Server reboots.

Disable autostarting a pool

```
> virsh pool-autostart POOL --disable
```

2.2.1.3 Adding volumes to a storage pool

`virsh` offers two ways to add volumes to storage pools: either from an XML definition with `vol-create` and `vol-create-from` or via command line arguments with `vol-create-as`. The first two methods are currently not supported by SUSE, therefore this section focuses on the subcommand `vol-create-as`.

To add a volume to an existing pool, enter the following command:

```
> virsh vol-create-as POOL❶NAME❷ 12G --format❸raw|qcow2❹ --allocation 4G❺
```

❶ Name of the pool to which the volume should be added

❷ Name of the volume

❸ Size of the image, in this example 12 gigabytes. Use the suffixes k, M, G, T for kilobyte, megabyte, gigabyte, and terabyte, respectively.

❹ Format of the volume. SUSE currently supports `raw` and `qcow2`.

❺ Optional parameter. By default, **`virsh`** creates a sparse image file that grows on demand. Specify the amount of space that should be allocated with this parameter (4 gigabytes in this example). Use the suffixes k, M, G, T for kilobyte, megabyte, gigabyte, and terabyte, respectively.
When not specifying this parameter, a sparse image file with no allocation is generated. To create a non-sparse volume, specify the whole image size with this parameter (would be `12G` in this example).

2.2.1.3.1 Cloning existing volumes

Another way to add volumes to a pool is to clone an existing volume. The new instance is always created in the same pool as the original.

```
> virsh vol-clone NAME_EXISTING_VOLUME❶NAME_NEW_VOLUME❷ --pool POOL❸
```

❶ Name of the existing volume that should be cloned

❷ Name of the new volume

3 Optional parameter. `libvirt` tries to locate the existing volume automatically. If that fails, specify this parameter.

2.2.1.4 Deleting volumes from a storage pool

To permanently delete a volume from a pool, use the subcommand `vol-delete`:

```
> virsh vol-delete NAME --pool POOL
```

`--pool` is optional. `libvirt` tries to locate the volume automatically. If that fails, specify this parameter.

Warning: No checks upon volume deletion

A volume is deleted in any case, regardless of whether it is currently used in an active or inactive VM Guest. There is no way to recover a deleted volume.

Whether a volume is used by a VM Guest can only be detected by using by the method described in *Procedure 1, "Listing all storage volumes currently used on a VM Host Server"*.

2.2.1.5 Attaching volumes to a VM Guest

After you create a volume as described in *Section 2.2.1.3, "Adding volumes to a storage pool"*, you can attach it to a virtual machine and use it as a hard disk:

```
> virsh attach-disk DOMAIN SOURCE_IMAGE_FILE TARGET_DISK_DEVICE
```

For example:

```
> virsh attach-disk sles12sp3 /virt/images/example_disk.qcow2 sda2
```

To check if the new disk is attached, inspect the result of the **`virsh dumpxml`** command:

```
# virsh dumpxml sles12sp3
[...]
<disk type='file' device='disk'>
 <driver name='qemu' type='raw'/>
 <source file='/virt/images/example_disk.qcow2'/>
 <backingStore/>
 <target dev='sda2' bus='scsi'/>
 <alias name='scsi0-0-0'/>
 <address type='drive' controller='0' bus='0' target='0' unit='0'/>
</disk>
[...]
```

2.2.1.5.1 Hotplug or persistent change

You can attach disks to both active and inactive domains. The attachment is controlled by the `--live` and `--config` options:

`--live`
: Hotplugs the disk to an active domain. The attachment is not saved in the domain configuration. Using `--live` on an inactive domain is an error.

`--config`
: Changes the domain configuration persistently. The attached disk is then available after the next domain start.

`--live--config`
: Hotplugs the disk and adds it to the persistent domain configuration.

Tip: `virsh attach-device`

`virsh attach-device` is the more generic form of **`virsh attach-disk`**. You can use it to attach other types of devices to a domain.

2.2.1.6 Detaching volumes from a VM Guest

To detach a disk from a domain, use **`virsh detach-disk`**:

```
# virsh detach-disk DOMAIN TARGET_DISK_DEVICE
```

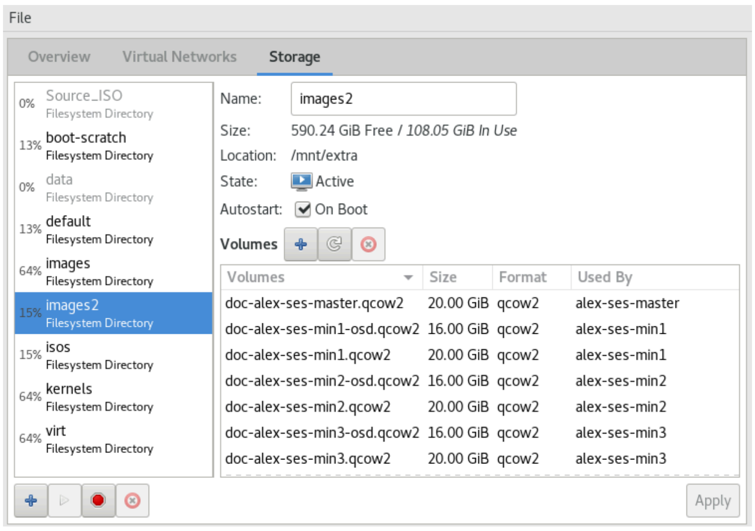
For example:

```
# virsh detach-disk sles12sp3 sda2
```

You can control the attachment with the `--live` and `--config` options as described in *Section 2.2.1.5, "Attaching volumes to a VM Guest"*.

2.2.2 Managing storage with Virtual Machine Manager

The Virtual Machine Manager provides a graphical interface—the Storage Manager—to manage storage volumes and pools. To access it, either right-click a connection and choose *Details*, or highlight a connection and choose *Edit* › *Connection Details*. Select the *Storage* tab.



0%	Source_ISO	Filesystem Directory
13%	boot-scratch	Filesystem Directory
0%	data	Filesystem Directory
13%	default	Filesystem Directory
64%	images	Filesystem Directory
15%	images2	Filesystem Directory
15%	isos	Filesystem Directory
64%	kernels	Filesystem Directory
64%	virt	Filesystem Directory

Name: images2

Size: 590.24 GiB Free / *108.05 GiB In Use*

Location: /mnt/extra

State: Active

Autostart: ☑ On Boot

Volumes

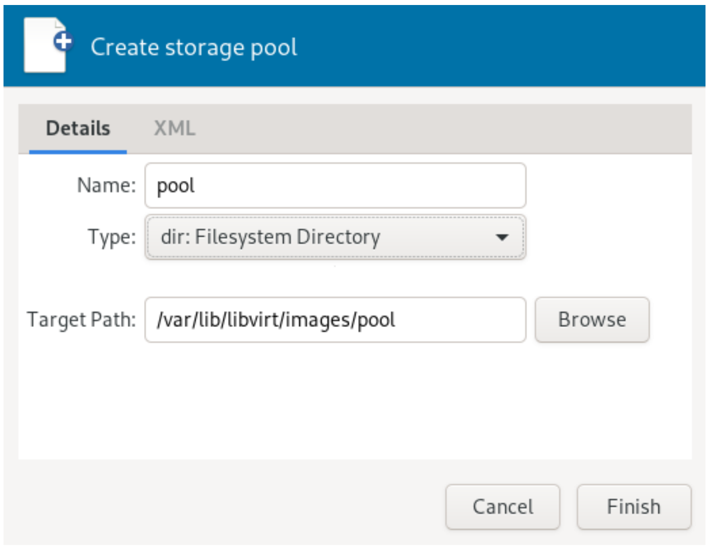
Volumes	Size	Format	Used By
doc-alex-ses-master.qcow2	20.00 GiB	qcow2	alex-ses-master
doc-alex-ses-min1-osd.qcow2	16.00 GiB	qcow2	alex-ses-min1
doc-alex-ses-min1.qcow2	20.00 GiB	qcow2	alex-ses-min1
doc-alex-ses-min2-osd.qcow2	16.00 GiB	qcow2	alex-ses-min2
doc-alex-ses-min2.qcow2	20.00 GiB	qcow2	alex-ses-min2
doc-alex-ses-min3-osd.qcow2	16.00 GiB	qcow2	alex-ses-min3
doc-alex-ses-min3.qcow2	20.00 GiB	qcow2	alex-ses-min3

2.2.2.1 Adding a storage pool

To add a storage pool, proceed as follows:

1. Click *Add* in the bottom left corner. The dialog *Add a New Storage Pool* appears.

2. Provide a *Name* for the pool (consisting of only alphanumeric characters and _, - or .) and select a *Type*.



Name: pool

Type: dir: Filesystem Directory

Target Path: /var/lib/libvirt/images/pool

3. Specify the required details below. They depend on the type of pool you are creating.

Important

ZFS pools are not supported.

Type *dir*

- *Target Path*: specify an existing directory.

Type *disk*

- *Format*: format of the device's partition table. Using *auto* should normally work. If not, get the required format by running the command **`parted`** `-l` on the VM Host Server.
- *Source Path*: path to the device. It is recommended to use a device name from `/dev/disk/by-*` rather than the simple `/dev/sdX`, since the latter can change, for example, when adding or removing hard disks. You need to specify the path that resembles the whole disk, not a partition on the disk (if existing).

Type *fs*

- *Target Path*: mount point on the VM Host Server file system.
- *Format:* file system format of the device. The default value `auto` should work.
- *Source Path*: path to the device file. It is recommended to use a device name from `/dev/disk/by-*` rather than `/dev/sdX`, because the latter can change, for example, when adding or removing hard disks.

Type *iscsi*

Get the necessary data by running the following command on the VM Host Server:

```
> sudo iscsiadm --mode node
```

It returns a list of iSCSI volumes with the following format. The elements in bold text are required:

```
IP_ADDRESS:PORT,TPGT TARGET_NAME_(IQN)
```

- *Target Path*: the directory containing the device file. Use `/dev/disk/by-path` (default) or `/dev/disk/by-id`.
- *Host Name*: host name or IP address of the iSCSI server.
- *Source IQN*: the iSCSI target name (iSCSI Qualified Name).
- *Initiator IQN*: the iSCSI initiator name.

Type ***logical***

- *Volgroup Name*: specify the device path of an existing volume group.

Type ***mpath***

- *Target Path*: support for multipathing is currently limited to making all multipath devices available. Therefore, specify an arbitrary string here. The path is required, otherwise the XML parser fails.

Type ***netfs***

- *Target Path*: mount point on the VM Host Server file system.
- *Host Name*: IP address or host name of the server exporting the network file system.
- *Source Path*: directory on the server that is being exported.

Type ***rbd***

- *Host Name*: host name of the server with an exported RADOS block device.
- *Source Name*: name of the RADOS block device on the server.

Type ***scsi***

- *Target Path*: directory containing the device file. Use `/dev/disk/by-path` (default) or `/dev/disk/by-id`.
- *Source Path*: name of the SCSI adapter.

> **Note: File browsing**
>
> Using the file browser by clicking *Browse* is not possible when operating remotely.

4. Click *Finish* to add the storage pool.

2.2.2.2 Managing storage pools

Virtual Machine Manager's Storage Manager lets you create or delete volumes in a pool. You may also temporarily deactivate or permanently delete existing storage pools. Changing the basic configuration of a pool is currently not supported by SUSE.

2.2.2.2.1 Starting, stopping, and deleting pools

The purpose of storage pools is to provide block devices located on the VM Host Server that can be added to a VM Guest when managing it from remote. To make a pool temporarily inaccessible from remote, click *Stop* in the bottom left corner of the Storage Manager. Stopped pools are marked with *State: Inactive* and are grayed out in the list pane. By default, a newly created pool is automatically started *On Boot* of the VM Host Server.

To start an inactive pool and make it available from remote again, click *Start* in the bottom left corner of the Storage Manager.

> **Note: A pool's state does not affect attached volumes**
>
> Volumes from a pool attached to VM Guests are always available, regardless of the pool's state (*Active* (stopped) or *Inactive* (started)). The state of the pool solely affects the ability to attach volumes to a VM Guest via remote management.

To permanently make a pool inaccessible, click *Delete* in the bottom left corner of the Storage Manager. You can only delete inactive pools. Deleting a pool does not physically erase its contents on VM Host Server—it only deletes the pool configuration. However, you need to be extra careful when deleting pools, especially when deleting LVM volume group-based tools:

Warning: Deleting storage pools

Deleting storage pools based on *local* file system directories, local partitions or disks has no effect on the availability of volumes from these pools currently attached to VM Guests.

Volumes located in pools of type iSCSI, SCSI, LVM group or Network Exported Directory become inaccessible from the VM Guest if the pool is deleted. Although the volumes themselves are not deleted, the VM Host Server can no longer access the resources.

Volumes on iSCSI/SCSI targets or Network Exported Directory become accessible again when creating an adequate new pool or when mounting/accessing these resources directly from the host system.

When deleting an LVM group-based storage pool, the LVM group definition is erased and the LVM group no longer exists on the host system. The configuration is not recoverable and all volumes from this pool are lost.

2.2.2.2.2 Adding volumes to a storage pool

Virtual Machine Manager lets you create volumes in all storage pools, except in pools of types Multipath, iSCSI or SCSI. A volume in these pools is equivalent to a LUN and cannot be changed from within `libvirt`.

1. A new volume can either be created using the Storage Manager or while adding a new storage device to a VM Guest. In either case, select a storage pool from the left panel, then click *Create new volume*.

2. Specify a *Name* for the image and choose an image format.
   SUSE currently only supports `raw` or `qcow2` images. The latter option is not available on LVM group-based pools.
   Next to *Max Capacity*, specify the maximum size that the disk image is allowed to reach. Unless you are working with a `qcow2` image, you can also set an amount for *Allocation* that should be allocated initially. If the two values differ, a sparse image file is created, which grows on demand.

For `qcow2` images, you can use a *Backing Store* (also called “backing file”), which constitutes a base image. The newly created `qcow2` image then only records the changes that are made to the base image.

3. Start the volume creation by clicking *Finish*.

2.2.2.2.3 Deleting volumes from a storage pool

Deleting a volume can only be done from the Storage Manager, by selecting a volume and clicking *Delete Volume*. Confirm with *Yes*.

Warning: Volumes can be deleted even while in use

Volumes can be deleted even if they are currently used in an active or inactive VM Guest. There is no way to recover a deleted volume.

Whether a volume is used by a VM Guest is indicated in the *Used By* column in the Storage Manager.

3 Guest installation

A VM Guest consists of an image containing an operating system and data files and a configuration file describing the VM Guest's virtual hardware resources. VM Guests are hosted on and controlled by the VM Host Server. This section provides generalized instructions for installing a VM Guest. For a list of supported VM Guests, refer to the section *Supported guest operating systems* in the article Virtualization Limits and Support (https://documentation.suse.com/sles/16.0/html/SLES-virtualization-support/) ↗.

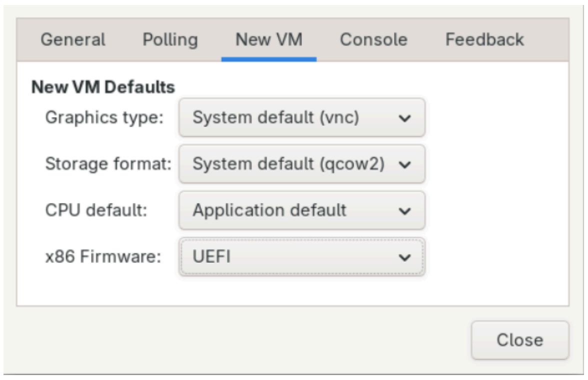
Virtual machines have few if any requirements above those required to run the operating system. If the operating system has not been optimized for the virtual machine host environment, it can only run on *hardware-assisted* virtualization computer hardware, in full virtualization mode, and requires specific device drivers to be loaded. The hardware that is presented to the VM Guest depends on the configuration of the host.

You should be aware of any licensing issues related to running a single licensed copy of an operating system on multiple virtual machines. Consult the operating system license agreement for more information.

3.1 GUI-based guest installation

Tip: Changing default options for new virtual machines

You can change default values that are applied when creating new virtual machines. For example, to set UEFI as the default firmware type for new virtual machines, select *Edit* › *Preferences* from Virtual Machine Manager's main menu, click *New VM* and set *UEFI* as the firmware default.



New VM Defaults

Graphics type: System default (vnc)

Storage format: System default (qcow2)

CPU default: Application default

x86 Firmware: UEFI

FIGURE 3: SPECIFYING DEFAULT OPTIONS FOR NEW VMS

The *New VM* wizard helps you through the steps required to create a virtual machine and install its operating system. To start it, open the Virtual Machine Manager and select *File* › *New Virtual Machine*.

1. Start the *New VM* with Virtual Machine Manager.
2. Choose an installation source—either a locally available media or a network installation source. To set up your VM Guest from an existing image, choose *import existing disk image*.
3. Depending on your choice in the previous step, you need to provide the following data:

    Local install media (ISO image or CDROM)

    Specify the path on the VM Host Server to an ISO image containing the installation data. If it is available as a volume in a libvirt storage pool, you can also select it using *Browse*.
    Alternatively, choose a physical CD-ROM or DVD inserted in the optical drive of the VM Host Server.

Network install (HTTP, HTTPS or FTP)

Provide the *URL* pointing to the installation source. Valid URL prefixes are, for example, `ftp://`, `http://` and `https://`.
Under *URL Options*, provide a path to an auto-installation file (AutoYaST or Kickstart, for example) and kernel parameters. Having provided a URL, the operating system should be automatically detected correctly. If this is not the case, deselect *Automatically Detect Operating System Based on Install-Media* and manually select the *OS Type* and *Version*.

Import existing disk image

To set up the VM Guest from an existing image, you need to specify the path on the VM Host Server to the image. If it is available as a volume in a libvirt storage pool, you can also select it using *Browse*.

Manual install

This installation method is suitable to create a virtual machine, manually configure its components and install its OS later. To adjust the VM to a specific product version, start typing its name, for example, `sles`—and select the desired version when a match appears.

4. Choose the memory size and number of CPUs for the new virtual machine.

5. This step is omitted when *Import an Existing Image* is chosen in the first step.
Set up a virtual hard disk for the VM Guest. Either create a new disk image or choose an existing one from a storage pool If you choose to create a disk, a `qcow2` image is created and stored under `/var/lib/libvirt/images` by default.
Setting up a disk is optional. If you are running a live system directly from CD or DVD, for example, you can omit this step by deactivating *Enable Storage for this Virtual Machine*.

6. On the last screen of the wizard, specify the name for the virtual machine. To be offered the possibility to review and make changes to the virtualized hardware selection, activate *Customize configuration before install*. Specify the network device under *Network Selection*. When using *Bridge device*, the first bridge found on the host is pre-filled. To use a different bridge, manually update the text box with its name.
Click *Finish*.

7. *(Optional)* If you kept the defaults in the previous step, the installation starts. If you selected *Customize configuration before install*, a VM Guest configuration dialog opens.
When you are done configuring, click *Begin Installation*.

Tip: Passing key combinations to virtual machines

The installation starts in a Virtual Machine Manager console window. Certain key combinations, such as `Ctrl`–`Alt`–`F1`, are recognized by the VM Host Server but are not passed to the virtual machine. To bypass the VM Host Server, Virtual Machine Manager provides the “sticky key” functionality. Pressing `Ctrl`, `Alt`, or `Shift` three times makes the key sticky, then you can press the remaining keys to pass the combination to the virtual machine.

For example, to pass `Ctrl`–`Alt`–`F2` to a Linux virtual machine, press `Ctrl` three times, then press `Alt`–`F2`. You can also press `Alt` three times, then press `Ctrl`–`F2`.

The sticky key functionality is available in the Virtual Machine Manager during and after installing a VM Guest.

3.1.1 Configuring the virtual machine for PXE boot

PXE boot enables your virtual machine to boot from the installation media via the network, instead of from a physical medium or an installation disk image.

To let your VM boot from a PXE server, follow these steps:

1. Start the installation wizard as described in *Section 3.1, “GUI-based guest installation”*.
2. Select the *Manual Install* method.
3. Proceed to the last step of the wizard and activate *Customize configuration before install*. Confirm with *Finish*.
4. On the *Customize* screen, select *Boot Options*.
5. Inspect *Boot device order* and select *Enable boot menu*.
   - To retain *VirtIO Disk* as the default boot option, confirm with *Apply*.
   - To force the virtual machine to use PXE as the default boot option:
     a. Select the NIC device in the boot menu configuration.
     b. Move it to the top using the arrow signs on the right.
     c. Confirm with *Apply*.

6. Start the installation by clicking *Begin Installation*. Now press `Esc` for boot menu and choose *1. iPXE*. If a PXE server is properly configured, the PXE menu screen appears.

3.2 Installing from the command line with **`virt-install`**

`virt-install` is a command-line tool that helps you create new virtual machines using the `libvirt` library. It is useful if you cannot use the graphical user interface, or need to automatize the process of creating virtual machines.

`virt-install` is a complex script with a lot of command line switches. The following are required. For more information, see the man page of **`virt-install`** (1).

General options

- `--name` *`VM_GUEST_NAME`*: Specify the name of the new virtual machine. The name must be unique across all guests known to the hypervisor on the same connection. It is used to create and name the guest's configuration file and you can access the guest with this name from **`virsh`**. Alphanumeric and `_-.:+` characters are allowed.
- `--memory` *`REQUIRED_MEMORY`*: Specify the amount of memory to allocate for the new virtual machine in megabytes.
- `--vcpus` *`NUMBER_OF_CPUS`*: Specify the number of virtual CPUs. For best performance, the number of virtual processors should be less than or equal to the number of physical processors.

Virtualization type

- `--paravirt`: set up a paravirtualized guest. This is the default if the VM Host Server supports paravirtualization and full virtualization.
- `--hvm`: set up a fully virtualized guest.
- `--virt-type` *`HYPERVISOR`*: Specify the hypervisor. Supported values is `kvm`.

Guest storage

Specify one of `--disk`, `--filesystem` or `--nodisks` the type of the storage for the new virtual machine. For example, `--disk size=10` creates a 10 GB disk in the default image location for the hypervisor and uses it for the VM Guest. `--filesystem` *`/export/path/on/vmhost`* specifies the directory on the VM Host Server to be exported to the guest. And `--nodisks` sets up a VM Guest without a local storage (good for Live CDs).

Installation method

Specify the installation method using one of `--location`, `--cdrom`, `--pxe`, `--import`, or `--boot` .

Accessing the installation

Use the `--graphics VALUE` option to specify how to access the installation. SUSE Linux Enterprise Server supports the values `vnc` or `none`.

If using VNC, **`virt-install`** tries to launch **`virt-viewer`**. If it is not installed or cannot be run, connect to the VM Guest manually with your preferred viewer. To explicitly prevent **`virt-install`** from launching the viewer, use `--noautoconsole`. To define a password for accessing the VNC session, use the following syntax: `--graphics vnc,password=PASSWORD`.

In case you are using `--graphics none`, you can access the VM Guest through operating system supported services, such as SSH or VNC. Refer to the operating system installation manual on how to set up these services in the installation system.

Passing kernel and initrd files

It is possible to directly specify the Kernel and Initrd of the installer, for example, from a network source.

To pass additional boot parameters, use the `--extra-args` option. This can be used to specify a network configuration.

EXAMPLE 5: LOADING KERNEL AND INITRD FROM HTTP SERVER

```
# virt-install --location "http://example.tld/REPOSITORY/DVD1/" \
--extra-args="textmode=1" --name "SLES15" --memory 2048 --virt-type kvm\
--connect qemu:///system --disk size=10 --graphics vnc \
--network network=vnet_nated
```

Enabling the console

By default, the console is not enabled for new virtual machines installed using **`virt-install`**. To enable it, use `--extra-args="console=ttyS0 textmode=1"` as in the following example:

```
> virt-install --virt-type kvm --name sles12 --memory 1024 \
 --disk /var/lib/libvirt/images/disk1.qcow2 --os-variant sles12
 --extra-args="console=ttyS0 textmode=1" --graphics none
```

After the installation finishes, the `/etc/default/grub` file in the VM image is updated with the `console=ttyS0` option on the `GRUB_CMDLINE_LINUX_DEFAULT` line.

Note

SUSE supports UEFI Secure Boot on AMD64/Intel 64 KVM guests only.

By default, new virtual machines installed using **`virt-install`** are configured with a legacy BIOS. They can be configured to use UEFI with `--boot firmware=efi`. A firmware that supports UEFI Secure Boot and has Microsoft keys enrolled will be selected. If secure boot is undesirable, the option `--boot firmware=efi,firmware.feature0.name=secure-boot,firmware.feature0.enabled=no` can be used to select a UEFI firmware without secure boot support.
It is also possible to explicitly specify a UEFI firmware image. See *Section 3.3.1, "Advanced UEFI configuration"* for advanced information and examples on using UEFI with virtual machines.

EXAMPLE 6: EXAMPLE OF A `virt-install` COMMAND LINE

The following command line example creates a new SLES 15 SP2 virtual machine with a virtio accelerated disk and network card. It creates a new 10 GB qcow2 disk image as a storage, the source installation media being the host CD-ROM drive. It uses VNC graphics, and it automatically launches the graphical client.

KVM

```
> virt-install --connect qemu:///system --virt-type kvm \
--name sle15sp2 --memory 1024 --disk size=10 --cdrom /dev/cdrom --graphics vnc
 \
--os-variant sle15sp2
```

3.3 Advanced guest installation scenarios

This section provides instructions for operations exceeding the scope of a normal installation, such as manually configuring UEFI firmware, memory ballooning and installing add-on products.

3.3.1 Advanced UEFI configuration

The UEFI firmware used by virtual machines is provided by *OVMF* (*Open Virtual Machine Firmware*). The `qemu-ovmf-x86_64` package provides firmware for AMD64/Intel 64 VM Guests. Firmware for AArch64 VM Guests is provided by the `qemu-uefi-aarch64` package. Both packages include several firmware variants, each supporting a different set of features and capabilities. The packages also include JSON firmware descriptor files, which describe the features and capabilities of each variant.

`libvirt` supports two methods of selecting virtual machine UEFI firmware: automatic and manual. With automatic selection, `libvirt` will select a firmware based on an optional set of features specified by the user. If no explicit features are specified, `libvirt` will select a firmware with secure boot enabled and Microsoft keys enrolled. When using manual selection, the full path of the firmware and any optional settings must be explicitly specified. Users can reference the JSON descriptor files to find a firmware that satisfies their requirements.

Tip

The directory `/usr/share/qemu/firmware` contains all the JSON files used by `libvirt`. This file gives you detailed information about the firmware, including the capabilities of the features.

When using **`virt-install`**, automatic firmware selection is enabled by specifying the *firmware=efi* parameter to the *boot* option, for example, `--boot firmware=efi`. The selection process can be influenced by requesting the presence or absence of firmware features. The following example illustrates automatic firmware selection with UEFI Secure Boot disabled.

```
> virt-install --connect qemu:///system --virt-type kvm \
--name sle15sp5 --memory 1024 --disk size=10 --cdrom /dev/cdrom --graphics vnc \
--boot firmware=efi,firmware.feature0.name=secure-boot,firmware.feature0.enabled=no \
--os-variant sle15sp5
```

Note

To ensure persistent VM Guests use the same firmware and variable store throughout their lifetime, `libvirt` will record automatically selected firmware in the VM Guest XML configuration. Automatic firmware selection is a one-time activity. Once firmware has been selected, it will only change if the VM Guest administrator explicitly does so using the manual firmware selection method.

The *loader* and *nvram* parameters are used for manual firmware selection. *loader* is required, and *nvram* defines an optional UEFI variable store. The following example illustrates manual firmware selection with secure boot enabled.

```
> virt-install --connect qemu:///system --virt-type kvm \
--name sle15sp5 --memory 1024 --disk size=10 --cdrom /dev/cdrom --graphics vnc \
--boot loader=/usr/share/qemu/ovmf-x86_64-smm-
code.bin,loader.readonly=yes,loader.type=pflash,loader.secure=yes,nvram.template=/usr/
share/qemu/ovmf-x86_64-smm-vars.bin \
--os-variant sle15sp5
```

Note

`libvirt` cannot modify any characteristics of the UEFI firmware. For example, it cannot disable UEFI Secure Boot in a firmware that has UEFI Secure Boot enabled, even when specifying *loader.secure=no*. `libvirt` will ensure the specified firmware can satisfy any specified features. For example, it will reject configuration that disables secure boot with *loader.secure=no*, but specifies a firmware that has UEFI Secure Boot enabled.

The `qemu-ovmf-x86_64` package contains several UEFI firmware images. For example, the following subset supports SMM, UEFI Secure Boot, and has either Microsoft, openSUSE or SUSE UEFI CA keys enrolled:

```
# rpm -ql qemu-ovmf-x86_64
[...]
/usr/share/qemu/ovmf-x86_64-smm-ms-code.bin
/usr/share/qemu/ovmf-x86_64-smm-ms-vars.bin
/usr/share/qemu/ovmf-x86_64-smm-opensuse-code.bin
/usr/share/qemu/ovmf-x86_64-smm-opensuse-vars.bin
/usr/share/qemu/ovmf-x86_64-smm-suse-code.bin
/usr/share/qemu/ovmf-x86_64-smm-suse-vars.bin
[...]
```

For the AArch64 architecture, the package is named `qemu-uefi-aarch32`:

```
# rpm -ql qemu-uefi-aarch32
[...]
/usr/share/qemu/aavmf-aarch32-code.bin
/usr/share/qemu/aavmf-aarch32-vars.bin
/usr/share/qemu/firmware
/usr/share/qemu/firmware/60-aavmf-aarch32.json
/usr/share/qemu/qemu-uefi-aarch32.bin
```

The `*-code.bin` files are the UEFI firmware files. The `*-vars.bin` files are corresponding variable store images that can be used as a template for a per-VM non-volatile store. `libvirt` copies the specified `vars` template to a per-VM path under `/var/lib/libvirt/qemu/nvram/` when first creating the VM. Files without `code` or `vars` in the name can be used as a single UEFI image. They are not as useful, since no UEFI variables persist across power cycles of the VM.

The `*-ms*.bin` files contain UEFI CA keys as found on real hardware. Therefore, they are configured as the default in `libvirt`. Likewise, the `*-suse*.bin` files contain preinstalled SUSE keys. There is also a set of files with no preinstalled keys.

For more details on OVMF, see http://www.linux-kvm.org/downloads/lersek/ovmf-whitepaper-c770f8c.txt ↗.

3.3.2 Memory ballooning with Windows guests

Memory ballooning is a method to change the amount of memory used by VM Guest at runtime. KVM hypervisors provides this method, but it needs to be supported by the guest as well.

While openSUSE and SLES-based guests support memory ballooning, Windows guests need the Virtual Machine Driver Pack (VMDP) (https://www.suse.com/products/vmdriverpack/) ↗ to provide ballooning. To set the maximum memory greater than the initial memory configured for Windows guests, follow these steps:

1. Install the Windows guest with the maximum memory equal or less than the initial value.
2. Install the Virtual Machine Driver Pack in the Windows guest to provide required drivers.
3. Shut down the Windows guest.
4. Reset the maximum memory of the Windows guest to the required value.
5. Start the Windows guest again.

3.3.3 Including add-on products in the installation

Certain operating systems, such as SUSE Linux Enterprise Server, offer to include add-on products in the installation process. If the add-on product installation source is provided via SUSE Customer Center, no special VM Guest configuration is needed. If it is provided via CD/DVD or ISO image, it is necessary to provide the VM Guest installation system with both the standard installation medium image and the image of the add-on product.

If you are using the GUI-based installation, select *Customize Configuration Before Install* in the last step of the wizard and add the add-on product ISO image via *Add Hardware* › *Storage*. Specify the path to the image and set the *Device Type* to *CD-ROM*.

If you are installing from the command line, you need to set up the virtual CD/DVD drives with the `--disk` parameter rather than with `--cdrom`. The device that is specified first is used for booting.

4 Configuring virtual machines with **`virsh`**

You can use **`virsh`** to configure virtual machines (VM) on the command line as an alternative to using the Virtual Machine Manager. With **`virsh`**, you can control the state of a VM, edit the configuration of a VM or even migrate a VM to another host. The following sections describe how to manage VMs by using **`virsh`**.

4.1 Editing the VM configuration

The configuration of a VM is stored in an XML file in `/etc/libvirt/qemu/` and looks like this:

EXAMPLE 7: EXAMPLE XML CONFIGURATION FILE

```
<domain type='kvm'>
  <name>sles15</name>
  <uuid>ab953e2f-9d16-4955-bb43-1178230ee625</uuid>
  <memory unit='KiB'>2097152</memory>
  <currentMemory unit='KiB'>2097152</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <os>
    <type arch='x86_64' machine='pc-q35-2.0'>hvm</type>
  </os>
  <features>...</features>
  <cpu mode='custom' match='exact' check='partial'>
    <model fallback='allow'>Skylake-Client-IBRS</model>
  </cpu>
  <clock>...</clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <pm>
    <suspend-to-mem enabled='no'/>
    <suspend-to-disk enabled='no'/>
  </pm>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='disk'>...</disk>
  </devices>
  ...
</domain>
```

To edit the configuration of a VM Guest, check if it is offline:

```
> sudo virsh list --inactive
```

If your VM Guest is in this list, you can safely edit its configuration:

```
> sudo virsh edit NAME_OF_VM_GUEST
```

Before saving the changes, **virsh** validates your input against a RelaxNG schema.

4.2 Changing the machine type

When installing with the **virt-install** tool, the machine type for a VM Guest is *pc-q35* by default. The machine type is stored in the VM Guest's configuration file in the `type` element:

```
<type arch='x86_64' machine='pc-q35-2.3'>hvm</type>
```

As an example, the following procedure shows how to change this value to the machine type `q35`. The value `q35` is an Intel* chipset and includes *PCIe*, supports up to 12 USB ports, and has support for *SATA* and *IOMMU*.

PROCEDURE 2: CHANGING MACHINE TYPE

1. Check whether your VM Guest is inactive:

   ```
   > sudo virsh list --inactive
   Id    Name                           State
   ----------------------------------------------------
   -     sles15                         shut off
   ```

2. Edit the configuration for this VM Guest:

   ```
   > sudo virsh edit sles15
   ```

3. Replace the value of the `machine` attribute with `pc-q35-2.0` :

   ```
   <type arch='x86_64' machine='pc-q35-2.0'>hvm</type>
   ```

4. Restart the VM Guest:

```
> sudo virsh start sles15
```

5. Check if the machine type has changed. Log in to the VM Guest and run the following command:

```
> sudo dmidecode | grep Product
Product Name: Standard PC (Q35 + ICH9, 2009)
```

Tip: Machine type update recommendations

Whenever the QEMU version on the host system is upgraded, for example, when upgrading the VM Host Server to a new service pack, upgrade the machine type of the VM Guests to the latest available version. To check, use the command **`qemu-system-x86_64 -M help`** on the VM Host Server.

The default machine type `pc-i440fx`, for example, is regularly updated. If your VM Guest still runs with a machine type of `pc-i440fx-1.`*X*, we strongly recommend an update to `pc-i440fx-2.`*X*. This allows taking advantage of the most recent updates and corrections in machine definitions, and ensures better future compatibility.

4.3 Configuring hypervisor features

`libvirt` automatically enables a default set of hypervisor features that are sufficient in most circumstances, but also allows enabling and disabling features as needed. Hypervisor features can be configured with **`virsh`**. Look for the `<features>` element in the VM Guest's configuration file and adjust its features as required.

See the *Hypervisor features* section of the libvirt *Domain XML format* manual at https://libvirt.org/formatdomain.html#elementsFeatures for more information.

4.4 Configuring CPU

Many aspects of the virtual CPUs presented to VM Guests are configurable with **`virsh`**. The number of current and maximum CPUs allocated to a VM Guest can be changed, as well as the model of the CPU and its feature set. The following subsections describe how to change the common CPU settings of a VM Guest.

4.4.1 Configuring the number of CPUs

The number of allocated CPUs is stored in the VM Guest's XML configuration file in `/etc/libvirt/qemu/` in the `vcpu` element:

```
<vcpu placement='static'>1</vcpu>
```

In this example, the VM Guest has only one allocated CPU. The following procedure shows how to change the number of allocated CPUs for the VM Guest:

1. Check whether your VM Guest is inactive:

   ```
   > sudo virsh list --inactive
   Id    Name                           State
   ----------------------------------------------------
   -     sles15                         shut off
   ```

2. Edit the configuration for an existing VM Guest:

   ```
   > sudo virsh edit sles15
   ```

3. Change the number of allocated CPUs:

   ```
   <vcpu placement='static'>2</vcpu>
   ```

4. Restart the VM Guest:

   ```
   > sudo virsh start sles15
   ```

5. Check if the number of CPUs in the VM has changed.

   ```
   > sudo virsh vcpuinfo sled15
   VCPU:           0
   CPU:            N/A
   State:          N/A
   CPU time        N/A
   CPU Affinity:   yy

   VCPU:           1
   CPU:            N/A
   State:          N/A
   CPU time        N/A
   CPU Affinity:   yy
   ```

You can also change the number of CPUs while the VM Guest is running. CPUs can be hotplugged until the maximum number configured at VM Guest start is reached. Likewise, they can be hot-unplugged until the lower limit of 1 is reached. The following example shows changing the active CPU count from 2 to a predefined maximum of 4.

1. Check the current live vcpu count:

   ```
   > sudo virsh vcpucount sles15 | grep live
   maximum      live           4
   current      live           2
   ```

2. Change the current, or active, number of CPUs to 4:

   ```
   > sudo virsh setvcpus sles15 --count 4 --live
   ```

3. Check that the current live vcpu count is now 4:

   ```
   > sudo virsh vcpucount sles15 | grep live
   maximum      live           4
   current      live           4
   ```

4.4.2 Configuring the CPU model

The CPU model exposed to a VM Guest can often influence the workload running within it. The default CPU model is derived from a CPU mode known as `host-model`.

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

When starting a VM Guest with the CPU mode `host-model`, `libvirt` copies its model of the host CPU into the VM Guest definition. The host CPU model and features copied to the VM Guest definition can be observed in the output of the **`virsh capabilities`**.

Another interesting CPU mode is `host-passthrough`.

```
<cpu mode='host-passthrough'/>
```

When starting a VM Guest with the CPU mode `host-passthrough`, it is presented with a CPU that is exactly the same as the VM Host Server CPU. This can be useful when the VM Guest workload requires CPU features not available in `libvirt`'s simplified `host-model` CPU. The `host-passthrough` CPU mode comes with the disadvantage of reduced migration flexibility. A VM Guest with `host-passthrough` CPU mode can only be migrated to a VM Host Server with identical hardware.

When using the `host-passthrough` CPU mode, it is still possible to disable undesirable features. The following configuration presents the VM Guest with a CPU that is exactly the same as the host CPU but with the `vmx` feature disabled.

```
<cpu mode='host-passthrough'>
  <feature policy='disable' name='vmx'/>
  </cpu>
```

The `custom` CPU mode is another common mode used to define a normalized CPU that can be migrated throughout dissimilar hosts in a cluster. For example, in a cluster with hosts containing Nehalem, IvyBridge and SandyBridge CPUs, the VM Guest can be configured with a `custom` CPU mode that contains a Nehalem CPU model.

```
<cpu mode='custom' match='exact'>
  <model fallback='allow'>Nehalem</model>
  <feature policy='require' name='vme'/>
  <feature policy='require' name='ds'/>
  <feature policy='require' name='acpi'/>
  <feature policy='require' name='ss'/>
  <feature policy='require' name='ht'/>
  <feature policy='require' name='tm'/>
  <feature policy='require' name='pbe'/>
  <feature policy='require' name='dtes64'/>
  <feature policy='require' name='monitor'/>
  <feature policy='require' name='ds_cpl'/>
  <feature policy='require' name='vmx'/>
  <feature policy='require' name='est'/>
  <feature policy='require' name='tm2'/>
  <feature policy='require' name='xtpr'/>
  <feature policy='require' name='pdcm'/>
  <feature policy='require' name='dca'/>
  <feature policy='require' name='rdtscp'/>
  <feature policy='require' name='invtsc'/>
  </cpu>
```

For more information on `libvirt`'s CPU model and topology options, see the *CPU model and topology* documentation at https://libvirt.org/formatdomain.html#cpu-model-and-topology ↗.

4.5 Changing boot options

The boot menu of the VM Guest can be found in the `os` element and looks similar to this example:

```
<os firmware='efi'>
```

```
    <type arch='x86_64' machine='pc-q35-10.0'>hvm</type>
    <firmware>
      <feature enabled='yes' name='enrolled-keys'/>
      <feature enabled='yes' name='secure-boot'/>
    </firmware>
    <loader readonly='yes' secure='yes' type='pflash' format='raw'>/usr/share/qemu/ovmf-
x86_64-smm-ms-code.bin</loader>
    <nvram template='/usr/share/qemu/ovmf-x86_64-smm-ms-vars.bin' templateFormat='raw'
 format='raw'>/var/lib/libvirt/qemu/nvram/win11_VARS.fd</nvram>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
  </os>
```

In this example, two devices are available, `hd` and `cdrom` . The configuration also reflects the actual boot order, so the `hd` comes before the `cdrom` .

4.5.1 Changing boot order

The VM Guest's boot order is represented through the order of devices in the XML configuration file. As the devices are interchangeable, it is possible to change the boot order of the VM Guest.

1. Open the VM Guest's XML configuration.

   ```
   > sudo virsh edit sles15
   ```

2. Change the sequence of the bootable devices.

   ```
   ...
   <boot dev='cdrom'/>
   <boot dev='hd'/>
   ...
   ```

3. Check if the boot order was changed successfully by looking at the boot menu in the BIOS of the VM Guest.

4.5.2 Using direct kernel boot

Direct Kernel Boot allows you to boot from a kernel and initrd stored on the host. Set the path to both files in the `kernel` and `initrd` elements:

```
<os>
    ...
  <kernel>/root/f8-i386-vmlinuz</kernel>
```

```
  <initrd>/root/f8-i386-initrd</initrd>
    ...
<os>
```

To enable Direct Kernel Boot:

1. Open the VM Guest's XML configuration:

   ```
   > sudo virsh edit sles15
   ```

2. Inside the `os` element, add a `kernel` element and the path to the kernel file on the host:

   ```
   ...
   <kernel>/root/f8-i386-vmlinuz</kernel>
   ...
   ```

3. Add an `initrd` element and the path to the initrd file on the host:

   ```
   ...
   <initrd>/root/f8-i386-initrd</initrd>
   ...
   ```

4. Start your VM to boot from the new kernel:

   ```
   > sudo virsh start sles15
   ```

4.6 Configuring memory allocation

The amount of memory allocated for the VM Guest can also be configured with **`virsh`**. It is stored in the `memory` element and defines the maximum allocation of memory for the VM Guest at boot time. The optional `currentMemory` element defines the actual memory allocated to the VM Guest. `currentMemory` can be less than `memory`, allowing for increasing (or *ballooning*) the memory while the VM Guest is running. If `currentMemory` is omitted, it defaults to the same value as the `memory` element.

You can adjust memory settings by editing the VM Guest configuration, but be aware that changes do not take place until the next boot. The following steps demonstrate changing a VM Guest to boot with 4G of memory, but allow later expansion to 8G:

1. Open the VM Guest's XML configuration:

   ```
   > sudo virsh edit sles15
   ```

2. Search for the `memory` element and set to 8G:

   ```
   ...
   <memory unit='KiB'>8388608</memory>
   ...
   ```

3. If the `currentMemory` element does not exist, add it below the `memory` element, or change its value to 4G:

   ```
   [...]
   <memory unit='KiB'>8388608</memory>
   <currentMemory unit='KiB'>4194304</currentMemory>
   [...]
   ```

Changing the memory allocation while the VM Guest is running can be done with the **`setmem`** subcommand. The following example shows increasing the memory allocation to 8G:

1. Check VM Guest existing memory settings:

   ```
   > sudo virsh dominfo sles15 | grep memory
   Max memory:     8388608 KiB
   Used memory:    4194608 KiB
   ```

2. Change the used memory to 8G:

   ```
   > sudo virsh setmem sles15 8388608
   ```

3. Check the updated memory settings:

   ```
   > sudo virsh dominfo sles15 | grep memory
   Max memory:     8388608 KiB
   Used memory:    8388608 KiB
   ```

Important: Large memory VM Guests

VM Guests with memory requirements of 4 TB or more must either use the `host-passthrough` CPU mode, or explicitly specify the virtual CPU address size when using `host-model` or `custom` CPU modes. The default virtual CPU address size may not be sufficient for memory configurations of 4 TB or more. The following example shows how to use the VM Host Server's physical CPU address size when using the `host-model` CPU mode.

```
[...]
<cpu mode='host-model' check='partial'>
```

```
<maxphysaddr mode='passthrough'>
</cpu>
[...]
```

For more information on specifying virtual CPU address size, see the `maxphysaddr` option in the *CPU model and topology* documentation at https://libvirt.org/formatdomain.html#cpu-model-and-topology.

4.7 Adding a PCI device

To assign a PCI device to VM Guest with **`virsh`**, follow these steps:

1. Identify the host PCI device to assign to the VM Guest. In the following example, we are assigning a DEC network card to the guest:

   ```
   > sudo lspci -nn
   [...]
   03:07.0 Ethernet controller [0200]: Digital Equipment Corporation DECchip \
   21140 [FasterNet] [1011:0009] (rev 22)
   [...]
   ```

   Write down the device ID, `03:07.0` in this example.

2. Gather detailed information about the device using **`virsh nodedev-dumpxml`** *`ID`*. To get the *`ID`*, replace the colon and the period in the device ID (`03:07.0`) with underscores. Prefix the result with “pci_0000_”: `pci_0000_03_07_0`.

   ```
   > sudo virsh nodedev-dumpxml pci_0000_03_07_0
   <device>
     <name>pci_0000_03_07_0</name>
     <path>/sys/devices/pci0000:00/0000:00:14.4/0000:03:07.0</path>
     <parent>pci_0000_00_14_4</parent>
     <driver>
       <name>tulip</name>
     </driver>
     <capability type='pci'>
       <domain>0</domain>
       <bus>3</bus>
       <slot>7</slot>
       <function>0</function>
       <product id='0x0009'>DECchip 21140 [FasterNet]</product>
       <vendor id='0x1011'>Digital Equipment Corporation</vendor>
       <numa node='0'/>
   ```

```
  </capability>
</device>
```

Write down the values for domain, bus and function (see the previous XML code printed in bold).

3. Detach the device from the host system before attaching it to the VM Guest:

```
> sudo virsh nodedev-detach pci_0000_03_07_0
  Device pci_0000_03_07_0 detached
```

Tip: Multi-function PCI devices

When using a multi-function PCI device that does not support FLR (function level reset) or PM (power management) reset, you need to detach all its functions from the VM Host Server. The whole device must be reset for security reasons. `libvirt` refuses to assign the device if one of its functions is still in use by the VM Host Server or another VM Guest.

4. Convert the domain, bus, slot, and function value from decimal to hexadecimal. In our example, domain = 0, bus = 3, slot = 7, and function = 0. Ensure that the values are inserted in the right order:

```
> printf "<address domain='0x%x' bus='0x%x' slot='0x%x' function='0x%x'/>\n" 0 3 7 0
```

This results in:

```
<address domain='0x0' bus='0x3' slot='0x7' function='0x0'/>
```

5. Run **`virsh edit`** on your domain, and add the following device entry in the `<devices>` section using the result from the previous step:

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0' bus='0x03' slot='0x07' function='0x0'/>
  </source>
</hostdev>
```

Tip: managed compared to unmanaged

`libvirt` recognizes two modes for handling PCI devices: they can be `managed` or `unmanaged`. In the managed case, `libvirt` handles all details of unbinding the device from the existing driver if needed, resetting the device, binding it to `vfio-pci` before starting the domain, etc. When the domain is terminated or the device is removed from the domain, `libvirt` unbinds from `vfio-pci` and rebinds to the original driver when using a managed device. If the device is unmanaged, the user must ensure that all these management aspects of the device are done before assigning it to a domain, and after the device is no longer used by the domain.

In the example above, the `managed='yes'` option means that the device is managed. To switch the device mode to unmanaged, set `managed='no'` in the listing above. If you do so, you need to take care of the related driver with the **`virsh nodedev-detach`** and **`virsh nodedev-reattach`** commands. Before starting the VM Guest, you need to detach the device from the host by running **`virsh nodedev-detach pci_0000_03_07_0`**. In case the VM Guest is not running, you can make the device available for the host by running **`virsh nodedev-reattach pci_0000_03_07_0`**.

6. Shut down the VM Guest and disable SELinux if it is running on the host.

```
> sudo setsebool -P virt_use_sysfs 1
```

7. Start your VM Guest to make the assigned PCI device available:

```
> sudo virsh start sles15
```

Important: SLES11 SP4 KVM guests

On a newer QEMU machine type (pc-i440fx-2.0 or higher) with SLES 11 SP4 KVM guests, the `acpiphp` module is not loaded by default in the guest. This module must be loaded to enable hotplugging of disk and network devices. To load the module manually, use the command **`modprobe acpiphp`**. It is also possible to autoload the module by adding `install acpiphp /bin/true` to the `/etc/modprobe.conf.local` file.

Important: KVM guests using QEMU Q35 machine type

KVM guests using the QEMU Q35 machine type have a PCI topology that includes a `pcie-root` controller and seven `pcie-root-port` controllers. The `pcie-root` controller does not support hotplugging. Each `pcie-root-port` controller supports hotplugging a single PCIe device. PCI controllers cannot be hotplugged, so plan accordingly and add more `pcie-root-ports` to hotplug more than seven PCIe devices. A `pcie-to-pci-bridge` controller can be added to support hotplugging legacy PCI devices. See https://libvirt.org/pci-hotplug.html for more information about PCI topology between QEMU machine types.

4.7.1 PCI Pass-Through for IBM Z

To support IBM Z, QEMU extended PCI representation by allowing the user to configure extra attributes. Two more attributes—`uid` and `fid`—were added to the `<zpci/>` `libvirt` specification. `uid` represents user-defined identifier, while `fid` represents PCI function identifier. These attributes are optional and if you do not specify them, they are automatically generated with non-conflicting values.

To include zPCI attribute in your domain specification, use the following example definition:

```
<controller type='pci' index='0' model='pci-root'/>
<controller type='pci' index='1' model='pci-bridge'>
  <model name='pci-bridge'/>
  <target chassisNr='1'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0'>
    <zpci uid='0x0001' fid='0x00000000'/>
  </address>
</controller>
<interface type='bridge'>
  <source bridge='virbr0'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x01' slot='0x01' function='0x0'>
    <zpci uid='0x0007' fid='0x00000003'/>
  </address>
</interface>
```

4.8 Adding a USB device

To assign a USB device to VM Guest using **virsh**, follow these steps:

1. Identify the host USB device to assign to the VM Guest:

```
> sudo lsusb
[...]
Bus 001 Device 003: ID 0557:2221 ATEN International Co., Ltd Winbond Hermon
[...]
```

   Write down the vendor and product IDs. In our example, the vendor ID is `0557` and the product ID is `2221`.

2. Run **virsh edit** on your domain, and add the following device entry in the `<devices>` section using the values from the previous step:

```
<hostdev mode='subsystem' type='usb'>
  <source startupPolicy='optional'>
   <vendor id='0557'/>
   <product id='2221'/>
  </source>
</hostdev>
```

   Tip: Vendor/product or device's address

   Instead of defining the host device with `<vendor/>` and `<product/>` IDs, you can use the `<address/>` element as described for host PCI devices in *Section 4.7, "Adding a PCI device"*.

3. Shut down the VM Guest and disable SELinux if it is running on the host:

```
> sudo setsebool -P virt_use_sysfs 1
```

4. Start your VM Guest to make the assigned PCI device available:

```
> sudo virsh start sles15
```

4.9 Adding SR-IOV devices

Single Root I/O Virtualization capable *PCIe* devices can replicate their resources, so they appear as multiple devices. Each of these "pseudo-devices" can be assigned to a VM Guest.

SR-IOV; is an industry specification that was created by the Peripheral Component Interconnect Special Interest Group (PCI-SIG) consortium. It introduces physical functions (PF) and virtual functions (VF). PFs are full *PCIe* functions used to manage and configure the device. PFs also can move data. VFs lack the configuration and management part—they only can move data and a reduced set of configuration functions. As VFs do not have all *PCIe* functions, the host operating system or the *Hypervisor* must support *SR-IOV*; to access and initialize VFs. The theoretical maximum for VFs is 256 per device (consequently the maximum for a dual-port Ethernet card would be 512). In practice, this maximum is much lower, since each VF consumes resources.

4.9.1 Requirements

The following requirements must be met to use Single Root I/O Virtualization:

- An Single Root I/O Virtualization capable network card (as of SUSE Linux Enterprise Server 15, only network cards support Single Root I/O Virtualization). For more information, refer to *SR-IOV*.
- An AMD64/Intel 64 host supporting hardware virtualization (AMD-V or Intel VT-x). For more information, see the section *Architecture Support* in the article Virtualization Limits and Support (https://documentation.suse.com/sles/16.0/html/SLES-virtualization-support/) ↗.
- A chipset that supports device assignment (AMD-Vi or Intel *VT-d*)
- `libvirt` 0.9.10 or better
- *SR-IOV*; drivers must be loaded and configured on the host system
- A host configuration that meets the requirements
- A list of the PCI addresses of the VFs assigned to VM Guests

Tip: Checking if a device is SR-IOV-capable

The information whether a device is SR-IOV-capable can be obtained from its PCI descriptor by running **`lspci`**. A device that supports *SR-IOV*; reports a capability similar to the following:

```
Capabilities: [160 v1] Single Root I/O Virtualization
```

Note: Adding an SR-IOV device at VM Guest creation

Before adding an SR-IOV device to a VM Guest when initially setting it up, the VM Host Server already needs to be configured as described in *Section 4.9.2, "Loading and configuring the SR-IOV host drivers"*.

4.9.2 Loading and configuring the SR-IOV host drivers

To access and initialize VFs, an SR-IOV-capable driver needs to be loaded on the host system.

1. Before loading the driver, make sure the card is properly detected by running **`lspci`**. The following example shows the **`lspci`** output for the dual-port Intel 82576NS network card:

   ```
   > sudo /sbin/lspci | grep 82576
   01:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   01:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   04:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   04:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   ```

   In case the card is not detected, the hardware virtualization support in the BIOS/EFI may not have been enabled. To check if hardware virtualization support is enabled, look at the settings in the host's BIOS.

2. Check whether the *SR-IOV*; driver is already loaded by running **`lsmod`**. In the following example, a check for the igb driver (for the Intel 82576NS network card) returns a result. That means the driver is already loaded. If the command returns nothing, the driver is not loaded.

   ```
   > sudo /sbin/lsmod | egrep "^igb "
   igb                    185649  0
   ```

3. Skip the following step if the driver is already loaded. If the *SR-IOV*; driver is not yet loaded, the non-*SR-IOV*; driver needs to be removed first, before loading the new driver. Use **`rmmod`** to unload a driver. The following example unloads the non-*SR-IOV*; driver for the Intel 82576NS network card:

   ```
   > sudo /sbin/rmmod igbvf
   ```

4. Load the *SR-IOV*; driver subsequently using the **`modprobe`** command—the VF parameter (`max_vfs`) is mandatory:

```
> sudo /sbin/modprobe igb max_vfs=8
```

As an alternative, you can also load the driver via SYSFS:

1. Find the PCI ID of the physical NIC by listing Ethernet devices:

```
> sudo lspci | grep Eth
06:00.0 Ethernet controller: Emulex Corporation OneConnect NIC (Skyhawk) (rev 10)
06:00.1 Ethernet controller: Emulex Corporation OneConnect NIC (Skyhawk) (rev 10)
```

2. To enable VFs, echo the number of desired VFs to load to the `sriov_numvfs` parameter:

```
> sudo echo 1 > /sys/bus/pci/devices/0000:06:00.1/sriov_numvfs
```

3. Verify that the VF NIC was loaded:

```
> sudo lspci | grep Eth
06:00.0 Ethernet controller: Emulex Corporation OneConnect NIC (Skyhawk) (rev 10)
06:00.1 Ethernet controller: Emulex Corporation OneConnect NIC (Skyhawk) (rev 10)
06:08.0 Ethernet controller: Emulex Corporation OneConnect NIC (Skyhawk) (rev 10)
```

4. Obtain the maximum number of VFs available:

```
> sudo lspci -vvv -s 06:00.1 | grep 'Initial VFs'
                       Initial VFs: 32, Total VFs: 32, Number of VFs: 0,
Function Dependency Link: 01
```

5. Create a `/etc/systemd/system/before.service` file which loads VF via SYSFS on boot:

```
[Unit]
Before=
[Service]
Type=oneshot
RemainAfterExit=true
ExecStart=/bin/bash -c "echo 1 > /sys/bus/pci/devices/0000:06:00.1/sriov_numvfs"
# beware, executable is run directly, not through a shell, check the man pages
# systemd.service and systemd.unit for full syntax
[Install]
# target in which to start the service
WantedBy=multi-user.target
#WantedBy=graphical.target
```

6. Before starting the VM, it is required to create another service file (`after-local.service`) pointing to the `/etc/init.d/after.local` script that detaches the NIC. Otherwise the VM would fail to start:

   ```
   [Unit]
   Description=/etc/init.d/after.local Compatibility
   After=libvirtd.service
   Requires=libvirtd.service
   [Service]
   Type=oneshot
   ExecStart=/etc/init.d/after.local
   RemainAfterExit=true

   [Install]
   WantedBy=multi-user.target
   ```

7. Copy it to `/etc/systemd/system`.

   ```
   #! /bin/sh
   # ...
   virsh nodedev-detach pci_0000_06_08_0
   ```

   Save it as `/etc/init.d/after.local`.

8. Reboot the machine and check if the SR-IOV driver is loaded by re-running the **`lspci`** command from the first step of this procedure. If the SR-IOV driver was loaded successfully you should see additional lines for the VFs:

   ```
   01:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   01:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   01:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   01:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   01:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   [...]
   04:00.0 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   04:00.1 Ethernet controller: Intel Corporation 82576NS Gigabit Network Connection
    (rev 01)
   04:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   04:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   04:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
   [...]
   ```

4.9.3 Adding a VF network device to a VM Guest

When the *SR-IOV*; hardware is properly set up on the VM Host Server, you can add VFs to VM Guests. To do so, you need to collect specific data first.

PROCEDURE 3: ADDING A VF NETWORK DEVICE TO AN EXISTING VM GUEST

The following procedure uses example data. Replace it with appropriate data from your setup.

1. Use the **`virsh nodedev-list`** command to get the PCI address of the VF you want to assign and its corresponding PF. Numerical values from the **`lspci`** output shown in *Section 4.9.2, “Loading and configuring the SR-IOV host drivers”*, for example, `01:00.0` or `04:00.1`, are transformed by adding the prefix `pci_0000_` and by replacing colons and dots with underscores. So a PCI ID listed as `04:00.0` by **`lspci`** is listed as `pci_0000_04_00_0` by virsh. The following example lists the PCI IDs for the second port of the Intel 82576NS network card:

```
> sudo virsh nodedev-list | grep 0000_04_
pci_0000_04_00_0
pci_0000_04_00_1
pci_0000_04_10_0
pci_0000_04_10_1
pci_0000_04_10_2
pci_0000_04_10_3
pci_0000_04_10_4
pci_0000_04_10_5
pci_0000_04_10_6
pci_0000_04_10_7
pci_0000_04_11_0
pci_0000_04_11_1
pci_0000_04_11_2
pci_0000_04_11_3
pci_0000_04_11_4
pci_0000_04_11_5
```

The first two entries represent the **PFs**, whereas the other entries represent the VFs.

2. Run the following **`virsh nodedev-dumpxml`** command on the PCI ID of the VF you want to add:

```
> sudo virsh nodedev-dumpxml pci_0000_04_10_0
<device>
  <name>pci_0000_04_10_0</name>
  <parent>pci_0000_00_02_0</parent>
  <capability type='pci'>
    <domain>0</domain>
    <bus>4</bus>
    <slot>16</slot>
    <function>0</function>
    <product id='0x10ca'>82576 Virtual Function</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='phys_function'>
      <address domain='0x0000' bus='0x04' slot='0x00' function='0x0'/>
    </capability>
  </capability>
</device>
```

The following data is needed for the next step:

- `<domain>0</domain>`
- `<bus>4</bus>`
- `<slot>16</slot>`
- `<function>0</function>`

3. Create a temporary XML file, for example, `/tmp/vf-interface.xml`, containing the data necessary to add a VF network device to an existing VM Guest. The minimal content of the file needs to look like the following:

```
<interface type='hostdev'>1
 <source>
  <address type='pci' domain='0' bus='11' slot='16' function='0'2/>2
 </source>
</interface>
```

1 VFs do not get a fixed MAC address; it changes every time the host reboots. When adding network devices the “traditional” way with `hostdev`, it would require to re-configure the VM Guest's network device after each reboot of the host, because of the MAC address change. To avoid this kind of problem, `libvirt` introduced the `hostdev` value, which sets up network-specific data *before* assigning the device.

2 Specify the data you acquired in the previous step here.

4. In case a device is already attached to the host, it cannot be attached to a VM Guest. To make it available for guests, detach it from the host first:

```
> sudo virsh nodedev-detach pci_0000_04_10_0
```

5. Add the VF interface to an existing VM Guest:

```
> sudo virsh attach-device GUEST /tmp/vf-interface.xml --OPTION
```

GUEST needs to be replaced by the domain name, ID or UUID of the VM Guest. --*OPTION* can be one of the following:

`--persistent`
: This option always adds the device to the domain's persistent XML. If the domain is running, the device is hotplugged.

`--config`
: This option affects the persistent XML only, even if the domain is running. The device appears in the VM Guest on next boot.

`--live`
: This option affects a running domain only. If the domain is inactive, the operation fails. The device is not persisted in the XML and becomes available in the VM Guest on next boot.

`--current`
: This option affects the current state of the domain. If the domain is inactive, the device is added to the persistent XML and becomes available on next boot. If the domain is active, the device is hotplugged but not added to the persistent XML.

6. To detach a VF interface, use the **`virsh detach-device`** command, which also takes the options listed above.

4.9.4 Dynamic allocation of VFs from a pool

If you define the PCI address of a VF into a VM Guest's configuration statically as described in *Section 4.9.3, "Adding a VF network device to a VM Guest"*, it is hard to migrate such guest to another host. The host must have identical hardware in the same location on the PCI bus, or the VM Guest configuration must be modified before each start.

Another approach is to create a `libvirt` network with a device pool that contains all the VFs of an *SR-IOV*; device. The VM Guest then references this network, and each time it is started, a single VF is dynamically allocated to it. When the VM Guest is stopped, the VF is returned to the pool, available for another guest.

4.9.4.1 Defining network with pool of VFs on VM Host Server

The following example of network definition creates a pool of all VFs for the *SR-IOV*; device with its physical function (PF) at the network interface `eth0` on the host:

```
<network>
  <name>passthrough</name>
    <forward mode='hostdev' managed='yes'>
      <pf dev='eth0'/>
    </forward>
  </network>
```

To use this network on the host, save the above code to a file, for example `/tmp/passthrough.xml`, and execute the following commands. Remember to replace `eth0` with the real network interface name of your *SR-IOV*; device's PF:

```
> sudo virsh net-define /tmp/passthrough.xml
> sudo virsh net-autostart passthrough
> sudo virsh net-start passthrough
```

4.9.4.2 Configuring VM Guests to use VF from the pool

The following example of VM Guest device interface definition uses a VF of the *SR-IOV*; device from the pool created in *Section 4.9.4.1, "Defining network with pool of VFs on VM Host Server"*. `libvirt` automatically derives the list of all VFs associated with that PF the first time the guest is started.

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

After the first VM Guest starts that uses the network with the pool of VFs, verify the list of associated VFs. Do so by running **`virsh net-dumpxml passthrough`** on the host.

```
<network connections='1'>
  <name>passthrough</name>
  <uuid>a6a26429-d483-d4ed-3465-4436ac786437</uuid>
  <forward mode='hostdev' managed='yes'>
    <pf dev='eth0'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x1'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x5'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x7'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x1'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x5'/>
  </forward>
  </network>
```

4.10 Listing attached devices

Although there is no mechanism in **virsh** to list all VM Host Server's devices that have already been attached to its VM Guests, you can list all devices attached to a specific VM Guest by running the following command:

```
virsh dumpxml VMGUEST_NAME | xpath -e /domain/devices/hostdev
```

For example:

```
> sudo virsh dumpxml sles16-clone | xpath  -e /domain/devices/hostdev
Found 1 nodes in stdin:
-- NODE --
<hostdev mode="subsystem" type="pci" managed="yes">
  <source>
        <address domain="0x0000" bus="0x01" slot="0x00" function="0x0" />
  </source>
      <address type="pci" domain="0x0000" bus="0x07" slot="0x00" function="0x0" />
</hostdev>
```

Tip: Listing SR-IOV devices attached via `<interface type='hostdev'>`

For SR-IOV devices that are attached to the VM Host Server via `<interface type='hostdev'>`, you need to use a different XPath query:

```
virsh dumpxml VMGUEST_NAME | xpath -e /domain/devices/interface/@type
```

4.11 Configuring storage devices

Storage devices are defined within the disk element. The usual disk element supports several attributes. The following two attributes are the most important:

- The type attribute describes the source of the virtual disk device. Valid values are file , block , dir , network , or volume .
- The device attribute shows how the disk is exposed to the VM Guest OS. As an example, possible values can include floppy , disk , cdrom , and others.

The following child elements are the most important:

- driver contains the driver and the bus. These are used by the VM Guest to work with the new disk device.
- The target element contains the device name under which the new disk is shown in the VM Guest. It also contains the optional bus attribute, which defines the type of bus on which the new disk should operate.

The following procedure shows how to add storage devices to the VM Guest:

1. Edit the configuration for an existing VM Guest:

```
> sudo virsh edit sles15
```

2. Add a disk element inside the devices element together with the attributes type and device:

```
<disk type='file' device='disk'>
```

3. Specify a driver element and use the default values:

```
<driver name='qemu' type='qcow2'/>
```

4. Create a disk image as a source for the new virtual disk device:

```
> sudo qemu-img create -f qcow2 /var/lib/libvirt/images/sles15.qcow2 32G
```

5. Add the path for the disk source:

```
<source file='/var/lib/libvirt/images/sles15.qcow2'/>
```

6. Define the target device name in the VM Guest and the bus on which the disk should work:

   ```
   <target dev='vda' bus='virtio'/>
   ```

7. Restart your VM:

   ```
   > sudo virsh start sles15
   ```

Your new storage device should be available in the VM Guest OS.

4.12 Configuring controller devices

`libvirt` manages controllers automatically based on the type of virtual devices used by the VM Guest. If the VM Guest contains PCI and SCSI devices, PCI and SCSI controllers are created and managed automatically. **`libvirt`** also models controllers that are hypervisor-specific, for example, a `virtio-serial` controller for KVM. Although the default controllers and their configuration are generally fine, there may be use cases where controllers or their attributes need to be adjusted manually.

The default of 32 is enough in most circumstances, but a VM Guest with multiple I/O devices and an I/O-intensive workload may experience performance issues because of grant frame exhaustion.

See the *Controllers* section of the libvirt *Domain XML format* manual at https://libvirt.org/format-domain.html#elementsControllers ↗ for more information.

4.13 Configuring video devices

When using the Virtual Machine Manager, only the Video device model can be defined. The amount of allocated VRAM or 2D/3D acceleration can only be changed in the XML configuration.

4.13.1 Changing the amount of allocated VRAM

1. Edit the configuration for an existing VM Guest:

   ```
   > sudo virsh edit sles15
   ```

2. Change the size of the allocated VRAM:

   ```
   <video>
   ```

```
<model type='vga' vram='65535' heads='1'>
...
</model>
</video>
```

3. Check if the amount of VRAM in the VM has changed by looking at the amount in the Virtual Machine Manager.

4.13.2 Changing the state of 2D/3D acceleration

1. Edit the configuration for an existing VM Guest:

```
> sudo virsh edit sles15
```

2. To enable/disable 2D/3D acceleration, change the value of `accel3d` and `accel2d` accordingly:

```
<video>
 <model>
  <acceleration accel3d='yes' accel2d='no'>
 </model>
</video>
```

Tip: Enabling 2D/3D acceleration

Only `virtio` and `vbox` video devices are capable of 2D/3D acceleration. You cannot enable it on other video devices.

4.14 Configuring network devices

This section describes how to configure specific aspects of virtual network devices by using **`virsh`**.

Find more details about `libvirt` network interface specification in https://libvirt.org/formatdomain.html#elementsDriverBackendOptions.

4.14.1 Scaling network performance with multiqueue virtio-net

The multiqueue virtio-net feature scales the network performance by allowing the VM Guest's virtual CPUs to transfer packets in parallel.

To enable multiqueue virtio-net for a specific VM Guest, edit its XML configuration as described in *Section 4.1, “Editing the VM configuration”* and modify its network interface as follows:

```
<interface type='network'>
 [...]
 <model type='virtio'/>
 <driver name='vhost' queues='NUMBER_OF_QUEUES'/>
</interface>
```

4.15 Using macvtap to share VM Host Server network interfaces

Macvtap provides direct attachment of a VM Guest virtual interface to a host network interface. The macvtap-based interface extends the VM Host Server network interface and has its own MAC address on the same Ethernet segment. Typically, this is used to make both the VM Guest and the VM Host Server show up directly on the switch that the VM Host Server is connected to.

Note: Macvtap cannot be used with a Linux bridge

Macvtap cannot be used with network interfaces already connected to a Linux bridge. Before attempting to create the macvtap interface, remove the interface from the bridge.

Note: VM Guest to VM Host Server communication with macvtap

When using macvtap, a VM Guest can communicate with other VM Guests, and with other external hosts on the network. But it cannot communicate with the VM Host Server on which the VM Guest runs. This is the defined behavior of macvtap, because of the way the VM Host Server's physical Ethernet is attached to the macvtap bridge. Traffic from the VM Guest into that bridge that is forwarded to the physical interface cannot be bounced back up to the VM Host Server's IP stack. Similarly, traffic from the VM Host Server's IP stack that is sent to the physical interface cannot be bounced back up to the macvtap bridge for forwarding to the VM Guest.

Virtual network interfaces based on macvtap are supported by libvirt by specifying an interface type of `direct`. For example:

```
<interface type='direct'>
```

```
   <mac address='aa:bb:cc:dd:ee:ff'/>
   <source dev='eth0' mode='bridge'/>
   <model type='virtio'/>
   </interface>
```

The operation mode of the macvtap device can be controlled with the `mode` attribute. The following list shows its possible values and a description for each:

- `vepa`: all VM Guest packets are sent to an external bridge. Packets whose destination is a VM Guest on the same VM Host Server as where the packet originates from are sent back to the VM Host Server by the VEPA capable bridge (today's bridges are typically not VEPA capable).
- `bridge`: packets whose destination is on the same VM Host Server as where they originate from are directly delivered to the target macvtap device. Both origin and destination devices need to be in `bridge` mode for direct delivery. If either of them is in `vepa` mode, a VEPA capable bridge is required.
- `private`: all packets are sent to the external bridge and delivered to a target VM Guest on the same VM Host Server if they are sent through an external router or gateway and that device sends them back to the VM Host Server. This procedure is followed if either the source or destination device is in private mode.
- `passthrough`: a special mode that gives more power to the network interface. All packets are forwarded to the interface, allowing virtio VM Guests to change the MAC address or set promiscuous mode to bridge the interface or create VLAN interfaces on top of it. A network interface is not shareable in `passthrough` mode. Assigning an interface to a VM Guest disconnects it from the VM Host Server. For this reason SR-IOV virtual functions are often assigned to the VM Guest in `passthrough` mode.

4.16 Disabling a memory balloon device

Memory Balloon has become a default option for KVM. The device is added to the VM Guest explicitly, so you do not need to add this element in the VM Guest's XML configuration. To disable Memory Balloon in the VM Guest for any reason, set `model='none'` as shown below:

```
<devices>
   <memballoon model='none'/>
</device>
```

4.17 Configuring multiple monitors (dual head)

`libvirt` supports a dual head configuration to display the video output of the VM Guest on multiple monitors.

PROCEDURE 4: CONFIGURING DUAL HEAD

1. While the virtual machine is running, verify that the `xf86-video-qxl` package is installed in the VM Guest:

```
> rpm -q xf86-video-qxl
```

2. Shut down the VM Guest and start editing its configuration XML as described in *Section 4.1, “Editing the VM configuration”*.

3. Verify that the model of the virtual graphics card is “qxl”:

```
<video>
 <model type='qxl' ... />
```

4. Increase the `heads` parameter in the graphics card model specification from the default `1` to `2`, for example:

```
<video>
 <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='2' primary='yes'/>
 <alias name='video0'/>
 <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0'/>
</video>
```

5. Configure the virtual machine to use the Spice display instead of VNC:

```
<graphics type='spice' port='5916' autoport='yes' listen='0.0.0.0'>
 <listen type='address' address='0.0.0.0'/>
</graphics>
```

6. Start the virtual machine and connect to its display with **`virt-viewer`**, for example:

```
> virt-viewer --connect qemu+ssh://USER@VM_HOST/system
```

7. From the list of VMs, select the one whose configuration you have modified and confirm with *Connect*.

8. After the graphical subsystem (Xorg) loads in the VM Guest, select *View* › *Displays* › *Display 2* to open a new window with the second monitor's output.

4.18 Crypto adapter pass-through to KVM guests on IBM Z

4.18.1 Introduction

IBM Z machines include cryptographic hardware with useful functions such as random number generation, digital signature generation, or encryption. KVM allows dedicating these crypto adapters to guests as pass-through devices. The means that the hypervisor cannot observe communications between the guest and the device.

4.18.2 What is covered

This section describes how to dedicate a crypto adapter and domains on an IBM Z host to a KVM guest. The procedure includes the following basic steps:

- Mask the crypto adapter and domains from the default driver on the host.
- Load the `vfio-ap` driver.
- Assign the crypto adapter and domains to the `vfio-ap` driver.
- Configure the guest to use the crypto adapter.

4.18.3 Requirements

- You need to have the QEMU / `libvirt` virtualization environment correctly installed and functional.
- The `vfio_ap` and `vfio_mdev` modules for the running kernel need to be available on the host operating system.

4.18.4 Dedicate a crypto adapter to a KVM host

1. Verify that the `vfio_ap` and `vfio_mdev` kernel modules are loaded on the host:

   ```
   > lsmod | grep vfio_
   ```

   If any of them is not listed, load it manually, for example:

   ```
   > sudo modprobe vfio_mdev
   ```

2. Create a new MDEV device on the host and verify that it was added:

   ```
   uuid=$(uuidgen)
   $ echo ${uuid} | sudo tee /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-
   passthrough/create
   dmesg | tail
   [...]
   [272197.818811] iommu: Adding device 24f952b3-03d1-4df2-9967-0d5f7d63d5f2 to group 0
   [272197.818815] vfio_mdev 24f952b3-03d1-4df2-9967-0d5f7d63d5f2: MDEV: group_id = 0
   ```

3. Identify the device on the host's logical partition that you intend to dedicate to a KVM guest:

   ```
   > ls -l /sys/bus/ap/devices/
   [...]
   lrwxrwxrwx 1 root root 0 Nov 23 03:29 00.0016 -> ../../../devices/ap/card00/00.0016/
   lrwxrwxrwx 1 root root 0 Nov 23 03:29 card00 -> ../../../devices/ap/card00/
   ```

   In this example, it is card `0` queue `16`. To match the Hardware Management Console (HMC) configuration, you need to convert from `16` hexadecimal to `22` decimal.

4. Mask the adapter from the `zcrypt` use:

   ```
   > lszcrypt
   CARD.DOMAIN TYPE MODE STATUS REQUEST_CNT
   -------------------------------------------------
   00 CEX5C CCA-Coproc online 5
   00.0016 CEX5C CCA-Coproc online 5
   ```

   Mask the adapter:

   ```
   > cat /sys/bus/ap/apmask
   0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
   echo -0x0 | sudo tee /sys/bus/ap/apmask
   0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
   ```

   Mask the domain:

   ```
   > cat /sys/bus/ap/aqmask
   0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
   echo -0x0 | sudo tee /sys/bus/ap/aqmask
   0xfffffdffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
   ```

5. Assign adapter 0 and domain 16 (22 decimal) to `vfio-ap`:

   ```
   > sudo echo +0x0 > /sys/devices/vfio_ap/matrix/${uuid}/assign_adapter
   ```

```
> echo +0x16 | sudo tee /sys/devices/vfio_ap/matrix/${uuid}/assign_domain
> echo +0x16 | sudo tee /sys/devices/vfio_ap/matrix/${uuid}/assign_control_domain
```

6. Verify the matrix that you have configured:

```
> cat /sys/devices/vfio_ap/matrix/${uuid}/matrix
00.0016
```

7. Either create a new VM and wait until it is initialized, or use an existing VM. In both cases, make sure the VM is shut down.

8. Change its configuration to use the MDEV device:

```
> sudo virsh edit VM_NAME
[...]
<hostdev mode='subsystem' type='mdev' model='vfio-ap'>
 <source>
  <address uuid='24f952b3-03d1-4df2-9967-0d5f7d63d5f2'/>
 </source>
</hostdev>
[...]
```

9. Restart the VM:

```
> sudo virsh reboot VM_NAME
```

10. Log in to the guest and verify that the adapter is present:

```
> lszcrypt
CARD.DOMAIN TYPE MODE STATUS REQUEST_CNT
-------------------------------------------------
00 CEX5C CCA-Coproc online 1
00.0016 CEX5C CCA-Coproc online 1
```

4.18.5 Further reading

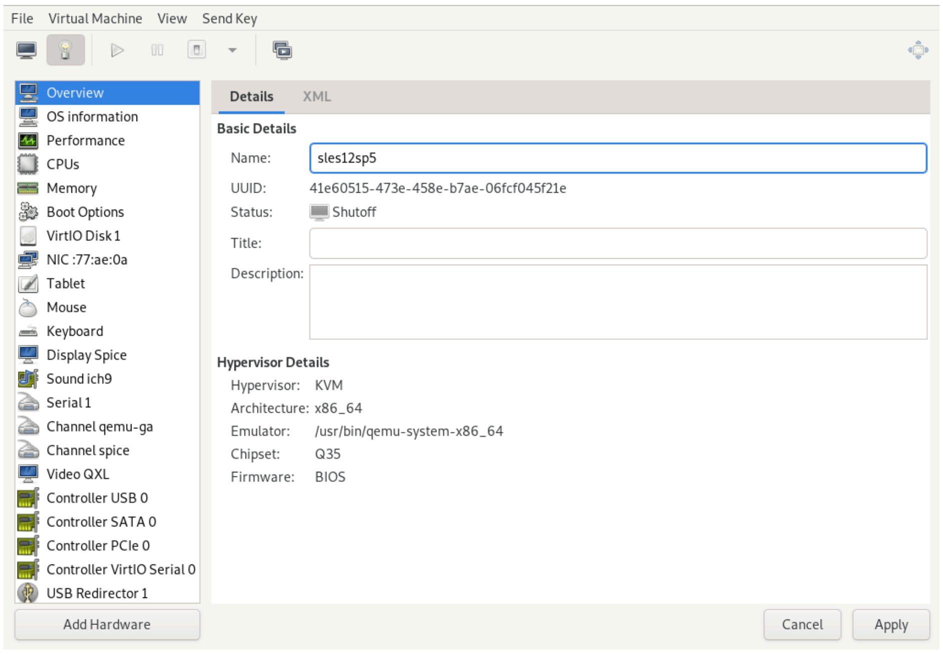
- The `vfio_ap` architecture is detailed in https://www.kernel.org/doc/Documentation/s390/vfio-ap.txt ↗.
- A general outline together with a detailed procedure is described in https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1787405 ↗.
- The architecture of VFIO Mediated devices (MDEVs) is detailed in https://www.kernel.org/doc/html/latest/driver-api/vfio-mediated-device.html ↗.

5 Configuring virtual machines with Virtual Machine Manager

Revision History

2024-06-27

Virtual Machine Manager's *Details* view offers in-depth information about the VM Guest's complete configuration and hardware equipment. Using this view, you can also change the guest configuration or add and modify virtual hardware. To access this view, open the guest's console in Virtual Machine Manager and either choose *View* › *Details* from the menu, or click *Show virtual hardware details* in the toolbar.



Basic Details

Name:	sles12sp5
UUID:	41e60515-473e-458e-b7ae-06fcf045f21e
Status:	Shutoff
Title:	
Description:	

Hypervisor Details

Hypervisor:	KVM
Architecture:	x86_64
Emulator:	/usr/bin/qemu-system-x86_64
Chipset:	Q35
Firmware:	BIOS

FIGURE 4: *DETAILS* VIEW OF A VM GUEST

The left panel of the window lists VM Guest overview and already installed hardware. After clicking an item on the list, you can access its detailed settings in the details view. You can change the hardware parameters to match your needs, then click *Apply* to confirm them. Certain changes take effect immediately, while others need a reboot of the machine—and `virt-manager` warns you about that fact.

To remove installed hardware from a VM Guest, select the appropriate list entry in the left panel and then click *Remove* in the bottom right of the window.

To add new hardware, click *Add Hardware* below the left panel, then select the type of the hardware you want to add in the *Add New Virtual Hardware* window. Modify its parameters and confirm with *Finish*.

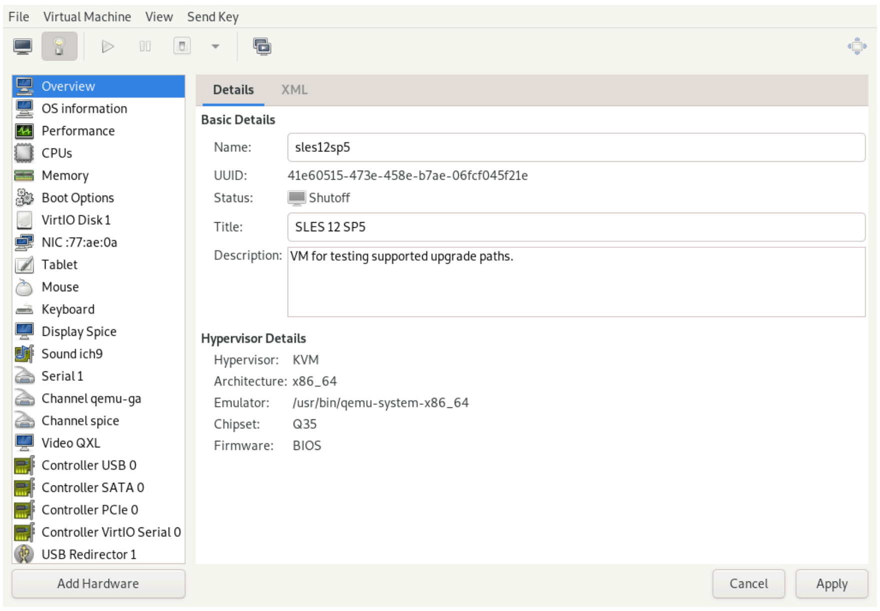
The following sections describe configuration options for the specific hardware type *being added*. They do not focus on modifying an existing piece of hardware, as the options are identical.

5.1 Machine setup

This section describes the setup of the virtualized processor and memory hardware. These components are vital to a VM Guest, therefore you cannot remove them. It also shows how to view the overview and performance information, and how to change boot parameters.

5.1.1 Overview

Overview shows basic details about VM Guest and the hypervisor.



Basic Details

Name: sles12sp5
UUID: 41e60515-473e-458e-b7ae-06fcf045f21e
Status: Shutoff
Title: SLES 12 SP5
Description: VM for testing supported upgrade paths.

Hypervisor Details

Hypervisor: KVM
Architecture: x86_64
Emulator: /usr/bin/qemu-system-x86_64
Chipset: Q35
Firmware: BIOS

FIGURE 5: OVERVIEW DETAILS

Name, *Title*, and *Description* are editable and help you identify VM Guest in the *Virtual Machine Manager* list of machines.

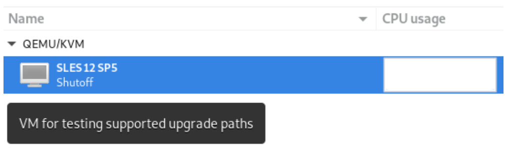


FIGURE 6: VM GUEST TITLE AND DESCRIPTION

UUID shows the universally unique identifier of the virtual machine, while *Status* shows its current status—*Running*, *Paused*, or *Shutoff*.

The *Hypervisor Details* section shows the hypervisor type, CPU architecture, used emulator, and chipset type. None of the hypervisor parameters can be changed.

5.1.2 Performance

Performance shows regularly updated charts of CPU and memory usage, and disk and network I/O.

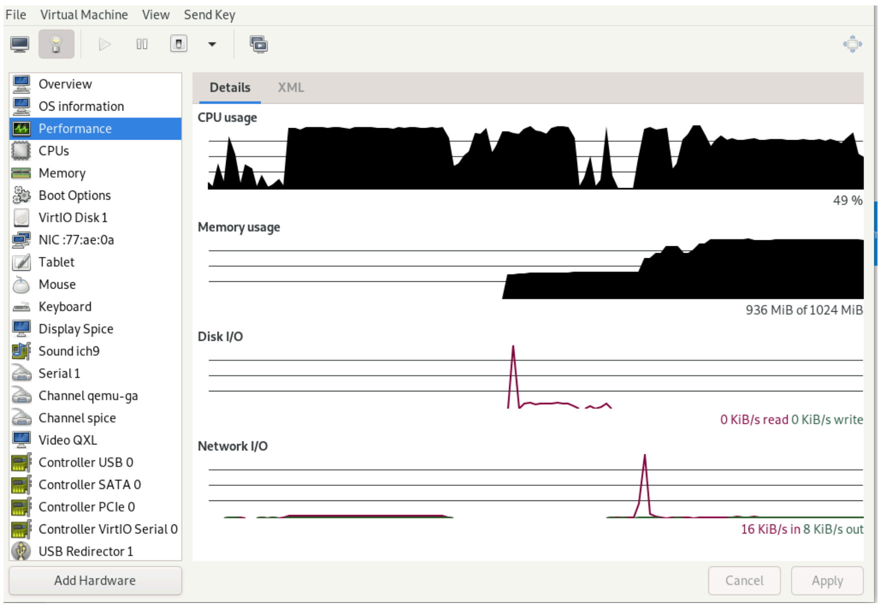
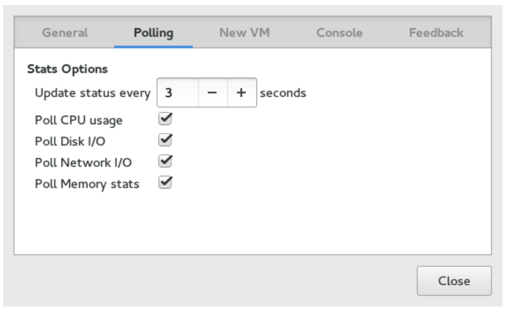


FIGURE 7: PERFORMANCE

Tip: Enabling disabled charts

Not all the charts in the *Graph* view are enabled by default. To enable these charts, go to *File* › *View Manager*, then select *Edit* › *Preferences* › *Polling*, and check the charts that you want to see regularly updated.



Stats Options

Update status every 3 seconds

Poll CPU usage ✓

Poll Disk I/O ✓

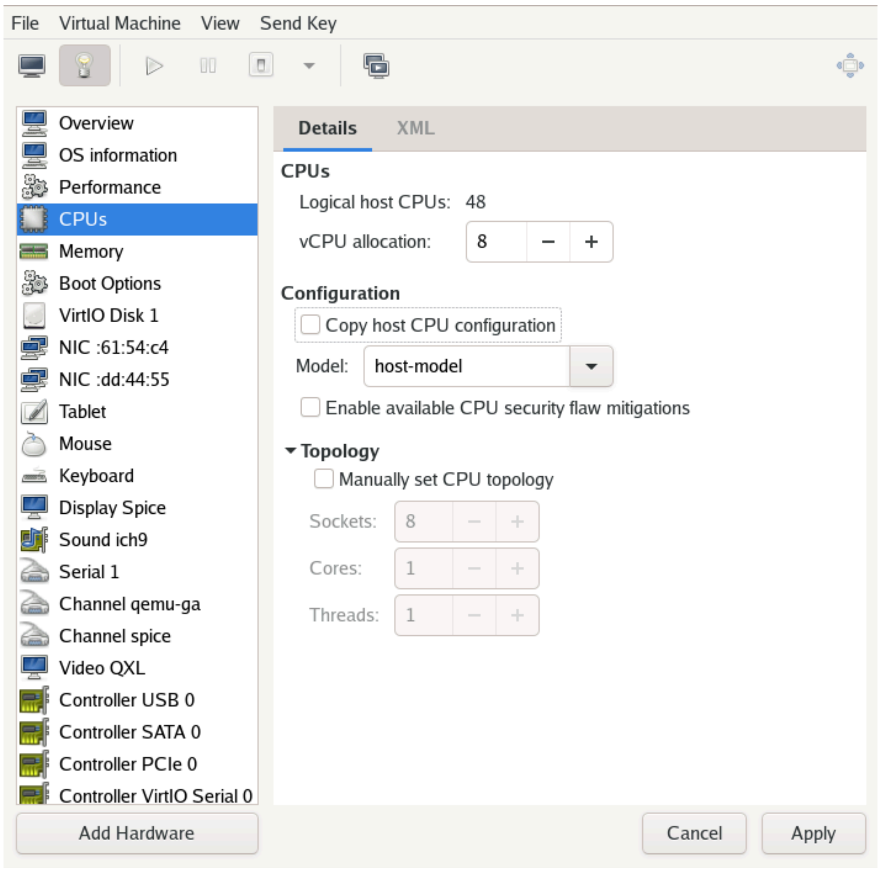
Poll Network I/O ✓

Poll Memory stats ✓

FIGURE 8: STATISTICS CHARTS

5.1.3 Processor

CPU includes detailed information about VM Guest processor configuration.



CPUs

Logical host CPUs: 48

vCPU allocation: 8

Configuration

☐ Copy host CPU configuration

Model: host-model

☐ Enable available CPU security flaw mitigations

Topology

☐ Manually set CPU topology

Sockets: 8

Cores: 1

Threads: 1

FIGURE 9: PROCESSOR VIEW

In the *CPUs* section, you can configure the number of virtual CPUs allocated to the VM Guest. *Logical host CPUs* shows the number of online and usable CPUs on the VM Host Server.

The *Configuration* section lets you configure the CPU model and topology.

When activated, the *Copy host CPU configuration* option uses the host CPU model for VM Guest. You can see the details of the host CPU model in the output of the **`virsh capabilities`** command. When deactivated, the CPU model needs to be specified from the models available in the drop-down box.

The host CPU model provides a good trade-off between CPU features and the ability to migrate the VM Guest. `libvirt` does not model every aspect of each CPU, so the VM Guest CPU does not match the VM Host Server CPU exactly. But the ABI provided to the VM Guest is reproducible and during migration the complete CPU model definition is transferred to the destination VM Host Server, ensuring the migrated VM Guest can see the exact same CPU model on the destination.

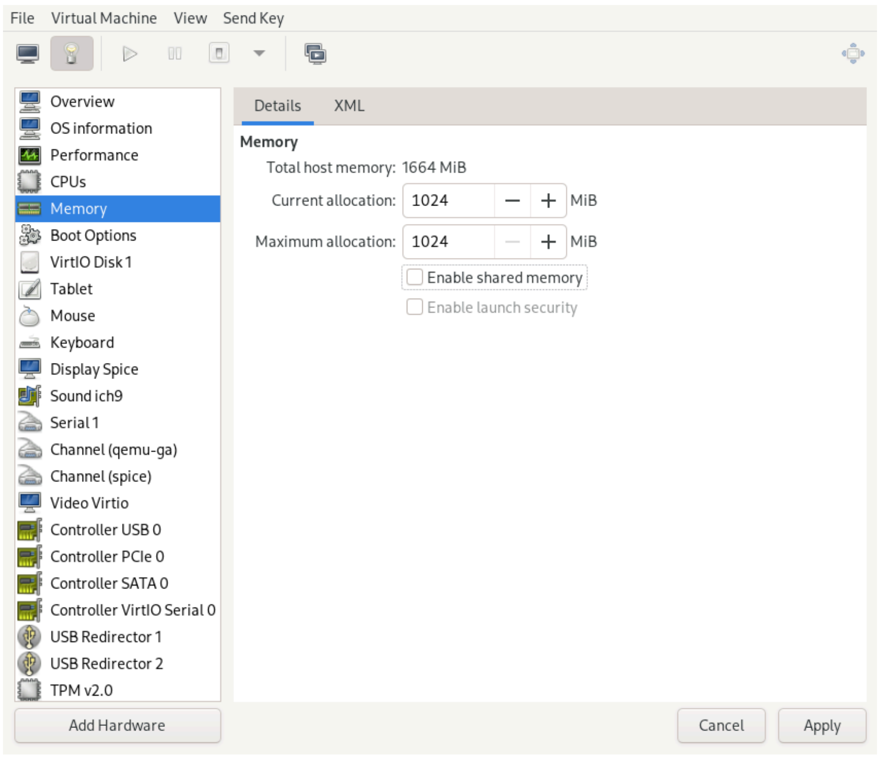
The `host-passthrough` model provides the VM Guest with a CPU that is exactly the same as the VM Host Server CPU. This can be useful when the VM Guest workload requires CPU features not available in `libvirt`'s simplified `host-model` CPU. The `host-passthrough` model comes with the disadvantage of reduced migration capability. A VM Guest with `host-passthrough` model CPU can only be migrated to a VM Host Server with identical hardware.

For more information on `libvirt`'s CPU model and topology options, see the *CPU model and topology* documentation at https://libvirt.org/formatdomain.html#cpu-model-and-topology.

After you activate *Manually set CPU topology*, you can specify a custom number of sockets, cores and threads for the CPU.

5.1.4 Memory

Memory contains information about the memory that is available to VM Guest.



Memory

Total host memory: 1664 MiB

Current allocation: 1024 MiB

Maximum allocation: 1024 MiB

☐ Enable shared memory

☐ Enable launch security

FIGURE 10: MEMORY VIEW

Total host memory

Total amount of memory installed on VM Host Server.

Current allocation

The amount of memory currently available to VM Guest. You can hotplug more memory by increasing this value up to the value of *Maximum allocation*.

Enable shared memory

Specify if the virtual machine can use shared memory via the `memfd` backed. It is a requirement for using the *virtiofs* file system. Find more details in https://libvirt.org/kbase/virtiofs.html ↗.

Maximum allocation

The maximum value to which you can hotplug the currently available memory. Any change to this value takes effect after the next VM Guest reboot.

Enable launch security

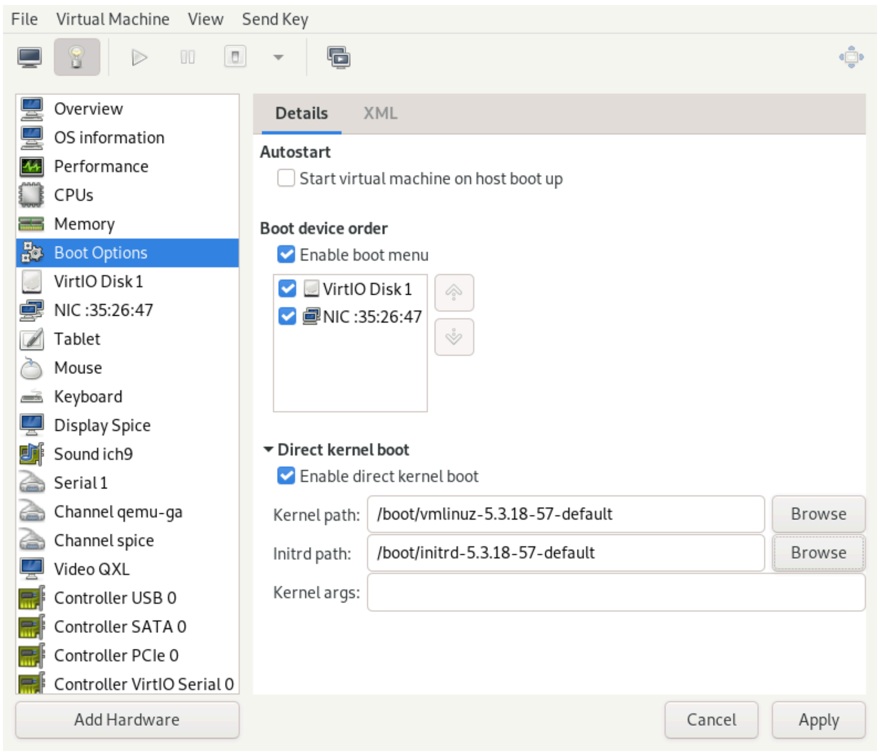
If the VM Host Server supports AMD-SEV technology, activating this option enables a secured guest with encrypted memory. This option requires a virtual machine with chipset type Q35. For more details, refer to https://documentation.suse.com/sles/html/SLES-amd-sev/article-amd-sev.html ↗.

Important: Large memory VM Guests

VM Guests with memory requirements of 4 TB or more must either use the `host-passthrough` CPU mode, or explicitly specify the virtual CPU address size when using `host-model` or `custom` CPU modes. The default virtual CPU address size for these modes may not be sufficient for memory configurations of 4 TB or more. The address size can only be specified by editing the VM Guests XML configuration. See *Section 4.6, "Configuring memory allocation"* for more information on specifying virtual CPU address size.

5.1.5 Boot options

Boot Options introduces options affecting the VM Guest boot process.



Autostart

[ ] Start virtual machine on host boot up

Boot device order

[✓] Enable boot menu

[✓] VirtIO Disk 1

[✓] NIC :35:26:47

Direct kernel boot

[✓] Enable direct kernel boot

Kernel path: /boot/vmlinuz-5.3.18-57-default

Initrd path: /boot/initrd-5.3.18-57-default

Kernel args:

FIGURE 11: BOOT OPTIONS

In the *Autostart* section, you can specify whether the virtual machine should automatically start during the VM Host Server boot phase.

In the *Boot device order*, activate the devices used for booting VM Guest. You can change their order with the up and down arrow buttons on the right side of the list. To choose from a list of bootable devices on VM Guest start, activate *Enable boot menu*.

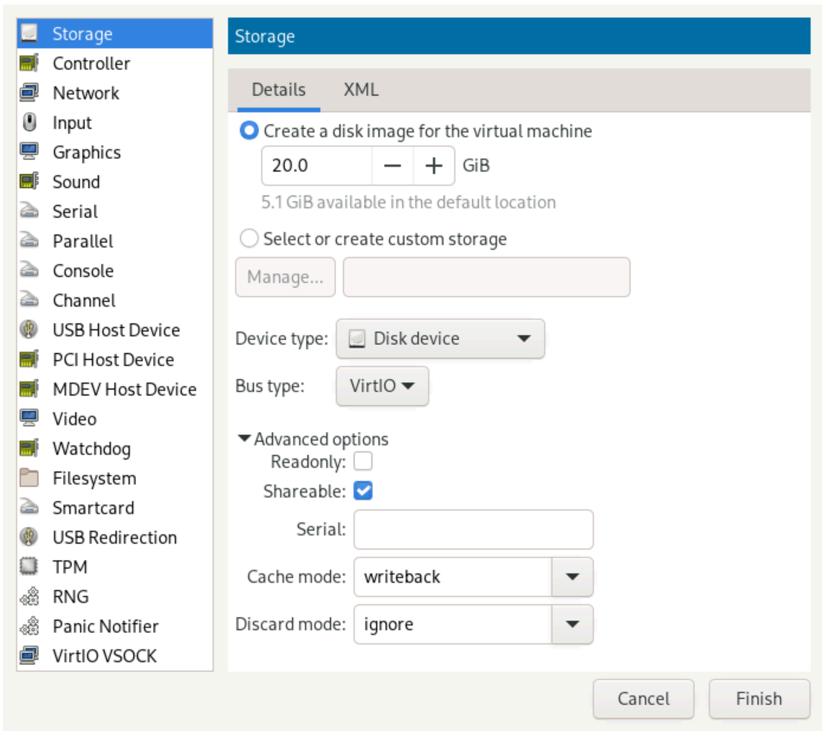
To boot a different kernel than the one on the boot device, activate *Enable direct kernel boot* and specify the paths to the alternative kernel and initrd placed on the VM Host Server file system. You can also specify kernel arguments that are passed to the loaded kernel.

5.2 Storage

This section gives you a detailed description of configuration options for storage devices. It includes both hard disks and removable media, such as USB or CD-ROM drives.

PROCEDURE 5: ADDING A NEW STORAGE DEVICE

1. Below the left panel, click *Add Hardware* to open the *Add New Virtual Hardware* window. There, select *Storage*.



(●) Create a disk image for the virtual machine
20.0 GiB
5.1 GiB available in the default location
( ) Select or create custom storage
Device type: Disk device
Bus type: VirtIO
Advanced options
Readonly: ☐
Shareable: ☑
Serial:
Cache mode: writeback
Discard mode: ignore

FIGURE 12: ADD A NEW STORAGE

2. To create a `qcow2` disk image in the default location, activate *Create a disk image for the virtual machine* and specify its size in gigabytes.
   To gain more control over the disk image creation, activate *Select or create custom storage* and click *Manage* to manage storage pools and images. The window *Choose Storage Volume* opens, which has almost identical functionality as the *Storage* tab described in *Section 2.2.2, "Managing storage with Virtual Machine Manager"*.

   ### Tip: Supported storage formats

   SUSE only supports the following storage formats: `raw` and `qcow2`.

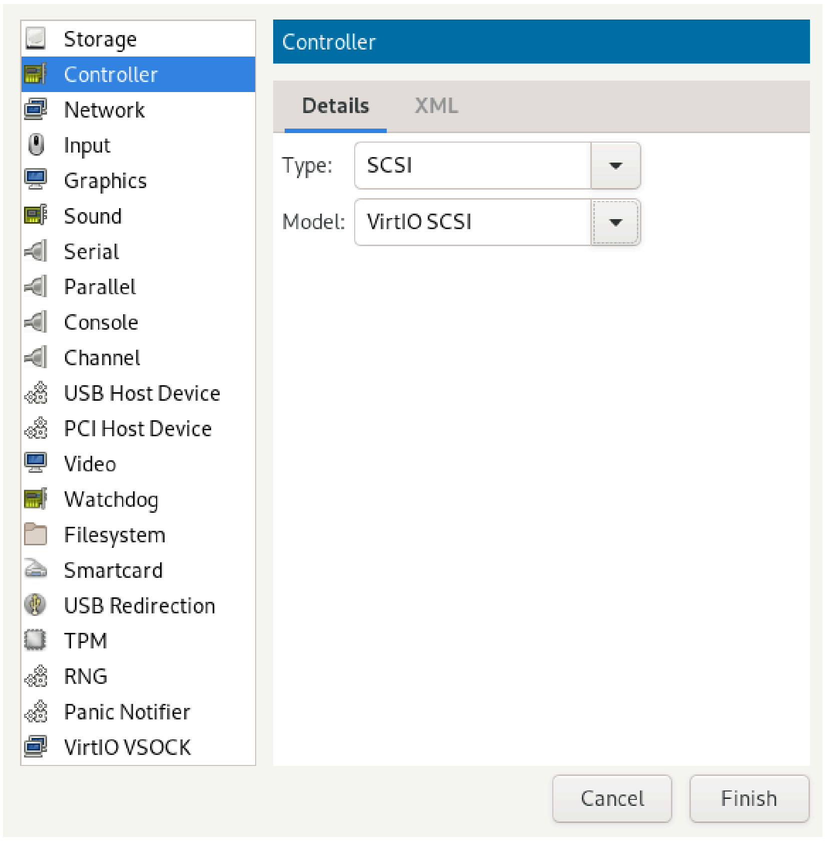
3. After you create and specify the disk image file, specify the *Device type*. It can be one of the following options:
   - *Disk device*
   - *CDROM device*: does not allow using *Create a disk image for the virtual machine*.
   - *Floppy device*: does not allow using *Create a disk image for the virtual machine*.
   - *LUN Passthrough*: required to use an existing SCSI storage directly without adding it into a storage pool.
4. Select the *Bus type* for your device. The list of available options depends on the device type you selected in the previous step. The types based on *VirtIO* use paravirtualized drivers.
5. In the *Advanced options* section, select the preferred *Cache mode*.
6. Confirm your settings with *Finish*. A new storage device appears in the left panel.

5.3 Controllers

This section focuses on adding and configuring new controllers.

PROCEDURE 6: ADDING A NEW CONTROLLER

1. Below the left panel, click *Add Hardware* to open the *Add New Virtual Hardware* window. There, select *Controller*.



Type: SCSI
Model: VirtIO SCSI

FIGURE 13: ADD A NEW CONTROLLER

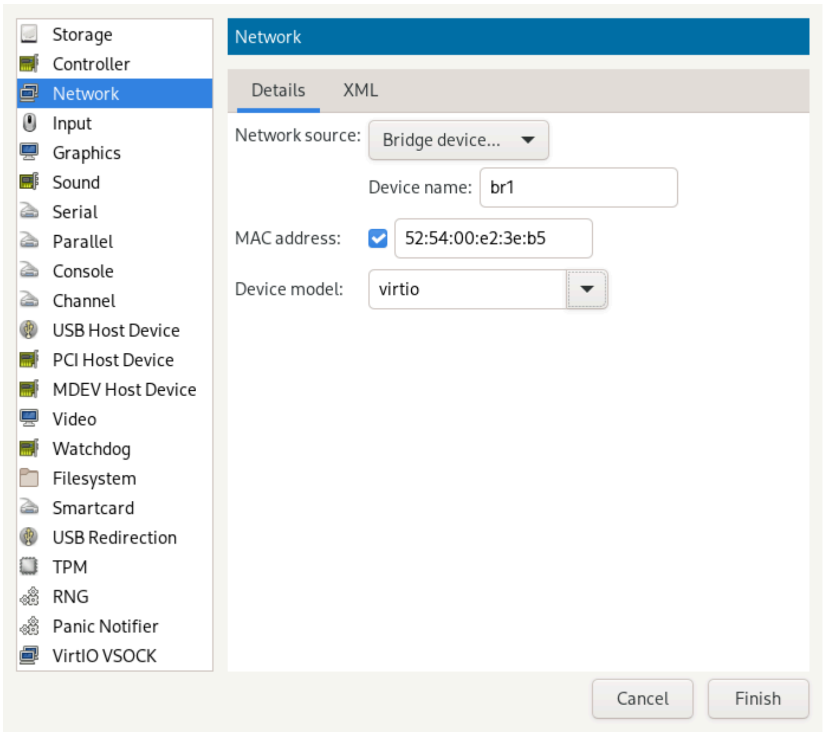
2. Select the type of the controller. You can choose from *IDE*, *Floppy*, *SCSI*, *SATA*, *VirtIO Serial* (paravirtualized), *USB*, or *CCID* (smart card devices).
3. Optionally, for a USB or SCSI controller, select a controller model.
4. Confirm your settings with *Finish*. A new controller appears in the left panel.

5.4 Networking

This section describes how to add and configure new network devices.

PROCEDURE 7: ADDING A NEW NETWORK DEVICE

1. Below the left panel, click *Add Hardware* to open the *Add New Virtual Hardware* window. There, select *Network*.



Network

Network source: Bridge device...

Device name: br1

MAC address: ✓ 52:54:00:e2:3e:b5

Device model: virtio

FIGURE 14: ADD A NEW NETWORK INTERFACE

2. From the *Network source* list, select the source for the network connection. The list includes VM Host Server's available physical network interfaces, network bridges, or network bonds. You can also assign the VM Guest to an already defined virtual network. See *Section 2.1, "Configuring networks"* for more information on setting up virtual networks with Virtual Machine Manager.

3. Specify a *MAC address* for the network device. While Virtual Machine Manager pre-fills a random value for your convenience, it is recommended to supply a MAC address appropriate for your network environment to avoid network conflicts.

4. Select a device model from the list. You can either leave the *Hypervisor default*, or specify one of *e1000*, *rtl8139*, or *virtio* models. *virtio* uses paravirtualized drivers.

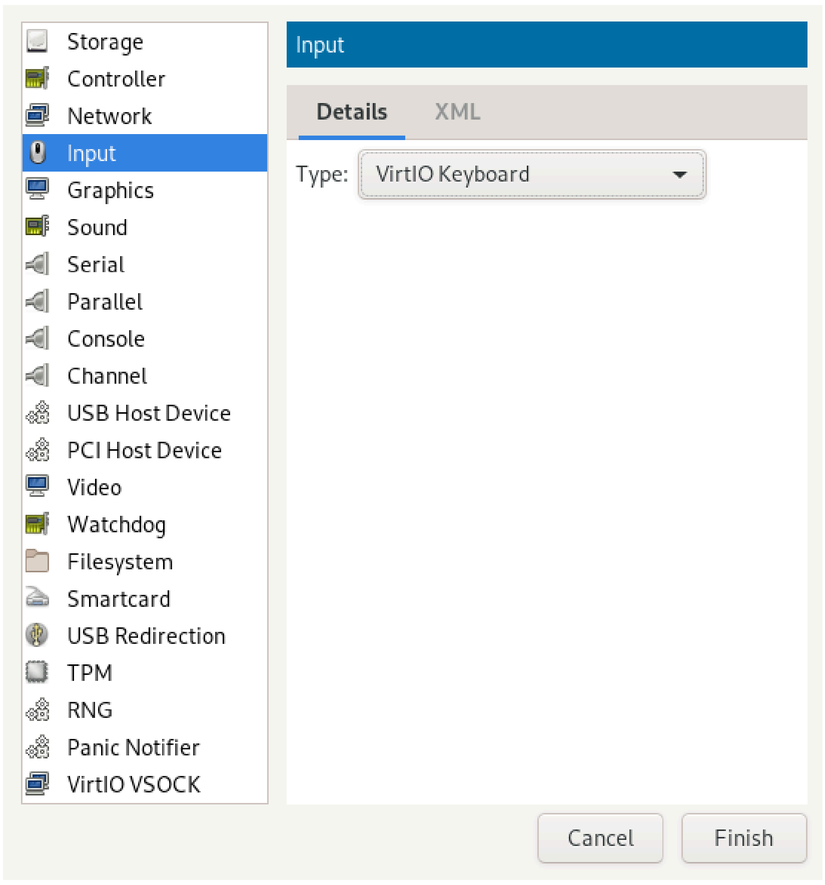
5. Confirm your settings with *Finish*. A new network device appears in the left panel.

5.5 Input devices

This section focuses on adding and configuring new input devices, such as a mouse, a keyboard or a tablet.

PROCEDURE 8: ADDING A NEW INPUT DEVICE

1. Below the left panel, click *Add Hardware* to open the *Add New Virtual Hardware* window. There, select *Input*.



Type: VirtIO Keyboard

FIGURE 15: ADD A NEW INPUT DEVICE

2. Select a device type from the list.

3. Confirm your settings with *Finish*. A new input device appears in the left panel.

Tip: Enabling seamless and synchronized mouse pointer movement

When you click within a VM Guest's console with the mouse, the pointer is captured by the console window and cannot be used outside the console unless it is explicitly released (by pressing `Alt`–`Ctrl`). To prevent the console from grabbing the key and to enable seamless pointer movement between host and guest instead, follow the instructions in *Procedure 8, "Adding a new input device"* to add an *EvTouch USB Graphics Tablet* to the VM Guest.

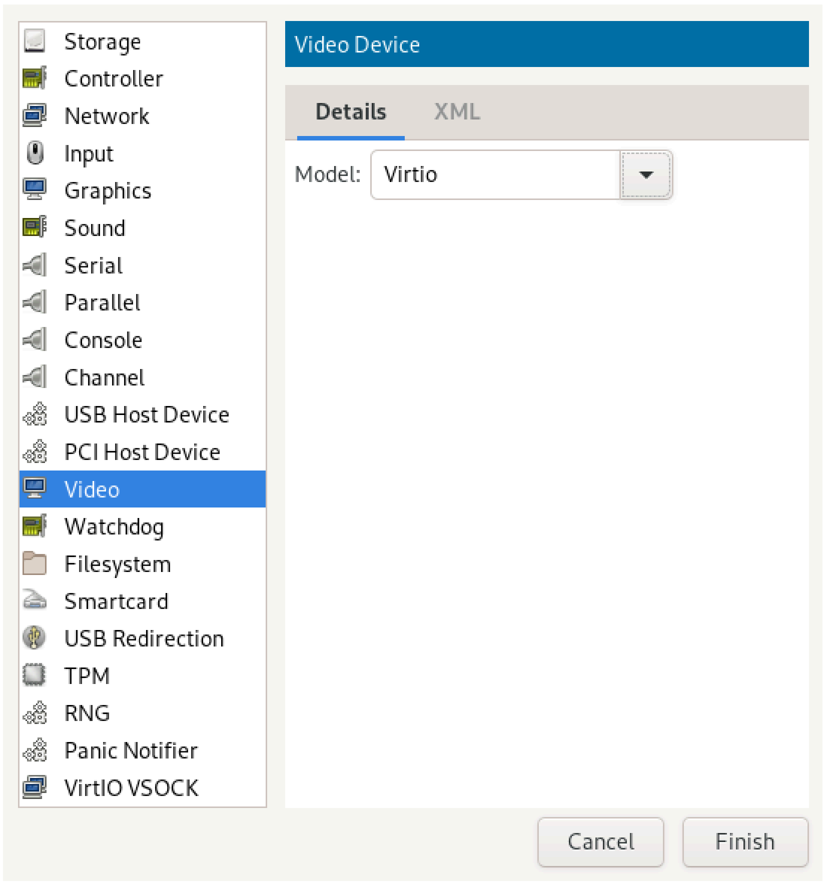
Adding a tablet has the additional advantage of synchronizing the mouse pointer movement between VM Host Server and VM Guest when using a graphical environment on the guest. With no tablet configured on the guest, you may often see two pointers with one dragging behind the other.

5.6 Video

This section describes how to add and configure new video devices.

PROCEDURE 9: ADDING A VIDEO DEVICE

1. Below the left panel, click *Add Hardware* to open the *Add New Virtual Hardware* window. There, select *Video*.

2. 

Model: Virtio

FIGURE 16: ADD A NEW VIDEO DEVICE

3. Select a model from the drop-down box.

> **Note: Secondary video devices**
>
> Only *QXL* and *Virtio* can be added as secondary video devices.

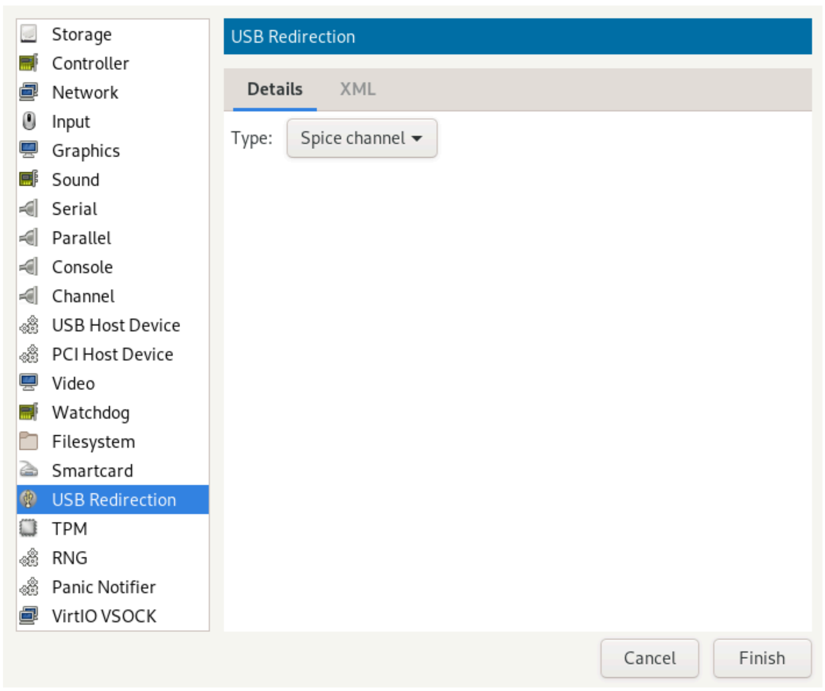
4. Confirm your settings with *Finish*. A new video device appears in the left panel.

5.7 USB redirectors

USB devices that are connected to the client machine can be redirected to the VM Guest by using *USB Redirectors*.

PROCEDURE 10: ADDING A USB REDIRECTOR

1. Below the left panel, click *Add Hardware* to open the *Add New Virtual Hardware* window. There, select *USB Redirection*.



Type: Spice channel

FIGURE 17: ADD A NEW USB REDIRECTOR

2. Select a device type from the list. Depending on your configuration, you can either select a *Spice channel* or a *TCP* redirector.

3. Confirm your settings with *Finish*. A new USB redirector appears in the left panel.

5.8 Miscellaneous

Smartcard

Smartcard functionality can be added via the *Smartcard* element. A physical USB smartcard reader can then be passed through to the VM Guest.

Watchdog

Virtual watchdog devices are also supported. They can be created via the *Watchdog* element. The model and the action of the device can be specified.

Tip: Requirements for virtual watchdog devices

QA virtual watchdog devices require a specific driver and daemon to be installed in the VM Guest. Otherwise, the virtual watchdog device does not work.

TPM

You can use the Host TPM device in the VM Guest by adding TPM functionality via the *TPM* element.

Tip: Virtual TPMs

The Host TPM can only be used in one VM Guest at a time.

5.9 Adding a CD/DVD-ROM device with Virtual Machine Manager

KVM supports CD or DVD-ROMs in VM Guest either by directly accessing a physical drive on the VM Host Server or by accessing ISO images. To create an ISO image from an existing CD or DVD, use **dd**:

```
> sudo dd if=/dev/CD_DVD_DEVICE of=my_distro.iso bs=2048
```

To add a CD/DVD-ROM device to your VM Guest, proceed as follows:

1. Double-click a VM Guest entry in the Virtual Machine Manager to open its console and switch to the *Details* view with *View* › *Details*.
2. Click *Add Hardware* and choose *Storage* in the pop-up window.

3. Change the *Device Type* to *IDE CDROM*.

4. Select *Select or create custom storage*.

   a. To assign the device to a physical medium, enter the path to the VM Host Server's CD/DVD-ROM device, for example, `/dev/cdrom`) next to *Manage*. Alternatively, use *Manage* to open a file browser and then click *Browse Local* to select the device. Assigning the device to a physical medium is only possible when the Virtual Machine Manager was started on the VM Host Server.

   b. To assign the device to an existing image, click *Manage* to choose an image from a storage pool. If the Virtual Machine Manager was started on the VM Host Server, alternatively choose an image from another location on the file system by clicking *Browse Local*. Select an image and close the file browser with *Choose Volume*.

5. Save the new virtualized device with *Finish*.

6. Reboot the VM Guest to make the new device available. For more information, see *Section 5.11, "Ejecting and changing floppy or CD/DVD-ROM media with Virtual Machine Manager"*.

5.10 Adding a floppy device with Virtual Machine Manager

Currently, KVM only supports the use of floppy disk images—using a physical floppy drive is not supported. Create a floppy disk image from an existing floppy using **dd**:

```
> sudo dd if=/dev/fd0 of=/var/lib/libvirt/images/floppy.img
```

To create an empty floppy disk image, use one of the following commands:

Raw image

```
> sudo dd if=/dev/zero of=/var/lib/libvirt/images/floppy.img bs=512 count=2880
```

FAT formatted image

```
> sudo mkfs.msdos -C /var/lib/libvirt/images/floppy.img 1440
```

To add a floppy device to your VM Guest, proceed as follows:

1. Double-click a VM Guest entry in the Virtual Machine Manager to open its console and switch to the *Details* view with *View* › *Details*.

2. Click *Add Hardware* and choose *Storage* in the pop-up window.
3. Change the *Device Type* to *Floppy Disk*.
4. Choose *Select or create custom storage* and click *Manage* to choose an existing image from a storage pool. If Virtual Machine Manager was started on the VM Host Server, alternatively choose an image from another location on the file system by clicking *Browse Local*. Select an image and close the file browser with *Choose Volume*.
5. Save the new virtualized device with *Finish*.
6. Reboot the VM Guest to make the new device available. For more information, see *Section 5.11, "Ejecting and changing floppy or CD/DVD-ROM media with Virtual Machine Manager"*.

5.11 Ejecting and changing floppy or CD/DVD-ROM media with Virtual Machine Manager

Whether you are using the VM Host Server's physical CD/DVD-ROM device or an ISO/floppy image: before you can change the media or image of an existing device in the VM Guest, you first need to `disconnect` the media from the guest.

1. Double-click a VM Guest entry in the Virtual Machine Manager to open its console and switch to the *Details* view with *View* › *Details*.
2. Choose the Floppy or CD/DVD-ROM device and "eject" the medium by clicking *Disconnect*.
3. To "insert" a new medium, click *Connect*.
    a. If using the VM Host Server's physical CD/DVD-ROM device, first change the media in the device (this may require unmounting it on the VM Host Server before it can be ejected). Then choose *CD-ROM or DVD* and select the device from the drop-down box.
    b. If you are using an ISO image, choose *ISO image Location* and select an image by clicking *Manage*. When connecting from a remote host, you may only choose images from existing storage pools.
4. Click *OK* to finish. The new media can now be accessed in the VM Guest.

5.12 Assigning a host PCI device to a VM Guest

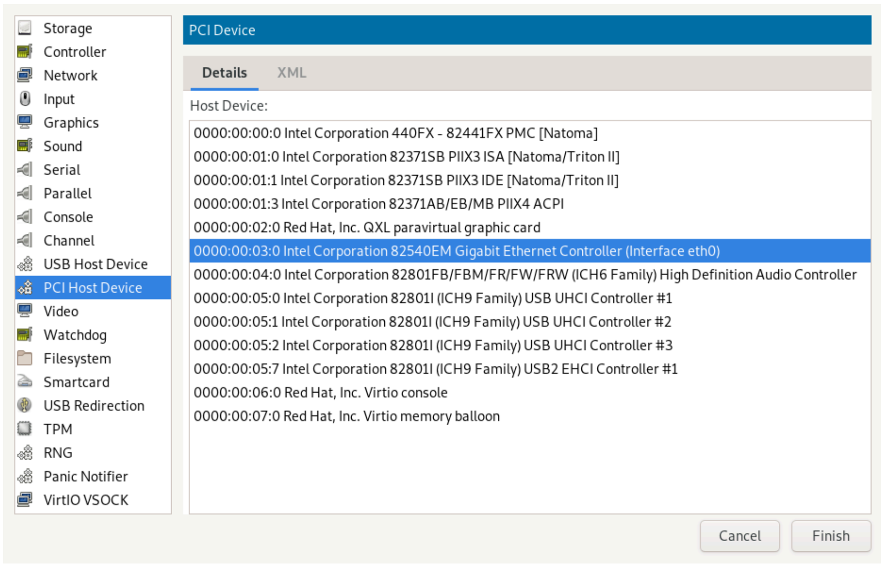
You can directly assign host-PCI devices to guests (PCI pass-through). When the PCI device is assigned to one VM Guest, it cannot be used on the host or by another VM Guest unless it is reassigned. A prerequisite for this feature is a VM Host Server configuration as described in *Important: Requirements for VFIO and SR-IOV*.

5.12.1 Adding a PCI device with Virtual Machine Manager

The following procedure describes how to assign a PCI device from the host machine to a VM Guest using Virtual Machine Manager:

1. Double-click a VM Guest entry in the Virtual Machine Manager to open its console and switch to the *Details* view with *View* › *Details*.

2. Click *Add Hardware* and choose the *PCI Host Device* category in the left panel. A list of available PCI devices appears in the right part of the window.



Host Device:

```
0000:00:00:0 Intel Corporation 440FX - 82441FX PMC [Natoma]
0000:00:01:0 Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
0000:00:01:1 Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
0000:00:01:3 Intel Corporation 82371AB/EB/MB PIIX4 ACPI
0000:00:02:0 Red Hat, Inc. QXL paravirtual graphic card
0000:00:03:0 Intel Corporation 82540EM Gigabit Ethernet Controller (Interface eth0)
0000:00:04:0 Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) High Definition Audio Controller
0000:00:05:0 Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #1
0000:00:05:1 Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #2
0000:00:05:2 Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #3
0000:00:05:7 Intel Corporation 82801I (ICH9 Family) USB2 EHCI Controller #1
0000:00:06:0 Red Hat, Inc. Virtio console
0000:00:07:0 Red Hat, Inc. Virtio memory balloon
```

FIGURE 18: ADDING A PCI DEVICE

3. From the list of available PCI devices, choose the one you want to pass to the guest. Confirm with *Finish*.

Important: SLES 11 SP4 KVM guests

On a newer QEMU machine type (pc-i440fx-2.0 or higher) with SLES 11 SP4 KVM guests, the `acpiphp` module is not loaded by default in the guest. This module must be loaded to enable hotplugging of disk and network devices. To load the module manually, use the command **`modprobe acpiphp`**. It is also possible to autoload the module by adding `install acpiphp /bin/true` to the `/etc/modprobe.conf.local` file.

Important: KVM guests using QEMU Q35 machine type

KVM guests using the QEMU Q35 machine type have a PCI topology that includes a `pcie-root` controller and seven `pcie-root-port` controllers. The `pcie-root` controller does not support hotplugging. Each `pcie-root-port` controller supports hotplugging a single PCIe device. PCI controllers cannot be hotplugged, so plan accordingly and add more `pcie-root-ports` for more than seven hotplugged PCIe devices. A `pcie-to-pci-bridge` controller can be added to support hotplugging legacy PCI devices. See https://libvirt.org/pci-hotplug.html ↗ for more information about PCI topology between QEMU machine types.

5.13 Assigning a host USB device to a VM Guest

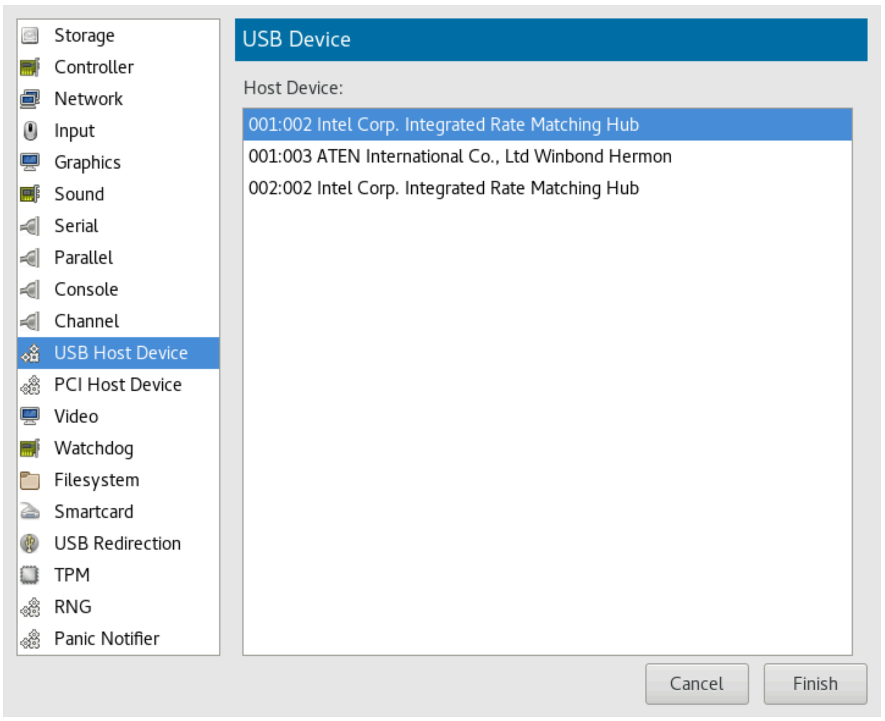
Analogous to assigning host PCI devices (see *Section 5.12, "Assigning a host PCI device to a VM Guest"*), you can directly assign host USB devices to guests. When the USB device is assigned to one VM Guest, it cannot be used on the host or by another VM Guest unless it is reassigned.

5.13.1 Adding a USB device with Virtual Machine Manager

To assign a host USB device to VM Guest using Virtual Machine Manager, follow these steps:

1. Double-click a VM Guest entry in the Virtual Machine Manager to open its console and switch to the *Details* view with *View* › *Details*.

2. Click *Add Hardware* and choose the *USB Host Device* category in the left panel. A list of available USB devices appears in the right part of the window.



001:002 Intel Corp. Integrated Rate Matching Hub
001:003 ATEN International Co., Ltd Winbond Hermon
002:002 Intel Corp. Integrated Rate Matching Hub

FIGURE 19: ADDING A USB DEVICE

3. From the list of available USB devices, choose the one you want to pass to the guest. Confirm with *Finish*. The new USB device appears in the left pane of the *Details* view.

Tip: USB device removal

To remove the host USB device assignment, click it in the left pane of the *Details* view and confirm with *Remove*.

6 Advanced storage topics

Revision History

2024-06-27

This chapter introduces advanced topics about manipulating storage from the perspective of the VM Host Server.

6.1 Locking disk files and block devices with `virtlockd`

Locking block devices and disk files prevents concurrent writes to these resources from different VM Guests. It provides protection against starting the same VM Guest twice, or adding the same disk to two different virtual machines. This reduces the risk of a virtual machine's disk image becoming corrupted because of a wrong configuration.

The locking is controlled by a daemon called `virtlockd`. Since it operates independently from the `libvirtd` daemon, locks endure a crash or a restart of `libvirtd`. Locks even persist during an update of the `virtlockd` itself, since it can re-execute itself. This ensures that VM Guests do *not* need to be restarted upon a `virtlockd` update. `virtlockd` is supported for KVM, QEMU.

6.1.1 Enable locking

Locking virtual disks is not enabled by default on SUSE Linux Enterprise Server. To enable and automatically start it upon rebooting, perform the following steps:

1. Edit `/etc/libvirt/qemu.conf` and set

```
lock_manager = "lockd"
```

2. Start the `virtlockd` daemon with the following command:

```
> sudo systemctl start virtlockd
```

3. Restart the `libvirtd` daemon with:

```
> sudo systemctl restart libvirtd
```

4. Make sure `virtlockd` is automatically started when booting the system:

```
> sudo systemctl enable virtlockd
```

6.1.2 Configure locking

By default `virtlockd` is configured to automatically lock all disks configured for your VM Guests. The default setting uses a “direct” lockspace, where the locks are acquired against the actual file paths associated with the VM Guest <disk> devices. For example, `flock(2)` is called directly on `/var/lib/libvirt/images/my-server/disk0.raw` when the VM Guest contains the following <disk> device:

```
<disk type='file' device='disk'>
 <driver name='qemu' type='raw'/>
 <source file='/var/lib/libvirt/images/my-server/disk0.raw'/>
 <target dev='vda' bus='virtio'/>
</disk>
```

The `virtlockd` configuration can be changed by editing the file `/etc/libvirt/qemu-lockd.conf`. It also contains detailed comments with further information. Make sure to activate configuration changes by reloading `virtlockd`:

```
> sudo systemctl reload virtlockd
```

6.1.2.1 Enabling an indirect lockspace

The default configuration of `virtlockd` uses a “direct” lockspace. This means that the locks are acquired against the actual file paths associated with the <disk> devices.

If the disk file paths are not accessible to all hosts, `virtlockd` can be configured to allow an “indirect” lockspace. This means that a hash of the disk image path is used to create a file in the indirect lockspace directory. The locks are then held on these hash files instead of the actual disk file paths. Indirect lockspace is also useful if the file system containing the disk files does not support `fcntl()` locks. An indirect lockspace is specified with the `file_lockspace_dir` setting:

```
file_lockspace_dir = "/MY_LOCKSPACE_DIRECTORY"
```

6.1.2.2 Enable locking on LVM or iSCSI volumes

When wanting to lock virtual disks placed on LVM or iSCSI volumes shared by several hosts, locking needs to be done by UUID rather than by path (which is used by default). Furthermore, the lockspace directory needs to be placed on a shared file system accessible by all hosts sharing the volume. Set the following options for LVM and/or iSCSI:

```
lvm_lockspace_dir = "/MY_LOCKSPACE_DIRECTORY"
```

```
iscsi_lockspace_dir = "/MY_LOCKSPACE_DIRECTORY"
```

6.2 Online resizing of guest block devices

Sometimes you need to change—extend or shrink—the size of the block device used by your guest system. For example, when the disk space originally allocated is no longer enough, it is time to increase its size. If the guest disk resides on a *logical volume*, you can resize it while the guest system is running. This is a big advantage over an offline disk resizing as the service provided by the guest is not interrupted by the resizing process. To resize a VM Guest disk, follow these steps:

PROCEDURE 11: ONLINE RESIZING OF GUEST DISK

1. Inside the guest system, check the current size of the disk (for example `/dev/vda`).

```
# fdisk -l /dev/vda
Disk /dev/sda: 160.0 GB, 160041885696 bytes, 312581808 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

2. On the host, resize the logical volume holding the `/dev/vda` disk of the guest to the required size, for example, 200 GB.

```
# lvresize -L 200G /dev/mapper/vg00-home
Extending logical volume home to 200 GiB
Logical volume home successfully resized
```

3. On the host, resize the block device related to the disk `/dev/mapper/vg00-home` of the guest. You can find the *`DOMAIN_ID`* with **`virsh list`**.

```
# virsh blockresize  --path /dev/vg00/home --size 200G DOMAIN_ID
Block device '/dev/vg00/home' is resized
```

4. Check that the new disk size is accepted by the guest.

```
# fdisk -l /dev/vda
Disk /dev/sda: 200.0 GB, 200052357120 bytes, 390727260 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

6.3 Sharing directories between host and guests (file system pass-through)

libvirt allows to share directories between host and guests using QEMU's file system pass-through (also called VirtFS) feature. Such a directory can be also be accessed by several VM Guests at once and therefore be used to exchange files between VM Guests.

Note: Windows guests and file system pass-through

Sharing directories between VM Host Server and Windows guests via File System Pass-Through does not work, because Windows lacks the drivers required to mount the shared directory.

To make a shared directory available on a VM Guest, proceed as follows:

1. Open the guest's console in Virtual Machine Manager and either choose *View* › *Details* from the menu or click *Show virtual hardware details* in the toolbar. Choose *Add Hardware* › *Filesystem* to open the *Filesystem Passthrough* dialog.

2. *Driver* allows you to choose between a *Handle* or *Path* base driver. The default setting is *Path*. *Mode* lets you choose the security model, which influences the way file permissions are set on the host. Three options are available:

   Passthrough **(default)**

   Files on the file system are directly created with the client-user's credentials. This is similar to what NFSv3 is using.

   Squash

   Same as *Passthrough*, but failure of privileged operations like **`chown`** are ignored. This is required when KVM is not run with `root` privileges.

   Mapped

   Files are created with the file server's credentials (`qemu.qemu`). The user credentials and the client-user's credentials are saved in extended attributes. This model is recommended when host and guest domains should be kept isolated.

3. Specify the path to the directory on the VM Host Server with *Source Path*. Enter a string at *Target Path* to be used as a tag to mount the shared directory. The string of this field is a tag only, not a path on the VM Guest.

4. *Apply* the setting. If the VM Guest is currently running, you need to shut it down to apply the new setting (rebooting the guest is not sufficient).

5. Boot the VM Guest. To mount the shared directory, enter the following command:

```
> sudo mount -t 9p -o trans=virtio,version=9p2000.L,rw TAG /MOUNT_POINT
```

   To make the shared directory permanently available, add the following line to the `/etc/fstab` file:

```
TAG   /MOUNT_POINT    9p  trans=virtio,version=9p2000.L,rw    0   0
```

6.4 Using RADOS block devices with `libvirt`

RADOS Block Devices (RBD) store data in a Ceph cluster. They allow snapshotting, replication and data consistency. You can use an RBD from your `libvirt`-managed VM Guests similarly to how you use other block devices.

For more details, refer to the SUSE Enterprise Storage *Administration Guide*, chapter *Using libvirt with Ceph*. The SUSE Enterprise Storage documentation is available from https://documentation.suse.com/ses/ .

7 Basic VM Guest management

Revision History

2025-06-19

Most management tasks, such as starting or stopping a VM Guest, can either be done using the graphical application Virtual Machine Manager or on the command line using **`virsh`**. Connecting to the graphical console via VNC is only possible from a graphical user interface.

Note: Managing VM Guests on a remote VM Host Server

If started on a VM Host Server, the `libvirt` tools Virtual Machine Manager, **`virsh`**, and **`virt-viewer`** can be used to manage VM Guests on the host. However, it is also possible to manage VM Guests on a remote VM Host Server. This requires configuring remote access for `libvirt` on the host. For instructions, see *Section 8, "Connecting and authorizing"*.

To connect to such a remote host with Virtual Machine Manager, you need to set up a connection as explained in *Section 8.2.2, "Managing connections with Virtual Machine Manager"*. If connecting to a remote host using `virsh` or `virt-viewer`, you need to specify a connection URI with the parameter `-c`, for example, `virsh -c qemu+tls://saturn.example.com/system`. The form of connection URI depends on the connection type and the hypervisor—see *Section 8.2, "Connecting to a VM Host Server"* for details.

Examples in this chapter are all listed without a connection URI.

7.1 Listing VM Guests

The VM Guest listing shows all VM Guests managed by `libvirt` on a VM Host Server.

7.1.1 Listing VM Guests with Virtual Machine Manager

The main window of the Virtual Machine Manager lists all VM Guests for each VM Host Server it is connected to. Each VM Guest entry contains the machine's name, its status (*Running*, *Paused*, or *Shutoff*) displayed as an icon and literally, and a CPU usage bar.

7.1.2 Listing VM Guests with **virsh**

Use the command **virsh** `list` to get a list of VM Guests:

List all running guests

```
> virsh list
```

List all running and inactive guests

```
> virsh list --all
```

For more information and further options, see **virsh help list** or **man 1 virsh**.

7.2 Accessing the VM Guest via console

VM Guests can be accessed via a VNC connection (graphical console) or, if supported by the guest operating system, via a serial console.

7.2.1 Opening a graphical console

Opening a graphical console to a VM Guest lets you interact with the machine like a physical host via a VNC connection. If accessing the VNC server requires authentication, you are prompted to enter a user name (if applicable) and a password.

When you click into the VNC console, the cursor is “grabbed” and cannot be used outside the console anymore. To release it, press `Alt`–`Ctrl`.

Tip: Seamless (absolute) cursor movement

To prevent the console from grabbing the cursor and to enable seamless cursor movement, add a tablet input device to the VM Guest.

Certain key combinations such as `Ctrl`–`Alt`–`Del` are interpreted by the host system and are not passed to the VM Guest. To pass such key combinations to a VM Guest, open the *Send Key* menu from the VNC window and choose the desired key combination entry. The *Send Key* menu is only available when using Virtual Machine Manager and **`virt-viewer`**. With Virtual Machine Manager, you can alternatively use the “sticky key” feature as explained in *Tip: Passing key combinations to virtual machines*.

Note: Supported VNC viewers

Principally all VNC viewers can connect to the console of a VM Guest. However, if you are using SASL authentication and/or TLS/SSL connection to access the guest, the options are limited. Common VNC viewers such as **`tightvnc`** or **`tigervnc`** support neither SASL authentication nor TLS/SSL.

7.2.1.1 Opening a graphical console with Virtual Machine Manager

1. In the Virtual Machine Manager, right-click a VM Guest entry.
2. Choose *Open* from the pop-up menu.

7.2.1.2 Opening a graphical console with `virt-viewer`

`virt-viewer` is a simple VNC viewer with added functionality for displaying VM Guest consoles. For example, it can be started in “wait” mode, where it waits for a VM Guest to start before it connects. It also supports automatically reconnecting to a VM Guest that is rebooted.

`virt-viewer` addresses VM Guests by name, by ID or by UUID. Use **`virsh`** `list --all` to get this data.

To connect to a guest that is running or paused, use either the ID, UUID or name. VM Guests that are shut off do not have an ID—you can only connect to them by UUID or name.

Connect to guest with the ID `8`

```
> virt-viewer 8
```

Connect to the inactive guest named `sles12`; the connection window opens once the guest starts

```
> virt-viewer --wait sles12
```

With the `--wait` option, the connection is upheld even if the VM Guest is not running at the moment. When the guest starts, the viewer is launched.

For more information, see **`virt-viewer`** `--help` or **`man 1 virt-viewer`**.

Note: Password input on remote connections with SSH

When using **`virt-viewer`** to open a connection to a remote host via SSH, the SSH password needs to be entered twice. The first time for authenticating with `libvirt`, the second time for authenticating with the VNC server. The second password needs to be provided on the command line where virt-viewer was started.

7.2.2 Opening a serial console

Accessing the graphical console of a virtual machine requires a graphical environment on the client accessing the VM Guest. As an alternative, virtual machines managed with libvirt can also be accessed from the shell via the serial console and **`virsh`**. To open a serial console to a VM Guest named “sles12”, run the following command:

```
> virsh console sles12
```

virsh console takes two optional flags: `--safe` ensures exclusive access to the console, `--force` disconnects any existing sessions before connecting. Both features need to be supported by the guest operating system.

Being able to connect to a VM Guest via serial console requires that the guest operating system supports serial console access and is properly supported. Refer to the guest operating system manual for more information.

Tip: Enabling serial console access for SUSE Linux Enterprise and openSUSE guests

Serial console access in SUSE Linux Enterprise and openSUSE is disabled by default. To enable it, proceed as follows:

SLES 15, 16 and openSUSE

Add `console=ttyS0` to the Kernel Command Line Parameter.

SLES 11

Add `console=ttyS0` to the Kernel Command Line Parameter. Additionally, edit `/etc/inittab` and uncomment the line with the following content:

```
#S0:12345:respawn:/sbin/agetty -L 9600 ttyS0 vt102
```

7.3 Changing a VM Guest's state: start, stop, pause

Starting, stopping or pausing a VM Guest can be done with either Virtual Machine Manager or **virsh**. You can also configure a VM Guest to be automatically started when booting the VM Host Server.

When shutting down a VM Guest, you may either shut it down gracefully, or force the shutdown. The latter is equivalent to pulling the power plug on a physical host and is only recommended if there are no alternatives. Forcing a shutdown may cause file system corruption and loss of data on the VM Guest.

Tip: Graceful shutdown

To be able to perform a graceful shutdown, the VM Guest must be configured to support *ACPI*. If you have created the guest with the Virtual Machine Manager, ACPI should be available in the VM Guest.

Depending on the guest operating system, availability of ACPI may not be sufficient to perform a graceful shutdown. It is strongly recommended to test shutting down and rebooting a guest before using it in production. openSUSE or SUSE Linux Enterprise Desktop, for example, can require Polkit authorization for shutdown and reboot. Make sure this policy is turned off on all VM Guests.

If ACPI was enabled during a Windows XP/Windows Server 2003 guest installation, turning it on in the VM Guest configuration only is not sufficient. For more information, see:

- https://support.microsoft.com/en-us/kb/314088
- https://support.microsoft.com/en-us/kb/309283

Regardless of the VM Guest's configuration, a graceful shutdown is always possible from within the guest operating system.

7.3.1 Changing a VM Guest's state with Virtual Machine Manager

Changing a VM Guest's state can be done either from Virtual Machine Manager's main window, or from a VNC window.

PROCEDURE 12: STATE CHANGE FROM THE VIRTUAL MACHINE MANAGER WINDOW

1. Right-click a VM Guest entry.
2. Choose *Run*, *Pause*, or one of the *Shutdown options* from the pop-up menu.

PROCEDURE 13: STATE CHANGE FROM THE VNC WINDOW

1. Open a VNC Window as described in *Section 7.2.1.1, “Opening a graphical console with Virtual Machine Manager”*.
2. Choose *Run*, *Pause*, or one of the *Shut Down* options either from the toolbar or from the *Virtual Machine* menu.

7.3.1.1 Automatically starting a VM Guest

You can automatically start a guest when the VM Host Server boots. This feature is not enabled by default and needs to be enabled for each VM Guest individually. There is no way to activate it globally.

1. Double-click the VM Guest entry in Virtual Machine Manager to open its console.
2. Choose *View* › *Details* to open the VM Guest configuration window.
3. Choose *Boot Options* and check *Start virtual machine on host boot up*.
4. Save the new configuration with *Apply*.

7.3.2 Changing a VM Guest's state with `virsh`

In the following examples, the state of a VM Guest named “sles12” is changed.

Start

```
> virsh start sles12
```

Pause

```
> virsh suspend sles12
```

Resume (a suspended VM Guest)

```
> virsh resume sles12
```

Reboot

```
> virsh reboot sles12
```

Graceful shutdown

```
> virsh shutdown sles12
```

Force shutdown

```
> virsh destroy sles12
```

Turn on automatic start

```
> virsh autostart sles12
```

Turn off automatic start

```
> virsh autostart --disable sles12
```

7.4 Saving and restoring the state of a VM Guest

Saving a VM Guest preserves the exact state of the guest's memory. The operation is similar to *hibernating* a computer. A saved VM Guest can be quickly restored to the same running condition prior to the save operation.

When saved, the VM Guest is paused, its current memory state is saved to a file, and then the guest is stopped. The operation does not make a copy of any portion of the VM Guest's virtual disk. The time required to save the virtual machine depends on the amount of memory allocated. The VM Guest's resources are returned to the VM Host Server following a successful save operation.

The restore operation loads a VM Guest's previously saved memory state file and starts it. The guest is not booted but instead resumed at the point where it was previously saved. The operation is similar to coming out of hibernation.

`libvirt` supports several save file formats. The default format is called `raw` and consists of a sequential stream of VM Guest memory pages. The sequential layout of the `raw` format is not well suited for multiple readers and writers.

In addition to the `raw` save file format, `libvirt` also supports several compressed formats: `zstd`, `lzop`, `gzip`, `bzip2` and `xz`. Similar to the `raw` format, the compressed formats consist of a sequential stream of VM Guest memory pages, but are compressed by the named compression algorithm before being written to or read from the save file. These formats conserve space on the device holding the save file, but increase save/restore times and host CPU usage.

The `sparse` save file format uses pre-calculated, fixed offsets within the save file to read/write the VM Guest memory pages. This results in a save file that is roughly the logical size of the VM Guest's memory, although its on-disk size depends on the VM Guest's actual memory usage. With fixed offsets for the VM Guest memory pages, the `sparse` format is well suited to support multiple readers and writers, which may improve save and restore times for VM Guests with large memory allocations.

The default save file format can be changed with the `save_image_format` setting in `/etc/libvirt/qemu.conf`. The format can also be specified when performing a save operation using **`virsh`**. See *Section 7.4.2, "Saving and restoring with **`virsh`**"* for more information on save and restore with **`virsh`**.

Because the VM Guest's state is saved to a file, make sure there is enough space on the device hosting the save file. When using the `sparse` save file format, the logical save file size will be approximately the same as the VM Guest memory allocation. However, the actual on-disk file size is usually smaller and depends on the VM Guest memory usage. Unused memory in the VM Guest is not written to the save file, hence the term `sparse`.

For the `raw` save file format, the logical and on-disk file size are equivalent, and both depend on the VM Guest's memory usage. Whether using the `raw` or `sparse` format, the on-disk save file size in megabytes can be estimated by running the following command in the VM Guest:

```
> free -mh | awk '/^Mem:/ {print $3}'
```

The compressed formats will result in a smaller on-disk size, depending on the efficiency of the specified compression algorithm.

Warning: Always restore saved guests

After a successful save operation, booting or starting the VM Guest by means other than a restore operation will render the saved state file obsolete. The save file may contain file system data that has not been flushed to disk. Attempting to restore the saved state after the VM Guest has executed by other means can result in file system corruption.

Always use the same application when saving and restoring VM Guests. For example, if **`virsh`** is used to save a VM Guest, do not restore it using Virtual Machine Manager. In this case, make sure to restore using **`virsh`**.

Important: Synchronize VM Guest's time after restoring it

If you restore the VM Guest after a long pause (hours) since it was saved, its time synchronization service, for example, `chronyd`, may refuse to synchronize its time. In this case, manually synchronize VM Guest's time. For example, for KVM hosts, you can use the QEMU guest agent and instruct the guest with the **`guest-set-time`**.

7.4.1 Saving/restoring with Virtual Machine Manager

PROCEDURE 14: SAVING A VM GUEST

1. Open a VNC connection window to a VM Guest. Make sure the guest is running.
2. Choose *Virtual Machine* › *Shutdown* › *Save*.

PROCEDURE 15: RESTORING A VM GUEST

1. Open a VNC connection window to a VM Guest. Make sure the guest is not running.
2. Choose *Virtual Machine* › *Restore*.
   If the VM Guest was previously saved using Virtual Machine Manager, you are not offered an option to *Run* the guest. However, note the caveats on machines saved with **`virsh`** outlined in *Warning: Always restore saved guests*.

7.4.2 Saving and restoring with **`virsh`**

`libvirt` provides more control over the save and restore operations than Virtual Machine Manager. **`virsh save`** and **`virsh restore`** support several options to modify the behavior of the operations. In the most basic form, a VM Guest is saved by providing its name, ID, or UUID and a file name. For example:

```
> virsh save openSUSE-Leap /virtual/saves/openSUSE-Leap.vmsav
```

A basic VM Guest restore operation only requires specifying the save file name. For example:

```
> virsh restore /virtual/saves/openSUSE-Leap.vmsav
```

As VM Guest memory size increases, save and restore operations may require additional options to attain satisfactory transfer rates, particularly when the save image files are backed by high throughput storage. The VM Host Server file system cache is often counterproductive in these scenarios and should be avoided with the `bypass-cache` option. For example:

```
> virsh save --bypass-cache openSUSE-Leap /virtual/saves/openSUSE-Leap.vmsav
```

```
> virsh restore --bypass-cache /virtual/saves/openSUSE-Leap.vmsav
```

The time required to save and restore VM Guests to high throughput storage can be improved by using multiple channels to write and read the VM Guest memory pages. As noted in *Section 7.4, "Saving and restoring the state of a VM Guest"*, the `sparse` image format is required to use multiple channels. When selecting the number of channels, care must be taken to ensure the operation does not adversely affect other workloads running on the VM Host Server. When the VM Host Server resources are statically partitioned, general advice would be to use the same number of channels as physical CPUs dedicated to the VM Guest. The vCPUs of a VM Guest are stopped at the start of a save operation, thus it would be safe to use those CPU resources to save VM Guest memory pages.

The following examples save and restore a VM Guest using 4 channels, while also bypassing the VM Host Server file system cache:

```
> virsh save --bypass-cache --image-format sparse --parallel-channels 4 openSUSE-Leap /
virtual/saves/openSUSE-Leap.vmsav
```

```
> virsh restore --bypass-cache --parallel-channels 4 /virtual/saves/openSUSE-Leap.vmsav
```

The image format is encoded in the save image file and does not need to be specified during a restore operation.

For more information on save/restore and the supported options, see **`virsh help save`**, **`virsh help restore`** or **`man 1 virsh`**.

7.5 Creating and managing snapshots

VM Guest snapshots are snapshots of the complete virtual machine including the state of CPU, RAM, devices and the content of all writable disks. To use virtual machine snapshots, all the attached hard disks need to use the qcow2 disk image format, and at least one of them needs to be writable.

Snapshots let you restore the state of the machine at a particular point in time. This is useful when undoing a faulty configuration or the installation of a lot of packages. After starting a snapshot that was created while the VM Guest was shut off, you need to boot it. Any changes written to the disk afterward are lost when starting the snapshot.

Note

Snapshots are supported on KVM VM Host Servers only.

7.5.1 Terminology

There are several specific terms used to describe the types of snapshots:

Internal snapshots

Snapshots that are saved into the qcow2 file of the original VM Guest. The file holds both the saved state of the snapshot and the changes made since the snapshot was taken. The main advantage of internal snapshots is that they are all stored in one file and therefore it is easy to copy or move them across multiple machines.

External snapshots

When creating an external snapshot, the original qcow2 file is saved and made read-only, while a new qcow2 file is created to hold the changes. The original file is sometimes called a *backing* or *base* file, while the new file with all the changes is called an *overlay* or *derived* file. External snapshots are useful when performing backups of VM Guests. However, external snapshots are not supported by Virtual Machine Manager, and cannot be deleted by **`virsh`** directly.

Live snapshots

Snapshots created when the original VM Guest is running. Internal live snapshots support saving the devices, and memory and disk states, while external live snapshots with **`virsh`** support saving either the memory state, or the disk state, or both.

Offline snapshots

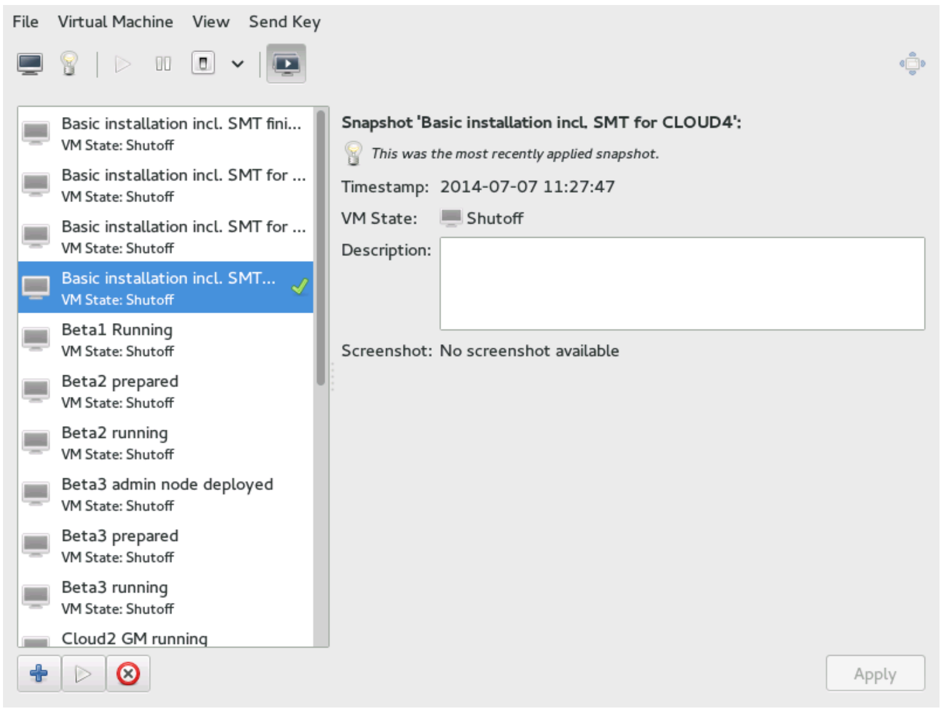
Snapshots created from a VM Guest that is shut off. This ensures data integrity as all the guest's processes are stopped and no memory is in use.

7.5.2 Creating and managing snapshots with Virtual Machine Manager

Important: Internal snapshots only

Virtual Machine Manager supports only internal snapshots, either live or offline.

To open the snapshot management view in Virtual Machine Manager, open the VNC window as described in *Section 7.2.1.1, “Opening a graphical console with Virtual Machine Manager”*. Now either choose *View* › *Snapshots* or click *Manage VM Snapshots* in the toolbar.



Basic installation incl. SMT fini...
VM State: Shutoff
Basic installation incl. SMT for ...
VM State: Shutoff
Basic installation incl. SMT for ...
VM State: Shutoff
Basic installation incl. SMT... ✓
VM State: Shutoff
Beta1 Running
VM State: Shutoff
Beta2 prepared
VM State: Shutoff
Beta2 running
VM State: Shutoff
Beta3 admin node deployed
VM State: Shutoff
Beta3 prepared
VM State: Shutoff
Beta3 running
VM State: Shutoff
Cloud2 GM running

Snapshot 'Basic installation incl. SMT for CLOUD4':
This was the most recently applied snapshot.
Timestamp: 2014-07-07 11:27:47
VM State: Shutoff
Description:
Screenshot: No screenshot available

The list of existing snapshots for the chosen VM Guest is displayed in the left-hand part of the window. The snapshot that was last started is marked with a green tick. The right-hand part of the window shows details of the snapshot currently marked in the list. These details include the snapshot's title and time stamp, the state of the VM Guest at the time the snapshot was taken and a description. Snapshots of running guests also include a screenshot. The *Description* can be changed directly from this view. Other snapshot data cannot be changed.

7.5.2.1 Creating a snapshot

To take a new snapshot of a VM Guest, proceed as follows:

1. Optionally, shut down the VM Guest to create an offline snapshot.
2. Click *Add* in the bottom left corner of the VNC window.
   The window *Create Snapshot* opens.
3. Provide a *Name* and, optionally, a description. The name cannot be changed after the snapshot has been taken. To be able to identify the snapshot later easily, use a “speaking name”.
4. Confirm with *Finish*.

7.5.2.2 Deleting a snapshot

To delete a snapshot of a VM Guest, proceed as follows:

1. Click *Delete* in the bottom left corner of the VNC window.
2. Confirm the deletion with *Yes*.

7.5.2.3 Starting a snapshot

To start a snapshot, proceed as follows:

1. Click *Run* in the bottom left corner of the VNC window.
2. Confirm the start with *Yes*.

7.5.3 Creating and managing snapshots with **`virsh`**

To list all existing snapshots for a domain (`admin_server` in the following), run the `snapshot-list` command:

```
> virsh snapshot-list --domain sle-ha-node1
 Name                     Creation Time             State
------------------------------------------------------------
 sleha_12_sp2_b2_two_node_cluster 2016-06-06 15:04:31 +0200 shutoff
 sleha_12_sp2_b3_two_node_cluster 2016-07-04 14:01:41 +0200 shutoff
 sleha_12_sp2_b4_two_node_cluster 2016-07-14 10:44:51 +0200 shutoff
 sleha_12_sp2_rc3_two_node_cluster 2016-10-10 09:40:12 +0200 shutoff
 sleha_12_sp2_gmc_two_node_cluster 2016-10-24 17:00:14 +0200 shutoff
 sleha_12_sp3_gm_two_node_cluster 2017-08-02 12:19:37 +0200 shutoff
 sleha_12_sp3_rc1_two_node_cluster 2017-06-13 13:34:19 +0200 shutoff
 sleha_12_sp3_rc2_two_node_cluster 2017-06-30 11:51:24 +0200 shutoff
 sleha_15_b6_two_node_cluster 2018-02-07 15:08:09 +0100 shutoff
 sleha_15_rc1_one-node 2018-03-09 16:32:38 +0100 shutoff
```

The snapshot that was last started is shown with the `snapshot-current command:`

```
> virsh snapshot-current --domain admin_server
Basic installation incl. SMT for CLOUD4
```

Details about a particular snapshot can be obtained by running the `snapshot-info` command:

```
> virsh snapshot-info --domain admin_server \
   -name  "Basic installation incl. SMT for CLOUD4"
```

```
Name:           Basic installation incl. SMT for CLOUD4
Domain:         admin_server
Current:        yes
State:          shutoff
Location:       internal
Parent:         Basic installation incl. SMT for CLOUD3-HA
Children:       0
Descendants:    0
Metadata:       yes
```

7.5.3.1 Creating internal snapshots

To take an internal snapshot of a VM Guest, either a live or offline, use the `snapshot-create-as` command as follows:

```
> virsh snapshot-create-as --domain admin_server❶ --name "Snapshot 1"❷ \
--description "First snapshot"❸
```

❶ Domain name. Mandatory.

❷ Name of the snapshot. It is recommended to use a “speaking name”, since that makes it easier to identify the snapshot. Mandatory.

❸ Description for the snapshot. Optional.

7.5.3.2 Creating external snapshots

With **`virsh`**, you can take external snapshots of the guest's memory state, disk state, or both.

To take both live and offline external snapshots of the guest's disk, specify the `--disk-only` option:

```
> virsh snapshot-create-as --domain admin_server --name \
 "Offline external snapshot" --disk-only
```

You can specify the `--diskspec` option to control how the external files are created:

```
> virsh snapshot-create-as --domain admin_server --name \
 "Offline external snapshot" \
 --disk-only --diskspec vda,snapshot=external,file=/path/to/snapshot_file
```

To take a live external snapshot of the guest's memory, specify the `--live` and `--memspec` options:

```
> virsh snapshot-create-as --domain admin_server --name \
 "Offline external snapshot" --live \
 --memspec snapshot=external,file=/path/to/snapshot_file
```

To take a live external snapshot of both the guest's disk and memory states, combine the `--live`, `--diskspec`, and `--memspec` options:

```
> virsh snapshot-create-as --domain admin_server --name \
 "Offline external snapshot" --live \
 --memspec snapshot=external,file=/path/to/snapshot_file
 --diskspec vda,snapshot=external,file=/path/to/snapshot_file
```

Refer to the *SNAPSHOT COMMANDS* section in **`man 1 virsh`** for more details.

7.5.3.3 Deleting a snapshot

External snapshots cannot be deleted with **`virsh`**. To delete an internal snapshot of a VM Guest and restore the disk space it occupies, use the `snapshot-delete` command:

```
> virsh snapshot-delete --domain admin_server --snapshotname "Snapshot 2"
```

7.5.3.4 Starting a snapshot

To start a snapshot, use the `snapshot-revert` command:

```
> virsh snapshot-revert --domain admin_server --snapshotname "Snapshot 1"
```

To start the current snapshot (the one the VM Guest was started off), it is sufficient to use `--current` rather than specifying the snapshot name:

```
> virsh snapshot-revert --domain admin_server --current
```

7.6 Deleting a VM Guest

By default, deleting a VM Guest using **`virsh`** removes only its XML configuration. Since attached storage is not deleted by default, you can reuse it with another VM Guest. With Virtual Machine Manager, you can also delete a guest's storage files as well.

7.6.1 Deleting a VM Guest with Virtual Machine Manager

1. In the Virtual Machine Manager, right-click a VM Guest entry.

2. From the context menu, choose *Delete*.
3. A confirmation window opens. Clicking *Delete* permanently erases the VM Guest. The deletion is not recoverable.
   You can also permanently delete the guest's virtual disk by activating *Delete Associated Storage Files*. The deletion is not recoverable either.

7.6.2 Deleting a VM Guest with **`virsh`**

To delete a VM Guest, it needs to be shut down first. It is not possible to delete a running guest. For information on shutting down, see *Section 7.3, "Changing a VM Guest's state: start, stop, pause"*.

To delete a VM Guest with **`virsh`**, run **`virsh`** `undefine` *`VM_NAME`*.

```
> virsh undefine sles12
```

There is no option to automatically delete the attached storage files. If they are managed by libvirt, delete them as described in *Section 2.2.1.4, "Deleting volumes from a storage pool"*.

7.7 Monitoring

7.7.1 Monitoring with Virtual Machine Manager

After starting Virtual Machine Manager and connecting to the VM Host Server, a CPU usage graph of all the running guests is displayed.

It is also possible to get information about disk and network usage with this tool, however, you must first activate this in *Preferences*:

1. Run **`virt-manager`**.
2. Select *Edit* › *Preferences*.
3. Change the tab from *General* to *Polling*.
4. Activate the check boxes for the kind of activity you want to see: *Poll Disk I/O*, *Poll Network I/O*, and *Poll Memory stats*.
5. If desired, also change the update interval using *Update status every n seconds*.
6. Close the *Preferences* dialog.

7. Activate the graphs that should be displayed under *View* › *Graph*.

Afterward, the disk and network statistics are also displayed in the main window of the Virtual Machine Manager.

More precise data is available from the VNC window. Open a VNC window as described in *Section 7.2.1, "Opening a graphical console"*. Choose *Details* from the toolbar or the *View* menu. The statistics are displayed from the *Performance* entry of the left-hand tree menu.

7.7.2 Monitoring with `virt-top`

`virt-top` is a command-line tool similar to the well-known process monitoring tool **`top`**. **`virt-top`** uses libvirt and therefore is capable of showing statistics for VM Guests running on different hypervisors.

By default **`virt-top`** shows statistics for all running VM Guests. Among the data that is displayed is the percentage of memory used (`%MEM`) and CPU (`%CPU`) and the uptime of the guest (`TIME`). The data is updated regularly (every three seconds by default). The following shows the output on a VM Host Server with seven VM Guests, four of them inactive:

```
virt-top 13:40:19 - x86_64 8/8CPU 1283MHz 16067MB 7.6% 0.5%
7 domains, 3 active, 3 running, 0 sleeping, 0 paused, 4 inactive D:0 O:0 X:0
CPU: 6.1%  Mem: 3072 MB (3072 MB by guests)

   ID S RDRQ WRRQ RXBY TXBY %CPU %MEM    TIME   NAME
    7 R  123    1  18K  196  5.8  6.0   0:24.35 sled12_sp1
    6 R    1    0  18K    0  0.2  6.0   0:42.51 sles12_sp1
    5 R    0    0  18K    0  0.1  6.0  85:45.67 opensuse_leap
    -                                           (Ubuntu_1410)
    -                                           (debian_780)
    -                                           (fedora_21)
    -                                           (sles11sp3)
```

By default the output is sorted by ID. Use the following key combinations to change the sort field:

`Shift`–`P` : CPU usage

`Shift`–`M` : total memory allocated by the guest

`Shift`–`T` : time

`Shift`–`I` : ID

To use any other field for sorting, press `Shift`–`F` and select a field from the list. To toggle the sort order, use `Shift`–`R` .

virt-top also supports different views on the VM Guests data, which can be changed on-the-fly by pressing the following keys:

`0` : default view
`1` : show physical CPUs
`2` : show network interfaces
`3` : show virtual disks

virt-top supports more hot keys to change the view of the data and many command line switches that affect the behavior of the program. For more information, see **man 1 virt-top**.

7.7.3 Monitoring with **kvm_stat**

kvm_stat can be used to trace KVM performance events. It monitors `/sys/kernel/debug/kvm`, so it needs the debugfs to be mounted. On SUSE Linux Enterprise Server it should be mounted by default. In case it is not mounted, use the following command:

```
> sudo mount -t debugfs none /sys/kernel/debug
```

kvm_stat can be used in three different modes:

```
kvm_stat                    # update in 1 second intervals
kvm_stat -1                 # 1 second snapshot
kvm_stat -l > kvmstats.log  # update in 1 second intervals in log format
                            # can be imported to a spreadsheet
```

EXAMPLE 8: TYPICAL OUTPUT OF **kvm_stat**

```
kvm statistics

 efer_reload                    0       0
 exits                   11378946  218130
 fpu_reload                 62144     152
 halt_exits                414866     100
 halt_wakeup               260358      50
 host_state_reload         539650     249
 hypercalls                     0       0
 insn_emulation           6227331  173067
 insn_emulation_fail            0       0
 invlpg                    227281      47
 io_exits                  113148      18
 irq_exits                 168474     127
 irq_injections          482804      123
 irq_window               51270       18
 largepages                   0        0
 mmio_exits                6925        0
 mmu_cache_miss           71820       19
 mmu_flooded              35420        9
 mmu_pde_zapped           64763       20
 mmu_pte_updated              0        0
 mmu_pte_write           213782       29
 mmu_recycled                 0        0
 mmu_shadow_zapped       128690       17
 mmu_unsync                  46       -1
 nmi_injections               0        0
 nmi_window                   0        0
 pf_fixed               1553821      857
 pf_guest               1018832      562
 remote_tlb_flush        174007       37
 request_irq                  0        0
 signal_exits                 0        0
 tlb_flush               394182      148
```

See https://clalance.blogspot.com/2009/01/kvm-performance-tools.html ↗ for further information on how to interpret these values.

8 Connecting and authorizing

Managing several VM Host Servers, each hosting multiple VM Guests, quickly becomes difficult. One benefit of `libvirt` is the ability to connect to several VM Host Servers at once, providing a single interface to manage all VM Guests and to connect to their graphical console.

To ensure only authorized users can connect, `libvirt` offers several connection types (via TLS, SSH, Unix sockets, and TCP) that can be combined with different authorization mechanisms (socket, Polkit, SASL and Kerberos).

8.1 Authentication

The power to manage VM Guests and to access their graphical console is something that should be restricted to a well-defined circle of persons. To achieve this goal, you can use the following authentication techniques on the VM Host Server:

- Access control for Unix sockets with permissions and group ownership. This method is available for `libvirtd` connections only.
- Access control for Unix sockets with Polkit. This method is available for local `libvirtd` connections only.
- User name and password authentication with SASL (Simple Authentication and Security Layer). This method is available for both `libvirtd` and VNC connections. Using SASL does not require real user accounts on the server, since it uses its own database to store user names and passwords. Connections authenticated with SASL are encrypted.
- Kerberos authentication. This method, available for `libvirtd` connections only, is not covered in this manual. Refer to https://libvirt.org/auth.html#ACL_server_kerberos ↗ for details.
- Single password authentication. This method is available for VNC connections only.

Important: Authentication for `libvirtd` and VNC needs to be configured separately

Access to the VM Guest's management functions (via `libvirtd`) and to its graphical console always needs to be configured separately. When restricting access to the management tools, these restrictions do *not* automatically apply to VNC connections.

When accessing VM Guests from remote via TLS/SSL connections, access can be indirectly controlled on each client by restricting read permissions to the certificate's key file to a certain group. See *Section 8.3.2.5, “Restricting access (security considerations)”* for details.

8.1.1 `libvirtd` authentication

`libvirtd` authentication is configured in `/etc/libvirt/libvirtd.conf`. The configuration made here applies to all `libvirt` tools such as the Virtual Machine Manager or **`virsh`**.

`libvirt` offers two sockets: a read-only socket for monitoring purposes and a read-write socket to be used for management operations. Access to both sockets can be configured independently. By default, both sockets are owned by `root.root`. Default access permissions on the read-write socket are restricted to the user `root` (`0700`) and fully open on the read-only socket (`0777`).

The following instructions describe how to configure access permissions for the read-write socket. The same instructions also apply to the read-only socket. All configuration steps need to be carried out on the VM Host Server.

Note: Default authentication settings on SUSE Linux Enterprise Server

The default authentication method on SUSE Linux Enterprise Server is access control for Unix sockets. Only the user `root` may authenticate. When accessing the `libvirt` tools as a non-root user directly on the VM Host Server, you need to provide the `root` password through Polkit once. You are then granted access for the current and for future sessions.

Alternatively, you can configure `libvirt` to allow “system” access to non-privileged users. See *Section 8.2.1, ““system” access for non-privileged users”* for details.

RECOMMENDED AUTHORIZATION METHODS

Local connections

- *Section 8.1.1.2, “Local access control for Unix sockets with Polkit”*
- *Section 8.1.1.1, “Access control for Unix sockets with permissions and group ownership”*

Remote tunnel over SSH

Section 8.1.1.1, “Access control for Unix sockets with permissions and group ownership”

Remote TLS/SSL connection

- *Section 8.1.1.3, “User name and password authentication with SASL”*
- none (access controlled on the client side by restricting access to the certificates)

8.1.1.1 Access control for Unix sockets with permissions and group ownership

To grant access for non-`root` accounts, configure the sockets to be owned and accessible by a certain group (`libvirt` in the following example). This authentication method can be used for local and remote SSH connections.

1. In case it does not exist, create the group that should own the socket:

```
> sudo groupadd libvirt
```

Important: Group needs to exist

The group must exist before restarting `libvirtd`. If not, the restart fails.

2. Add the desired users to the group:

```
> sudo usermod --append --groups libvirt tux
```

3. Change the configuration in `/etc/libvirt/libvirtd.conf` as follows:

```
unix_sock_group = "libvirt"(1)
      unix_sock_rw_perms = "0770"(2)
      auth_unix_rw = "none"(3)
```

(1) Group ownership is set to the group `libvirt`.

(2) Sets the access permissions for the socket (`srwxrwx---`).

(3) Disables other authentication methods (Polkit or SASL). Access is solely controlled by the socket permissions.

4. Restart `libvirtd`:

```
> sudo systemctl start libvirtd
```

8.1.1.2 Local access control for Unix sockets with Polkit

Access control for Unix sockets with Polkit is the default authentication method on SUSE Linux Enterprise Server for non-remote connections. Therefore, no `libvirt` configuration changes are needed. With Polkit authorization enabled, permissions on both sockets default to `0777` and each application trying to access a socket needs to authenticate via Polkit.

Important: Polkit authentication for local connections only

Authentication with Polkit can only be used for local connections on the VM Host Server itself, since Polkit does not handle remote authentication.

Two policies for accessing `libvirt`'s sockets exist:

- *org.libvirt.unix.monitor*: accessing the read-only socket
- *org.libvirt.unix.manage*: accessing the read-write socket

By default, the policy for accessing the read-write socket is to authenticate with the `root` password once and grant the privilege for the current and for future sessions.

To grant users access to a socket without having to provide the `root` password, you need to create a rule in `/etc/polkit-1/rules.d`. Create the file `/etc/polkit-1/rules.d/10-grant-libvirt` with the following content to grant access to the read-write socket to all members of the group `libvirt`:

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.libvirt.unix.manage" && subject.isInGroup("libvirt")) {
    return polkit.Result.YES;
    }
    });
```

8.1.1.3 User name and password authentication with SASL

SASL provides user name and password authentication and data encryption (digest-md5, by default). Since SASL maintains its own user database, the users do not need to exist on the VM Host Server. SASL is required by TCP connections and on top of TLS/SSL connections.

Important: Plain TCP and SASL with digest-md5 encryption

Using digest-md5 encryption on an otherwise not encrypted TCP connection does not provide enough security for production environments. It is recommended to only use it in testing environments.

Tip: SASL authentication on top of TLS/SSL

Access from remote TLS/SSL connections can be indirectly controlled on the *client side* by restricting access to the certificate's key file. However, this may prove error-prone when dealing with many clients. Using SASL with TLS adds security by additionally controlling access on the server side.

To configure SASL authentication, proceed as follows:

1. Change the configuration in /etc/libvirt/libvirtd.conf as follows:

   a. To enable SASL for TCP connections:

      ```
      auth_tcp = "sasl"
      ```

   b. To enable SASL for TLS/SSL connections:

      ```
      auth_tls = "sasl"
      ```

2. Restart libvirtd:

   ```
   > sudo systemctl restart libvirtd
   ```

3. The libvirt SASL configuration file is located at /etc/sasl2/libvirtd.conf. Normally, there is no need to change the defaults. However, if using SASL on top of TLS, you may turn off session encryption to avoid additional overhead (TLS connections are already encrypted) by commenting the line setting the mech_list parameter. Only do this for TLS/SASL. For TCP connections, this parameter must be set to digest-md5.

   ```
   #mech_list: digest-md5
   ```

4. By default, no SASL users are configured, so no logins are possible. Use the following commands to manage users:

   Add the user tux

   ```
   saslpasswd2 -a libvirt tux
   ```

   Delete the user tux

   ```
   saslpasswd2 -a libvirt -d tux
   ```

   List existing users

   ```
   sasldblistusers2 -f /etc/libvirt/passwd.db
   ```

Tip: **`virsh`** and SASL authentication

When using SASL authentication, you are prompted for a user name and password every time you issue a **`virsh`** command. Avoid this by using **`virsh`** in shell mode.

8.1.2 VNC authentication

Since access to the graphical console of a VM Guest is not controlled by `libvirt`, but by the specific hypervisor, it is always necessary to additionally configure VNC authentication. The main configuration file is `/etc/libvirt/<hypervisor>.conf`. This section describes the QEMU/KVM hypervisor, so the target configuration file is `/etc/libvirt/qemu.conf`.

Two authentication types are available: SASL and single-password authentication. If you are using SASL for `libvirt` authentication, it is strongly recommended to use it for VNC authentication as well—it is possible to share the same database.

A third method to restrict access to the VM Guest is to enable the use of TLS encryption on the VNC server. This requires the VNC clients to have access to x509 client certificates. By restricting access to these certificates, access can indirectly be controlled on the client side. Refer to *Section 8.3.2.4.2, "VNC over TLS/SSL: client configuration"* for details.

8.1.2.1 User name and password authentication with SASL

SASL provides user name and password authentication and data encryption. Since SASL maintains its own user database, the users do not need to exist on the VM Host Server. As with SASL authentication for `libvirt`, you may use SASL on top of TLS/SSL connections. Refer to *Section 8.3.2.4.2, "VNC over TLS/SSL: client configuration"* for details on configuring these connections.

To configure SASL authentication for VNC, proceed as follows:

1. Create a SASL configuration file. It is recommended to use the existing `libvirt` file. If you have already configured SASL for `libvirt` and are planning to use the same settings, including the same user name and password database, a simple link is suitable:

   ```
   > sudo ln -s /etc/sasl2/libvirt.conf /etc/sasl2/qemu.conf
   ```

   If are setting up SASL for VNC only or you are planning to use a different configuration than for `libvirt`, copy the existing file to use as a template:

   ```
   > sudo cp /etc/sasl2/libvirt.conf /etc/sasl2/qemu.conf
   ```

Then edit it according to your needs.

2. Change the configuration in `/etc/libvirt/qemu.conf` as follows:

```
vnc_listen = "0.0.0.0"
      vnc_sasl = 1
      sasldb_path: /etc/libvirt/qemu_passwd.db
```

The first parameter enables VNC to listen on all public interfaces (rather than to the local host only), and the second parameter enables SASL authentication.

3. By default, no SASL users are configured, so no logins are possible. Use the following commands to manage users:

Add the user `tux`

```
> saslpasswd2 -f /etc/libvirt/qemu_passwd.db -a qemu tux
```

Delete the user `tux`

```
> saslpasswd2 -f /etc/libvirt/qemu_passwd.db -a qemu -d tux
```

List existing users

```
> sasldblistusers2 -f /etc/libvirt/qemu_passwd.db
```

4. Restart `libvirtd`:

```
> sudo systemctl restart libvirtd
```

5. Restart all VM Guests that have been running before changing the configuration. VM Guests that have not been restarted cannot use SASL authentication for VNC connects.

Note: Supported VNC viewers

SASL authentication is currently supported by Virtual Machine Manager and **`virt-viewer`**. Both viewers also support TLS/SSL connections.

8.1.2.2 Single password authentication

Access to the VNC server may also be controlled by setting a VNC password. You can either set a global password for all VM Guests or set individual passwords for each guest. The latter requires editing the VM Guest's configuration files.

Note: Always set a global password

If you are using single password authentication, it is good practice to set a global password even if setting passwords for each VM Guest. This protects your virtual machines with a “fallback” password if you forget to set a per-machine password. The global password is only used if no other password is set for the machine.

PROCEDURE 16: SETTING A GLOBAL VNC PASSWORD

1. Change the configuration in `/etc/libvirt/qemu.conf` as follows:

   ```
   vnc_listen = "0.0.0.0"
         vnc_password = "PASSWORD"
   ```

   The first parameter enables VNC to listen on all public interfaces (rather than to the local host only), and the second parameter sets the password. The maximum length of the password is eight characters.

2. Restart `libvirtd`:

   ```
   > sudo systemctl restart libvirtd
   ```

3. Restart all VM Guests that have been running before changing the configuration. VM Guests that have not been restarted cannot use password authentication for VNC connects.

PROCEDURE 17: SETTING A VM GUEST SPECIFIC VNC PASSWORD

1. Change the configuration in `/etc/libvirt/qemu.conf` as follows to enable VNC to listen on all public interfaces (rather than to the local host only).

   ```
   vnc_listen = "0.0.0.0"
   ```

2. Open the VM Guest's XML configuration file in an editor. Replace *`VM_NAME`* in the following example with the name of the VM Guest. The editor that is used defaults to `$EDITOR`. If that variable is not set, **`vi`** is used.

```
> virsh edit VM_NAME
```

3. Search for the element `<graphics>` with the attribute `type='vnc'`, for example:

   ```
   <graphics type='vnc' port='-1' autoport='yes'/>
   ```

4. Add the `passwd=PASSWORD` attribute, save the file and exit the editor. The maximum length of the password is eight characters.

   ```
   <graphics type='vnc' port='-1' autoport='yes' passwd='PASSWORD'/>
   ```

5. Restart `libvirtd`:

   ```
   > sudo systemctl restart libvirtd
   ```

6. Restart all VM Guests that have been running before changing the configuration. VM Guests that have not been restarted cannot use password authentication for VNC connects.

Warning: Security of the VNC protocol

The VNC protocol is not considered to be safe. Although the password is sent encrypted, it may be vulnerable when an attacker can sniff both the encrypted password and the encryption key. Therefore, it is recommended to use VNC with TLS/SSL or tunneled over SSH. **`virt-viewer`**, and Virtual Machine Manager support both methods.

8.2 Connecting to a VM Host Server

To connect to a hypervisor with `libvirt`, you need to specify a uniform resource identifier (URI). This URI is needed with **`virsh`** and **`virt-viewer`** (except when working as `root` on the VM Host Server) and is optional for the Virtual Machine Manager. Although the latter can be called with a connection parameter (for example, **`virt-manager -c qemu:///system`**), it also offers a graphical interface to create connection URIs. See *Section 8.2.2, “Managing connections with Virtual Machine Manager”* for details.

```
HYPERVISOR❶+PROTOCOL❷://USER@REMOTE❸/CONNECTION_TYPE❹
```

❶ Specify the hypervisor. SUSE Linux Enterprise Server currently supports the following hypervisors: `test` (testing purposes), `qemu` (KVM) option. This parameter is mandatory.

2. When connecting to a remote host, specify the protocol here. It can be one of: `ssh` (connection via SSH tunnel), `tcp` (TCP connection with SASL/Kerberos authentication), `tls` (TLS/SSL encrypted connection with authentication via x509 certificates).
3. When connecting to a remote host, specify the user name and the remote host name. If no user name is specified, the user name that has called the command (`$USER`) is used. See below for more information. For TLS connections, the host name needs to be specified exactly as in the x509 certificate.
4. When connecting to the `QEMU/KVM` hypervisor, two connection types are accepted: `system` for full access rights, or `session` for restricted access. Since `session` access is not supported on SUSE Linux Enterprise Server, this documentation focuses on `system` access.

EXAMPLE HYPERVISOR CONNECTION URIS

`test:///default`
: Connect to the local testing hypervisor.

`qemu+ssh://tux@mercury.example.com/system`
: Connect to the QEMU hypervisor on the remote host mercury.example.com. The connection is established via an SSH tunnel.

`qemu+tls://saturn.example.com/system`
: Connect to the QEMU hypervisor on the remote host mercury.example.com. The connection is established using TLS/SSL.

For more details and examples, refer to the `libvirt` documentation at https://libvirt.org/uri.html.

Note: User names in URIs

A user name needs to be specified when using Unix socket authentication (regardless of whether using the user/password authentication scheme or Polkit). This applies to all SSH and local connections.

There is no need to specify a user name when using SASL authentication (for TCP or TLS connections) or when doing no additional server-side authentication for TLS connections. With SASL, the user name is not evaluated—you are prompted for an SASL user/password combination in any case.

8.2.1 “system” access for non-privileged users

As mentioned above, a connection to the QEMU hypervisor can be established using two different protocols: `session` and `system`. A “session” connection is spawned with the same privileges as the client program. Such a connection is intended for desktop virtualization, since it is restricted, for example, no USB/PCI device assignments, no virtual network setup, limited remote access to `libvirtd`.

The “system” connection intended for server virtualization has no functional restrictions but is, by default, only accessible by `root`. However, with the addition of the DAC (Discretionary Access Control) driver to `libvirt`, it is now possible to grant non-privileged users “system” access. To grant “system” access to the user `tux`, proceed as follows:

PROCEDURE 18: GRANTING “SYSTEM” ACCESS TO A REGULAR USER

1. Enable access via Unix sockets, as described in *Section 8.1.1.1, “Access control for Unix sockets with permissions and group ownership”*. In that example, access to libvirt is granted to all members of the group `libvirt` and `tux` made a member of this group. This ensures that `tux` can connect using **`virsh`** or Virtual Machine Manager.

2. Edit `/etc/libvirt/qemu.conf` and change the configuration as follows:

   ```
   user = "tux"
        group = "libvirt"
        dynamic_ownership = 1
   ```

   This ensures that the VM Guests are started by `tux` and that resources bound to the guest, for example, virtual disks, can be accessed and modified by `tux`.

3. Make `tux` a member of the group `kvm`:

   ```
   > sudo usermod --append --groups kvm tux
   ```

   This step is needed to grant access to `/dev/kvm`, which is required to start VM Guests.

4. Restart `libvirtd`:

   ```
   > sudo systemctl restart libvirtd
   ```

8.2.2 Managing connections with Virtual Machine Manager

The Virtual Machine Manager uses a `Connection` for every VM Host Server it manages. Each connection contains all VM Guests on the respective host. By default, a connection to the local host is already configured and connected.

All configured connections are displayed in the Virtual Machine Manager main window. Active connections are marked with a small triangle, which you can click to fold or unfold the list of VM Guests for this connection.

Inactive connections are listed gray and are marked with `Not Connected`. Either double-click or right-click it and choose *Connect* from the context menu. You can also *Delete* an existing connection from this menu.

Note: Editing existing connections

It is not possible to edit an existing connection. To change a connection, create a new one with the desired parameters and delete the “old” one.

To add a new connection in the Virtual Machine Manager, proceed as follows:

1. Choose *File* › *Add Connection*
2. Choose the host's *Hypervisor* (*QEMU/KVM*)
3. *(Optional)* To set up a remote connection, choose *Connect to remote host*. For more information, see *Section 8.3, “Configuring remote connections”*.
   In case of a remote connection, specify the *Hostname* of the remote machine in the format *`USERNAME@REMOTE _HOST`*.

   #### Important: Specifying a user name

   There is no need to specify a user name for TCP and TLS connections: in these cases, it is not evaluated. However, for SSH connections, specifying a user name is necessary when you want to connect as a user other than `root`.

4. If you do not want the connection to be automatically started when starting the Virtual Machine Manager, deactivate *Autoconnect*.
5. Finish the configuration by clicking *Connect*.

8.3 Configuring remote connections

A major benefit of `libvirt` is the ability to manage VM Guests on different remote hosts from a central location. This section gives detailed instructions on how to configure server and client to allow remote connections.

8.3.1 Remote tunnel over SSH (`qemu+ssh`)

Enabling a remote connection that is tunneled over SSH on the VM Host Server only requires the ability to accept SSH connections. Make sure the SSH daemon is started (**`systemctl status sshd`**) and that the ports for service `SSH` are opened in the firewall.

User authentication for SSH connections can be done using traditional file user/group ownership and permissions as described in *Section 8.1.1.1, "Access control for Unix sockets with permissions and group ownership"*. Connecting as user tux (`qemu+ssh://tuxsIVname;/system`) works out of the box and does not require additional configuration on the `libvirt` side.

When connecting via SSH `qemu+ssh://USER@SYSTEM` you need to provide the password for *USER*. This can be avoided by copying your public key to `~USER/.ssh/authorized_keys` on the VM Host Server.

8.3.2 Remote TLS/SSL connection with x509 certificate (`qemu+tls`

Using TCP connections with TLS/SSL encryption and authentication via x509 certificates is much more complicated to set up than SSH, but it is a lot more scalable. Use this method if you need to manage several VM Host Servers with a varying number of administrators.

8.3.2.1 Basic concept

TLS (Transport Layer Security) encrypts the communication between two computers by using certificates. The computer starting the connection is always considered the "client", using a "client certificate", while the receiving computer is always considered the "server", using a "server certificate". This scenario applies, for example, if you manage your VM Host Servers from a central desktop.

If connections are initiated from both computers, each needs to have a client *and* a server certificate. This is the case, for example, if you migrate a VM Guest from one host to another.

Each x509 certificate has a matching private key file. Only the combination of certificate and private key file can identify itself correctly. To assure that a certificate was issued by the assumed owner, it is signed and issued by a central certificate called certificate authority (CA). Both the client and the server certificates must be issued by the same CA.

Important: User authentication

Using a remote TLS/SSL connection only ensures that two computers are allowed to communicate in a certain direction. Restricting access to certain users can indirectly be achieved on the client side by restricting access to the certificates. For more information, see *Section 8.3.2.5, “Restricting access (security considerations)”*.

`libvirt` also supports user authentication on the server with SASL. For more information, see *Section 8.3.2.6, “Central user authentication with SASL for TLS sockets”*.

8.3.2.2 Configuring the VM Host Server

The VM Host Server is the machine receiving connections. Therefore, the *server* certificates need to be installed. The CA certificate needs to be installed, too. When the certificates are in place, TLS support can be turned on for `libvirt`.

1. Create the server certificate and export it together with the respective CA certificate.

2. Create the following directories on the VM Host Server:

   ```
   > sudo mkdir -p /etc/pki/CA/ /etc/pki/libvirt/private/
   ```

   Install the certificates as follows:

   ```
   > sudo /etc/pki/CA/cacert.pem
         > sudo /etc/pki/libvirt/servercert.pem
         > sudo /etc/pki/libvirt/private/serverkey.pem
   ```

   #### Important: Restrict access to certificates

   Make sure to restrict access to certificates, as explained in *Section 8.3.2.5, “Restricting access (security considerations)”*.

3. Enable TLS support by enabling the relevant socket and restarting `libvirtd`:

```
        > sudo systemctl stop libvirtd.service
        > sudo systemctl enable --now libvirtd-tls.socket
        > sudo systemctl start libvirtd.service
```

4. By default, `libvirt` uses the TCP port 16514 for accepting secure TLS connections. Open this port in the firewall.

Important: Restarting `libvirtd` with TLS enabled

If you enable TLS for `libvirt`, the server certificates need to be in place, otherwise restarting `libvirtd` fails. You also need to restart `libvirtd` in case you change the certificates.

8.3.2.3 Configuring the client and testing the setup

The client is the machine initiating connections. Therefore the *client* certificates need to be installed. The CA certificate needs to be installed, too.

1. Create the client certificate and export it together with the respective CA certificate.

2. Create the following directories on the client:

```
> sudo mkdir -p /etc/pki/CA/ /etc/pki/libvirt/private/
```

Install the certificates as follows:

```
> sudo /etc/pki/CA/cacert.pem
      > sudo /etc/pki/libvirt/clientcert.pem
      > sudo /etc/pki/libvirt/private/clientkey.pem
```

Important: Restrict access to certificates

Make sure to restrict access to certificates, as explained in *Section 8.3.2.5, “Restricting access (security considerations)”*.

3. Test the client/server setup by issuing the following command. Replace *`mercury.example.com`* with the name of your VM Host Server. Specify the same fully qualified host name as used when creating the server certificate.

```
#QEMU/KVM
```

```
virsh -c qemu+tls://mercury.example.com/system list --all
```

If your setup is correct, you can see a list of all VM Guests registered with `libvirt` on the VM Host Server.

8.3.2.4 Enabling VNC for TLS/SSL connections

Currently, VNC communication over TLS is only supported by a few tools. Common VNC viewers such as **`tightvnc`** or **`tigervnc`** do not support TLS/SSL.

8.3.2.4.1 VNC over TLS/SSL: VM Host Server configuration

To access the graphical console via VNC over TLS/SSL, you need to configure the VM Host Server as follows:

1. Open ports for the service `VNC` in your firewall.
2. Create a directory `/etc/pki/libvirt-vnc` and link the certificates into this directory as follows:

   ```
   > sudo mkdir -p /etc/pki/libvirt-vnc && cd /etc/pki/libvirt-vnc
          > sudo ln -s /etc/pki/CA/cacert.pem ca-cert.pem
          > sudo ln -s /etc/pki/libvirt/servercert.pem server-cert.pem
          > sudo ln -s /etc/pki/libvirt/private/serverkey.pem server-key.pem
   ```

3. Edit `/etc/libvirt/qemu.conf` and set the following parameters:

   ```
   vnc_listen = "0.0.0.0"
          vnc_tls = 1
          vnc_tls_x509_verify = 1
   ```

4. Restart the `libvirtd`:

   ```
   > sudo systemctl restart libvirtd
   ```

   Important: VM Guests need to be restarted

   The VNC TLS setting is only set when starting a VM Guest. Therefore, you need to restart all machines that have been running before making the configuration change.

8.3.2.4.2 VNC over TLS/SSL: client configuration

The only action needed on the client side is to place the x509 client certificates in a location recognized by the client of choice. However, Virtual Machine Manager and **`virt-viewer`** expect the certificates in a different location. Virtual Machine Manager can either read from a system-wide location applying to all users, or from a per-user location.

Virtual Machine Manager (`virt-manager`)

To connect to the remote host, Virtual Machine Manager requires the setup explained in *Section 8.3.2.3, "Configuring the client and testing the setup"*. To be able to connect via VNC, the client certificates also need to be placed in the following locations:

System-wide location

`/etc/pki/CA/cacert.pem`
`/etc/pki/libvirt-vnc/clientcert.pem`
`/etc/pki/libvirt-vnc/private/clientkey.pem`

Per-user location

`/etc/pki/CA/cacert.pem`
`~/.pki/libvirt-vnc/clientcert.pem`
`~/.pki/libvirt-vnc/private/clientkey.pem`

`virt-viewer`

`virt-viewer` only accepts certificates from a system-wide location:

`/etc/pki/CA/cacert.pem`
`/etc/pki/libvirt-vnc/clientcert.pem`
`/etc/pki/libvirt-vnc/private/clientkey.pem`

Important: Restrict access to certificates

Make sure to restrict access to certificates, as explained in *Section 8.3.2.5, "Restricting access (security considerations)"*.

8.3.2.5 Restricting access (security considerations)

Each x509 certificate consists of two pieces: the public certificate and a private key. A client can only authenticate using both pieces. Therefore, any user that has read access to the client certificate and its private key can access your VM Host Server. On the other hand, an arbitrary machine equipped with the full server certificate can pretend to be the VM Host Server. Since this is not desirable, access to at least the private key files needs to be restricted as much as possible. The easiest way to control access to a key file is to use access permissions.

Server certificates

Server certificates need to be readable for QEMU processes. On SUSE Linux Enterprise Server QEMU, processes started from `libvirt` tools are owned by `root`, so it is sufficient if the `root` can read the certificates:

```
> chmod 700 /etc/pki/libvirt/private/
       > chmod 600 /etc/pki/libvirt/private/serverkey.pem
```

If you change the ownership for QEMU processes in `/etc/libvirt/qemu.conf`, you also need to adjust the ownership of the key file.

System-wide client certificates

To control access to a key file that is available system-wide, restrict read access to a certain group, so that only members of that group can read the key file. In the following example, a group `libvirt` is created, and group ownership of the `clientkey.pem` file and its parent directory is set to `libvirt`. Afterward, the access permissions are restricted to owner and group. Finally, the user `tux` is added to the group `libvirt`, and thus can access the key file.

```
CERTPATH="/etc/pki/libvirt/"
       # create group libvirt
       groupadd libvirt
       # change ownership to user root and group libvirt
       chown root.libvirt $CERTPATH/private $CERTPATH/clientkey.pem
       # restrict permissions
       chmod 750 $CERTPATH/private
       chmod 640 $CERTPATH/private/clientkey.pem
       # add user tux to group libvirt
       usermod --append --groups libvirt tux
```

Per-user certificates

User-specific client certificates for accessing the graphical console of a VM Guest via VNC need to be placed in the user's home directory in `~/.pki`. Contrary to SSH, for example, the VNC viewer using these certificates does not check the access permissions of the private key file. Therefore, it is solely the user's responsibility to make sure the key file is not readable by others.

8.3.2.5.1 Restricting access from the server side

By default, every client that is equipped with appropriate client certificates may connect to a VM Host Server accepting TLS connections. Therefore, it is possible to use additional server-side authentication with SASL as described in *Section 8.1.1.3, “User name and password authentication with SASL”*.

It is also possible to restrict access with a allowlist of DNs (distinguished names), so only clients with a certificate matching a DN from the list can connect.

Add a list of allowed DNs to `tls_allowed_dn_list` in `/etc/libvirt/libvirtd.conf`. This list may contain wild cards. Do not specify an empty list, since that would result in refusing all connections.

```
tls_allowed_dn_list = [
     "C=US,L=Provo,O=SUSE Linux Products GmbH,OU=*,CN=venus.example.com,EMAIL=*",
     "C=DE,L=Nuremberg,O=SUSE Linux Products GmbH,OU=Documentation,CN=*"]
```

Get the distinguished name of a certificate with the following command:

```
> certtool -i --infile /etc/pki/libvirt/clientcert.pem | grep "Subject:"
```

Restart `libvirtd` after having changed the configuration:

```
> sudo systemctl restart libvirtd
```

8.3.2.6 Central user authentication with SASL for TLS sockets

A direct user authentication via TLS is not possible—this is handled indirectly on each client via the read permissions for the certificates as explained in *Section 8.3.2.5, “Restricting access (security considerations)”*. However, if a central, server-based user authentication is needed, `libvirt` also allows to use SASL (Simple Authentication and Security Layer) on top of TLS for direct user authentication. See *Section 8.1.1.3, “User name and password authentication with SASL”* for configuration details.

8.3.2.7 Troubleshooting

8.3.2.7.1 Virtual Machine Manager/**`virsh`** cannot connect to server

Check the following in the given order:

Is it a firewall issue (TCP port 16514 needs to be open on the server)?

Is the client certificate (certificate and key) readable by the user that has started Virtual Machine Manager/ **virsh**?
Has the same full qualified host name as in the server certificate been specified with the connection?
Is TLS enabled on the server (`listen_tls = 1`)?
Has `libvirtd` been restarted on the server?

8.3.2.7.2 VNC connection fails

Ensure that you can connect to the remote server using Virtual Machine Manager. If so, check whether the virtual machine on the server has been started with TLS support. The virtual machine's name in the following example is `sles`.

```
> ps ax | grep qemu | grep "\-name sles" | awk -F" -vnc " '{ print FS $2 }'
```

If the output does not begin with a string similar to the following, the machine has not been started with TLS support and must be restarted.

```
 -vnc 0.0.0.0:0,tls,x509verify=/etc/pki/libvirt
```

9 Migrating VM Guests

Revision History

2025-06-12

One of the major advantages of virtualization is that VM Guests are portable. When a VM Host Server needs maintenance, or when the host becomes overloaded, the guests can be moved to another VM Host Server. KVM even support “live” migrations, during which the VM Guest is constantly available.

9.1 Types of migration

Depending on the required scenario, there are three ways you can migrate virtual machines (VMs).

Live migration

The source VM continues to run while its configuration and memory are transferred to the target host. When the transfer is complete, the source VM is suspended and the target VM is resumed.

Live migration is useful for VMs that need to be online without any downtime.

Note

VMs experiencing heavy I/O load or frequent memory page writes are challenging to live migrate. In such cases, consider using non-live or offline migration.

Non-live migration

The source VM is suspended and its configuration and memory are transferred to the target host. Then the target VM is resumed.

Non-live migration is more reliable than live migration, although it creates downtime for the VM. If downtime is tolerable, non-live migration can be an option for VMs that are difficult to live migrate.

Offline migration

The VM definition is transferred to the target host. The source VM is not stopped and the target VM is not resumed.

Offline migration can be used to migrate inactive VMs.

Important

With offline migration, the `--persistent` option must be used.

9.2 Migration requirements

To successfully migrate a VM Guest to another VM Host Server, the following requirements need to be met:

- The source and target systems must have the same architecture.
- Storage devices must be accessible from both machines, for example, via NFS or iSCSI. For more information, see *Section 6, "Advanced storage topics"*.

This is also true for CD-ROM or floppy images that are connected during the move. However, you can disconnect them before the move as described in *Section 5.11, "Ejecting and changing floppy or CD/DVD-ROM media with Virtual Machine Manager"*.

- `libvirtd` needs to run on both VM Host Servers and you must be able to open a remote `libvirt` connection between the target and the source host (or vice versa). Refer to *Section 8.3, "Configuring remote connections"* for details.
- If a firewall is running on the target host, ports need to be opened to allow the migration. If you do not specify a port during the migration process, `libvirt` chooses one from the range 49152:49215. Make sure that either this range (recommended) or a dedicated port of your choice is opened in the firewall on the *target host*.
- The source and target machines should be in the same subnet on the network, otherwise networking fails after the migration.
- All VM Host Servers participating in migration must have the same UID for the qemu user and the same GIDs for the kvm, qemu and libvirt groups.
- No running or paused VM Guest with the same name must exist on the target host. If a shut-down machine with the same name exists, its configuration is overwritten.
- All CPU models, except the *host cpu* model, are supported when migrating VM Guests.
- The *SATA* disk device type is not migratable.
- File system pass-through feature is incompatible with migration.
- The VM Host Server and VM Guest need to have proper timekeeping installed.
- No physical devices can be passed from host to guest. Live migration is currently not supported when using devices with PCI pass-through or *SR-IOV*. If live migration needs to be supported, use software virtualization (paravirtualization or full virtualization).
- The cache mode setting is an important setting for migration.
- Backward migration, for example, from SLES 15 SP2 to 15 SP1, is not supported.
- SUSE strives to support live migration of VM Guests from a VM Host Server running a service pack under LTSS to a VM Host Server running a newer service pack within the same SLES major version. For example, VM Guest migration from a SLES 12 SP2 host to a SLES 12 SP5 host. SUSE only performs minimal testing of LTSS-to-newer migration scenarios and recommends thorough on-site testing before attempting to migrate critical VM Guests.

- The image directory should be located at the same path on both hosts.
- All hosts should be on the same level of microcode (especially the Spectre microcode updates). This can be achieved by installing the latest updates of SUSE Linux Enterprise Server on all hosts.

9.3 Live-migrating with Virtual Machine Manager

When using the Virtual Machine Manager to migrate VM Guests, it does not matter on which machine it is started. You can start Virtual Machine Manager on the source or the target host or even on a third host. In the latter case, you need to be able to open remote connections to both the target and the source host.

1. Start Virtual Machine Manager and establish a connection to the target or the source host. If the Virtual Machine Manager was started neither on the target nor the source host, connections to both hosts need to be opened.

2. Right-click the VM Guest that you want to migrate and choose *Migrate*. Make sure the guest is running or paused—it is not possible to migrate guests that are shut down.

   ### Tip: Increasing the speed of the migration

   To increase the speed of the migration, pause the VM Guest. This is the equivalent of “non-live migration” described in *Section 9.1, “Types of migration”*.

3. Choose a *New Host* for the VM Guest. If the desired target host does not show up, make sure that you are connected to the host.
   To change the default options for connecting to the remote host, under *Connection*, set the *Mode*, and the target host's *Address* (IP address or host name) and *Port*. If you specify a *Port*, you must also specify an *Address*.
   Under *Advanced options*, choose whether the move should be permanent (default) or temporary, using *Temporary move*.
   Additionally, there is the option *Allow unsafe*, which allows migrating without disabling the cache of the VM Host Server. This can speed up the migration but only works when the current configuration allows for a consistent view of the VM Guest storage without using `cache="none"`/`0_DIRECT`.

Note: Bandwidth option

In recent versions of Virtual Machine Manager, the option of setting a bandwidth for the migration has been removed. To set a specific bandwidth, use **`virsh`** instead.

4. To perform the migration, click *Migrate*.
   When the migration is complete, the *Migrate* window closes and the VM Guest is now listed on the new host in the Virtual Machine Manager window. The original VM Guest is still available on the source host in the shut-down state.

9.4 Migrating with `virsh`

To migrate a VM Guest with **`virsh`** `migrate`, you need to have direct or remote shell access to the VM Host Server, because the command needs to be run on the host. The migration command looks like this:

```
> virsh migrate [OPTIONS] VM_ID_or_NAME CONNECTION_URI [--migrateuri
 tcp://REMOTE_HOST:PORT]
```

The most important options are listed below. See **`virsh help migrate`** for a full list.

`--live`
: Does a live migration. If not specified, the guest is paused during the migration (“non-live migration”).

`--suspend`
: Leaves the VM paused on the target host during live or non-live migration.

`--persistent`
: Persists the migrated VM on the target host. Without this option, the VM is not included in the list of domains reported by **`virsh list --all`** when shut down.

`--undefinesource`
: When specified, the VM Guest definition on the source host is deleted after a successful migration. However, virtual disks attached to this guest are *not* deleted.

`--parallel --parallel-connections NUM_OF_CONNECTIONS`

Parallel migration can be used to increase migration data throughput in cases where a single migration thread is not capable of saturating the network link between source and target hosts. On hosts with 40 GB network interfaces, it may require four migration threads to saturate the link. With parallel migration, the time required to migrate large memory VMs can be reduced.

The following examples use mercury.example.com as the source system and jupiter.example.com as the target system; the VM Guest's name is `opensuse131` with ID `37`.

Non-live migration with default parameters

```
> virsh migrate 37 qemu+ssh://tux@jupiter.example.com/system
```

Transient live migration with default parameters

```
> virsh migrate --live opensuse131 qemu+ssh://tux@jupiter.example.com/system
```

Persistent live migration; delete VM definition on source

```
> virsh migrate --live --persistent --undefinesource 37 \
qemu+tls://tux@jupiter.example.com/system
```

Non-live migration using port 49152

```
> virsh migrate opensuse131 qemu+ssh://tux@jupiter.example.com/system \
--migrateuri tcp://@jupiter.example.com:49152
```

Live migration transferring all used storage

```
> virsh migrate --live --persistent --copy-storage-all \
opensuse156 qemu+ssh://tux@jupiter.example.com/system
```

Important

When migrating VM's storage using the `--copy-storage-all` option, the storage must be placed in a `libvirt` storage pool. The target storage pool must exist with an identical type and name as the source pool.

To obtain the XML representation of the source pool, use the following command:

```
> sudo virsh pool-dumpxml EXAMPLE_VM > EXAMPLE_POOL.xml
```

To create and start the storage pool on the target host, copy its XML representation there and use the following commands:

```
> sudo virsh pool-define EXAMPLE_POOL.xml
> sudo virsh pool-start EXAMPLE_VM
```

Note: Transient compared to persistent migrations

By default, **`virsh migrate`** creates a temporary (transient) copy of the VM Guest on the target host. A shut-down version of the original guest description remains on the source host. A transient copy is deleted from the server after it is shut down.

To create a permanent copy of a guest on the target host, use the switch `--persistent`. A shut-down version of the original guest description remains on the source host, too. Use the option `--undefinesource` together with `--persistent` for a “real” move where a permanent copy is created on the target host and the version on the source host is deleted.

It is not recommended to use `--undefinesource` without the `--persistent` option, since this results in the loss of both VM Guest definitions when the guest is shut down on the target host.

9.5 Step-by-step example

9.5.1 Exporting the storage

First, you need to export the storage to share the guest image between hosts. This can be done by an NFS server. In the following example, we want to share the `/volume1/VM` directory for all machines that are on the network 10.0.1.0/24. We are using a SUSE Linux Enterprise NFS server. As root user, edit the `/etc/exports` file and add:

```
/volume1/VM 10.0.1.0/24  (rw,sync,no_root_squash)
```

You need to restart the NFS server:

```
> sudo systemctl restart nfsserver
> sudo exportfs
/volume1/VM      10.0.1.0/24
```

9.5.2 Defining the pool on the target hosts

On each host where you want to migrate the VM Guest, the pool must be defined to be able to access the volume (that contains the Guest image). Our NFS server IP address is 10.0.1.99, its share is the `/volume1/VM` directory, and we want to get it mounted in the `/var/lib/libvirt/images/VM` directory. The pool name is *VM*. To define this pool, create a `VM.xml` file with the following content:

```
<pool type='netfs'>
  <name>VM</name>
  <source>
    <host name='10.0.1.99'/>
    <dir path='/volume1/VM'/>
    <format type='auto'/>
  </source>
  <target>
    <path>/var/lib/libvirt/images/VM</path>
    <permissions>
      <mode>0755</mode>
      <owner>-1</owner>
      <group>-1</group>
    </permissions>
  </target>
  </pool>
```

Then load it into `libvirt` using the **`pool-define`** command:

```
# virsh pool-define VM.xml
```

An alternative way to define this pool is to use the **`virsh`** command:

```
# virsh pool-define-as VM --type netfs --source-host 10.0.1.99 \
     --source-path /volume1/VM --target /var/lib/libvirt/images/VM
Pool VM created
```

The following commands assume that you are in the interactive shell of **`virsh`**, which can also be reached by using the command **`virsh`** without any arguments. Then the pool can be set to start automatically at host boot (autostart option):

```
virsh # pool-autostart VM
Pool VM marked as autostarted
```

To disable the autostart:

```
virsh # pool-autostart VM --disable
Pool VM unmarked as autostarted
```

Check if the pool is present:

```
virsh # pool-list --all
 Name                 State      Autostart
-------------------------------------------
 default              active     yes
 VM                   active     yes

virsh # pool-info VM
Name:           VM
UUID:           42efe1b3-7eaa-4e24-a06a-ba7c9ee29741
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       2,68 TiB
Allocation:     2,38 TiB
Available:      306,05 GiB
```

Warning: Pool needs to exist on all target hosts

Remember: This pool must be defined on each host where you want to be able to migrate your VM Guest.

9.5.3 Creating the volume

The pool has been defined—now we need a volume that contains the disk image:

```
virsh # vol-create-as VM sled12.qcow2 8G --format qcow2
Vol sled12.qcow2 created
```

The volume names shown are used later to install the guest with virt-install.

9.5.4 Creating the VM Guest

Let us create a SUSE Linux Enterprise Server VM Guest with the **`virt-install`** command. The *VM* pool is specified with the **`--disk`** option, *cache=none* is recommended if you do not want to use the **`--unsafe`** option while doing the migration.

```
# virt-install --connect qemu:///system --virt-type kvm --name \
   sles15 --memory 1024 --disk vol=VM/sled12.qcow2,cache=none --cdrom \
   /mnt/install/ISO/SLE-15-Server-DVD-x86_64-Build0327-Media1.iso --graphics \
   vnc --os-variant sled15
Starting install...
Creating domain...
```

9.5.5 Migrating the VM Guest

Everything is ready to do the migration now. Run the **`migrate`** command on the VM Host Server that is currently hosting the VM Guest, and choose the target.

```
virsh # migrate --live sled12 --verbose qemu+ssh://IP/Hostname/system
Password:
Migration: [ 12 %]
```

10 I/O virtualization

VM Guests not only share CPU and memory resources of the host system, but also the I/O subsystem. Because software I/O virtualization techniques deliver less performance than bare metal, hardware solutions that deliver almost “native” performance have been developed recently. SUSE Linux Enterprise Server supports the following I/O virtualization techniques:

Full virtualization

Fully Virtualized (FV) drivers emulate widely supported real devices, which can be used with an existing driver in the VM Guest. The guest is also called *Hardware Virtual Machine* (HVM). Since the physical device on the VM Host Server may differ from the emulated one, the hypervisor needs to process all I/O operations before handing them over to the physical device. Therefore all I/O operations need to traverse two software layers, a process that not only significantly impacts I/O performance, but also consumes CPU time.

Paravirtualization

Paravirtualization (PV) allows direct communication between the hypervisor and the VM Guest. With less overhead involved, performance is much better than with full virtualization. However, paravirtualization requires either the guest operating system to be modified to support the paravirtualization API, or availability of paravirtualized drivers. For a list of guest operating systems supporting paravirtualization, refer to the section *Availability of paravirtualized drivers* in the article Virtualization Limits and Support (https://documentation.suse.com/sles/16.0/html/SLES-virtualization-support/) ↗.

PVHVM

This type of virtualization enhances HVM (see *Full virtualization*) with paravirtualized (PV) drivers, and PV interrupt and timer handling.

VFIO

VFIO stands for *Virtual Function I/O* and is a new user-level driver framework for Linux. It replaces the traditional KVM PCI Pass-Through device assignment. The VFIO driver exposes direct device access to user space in a secure memory Input/Output Memory Management Unit (IOMMU) protected environment. With VFIO, a VM Guest can directly access hardware devices on the VM Host Server (pass-through), avoiding performance issues caused by emulation in performance critical paths. This method does not allow to share devices—each device can only be assigned to a single VM Guest. VFIO needs to be supported by the VM Host Server CPU, chipset and the BIOS/EFI.
Compared to the legacy KVM PCI device assignment, VFIO has the following advantages:

- Resource access is compatible with UEFI Secure Boot.
- Device is isolated and its memory access protected.
- Offers a user space device driver with more flexible device ownership model.
- Is independent of KVM technology, and not bound to x86 architecture only.

In SUSE Linux Enterprise Server the USB and PCI pass-through methods of device assignment are considered deprecated and are superseded by the VFIO model.

SR-IOV

The latest I/O virtualization technique, Single Root I/O Virtualization SR-IOV combines the benefits of the aforementioned techniques—performance and the ability to share a device with several VM Guests. SR-IOV requires special I/O devices, that are capable of replicating resources so they appear as multiple separate devices. Each such “pseudo” device can be directly used by a single guest. However, for network cards for example the number of concurrent queues that can be used is limited, potentially reducing performance for the VM Guest compared to paravirtualized drivers. On the VM Host Server, SR-IOV must be supported by the I/O device, the CPU and chipset, the BIOS/EFI and the hypervisor—.

Important: Requirements for VFIO and SR-IOV

To be able to use the VFIO and SR-IOV features, the VM Host Server needs to fulfill the following requirements:

- IOMMU needs to be enabled in the BIOS/EFI.
- For Intel CPUs, the kernel parameter `intel_iommu=on` needs to be provided on the kernel command line. For more information, see https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/kernel-parameters.txt#L1951 ↗.
- The VFIO infrastructure needs to be available. This can be achieved by loading the kernel module `vfio_pci`.

11 For more information

For further steps in virtualization, refer to the following sources:

- Managing virtual machines with `libvirt` (https://documentation.suse.com/smart/virtualization-cloud/html/concept-manage-virtual-machines-libvirt/concept-manage-virtual-machines-libvirt.html) ↗
- Configuring Virtual Machines with Virtual Machine Manager (https://documentation.suse.com/smart/virtualization-cloud/html/task-configure-virtual-machine-manager/task-configure-virtual-machine-manager.html) ↗
- Assigning Host Devices to Virtual Machines (https://documentation.suse.com/smart/virtualization-cloud/html/vm-assign-pci-device/vm-assign-pci-device.html) ↗
- Configuring a Virtual Disk Cache Mode (https://documentation.suse.com/smart/virtualization-cloud/html/virtual-disk-cache-mode-configure/virtual-disk-cache-mode-configure.html) ↗

12 Legal Notice

A GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/copyleft/ ↗.

Acronyms

ACPI

Advanced Configuration and Power Interface (ACPI) specification provides an open standard for device configuration and power management by the operating system.

AER

Advanced Error Reporting

AER is a capability provided by the PCI Express specification which allows for reporting of PCI errors and recovery from some of them.

APIC

Advanced Programmable Interrupt Controller (APIC) is a family of interrupt controllers.

BDF

Bus:Device:Function

Notation used to succinctly describe PCI and PCIe devices.

CG

Control Groups

Feature to limit, account and isolate resource usage (CPU, memory, disk I/O, etc.).

EDF

Earliest Deadline First

This scheduler provides weighted CPU sharing in an intuitive way and uses real-time algorithms to ensure time guarantees.

EPT

Extended Page Tables

Performance in a virtualized environment is close to that in a native environment. Virtualization does create some overheads, however. These come from the virtualization of the CPU, the *MMU*, and the I/O devices. In some recent x86 processors AMD and Intel have begun to provide hardware extensions to help bridge this performance gap. In 2006, both vendors introduced their first generation hardware support for x86 virtualization with AMD-Virtualization (AMD-V) and Intel® VT-x technologies. Recently Intel introduced its second generation of hardware support that incorporates MMU-virtualization, called Extended Page Tables (EPT). EPT-enabled systems can improve performance compared to using shadow paging for *MMU* virtualization. EPT increases memory access latencies for a few workloads. This cost can be reduced by effectively using large pages in the guest and the hypervisor.

HAP

High Assurance Platform

HAP combines hardware and software technologies to improve workstation and network security.

HVM

Hardware Virtual Machine.

IOMMU

Input/Output Memory Management Unit

IOMMU (AMD* technology) is a memory management unit (*MMU*) that connects a direct memory access-capable (DMA-capable) I/O bus to the main memory.

KSM

Kernel Same Page Merging

KSM allows for automatic sharing of identical memory pages between guests to save host memory. KVM is optimized to use KSM if enabled on the VM Host Server.

MMU

Memory Management Unit

is a computer hardware component responsible for handling accesses to memory requested by the CPU. Its functions include translation of virtual addresses to physical addresses (that is, virtual memory management), memory protection, cache control, bus arbitration and in simpler computer architectures (especially 8-bit systems) bank switching.

PAE

Physical Address Extension

32-bit x86 operating systems use Physical Address Extension (PAE) mode to enable addressing of more than 4 GB of physical memory. In PAE mode, page table entries (PTEs) are 64 bits in size.

PCID

Process-context identifiers

These are a facility by which a logical processor may cache information for multiple linear-address spaces so that the processor may retain cached information when software switches to a different linear address space. INVPCID instruction is used for fine-grained *TLB* flush, which is benefit for kernel.

PCIe

Peripheral Component Interconnect Express

PCIe was designed to replace older PCI, PCI-X and AGP bus standards. PCIe has numerous improvements including a higher maximum system bus throughput, a lower I/O pin count and smaller physical footprint. Moreover it also has a more detailed error detection and reporting mechanism (*AER*), and a native hotplug functionality. It is also backward compatible with PCI.

PSE and PSE36

Page Size Extended

PSE refers to a feature of x86 processors that allows for pages larger than the traditional 4 KiB size. PSE-36 capability offers 4 more bits, in addition to the normal 10 bits, which are used inside a page directory entry pointing to a large page. This allows a large page to be located in 36-bit address space.

PT

Page Table

A page table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses. Virtual addresses are those unique to the accessing process. Physical addresses are those unique to the hardware (RAM).

QXL

QXL is a cirrus VGA framebuffer (8M) driver for virtualized environment.

RVI or NPT

Rapid Virtualization Indexing, Nested Page Tables

An AMD second generation hardware-assisted virtualization technology for the processor memory management unit (*MMU*).

SATA

Serial ATA

SATA is a computer bus interface that connects host bus adapters to mass storage devices such as hard disks and optical drives.

Seccomp2-based sandboxing

Sandboxed environment where only predetermined system calls are permitted for added protection against malicious behavior.

SPICE

Simple Protocol for Independent Computing Environments

TCG

Tiny Code Generator

Instructions are emulated rather than executed by the CPU.

THP

Transparent Huge Pages

This allows CPUs to address memory using pages larger than the default 4 KB. This helps reduce memory consumption and CPU cache usage. KVM is optimized to use THP (via madvise and opportunistic methods) if enabled on the VM Host Server.

TLB

Translation Lookaside Buffer

TLB is a cache that memory management hardware uses to improve virtual address translation speed. All current desktop, notebook, and server processors use a TLB to map virtual and physical address spaces, and it is nearly always present in any hardware that uses virtual memory.

VCPU

A scheduling entity, containing each state for virtualized CPU.

VDI

Virtual Desktop Infrastructure

VFIO

Since kernel v3.6; a new method of accessing PCI devices from user space called VFIO.

VHS

Virtualization Host Server

VM root

VMM will run in *VMX* root operation and guest software will run in *VMX* non-root operation. Transitions between *VMX* root operation and *VMX* non-root operation are called *VMX* transitions.

VMCS

Virtual Machine Control Structure

VMX non-root operation and VMX transitions are controlled by a data structure called a virtual-machine control structure (VMCS). Access to the VMCS is managed through a component of processor state called the VMCS pointer (one per logical processor). The value of the VMCS pointer is the 64-bit address of the VMCS. The VMCS pointer is read and written using the instructions VMPTRST and VMPTRLD. The *VMM* configures a VMCS using the VMREAD, VMWRITE, and VMCLEAR instructions. A *VMM* could use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the *VMM* could use a different VMCS for each virtual processor.

VMDq

Virtual Machine Device Queue

Multi-queue network adapters exist which support multiple VMs at the hardware level, having separate packet queues associated to the different hosted VMs (by means of the IP addresses of the VMs).

VMM

Virtual Machine Monitor (Hypervisor)

When the processor encounters an instruction or event of interest to the Hypervisor (*VMM*), it exits from guest mode back to the VMM. The VMM emulates the instruction or other event, at a fraction of native speed, and then returns to guest mode. The transitions from guest mode to the VMM and back again are high-latency operations, during which guest execution is completely stalled.

VMX

Virtual Machine eXtensions

VPID

New support for software control of *TLB* (VPID improves *TLB* performance with small *VMM* development effort).

VT-d

Virtualization Technology for Directed I/O

Like *IOMMU* for Intel* (https://software.intel.com/en-us/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices) ↗.

vTPM

Component to establish end-to-end integrity for guests via Trusted Computing.

Glossary

General

Create Virtual Machine Wizard

Virtual Machine Manager provides a graphical interface to guide you through the steps to create virtual machines. It can also be run in text mode by entering **`virt-install`** at a command prompt in the host environment.

hardware-assisted

Intel* and AMD* provide virtualization hardware-assisted technology. This reduces the frequency of VM IN/OUT (fewer VM traps), because software is a major source of overhead, and increases the efficiency (the execution is done by the hardware). Moreover, this reduces the memory footprint, provides better resource control, and allows secure assignment of specific I/O devices.

Host Environment

The desktop or command line environment that allows interaction with the host computer's environment. It provides a command line environment and can also include a graphical desktop, such as GNOME or IceWM. The host environment runs as a special type of virtual machine that has privileges to control and manage other virtual machines.

Hypervisor

The software that coordinates the low-level interaction between virtual machines and the underlying physical computer hardware.

Paravirtualized Frame Buffer

The video output device that drives a video display from a memory buffer containing a complete frame of data for virtual machine displays running in paravirtual mode.

VHS

Virtualization Host Server

The physical computer running a SUSE virtualization platform software. The virtualization environment consists of the hypervisor, the host environment, virtual machines and associated tools, commands and configuration files. Other commonly used terms include host, Host Computer, Host Machine (HM), Virtual Server (VS), Virtual Machine Host (VMH), and VM Host Server (VHS).

VirtFS

VirtFS is a new paravirtualized file system interface designed for improving pass-through technologies in the KVM environment. It is based on the VirtIO framework.

Virtual Machine

A virtualized PC environment (VM) capable of hosting a guest operating system and associated applications. Could be also called a VM Guest.

Virtual Machine Manager

A software program that provides a graphical user interface for creating and managing virtual machines.

Virtualized

A guest operating system or application running on a virtual machine.

CPU

CPU capping

Virtual CPU capping allows you to set vCPU capacity to 1–100 percent of the physical CPU capacity.

CPU hotplugging

CPU hotplugging is used to describe the functions of replacing/adding/removing a CPU without shutting down the system.

CPU over-commitment

Virtual CPU over-commitment is the ability to assign more virtual CPUs to VMs than the actual number of physical CPUs present in the physical system. This procedure does not increase the overall performance of the system, but may be useful for testing purposes.

CPU pinning

Processor affinity, or CPU pinning enables the binding and unbinding of a process or a thread to a central processing unit (CPU) or a range of CPUs.

Network

Bridged Networking

A type of network connection that lets a virtual machine be identified on an external network as a unique identity that is separate from and unrelated to its host computer.

Empty Bridge

A type of network bridge that has no physical network device or virtual network device provided by the host. This lets virtual machines communicate with other virtual machines on the same host but not with the host or on an external network.

External Network

The network outside a host's internal network environment.

Internal Network

A type of network configuration that restricts virtual machines to their host environment.

Local Bridge

A type of network bridge that has a virtual network device but no physical network device provided by the host. This lets virtual machines communicate with the host and other virtual machines on the host. Virtual machines can communicate on an external network through the host.

Network Address Translation (NAT)

A type of network connection that lets a virtual machine use the IP address and MAC address of the host.

No Host Bridge

A type of network bridge that has a physical network device but no virtual network device provided by the host. This lets virtual machines communicate on an external network but not with the host. This lets you separate virtual machine network communications from the host environment.

Traditional Bridge

A type of network bridge that has both a physical network device and a virtual network device provided by the host.

Storage

AHCI

The Advanced Host Controller Interface (AHCI) is a technical standard defined by Intel* that specifies the operation of Serial ATA (SATA) host bus adapters in a non-implementation-specific manner.

Block Device

Data storage devices, such as CD-ROM drives or disk drives, that move data in the form of blocks. Partitions and volumes are also considered block devices.

File-Backed Virtual Disk

A virtual disk based on a file, also called a disk image file.

Raw Disk

A method of accessing data on a disk at the individual byte level instead of through its file system.

Sparse image file

A disk image file that does not reserve its entire amount of disk space but expands as data is written to it.

xvda

The drive designation given to the first virtual disk on a paravirtual machine.