# Red Hat Enterprise Linux 6 Virtualization Host Configuration and Guest Installation Guide

Installing and configuring your virtual environment

Jiri Herrmann
Laura Bailey
Tahlia Richardson
Scott Radvan
Dayle Parker

# Red Hat Enterprise Linux 6 Virtualization Host Configuration and Guest Installation Guide

---

## Installing and configuring your virtual environment

Jiri Herrmann
Red Hat Customer Content Services
jherrman@redhat.com

Tahlia Richardson
Red Hat Customer Content Services

Dayle Parker
Red Hat Customer Content Services

Laura Bailey
Red Hat Customer Content Services

Scott Radvan
Red Hat Customer Content Services

## Legal Notice



## Abstract

This guide covers KVM packages, compatibility and restrictions. Also included are host configuration details and instructions for installing guest virtual machines of different types, PCI device configuration and SR-IOV.

# Table of Contents

# Chapter 1. Introduction

## 1.1. What is in This Guide?

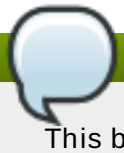
This guide provides information on installing virtualization software and configuring guest machines on a Red Hat Enterprise Linux virtualization host.

The initial chapters in this guide outline the prerequisites to enable a Red Hat Enterprise Linux host machine to deploy virtualization. System requirements, compatible hardware, support and product restrictions are covered in detail.

Basic host configuration, including mandatory and optional virtualization packages, are covered in Chapter 5, *Installing the Virtualization Packages*.

Guest virtual machine installation is covered in detail starting from Chapter 6, *Guest Virtual Machine Installation Overview*, with procedures for installing fully virtualized Red Hat Enterprise Linux guests and Windows paravirtualized guests using virt-manager and virsh.

More detailed information on networking, PCI device configuration, SR-IOV, KVM guest timing management, and troubleshooting help for libvirt and SR-IOV is included later in the guide.



**Note**

This book provides guidance for virtualization host configuration and guest installation. For more detailed system configuration information, refer to the *Red Hat Enterprise Linux — Virtualization Administration Guide*.

# Chapter 2. System Requirements

This chapter lists system requirements for successfully running virtual machines, referred to as VMs on Red Hat Enterprise Linux 6. Virtualization is available for Red Hat Enterprise Linux 6 on the Intel 64 and AMD64 architecture.

The KVM hypervisor is provided with Red Hat Enterprise Linux 6.

For information on installing the virtualization packages, see Chapter 5, *Installing the Virtualization Packages*.

### Minimum system requirements

- 6 GB free disk space.
- 2 GB of RAM.

### Recommended system requirements

- One processor core or hyper-thread for the maximum number of virtualized CPUs in a guest virtual machine and one for the host.
- 2 GB of RAM plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for each virtual machine.

  Most guest operating systems will require at least 6GB of disk space, but the additional storage space required for each guest depends on its image format.

  For guest virtual machines using raw images, the guest's total required space **(total for raw format)** is equal to or greater than the sum of the space required by the guest's raw image files **(images)**, the 6GB space required by the host operating system **(host)**, and the swap space that guest will require **(swap)**.

  **Equation 2.1. Calculating required space for guest virtual machines using raw images**

  total for raw format = images + host + swap

  For qcow images, you must also calculate the expected maximum storage requirements of the guest **(total for qcow format)**, as qcow and qcow2 images grow as required. To allow for this expansion, first multiply the expected maximum storage requirements of the guest **(expected maximum guest storage)** by 1.01, and add to this the space required by the host **(host)**, and the necessary swap space **(swap)**.

  **Equation 2.2. Calculating required space for guest virtual machines using qcow images**

  total for qcow format = (expected maximum guest storage * 1.01) + host + swap

Guest virtual machine requirements are further outlined in the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* in Chapter 6. Overcommitting with KVM.

### Calculating Swap Space

Using swap space can provide additional memory beyond the available physical memory. The swap partition is used for swapping underused memory to the hard drive to speed up memory performance. The default size of the swap partition is calculated from the physical RAM of the host.

Red Hat Knowledgebase contains an article on safely and efficiently determining an appropriate size for the swap partition, available here: https://access.redhat.com/site/solutions/15244.

### KVM Requirements

The KVM hypervisor requires:

- an Intel processor with the Intel VT-x and Intel 64 extensions for x86-based systems, or
- an AMD processor with the AMD-V and the AMD64 extensions.

Refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* to determine if your processor has the virtualization extensions.

### Storage Support

The guest virtual machine storage methods are:

- files on local storage,
- physical disk partitions,
- locally connected physical LUNs,
- LVM partitions,
- NFS shared file systems,
- iSCSI,
- GFS2 clustered file systems,
- Fibre Channel-based LUNs, and
- Fibre Channel over Ethernet (FCoE).

# Chapter 3. KVM Guest Virtual Machine Compatibility

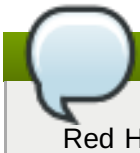
To verify whether your processor supports the virtualization extensions and for information on enabling the virtualization extensions if they are disabled, refer to the *Red Hat Enterprise Linux Virtualization Administration Guide*.

## 3.1. Red Hat Enterprise Linux 6 Support Limits

Red Hat Enterprise Linux 6 servers have certain support limits.

The following URLs explain the processor and memory amount limitations for Red Hat Enterprise Linux:

- For host systems: http://www.redhat.com/resourcelibrary/articles/articles-red-hat-enterprise-linux-6-technology-capabilities-and-limits
- For hypervisors: http://www.redhat.com/resourcelibrary/articles/virtualization-limits-rhel-hypervisors



**Note**

Red Hat Enterprise Linux 6.5 now supports 4TiB of memory per KVM guest.

The following URL is a complete reference showing supported operating systems and host and guest combinations:

- http://www.redhat.com/resourcelibrary/articles/enterprise-linux-virtualization-support

## 3.2. Supported CPU Models

Every hypervisor has its own policy for which CPU features the guest will see by default. The set of CPU features presented to the guest by QEMU/KVM depends on the CPU model chosen in the guest virtual machine configuration. ***qemu32*** and ***qemu64*** are basic CPU models but there are other models (with additional features) available.

Red Hat Enterprise Linux 6 supports the use of the following QEMU CPU model definitions:

```
<!-- This is only a partial file, only containing the CPU models. The
XML file has more information (including supported features per model)
which you can see when you open the file yourself -->
<cpus>
  <arch name='x86'>
...

    <!-- Intel-based QEMU generic CPU models -->
    <model name='pentium'>
      <model name='486'/>
     </model>

    <model name='pentium2'>
      <model name='pentium'/>
```

```
    </model>

    <model name='pentium3'>
      <model name='pentium2'/>
    </model>

    <model name='pentiumpro'>
    </model>

    <model name='coreduo'>
      <model name='pentiumpro'/>
      <vendor name='Intel'/>
    </model>

    <model name='n270'>
      <model name='coreduo'/>
    </model>

    <model name='core2duo'>
      <model name='n270'/>
    </model>

    <!-- Generic QEMU CPU models -->
    <model name='qemu32'>
      <model name='pentiumpro'/>
    </model>

    <model name='kvm32'>
      <model name='qemu32'/>
    </model>

    <model name='cpu64-rhel5'>
      <model name='kvm32'/>
    </model>

    <model name='cpu64-rhel6'>
      <model name='cpu64-rhel5'/>
    </model>

    <model name='kvm64'>
      <model name='cpu64-rhel5'/>
    </model>

    <model name='qemu64'>
      <model name='kvm64'/>
    </model>

    <!-- Intel CPU models -->
    <model name='Conroe'>
      <model name='pentiumpro'/>
      <vendor name='Intel'/>
    </model>

    <model name='Penryn'>
      <model name='Conroe'/>
    </model>
```

```
    <model name='Nehalem'>
      <model name='Penryn'/>
    </model>

    <model name='Westmere'>
      <model name='Nehalem'/>
      <feature name='aes'/>
    </model>

    <model name='SandyBridge'>
      <model name='Westmere'/>
    </model>

    <model name='Haswell'>
      <model name='SandyBridge'/>
    </model>

    <!-- AMD CPUs -->
    <model name='athlon'>
      <model name='pentiumpro'/>
      <vendor name='AMD'/>
     </model>

    <model name='phenom'>
      <model name='cpu64-rhel5'/>
      <vendor name='AMD'/>
    </model>

    <model name='Opteron_G1'>
      <model name='cpu64-rhel5'/>
      <vendor name='AMD'/>
    </model>

    <model name='Opteron_G2'>
      <model name='Opteron_G1'/>
    </model>

    <model name='Opteron_G3'>
      <model name='Opteron_G2'/>
    </model>

    <model name='Opteron_G4'>
      <model name='Opteron_G2'/>
    </model>

    <model name='Opteron_G5'>
      <model name='Opteron_G4'/>
    </model>
  </arch>
</cpus>
```

**Note**

A full list of supported CPU models and recognized CPUID flags can also be found using the **`qemu-kvm -cpu ?`** command.

# Chapter 4. Virtualization Restrictions

This chapter covers additional support and product restrictions of the virtualization packages in Red Hat Enterprise Linux 6.

## 4.1. KVM Restrictions

The following restrictions apply to the KVM hypervisor:

**Maximum vCPUs per guest**

The maximum amount of virtual CPUs that is supported per guest varies depending on which minor version of Red Hat Enterprise Linux 6 you are using as a host machine. The release of 6.0 introduced a maximum of 64, while 6.3 introduced a maximum of 160. Currently with the release of 6.7, a maximum of 240 virtual CPUs per guest is supported.

**Constant TSC bit**

Systems without a Constant Time Stamp Counter require additional configuration. Refer to Chapter 14, *KVM Guest Timing Management* for details on determining whether you have a Constant Time Stamp Counter and configuration steps for fixing any related issues.

**Memory overcommit**

KVM supports memory overcommit and can store the memory of guest virtual machines in swap. A virtual machine will run slower if it is swapped frequently. Red Hat Knowledgebase has an article on safely and efficiently determining an appropriate size for the swap partition, available here: https://access.redhat.com/site/solutions/15244. When KSM is used for memory overcommitting, make sure that the swap size follows the recommendations described in this article.

**Important**

When device assignment is in use, all virtual machine memory must be statically pre-allocated to enable DMA with the assigned device. Memory overcommit is therefore not supported with device assignment.

**CPU overcommit**

It is not recommended to have more than 10 virtual CPUs per physical processor core. Customers are encouraged to use a capacity planning tool in order to determine the CPU overcommit ratio. Estimating an ideal ratio is difficult as it is highly dependent on each workload. For instance, a guest virtual machine may consume 100% CPU on one use case, and multiple guests may be completely idle on another.

Red Hat does not support running more vCPUs to a single guest than the amount of overall physical cores that exist on the system. While Hyperthreads can be considered as cores, their performance can also vary from one scenario to the next, and they should not be expected to perform as well as regular cores.

Refer to the *Red Hat Enterprise Linux Virtualization Administration Guide* for tips and recommendations on overcommitting CPUs.

**Virtualized SCSI devices**

SCSI emulation is not supported with KVM in Red Hat Enterprise Linux.

### Virtualized IDE devices

KVM is limited to a maximum of four virtualized (emulated) IDE devices per guest virtual machine.

### PCI devices

Red Hat Enterprise Linux 6 supports 32 PCI device slots per virtual machine, and 8 PCI functions per device slot. This gives a theoretical maximum of 256 PCI functions per guest when multi-function capabilities are enabled.

However, this theoretical maximum is subject to the following limitations:

- Each virtual machine supports a maximum of 8 assigned device functions.
- 4 PCI device slots are configured with 5 emulated devices (two devices are in slot 1) by default. However, users can explicitly remove 2 of the emulated devices that are configured by default if the guest operating system does not require them for operation (the video adapter device in slot 2; and the memory balloon driver device in the lowest available slot, usually slot 3). This gives users a supported functional maximum of 30 PCI device slots per virtual machine.

The following restrictions also apply to PCI device assignment:

- PCI device assignment (attaching PCI devices to virtual machines) requires host systems to have AMD IOMMU or Intel VT-d support to enable device assignment of PCI-e devices.
- For parallel/legacy PCI, only single devices behind a PCI bridge are supported.
- Multiple PCIe endpoints connected through a non-root PCIe switch require ACS support in the PCIe bridges of the PCIe switch. To disable this restriction, edit the **`/etc/libvirt/qemu.conf`** file and insert the line:

  ```
  relaxed_acs_check=1
  ```

- Red Hat Enterprise Linux 6 has limited PCI configuration space access by guest device drivers. This limitation could cause drivers that are dependent on PCI configuration space to fail configuration.
- Red Hat Enterprise Linux 6.2 introduced interrupt remapping as a requirement for PCI device assignment. If your platform does not provide support for interrupt remapping, circumvent the KVM check for this support with the following command as the root user at the command line prompt:

  ```
  # echo 1 >
  /sys/module/kvm/parameters/allow_unsafe_assigned_interrupts
  ```

### Migration restrictions

Device assignment refers to physical devices that have been exposed to a virtual machine, for the exclusive use of that virtual machine. Because device assignment uses hardware on the specific host where the virtual machine runs, migration and save/restore are not supported when device assignment is in use. If the guest operating system supports hot-plugging, assigned devices can be removed prior to the migration or save/restore operation to enable this feature.

Live migration is only possible between hosts with the same CPU type (that is, Intel to Intel or AMD to AMD only).

For live migration, both hosts must have the same value set for the No eXecution (NX) bit, either **on** or **off**.

For migration to work, **cache=none** must be specified for all block devices opened in write mode.

**Warning**

Failing to include the **cache=none** option can result in disk corruption.

### Storage restrictions

There are risks associated with giving guest virtual machines write access to entire disks or block devices (such as **/dev/sdb**). If a guest virtual machine has access to an entire block device, it can share any volume label or partition table with the host machine. If bugs exist in the host system's partition recognition code, this can create a security risk. Avoid this risk by configuring the host machine to ignore devices assigned to a guest virtual machine.

**Warning**

Failing to adhere to storage restrictions can result in risks to security.

### SR-IOV restrictions

SR-IOV is only thoroughly tested with the following devices (other SR-IOV devices may work but have not been tested at the time of release):

- Intel® 82576NS Gigabit Ethernet Controller (**igb** driver)
- Intel® 82576EB Gigabit Ethernet Controller (**igb** driver)
- Intel® 82599ES 10 Gigabit Ethernet Controller (**ixgbe** driver)
- Intel® 82599EB 10 Gigabit Ethernet Controller (**ixgbe** driver)

### Core dumping restrictions

Because core dumping is currently implemented on top of migration, it is not supported when device assignment is in use.

## 4.2. Application Restrictions

There are aspects of virtualization which make it unsuitable for certain types of applications.

Applications with high I/O throughput requirements should use the paravirtualized drivers for fully-virtualized guests. Without the paravirtualized drivers certain applications may be unpredictable under heavy I/O loads.

The following applications should be avoided due to high I/O requirements:

- **kdump** server

- **netdump** server

You should carefully evaluate applications and tools that heavily utilize I/O or those that require real-time performance. Consider the paravirtualized drivers or PCI device assignment for increased I/O performance. Refer to Chapter 10, *KVM Paravirtualized (virtio) Drivers* for more information on the paravirtualized drivers for fully virtualized guests. Refer to Chapter 12, *PCI Device Assignment* for more information on PCI device assignment.

Applications suffer a small performance loss from running in virtualized environments. The performance benefits of virtualization through consolidating to newer and faster hardware should be evaluated against the potential application performance issues associated with using virtualization.

## 4.3. Other Restrictions

For the list of all other restrictions and issues affecting virtualization read the *Red Hat Enterprise Linux 6 Release Notes*. The *Red Hat Enterprise Linux 6 Release Notes* cover the present new features, known issues and restrictions as they are updated or discovered.

# Chapter 5. Installing the Virtualization Packages

Before you can use virtualization, the virtualization packages must be installed on your computer. Virtualization packages can be installed either during the host installation sequence or after host installation using Subscription Manager.

The KVM hypervisor uses the default Red Hat Enterprise Linux kernel with the *kvm* kernel module.

## 5.1. Configuring a Virtualization Host Installation

This section covers installing virtualization tools and virtualization packages as part of a fresh Red Hat Enterprise Linux installation.

**Note**

The *Red Hat Enterprise Linux Installation Guide*, available from https://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/, covers installing Red Hat Enterprise Linux in detail.

**Procedure 5.1. Installing the virtualization package group**

1. **Launch the Red Hat Enterprise Linux 6 installation program**

   Start an interactive Red Hat Enterprise Linux 6 installation from the Red Hat Enterprise Linux Installation CD-ROM, DVD or PXE.

2. **Continue installation up to package selection**

   Complete the other steps up to the package selection step.

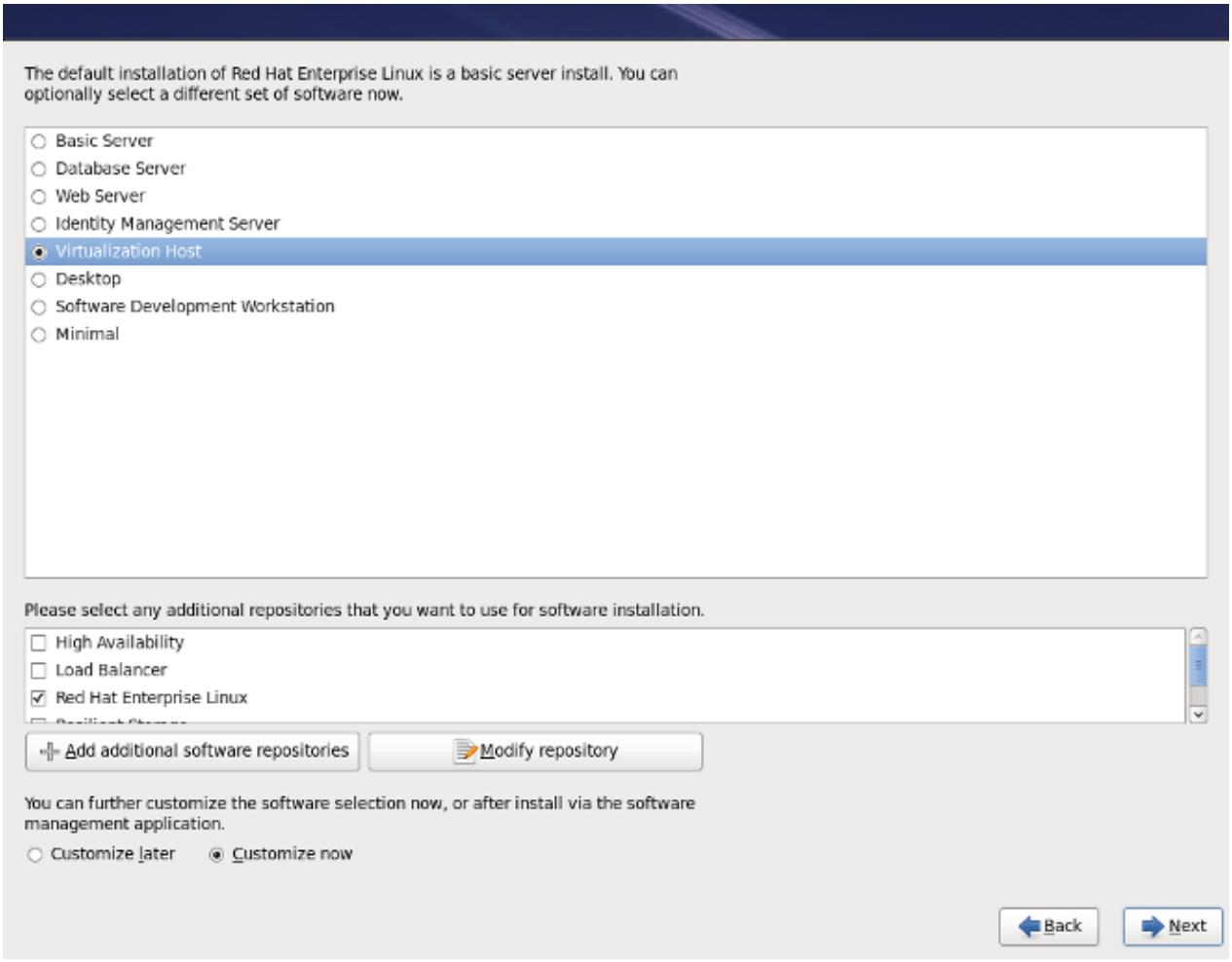



**Figure 5.1. The Red Hat Enterprise Linux package selection screen**

Select the **Virtualization Host** server role to install a platform for guest virtual machines. Alternatively, ensure that the **Customize Now** radio button is selected before proceeding, to specify individual packages.

3. **Select the Virtualization package group**

   This selects the qemu-kvm emulator, **virt-manager**, **libvirt** and **virt-viewer** for installation.

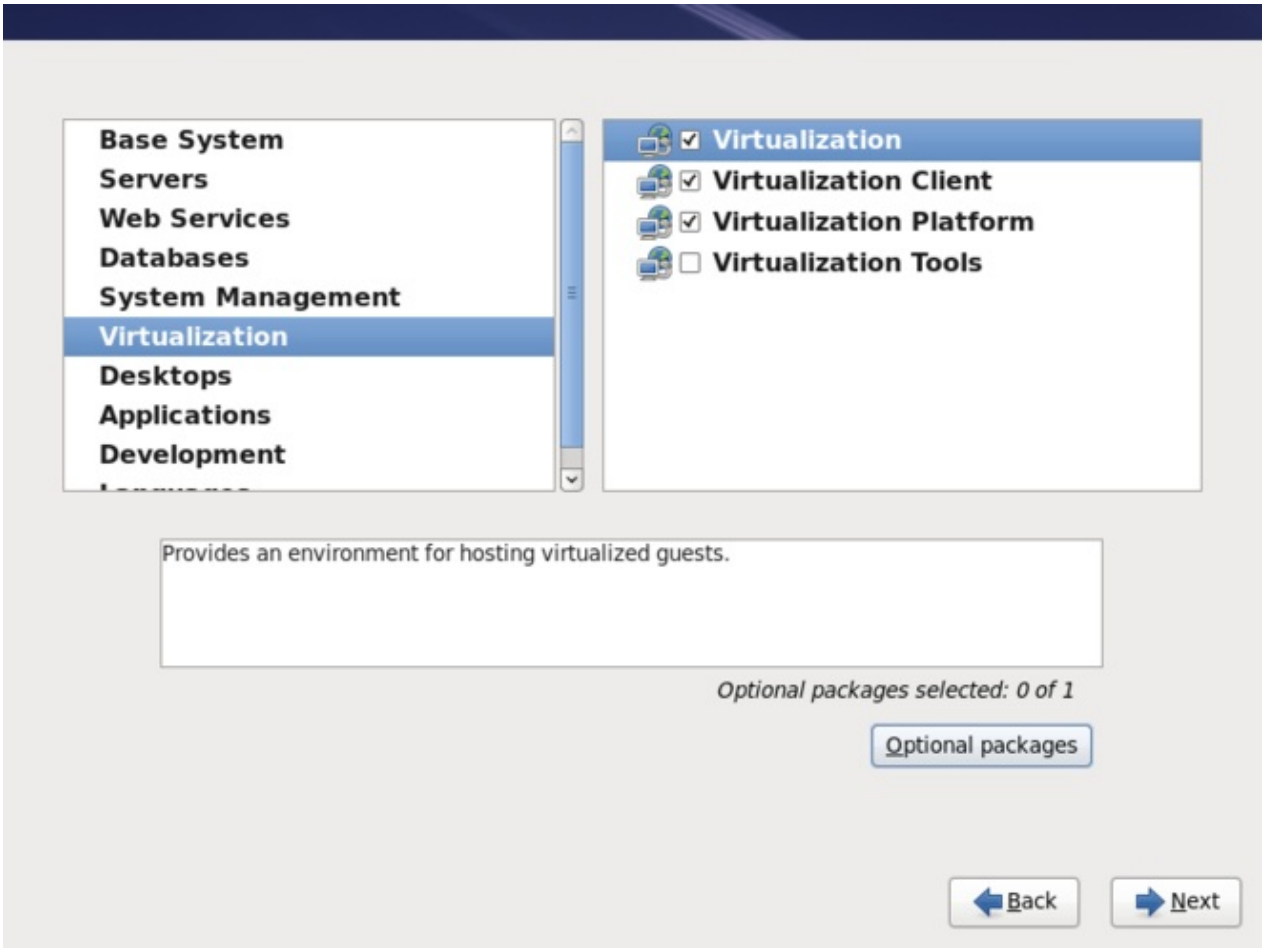



**Figure 5.2. The Red Hat Enterprise Linux package selection screen**

**Note**

If you wish to create virtual machines in a graphical user interface (**virt-manager**) later, you should also select the **General Purpose Desktop** package group.

4. **Customize the packages (if required)**

   Customize the **Virtualization** group if you require other virtualization packages.

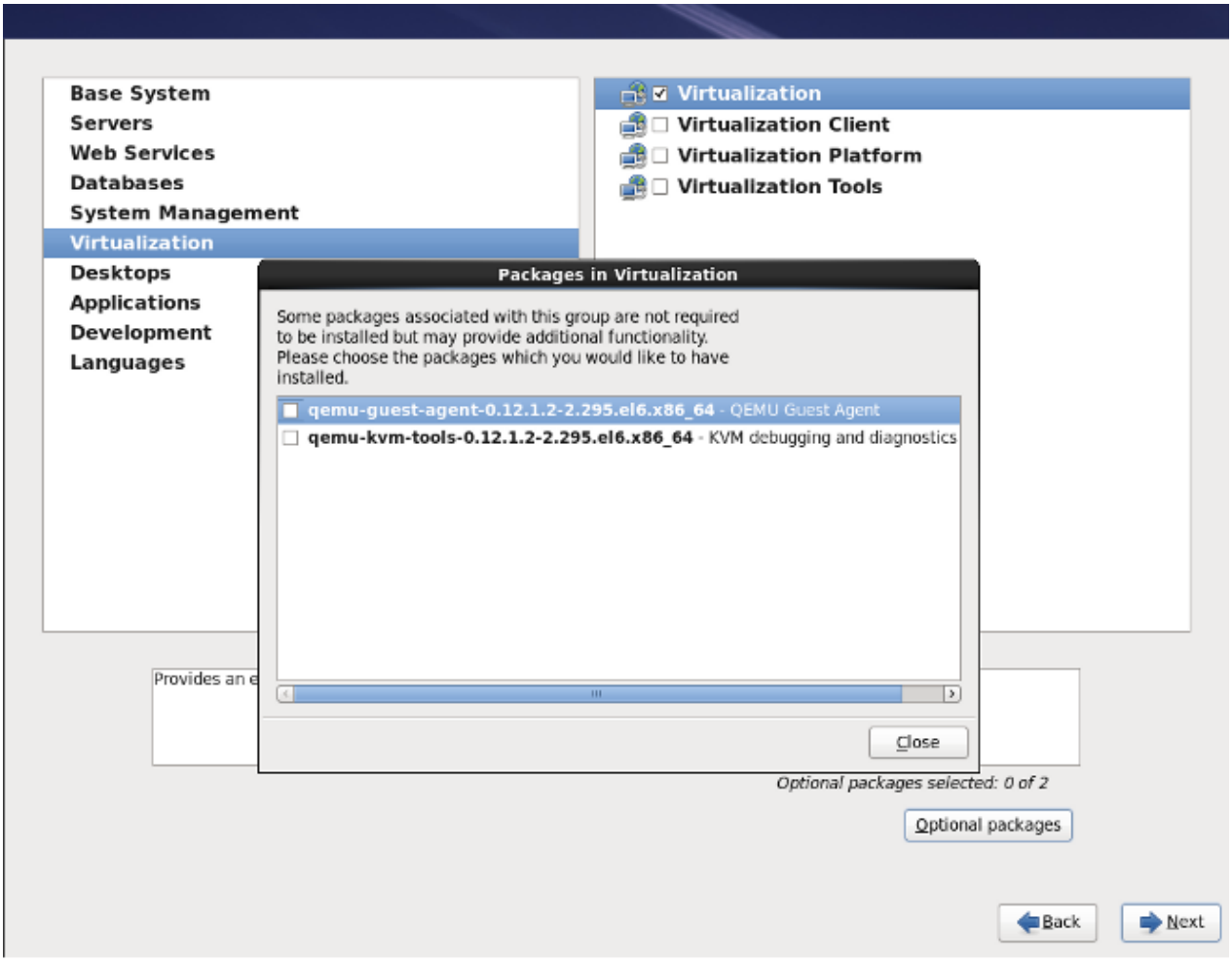



**Figure 5.3. The Red Hat Enterprise Linux package selection screen**

Click on the **Close** button, then the **Next** button to continue the installation.

When the installation is complete, reboot the system.

**Important**

You require a valid virtualization entitlement to receive updates for the virtualization packages.

### Installing KVM Packages with Kickstart Files

Kickstart files allow for large, automated installations without a user manually installing each individual host system. This section describes how to create and use a Kickstart file to install Red Hat Enterprise Linux with the Virtualization packages.

In the **%packages** section of your Kickstart file, append the following package groups:

```
@virtualization
@virtualization-client
@virtualization-platform
@virtualization-tools
```

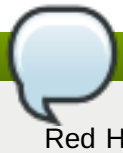
For more information about Kickstart files, refer to the *Red Hat Enterprise Linux Installation Guide*, available from https://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

## 5.2. Installing Virtualization Packages on an Existing Red Hat Enterprise Linux System

This section describes the steps for installing the KVM hypervisor on a working Red Hat Enterprise Linux 6 or newer system.

To install the packages, your machines must be registered. To register via Red Hat Subscription Manager, run the **subscription-manager register** command and follow the prompts.

If you do not have a valid Red Hat subscription, visit the Red Hat online store to obtain one.



**Note**

Red Hat Network (RHN) has now been deprecated. Subscription Manager should now be used for registration tasks.

### Installing the Virtualization Packages with yum

To use virtualization on Red Hat Enterprise Linux you require at least the **qemu-kvm** and **qemu-img** packages. These packages provide the user-level KVM emulator and disk image manager on the host Red Hat Enterprise Linux system.

To install the **qemu-kvm** and **qemu-img** packages, run the following command:

```
# yum install qemu-kvm qemu-img
```

Several additional virtualization management packages are also available:

**Recommended virtualization packages**

***python-virtinst***

Provides the **virt-install** command for creating virtual machines.

***libvirt***

The *libvirt* package provides the server and host side libraries for interacting with hypervisors and host systems. The *libvirt* package provides the **libvirtd** daemon that handles the library calls, manages virtual machines and controls the hypervisor.

***libvirt-python***

The *libvirt-python* package contains a module that permits applications written in the Python programming language to use the interface supplied by the *libvirt* API.

***virt-manager***

**virt-manager**, also known as **Virtual Machine Manager**, provides a graphical tool for administering virtual machines. It uses *libvirt-client* library as the management API.

***libvirt-client***

The *libvirt-client* package provides the client-side APIs and libraries for accessing *libvirt* servers. The *libvirt-client* package includes the **virsh** command line tool to manage and control virtual machines and hypervisors from the command line or a special virtualization shell.

Install all of these recommended virtualization packages with the following command:

```
# yum install virt-manager libvirt libvirt-python python-virtinst
libvirt-client
```

### Installing Virtualization Package Groups

The virtualization packages can also be installed from package groups. The following table describes the virtualization package groups and what they provide.

**Note**

Note that the **qemu-img** package is installed as a dependency of the **Virtualization** package group if it is not already installed on the system. It can also be installed manually with the **yum install qemu-img** command as described previously.

**Table 5.1. Virtualization package groups**

| Package Group | Description | Mandatory Packages | Optional Packages |
|---|---|---|---|
| Virtualization | Provides an environment for hosting virtual machines | qemu-kvm | qemu-guest-agent, qemu-kvm-tools |
| Virtualization Client | Clients for installing and managing virtualization instances | python-virtinst, virt-manager, virt-viewer | virt-top |
| Virtualization Platform | Provides an interface for accessing and controlling virtual machines and containers | libvirt, libvirt-client, virt-who, virt-what | fence-virtd-libvirt, fence-virtd-multicast, fence-virtd-serial, libvirt-cim, libvirt-java, libvirt-qmf, libvirt-snmp, perl-Sys-Virt |
| Virtualization Tools | Tools for offline virtual image management | libguestfs | libguestfs-java, libguestfs-tools, virt-v2v |

To install a package group, run the **yum groupinstall <groupname>** command. For instance, to install the **Virtualization Tools** package group, run the **yum groupinstall "Virtualization Tools"** command.

# Chapter 6. Guest Virtual Machine Installation Overview

After you have installed the virtualization packages on the host system you can create guest operating systems. This chapter describes the general processes for installing guest operating systems on virtual machines. You can create guest virtual machines using the **New** button in **virt-manager** or use the command line interface **virt-install**. Both methods are covered by this chapter.

Detailed installation instructions are available in the following chapters for specific versions of Red Hat Enterprise Linux and Microsoft Windows.

## 6.1. Guest Virtual Machine Prerequisites and Considerations

Various factors should be considered before creating any guest virtual machines. Not only should the role of a virtual machine be considered before deployment, but regular ongoing monitoring and assessment based on variable factors (load, amount of clients) should be performed. Some factors include:

**Performance**

Guest virtual machines should be deployed and configured based on their intended tasks. Some guest systems (for instance, guests running a database server) may require special performance considerations. Guests may require more assigned CPUs or memory based on their role and projected system load.

**Input/Output requirements and types of Input/Output**

Some guest virtual machines may have a particularly high I/O requirement or may require further considerations or projections based on the type of I/O (for instance, typical disk block size access, or the amount of clients).

**Storage**

Some guest virtual machines may require higher priority access to storage or faster disk types, or may require exclusive access to areas of storage. The amount of storage used by guests should also be regularly monitored and taken into account when deploying and maintaining storage.

**Networking and network infrastructure**

Depending upon your environment, some guest virtual machines could require faster network links than other guests. Bandwidth or latency are often factors when deploying and maintaining guests, especially as requirements or load changes.

**Request requirements**

SCSI requests can only be issued to guest virtual machines on virtio drives if the virtio drives are backed by whole disks, and the disk device parameter is set to **lun**, as shown in the following example:

```
<devices>
   <emulator>/usr/libexec/qemu-kvm</emulator>
   <disk type='block' device='lun'>
```

## 6.2. Creating Guests with virt-install

You can use the **virt-install** command to create guest virtual machines from the command line. **virt-install** is used either interactively or as part of a script to automate the creation of virtual machines. Using **virt-install** with Kickstart files allows for unattended installation of virtual machines.

The **virt-install** tool provides a number of options that can be passed on the command line. To see a complete list of options run the following command:

```
# virt-install --help
```

Note that you need root privileges in order for **virt-install** commands to complete successfully. The **virt-install** man page also documents each command option and important variables.

**qemu-img** is a related command which may be used before **virt-install** to configure storage options.

An important option is the ***--graphics*** option which allows graphical installation of a virtual machine.

**Example 6.1. Using virt-install to install a Red Hat Enterprise Linux 5 guest virtual machine**

This example creates a Red Hat Enterprise Linux 5 guest:

```
virt-install \
   --name=guest1-rhel5-64 \
   --file=/var/lib/libvirt/images/guest1-rhel5-64.dsk \
   --file-size=8 \
   --nonsparse --graphics spice \
   --vcpus=2 --ram=2048 \
   --location=http://example1.com/installation_tree/RHEL5.6-Server-
x86_64/os \
   --network bridge=br0 \
   --os-type=linux \
   --os-variant=rhel5.4
```

Ensure that you select the correct ***os-type*** for your operating system when running this command.

Refer to **man virt-install** for more examples.

**Note**

When installing a Windows guest with **virt-install**, the **--os-type=*windows*** option is recommended. This option prevents the CD-ROM from disconnecting when rebooting during the installation procedure. The **--os-variant** option further optimizes the configuration for a specific guest operating system.

## 6.3. Creating Guests with virt-manager

**virt-manager**, also known as Virtual Machine Manager, is a graphical tool for creating and managing guest virtual machines.

**Procedure 6.1. Creating a guest virtual machine with virt-manager**

1. **Open virt-manager**

   Start **virt-manager**. Launch the **Virtual Machine Manager** application from the **Applications** menu and **System Tools** submenu. Alternatively, run the **virt-manager** command as root.

2. **Optional: Open a remote hypervisor**

   Select the hypervisor and click the **Connect** button to connect to the remote hypervisor.

3. **Create a new virtual machine**

   The **virt-manager** window allows you to create a new virtual machine. Click the **Create a new virtual machine** button (Figure 6.1, “Virtual Machine Manager window”) to open the **New VM** wizard.

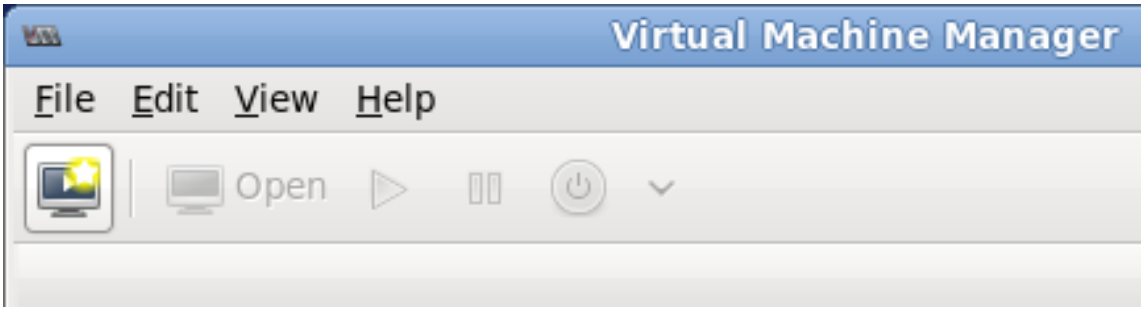
   
   

   **Figure 6.1. Virtual Machine Manager window**

   The **New VM** wizard breaks down the virtual machine creation process into five steps:

   a. Naming the guest virtual machine and choosing the installation type

   b. Locating and configuring the installation media

   c. Configuring memory and CPU options

   d. Configuring the virtual machine's storage

   e. Configuring networking, architecture, and other hardware settings

   Ensure that **virt-manager** can access the installation media (whether locally or over the network) before you continue.

4. **Specify name and installation type**

   The guest virtual machine creation process starts with the selection of a name and installation type. Virtual machine names can have underscores (_), periods (**.**), and hyphens (**-**).


**Figure 6.2. Name virtual machine and select installation method**

Type in a virtual machine name and choose an installation type:

**Local install media (ISO image or CDROM)**

This method uses a CD-ROM, DVD, or image of an installation disk (for example, `.iso`).

**Network Install (HTTP, FTP, or NFS)**

This method involves the use of a mirrored Red Hat Enterprise Linux or Fedora installation tree to install a guest. The installation tree must be accessible through either HTTP, FTP, or NFS.

**Network Boot (PXE)**

This method uses a Preboot eXecution Environment (PXE) server to install the guest virtual machine. Setting up a PXE server is covered in the *Deployment Guide*. To install via network boot, the guest must have a routable IP address or shared network device. For information on the required networking configuration for PXE installation, refer to Section 6.4, “Creating Guests with PXE”.

**Import existing disk image**

This method allows you to create a new guest virtual machine and import a disk image (containing a pre-installed, bootable operating system) to it.

Click **Forward** to continue.

5. **Configure installation**

Next, configure the **OS type** and **Version** of the installation. Ensure that you select the appropriate OS type for your virtual machine. Depending on the method of installation, provide the install URL or existing storage path.

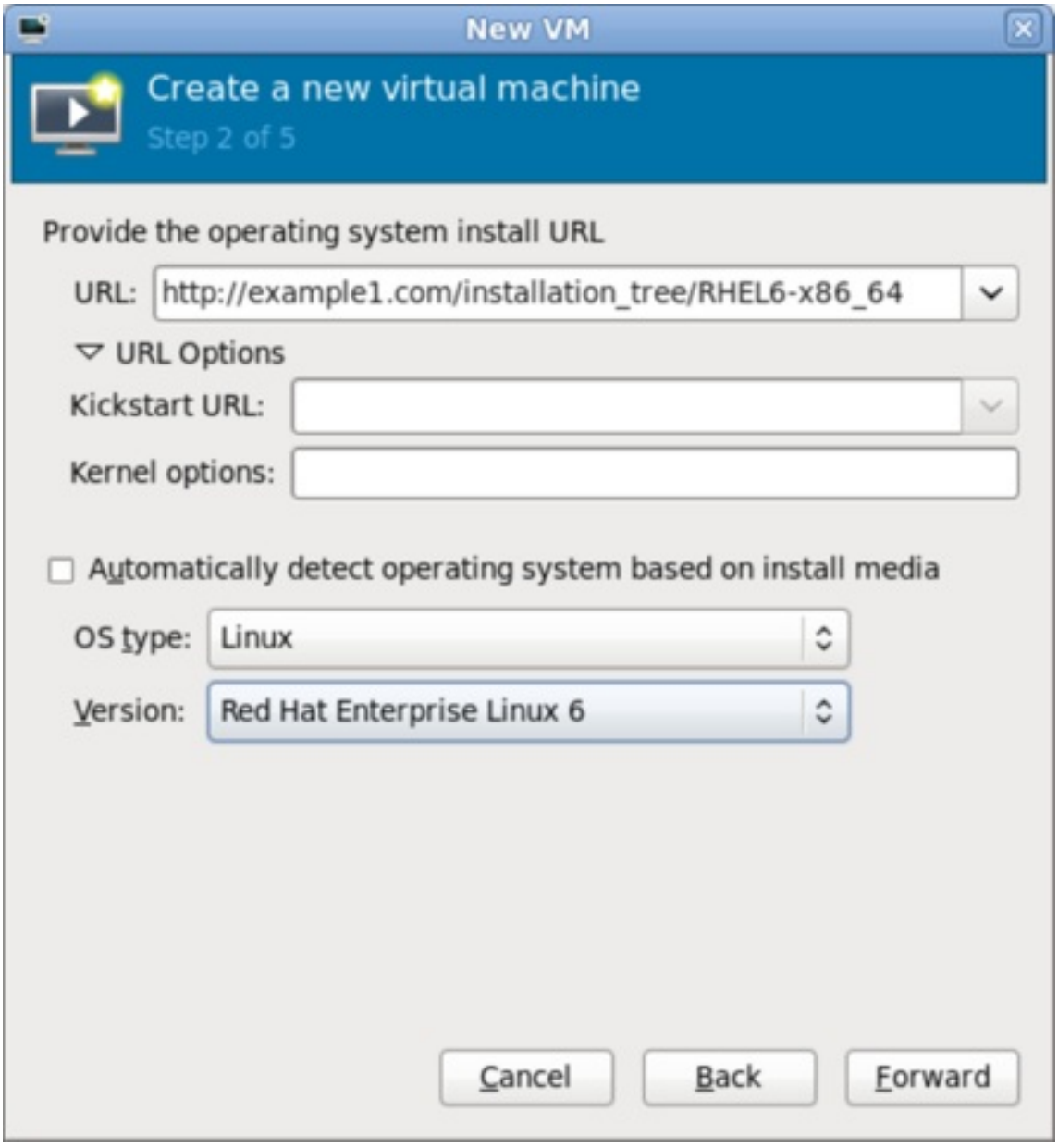



**Figure 6.3. Remote installation URL**

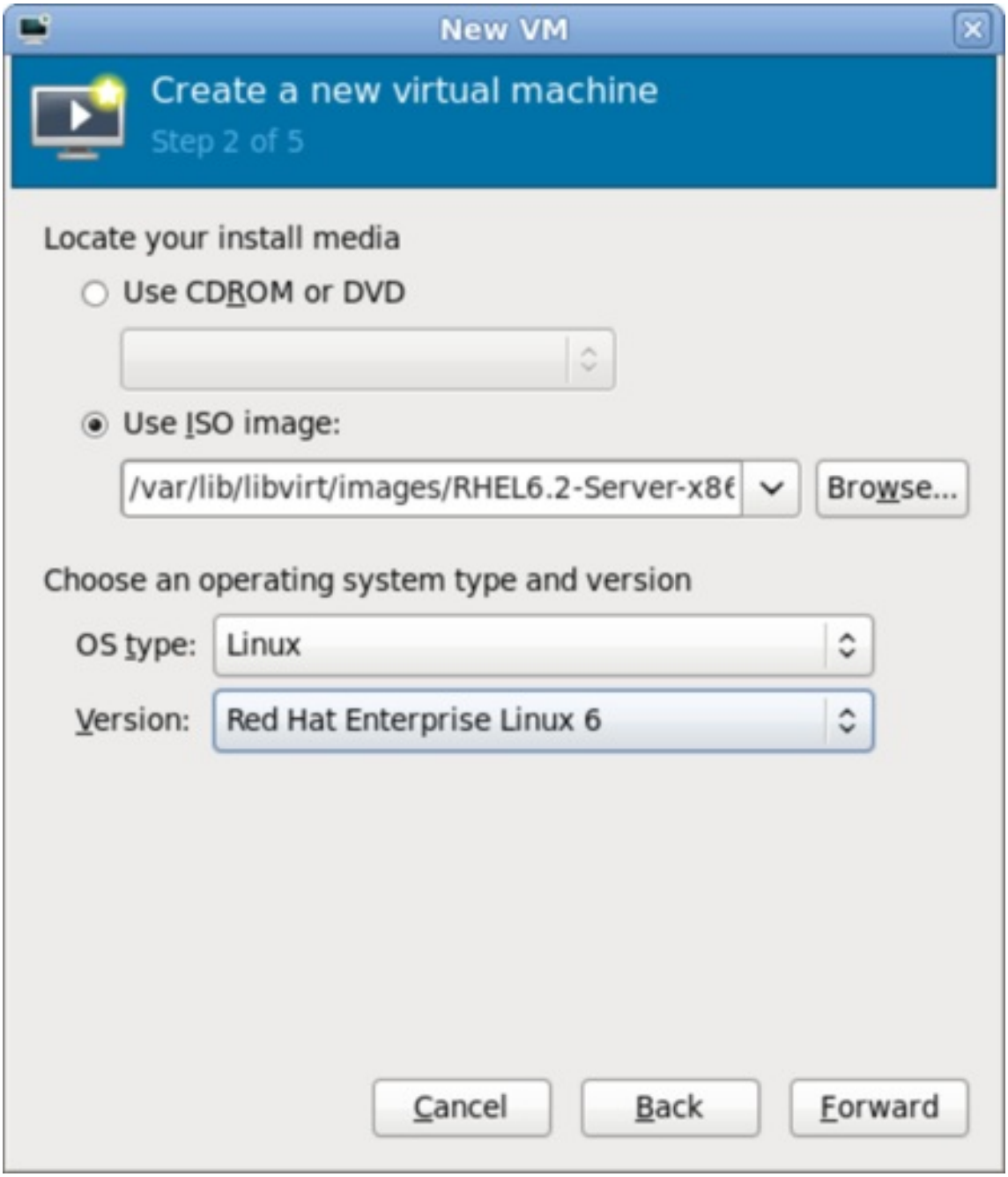



**Figure 6.4. Local ISO image installation**

6. **Configure CPU and memory**

   The next step involves configuring the number of CPUs and amount of memory to allocate to the virtual machine. The wizard shows the number of CPUs and amount of memory you can allocate; configure these settings and click **Forward**.

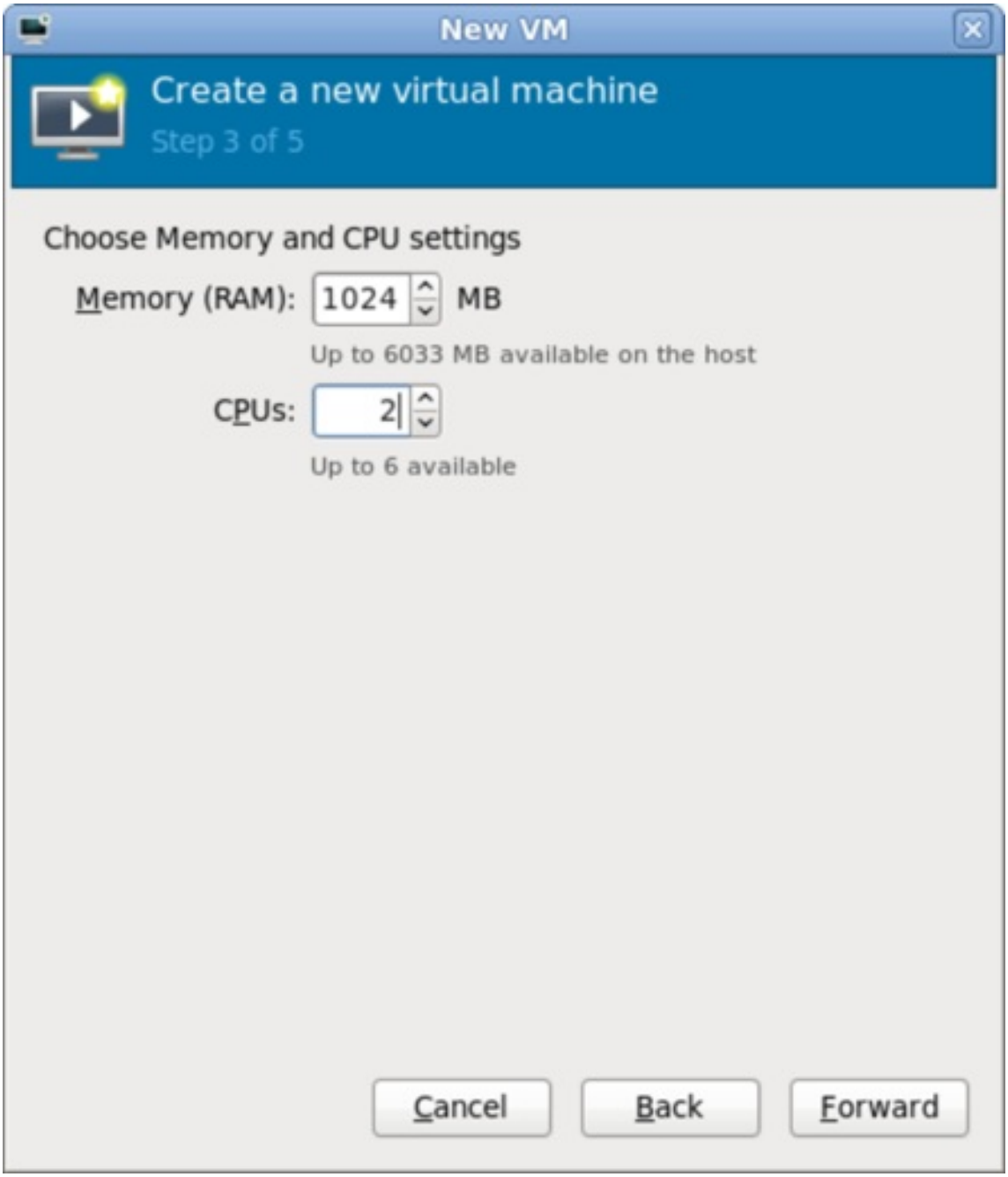



**Figure 6.5. Configuring CPU and memory**

7. **Configure storage**

   Assign storage to the guest virtual machine.

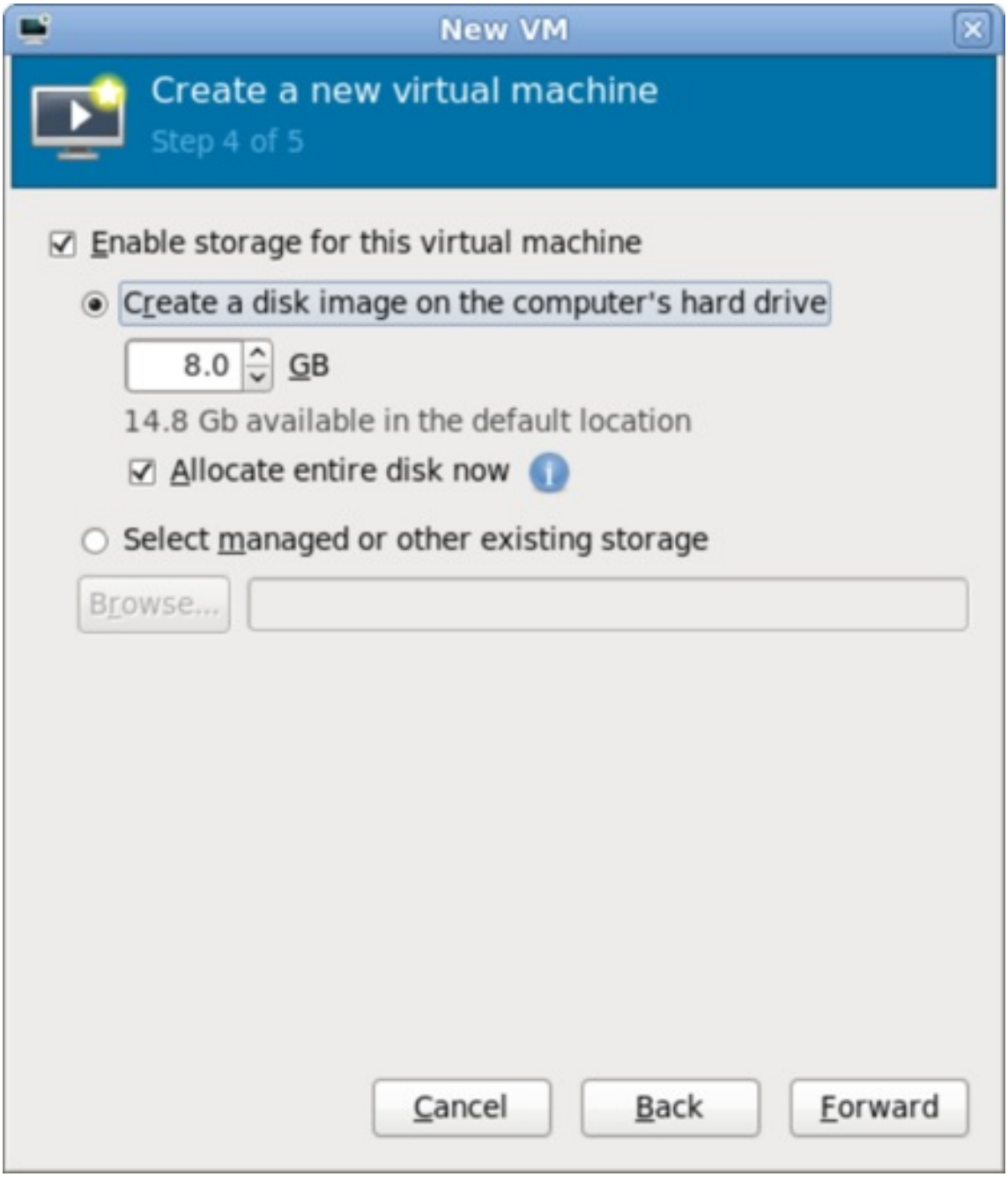



**Figure 6.6. Configuring virtual storage**

If you chose to import an existing disk image during the first step, **virt-manager** will skip this step.

Assign sufficient space for your virtual machine and any applications it requires, then click **Forward** to continue.

8. **Final configuration**

   Verify the settings of the virtual machine and click **Finish** when you are satisfied; doing so will create the virtual machine with default networking settings, virtualization type, and architecture.

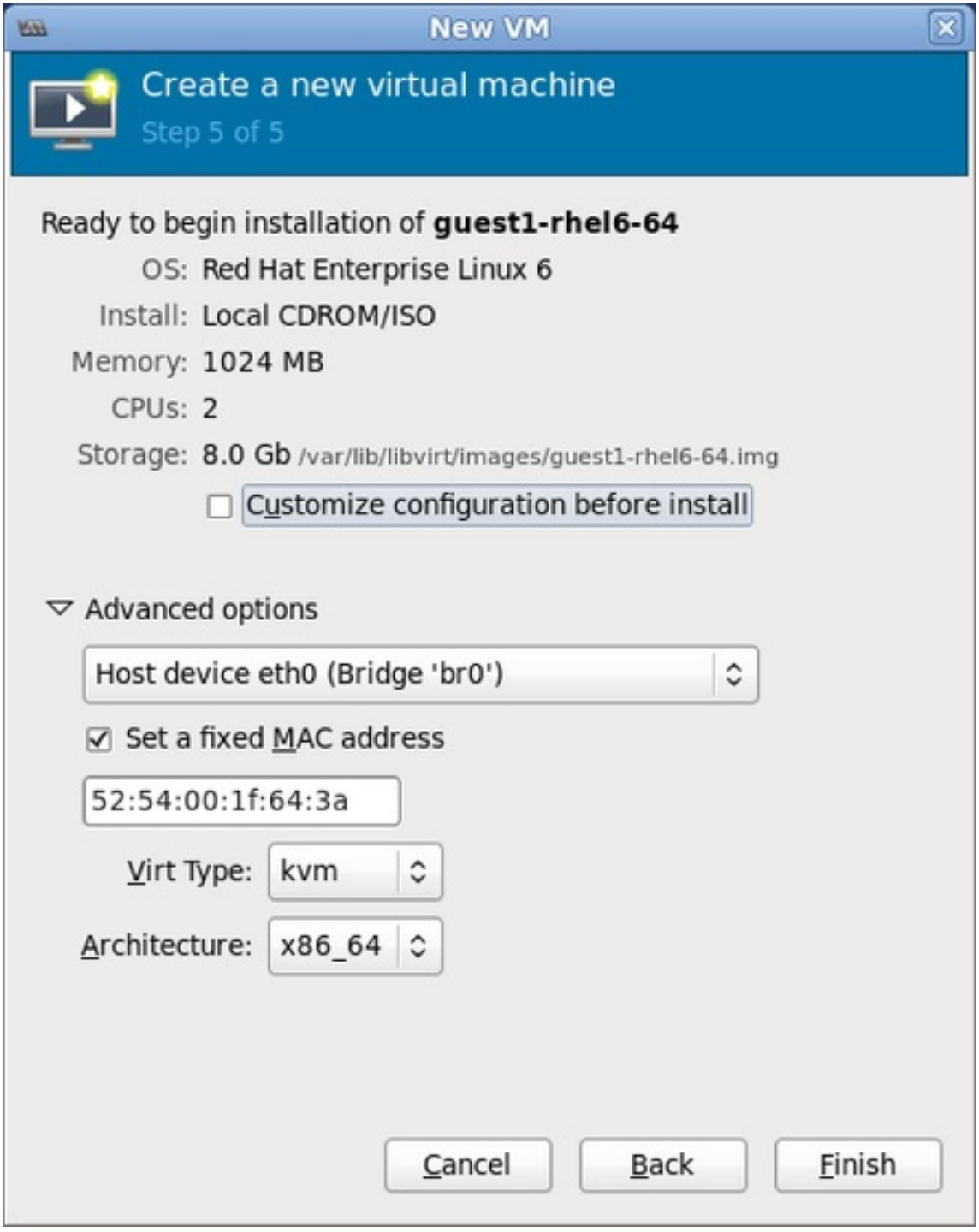



**Figure 6.7. Verifying the configuration**

If you prefer to further configure the virtual machine's hardware first, check the **Customize configuration before install** box first before clicking **Finish**. Doing so will open another wizard that will allow you to add, remove, and configure the virtual machine's hardware settings.

After configuring the virtual machine's hardware, click **Apply**. **virt-manager** will then create the virtual machine with your specified hardware settings.

## 6.4. Creating Guests with PXE

This section provides information on creating guests with PXE.

### Requirements

PXE guest installation requires a PXE server running on the same subnet as the guest virtual machines you wish to install. The method of accomplishing this depends on how the virtual machines are connected to the network. Contact Support if you require assistance setting up a PXE server.

### PXE Installation with virt-install

**`virt-install`** PXE installations require both the **`--network=bridge:installation`** parameter, where ***`installation`*** is the name of your bridge, and the ***`--pxe`*** parameter.

By default, if no network is found, the guest virtual machine attempts to boot from alternative bootable devices. If there is no other bootable device found, the guest pauses. You can use the **`qemu-kvm`** boot parameter ***`reboot-timeout`*** to allow the guest to retry booting if no bootable device is found, like so:

```
# qemu-kvm -boot reboot-timeout=1000
```

**Example 6.2. Fully-virtualized PXE installation with virt-install**

```
# virt-install --hvm --connect qemu:///system \
--network=bridge:installation --pxe --graphics spice \
--name rhel6-machine --ram=756 --vcpus=4 \
--os-type=linux --os-variant=rhel6 \
--disk path=/var/lib/libvirt/images/rhel6-machine.img,size=10
```

Note that the command above cannot be executed in a text-only environment. A fully-virtualized (***`--hvm`***) guest can only be installed in a text-only environment if the ***`--location`*** and ***`--extra-args "console=console_type"`*** are provided instead of the ***`--graphics spice`*** parameter.

**Procedure 6.2. PXE installation with virt-manager**

1. **Select PXE**

   Select PXE as the installation method and follow the rest of the steps to configure the OS type, memory, CPU and storage settings.

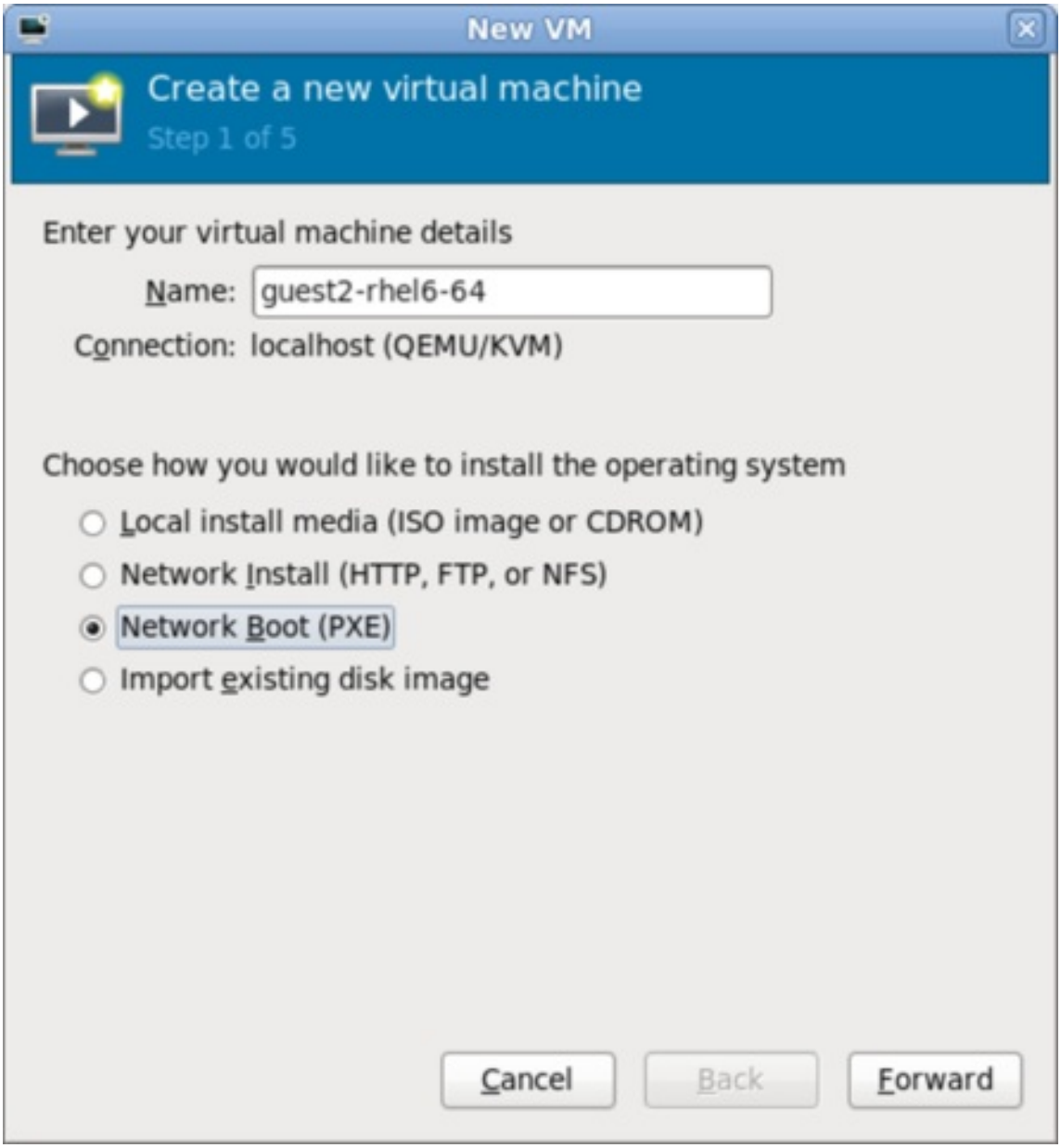



**Figure 6.8. Selecting the installation method**

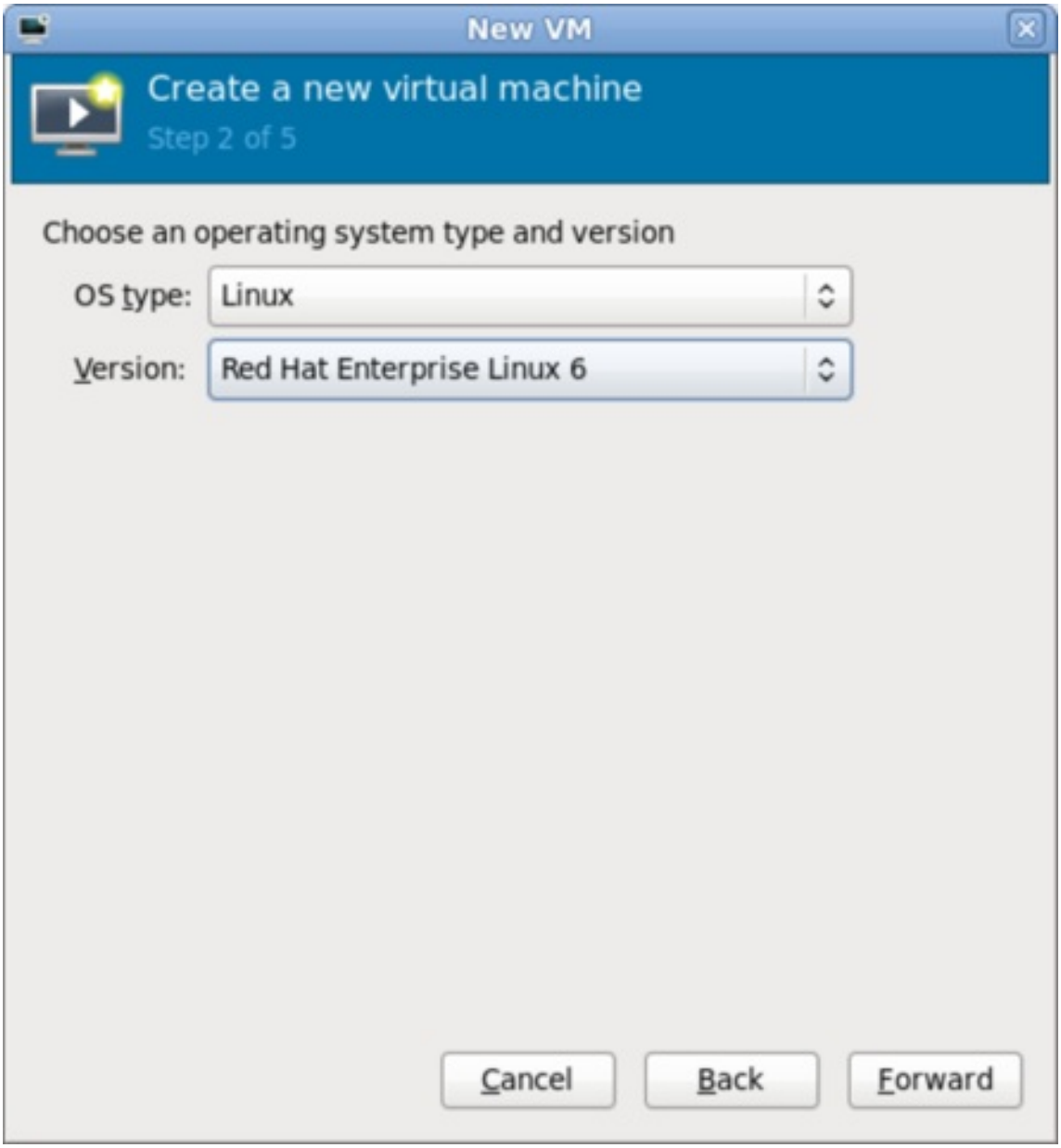



**Figure 6.9. Selecting the installation type**


**Figure 6.10. Specifying virtualized hardware details**

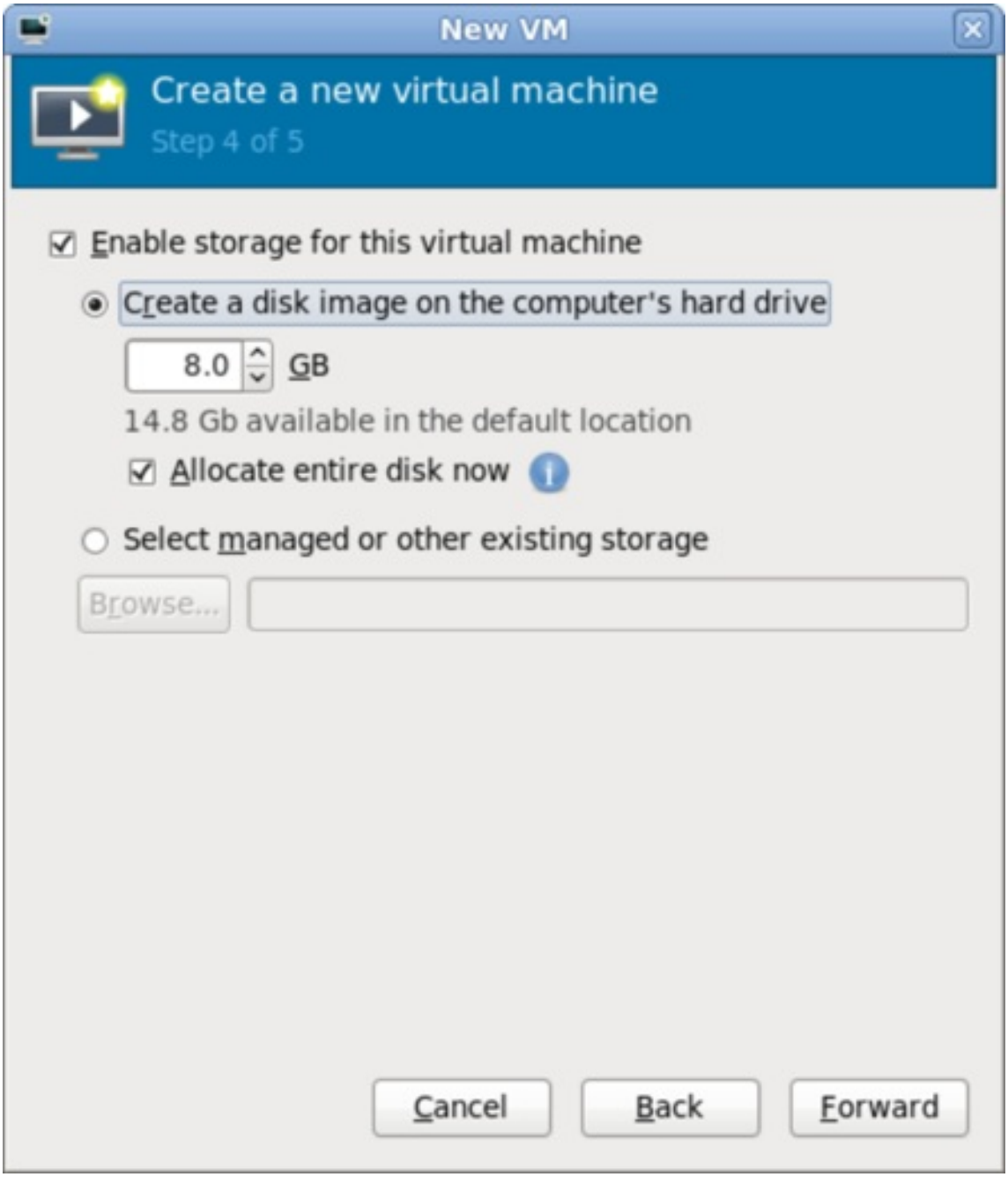



**Figure 6.11. Specifying storage details**

2. **Start the installation**

   The installation is ready to start.

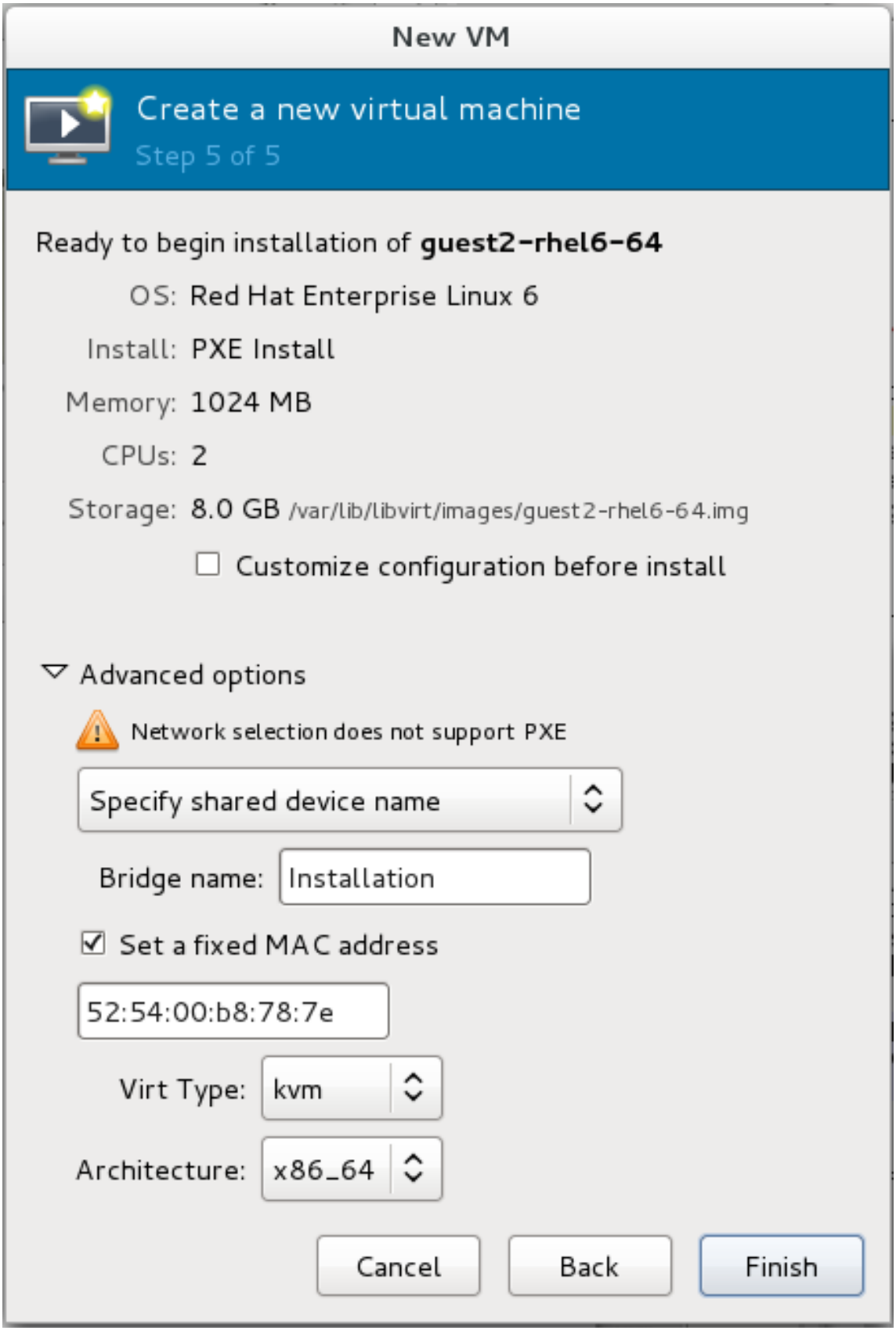



**Figure 6.12. Finalizing virtual machine details**

A DHCP request is sent and if a valid PXE server is found the guest virtual machine's installation processes will start.

# Chapter 7. Installing a Red Hat Enterprise Linux 6 Guest Virtual Machine on a Red Hat Enterprise Linux 6 Host

This chapter covers how to install a Red Hat Enterprise Linux 6 guest virtual machine on a Red Hat Enterprise Linux 6 host.

These procedures assume that the KVM hypervisor and all other required packages are installed and the host is configured for virtualization.

**Note**

For more information on installing the virtualization packages, refer to Chapter 5, *Installing the Virtualization Packages*.

## 7.1. Creating a Red Hat Enterprise Linux 6 Guest with Local Installation Media

This procedure covers creating a Red Hat Enterprise Linux 6 guest virtual machine with a locally stored installation DVD or DVD image. DVD images are available from http://access.redhat.com for Red Hat Enterprise Linux 6.

**Procedure 7.1. Creating a Red Hat Enterprise Linux 6 guest virtual machine with virt-manager**

1. **Optional: Preparation**

   Prepare the storage environment for the virtual machine. For more information on preparing storage, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

   **Important**

   Various storage types may be used for storing guest virtual machines. However, for a virtual machine to be able to use migration features the virtual machine must be created on networked storage.

   Red Hat Enterprise Linux 6 requires at least 1GB of storage space. However, Red Hat recommends at least 5GB of storage space for a Red Hat Enterprise Linux 6 installation and for the procedures in this guide.

2. **Open virt-manager and start the wizard**

   Open virt-manager by executing the **`virt-manager`** command as root or opening **`Applications → System Tools → Virtual Machine Manager`**.

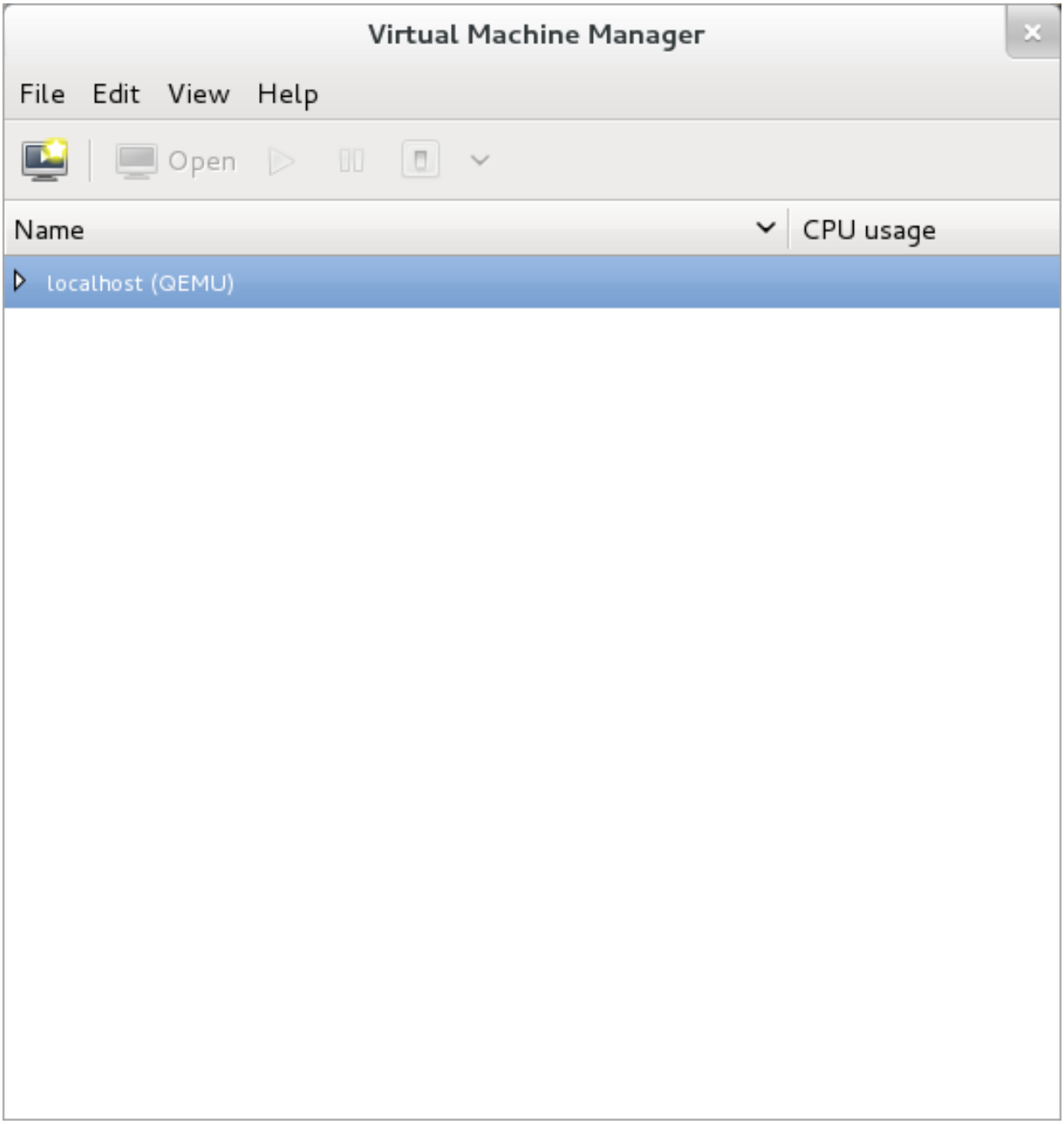



**Figure 7.1. The Virtual Machine Manager window**

Click on the **Create a new virtual machine** button to start the new virtualized guest wizard.

**Figure 7.2. The Create a new virtual machine button**

The **New VM** window opens.

3. **Name the virtual machine**

   Virtual machine names can contain letters, numbers and the following characters: '_', '**.**' and '**-**'. Virtual machine names must be unique for migration and cannot consist only of numbers.

Choose the **Local install media (ISO image or CDROM)** radio button.

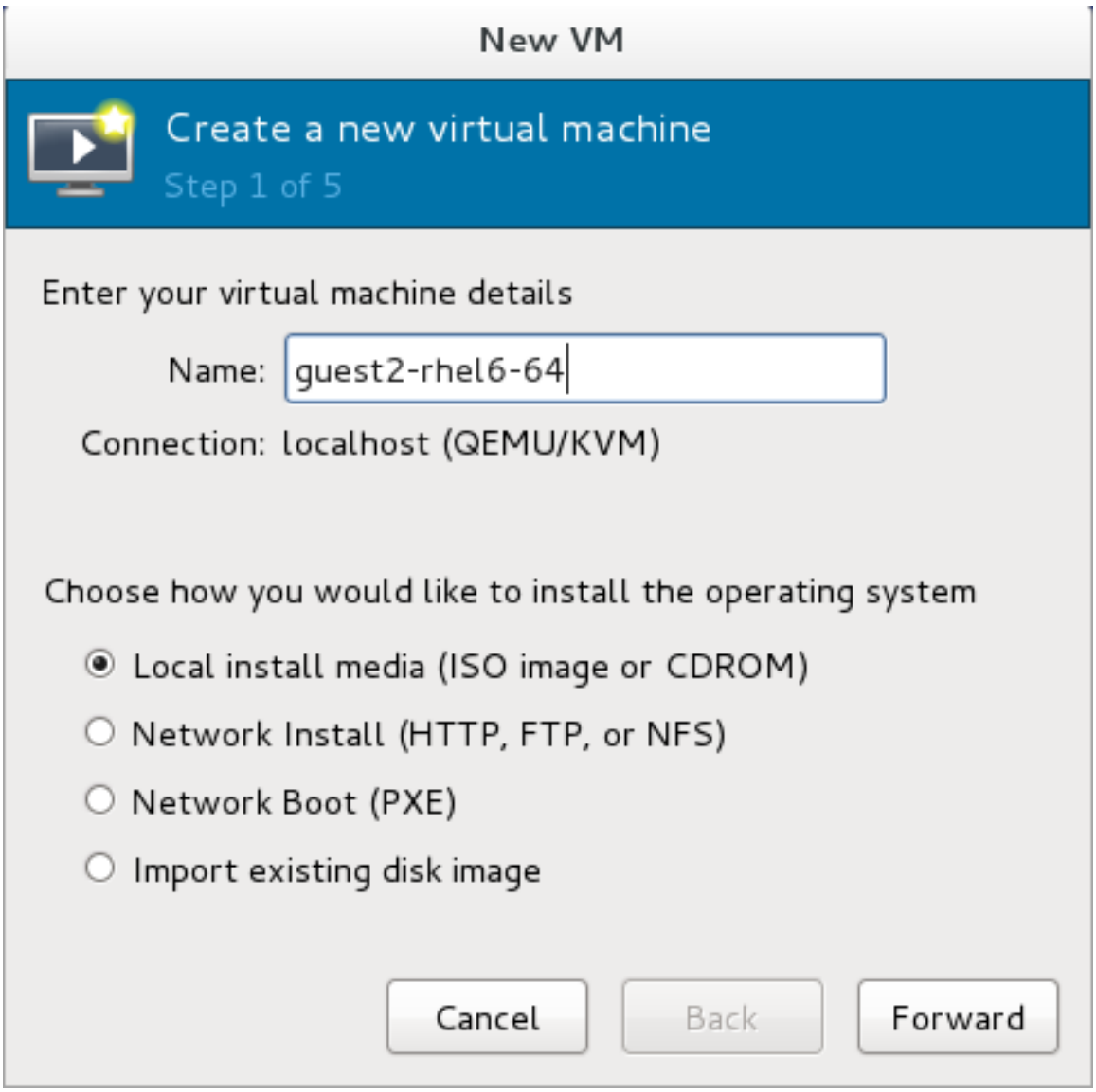



**Figure 7.3. The New VM window - Step 1**

Click **Forward** to continue.

4. **Select the installation media**

   Select the appropriate radio button for your installation media.

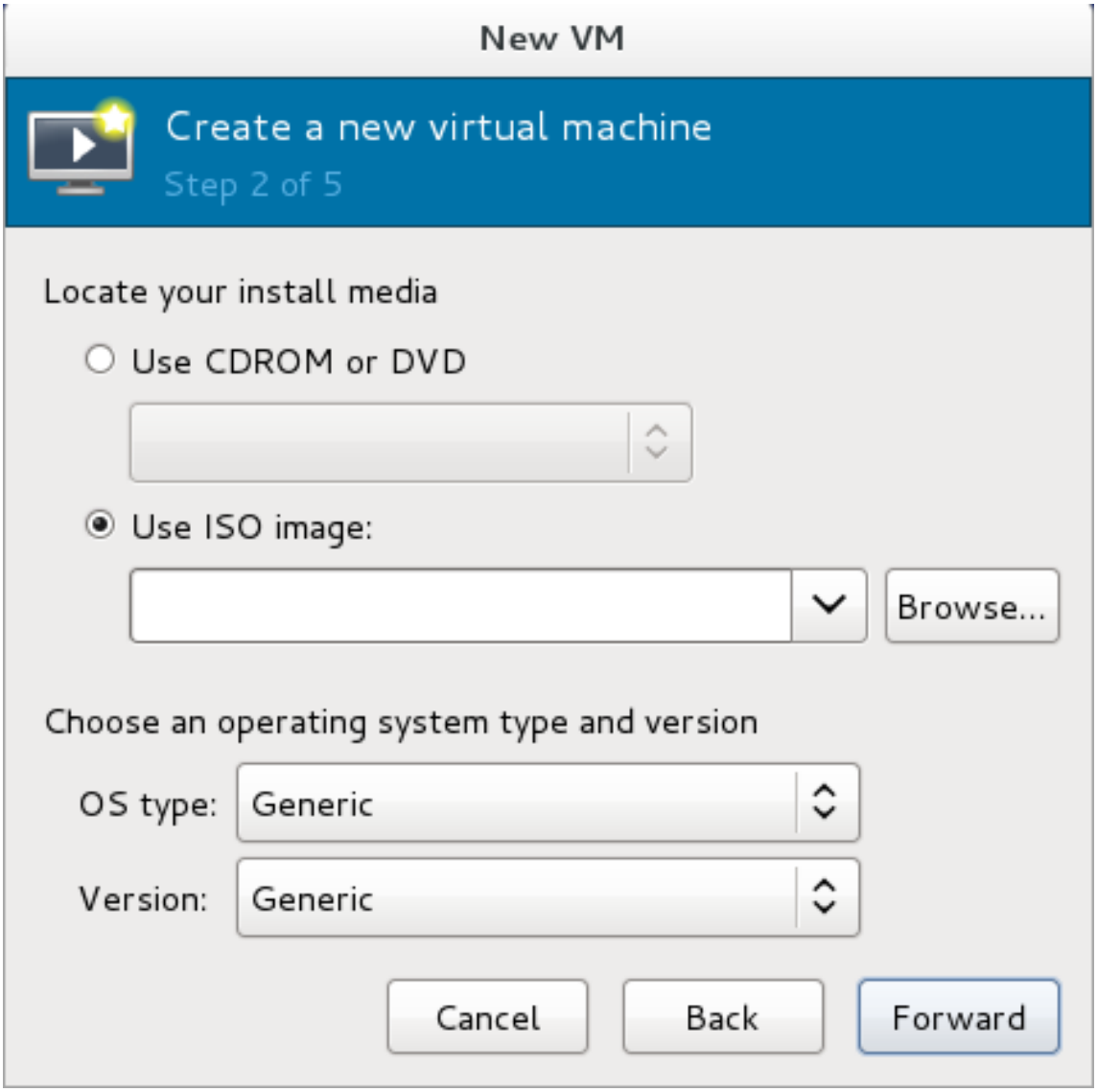



**Figure 7.4. Locate your install media**

A. If you wish to install from a CD-ROM or DVD, select the **Use CDROM or DVD** radio button, and select the appropriate disk drive from the drop-down list of drives available.

B. If you wish to install from an ISO image, select **Use ISO image**, and then click the **Browse...** button to open the **Locate media volume** window.

   Select the installation image you wish to use, and click **Choose Volume**.

   If no images are displayed in the **Locate media volume** window, click on the **Browse Local** button to browse the host machine for the installation image or DVD drive containing the installation disk. Select the installation image or DVD drive containing the installation disk and click **Open**; the volume is selected for use and you are returned to the **Create a new virtual machine** wizard.

**Important**

For ISO image files and guest storage images, the recommended location to use is **`/var/lib/libvirt/images/`**. Any other location may require additional configuration by SELinux. Refer to the Red Hat Enterprise Linux 6 *Virtualization Administration Guide* for more details on configuring SELinux.

Select the operating system type and version which match the installation media you have selected.

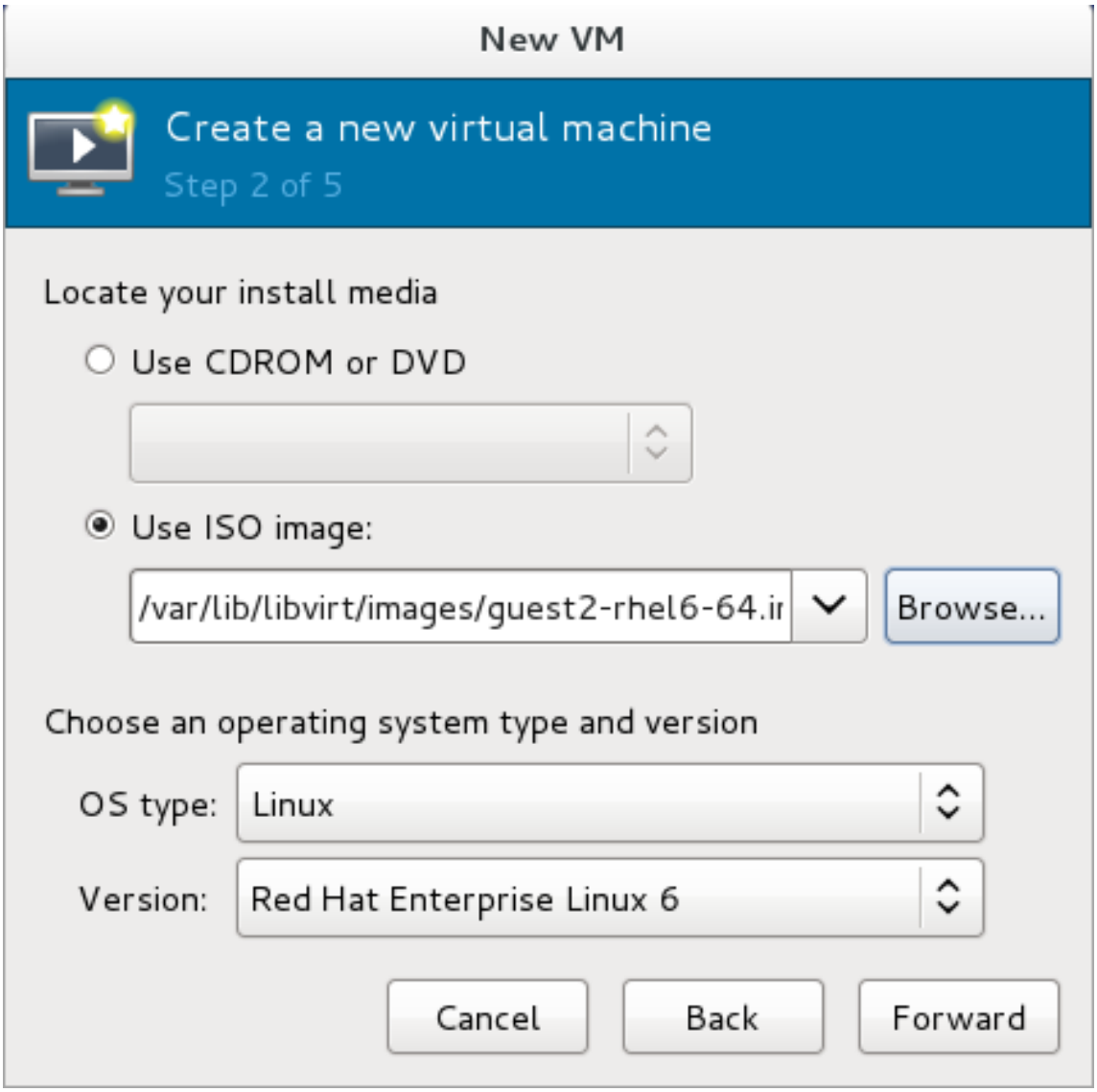



**Figure 7.5. The New VM window - Step 2**

Click **Forward** to continue.

5. **Set RAM and virtual CPUs**

   Choose appropriate values for the virtual CPUs and RAM allocation. These values affect the host's and guest's performance. Memory and virtual CPUs can be overcommitted. For more information on overcommitting, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

Virtual machines require sufficient physical memory (RAM) to run efficiently and effectively. Red Hat supports a minimum of 512MB of RAM for a virtual machine. Red Hat recommends at least 1024MB of RAM for each logical core.

Assign sufficient virtual CPUs for the virtual machine. If the virtual machine runs a multithreaded application, assign the number of virtual CPUs the guest virtual machine will require to run efficiently.

You cannot assign more virtual CPUs than there are physical processors (or hyper-threads) available on the host system. The number of virtual CPUs available is noted in the **`Up to X available`** field.

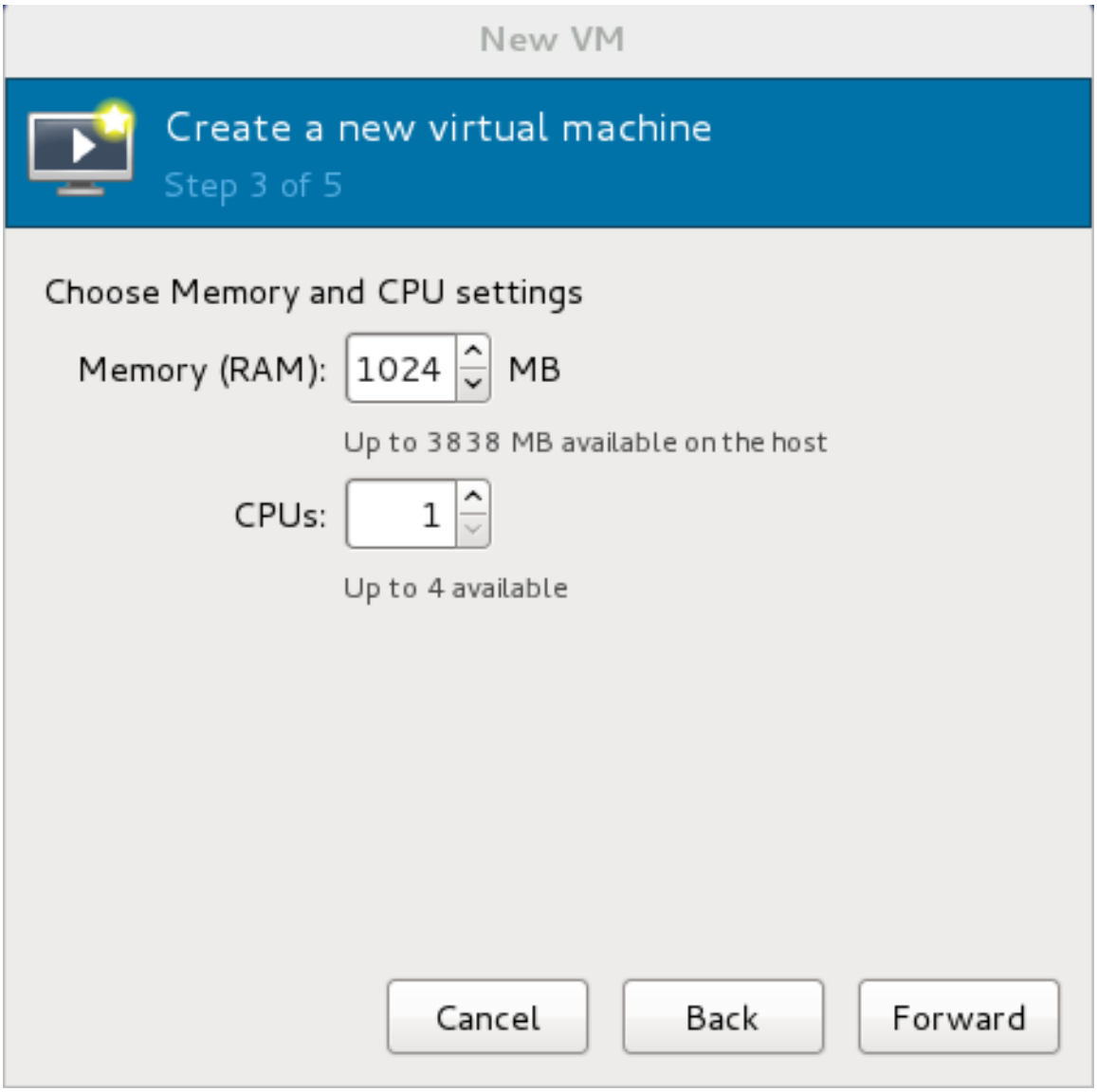



**Figure 7.6. The new VM window - Step 3**

Click **Forward** to continue.

6. **Enable and assign storage**

   Enable and assign storage for the Red Hat Enterprise Linux 6 guest virtual machine. Assign at least 5GB for a desktop installation or at least 1GB for a minimal installation.

**Note**

Live and offline migrations require virtual machines to be installed on shared network storage. For information on setting up shared storage for virtual machines, refer to the *Red Hat Enterprise Linux Virtualization Administration Guide*.

a. **With the default local storage**

Select the **Create a disk image on the computer's hard drive** radio button to create a file-based image in the default storage pool, the **/var/lib/libvirt/images/** directory. Enter the size of the disk image to be created. If the **Allocate entire disk now** check box is selected, a disk image of the size specified will be created immediately. If not, the disk image will grow as it becomes filled.

**Note**

Although the storage pool is a virtual container it is limited by two factors: maximum size allowed to it by qemu-kvm and the size of the disk on the host physical machine. Storage pools may not exceed the size of the disk on the host physical machine. The maximum sizes are as follows:

- virtio-blk = 2^63 bytes or 8 Exabytes(using raw files or disk)
- Ext4 = ~ 16 TB (using 4 KB block size)
- XFS = ~8 Exabytes
- qcow2 and host file systems keep their own metadata and scalability should be evaluated/tuned when trying very large image sizes. Using raw disks means fewer layers that could affect scalability or max size.

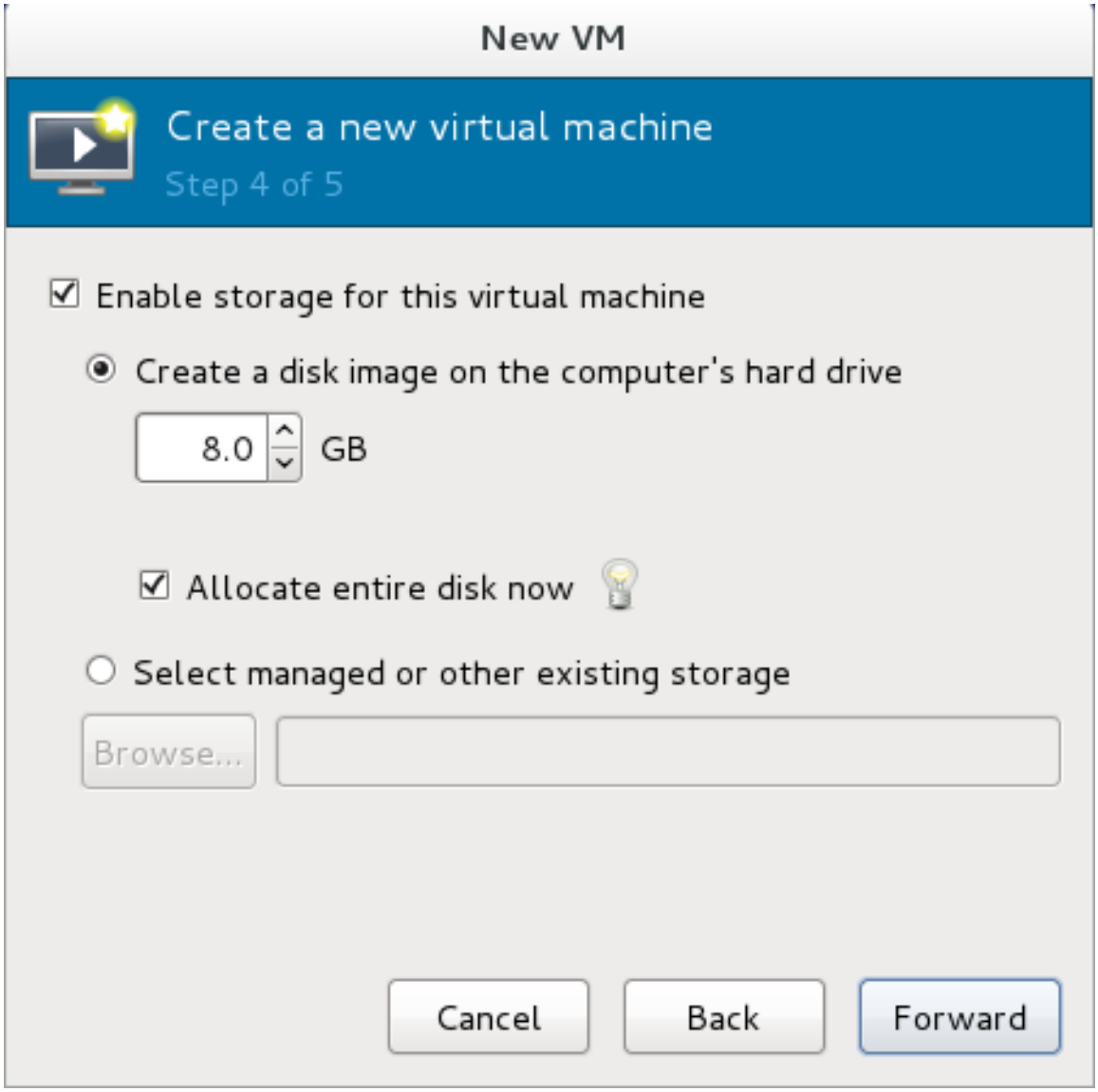



**Figure 7.7. The New VM window - Step 4**

Click **Forward** to create a disk image on the local hard drive. Alternatively, select **Select managed or other existing storage**, then select **Browse** to configure managed storage.

b. **With a storage pool**

If you selected **Select managed or other existing storage** in the previous step to use a storage pool and clicked **Browse**, the **Locate or create storage volume** window will appear.

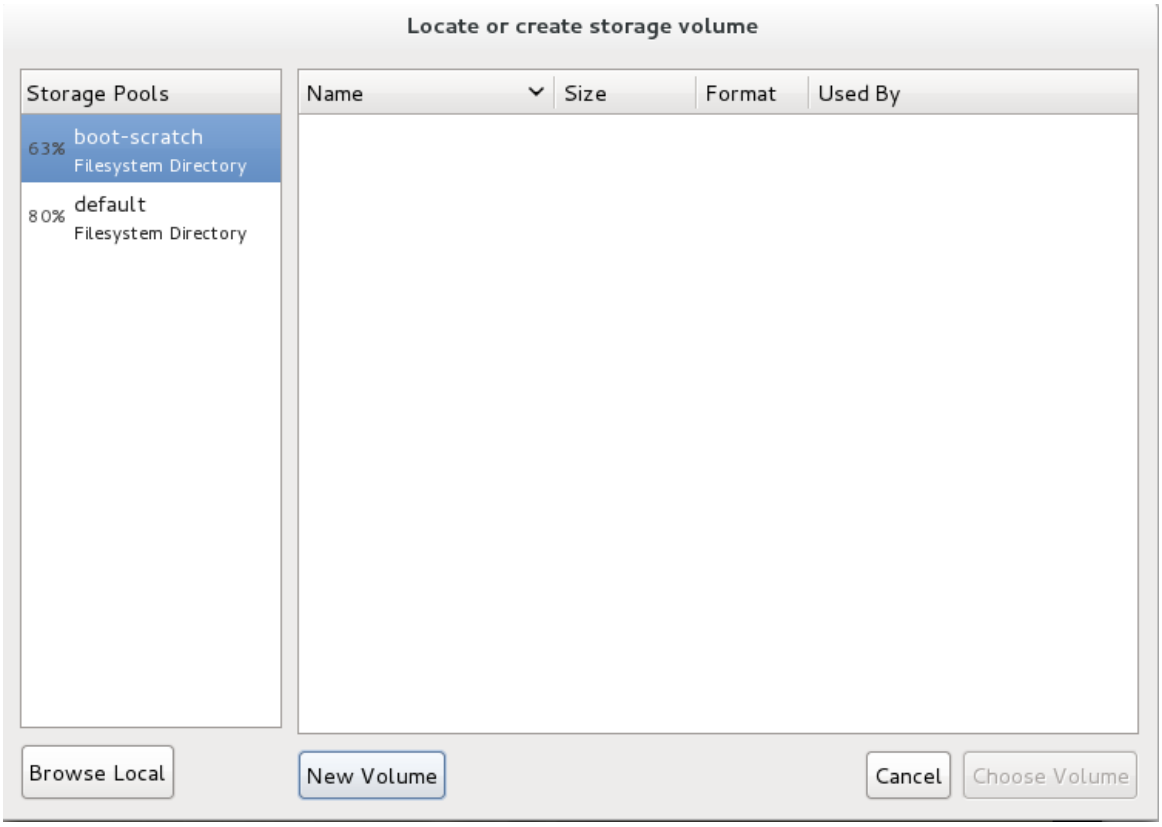



**Figure 7.8. The Locate or create storage volume window**

i. Select a storage pool from the **Storage Pools** list.

ii. Optional: Click on the **New Volume** button to create a new storage volume. The **Add a Storage Volume** screen will appear. Enter the name of the new storage volume.

Choose a format option from the **Format** dropdown menu. Format options include raw, cow, qcow, qcow2, qed, vmdk, and vpc. Adjust other fields as desired.

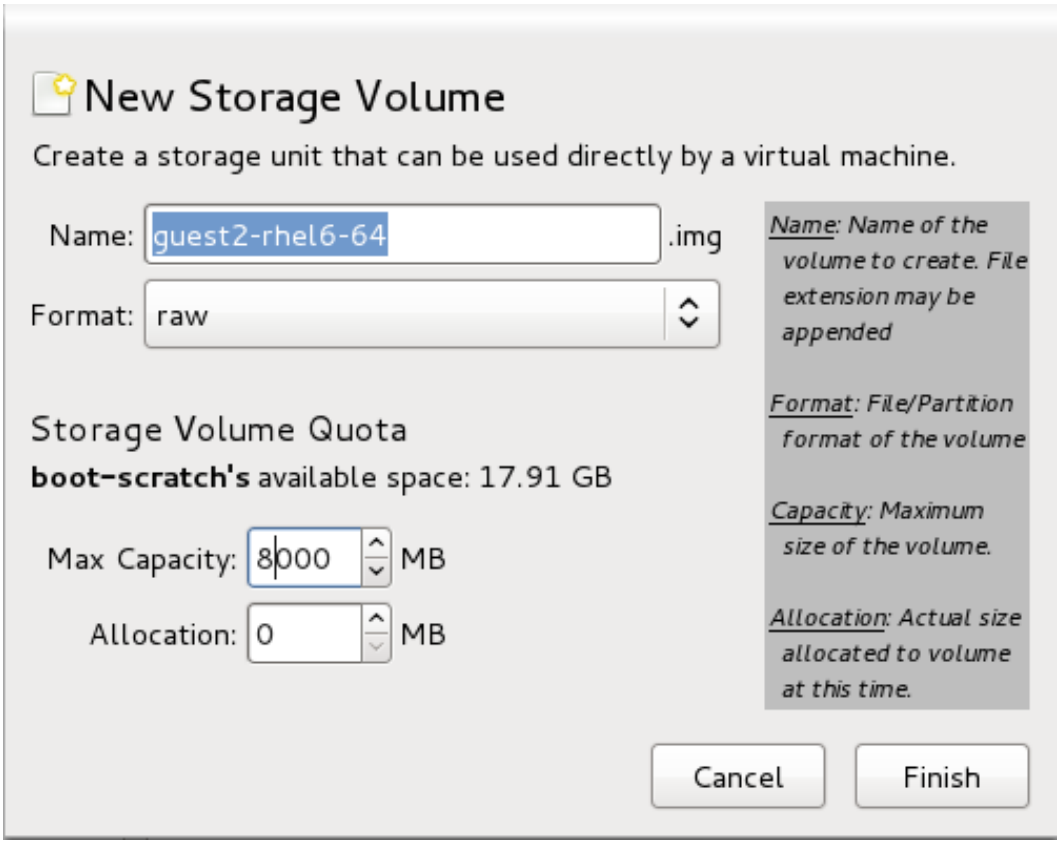



**Figure 7.9. The Add a Storage Volume window**

Click **Finish** to continue.

7. **Verify and finish**

   Verify there were no errors made during the wizard and everything appears as expected.

   Select the **Customize configuration before install** check box to change the guest's storage or network devices, to use the paravirtualized drivers or to add additional devices.

   Click on the **Advanced options** down arrow to inspect and modify advanced options. For a standard Red Hat Enterprise Linux 6 installation, none of these options require modification.

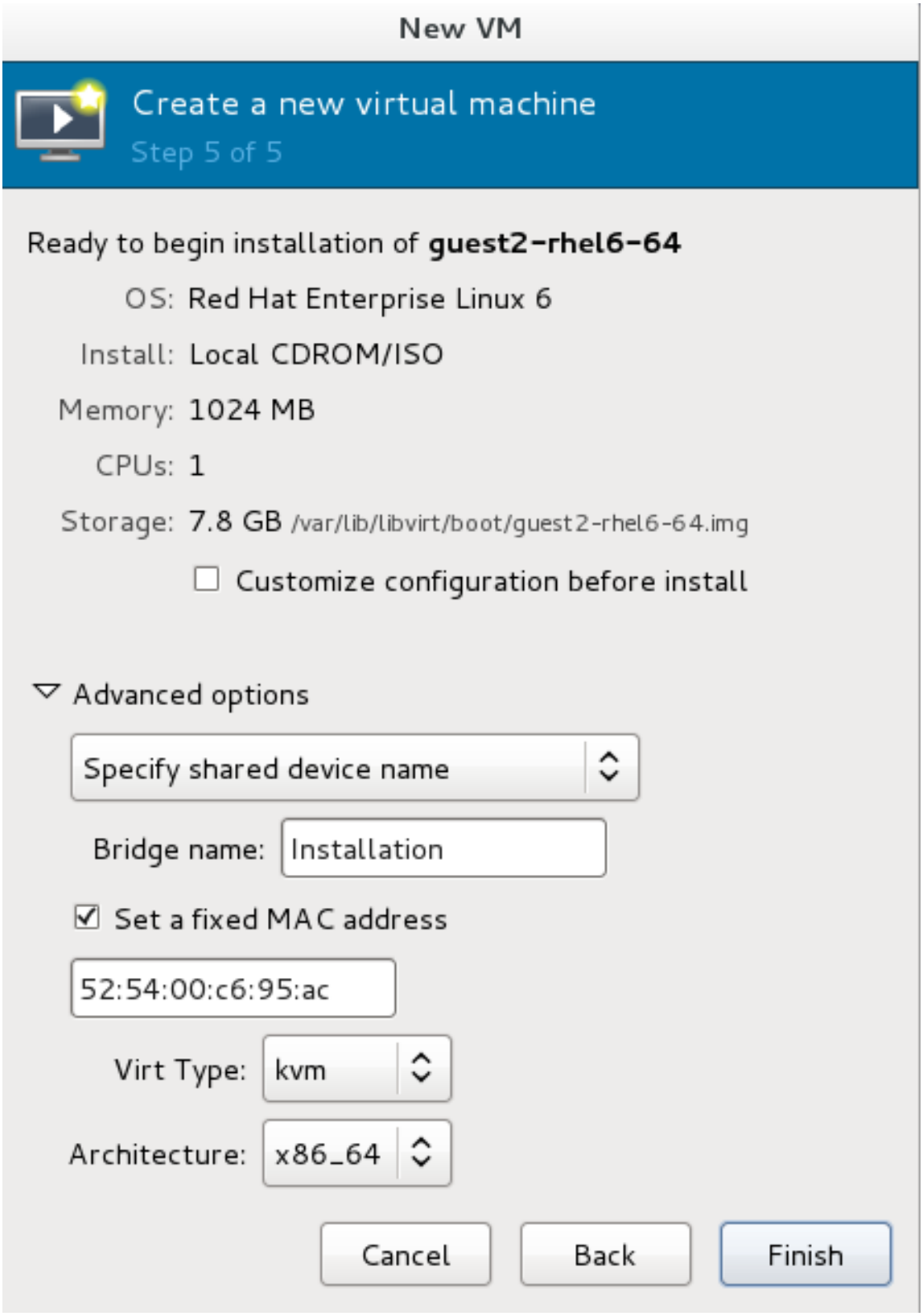



**Figure 7.10. The New VM window - local storage**

Click **Finish** to continue into the Red Hat Enterprise Linux installation sequence. For more information on installing Red Hat Enterprise Linux 6 refer to the *Red Hat Enterprise Linux 6 Installation Guide*.

A Red Hat Enterprise Linux 6 guest virtual machine is now created from an ISO installation disc image.

## 7.2. Creating a Red Hat Enterprise Linux 6 Guest with a Network Installation Tree

**Procedure 7.2. Creating a Red Hat Enterprise Linux 6 guest with virt-manager**

1. **Optional: Preparation**

   Prepare the storage environment for the guest virtual machine. For more information on preparing storage, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

   **Important**

   Various storage types may be used for storing guest virtual machines. However, for a virtual machine to be able to use migration features the virtual machine must be created on networked storage.

   Red Hat Enterprise Linux 6 requires at least 1GB of storage space. However, Red Hat recommends at least 5GB of storage space for a Red Hat Enterprise Linux 6 installation and for the procedures in this guide.

2. **Open virt-manager and start the wizard**

   Open virt-manager by executing the **virt-manager** command as root or opening **Applications** → **System Tools** → **Virtual Machine Manager**.

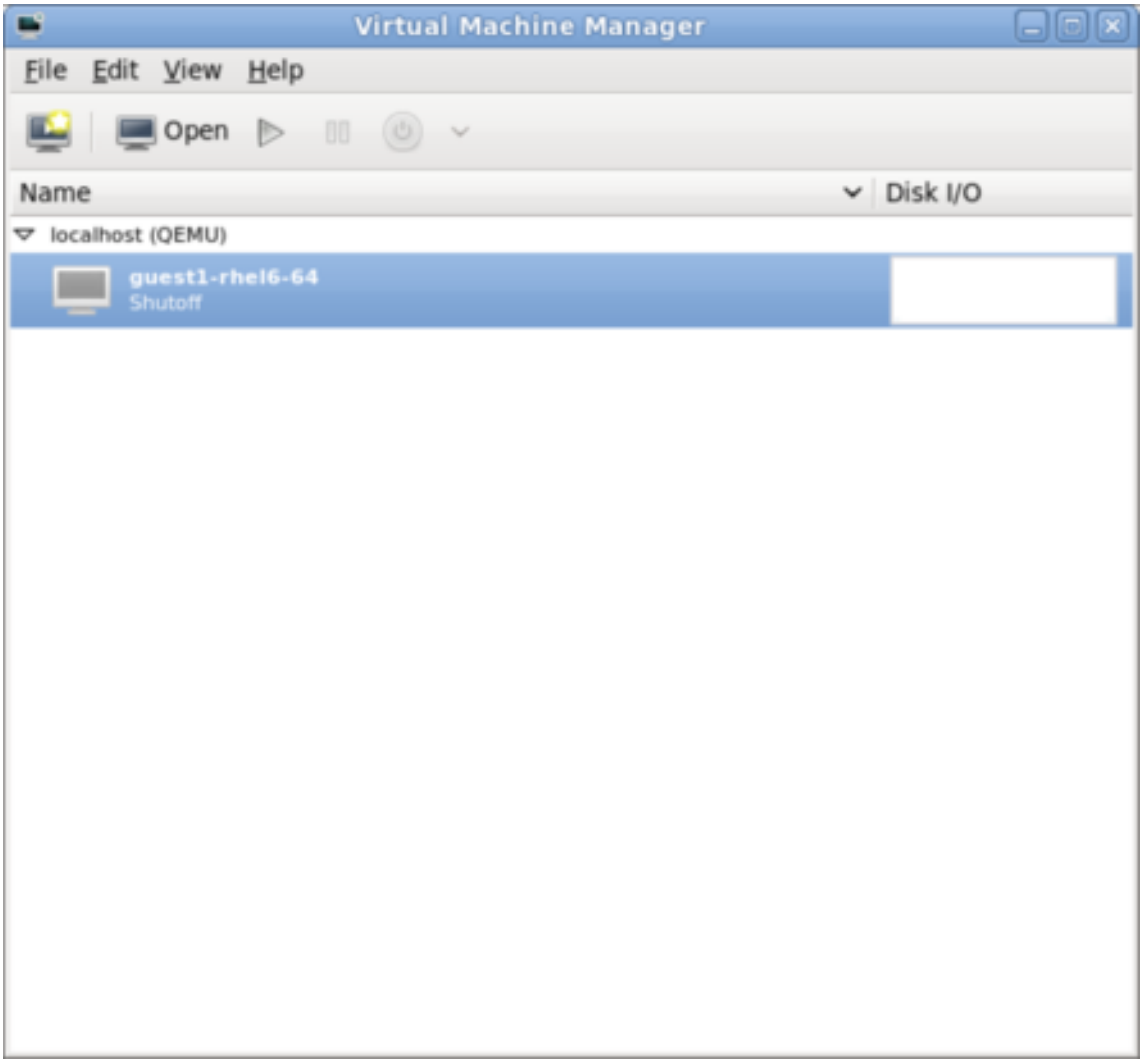



**Figure 7.11. The main virt-manager window**

Click on the **Create a new virtual machine** button to start the new virtual machine wizard.

**Figure 7.12. The Create a new virtual machine button**

The **Create a new virtual machine** window opens.

3. **Name the virtual machine**

   Virtual machine names can contain letters, numbers and the following characters: '_', '**.**' and '**-**'. Virtual machine names must be unique for migration and cannot consist only of numbers.

   Choose the installation method from the list of radio buttons.

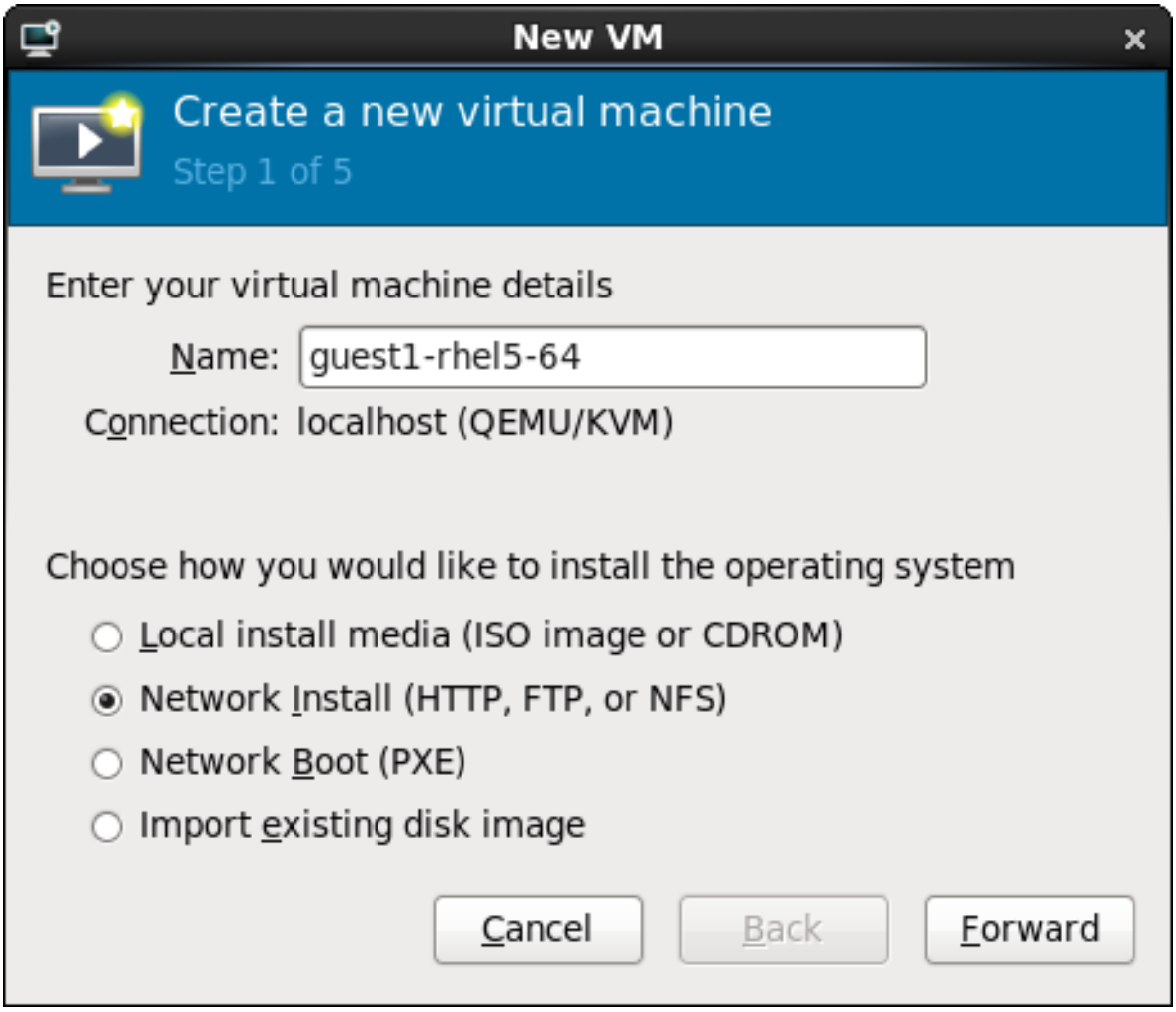



**Figure 7.13. The New VM window - Step 1**

Click **Forward** to continue.

4. Provide the installation URL, and the Kickstart URL and Kernel options if required.

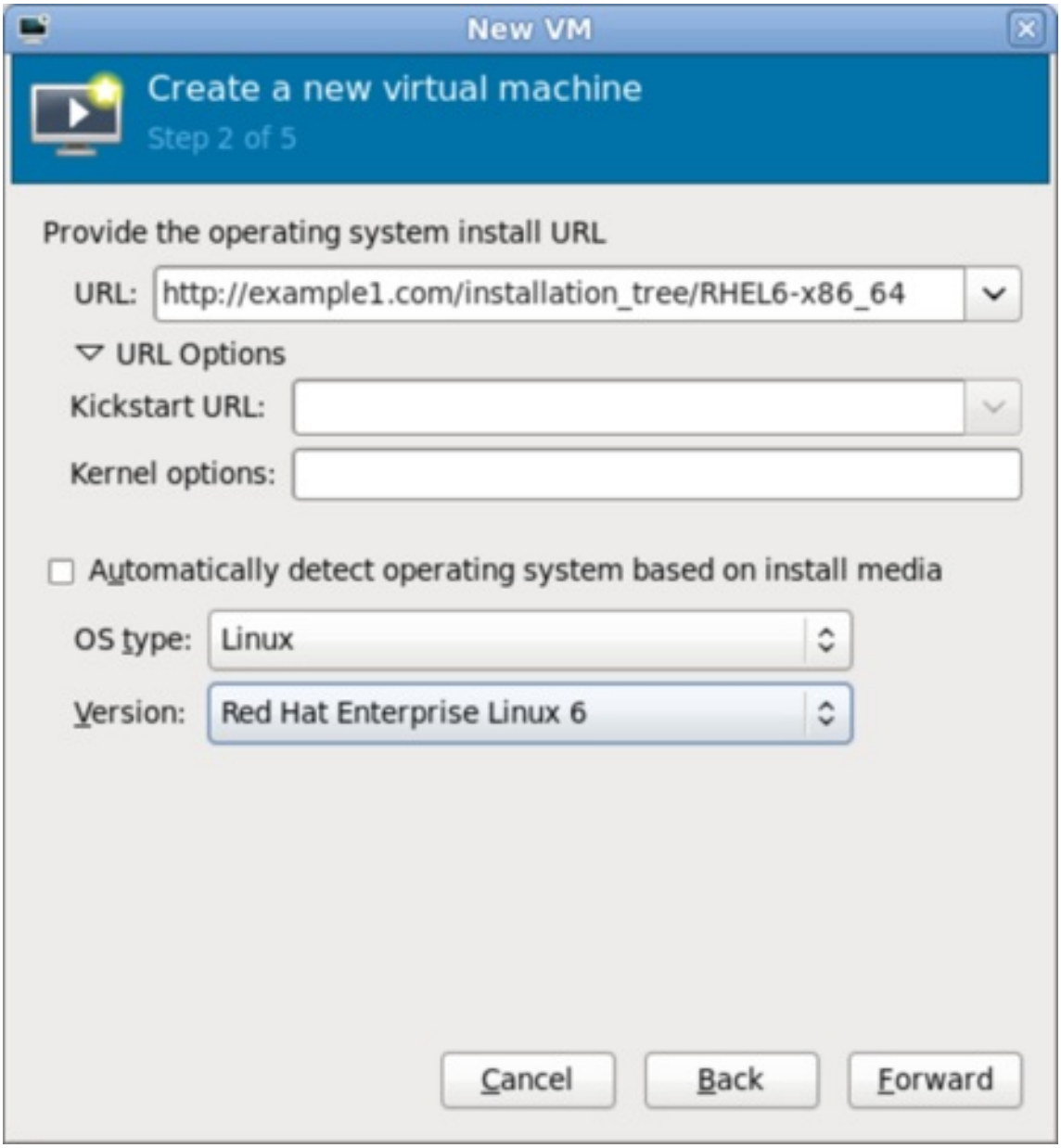



**Figure 7.14. The New VM window - Step 2**

Click **Forward** to continue.

5. The remaining steps are the same as the ISO installation procedure. Continue from Step 5 of the ISO installation procedure.

## 7.3. Creating a Red Hat Enterprise Linux 6 Guest with PXE

**Procedure 7.3. Creating a Red Hat Enterprise Linux 6 guest with virt-manager**

1. **Optional: Preparation**

   Prepare the storage environment for the virtual machine. For more information on preparing storage, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide*.

**Important**

Various storage types may be used for storing guest virtual machines. However, for a virtual machine to be able to use migration features the virtual machine must be created on networked storage.

Red Hat Enterprise Linux 6 requires at least 1GB of storage space. However, Red Hat recommends at least 5GB of storage space for a Red Hat Enterprise Linux 6 installation and for the procedures in this guide.

2. **Open virt-manager and start the wizard**

   Open virt-manager by executing the **virt-manager** command as root or opening **Applications** → **System Tools** → **Virtual Machine Manager**.

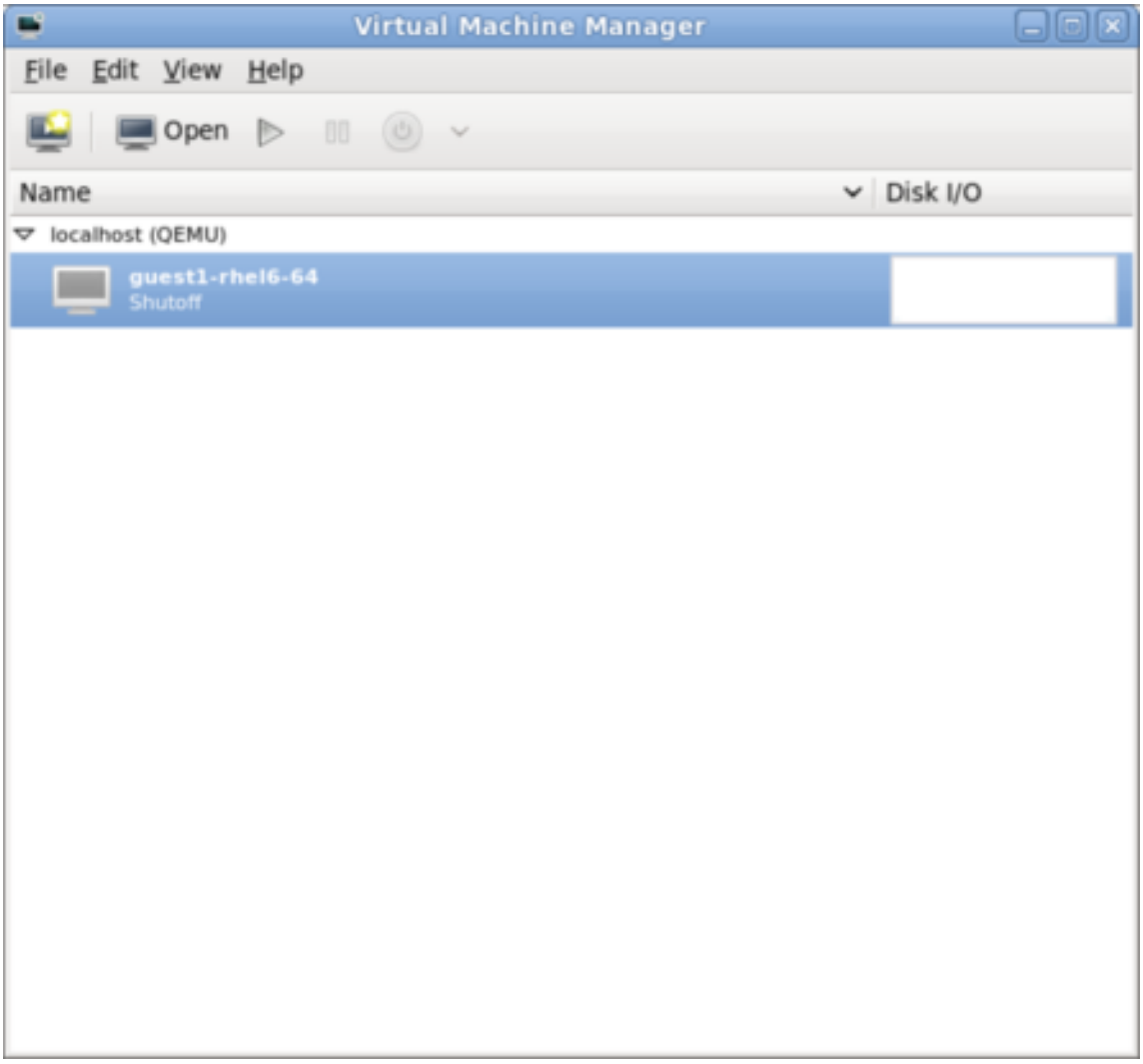



**Figure 7.15. The main virt-manager window**

Click on the **Create new virtualized guest** button to start the new virtualized guest wizard.

**Figure 7.16. The create new virtualized guest button**

The **New VM** window opens.

3. **Name the virtual machine**

   Virtual machine names can contain letters, numbers and the following characters: '_', '**.**' and '**-**'. Virtual machine names must be unique for migration and cannot consist only of numbers.

   Choose the installation method from the list of radio buttons.

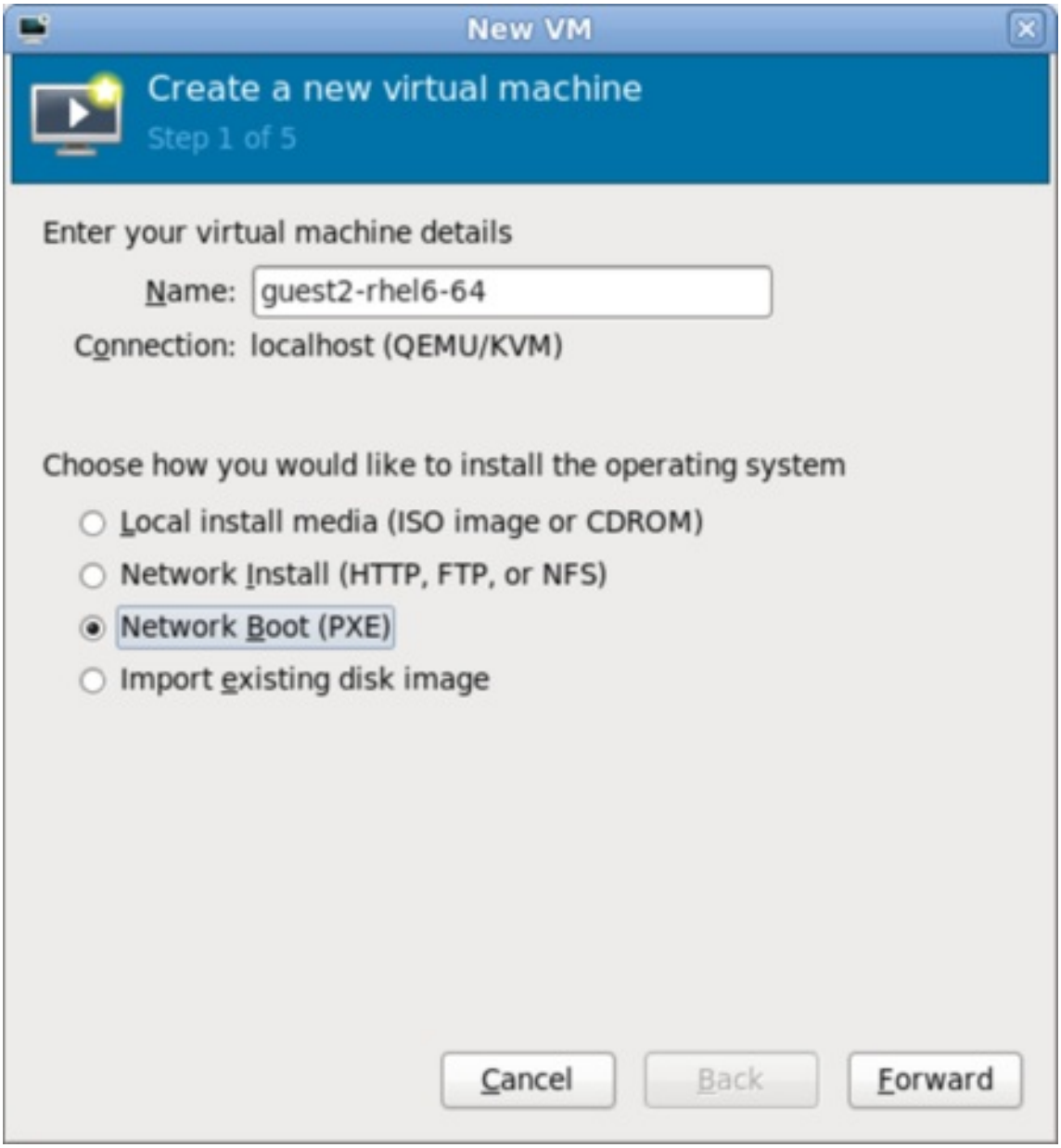



**Figure 7.17. The New VM window - Step 1**

Click **Forward** to continue.

4. The remaining steps are the same as the ISO installation procedure. Continue from Step 5 of the ISO installation procedure. From this point, the only difference in this PXE procedure is on the final **New VM** screen, which shows the **Install: PXE Install** field.

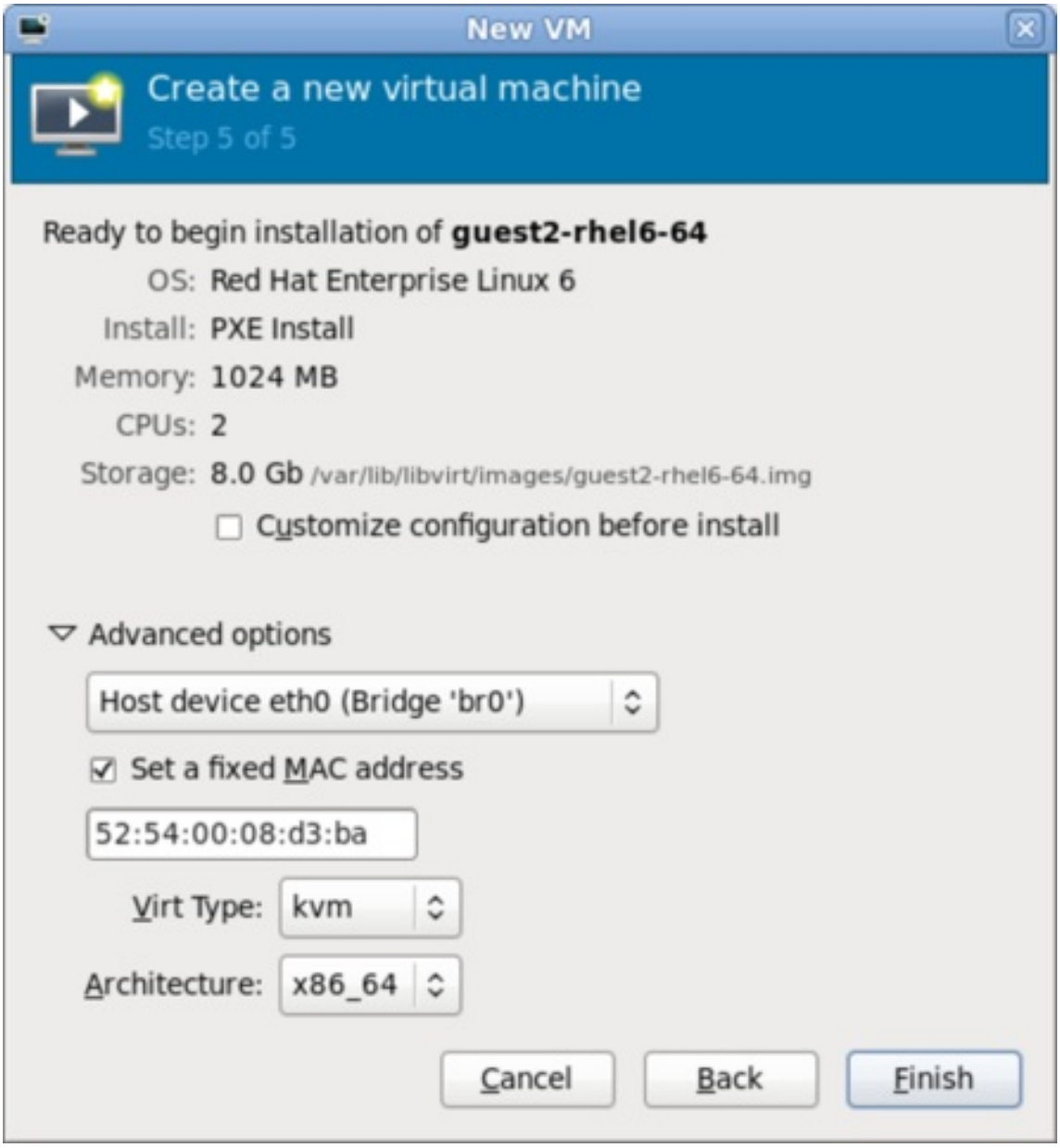



**Figure 7.18. The New VM window - Step 5 - PXE Install**

# Chapter 8. Virtualizing Red Hat Enterprise Linux on Other Platforms

This chapter contains useful reference material for customers running Red Hat Enterprise Linux 6 as a virtualized operating system on other virtualization hosts.

## 8.1. On VMware ESX

Red Hat Enterprise Linux 6.0 and onward provide the **vmw_balloon** driver, a paravirtualized memory ballooning driver used when running Red Hat Enterprise Linux on VMware hosts. For further information about this driver, refer to http://kb.VMware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1002586.

Red Hat Enterprise Linux 6.3 and onward provide the **vmmouse_drv** driver, a paravirtualized mouse driver used when running Red Hat Enterprise Linux on VMware hosts. For further information about this driver, refer to http://kb.VMware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=5739104.

Red Hat Enterprise Linux 6.3 and onward provide the **vmware_drv** driver, a paravirtualized video driver used when running Red Hat Enterprise Linux on VMware hosts. For further information about this driver, refer to http://kb.VMware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1033557.

Red Hat Enterprise Linux 6.3 and onward provide the **vmxnet3** driver, a paravirtualized network adapter used when running Red Hat Enterprise Linux on VMware hosts. For further information about this driver, refer to http://kb.VMware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1001805.

Red Hat Enterprise Linux 6.4 and onward provide the **vmw_pvscsi** driver, a paravirtualized SCSI adapter used when running Red Hat Enterprise Linux on VMware hosts. For further information about this driver, refer to http://kb.VMware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1010398.

## 8.2. On Hyper-V

Red Hat Enterprise Linux 6.4 and onward ships with a Microsoft's Linux Integration Services, a set of drivers that enable synthetic device support in supported virtualized operating systems. Further details about the drivers provided are available from http://technet.microsoft.com/en-us/library/dn531030.aspx.

**Important**

Note that the built-in Red Hat Enterprise Linux Integration Service drivers for Hyper-V are sufficient for Red Hat Enterprise Linux guests to run using the high performance synthetic devices on Hyper-V hosts. These built-in drivers are certified by Red Hat for this use, and certified configurations can be viewed on the Red Hat Customer Portal.

Therefore, it is not necessary to download and install Linux Integration Services (LIS) packages, and doing so may limit your Red Hat support, as described in the related Knowledgebase article.

The following enhancements have been made to allow for easier deployment and management of Red Hat Enterprise Linux guest virtual machines:

- Upgraded VMBUS protocols - VMBUS protocols have been upgraded to Windows 8 level. As part of this work, now VMBUS interrupts can be processed on all available virtual CPUs in the guest. Furthermore, the signaling protocol between the Red Hat Enterprise Linux guest virtual machine and the Windows host physical machine has been optimized.
- Synthetic frame buffer driver - Provides enhanced graphics performance and superior resolution for Red Hat Enterprise Linux desktop users.
- Live Virtual Machine Backup support - Provisions uninterrupted backup support for live Red Hat Enterprise Linux guest virtual machines.
- Dynamic expansion of fixed size Linux VHDXs - Allows expansion of live mounted fixed size Red Hat Enterprise Linux VHDXs.
- Boot using UEFI - Allows virtual machines to boot using Unified Extensible Firmware Interface (UEFI) on a Hyper-V 2012 R2 host.

For more information and features, refer to *Red Hat Enterprise Linux — Virtualization Administration Guide* and to the following article:" Enabling Linux Support on Windows Server 2012 R2 Hyper-V".

# Chapter 9. Installing a Fully-virtualized Windows Guest

This chapter describes how to create a fully-virtualized Windows guest using the command-line (**virt-install**), launch the operating system's installer inside the guest, and access the installer through **virt-viewer**.

To install a Windows operating system on the guest, use the **virt-viewer** tool. This tool allows you to display the graphical console of a virtual machine (via the spice or VNC protocol). In doing so, **virt-viewer** allows you to install a fully-virtualized guest's operating system with that operating system's installer .

Installing a Windows operating system involves two major steps:

1. Creating the guest virtual machine, using either **virt-install** or **virt-manager**.
2. Installing the Windows operating system on the guest virtual machine, using **virt-viewer**.

Refer to Chapter 6, *Guest Virtual Machine Installation Overview* for details about creating a guest virtual machine with **virt-install** or **virt-manager**.

Note that this chapter does not describe how to install a Windows operating system on a fully-virtualized guest. Rather, it only covers how to create the guest and launch the installer within the guest. For information on how to install a Windows operating system, refer to the relevant Microsoft installation documentation.

## 9.1. Using virt-install to Create a Guest

The **virt-install** command allows you to create a fully-virtualized guest from a terminal, for example, without a GUI.

**Important**

Before creating the guest, consider first if the guest needs to use KVM Windows paravirtualized drivers. If it does, keep in mind that you can do so *during* or *after* installing the Windows operating system on the guest. For more information about paravirtualized drivers, refer to Chapter 10, *KVM Paravirtualized (virtio) Drivers*.

For instructions on how to install KVM paravirtualized drivers, refer to Section 10.1, “Installing the KVM Windows virtio Drivers”.

It is possible to create a fully-virtualized guest with only a single command. To do so, run the following program (replace the values accordingly):

```
# virt-install \
   --name=guest-name \
   --os-type=windows \
   --network network=default \
   --disk path=path-to-disk,size=disk-size \
   --cdrom=path-to-install-disk \
   --graphics spice --ram=1024
```

The ***path-to-disk*** must be a device (e.g. **/dev/sda3**) or image file (**/var/lib/libvirt/images/*name*.img**). It must also have enough free space to support the ***disk-size***.

**Important**

All image files are stored in **/var/lib/libvirt/images/** by default. Other directory locations for file-based images are possible, but may require SELinux configuration. If you run SELinux in enforcing mode, refer to the *Red Hat Enterprise Linux 6 Virtualization Administration Guide* for more information on SELinux.

You can also run **virt-install** interactively. To do so, use the **--prompt** command, as in:

```
# virt-install --prompt
```

Once the fully-virtualized guest is created, **virt-viewer** will launch the guest and run the operating system's installer. Refer to the relevant Microsoft installation documentation for instructions on how to install the operating system.

# Chapter 10. KVM Paravirtualized (virtio) Drivers

Paravirtualized drivers enhance the performance of guests, decreasing guest I/O latency and increasing throughput to near bare-metal levels. It is recommended to use the paravirtualized drivers for fully virtualized guests running I/O heavy tasks and applications.

Virtio drivers are KVM's paravirtualized device drivers, available for Windows guest virtual machines running on KVM hosts. These drivers are included in the virtio package. The virtio package supports block (storage) devices and network interface controllers.

The KVM virtio drivers are automatically loaded and installed on the following:

- Red Hat Enterprise Linux 4.8 and newer
- Red Hat Enterprise Linux 5.3 and newer
- Red Hat Enterprise Linux 6 and newer
- Red Hat Enterprise Linux 7 and newer
- Some versions of Linux based on the 2.6.27 kernel or newer kernel versions.

Versions of Red Hat Enterprise Linux in the list above detect and install the drivers, additional installation steps are not required.

In Red Hat Enterprise Linux 3 (3.9 and above), manual installation is required.

**Note**

PCI devices are limited by the virtualized system architecture. Refer to Section 4.1, “KVM Restrictions” for additional limitations when using assigned devices.

Using KVM virtio drivers, the following Microsoft Windows versions are expected to run similarly to bare-metal-based systems.

- Windows Server 2003 (32-bit and 64-bit versions)
- Windows Server 2008 (32-bit and 64-bit versions)
- Windows Server 2008 R2 (64-bit only)
- Windows 7 (32-bit and 64-bit versions)
- Windows Server 2012 (64-bit only)
- Windows Server 2012 R2 (64-bit only)
- Windows 8 (32-bit and 64-bit versions)
- Windows 8.1 (32-bit and 64-bit versions)

## 10.1. Installing the KVM Windows virtio Drivers

This section covers the installation process for the KVM Windows virtio drivers. The KVM virtio drivers can be loaded during the Windows installation or installed after the guest is installed.

You can install the virtio drivers on a guest virtual machine using one of the following methods:

- hosting the installation files on a network accessible to the virtual machine
- using a virtualized CD-ROM device of the driver installation disk .iso file
- using a USB drive, by mounting the same (provided) .ISO file that you would use for the CD-ROM
- using a virtualized floppy device to install the drivers during boot time (required and recommended only for XP/2003)

This guide describes installation from the paravirtualized installer disk as a virtualized CD-ROM device.

1. **Download the drivers**

   The *virtio-win* package contains the virtio block and network drivers for all supported Windows guest virtual machines.

   Download and install the *virtio-win* package on the host with the **yum** command.

   ```
   # yum install virtio-win
   ```

   The list of virtio-win packages that are supported on Windows operating systems, and the current certified package version, can be found at the following URL: windowsservercatalog.com.

   Note that the Red Hat Enterprise Virtualization Hypervisor and Red Hat Enterprise Linux are created on the same code base so the drivers for the same version (for example, Red Hat Enterprise Virtualization Hypervisor 3.3 and Red Hat Enterprise Linux 6.5) are supported for both environments.

   The *virtio-win* package installs a CD-ROM image, **virtio-win.iso**, in the **/usr/share/virtio-win/** directory.

2. **Install the virtio drivers**

   When booting a Windows guest that uses virtio-win devices, the relevant virtio-win device drivers must already be installed on this guest. The virtio-win drivers are not provided as inbox drivers in Microsoft's Windows installation kit, so installation of a Windows guest on a virtio-win storage device (viostor/virtio-scsi) requires that you provide the appropriate driver *during* the installation, either directly from the **virtio-win.iso** or from the supplied Virtual Floppy image **virtio-win<*version*>.vfd**.

## 10.2. Installing the Drivers on an Installed Windows Guest Virtual Machine

This procedure covers installing the virtio drivers with a virtualized CD-ROM after Windows is installed.

Follow this procedure to add a CD-ROM image with **virt-manager** and then install the drivers.

**Procedure 10.1. Installing from the driver CD-ROM image with virt-manager**

1. **Open virt-manager and the guest virtual machine**

Open **virt-manager**, then open the guest virtual machine from the list by double-clicking the guest name.

2. **Open the hardware window**

   Click the lightbulb icon on the toolbar at the top of the window to view virtual hardware details.

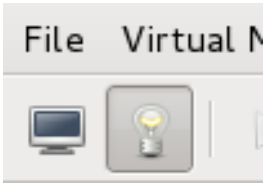



**Figure 10.1. The virtual hardware details button**

Then click the **Add Hardware** button at the bottom of the new view that appears. This opens a wizard for adding the new device.

3. **Select the device type — for Red Hat Enterprise Linux 6 versions prior to 6.2**

   Skip this step if you are using Red Hat Enterprise Linux 6.2 or later.

   On Red Hat Enterprise Linux 6 versions prior to version 6.2, you must select the type of device you wish to add. In this case, select **Storage** from the dropdown menu.


**Figure 10.2. The Add new virtual hardware wizard in Red Hat Enterprise Linux 6.1**

Click the **Finish** button to proceed.

4. **Select the ISO file**

   Ensure that the **Select managed or other existing storage** radio button is selected, and browse to the virtio driver's **.iso** image file. The default location for the latest version of the drivers is **/usr/share/virtio-win/virtio-win.iso**.

   Change the **Device type** to **IDE cdrom** and click the **Forward** button to proceed.

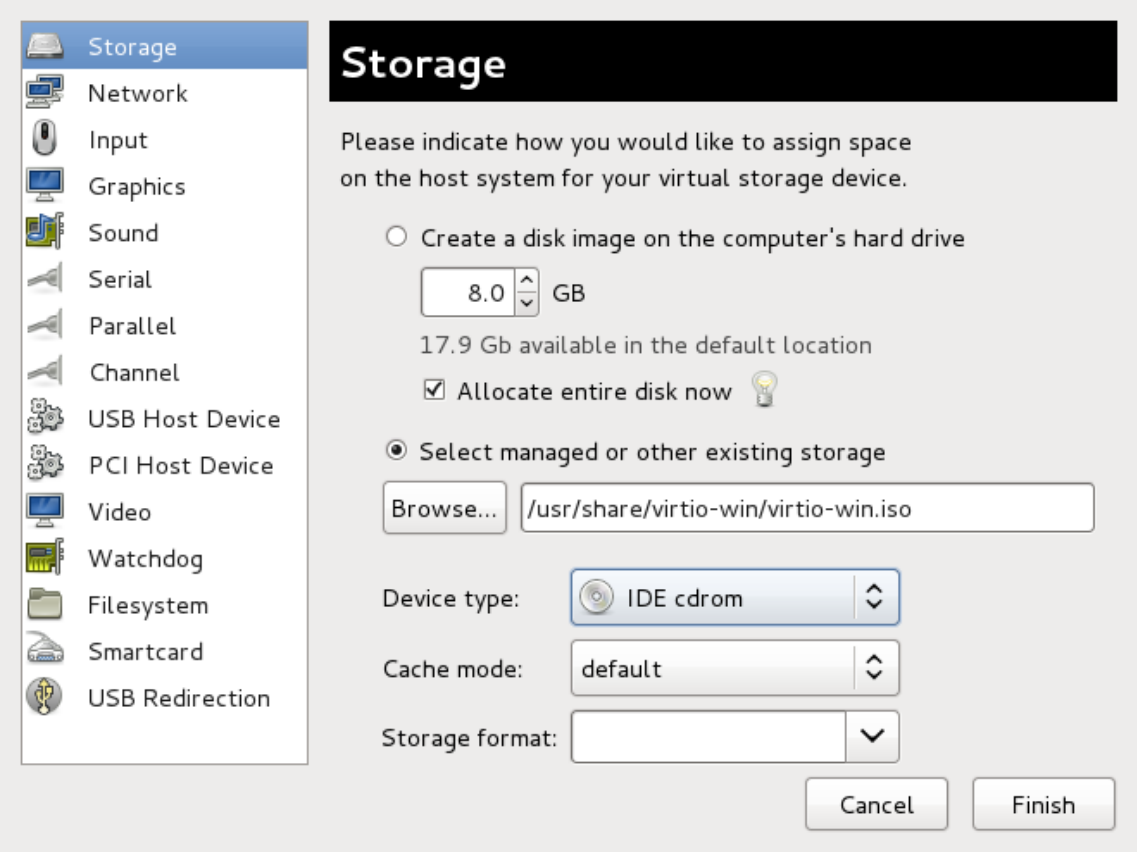



**Figure 10.3. The Add new virtual hardware wizard**

5. **Finish adding virtual hardware — for Red Hat Enterprise Linux 6 versions prior to 6.2**

   If you are using Red Hat Enterprise Linux 6.2 or later, skip this step.

   On Red Hat Enterprise Linux 6 versions prior to version 6.2, click on the **Finish** button to finish adding the virtual hardware and close the wizard.


**Figure 10.4. The Add new virtual hardware wizard in Red Hat Enterprise Linux 6.1**

6. **Reboot**

   Reboot or start the virtual machine to begin using the driver disc. Virtualized IDE devices require a restart to for the virtual machine to recognize the new device.

Once the CD-ROM with the drivers is attached and the virtual machine has started, proceed with Procedure 10.2, “Windows installation on a Windows 7 virtual machine”.

**Procedure 10.2. Windows installation on a Windows 7 virtual machine**

This procedure installs the drivers on a Windows 7 virtual machine as an example. Adapt the Windows installation instructions to your guest's version of Windows.

1. **Open the Computer Management window**

   On the desktop of the Windows virtual machine, click the **Windows** icon at the bottom corner of the screen to open the Start menu.

   Right-click on **Computer** and select **Manage** from the pop-up menu.

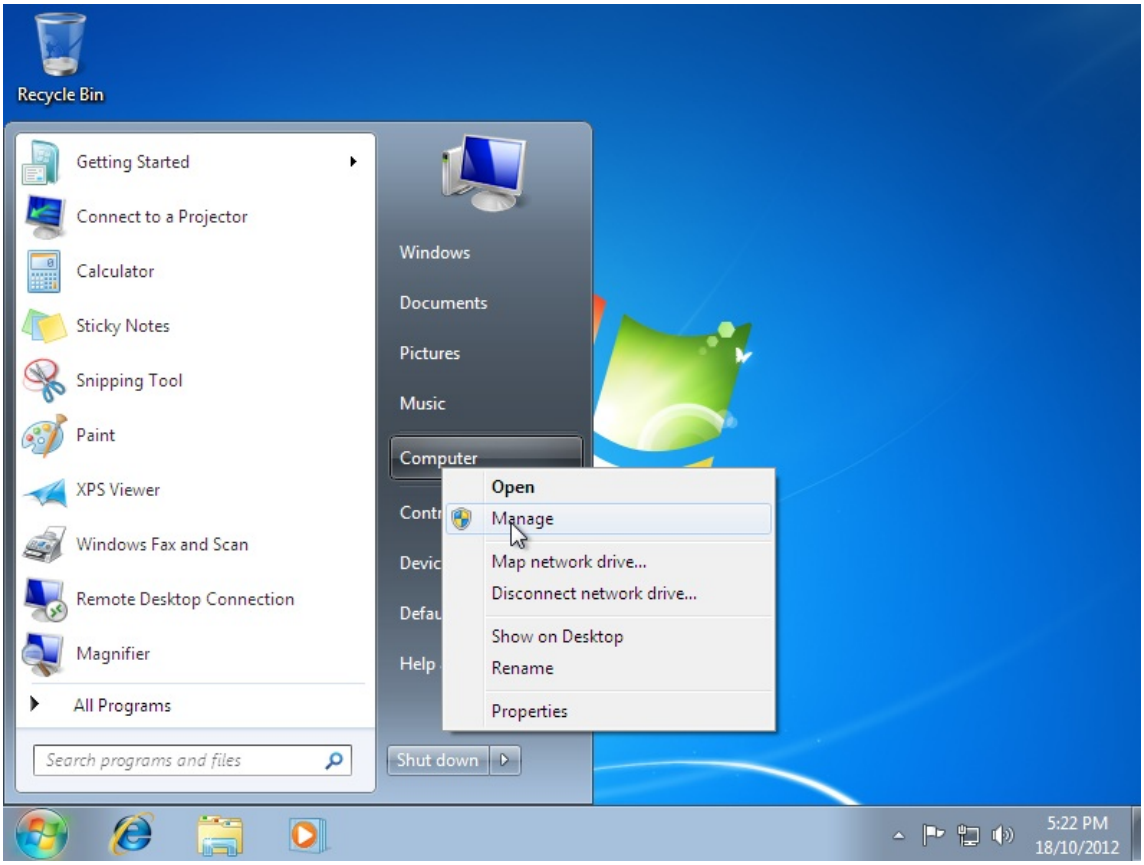



**Figure 10.5. The Computer Management window**

2. **Open the Device Manager**

   Select the **`Device Manager`** from the left-most pane. This can be found under **`Computer Management`** > **`System Tools`**.

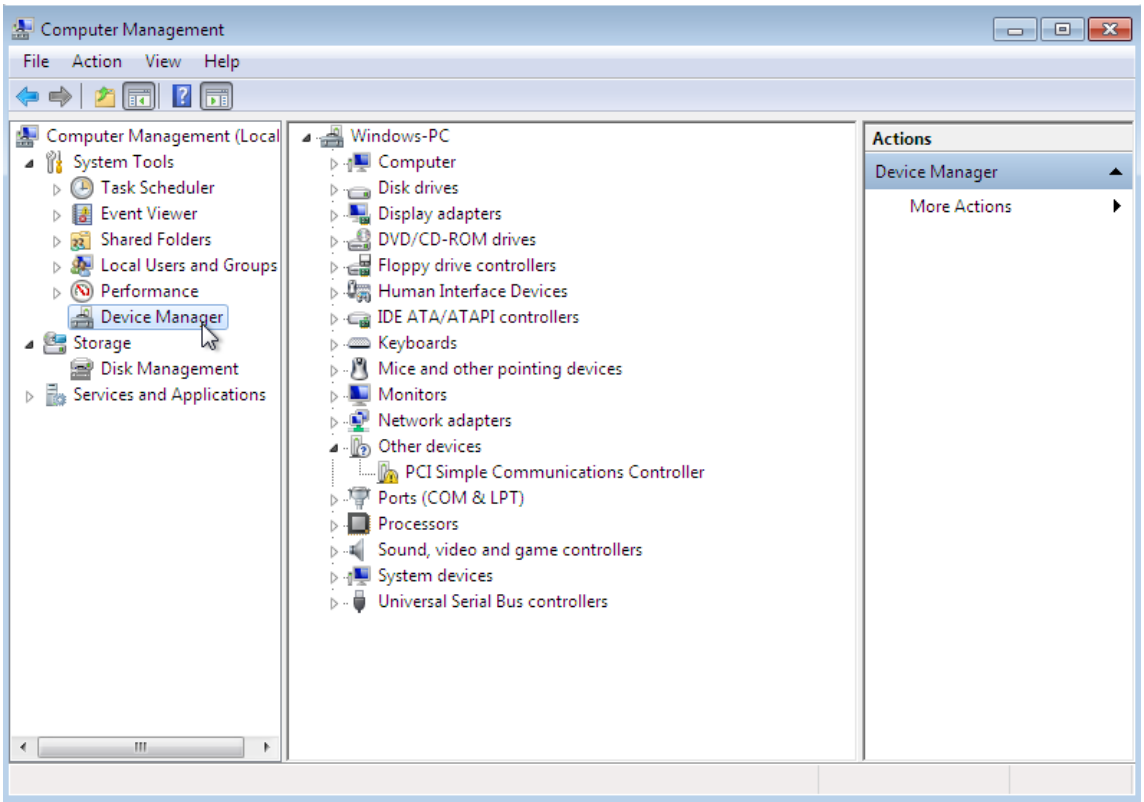


**Figure 10.6. The Computer Management window**

3. **Start the driver update wizard**

   a. **View available system devices**

   Expand **System devices** by clicking on the arrow to its left.

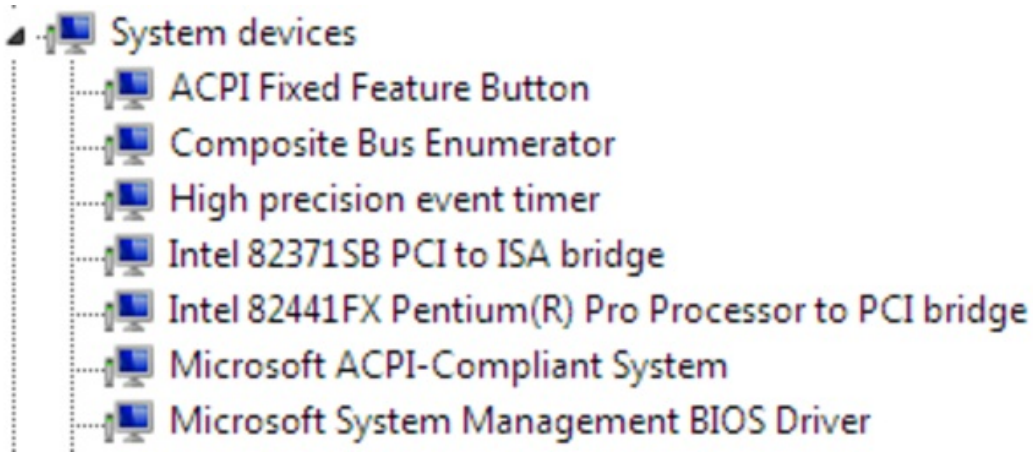



**Figure 10.7. Viewing available system devices in the Computer Management window**

   b. **Locate the appropriate device**

   There are up to four drivers available: the balloon driver, the serial driver, the network driver, and the block driver.

   - **Balloon**, the balloon driver, affects the **PCI standard RAM Controller** in the **System devices** group.
   - **vioserial**, the serial driver, affects the **PCI Simple Communication Controller** in the **System devices** group.
   - **NetKVM**, the network driver, affects the **Network adapters** group. This driver is only available if a virtio NIC is configured. Configurable parameters for this driver are documented in Appendix A, *NetKVM Driver Parameters*.
   - **viostor**, the block driver, affects the **Disk drives** group. This driver is only available if a virtio disk is configured.

   Right-click on the device whose driver you wish to update, and select **Update Driver...** from the pop-up menu.

   This example installs the balloon driver, so right-click on **PCI standard RAM Controller**.

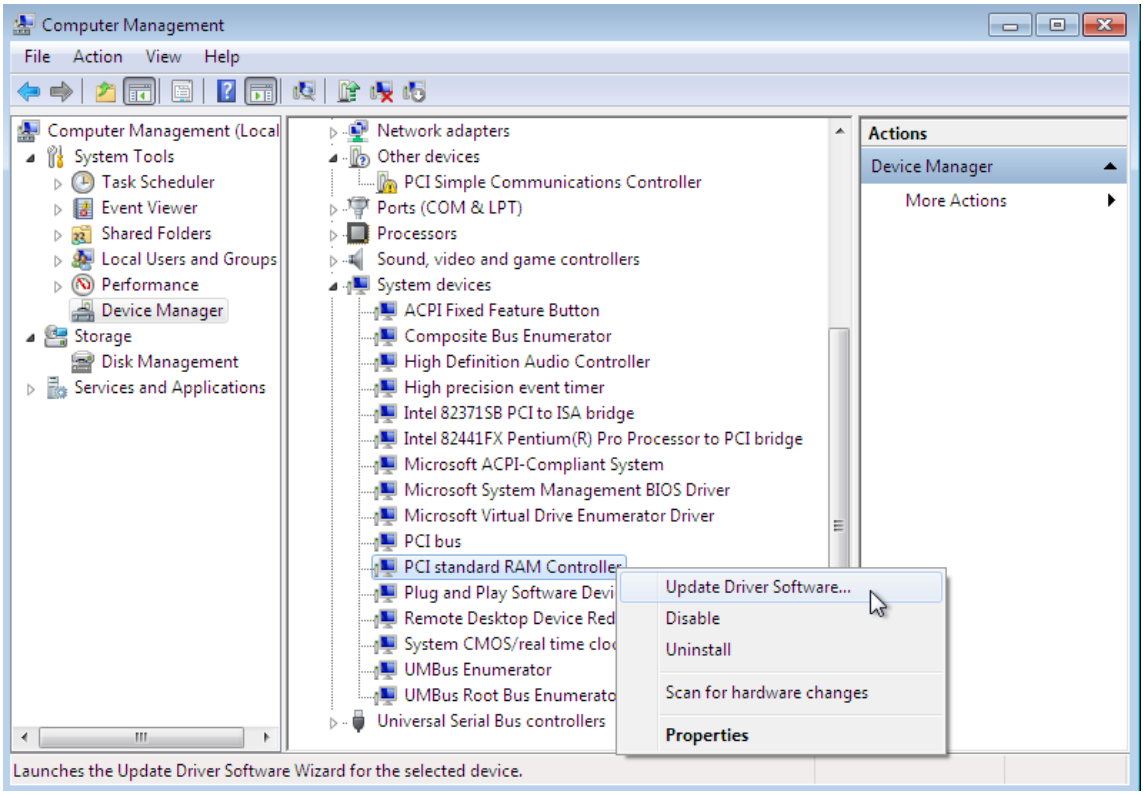



**Figure 10.8. The Computer Management window**

c. **Open the driver update wizard**

From the drop-down menu, select **`Update Driver Software...`** to access the driver update wizard.

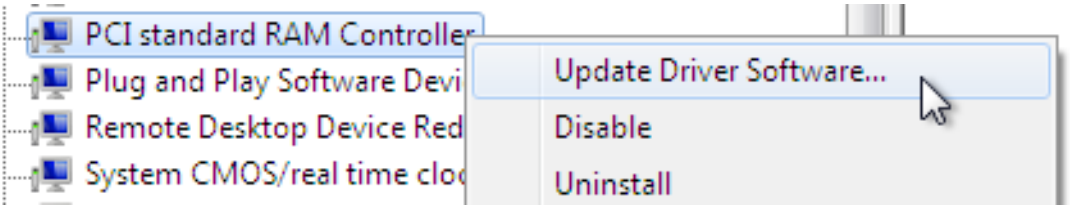



**Figure 10.9. Opening the driver update wizard**

4. **Specify how to find the driver**

The first page of the driver update wizard asks how you want to search for driver software. Click on the second option, **`Browse my computer for driver software`**.

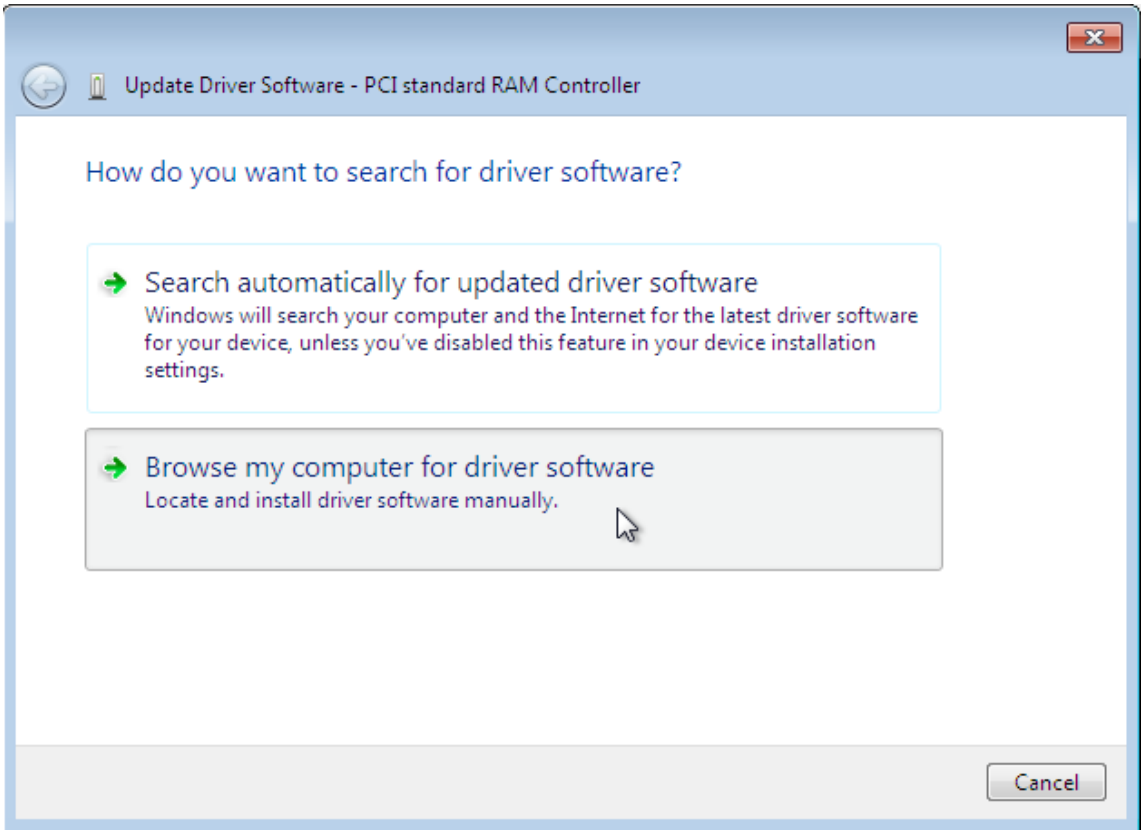



**Figure 10.10. The driver update wizard**

5. **Select the driver to install**

   a. **Open a file browser**

   Click on **Browse...**

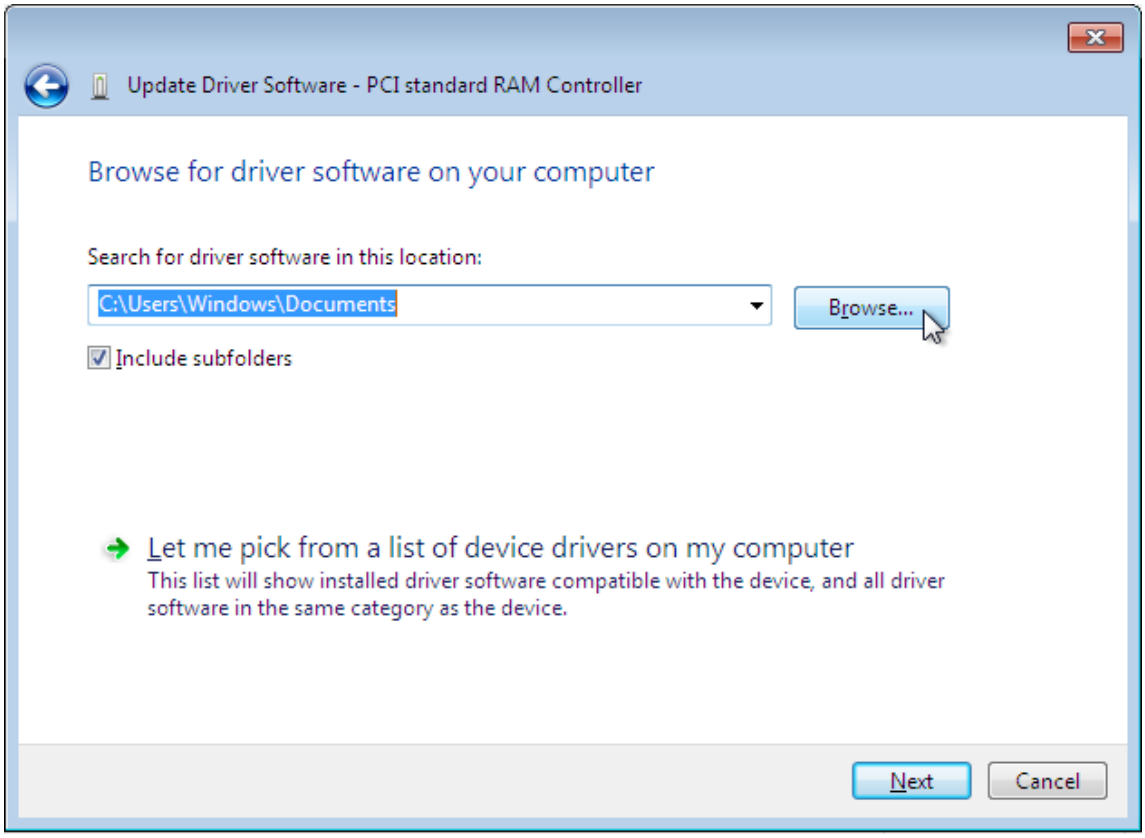


**Figure 10.11. The driver update wizard**

b. **Browse to the location of the driver**

A separate driver is provided for each of the various combinations of operating system and architecture. The drivers are arranged hierarchically according to their driver type, the operating system, and the architecture on which they will be installed: ***driver_type/os/arch/***. For example, the Balloon driver for a Windows 7 operating system with an x86 (32-bit) architecture, resides in the **Balloon/w7/x86** directory.

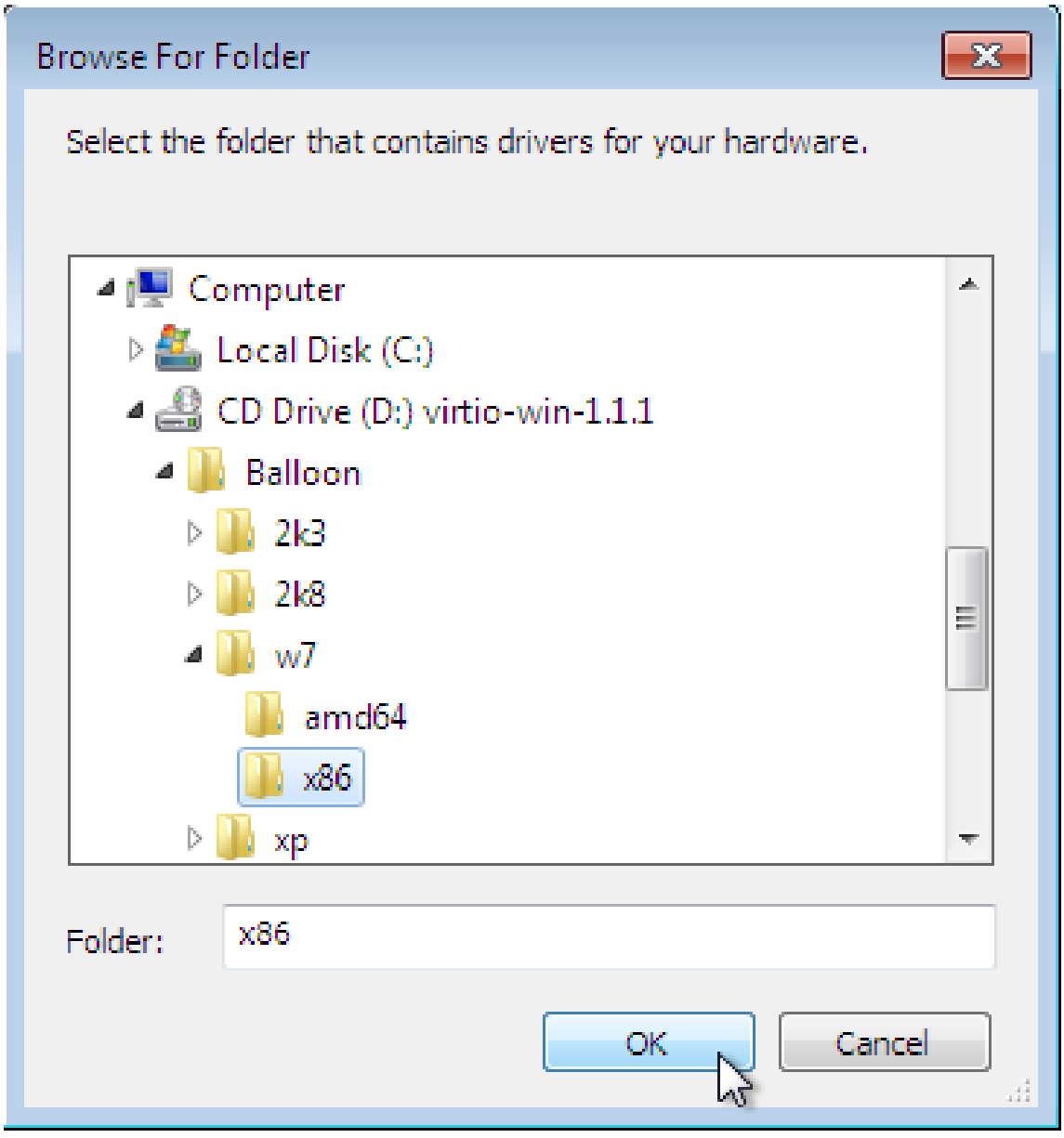



**Figure 10.12. The Browse for driver software pop-up window**

Once you have navigated to the correct location, click **OK**.

c. **Click Next to continue**

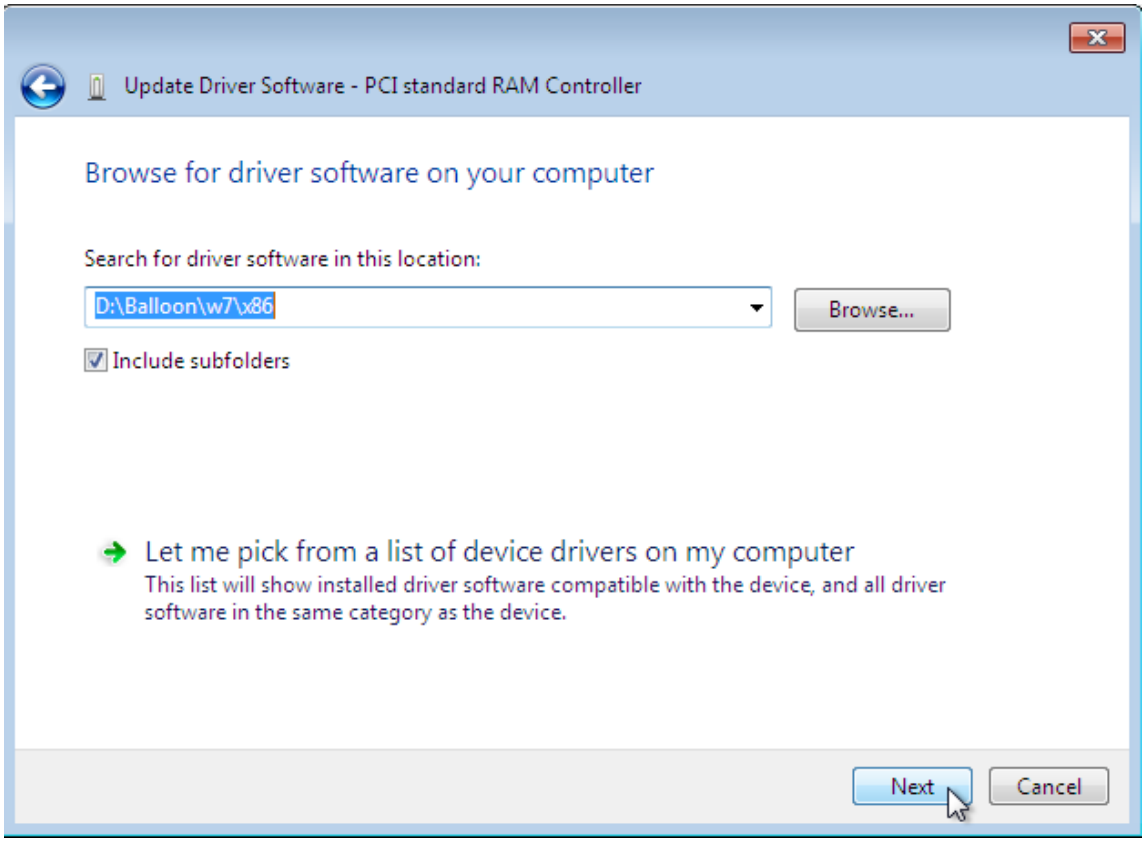



**Figure 10.13. The Update Driver Software wizard**

The following screen is displayed while the driver installs:

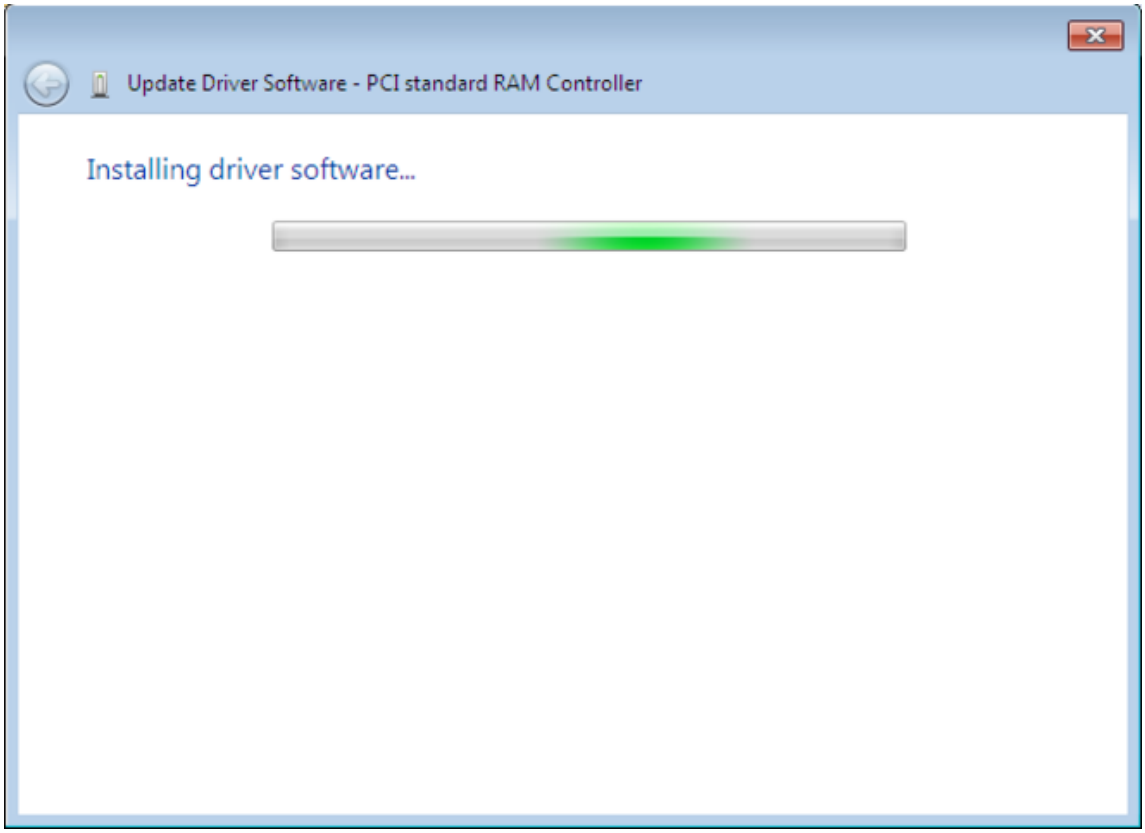



**Figure 10.14. The Update Driver Software wizard**

6. **Close the installer**

   The following screen is displayed when installation is complete:

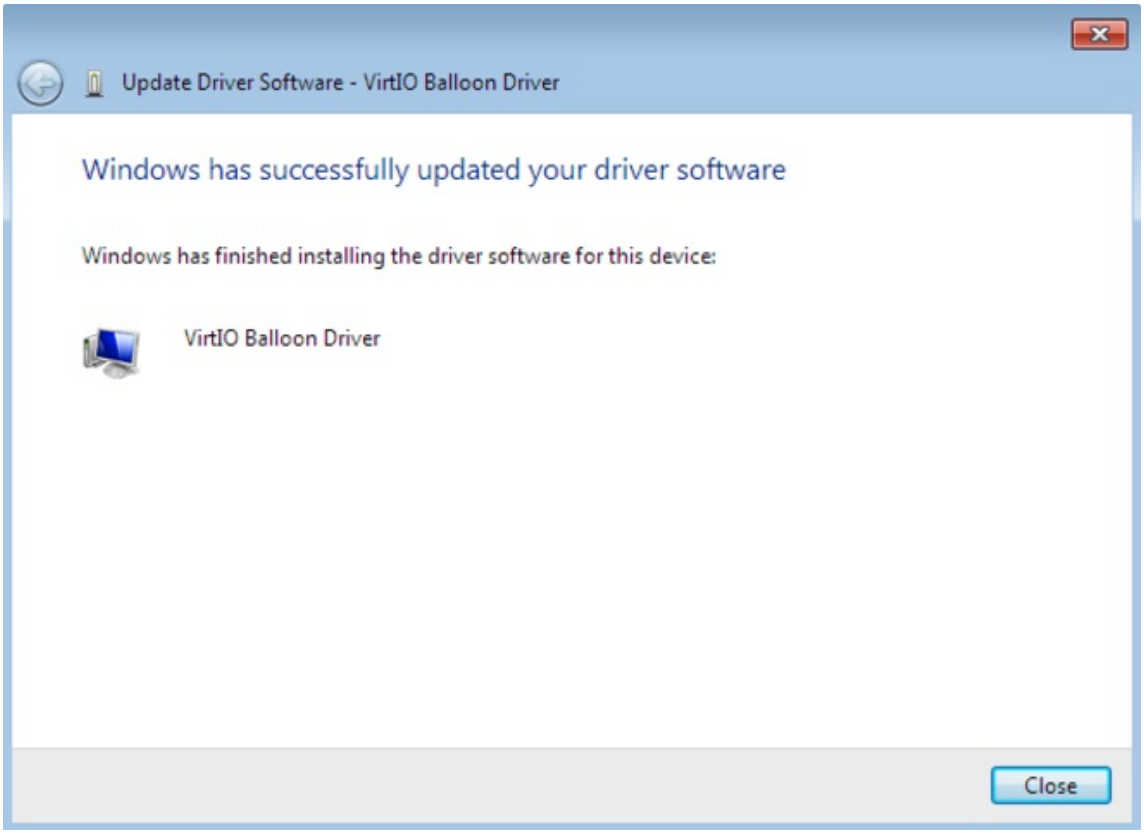



**Figure 10.15. The Update Driver Software wizard**

   Click **Close** to close the installer.

7. **Reboot**

   Reboot the virtual machine to complete the driver installation.

## 10.3. Installing Drivers during the Windows Installation

This procedure covers installing the virtio drivers during a Windows installation.

This method allows a Windows guest virtual machine to use the virtio drivers for the default storage device.

**Procedure 10.3. Installing virtio drivers during the Windows installation**

1. **Install the virtio-win package**

   Use the following command to install the *virtio-win* package:

```
# yum install virtio-win
```

2. **Create the guest virtual machine**

**Important**

Create the virtual machine, as normal, without starting the virtual machine. Follow one of the procedures below.

Select *one* of the following guest-creation methods, and follow the instructions.

a. **Create the guest virtual machine with virsh**

This method attaches the virtio driver floppy disk to a Windows guest *before* the installation.

If the virtual machine is created from an XML definition file with **virsh**, use the **virsh define** command not the **virsh create** command.

i. Create, but do not start, the virtual machine. Refer to the *Red Hat Enterprise Linux Virtualization Administration Guide* for details on creating virtual machines with the **virsh** command.

ii. Add the driver disk as a virtualized floppy disk with the **virsh** command. This example can be copied and used if there are no other virtualized floppy devices attached to the guest virtual machine. Note that *vm_name* should be replaced with the name of the virtual machine.

```
# virsh attach-disk vm_name /usr/share/virtio-
win/virtio-win.vfd fda --type floppy
```

You can now continue with Step 3.

b. **Create the guest virtual machine with virt-manager and changing the disk type**

i. At the final step of the virt-manager guest creation wizard, check the **Customize configuration before install** checkbox.

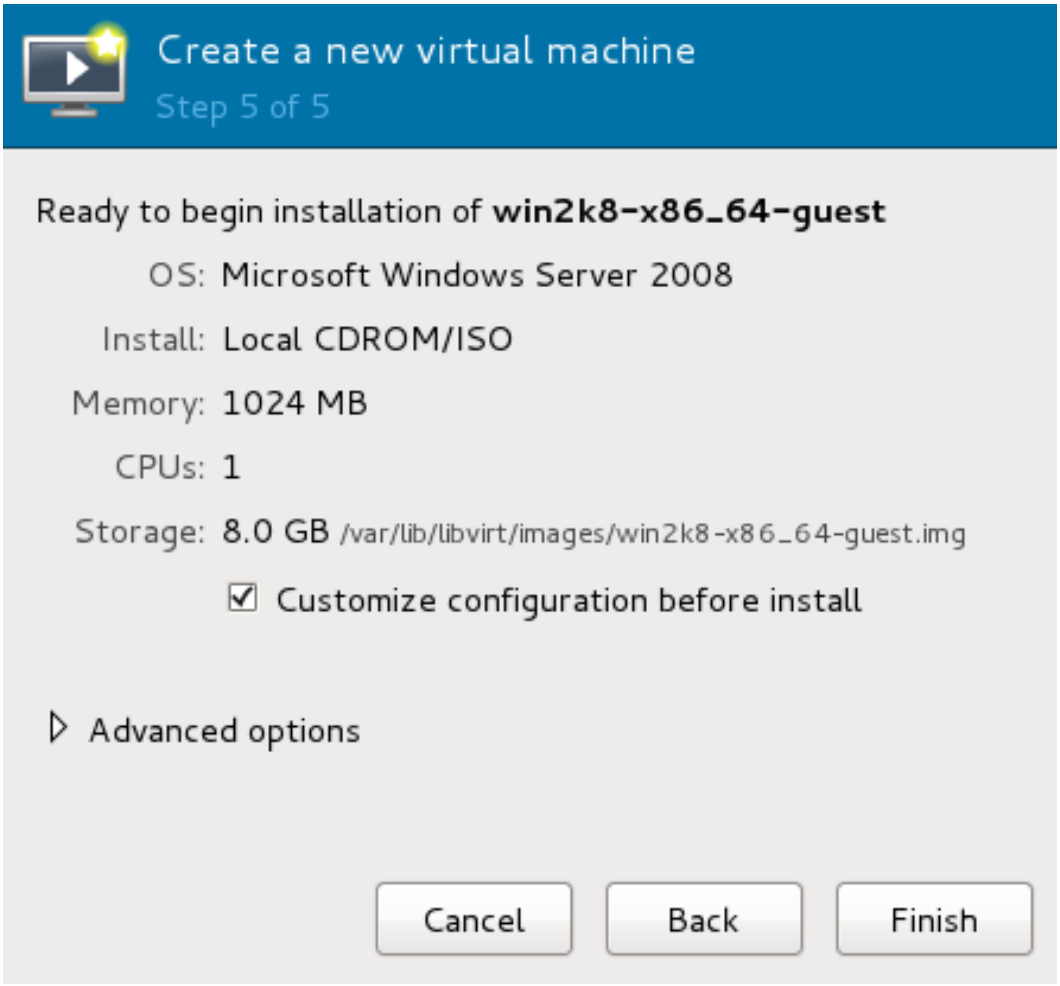



**Figure 10.16. The virt-manager guest creation wizard**

Click on the **Finish** button to continue.

ii. **Open the Add Hardware wizard**

Click the **Add Hardware** button in the bottom left of the new panel.

iii. **Select storage device**

**Storage** is the default selection in the **Hardware type** list.


**Figure 10.17. The Add new virtual hardware wizard**

Ensure the **Select managed or other existing storage** radio button is selected. Click **Browse...**.

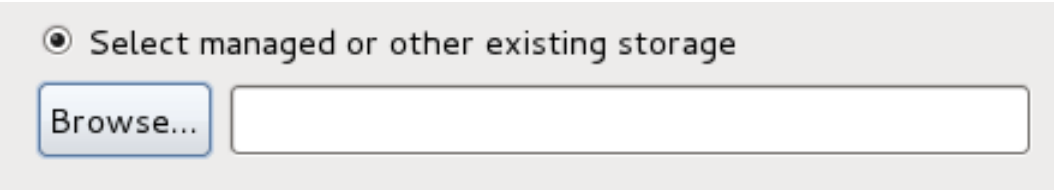



**Figure 10.18. Select managed or existing storage**

In the new window that opens, click **Browse Local**. Navigate to **/usr/share/virtio-win/virtio-win.vfd**, and click **Select** to confirm.

Change **Device type** to **Floppy disk**, and click **Finish** to continue.

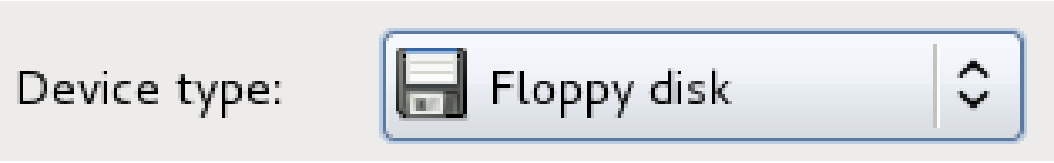



**Figure 10.19. Change the Device type**

iv. **Confirm settings**

Review the device settings.

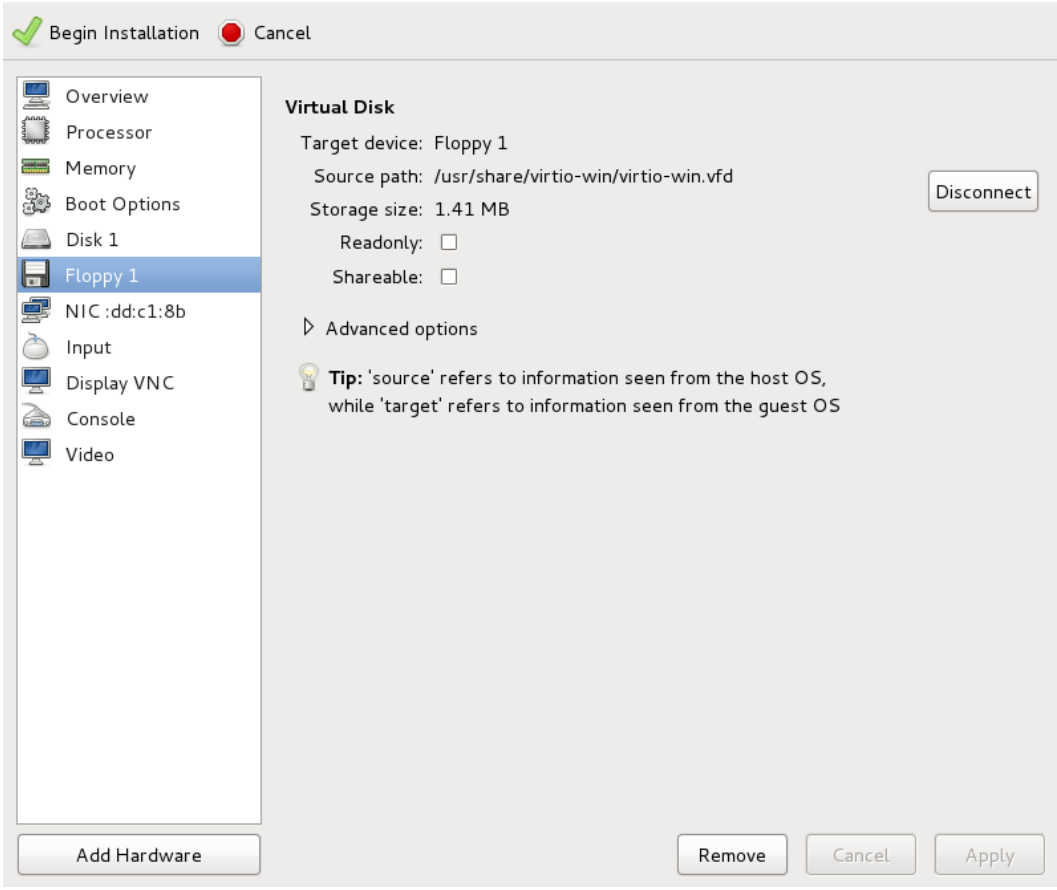



**Figure 10.20. The virtual machine hardware information window**

You have now created a removable device accessible by your virtual machine.

v. **Change the hard disk type**

To change the hard disk type from *IDE Disk* to *Virtio Disk*, we must first remove the existing hard disk, Disk 1. Select the disk and click on the **Remove** button.

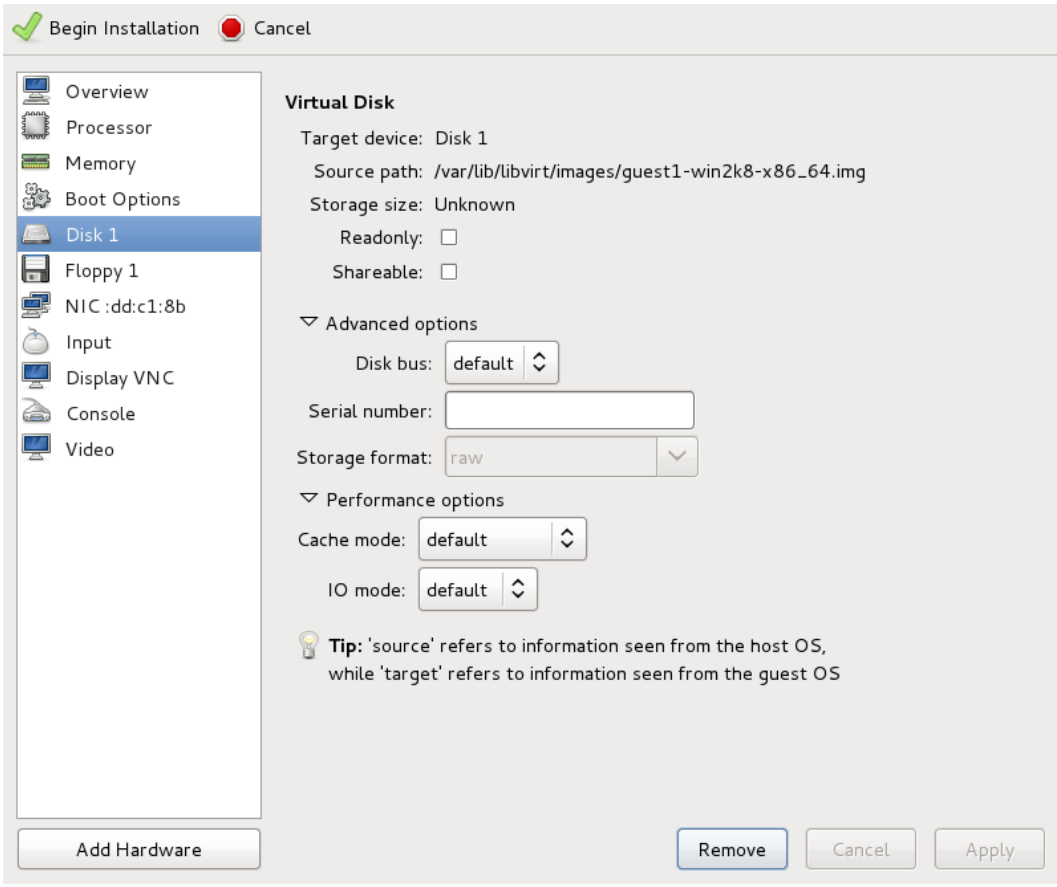



**Figure 10.21. The virtual machine hardware information window**

Add a new virtual storage device by clicking **Add Hardware**. Then, change the **Device type** from *IDE disk* to *Virtio Disk*. Click **Finish** to confirm the operation.

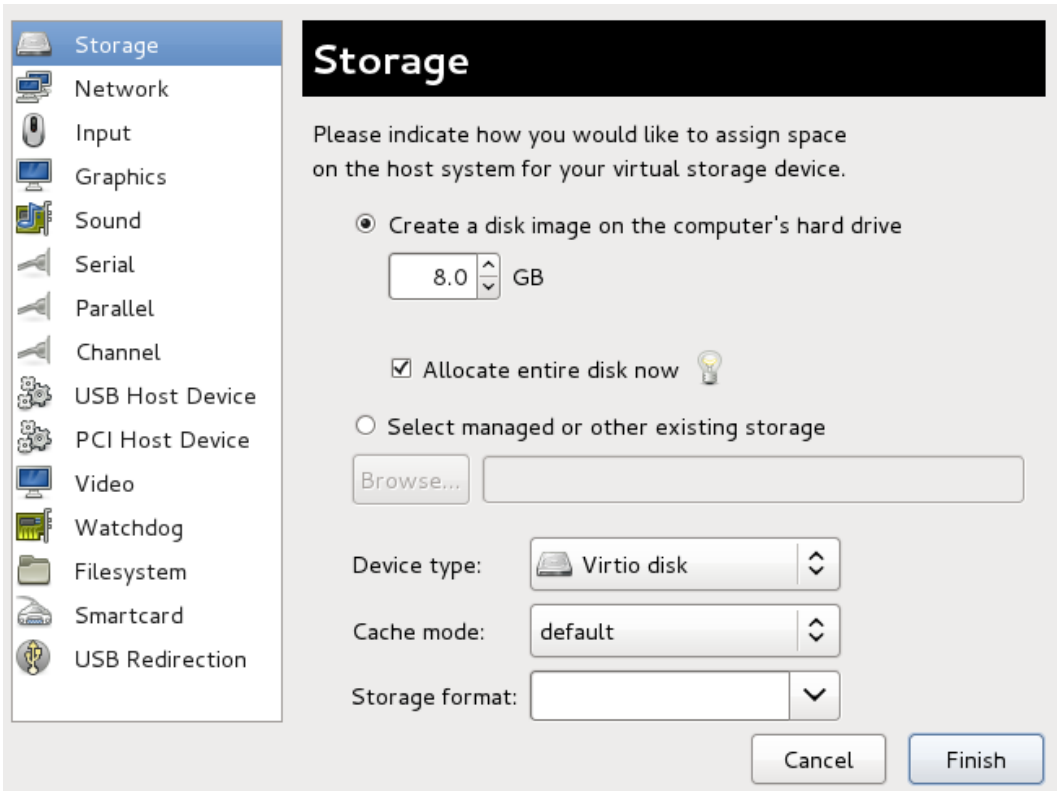


**Figure 10.22. The virtual machine hardware information window**

vi. **Ensure settings are correct**

Review the settings for *VirtIO Disk 1*.

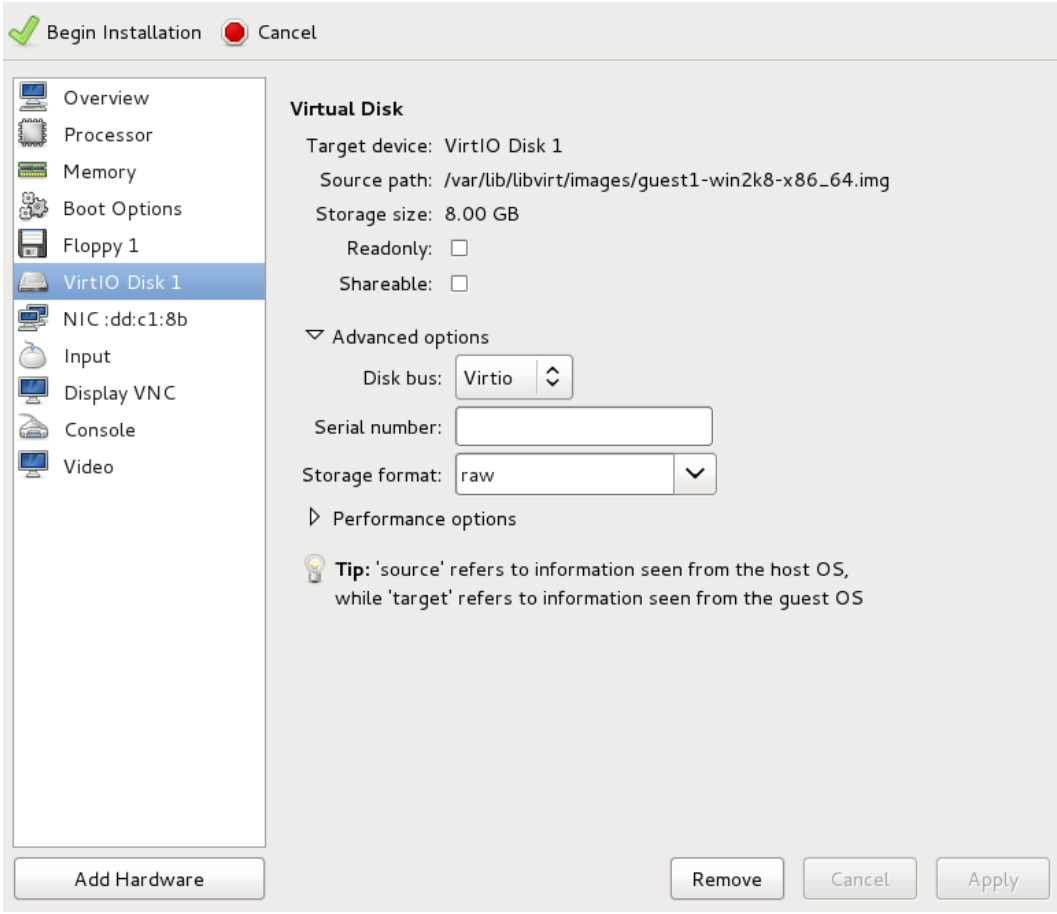



**Figure 10.23. The virtual machine hardware information window**

When you are satisfied with the configuration details, click the **`Begin Installation`** button.

You can now continue with Step 3.

c. **Create the guest virtual machine with virt-install**

Append the following parameter exactly as listed below to add the driver disk to the installation with the **`virt-install`** command:

```
--disk path=/usr/share/virtio-win/virtio-
win.vfd,device=floppy
```

**Important**

If the device you wish to add is a **disk** (that is, not a **floppy** or a **cdrom**), you will also need to add the **bus=virtio** option to the end of the ***--disk*** parameter, like so:

```
--disk path=/usr/share/virtio-win/virtio-
win.vfd,device=disk,bus=virtio
```

According to the version of Windows you are installing, append one of the following options to the **virt-install** command:

```
--os-variant win2k3
```

```
--os-variant win7
```

You can now continue with Step 3.

3. **Additional steps for driver installation**

   During the installation, additional steps are required to install drivers, depending on the type of Windows guest.

   a. **Windows Server 2003**

      Before the installation blue screen repeatedly press **F6** for third party drivers.


**Figure 10.24. The Windows Setup screen**

Press **S** to install additional device drivers.


**Figure 10.25. The Windows Setup screen**

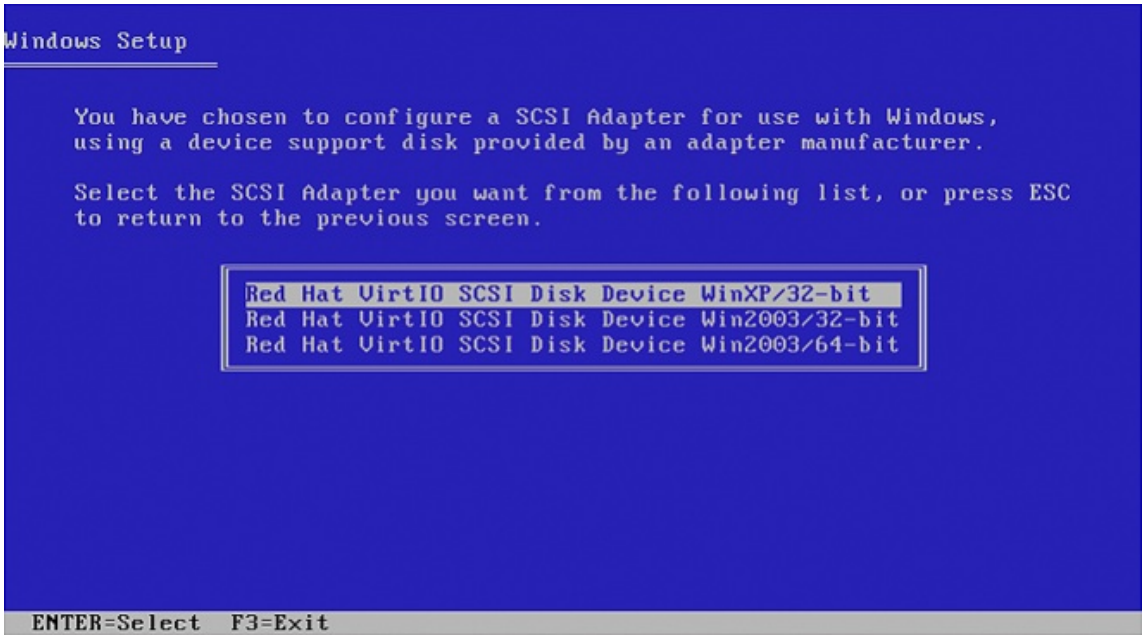



**Figure 10.26. The Windows Setup screen**

Press **Enter** to continue the installation.

b. **Windows Server 2008**

Follow the same procedure for Windows Server 2003, but when the installer prompts you for the driver, click on **Load Driver**, point the installer to **Drive A:** and pick the driver that suits your guest operating system and architecture.

## 10.4. Using the virtio Drivers with Red Hat Enterprise Linux 3.9 Guests

Virtio drivers for Red Hat Enterprise Linux 3.9 consist of five kernel modules: **virtio**, **virtio_blk**, **virtio_net**, **virtio_pci** and **virtio_ring**. All five modules must be loaded to use both the virtio block and network devices drivers.

**Important**

For Red Hat Enterprise Linux 3.9 guests, the *kmod-virtio* package is a requirement for the **virtio** module.

**Note**

To use the network device driver only, load the **virtio**, **virtio_net** and **virtio_pci** modules. To use the block device driver only, load the **virtio**, **virtio_ring**, **virtio_blk** and **virtio_pci** modules.

**Important**

The *virtio* package modifies the initrd RAM disk file in the **/boot** directory. The original initrd file is saved to **/boot/initrd-*kernel-version*.img.virtio.orig**. The original initrd file is replaced with a new initrd RAM disk containing the **virtio** driver modules. The initrd RAM disk is modified to allow the virtual machine to boot from a storage device using the virtio drivers. To use a different initrd file, you must ensure that drivers are loaded with the **sysinit** script (Loading the virtio Drivers with the sysinit Script) or when creating new initrd RAM disk (Adding the virtio Drivers to the initrd RAM Disk).

### Loading the virtio Drivers with the sysinit Script

This procedure covers loading the virtio driver modules during the boot sequence on a Red Hat Enterprise Linux 3.9 or newer guest with the **sysinit** script. Note that the guest virtual machine cannot use the virtio drivers for the default boot disk if the modules are loaded with the **sysinit** script.

The drivers must be loaded in the following order:

1. **virtio**
2. **virtio_ring**
3. **virtio_pci**
4. **virtio_blk**
5. **virtio_net**

**virtio_net** and **virtio_blk** are the only drivers whose order can be changed. If other drivers are loaded in a different order, they will not work.

Next, configure the modules. Locate the following section of the **/etc/rc.d/rc.sysinit** file.

```
if [ -f /etc/rc.modules ]; then
   /etc/rc.modules
fi
```

Append the following lines after that section:

```
if [ -f /etc/rc.modules ]; then
  /etc/rc.modules
fi

modprobe virtio
modprobe virtio_ring # Comment this out if you do not need block driver
modprobe virtio_blk  # Comment this out if you do not need block driver
modprobe virtio_net  # Comment this out if you do not need net driver
modprobe virtio_pci
```

Reboot the guest virtual machine to load the kernel modules.

### Adding the virtio Drivers to the initrd RAM Disk

This procedure covers loading the virtio driver modules with the kernel on a Red Hat Enterprise Linux 3.9 or newer guest by including the modules in the initrd RAM disk. The mkinitrd tool configures the initrd RAM disk to load the modules. Specify the additional modules with the **`--with`** parameter for the **`mkinitrd`** command. Append following set of parameters, in the exact order, when using the **`mkinitrd`** command to create a custom initrd RAM disk:

```
--with virtio --with virtio_ring --with virtio_blk --with virtio_net --
with virtio_pci
```

### AMD64 and Intel 64 Issues

Use the **`x86_64`** version of the *virtio* package for AMD64 systems.

Use the **`ia32e`** version of the *virtio* package for Intel 64 systems. Using the **`x86_64`** version of the *virtio* may cause a '**`Unresolved symbol`**' error during the boot sequence on Intel 64 systems.

### Network Performance Issues

If you experience low performance with the virtio network drivers, verify the setting for the GSO and TSO features on the host system. The virtio network drivers require that the GSO and TSO options are disabled for optimal performance.

Verify the status of the GSO and TSO settings, use the command on the host (replacing *interface* with the network interface used by the guest):

```
# ethtool -k interface
```

Disable the GSO and TSO options with the following commands on the host:

```
# ethtool -K interface gso off
# ethtool -K interface tso off
```

### virtio Driver Swap Partition issue

After activating the virtio block device driver the swap partition may not be available. This issue is may be caused by a change in disk device name. To fix this issue, open the **/etc/fstab** file and locate the lines containing swap partitions, for example:

```
/dev/hda3       swap                    swap defaults 0 0
```

The virtio drivers use the **/dev/vd*** naming convention, not the **/dev/hd*** naming convention. To resolve this issue modify the incorrect swap entries in the **/etc/fstab** file to use the **/dev/vd*** convention, for the example above:

```
/dev/vda3 swap                    swap defaults 0 0
```

Save the changes and reboot the guest virtual machine. The virtual machine should now have swap partitions.

## 10.5. Using KVM virtio Drivers for Existing Devices

You can modify an existing hard disk device attached to the guest to use the **virtio** driver instead of the virtualized IDE driver. The example shown in this section edits libvirt configuration files. Note that the guest virtual machine does not need to be shut down to perform these steps, however the change will not be applied until the guest is completely shut down and rebooted.

**Procedure 10.4. Using KVM virtio drivers for existing devices**

1. Ensure that you have installed the appropriate driver (**viostor**), as described in Section 10.1, “Installing the KVM Windows virtio Drivers”, before continuing with this procedure.

2. Run the **virsh edit <guestname>** command as root to edit the XML configuration file for your device. For example, **virsh edit guest1**. The configuration files are located in **/etc/libvirt/qemu**.

3. Below is a file-based block device using the virtualized IDE driver. This is a typical entry for a virtual machine not using the virtio drivers.

   ```
   <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/disk1.img'/>
      <target dev='hda' bus='ide'/>
   </disk>
   ```

4. Change the entry to use the virtio device by modifying the **bus=** entry to **virtio**. Note that if the disk was previously IDE it will have a target similar to hda, hdb, or hdc and so on. When changing to **bus=virtio** the target needs to be changed to vda, vdb, or vdc accordingly.

   ```
   <disk type='file' device='disk'>
      <source file='/var/lib/libvirt/images/disk1.img'/>
      <target dev='vda' bus='virtio'/>
   </disk>
   ```

5. Remove the **address** tag inside the **disk** tags. This must be done for this procedure to work. Libvirt will regenerate the **address** tag appropriately the next time the virtual machine is started.

Alternatively, **virt-manager**, **virsh attach-disk** or **virsh attach-interface** can add a new device using the virtio drivers.

Refer to the libvirt website for more details on using Virtio: http://www.linux-kvm.org/page/Virtio

## 10.6. Using KVM virtio Drivers for New Devices

This procedure covers creating new devices using the KVM virtio drivers with **virt-manager**.

Alternatively, the **virsh attach-disk** or **virsh attach-interface** commands can be used to attach devices using the virtio drivers.

**Important**

Ensure the drivers have been installed on the Windows guest before proceeding to install new devices. If the drivers are unavailable the device will not be recognized and will not work.

**Procedure 10.5. Adding a storage device using the virtio storage driver**

1. Open the guest virtual machine by double clicking on the name of the guest in **virt-manager**.
2. Open the **Show virtual hardware details** tab by clicking the **lightbulb** button.


**Figure 10.27. The Show virtual hardware details tab**

3. In the **Show virtual hardware details** tab, click on the **Add Hardware** button.
4. **Select hardware type**

   Select **Storage** as the **Hardware type**.

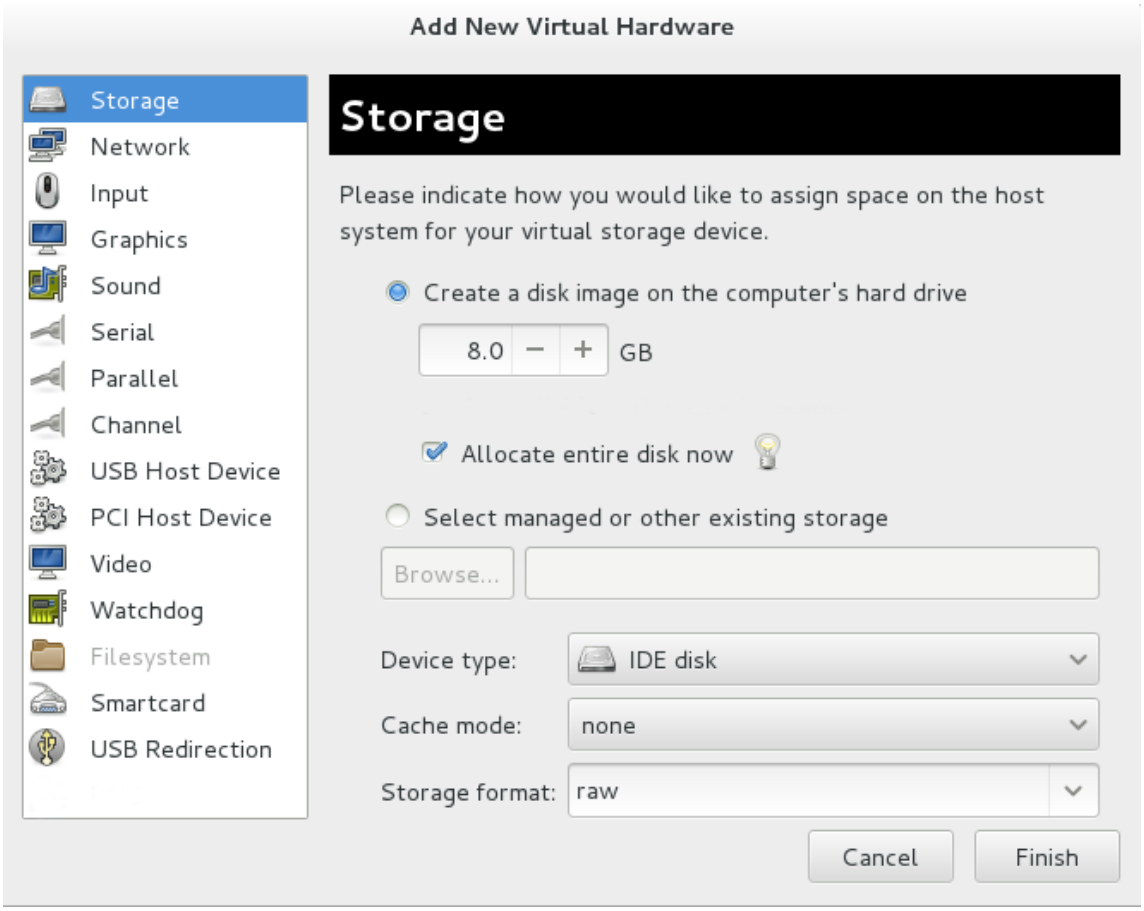



**Figure 10.28. The Add new virtual hardware wizard**

5. **Select the storage device and driver**

   Create a new disk image or select a storage pool volume.

   Set the **`Device type`** to **`Virtio disk`** to use the virtio drivers.

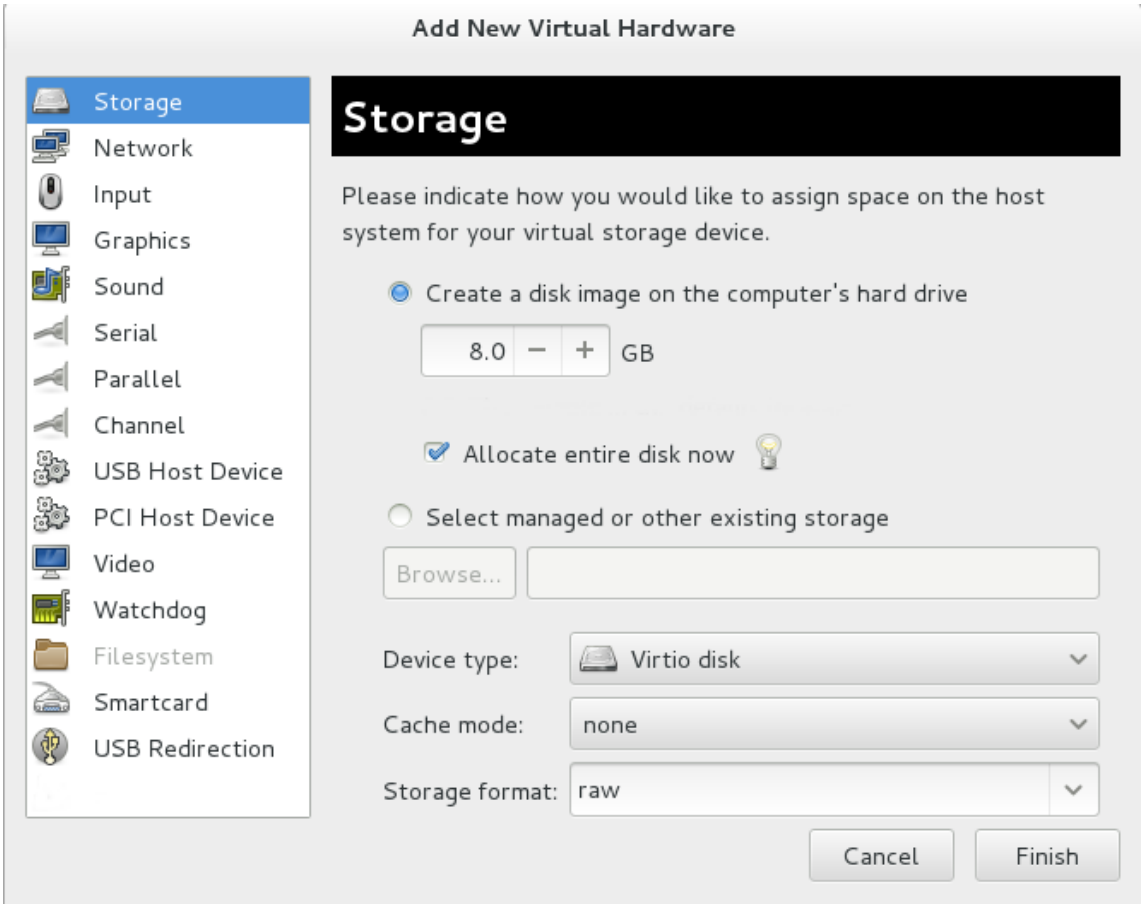



**Figure 10.29. The Add new virtual hardware wizard**

Click **Finish** to complete the procedure.

**Procedure 10.6. Adding a network device using the virtio network driver**

1. Open the guest virtual machine by double clicking on the name of the guest in **virt-manager**.
2. Open the **Show virtual hardware details** tab by clicking the **lightbulb** button.

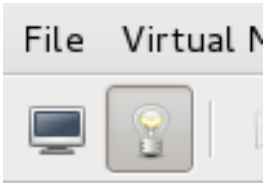



**Figure 10.30. The Show virtual hardware details tab**

3. In the **Show virtual hardware details** tab, click on the **Add Hardware** button.
4. **Select hardware type**

   Select **Network** as the **Hardware type**.

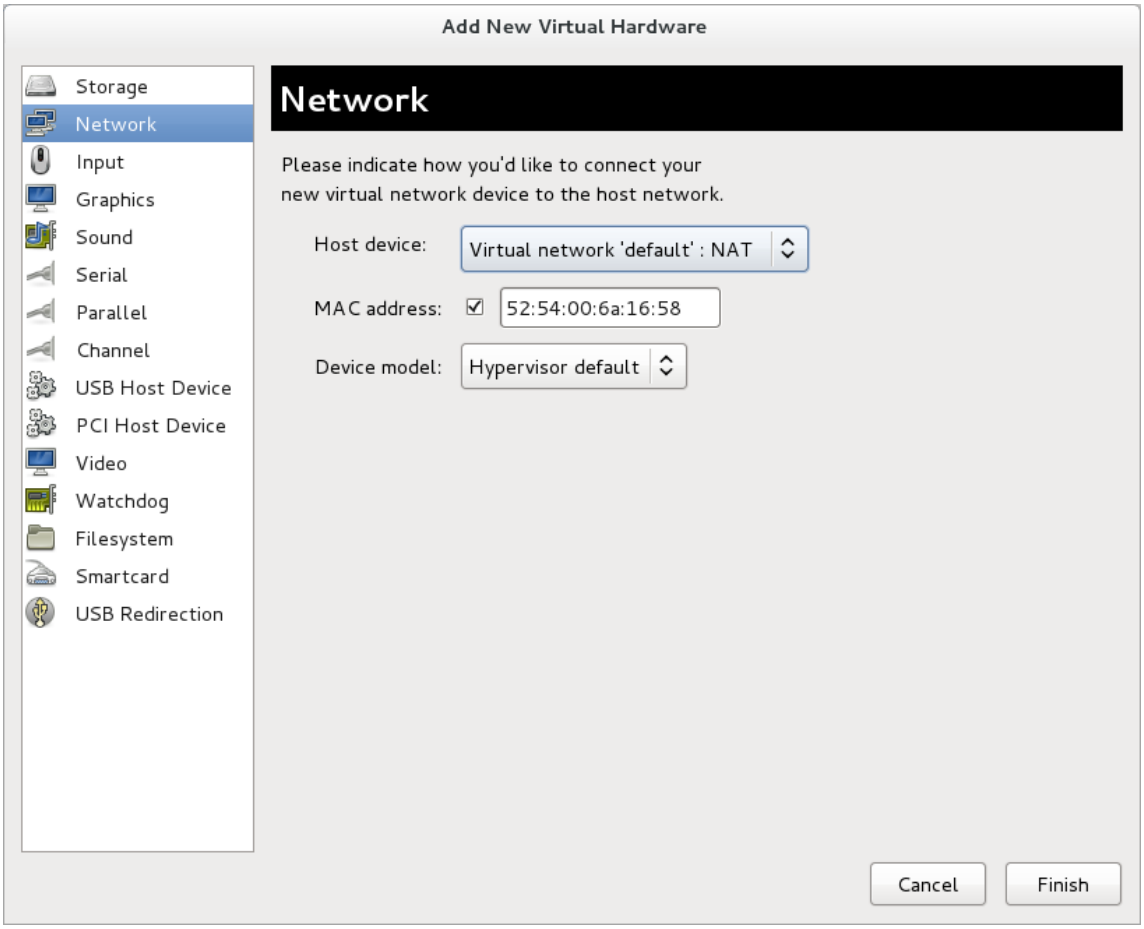



**Figure 10.31. The Add new virtual hardware wizard**

5. **Select the network device and driver**

   Set the **`Device model`** to **`virtio`** to use the virtio drivers. Choose the desired **`Host device`**.

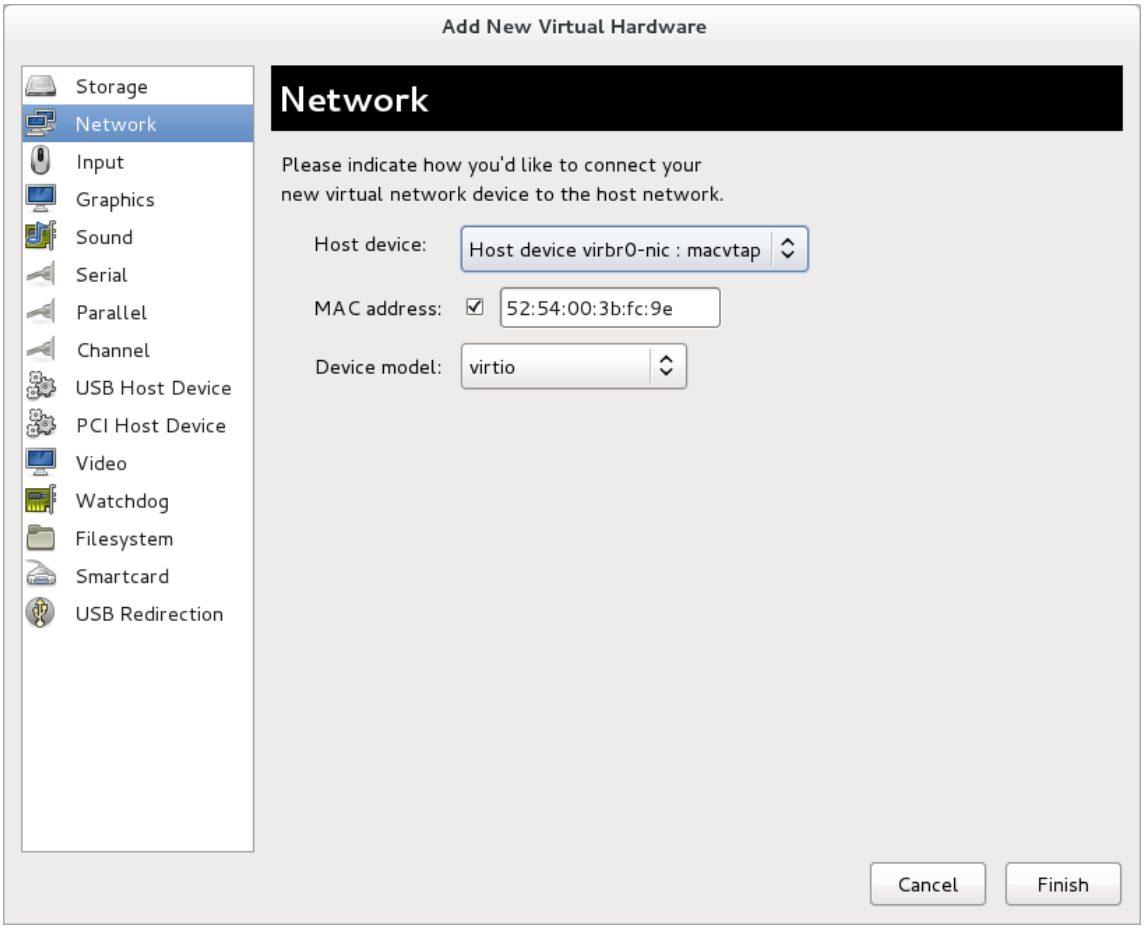



**Figure 10.32. The Add new virtual hardware wizard**

Click **Finish** to complete the procedure.

Once all new devices are added, reboot the virtual machine. Windows virtual machines may not recognize the devices until the guest is rebooted.

# Chapter 11. Network Configuration

This chapter provides an introduction to the common networking configurations used by libvirt based guest virtual machines. For additional information, consult the libvirt network architecture documentation: http://libvirt.org/intro.html.

Red Hat Enterprise Linux 6 supports the following networking setups for virtualization:

- virtual networks using Network Address Translation (*NAT*)
- directly allocated physical devices using PCI device assignment
- directly allocated virtual functions using PCIe SR-IOV
- bridged networks

You must enable NAT, network bridging or directly assign a PCI device to allow external hosts access to network services on guest virtual machines.

## 11.1. Network Address Translation (NAT) with libvirt

One of the most common methods for sharing network connections is to use Network Address Translation (NAT) forwarding (also known as virtual networks).

### Host Configuration

Every standard **libvirt** installation provides NAT-based connectivity to virtual machines as the default virtual network. Verify that it is available with the **virsh net-list --all** command.

```
# virsh net-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
```

If it is missing, the example XML configuration file can be reloaded and activated:

```
# virsh net-define /usr/share/libvirt/networks/default.xml
```

The default network is defined from **/usr/share/libvirt/networks/default.xml**

Mark the default network to automatically start:

```
# virsh net-autostart default
Network default marked as autostarted
```

Start the default network:

```
# virsh net-start default
Network default started
```

Once the **libvirt** default network is running, you will see an isolated bridge device. This device does *not* have any physical interfaces added. The new device uses NAT and IP forwarding to connect to the physical network. Do not add new interfaces.

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.000000000000       yes
```

**libvirt** adds **iptables** rules which allow traffic to and from guest virtual machines attached to the **virbr0** device in the **INPUT**, **FORWARD**, **OUTPUT** and **POSTROUTING** chains. **libvirt** then attempts to enable the **ip_forward** parameter. Some other applications may disable **ip_forward**, so the best option is to add the following to **/etc/sysctl.conf**.

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

### Guest Virtual Machine Configuration

Once the host configuration is complete, a guest virtual machine can be connected to the virtual network based on its name. To connect a guest to the 'default' virtual network, the following could be used in the XML configuration file (such as **/etc/libvirtd/qemu/myguest.xml**) for the guest:

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

**Note**

Defining a MAC address is optional. If you do not define one, a MAC address is automatically generated and used as the MAC address of the bridge device used by the network. Manually setting the MAC address may be useful to maintain consistency or easy reference throughout your environment, or to avoid the very small chance of a conflict.

```
<interface type='network'>
  <source network='default'/>
  <mac address='00:16:3e:1a:b3:4a'/>
</interface>
```

## 11.2. Disabling vhost-net

The **vhost-net** module is a kernel-level backend for virtio networking that reduces virtualization overhead by moving virtio packet processing tasks out of user space (the qemu process) and into the kernel (the **vhost-net** driver). vhost-net is only available for virtio network interfaces. If the vhost-net kernel module is loaded, it is enabled by default for all virtio interfaces, but can be disabled in the interface configuration in the case that a particular workload experiences a degradation in performance when vhost-net is in use.

Specifically, when UDP traffic is sent from a host machine to a guest virtual machine on that host, performance degradation can occur if the guest virtual machine processes incoming data at a rate slower than the host machine sends it. In this situation, enabling **vhost-net** causes the UDP socket's receive buffer to overflow more quickly, which results in greater packet loss. It is therefore better to disable **vhost-net** in this situation to slow the traffic, and improve overall performance.

To disable **vhost-net**, edit the **<interface>** sub-element in the guest virtual machine's XML configuration file and define the network as follows:

```
<interface type="network">
   ...
   <model type="virtio"/>
   <driver name="qemu"/>
   ...
</interface>
```

Setting the driver name to **qemu** forces packet processing into qemu user space, effectively disabling vhost-net for that interface.

## 11.3. Bridged Networking with libvirt

Bridged networking (also known as physical device sharing) is used to dedicate a physical device to a virtual machine. Bridging is often used for more advanced setups and on servers with multiple network interfaces.

To create a bridge (**br0**) based on the **eth0** interface, execute the following command on the host:

```
# virsh iface-bridge eth0 br0
```

**Important**

NetworkManager does not support bridging. NetworkManager must be disabled to use networking with the network scripts (located in the **/etc/sysconfig/network-scripts/** directory).

```
# chkconfig NetworkManager off
# chkconfig network on
# service NetworkManager stop
# service network start
```

If you do not want to disable **NetworkManager** entirely, add "***NM_CONTROLLED=no***" to the **ifcfg-*** network script being used for the bridge.

# Chapter 12. PCI Device Assignment

Red Hat Enterprise Linux 6 exposes three classes of device to its virtual machines:

- *Emulated devices* are purely virtual devices that mimic real hardware, allowing unmodified guest operating systems to work with them using their standard in-box drivers.
- *Virtio devices* are purely virtual devices designed to work optimally in a virtual machine. Virtio devices are similar to emulated devices, however, non-Linux virtual machines do not include the drivers they require by default. Virtualization management software like the Virtual Machine Manager (**virt-manager**) and the Red Hat Enterprise Virtualization Hypervisor install these drivers automatically for supported non-Linux guest operating systems.
- *Assigned devices* are physical devices that are exposed to the virtual machine. This method is also known as 'passthrough'. Device assignment allows virtual machines exclusive access to PCI devices for a range of tasks, and allows PCI devices to appear and behave as if they were physically attached to the guest operating system.

  Device assignment is supported on PCI Express devices, except graphics cards. Parallel PCI devices may be supported as assigned devices, but they have severe limitations due to security and system configuration conflicts.

Red Hat Enterprise Linux 6 supports 32 PCI device slots per virtual machine, and 8 PCI functions per device slot. This gives a theoretical maximum of 256 configurable PCI functions per guest.

However, this theoretical maximum is subject to the following limitations:

- Each virtual machine supports a maximum of 8 assigned device functions.
- 4 PCI device slots are configured with 5 emulated devices (two devices are in slot 1) by default. However, users can explicitly remove 2 of the emulated devices that are configured by default if the guest operating system does not require them for operation (the video adapter device in slot 2; and the memory balloon driver device in the lowest available slot, usually slot 3). This gives users a supported functional maximum of 30 PCI device slots per virtual machine.

Red Hat Enterprise Linux 6.0 and newer supports hot plugging assigned PCI devices into virtual machines. However, PCI device hot plugging operates at the slot level and therefore does not support multi-function PCI devices. Multi-function PCI devices are recommended for static device configuration only.

**Note**

Red Hat Enterprise Linux 6.0 limited guest operating system driver access to a device's standard and extended configuration space. Limitations that were present in Red Hat Enterprise Linux 6.0 were significantly reduced in Red Hat Enterprise Linux 6.1, and enable a much larger set of PCI Express devices to be successfully assigned to KVM guests.

Secure device assignment also requires interrupt remapping support. If a platform does not support interrupt remapping, device assignment will fail. To use device assignment without interrupt remapping support in a development environment, set the ***allow_unsafe_assigned_interrupts*** KVM module parameter to **1**.

PCI device assignment is only available on hardware platforms supporting either Intel VT-d or AMD IOMMU. These Intel VT-d or AMD IOMMU specifications must be enabled in BIOS for PCI device assignment to function.

**Procedure 12.1. Preparing an Intel system for PCI device assignment**

1. **Enable the Intel VT-d specifications**

   The Intel VT-d specifications provide hardware support for directly assigning a physical device to a virtual machine. These specifications are required to use PCI device assignment with Red Hat Enterprise Linux.

   The Intel VT-d specifications must be enabled in the BIOS. Some system manufacturers disable these specifications by default. The terms used to refer to these specifications can differ between manufacturers; consult your system manufacturer's documentation for the appropriate terms.

2. **Activate Intel VT-d in the kernel**

   Activate Intel VT-d in the kernel by adding the ***intel_iommu=on*** parameter to the kernel line in the **/boot/grub/grub.conf** file.

   The example below is a modified **grub.conf** file with Intel VT-d activated.

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.32-330.x86_645)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-330.x86_64 ro
root=/dev/VolGroup00/LogVol00 rhgb quiet intel_iommu=on
        initrd /initrd-2.6.32-330.x86_64.img
```

3. **Ready to use**

   Reboot the system to enable the changes. Your system is now capable of PCI device assignment.

**Procedure 12.2. Preparing an AMD system for PCI device assignment**

1. **Enable the AMD IOMMU specifications**

   The AMD IOMMU specifications are required to use PCI device assignment in Red Hat Enterprise Linux. These specifications must be enabled in the BIOS. Some system manufacturers disable these specifications by default.

2. **Enable IOMMU kernel support**

   Append ***amd_iommu=on*** to the kernel command line in **/boot/grub/grub.conf** so that AMD IOMMU specifications are enabled at boot.

3. **Ready to use**

   Reboot the system to enable the changes. Your system is now capable of PCI device assignment.

## 12.1. Assigning a PCI Device with virsh

These steps cover assigning a PCI device to a virtual machine on a KVM hypervisor.

This example uses a PCIe network controller with the PCI identifier code, **pci_0000_01_00_0**, and a fully virtualized guest machine named *guest1-rhel6-64*.

**Procedure 12.3. Assigning a PCI device to a guest virtual machine with virsh**

1. **Identify the device**

   First, identify the PCI device designated for device assignment to the virtual machine. Use the **lspci** command to list the available PCI devices. You can refine the output of **lspci** with **grep**.

   This example uses the Ethernet controller highlighted in the following output:

   ```
   # lspci | grep Ethernet
   00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit
   Network Connection
   01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit
   Network Connection (rev 01)
   01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit
   Network Connection (rev 01)
   ```

   This Ethernet controller is shown with the short identifier **00:19.0**. We need to find out the full identifier used by **virsh** in order to assign this PCI device to a virtual machine.

   To do so, combine the **virsh nodedev-list** command with the **grep** command to list all devices of a particular type (**pci**) that are attached to the host machine. Then look at the output for the string that maps to the short identifier of the device you wish to use.

   This example highlights the string that maps to the Ethernet controller with the short identifier **00:19.0**. Note that the **:** and **.** characters are replaced with underscores in the full identifier.

   ```
   # virsh nodedev-list --cap pci
   pci_0000_00_00_0
   pci_0000_00_01_0
   pci_0000_00_03_0
   pci_0000_00_07_0
   pci_0000_00_10_0
   pci_0000_00_10_1
   pci_0000_00_14_0
   pci_0000_00_14_1
   pci_0000_00_14_2
   pci_0000_00_14_3
   pci_0000_00_19_0
   pci_0000_00_1a_0
   pci_0000_00_1a_1
   pci_0000_00_1a_2
   pci_0000_00_1a_7
   pci_0000_00_1b_0
   pci_0000_00_1c_0
   pci_0000_00_1c_1
   pci_0000_00_1c_4
   pci_0000_00_1d_0
   pci_0000_00_1d_1
   ```

```
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

Record the PCI device number that maps to the device you want to use; this is required in other steps.

2. **Review device information**

   Information on the domain, bus, and function are available from output of the **`virsh nodedev-dumpxml`** command:

```
virsh nodedev-dumpxml pci_0000_00_19_0
<device>
  <name>pci_0000_00_19_0</name>
  <parent>computer</parent>
  <driver>
    <name>e1000e</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>0</bus>
    <slot>25</slot>
    <function>0</function>
    <product id='0x1502'>82579LM Gigabit Network
Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
    </capability>
  </capability>
</device>
```

3. **Determine required configuration details**

   Refer to the output from the **`virsh nodedev-dumpxml pci_0000_00_19_0`** command for the values required for the configuration file.

   Optionally, convert slot and function values to hexadecimal values (from decimal) to get the PCI bus addresses. Append "0x" to the beginning of the output to tell the computer that the value is a hexadecimal number.

   The example device has the following values: bus = 0, slot = 25 and function = 0. The decimal configuration uses those three values:

```
bus='0'
slot='25'
function='0'
```

If you want to convert to hexadecimal values, you can use the **printf** utility to convert from decimal values, as shown in the following example:

```
$ printf %x 0
0
$ printf %x 25
19
$ printf %x 0
0
```

The example device would use the following hexadecimal values in the configuration file:

```
bus='0x0'
slot='0x19'
function='0x0'
```

4. **Add configuration details**

   Run **virsh edit**, specifying the virtual machine name, and add a device entry in the **<source>** section to assign the PCI device to the guest virtual machine.

```
# virsh edit guest1-rhel6-64
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
     <address domain='0x0' bus='0x0' slot='0x19' function='0x0'/>
  </source>
</hostdev>
```

   Alternately, run **virsh attach-device**, specifying the virtual machine name and the guest's XML file:

```
virsh attach-device guest1-rhel6-64 file.xml
```

5. **Allow device management**

   Set an SELinux boolean to allow the management of the PCI device from the virtual machine:

```
# setsebool -P virt_use_sysfs 1
```

6. **Start the virtual machine**

```
# virsh start guest1-rhel6-64
```

The PCI device should now be successfully assigned to the virtual machine, and accessible to the guest operating system.

## 12.2. Assigning a PCI Device with virt-manager

PCI devices can be added to guest virtual machines using the graphical **virt-manager** tool. The following procedure adds a Gigabit Ethernet controller to a guest virtual machine.

**Procedure 12.4. Assigning a PCI device to a guest virtual machine using virt-manager**

1. **Open the hardware settings**

   Open the guest virtual machine and click the **Add Hardware** button to add a new device to the virtual machine.

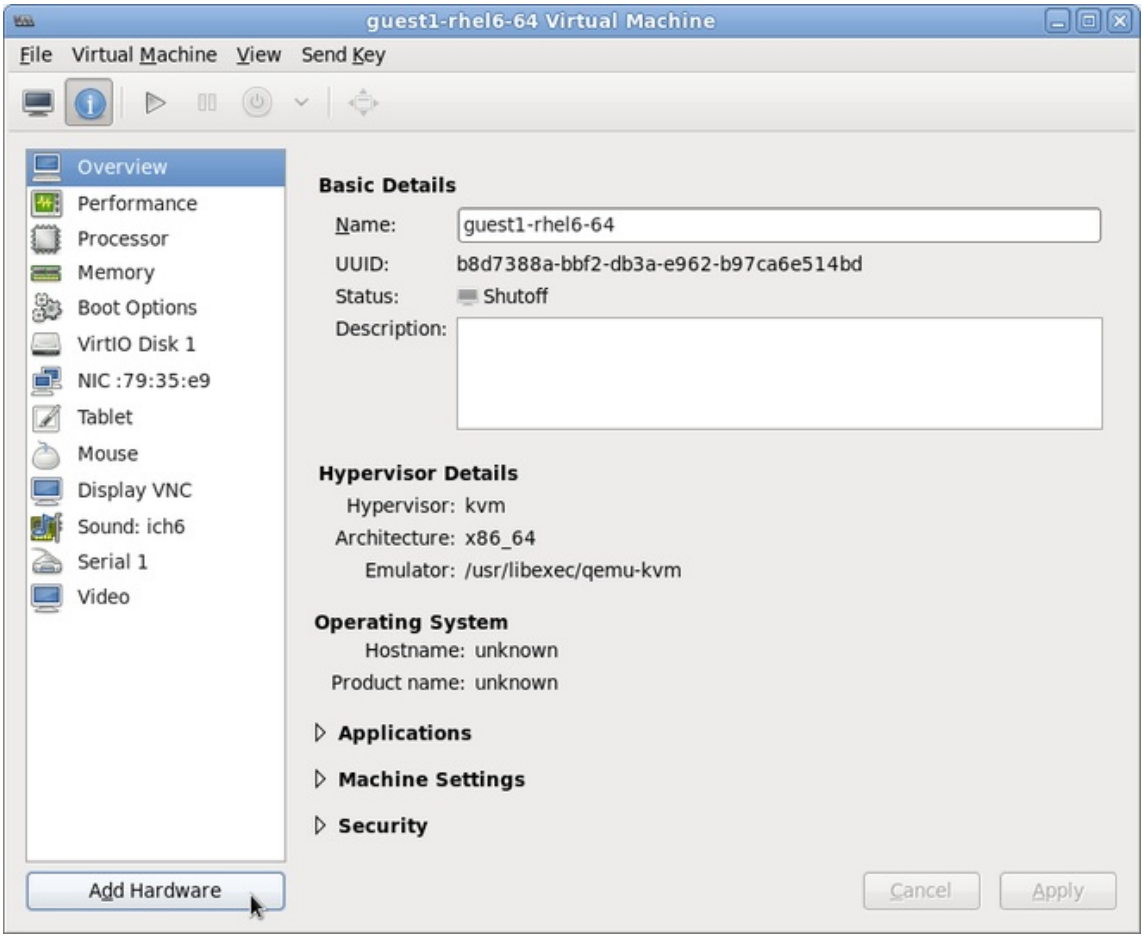



**Figure 12.1. The virtual machine hardware information window**

2. **Select a PCI device**

   Select **PCI Host Device** from the **Hardware** list on the left.

   Select an unused PCI device. Note that selecting PCI devices presently in use on the host causes errors. In this example, a spare 82576 network device is used. Click **Finish** to complete setup.

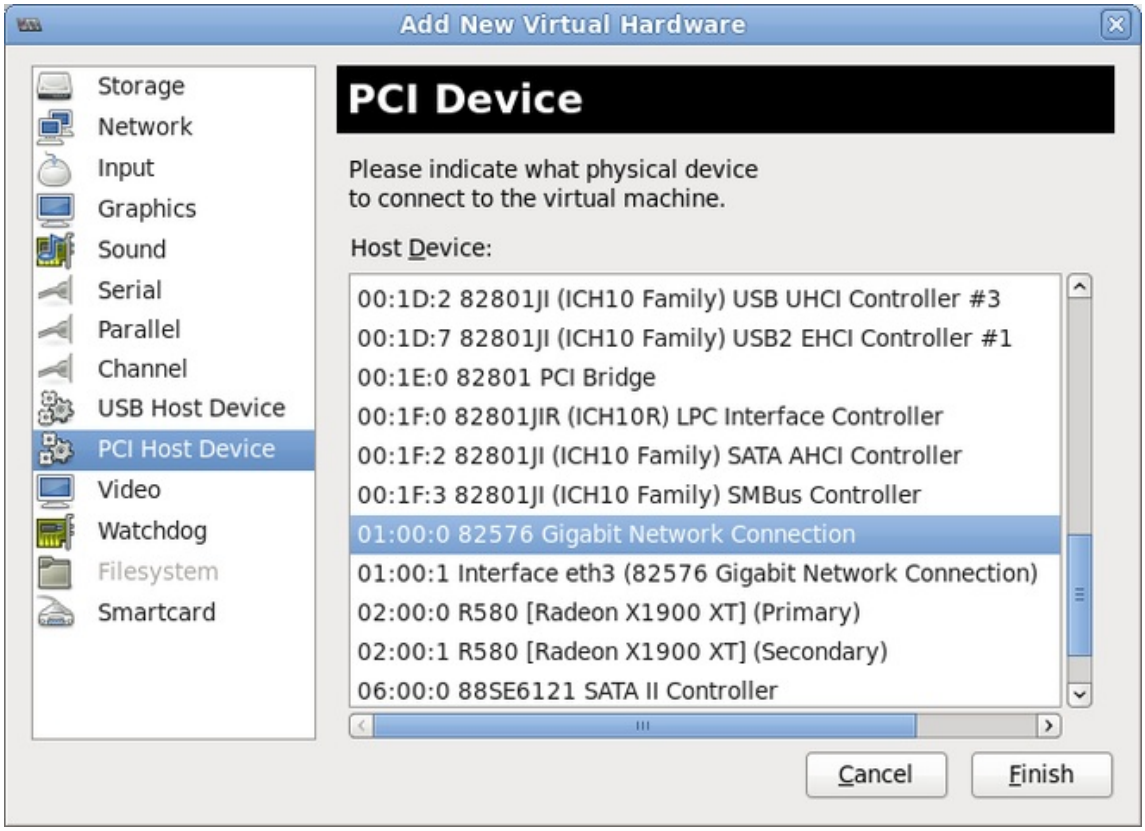



**Figure 12.2. The Add new virtual hardware wizard**

3. **Add the new device**

   The setup is complete and the guest virtual machine now has direct access to the PCI device.

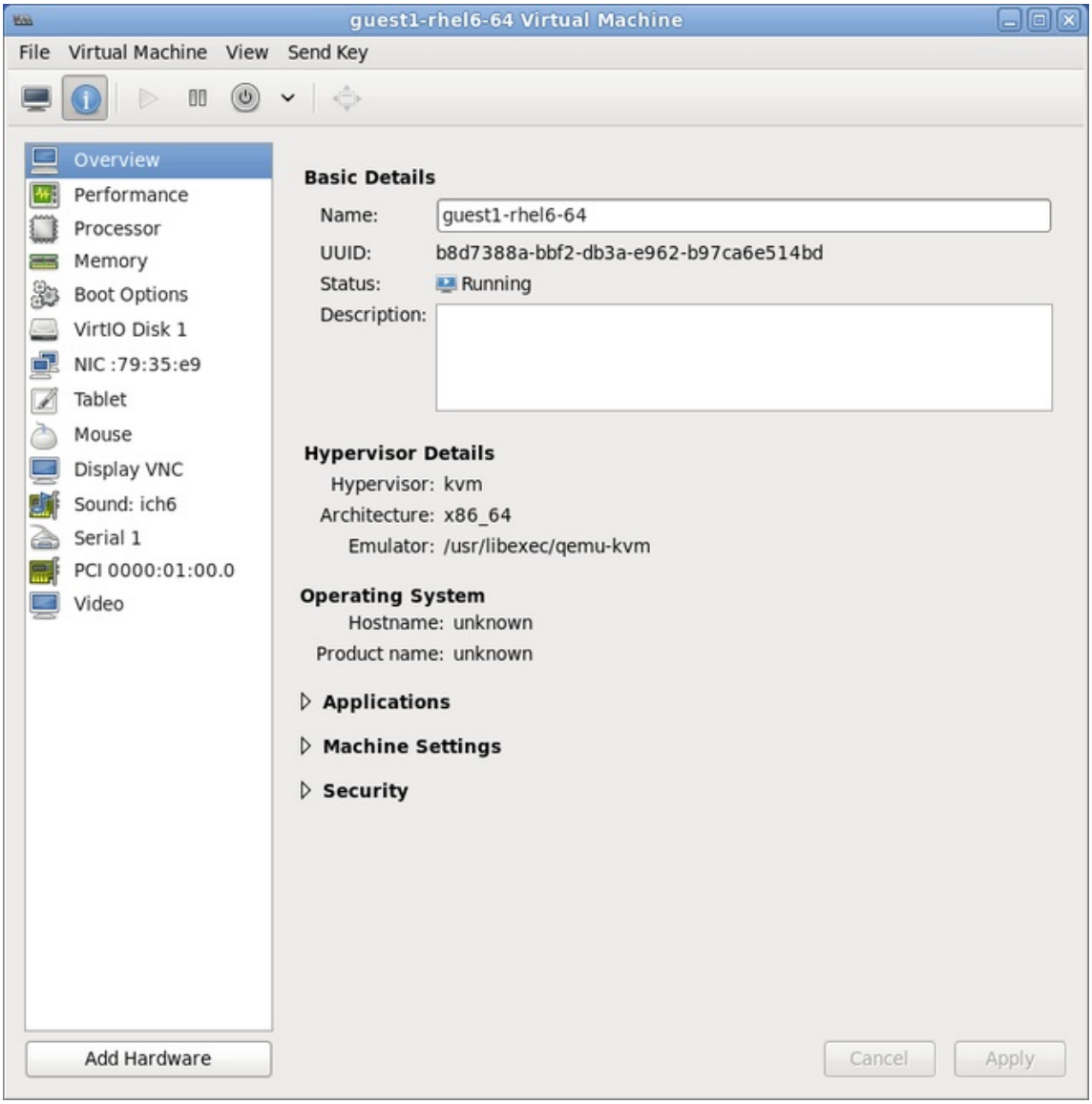



**Figure 12.3. The virtual machine hardware information window**

## 12.3. Assigning a PCI Device with virt-install

To use **virt-install** to assign a PCI device, use the ***--host-device*** parameter.

**Procedure 12.5. Assigning a PCI device to a virtual machine with virt-install**

1. **Identify the device**

   Identify the PCI device designated for device assignment to the guest virtual machine.

```
# lspci | grep Ethernet
00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit
Network Connection
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit
Network Connection (rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit
Network Connection (rev 01)
```

   The **virsh nodedev-list** command lists all devices attached to the system, and identifies each PCI device with a string. To limit output to only PCI devices, run the following command:

```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

Record the PCI device number; the number is needed in other steps.

Information on the domain, bus and function are available from output of the **`virsh nodedev-dumpxml`** command:

```
# virsh nodedev-dumpxml pci_0000_01_00_0
<device>
  <name>pci_0000_01_00_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>1</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
```

```
    <capability type='virt_functions'>
    </capability>
  </capability>
</device>
```

2. **Add the device**

   Use the PCI identifier output from the **virsh nodedev** command as the value for the ***--host-device*** parameter.

```
virt-install \
--name=guest1-rhel6-64 \
--disk path=/var/lib/libvirt/images/guest1-rhel6-64.img,size=8 \
--nonsparse --graphics spice \
--vcpus=2 --ram=2048 \
--location=http://example1.com/installation_tree/RHEL6.1-Server-
x86_64/os \
--nonetworks \
--os-type=linux \
--os-variant=rhel6 \
--host-device=pci_0000_01_00_0
```

3. **Complete the installation**

   Complete the guest installation. The PCI device should be attached to the guest.

## 12.4. Detaching an Assigned PCI Device

When a host PCI device has been assigned to a guest machine, the host can no longer use the device. Read this section to learn how to detach the device from the guest with **virsh** or **virt-manager** so it is available for host use.

**Procedure 12.6. Detaching a PCI device from a guest with virsh**

1. **Detach the device**

   Use the following command to detach the PCI device from the guest by removing it in the guest's XML file:

```
# virsh detach-device name_of_guest file.xml
```

2. **Re-attach the device to the host (optional)**

   If the device is in ***managed*** mode, skip this step. The device will be returned to the host automatically.

   If the device is not using ***managed*** mode, use the following command to re-attach the PCI device to the host machine:

```
# virsh nodedev-reattach device
```

   For example, to re-attach the **pci_0000_01_00_0** device to the host:

```
virsh nodedev-reattach pci_0000_01_00_0
```

The device is now available for host use.

**Procedure 12.7. Detaching a PCI Device from a guest with virt-manager**

1. **Open the virtual hardware details screen**

   In **virt-manager**, double-click on the virtual machine that contains the device. Select the **Show virtual hardware details** button to display a list of virtual hardware.


**Figure 12.4. The virtual hardware details button**

2. **Select and remove the device**

   Select the PCI device to be detached from the list of virtual devices in the left panel.

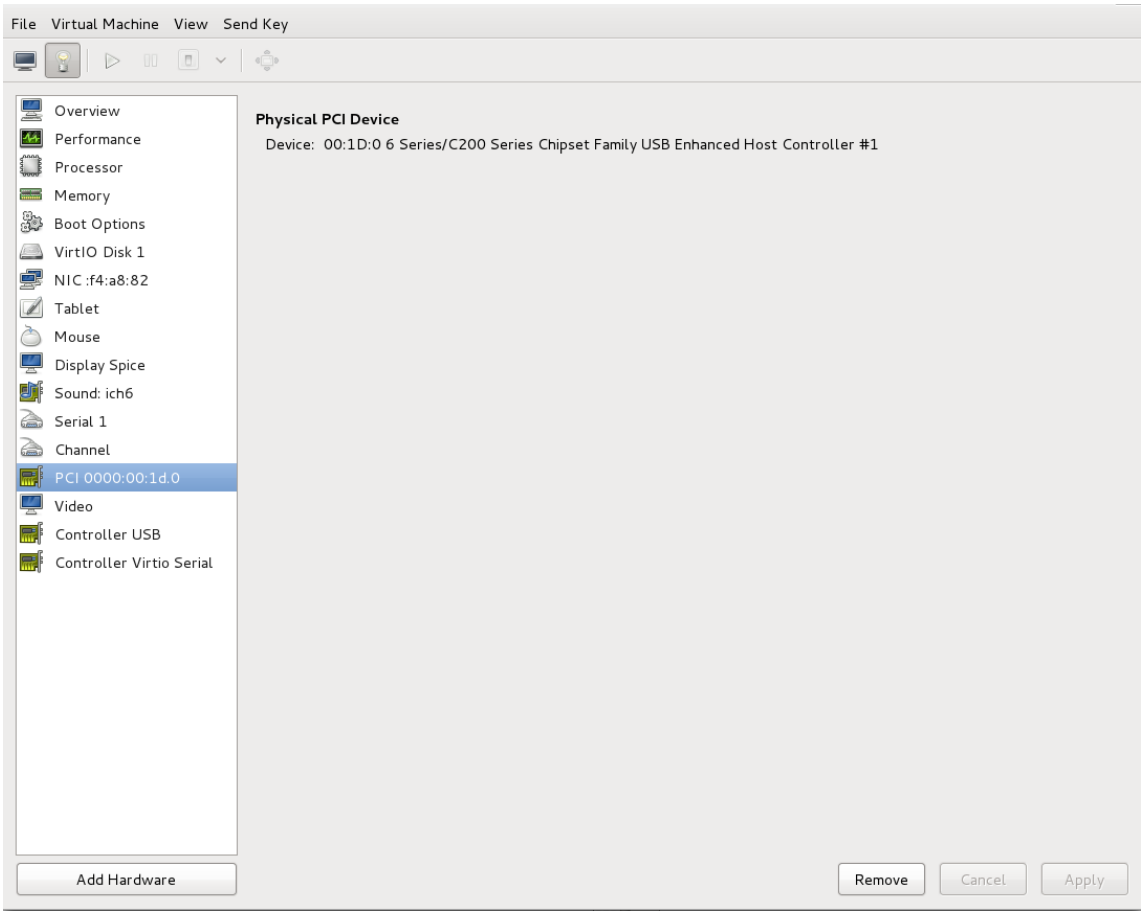



**Figure 12.5. Selecting the PCI device to be detached**

Click the **Remove** button to confirm. The device is now available for host use.

# Chapter 13. SR-IOV

## 13.1. Introduction

Developed by the PCI-SIG (PCI Special Interest Group), the Single Root I/O Virtualization (SR-IOV) specification is a standard for a type of PCI device assignment that can share a single device to multiple virtual machines. SR-IOV improves device performance for virtual machines.

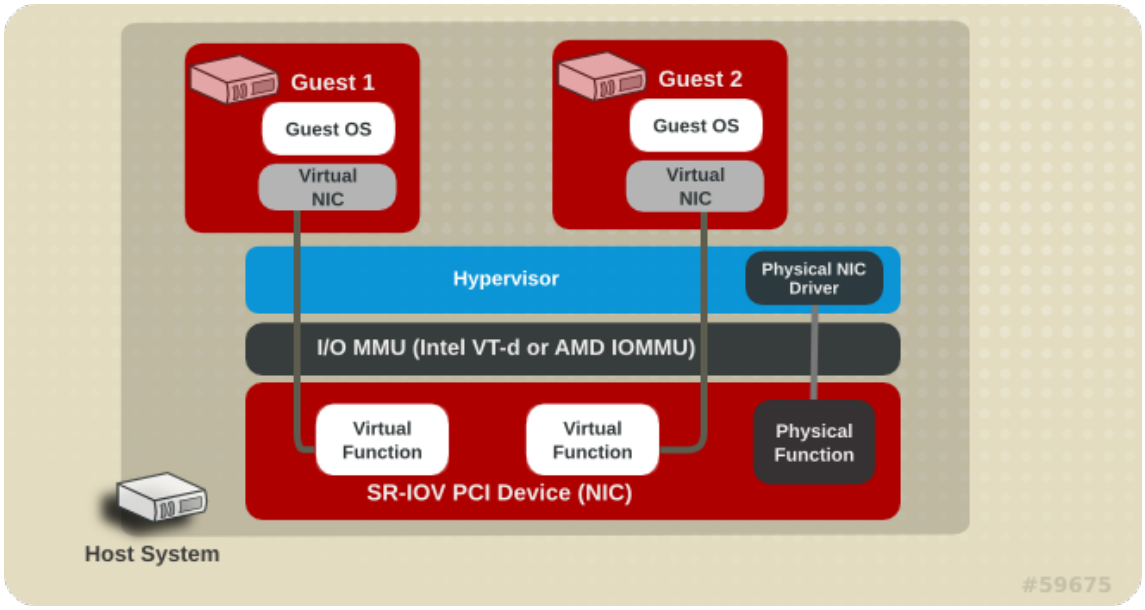



**Figure 13.1. How SR-IOV works**

SR-IOV enables a Single Root Function (for example, a single Ethernet port), to appear as multiple, separate, physical devices. A physical device with SR-IOV capabilities can be configured to appear in the PCI configuration space as multiple functions. Each device has its own configuration space complete with Base Address Registers (BARs).

SR-IOV uses two PCI functions:

- Physical Functions (PFs) are full PCIe devices that include the SR-IOV capabilities. Physical Functions are discovered, managed, and configured as normal PCI devices. Physical Functions configure and manage the SR-IOV functionality by assigning Virtual Functions.
- Virtual Functions (VFs) are simple PCIe functions that only process I/O. Each Virtual Function is derived from a Physical Function. The number of Virtual Functions a device may have is limited by the device hardware. A single Ethernet port, the Physical Device, may map to many Virtual Functions that can be shared to virtual machines.

The hypervisor can map one or more Virtual Functions to a virtual machine. The Virtual Function's configuration space is then mapped to the configuration space presented to the guest.

Each Virtual Function can only be mapped to a single guest at a time, as Virtual Functions require real hardware resources. A virtual machine can have multiple Virtual Functions. A Virtual Function appears as a network card in the same way as a normal network card would appear to an operating system.

The SR-IOV drivers are implemented in the kernel. The core implementation is contained in the PCI subsystem, but there must also be driver support for both the Physical Function (PF) and Virtual Function (VF) devices. An SR-IOV capable device can allocate VFs from a PF. The VFs appear as

PCI devices which are backed on the physical PCI device by resources such as queues and register sets.

### Advantages of SR-IOV

SR-IOV devices can share a single physical port with multiple virtual machines.

Virtual Functions have near-native performance and provide better performance than paravirtualized drivers and emulated access. Virtual Functions provide data protection between virtual machines on the same physical server as the data is managed and controlled by the hardware.

These features allow for increased virtual machine density on hosts within a data center.

SR-IOV is better able to utilize the bandwidth of devices with multiple guests.

## 13.2. Using SR-IOV

This section covers the use of PCI passthrough to assign a Virtual Function of an SR-IOV capable multiport network card to a virtual machine as a network device.

SR-IOV Virtual Functions (VFs) can be assigned to virtual machines by adding a device entry in **<*hostdev*>** with the **`virsh edit`** or **`virsh attach-device`** command. However, this can be problematic because unlike a regular network device, an SR-IOV VF network device does not have a permanent unique MAC address, and is assigned a new MAC address each time the host is rebooted. Because of this, even if the guest is assigned the same VF after a reboot, when the host is rebooted the guest determines its network adapter to have a new MAC address. As a result, the guest believes there is new hardware connected each time, and will usually require re-configuration of the guest's network settings.

*libvirt-0.9.10* and later contains the **<*interface type='hostdev'*>** interface device. Using this interface device, **libvirt** will first perform any network-specific hardware/switch initialization indicated (such as setting the MAC address, VLAN tag, or 802.1Qbh virtualport parameters), then perform the PCI device assignment to the guest.

Using the **<*interface type='hostdev'*>** interface device requires:

- an SR-IOV-capable network card,
- host hardware that supports either the Intel VT-d or the AMD IOMMU extensions, and
- the PCI address of the VF to be assigned.

For a list of network interface cards (NICs) with SR-IOV support, see https://access.redhat.com/articles/1390483 .

**Important**

Assignment of an SR-IOV device to a virtual machine requires that the host hardware supports the Intel VT-d or the AMD IOMMU specification.

To attach an SR-IOV network device on an Intel or an AMD system, follow this procedure:

**Procedure 13.1. Attach an SR-IOV network device on an Intel or AMD system**

1. **Enable Intel VT-d or the AMD IOMMU specifications in the BIOS and kernel**

   On an Intel system, enable Intel VT-d in the BIOS if it is not enabled already. Refer to Procedure 12.1, “Preparing an Intel system for PCI device assignment” for procedural help on enabling Intel VT-d in the BIOS and kernel.

   Skip this step if Intel VT-d is already enabled and working.

   On an AMD system, enable the AMD IOMMU specifications in the BIOS if they are not enabled already. Refer to Procedure 12.2, “Preparing an AMD system for PCI device assignment” for procedural help on enabling IOMMU in the BIOS.

2. **Verify support**

   Verify if the PCI device with SR-IOV capabilities is detected. This example lists an Intel 82576 network interface card which supports SR-IOV. Use the **lspci** command to verify whether the device was detected.

   ```
   # lspci
   03:00.0 Ethernet controller: Intel Corporation 82576 Gigabit
   Network Connection (rev 01)
   03:00.1 Ethernet controller: Intel Corporation 82576 Gigabit
   Network Connection (rev 01)
   ```

   Note that the output has been modified to remove all other devices.

3. **Start the SR-IOV kernel modules**

   If the device is supported the driver kernel module should be loaded automatically by the kernel. Optional parameters can be passed to the module using the **modprobe** command. The Intel 82576 network interface card uses the **igb** driver kernel module.

   ```
   # modprobe igb [<option>=<VAL1>,<VAL2>,]
   # lsmod |grep igb
   igb    87592  0
   dca    6708    1 igb
   ```

4. **Activate Virtual Functions**

   The ***max_vfs*** parameter of the **igb** module allocates the maximum number of Virtual Functions. The ***max_vfs*** parameter causes the driver to spawn, up to the value of the parameter in, Virtual Functions. For this particular card the valid range is ***0*** to ***7***.

   Remove the module to change the variable.

   ```
   # modprobe -r igb
   ```

   Restart the module with the ***max_vfs*** set to ***7*** or any number of Virtual Functions up to the maximum supported by your device.

   ```
   # modprobe igb max_vfs=7
   ```

5. **Make the Virtual Functions persistent**

Add the line **options igb max_vfs=7** to any file in **/etc/modprobe.d** to make the Virtual Functions persistent. For example:

```
# echo "options igb max_vfs=7" >>/etc/modprobe.d/igb.conf
```

6. **Inspect the new Virtual Functions**

   Using the **lspci** command, list the newly added Virtual Functions attached to the Intel 82576 network device. (Alternatively, use **grep** to search for **Virtual Function**, to search for devices that support Virtual Functions.)

```
# lspci | grep 82576
0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit
Network Connection (rev 01)
0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit
Network Connection(rev 01)
0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.2 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.3 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.4 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.5 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.6 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.1 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.2 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.3 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.4 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
0b:11.5 Ethernet controller: Intel Corporation 82576 Virtual
Function (rev 01)
```

   The identifier for the PCI device is found with the ***-n*** parameter of the **lspci** command. The Physical Functions correspond to **0b:00.0** and **0b:00.1**. All Virtual Functions have **Virtual Function** in the description.

7. **Verify devices exist with virsh**

   The **libvirt** service must recognize the device before adding a device to a virtual machine. **libvirt** uses a similar notation to the **lspci** output. All punctuation characters, **;** and **.**, in **lspci** output are changed to underscores (_).

   Use the **virsh nodedev-list** command and the **grep** command to filter the Intel 82576

network device from the list of available host devices. ***0b*** is the filter for the Intel 82576 network devices in this example. This may vary for your system and may result in additional devices.

```
# virsh nodedev-list | grep 0b
pci_0000_0b_00_0
pci_0000_0b_00_1
pci_0000_0b_10_0
pci_0000_0b_10_1
pci_0000_0b_10_2
pci_0000_0b_10_3
pci_0000_0b_10_4
pci_0000_0b_10_5
pci_0000_0b_10_6
pci_0000_0b_11_7
pci_0000_0b_11_1
pci_0000_0b_11_2
pci_0000_0b_11_3
pci_0000_0b_11_4
pci_0000_0b_11_5
```

The serial numbers for the Virtual Functions and Physical Functions should be in the list.

8. **Get device details with virsh**

   The **pci_0000_0b_00_0** is one of the Physical Functions and **pci_0000_0b_10_0** is the first corresponding Virtual Function for that Physical Function. Use the **virsh nodedev-dumpxml** command to get advanced output for both devices.

```
# virsh nodedev-dumpxml pci_0000_0b_00_0
<device>
   <name>pci_0000_0b_00_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igb</name>
   </driver>
   <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>0</slot>
      <function>0</function>
      <product id='0x10c9'>82576 Gigabit Network
Connection</product>
      <vendor id='0x8086'>Intel Corporation</vendor>
   </capability>
</device>
```

```
# virsh nodedev-dumpxml pci_0000_0b_10_0
<device>
   <name>pci_0000_0b_10_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igbvf</name>
   </driver>
   <capability type='pci'>
```

```
      <domain>0</domain>
      <bus>11</bus>
      <slot>16</slot>
      <function>0</function>
      <product id='0x10ca'>82576 Virtual Function</product>
      <vendor id='0x8086'>Intel Corporation</vendor>
   </capability>
</device>
```

This example adds the Virtual Function **pci_0000_0b_10_0** to the virtual machine in Step 9. Note the **bus**, **slot** and **function** parameters of the Virtual Function: these are required for adding the device.

Copy these parameters into a temporary XML file, such as **/tmp/new-interface.xml** for example.

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

**Note**

If you do not specify a MAC address, one will be automatically generated. The ***<virtualport>*** element is only used when connecting to an 802.11Qbh hardware switch. The ***<vlan>*** element is new for Red Hat Enterprise Linux 6.4 and this will transparently put the guest's device on the VLAN tagged ***42***.

When the virtual machine starts, it should see a network device of the type provided by the physical adapter, with the configured MAC address. This MAC address will remain unchanged across host and guest reboots.

The following ***<interface>*** example shows the syntax for the optional ***<mac address>***, ***<virtualport>***, and ***<vlan>*** elements. In practice, use either the ***<vlan>*** or ***<virtualport>*** element, not both simultaneously as shown in the example:

```
...
 <devices>
   ...
   <interface type='hostdev' managed='yes'>
     <source>
       <address type='pci' domain='0' bus='11' slot='16'
function='0'/>
     </source>
     <mac address='52:54:00:6d:90:02'>
     <vlan>
        <tag id='42'/>
     </vlan>
     <virtualport type='802.1Qbh'>
       <parameters profileid='finance'/>
     </virtualport>
   </interface>
   ...
 </devices>
```

9. **Add the Virtual Function to the virtual machine**

   Add the Virtual Function to the virtual machine using the following command with the temporary file created in the previous step. This attaches the new device immediately and saves it for subsequent guest restarts.

   ```
   virsh attach-device MyGuest /tmp/new-interface.xml  --config
   ```

   Using the **--config** option ensures the new device is available after future guest restarts.

The virtual machine detects a new network interface card. This new card is the Virtual Function of the SR-IOV device.

## 13.3. Troubleshooting SR-IOV

This section contains solutions for problems which may affect SR-IOV.

**Error starting the guest**

When starting a configured virtual machine, an error occurs as follows:

```
# virsh start test
error: Failed to start domain test
error: internal error unable to start guest: char device
redirected to
/dev/pts/2
get_real_device: /sys/bus/pci/devices/0000:03:10.0/config:
Permission denied
init_assigned_device: Error: Couldn't get real device (03:10.0)!
Failed to initialize assigned device host=03:10.0
```

This error is often caused by a device that is already assigned to another guest or to the host itself.

**Error migrating, saving, or dumping the guest**

Attempts to migrate and dump the virtual machine cause an error similar to the following:

```
# virsh dump --crash 5 /tmp/vmcore
error: Failed to core dump domain 5 to /tmp/vmcore
error: internal error unable to execute QEMU command 'migrate':
An undefined
error has occurred
```

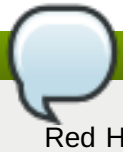
Because device assignment uses hardware on the specific host where the virtual machine was started, guest migration and save are not supported when device assignment is in use. Currently, the same limitation also applies to core-dumping a guest; this may change in the future.

# Chapter 14. KVM Guest Timing Management

Virtualization involves several intrinsic challenges for time keeping in guest virtual machines. Interrupts cannot always be delivered simultaneously and instantaneously to all guest virtual machines, because interrupts in virtual machines are not true interrupts; they are injected into the guest virtual machine by the host machine. The host may be running another guest virtual machine, or a different process, meaning that the precise timing typically required by interrupts may not always be possible.

Guest virtual machines without accurate time keeping may experience issues with network applications and processes, as session validity, migration, and other network activities rely on timestamps to remain correct.

KVM avoids these issues by providing guest virtual machines with a paravirtualized clock (**kvm-clock**). However, it is still vital to test timing before attempting activities that may be affected by time keeping inaccuracies.



**Note**

Red Hat Enterprise Linux 5.5 and newer, and Red Hat Enterprise Linux 6.0 and newer, use **kvm-clock** as their default clock source. Running without **kvm-clock** requires special configuration, and is not recommended.

**Important**

The Network Time Protocol (NTP) daemon should be running on the host and the guest virtual machines. Enable the **ntpd** service:

```
# service ntpd start
```

Add the ntpd service to the default startup sequence:

```
# chkconfig ntpd on
```

The **ntpd** service will correct the effects of clock skew as long as the clock runs no more than 0.05% faster or slower than the reference time source. The **ntp** startup script adjusts the clock offset from the reference time by adjusting the system clock at startup time, if required.

## 14.1. Constant Time Stamp Counter (TSC)

Modern Intel and AMD CPUs provide a constant Time Stamp Counter (TSC). The count frequency of the constant TSC does not vary when the CPU core itself changes frequency, for example, to comply with a power saving policy. A CPU with a constant TSC frequency is necessary in order to use the TSC as a clock source for KVM guests.

Your CPU has a constant Time Stamp Counter if the **constant_tsc** flag is present. To determine if your CPU has the **constant_tsc** flag run the following command:

```
$ cat /proc/cpuinfo | grep constant_tsc
```

If any output is given your CPU has the **constant_tsc** bit. If no output is given follow the instructions below.

### 14.1.1. Configuring Hosts without a Constant Time Stamp Counter

Systems without a constant TSC frequency cannot use the TSC as a clock source for virtual machines, and require additional configuration. Power management features interfere with accurate time keeping and must be disabled for guest virtual machines to accurately keep time with KVM.

**Important**

These instructions are for AMD revision F CPUs only.

If the CPU lacks the **constant_tsc** bit, disable all power management features (BZ#513138). Each system has several timers it uses to keep time. The TSC is not stable on the host, which is sometimes caused by **cpufreq** changes, deep C state, or migration to a host with a faster TSC. Deep C sleep states can stop the TSC. To prevent the kernel using deep C states append **processor.max_cstate=1** to the kernel boot options in the **grub.conf** file on the host:

```
title Red Hat Enterprise Linux (2.6.32-330.x86_64)
        root (hd0,0)
 kernel /vmlinuz-2.6.32-330.x86_64 ro root=/dev/VolGroup00/LogVol00 rhgb
quiet \
   processor.max_cstate=1
```

Disable **cpufreq** (only necessary on hosts without the **constant_tsc**) by editing the **/etc/sysconfig/cpuspeed** configuration file and change the **MIN_SPEED** and **MAX_SPEED** variables to the highest frequency available. Valid limits can be found in the **/sys/devices/system/cpu/cpu*/cpufreq/scaling_available_frequencies** files.

## 14.2. Required Parameters for Red Hat Enterprise Linux Guests

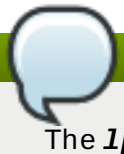
For certain Red Hat Enterprise Linux guest virtual machines, additional kernel parameters are required. These parameters can be set by appending them to the end of the **/kernel** line in the **/boot/grub/grub.conf** file of the guest virtual machine.

The table below lists versions of Red Hat Enterprise Linux and the parameters required on the specified systems.

**Table 14.1. Kernel parameter requirements**

| Red Hat Enterprise Linux version | Additional guest kernel parameters |
|---|---|
| 6.0 AMD64/Intel 64 with the paravirtualized clock | Additional parameters are not required |
| 6.0 AMD64/Intel 64 without the paravirtualized clock | notsc lpj=*n* |
| 5.5 AMD64/Intel 64 with the paravirtualized clock | Additional parameters are not required |
| 5.5 AMD64/Intel 64 without the paravirtualized clock | notsc lpj=*n* |

| Red Hat Enterprise Linux version | Additional guest kernel parameters |
|---|---|
| 5.5 x86 with the paravirtualized clock | Additional parameters are not required |
| 5.5 x86 without the paravirtualized clock | clocksource=acpi_pm lpj=*n* |
| 5.4 AMD64/Intel 64 | notsc |
| 5.4 x86 | clocksource=acpi_pm |
| 5.3 AMD64/Intel 64 | notsc |
| 5.3 x86 | clocksource=acpi_pm |
| 4.8 AMD64/Intel 64 | notsc |
| 4.8 x86 | clock=pmtmr |
| 3.9 AMD64/Intel 64 | Additional parameters are not required |
| 3.9 x86 | Additional parameters are not required |



**Note**

The ***lpj*** parameter requires a numeric value equal to the *loops per jiffy* value of the specific CPU on which the guest virtual machine runs. If you do not know this value, do not set the ***lpj*** parameter.

**Warning**

The ***divider*** kernel parameter was previously recommended for Red Hat Enterprise Linux 4 and 5 guest virtual machines that did not have high responsiveness requirements, or exist on systems with high guest density. It is no longer recommended for use with guests running Red Hat Enterprise Linux 4, or Red Hat Enterprise Linux 5 versions prior to version 5.8.

***divider*** can improve throughput on Red Hat Enterprise Linux 5 versions equal to or later than 5.8 by lowering the frequency of timer interrupts. For example, if **HZ=1000**, and ***divider*** is set to **10** (that is, **divider=10**), the number of timer interrupts per period changes from the default value (1000) to 100 (the default value, 1000, divided by the divider value, 10).

BZ#698842 details a bug in the way that the ***divider*** parameter interacts with interrupt and tick recording. This bug is fixed as of Red Hat Enterprise Linux 5.8. However, the ***divider*** parameter can still cause kernel panic in guests using Red Hat Enterprise Linux 4, or Red Hat Enterprise Linux 5 versions prior to version 5.8.

This parameter was not implemented in Red Hat Enterprise Linux 3, so this bug does not affect Red Hat Enterprise Linux 3 guests.

Red Hat Enterprise Linux 6 does not have a fixed-frequency clock interrupt; it operates in *tickless mode* and uses the timer dynamically as required. The ***divider*** parameter is therefore not useful for Red Hat Enterprise Linux 6, and Red Hat Enterprise Linux 6 guests are not affected by this bug.

## 14.3. Using the Real-Time Clock with Windows Server 2008, Windows Server 2008 R2, and Windows 7 Guests

Windows uses both the Real-Time Clock (RTC) and the Time Stamp Counter (TSC). For Windows guest virtual machines the Real-Time Clock can be used instead of the TSC for all time sources, which resolves guest timing issues.

The **boot.ini** file is no longer used as of Windows Server 2008 and newer. Windows Server 2008, Windows Server 2008 R2, and Windows 7 do not use the TSC as a time source if the **hypervisor-present** bit is set. The Red Hat Enterprise Linux 6 KVM hypervisor enables this CPUID bit by default, so it is no longer necessary to use the **Boot Configuration Data Editor** (**bcdedit.exe**) to modify the Windows boot parameters.

**Procedure 14.1. Using the Real-Time Clock with Windows Server 2008 R2 and Windows 7 guests**

1. Open the Windows guest virtual machine.
2. Open the **Accessories** menu of the **start** menu. Right click on the **Command Prompt** application, select **Run as Administrator**.
3. Confirm the security exception, if prompted.
4. Set the boot manager to use the platform clock. This should instruct Windows to use the PM timer for the primary clock source. The system UUID (***{default}***) in the example below) should be changed if the system UUID is different than the default boot device.

```
C:\Windows\system32>bcdedit /set {default} USEPLATFORMCLOCK on
The operation completed successfully
```

This fix should improve time keeping for Windows Server 2008 R2 and Windows 7 guests. Windows 2008 (non-R2) does not support the ***USEPLATFORMCLOCK*** parameter, but already uses the Real-Time Clock by default.

## 14.4. Steal Time Accounting

Steal time is the amount of CPU time desired by a guest virtual machine that is not provided by the host. Steal time occurs when the host allocates these resources elsewhere: for example, to another guest.

Steal time is reported in the CPU time fields in **/proc/stat** as ***st***. It is automatically reported by utilities such as **top** and **vmstat**, and cannot be switched off.

Large amounts of steal time indicate CPU contention, which can reduce guest performance. To relieve CPU contention, increase the guest's CPU priority or CPU quota, or run fewer guests on the host.

# Chapter 15. Network Booting with libvirt

Guest virtual machines can be booted with PXE enabled. PXE allows guest virtual machines to boot and load their configuration off the network itself. This section demonstrates some basic configuration steps to configure PXE guests with libvirt.

This section does not cover the creation of boot images or PXE servers. It is used to explain how to configure libvirt, in a private or bridged network, to boot a guest virtual machine with PXE booting enabled.

**Warning**

These procedures are provided only as an example. Ensure that you have sufficient backups before proceeding.

## 15.1. Preparing the Boot Server

To perform the steps in this chapter you will need:

- A PXE Server (DHCP and TFTP) - This can be a libvirt internal server, manually-configured dhcpd and tftpd, dnsmasq, a server configured by Cobbler, or some other server.
- Boot images - for example, PXELINUX configured manually or by Cobbler.

### 15.1.1. Setting up a PXE Boot Server on a Private libvirt Network

This example uses the *default* network. Perform the following steps:

**Procedure 15.1. Configuring the PXE boot server**

1. Place the PXE boot images and configuration in **`/var/lib/tftp`**.
2. Run the following commands:

```
# virsh net-destroy default
# virsh net-edit default
```

3. Edit the **`<ip>`** element in the configuration file for the *default* network to include the appropriate address, network mask, DHCP address range, and boot file, where *BOOT_FILENAME* represents the file name you are using to boot the guest virtual machine.

```
<ip address='192.168.122.1' netmask='255.255.255.0'>
   <tftp root='/var/lib/tftp' />
   <dhcp>
      <range start='192.168.122.2' end='192.168.122.254' />
      <bootp file='BOOT_FILENAME' />
   </dhcp>
</ip>
```

4. Boot the guest using PXE (refer to Section 15.2, “Booting a Guest Using PXE”).

## 15.2. Booting a Guest Using PXE

## 15.2. Booting a Guest Using PXE

This section demonstrates how to boot a guest virtual machine with PXE.

### 15.2.1. Using Bridged Networking

**Procedure 15.2. Booting a guest using PXE and bridged networking**

1. Ensure bridging is enabled such that the PXE boot server is available on the network.
2. Boot a guest virtual machine with PXE booting enabled. You can use the **virt-install** command to create a new virtual machine with PXE booting enabled, as shown in the following example command:

```
virt-install --pxe --network bridge=breth0 --prompt
```

Alternatively, ensure that the guest network is configured to use your bridged network, and that the XML guest configuration file has a **<boot dev='network'/>** element inside the **<os>** element, as shown in the following example:

```
<os>
   <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
   <boot dev='network'/>
   <boot dev='hd'/>
</os>
<interface type='bridge'>
   <mac address='52:54:00:5a:ad:cb'/>
   <source bridge='breth0'/>
   <target dev='vnet0'/>
   <alias name='net0'/>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
</interface>
```

### 15.2.2. Using a Private libvirt Network

**Procedure 15.3. Using a private libvirt network**

1. Configure PXE booting on libvirt as shown in Section 15.1.1, “Setting up a PXE Boot Server on a Private libvirt Network”.
2. Boot a guest virtual machine using libvirt with PXE booting enabled. You can use the **virt-install** command to create/install a new virtual machine using PXE:

```
virt-install --pxe --network network=default --prompt
```

Alternatively, ensure that the guest network is configured to use your bridged network, and that the XML guest configuration file has a **<boot dev='network'/>** element inside the **<os>** element, as shown in the following example:

```
<os>
   <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
   <boot dev='network'/>
   <boot dev='hd'/>
```

```
</os>
```

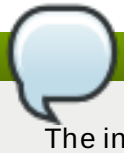
Also ensure that the guest virtual machine is connected to the private network:

```
<interface type='network'>
   <mac address='52:54:00:66:79:14'/>
   <source network='default'/>
   <target dev='vnet0'/>
   <alias name='net0'/>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
</interface>
```

# Chapter 16. Registering the Hypervisor and Virtual Machine

Red Hat Enterprise Linux 6 and 7 require that every guest virtual machine is mapped to a specific hypervisor in order to ensure that every guest is allocated the same level of subscription service. To do this you need to install a subscription agent that automatically detects all guest Virtual Machines (VMs) on each KVM hypervisor that is installed and registered, which in turn will create a mapping file that sits on the host. This mapping file ensures that all guest VMs receive the following benefits:

- Subscriptions specific to virtual systems are readily available and can be applied to all of the associated guest VMs.
- All subscription benefits that can be inherited from the hypervisor are readily available and can be applied to all of the associated guest VMs.



**Note**

The information provided in this chapter is specific to Red Hat Enterprise Linux subscriptions only. If you also have a Red Hat Enterprise Virtualization subscription, or a Red Hat Satellite subscription, you should also consult the virt-who information provided with those subscriptions. More information on Red Hat Subscription Management can also be found in the Red Hat Subscription Management Guide found on the customer portal.

## 16.1. Installing virt-who on the Host Physical Machine

1. **Register the KVM hypervisor**

   Register the KVM Hypervisor by running the **subscription-manager register [options]** command in a terminal as the root user on the host physical machine. More options are available using the # **subscription-manager register --help** menu. In cases where you are using a username and password, use the credentials that are known to the subscription manager. If this is your very first time subscribing and you do not have a user account, contact customer support. For example to register the VM as 'admin' with 'secret' as a password, you would send the following command:

   ```
   [root@rhel-server ~]# subscription-manager register --
   username=admin --password=secret --auto-attach --type=hypervisor
   ```

2. **Install the virt-who packages**

   Install the virt-who packages, by running the following command in a terminal as root on the host physical machine:

   ```
   [root@rhel-server ~]# yum install virt-who
   ```

3. **Create a virt-who configuration file**

   Add a configuration file in the **/etc/virt-who.d/** directory. It does not matter what the name of the file is, but you should give it a name that makes sense and the file must be located in the **/etc/virt-who.d/** directory. Inside that file add the following snippet and remember to save the file before closing it.

```
[libvirt]
type=libvirt
```

4. **Start the virt-who service**

   Start the virt-who service by running the following command in a terminal as root on the host physical machine:

```
[root@virt-who ~]# systemctl start virt-who.service
[root@virt-who ~]# systemctl enable virt-who.service
```

5. **Confirm virt-who service is receiving guest information**

   At this point, the virt-who service will start collecting a list of domains from the host. Check the **`/var/log/rhsm/rhsm.log`** file on the host physical machine to confirm that the the file contains a list of the guest VMs. For example:

```
2015-05-28 12:33:31,424 DEBUG: Libvirt domains found: [{'guestId':
'58d59128-cfbb-4f2c-93de-230307db2ce0', 'attributes': {'active': 0,
'virtWhoType': 'libvirt', 'hypervisorType': 'QEMU'}, 'state': 5}]
```

**Procedure 16.1. Managing the subscription on the customer portal**

1. **Subscribing the hypervisor**

   As the virtual machines will be receiving the same subscription benefits as the hypervisor, it is important that the hypervisor has a valid subscription and that the subscription is available for the VMs to use.

   a. **Login to the customer portal**

      Login to the Red Hat customer portal https://access.redhat.com/ and click the **`Subscriptions`** button at the top of the page.

   b. **Click the Systems link**

      In the **`Subscriber Inventory`** section (towards the bottom of the page), click **`Systems`** link.

   c. **Select the hypervisor**

      On the Systems page, there is a table of all subscribed systems. Click on the name of the hypervisor (localhost.localdomain for example). In the details page that opens, click **`Attach a subscription`** and select all the subscriptions listed. Click **`Attach Selected`**. This will attach the host's physical subscription to the hypervisor so that the guests can benefit from the subscription.

2. **Subscribing the guest virtual machines - first time use**

   This step is for those who have a new subscription and have never subscribed a guest virtual machine before. If you are adding virtual machines, skip this step. To consume the subscription assigned to the hypervisor profile on the machine running the virt-who service, auto subscribe by running the following command in a terminal, on the guest virtual machine as root.

```
[root@virt-who ~]# subscription-manager attach --auto
```

3. **Subscribing additional guest virtual machines**

   If you just subscribed a for the first time, skip this step. If you are adding additional virtual machines, it should be noted that running this command will not necessarily re-attach the same subscriptions to the guest virtual machine. This is because removing all subscriptions then allowing auto attach to resolve what is necessary for a given guest virtual machine may result in different subscriptions consumed than before. This may not have any effect on your system, but it is something you should be aware about. If you used a manual attachment procedure to attach the virtual machine, which is not described below, you will need to re-attach those virtual machines manually as the auto-attach will not work. Use the following command as root in a terminal to first remove the subscriptions for the old guests and then use the auto-attach to attach subscriptions to all the guests. Run these commands on the guest virtual machine.

```
[root@virt-who ~]# subscription-manager remove --all
[root@virt-who ~]# subscription-manager attach --auto
```

4. **Confirm subscriptions are attached**

   Confirm that the subscription is attached to the hypervisor by running the following command as root in a terminal on the guest virtual machine:

```
[root@virt-who ~]# subscription-manager list --consumed
```

   Output similar to the following will be displayed. Pay attention to the Subscription Details. It should say 'Subscription is current'.

```
[root@virt-who ~]#  subscription-manager list --consumed
+-------------------------------------------+
   Consumed Subscriptions
+-------------------------------------------+
Subscription Name: Awesome OS with unlimited virtual guests
Provides:   Awesome OS Server Bits
SKU:    awesomeos-virt-unlimited
Contract:   0
Account:   ######### Your account number #####
Serial:   ######### Your serial number ######
Pool ID:   XYZ123                                          (1)
Provides Management:  No
Active:   True
Quantity Used:   1
Service Level:
Service Type:
Status Details:  Subscription is current                   (2)
Subscription Type:
Starts:   01/01/2015
Ends:    12/31/2015
System Type:   Virtual
```

1. The ID for the subscription to attach to the system is displayed here. You will need this ID if you need to attach the subscription manually.
2. Indicates if your subscription is current. If your subscription is not current, an error message appears. One example is Guest has not been reported on any host and is using a temporary unmapped guest subscription. In this case the guest needs to be subscribed. In other cases, use the information as indicated in Section 16.5.2, “I have subscription status errors, what do I do?”.

5. **Register additional guests**

   When you install new guest VMs on the hypervisor, you must register the new VM and use the subscription attached to the hypervisor, by running the following commands in a terminal as root on the guest virtual machine:

```
[root@server1 ~]# subscription-manager register
[root@server1 ~]# subscription-manager attach --auto
[root@server1 ~]# subscription-manager list --consumed
```

## 16.2. Registering a New Guest Virtual Machine

In cases where a new guest virtual machine is to be created on a host that is already registered and running, the virt-who service must also be running. This ensures that the virt-who service maps the guest to a hypervisor, so the system is properly registered as a virtual system. To register the virtual machine, run the following command as root in a terminal:

```
[root@virt-server ~]# subscription-manager register --username=admin --
password=secret --auto-attach
```

## 16.3. Removing a Guest Virtual Machine Entry

If the guest virtual machine is running, unregister the system, by running the following command in a terminal window as root on the guest:

```
[root@virt-guest ~]# subscription-manager unregister
```

If the system has been deleted, however, the virtual service cannot tell whether the service is deleted or paused. In that case, you must manually remove the system from the server side, using the following steps:

1. **Login to the Subscription Manager**

   The Subscription Manager is located on the Red Hat Customer Portal. Login to the Customer Portal using your username and password, by clicking the login icon at the top of the screen.

2. **Click the Subscriptions tab**

   Click the **Subscriptions** tab.

3. **Click the Systems link**

   Scroll down the page and click the **Systems** link.

4. **Delete the system**

   To delete the system profile, locate the specified system's profile in the table, select the check box beside its name and click **Delete**.

## 16.4. Installing virt-who Manually

This section will describe how to manually attach the subscription provided by the hypervisor.

**Procedure 16.2. How to attach a subscription manually**

1. **List subscription information and find the Pool ID**

   First you need to list the available subscriptions which are of the virtual type. Run the following command in a terminal as root:

```
[root@server1 ~]# subscription-manager list --avail --match-
installed | grep 'Virtual' -B12
Subscription Name: Red Hat Enterprise Linux ES (Basic for
Virtualization)
Provides:          Red Hat Beta
                   Oracle Java (for RHEL Server)
                   Red Hat Enterprise Linux Server
SKU:               -------
Pool ID:           XYZ123
Available:         40
Suggested:         1
Service Level:     Basic
Service Type:      L1-L3
Multi-Entitlement: No
Ends:              01/02/2017
System Type:       Virtual
```

   Note the Pool ID displayed. Copy this ID as you will need it in the next step.

2. **Attach the subscription with the Pool ID**

   Using the Pool ID you copied in the previous step run the attach command. Replace the Pool ID XYZ123 with the Pool ID you retrieved. Run the following command in a terminal as root:

```
[root@server1 ~]# subscription-manager attach --pool=XYZ123

Successfully attached a subscription for: Red Hat Enterprise Linux
ES (Basic for Virtualization)
```

## 16.5. Troubleshooting virt-who

This section provides information on troubleshooting **virt-who**.

### 16.5.1. Why is the hypervisor status red?

Scenario: On the server side, you deploy a guest on a hypervisor that does not have a subscription. 24 hours later, the hypervisor displays its status as red. To remedy this situation you must get a subscription for that hypervisor. Or, permanently migrate the guest to a hypervisor with a subscription.

### 16.5.2. I have subscription status errors, what do I do?

Scenario: Any of the following error messages display:

- System not properly subscribed
- Status unknown
- Late binding of a guest to a hypervisor through virt-who (host/guest mapping)

To find the reason for the error open the virt-who log file, named **`rhsm.log`**, located in the **`/var/log/rhsm/`** directory.

# Appendix A. NetKVM Driver Parameters

After the NetKVM driver is installed, you can configure it to better suit your environment. The parameters listed in this section can be configured in the Windows **Device Manager** (**devmgmt.msc**).

**Important**

Modifying the driver's parameters causes Windows to re-load that driver. This interrupts existing network activity.

**Procedure A.1. Configuring NetKVM Parameters**

1. **Open Device Manager**

   Click on the **Start** button. In the right-hand pane, right-click on **Computer**, and click **Manage**. If prompted, click **Continue** on the **User Account Control** window. This opens the **Computer Management** window.

   In the left-hand pane of the **Computer Management** window, click **Device Manager**.

2. **Locate the correct device**

   In the central pane of the **Computer Management** window, click on the **+** symbol beside **Network adapters**.

   Under the list of **Red Hat VirtIO Ethernet Adapter** devices, double-click on **NetKVM**. This opens the **Properties** window for that device.

3. **View device parameters**

   In the **Properties** window, click on the **Advanced** tab.

4. **Modify device parameters**

   Click on the parameter you wish to modify to display the options for that parameter.

   Modify the options as appropriate, then click on **OK** to save your changes.

## A.1. Configurable Parameters for NetKVM

### Logging parameters

#### Logging.Enable

A Boolean value that determines whether logging is enabled. The default value is **1** (enabled).

#### Logging.Level

An integer that defines the logging level. As the integer increases, so does the verbosity of the log. The default value is **0** (errors only). **1-2** adds configuration messages. **3-4** adds packet flow information. **5-6** adds interrupt and DPC level trace information.

**Important**

High logging levels will slow down your guest virtual machine.

#### Logging.Statistics(sec)

An integer that defines whether log statistics are printed, and the time in seconds between each periodical statistics printout. The default value is **0** (no logging statistics).

### Initial parameters

#### Assign MAC

A string that defines the locally-administered MAC address for the paravirtualized NIC. This is not set by default.

#### Init.ConnectionRate(Mb)

An integer that represents the connection rate in megabytes. The default value for Windows 2008 and later is **10000**.

#### Init.Do802.1PQ

A Boolean value that enables Priority/VLAN tag population and removal support. The default value is **1** (enabled).

#### Init.UseMergedBuffers

A Boolean value that enables merge-able RX buffers. The default value is **1** (enabled).

#### Init.UsePublishEvents

A Boolean value that enables published event use. The default value is **1** (enabled).

#### Init.MTUSize

An integer that defines the maximum transmission unit (MTU). The default value is **1500**. Any value from 500 to 65500 is acceptable.

#### Init.IndirectTx

Controls whether indirect ring descriptors are in use. The default value is **Disable**, which disables use of indirect ring descriptors. Other valid values are **Enable**, which enables indirect ring descriptor usage; and **Enable***, which enables conditional use of indirect ring descriptors.

#### Init.MaxTxBuffers

An integer that represents the amount of TX ring descriptors that will be allocated. The default value is **1024**. Valid values are: 16, 32, 64, 128, 256, 512, or 1024.

#### Init.MaxRxBuffers

An integer that represents the amount of RX ring descriptors that will be allocated. The default value is **256**. Valid values are: 16, 32, 64, 128, 256, 512, or 1024.

#### Offload.Tx.Checksum

Specifies the TX checksum offloading mode.

In Red Hat Enterprise Linux 6.4 and onward, the valid values for this parameter are **All** (the default), which enables IP, TCP and UDP checksum offloading for both IPv4 and IPv6; **TCP/UDP(v4,v6)**, which enables TCP and UDP checksum offloading for both IPv4 and IPv6; **TCP/UDP(v4)**, which enables TCP and UDP checksum offloading for IPv4 only; and **TCP(v4)**, which enables only TCP checksum offloading for IPv4 only.

In Red Hat Enterprise Linux 6.3 and earlier, the valid values for this parameter are **TCP/UDP** (the default value), which enables TCP and UDP checksum offload; **TCP**, which enables only TCP checksum offload; or **Disable**, which disables TX checksum offload.

#### Offload.Tx.LSO

A Boolean value that enables TX TCP Large Segment Offload (LSO). The default value is **1** (enabled).

#### Offload.Rx.Checksum

Specifies the RX checksum offloading mode.

In Red Hat Enterprise Linux 6.4 and onward, the valid values for this parameter are **All** (the default), which enables IP, TCP and UDP checksum offloading for both IPv4 and IPv6; **TCP/UDP(v4,v6)**, which enables TCP and UDP checksum offloading for both IPv4 and IPv6; **TCP/UDP(v4)**, which enables TCP and UDP checksum offloading for IPv4 only; and **TCP(v4)**, which enables only TCP checksum offloading for IPv4 only.

In Red Hat Enterprise Linux 6.3 and earlier, the valid values are **Disable** (the default), which disables RX checksum offloading; **All**, which enables TCP, UDP, and IP checksum offloading; **TCP/UDP**, which enables TCP and UDP checksum offloading; and **TCP**, which enables only TCP checksum offloading.

### Test and debug parameters

**Important**

Test and debug parameters should only be used for testing or debugging; they should not be used in production.

#### TestOnly.DelayConnect(ms)

The period for which to delay connection upon startup, in milliseconds. The default value is **0**.

#### TestOnly.DPCChecking

Sets the DPC checking mode. **0** (the default) disables DPC checking. **1** enables DPC checking; each hang test verifies DPC activity and acts as if the DPC was spawned. **2** clears the device interrupt status and is otherwise identical to **1**.

#### TestOnly.Scatter-Gather

A Boolean value that determines whether scatter-gather functionality is enabled. The default value is **1** (enabled). Setting this value to **0** disables scatter-gather functionality and all dependent capabilities.

#### TestOnly.InterruptRecovery

A Boolean value that determines whether interrupt recovery is enabled. The default value is **1** (enabled).

**1** (enabled).

**TestOnly.PacketFilter**

A Boolean value that determines whether packet filtering is enabled. The default value is **1** (enabled).

**TestOnly.BatchReceive**

A Boolean value that determines whether packets are received in batches, or singularly. The default value is **1**, which enables batched packet receipt.

**TestOnly.Promiscuous**

A Boolean value that determines whether promiscuous mode is enabled. The default value is **0** (disabled).

**TestOnly.AnalyzeIPPackets**

A Boolean value that determines whether the checksum fields of outgoing IP packets are tested and verified for debugging purposes. The default value is **0** (no checking).

**TestOnly.RXThrottle**

An integer that determines the number of receive packets handled in a single DPC. The default value is **1000**.

**TestOnly.UseSwTxChecksum**

A Boolean value that determines whether hardware checksumming is enabled. The default value is **0** (disabled).

# Appendix B. Common libvirt Errors and Troubleshooting

This appendix documents common **libvirt**-related problems and errors along with instructions for dealing with them.

Locate the error on the table below and follow the corresponding link under **Solution** for detailed troubleshooting information.

**Table B.1. Common libvirt errors**

| Error | Description of problem | Solution |
|---|---|---|
| **libvirtd Failed to Start** | The **libvirt** daemon failed to start. However, there is no information about this error in **/var/log/messages**. | Section B.1, "**libvirtd** failed to start" |
| **Cannot read CA certificate** | This is one of several errors that occur when the URI fails to connect to the hypervisor. | Section B.2, "The URI Failed to Connect to the Hypervisor" |
| **Failed to connect socket ... : Permission denied** | This is one of several errors that occur when the URI fails to connect to the hypervisor. | Section B.2, "The URI Failed to Connect to the Hypervisor" |
| Other connectivity errors | These are other errors that occur when the URI fails to connect to the hypervisor. | Section B.2, "The URI Failed to Connect to the Hypervisor" |
| **Internal error guest CPU is not compatible with host CPU** | The guest virtual machine cannot be started because the host and guest processors are different. | Section B.3, "The guest virtual machine cannot be started: **internal error guest CPU is not compatible with host CPU**" |
| **Failed to create domain from vm.xml error: monitor socket did not show up.: Connection refused** | The guest virtual machine (or domain) starting fails and returns this error or similar. | Section B.4, "Guest starting fails with error: **monitor socket did not show up**" |
| **Internal error cannot find character device (null)** | This error can occur when attempting to connect a guest's console. It reports that there is no serial console configured for the guest virtual machine. | Section B.5, "**Internal error cannot find character device (null)**" |
| **No boot device** | After building a guest virtual machine from an existing disk image, the guest booting stalls. However, the guest can start successfully using the **QEMU** command directly. | Section B.6, "Guest virtual machine booting stalls with error: **No boot device**" |
| **The virtual network *"default"* has not been started** | If the *default* network (or other locally-created network) is unable to start, any virtual machine configured to use that network for its connectivity will also fail to start. | Section B.7, "Virtual network *default* has not been started" |

| Error | Description of problem | Solution |
|---|---|---|
| PXE boot (or DHCP) on guest failed | A guest virtual machine starts successfully, but is unable to acquire an IP address from DHCP, boot using the PXE protocol, or both. This is often a result of a long forward delay time set for the bridge, or when the *iptables* package and kernel do not support checksum mangling rules. | Section B.8, “PXE Boot (or DHCP) on Guest Failed” |
| Guest can reach outside network, but cannot reach host when using macvtap interface | A guest can communicate with other guests, but cannot connect to the host machine after being configured to use a macvtap (or **type='direct'**) network interface.<br><br>This is actually not an error — it is the defined behavior of macvtap. | Section B.9, “Guest Can Reach Outside Network, but Cannot Reach Host when Using macvtap Interface” |
| **Could not add rule to fixup DHCP response checksums on network *'default'*** | This warning message is almost always harmless, but is often mistakenly seen as evidence of a problem. | Section B.10, “Could not add rule to fixup DHCP response checksums on network *'default'*” |
| **Unable to add bridge br0 port vnet0: No such device** | This error message or the similar **Failed to add tap interface to bridge *'br0'*: No such device** reveal that the bridge device specified in the guest's (or domain's) **<interface>** definition does not exist. | Section B.11, “Unable to add bridge br0 port vnet0: No such device” |
| **Warning: could not open /dev/net/tun: no virtual network emulation qemu-kvm: -netdev tap,script=/etc/my-qemu-ifup,id=hostnet0: Device 'tap' could not be initialized** | The guest virtual machine does not start after configuring a **type='ethernet'** (or 'generic ethernet') interface in the host system. This error or similar appears either in **libvirtd.log**, **/var/log/libvirt/qemu/*name_of_guest*.log**, or in both. | Section B.12, “Guest is Unable to Start with Error: **warning: could not open /dev/net/tun**” |
| **Unable to resolve address *name_of_host* service '49155': Name or service not known** | **QEMU** guest migration fails and this error message appears with an unfamiliar hostname. | Section B.13, “Migration Fails with **Error: unable to resolve address**” |
| **Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or directory** | A guest virtual machine cannot be migrated because **libvirt** cannot access the disk image(s). | Section B.14, “Migration Fails with **Unable to allow access for disk path: No such file or directory**” |

| Error | Description of problem | Solution |
|---|---|---|
| No guest virtual machines are present when **libvirtd** is started | The **libvirt** daemon is successfully started, but no guest virtual machines appear to be present when running **virsh list --all**. | Section B.15, “No Guest Virtual Machines are Present when **libvirtd** is Started” |
| **Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor** | While **libvirtd** should listen on TCP ports for connections, the connection to the hypervisor fails. | Section B.16, “Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor” |
| Common XML errors | **libvirt** uses XML documents to store structured data. Several common errors occur with XML documents when they are passed to **libvirt** through the API. This entry provides instructions for editing guest XML definitions, and details common errors in XML syntax and configuration. | Section B.17, “Common XML Errors” |

## B.1. libvirtd failed to start

### Symptom

The **libvirt** daemon does not start automatically. Starting the **libvirt** daemon manually fails as well:

```
# /etc/init.d/libvirtd start
* Caching service dependencies ...
[ ok ]
* Starting libvirtd ...
/usr/sbin/libvirtd: error: Unable to initialize network sockets.
Check /var/log/messages or run without --daemon for more info.
* start-stop-daemon: failed to start `/usr/sbin/libvirtd'
[ !! ]
* ERROR: libvirtd failed to start
```

Moreover, there is not **'more info'** about this error in **/var/log/messages**.

### Investigation

Change **libvirt's** logging in **/etc/libvirt/libvirtd.conf** by uncommenting the line below. To uncomment the line, open the **/etc/libvirt/libvirtd.conf** file in a text editor, remove the hash (or #) symbol from the beginning of the following line, and save the change:

```
log_outputs="3:syslog:libvirtd"
```

**Note**

This line is commented out by default to prevent **libvirt** from producing excessive log messages. After diagnosing the problem, it is recommended to comment this line again in the **/etc/libvirt/libvirtd.conf** file.

Restart **libvirt** to determine if this has solved the problem.

If **libvirtd** still does not start successfully, an error similar to the following will be shown in the **/var/log/messages** file:

```
Feb  6 17:22:09 bart libvirtd: 17576: info : libvirt version:
0.9.9
Feb  6 17:22:09 bart libvirtd: 17576: error :
virNetTLSContextCheckCertFile:92: Cannot read CA certificate
'/etc/pki/CA/cacert.pem': No such file or directory
Feb  6 17:22:09 bart /etc/init.d/libvirtd[17573]: start-stop-
daemon: failed to start `/usr/sbin/libvirtd'
Feb  6 17:22:09 bart /etc/init.d/libvirtd[17565]: ERROR: libvirtd
failed to start
```

The **libvirtd** man page shows that the missing **cacert.pem** file is used as TLS authority when **libvirt** is run in **Listen for TCP/IP connections** mode. This means the ***--listen*** parameter is being passed.

**Solution**

Configure the **libvirt** daemon's settings with one of the following methods:

- Install a CA certificate.

**Note**

For more information on CA certificates and configuring system authentication, refer to the Configuring Authentication chapter in the *Red Hat Enterprise Linux 6 Deployment Guide*.

- Do not use TLS; use bare TCP instead. In **/etc/libvirt/libvirtd.conf** set **listen_tls = 0** and **listen_tcp = 1**. The default values are **listen_tls = 1** and **listen_tcp = 0**.
- Do not pass the ***--listen*** parameter. In **/etc/sysconfig/libvirtd.conf** change the ***LIBVIRTD_ARGS*** variable.

## B.2. The URI Failed to Connect to the Hypervisor

Several different errors can occur when connecting to the server (for example, when running **virsh**).

### B.2.1. Cannot read CA certificate

**Symptom**

When running a command, the following error (or similar) appears:

```
$ virsh -c name_of_uri list
error: Cannot read CA certificate '/etc/pki/CA/cacert.pem': No
such file or directory
error: failed to connect to the hypervisor
```

**Investigation**

The error message is misleading about the actual cause. This error can be caused by a variety of factors, such as an incorrectly specified URI, or a connection that is not configured.

**Solution**

**Incorrectly specified URI**

When specifying ***qemu:///system*** or ***qemu:///session*** as a connection URI, **virsh** attempts to connect to hostnames ***system*** or ***session*** respectively. This is because **virsh** recognizes the text after the second forward slash as the host.

Use three forward slashes to connect to the local host. For example, specifying ***qemu:///system*** instructs **virsh** connect to the ***system*** instance of **libvirtd** on the local host.

When a hostname is specified, the **QEMU** transport defaults to **TLS**. This results in certificates.

**Connection is not configured**

The URI is correct (for example, ***qemu[+tls]://server/system***) but the certificates are not set up properly on your machine. For information on configuring TLS, see Setting up **libvirt** for TLS available from the **libvirt** website.

## B.2.2. Failed to connect socket ... : Permission denied

**Symptom**

When running a **virsh** command, the following error (or similar) appears:

```
$ virsh -c qemu:///system list
error: Failed to connect socket to '/var/run/libvirt/libvirt-
sock': Permission denied
error: failed to connect to the hypervisor
```

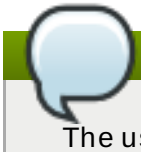
**Investigation**

Without any hostname specified, the connection to **QEMU** uses UNIX sockets by default. If there is no error running this command as root, the UNIX socket options in **/etc/libvirt/libvirtd.conf** are likely misconfigured.

**Solution**

To connect as a non-root user using UNIX sockets, configure the following options in **/etc/libvirt/libvirtd.conf**:

```
unix_sock_group = <group>
unix_sock_ro_perms = <perms>
unix_sock_rw_perms = <perms>
```



**Note**

The user running **virsh** must be a member of the **group** specified in the **unix_sock_group** option.

### B.2.3. Other Connectivity Errors

**Unable to connect to server at server: port: Connection refused**

The daemon is not running on the server or is configured not to listen, using configuration option **listen_tcp** or **listen_tls**.

**End of file while reading data: nc: using stream socket: Input/output error**

If you specified **ssh** transport, the daemon is likely not running on the server. Solve this error by verifying that the daemon is running on the server.

## B.3. The guest virtual machine cannot be started: internal error guest CPU is not compatible with host CPU

**Symptom**

Running on an Intel Core i7 processor (which **virt-manager** refers to as **Nehalem**, or the older Core 2 Duo, referred to as **Penryn**), a KVM guest (or domain) is created using **virt-manager**. After installation, the guest's processor is changed to match the host's CPU. The guest is then unable to start and reports this error:

```
2012-02-06 17:49:15.985+0000: 20757: error :
qemuBuildCpuArgStr:3565 : internal error guest CPU is not
compatible with host CPU
```

Additionally, clicking **Copy host CPU configuration** in **virt-manager** shows Pentium III instead of **Nehalem** or **Penryn**.

**Investigation**

The **/usr/share/libvirt/cpu_map.xml** file lists the flags that define each CPU model. The **Nehalem** and **Penryn** definitions contain this:

```
<feature name='nx'/>
```

As a result, the **NX** (or **No eXecute**) flag needs to be presented to identify the CPU as **Nehalem** or **Penryn**. However, in **/proc/cpuinfo**, this flag is missing.

**Solution**

Nearly all new BIOSes allow enabling or disabling of the **No eXecute** bit. However, if disabled, some CPUs do not report this flag and thus **libvirt** detects a different CPU. Enabling this functionality instructs **libvirt** to report the correct CPU. Refer to your hardware documentation for further instructions on this subject.

## B.4. Guest starting fails with error: `monitor socket did not show up`

### Symptom

The guest virtual machine (or domain) starting fails with this error (or similar):

```
# virsh -c qemu:///system create name_of_guest.xml error: Failed
to create domain from name_of_guest.xml error: monitor socket did
not show up.: Connection refused
```

### Investigation

This error message shows:

1. **libvirt** is working;
2. The **QEMU** process failed to start up; and
3. **libvirt** quits when trying to connect **QEMU** or the QEMU agent monitor socket.

To understand the error details, examine the guest log:

```
# cat /var/log/libvirt/qemu/name_of_guest.log
LC_ALL=C PATH=/sbin:/usr/sbin:/bin:/usr/bin QEMU_AUDIO_DRV=none
/usr/bin/qemu-kvm -S -M pc -enable-kvm -m 768 -smp
1,sockets=1,cores=1,threads=1 -name name_of_guest -uuid ebfaadbe-
e908-ba92-fdb8-3fa2db557a42 -nodefaults -chardev
socket,id=monitor,path=/var/lib/libvirt/qemu/name_of_guest.monito
r,server,nowait -mon chardev=monitor,mode=readline -no-reboot -
boot c -kernel /var/lib/libvirt/boot/vmlinuz -initrd
/var/lib/libvirt/boot/initrd.img -append
method=http://www.example.com/pub/product/release/version/x86_64/
os/ -drive
file=/var/lib/libvirt/images/name_of_guest.img,if=none,id=drive-
ide0-0-0,boot=on -device ide-drive,bus=ide.0,unit=0,drive=drive-
ide0-0-0,id=ide0-0-0 -device virtio-net-
pci,vlan=0,id=net0,mac=52:40:00:f4:f1:0a,bus=pci.0,addr=0x4 -net
tap,fd=42,vlan=0,name=hostnet0 -chardev pty,id=serial0 -device
isa-serial,chardev=serial0 -usb -vnc 127.0.0.1:0 -k en-gb -vga
cirrus -device virtio-balloon-pci,id=balloon0,bus=pci.0,
addr=0x3
char device redirected to /dev/pts/1
qemu: could not load kernel '/var/lib/libvirt/boot/vmlinuz':
Permission denied
```

### Solution

The guest log contains the details needed to fix the error.

If a host is shut down while the guest is still running a **libvirt** version prior to 0.9.5, the libvirt-guest's init script attempts to perform a managed save of the guest. If the managed

save was incomplete (for example, due to loss of power before the managed save image was flushed to disk), the save image is corrupted and will not be loaded by **QEMU**. The older version of **libvirt** does not recognize the corruption, making the problem perpetual. In this case, the guest log will show an attempt to use **-incoming** as one of its arguments, meaning that **libvirt** is trying to start **QEMU** by migrating in the saved state file.

This problem can be fixed by running **virsh managedsave-remove *name_of_guest*** to remove the corrupted managed save image. Newer versions of **libvirt** take steps to avoid the corruption in the first place, as well as adding **virsh start --force-boot *name_of_guest*** to bypass any managed save image.

## B.5. Internal error cannot find character device (null)

### Symptom

This error message appears when attempting to connect to a guest virtual machine's console:

```
# virsh console test2 Connected to domain test2 Escape character
is ^] error: internal error cannot find character device (null)
```

### Investigation

This error message shows that there is no serial console configured for the guest virtual machine.

### Solution

Set up a serial console in the guest's XML file.

**Procedure B.1. Setting up a serial console in the guest's XML**

1. Add the following XML to the guest virtual machine's XML using **virsh edit**:

```
<serial type='pty'>
  <target port='0'/>
</serial>
<console type='pty'>
  <target type='serial' port='0'/>
</console>
```

2. Set up the console in the guest kernel command line.

   To do this, either log in to the guest virtual machine to edit the **/boot/grub/grub.conf** file directly, or use the **virt-edit** command line tool. Add the following to the guest kernel command line:

```
console=ttyS0,115200
```

3. Run the followings command:

```
# virsh start vm && virsh console vm
```

## B.6. Guest virtual machine booting stalls with error: No boot device

### Symptom

After building a guest virtual machine from an existing disk image, the guest booting stalls with the error message **No boot device**. However, the guest virtual machine can start successfully using the **QEMU** command directly.

### Investigation

The disk's bus type is not specified in the command for importing the existing disk image:

```
# virt-install \
--connect qemu:///system \
--ram 2048 -n rhel_64 \
--os-type=linux --os-variant=rhel5 \
--disk  path=/root/RHEL-Server-5.8-64-
virtio.qcow2,device=disk,format=qcow2 \
--vcpus=2 --graphics spice --noautoconsole --import
```

However, the command line used to boot up the guest virtual machine using **QEMU** directly shows that it uses **virtio** for its bus type:

```
# ps -ef | grep qemu
/usr/libexec/qemu-kvm -monitor stdio -drive file=/root/RHEL-
Server-5.8-32-
virtio.qcow2,index=0,if=virtio,media=disk,cache=none,format=qcow2
-net nic,vlan=0,model=rtl8139,macaddr=00:30:91:aa:04:74 -net
tap,vlan=0,script=/etc/qemu-ifup,downscript=no -m 2048 -smp
2,cores=1,threads=1,sockets=2 -cpu qemu64,+sse2 -soundhw ac97 -
rtc-td-hack -M rhel5.6.0 -usbdevice tablet -vnc :10 -boot c -no-
kvm-pit-reinjection
```

Note the ***bus=*** in the guest's XML generated by **libvirt** for the imported guest:

```
<domain type='qemu'>
 <name>rhel_64</name>
 <uuid>6cd34d52-59e3-5a42-29e4-1d173759f3e7</uuid>
 <memory>2097152</memory>
 <currentMemory>2097152</currentMemory>
 <vcpu>2</vcpu>
 <os>
   <type arch='x86_64' machine='rhel5.4.0'>hvm</type>
   <boot dev='hd'/>
 </os>
 <features>
   <acpi/>
   <apic/>
   <pae/>
 </features>
 <clock offset='utc'>
   <timer name='pit' tickpolicy='delay'/>
 </clock>
 <on_poweroff>destroy</on_poweroff>
 <on_reboot>restart</on_reboot>
 <on_crash>restart</on_crash>
 <devices>
```

```
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none'/>
      <source file='/root/RHEL-Server-5.8-64-virtio.qcow2'/>
      <emphasis role="bold"><target dev='hda' bus='ide'/>
</emphasis>
      <address type='drive' controller='0' bus='0' unit='0'/>
    </disk>
    <controller type='ide' index='0'/>
    <interface type='bridge'>
      <mac address='54:52:00:08:3e:8c'/>
      <source bridge='br0'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes' keymap='en-us'/>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
    </video>
  </devices>
  </domain>
```

The bus type for the disk is set as ***ide***, which is the default value set by **libvirt**. This is the incorrect bus type, and has caused the unsuccessful boot for the imported guest.

### Solution

#### Procedure B.2. Correcting the disk bus type

1. Undefine the imported guest, then re-import it with ***bus=virtio*** and the following:

```
# virsh destroy rhel_64
# virsh undefine rhel_64
# virt-install \
--connect qemu:///system \
--ram 1024 -n rhel_64 -r 2048 \
--os-type=linux --os-variant=rhel5  \
--disk path=/root/RHEL-Server-5.8-64-
virtio.qcow2,device=disk,bus=virtio,format=qcow2 \
--vcpus=2 --graphics spice --noautoconsole --import
```

2. Edit the imported guest's XML using **virsh edit** and correct the disk bus type.

## B.7. Virtual network *default* has not been started

### Symptom

Normally, the configuration for a virtual network named *default* is installed as part of the *libvirt* package, and is configured to autostart when **libvirtd** is started.

If the *default* network (or any other locally-created network) is unable to start, any virtual machine configured to use that network for its connectivity will also fail to start, resulting in this error message:

```
Virtual network default has not been started
```

**Investigation**

One of the most common reasons for a **libvirt** virtual network's failure to start is that the dnsmasq instance required to serve DHCP and DNS requests from clients on that network has failed to start.

To determine if this is the cause, run **virsh net-start default** from a root shell to start the *default* virtual network.

If this action does not successfully start the virtual network, open **/var/log/libvirt/libvirtd.log** to view the complete error log message.

If a message similar to the following appears, the problem is likely a systemwide dnsmasq instance that is already listening on **libvirt**'s bridge, and is preventing **libvirt**'s own dnsmasq instance from doing so. The most important parts to note in the error message are **dnsmasq** and **exit status 2**:

```
Could not start virtual network default: internal error
Child process (/usr/sbin/dnsmasq --strict-order --bind-interfaces
--pid-file=/var/run/libvirt/network/default.pid --conf-file=
--except-interface lo --listen-address 192.168.122.1
--dhcp-range 192.168.122.2,192.168.122.254
--dhcp-leasefile=/var/lib/libvirt/dnsmasq/default.leases
--dhcp-lease-max=253 --dhcp-no-override) status unexpected: exit
status 2
```

**Solution**

If the machine is not using dnsmasq to serve DHCP for the physical network, disable dnsmasq completely.

If it is necessary to run dnsmasq to serve DHCP for the physical network, edit the **/etc/dnsmasq.conf** file. Add or uncomment the first line, as well as one of the two lines following that line. Do not add or uncomment all three lines:

```
bind-interfaces
interface=name_of_physical_interface
listen-address=chosen_IP_address
```

After making this change and saving the file, restart the systemwide dnsmasq service.

Next, start the *default* network with the **virsh net-start default** command.

Start the virtual machines.

## B.8. PXE Boot (or DHCP) on Guest Failed

**Symptom**

A guest virtual machine starts successfully, but is then either unable to acquire an IP address from DHCP or boot using the PXE protocol, or both. There are two common causes

address from DHCP or boot using the PXE protocol, or both. There are two common causes of this error: having a long forward delay time set for the bridge, and when the *iptables* package and kernel do not support checksum mangling rules.

#### Long forward delay time on bridge

##### Investigation

This is the most common cause of this error. If the guest network interface is connecting to a bridge device that has STP (Spanning Tree Protocol) enabled, as well as a long forward delay set, the bridge will not forward network packets from the guest virtual machine onto the bridge until at least that number of forward delay seconds have elapsed since the guest connected to the bridge. This delay allows the bridge time to watch traffic from the interface and determine the MAC addresses behind it, and prevent forwarding loops in the network topology.

If the forward delay is longer than the timeout of the guest's PXE or DHCP client, then the client's operation will fail, and the guest will either fail to boot (in the case of PXE) or fail to acquire an IP address (in the case of DHCP).

##### Solution

If this is the case, change the forward delay on the bridge to 0, or disable STP on the bridge.

**Note**

This solution applies only if the bridge is not used to connect multiple networks, but just to connect multiple endpoints to a single network (the most common use case for bridges used by **libvirt**).

If the guest has interfaces connecting to a **libvirt**-managed virtual network, edit the definition for the network, and restart it. For example, edit the default network with the following command:

```
# virsh net-edit default
```

Add the following attributes to the **<bridge>** element:

```
<name_of_bridge='virbr0' delay='0' stp='on'/>
```

**Note**

**delay='0'** and **stp='on'** are the default settings for virtual networks, so this step is only necessary if the configuration has been modified from the default.

If the guest interface is connected to a host bridge that was configured outside of **libvirt**, change the delay setting.

Add or edit the following lines in the **/etc/sysconfig/network-**

**scripts/ifcfg-*name_of_bridge*** file to turn STP on with a 0 second delay:

```
STP=on
DELAY=0
```

After changing the configuration file, restart the bridge device:

```
/sbin/ifdown name_of_bridge
/sbin/ifup name_of_bridge
```

**Note**

If *name_of_bridge* is not the root bridge in the network, that bridge's delay will eventually reset to the delay time configured for the root bridge. In this case, the only solution is to disable STP completely on *name_of_bridge*.

### The *iptables* package and kernel do not support checksum mangling rules

#### Investigation

This message is only a problem if all four of the following conditions are true:

- The guest is using **virtio** network devices.

  If so, the configuration file will contain **model type='virtio'**

- The host has the **vhost-net** module loaded.

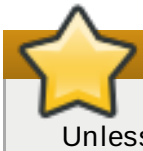
  This is true if **ls /dev/vhost-net** does not return an empty result.

- The guest is attempting to get an IP address from a DHCP server that is running directly on the host.

- The *iptables* version on the host is older than 1.4.10.

  *iptables* 1.4.10 was the first version to add the **libxt_CHECKSUM** extension. This is the case if the following message appears in the **libvirtd** logs:

```
warning: Could not add rule to fixup DHCP
response checksums on network default
warning: May need to update iptables package and
kernel to support CHECKSUM rule.
```



**Important**

Unless all of the other three conditions in this list are also true, the above warning message can be disregarded, and is not an indicator of any other problems.

When these conditions occur, UDP packets sent from the host to the guest have uncomputed checksums. This makes the host's UDP packets seem invalid to the guest's network stack.

### Solution

To solve this problem, invalidate any of the four points above. The best solution is to update the host *iptables* and kernel to *iptables-1.4.10* or later where possible. Otherwise, the most specific fix is to disable the **vhost-net** driver for this particular guest. To do this, edit the guest configuration with this command:

```
virsh edit name_of_guest
```

Change or add a **<driver>** line to the **<interface>** section:

```
<interface type='network'>
  <model type='virtio'/>
  <driver name='qemu'/>
  ...
</interface>
```

Save the changes, shut down the guest, and then restart it.

If this problem is still not resolved, the issue may be due to a conflict between **firewalld** and the default **libvirt** network.

To fix this, stop **firewalld** with the **service firewalld stop** command, then restart **libvirt** with the **service libvirtd restart** command.

## B.9. Guest Can Reach Outside Network, but Cannot Reach Host when Using macvtap Interface

### Symptom

A guest virtual machine can communicate with other guests, but cannot connect to the host machine after being configured to use a macvtap (also known as ***type='direct'***) network interface.

### Investigation

Even when not connecting to a Virtual Ethernet Port Aggregator (VEPA) or VN-Link capable switch, macvtap interfaces can be useful. Setting the mode of such an interface to **bridge** allows the guest to be directly connected to the physical network in a very simple manner without the setup issues (or *NetworkManager* incompatibility) that can accompany the use of a traditional host bridge device.

However, when a guest virtual machine is configured to use a ***type='direct'*** network interface such as macvtap, despite having the ability to communicate with other guests and other external hosts on the network, the guest cannot communicate with its own host.

This situation is actually not an error — it is the defined behavior of macvtap. Due to the way in which the host's physical Ethernet is attached to the macvtap bridge, traffic into that bridge from the guests that is forwarded to the physical interface cannot be bounced back up to the host's IP stack. Additionally, traffic from the host's IP stack that is sent to the

physical interface cannot be bounced back up to the macvtap bridge for forwarding to the guests.

**Solution**

Use **libvirt** to create an isolated network, and create a second interface for each guest virtual machine that is connected to this network. The host and guests can then directly communicate over this isolated network, while also maintaining compatibility with *NetworkManager*.

**Procedure B.3. Creating an isolated network with libvirt**

1. Add and save the following XML in the **/tmp/isolated.xml** file. If the 192.168.254.0/24 network is already in use elsewhere on your network, you can choose a different network.

```
<network>
  <name>isolated</name>
  <ip address='192.168.254.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.254.2' end='192.168.254.254' />
    </dhcp>
  </ip>
</network>
```

2. Create the network with this command: **virsh net-define /tmp/isolated.xml**

3. Set the network to autostart with the **virsh net-autostart isolated** command.

4. Start the network with the **virsh net-start isolated** command.

5. Using **virsh edit *name_of_guest***, edit the configuration of each guest that uses macvtap for its network connection and add a new ***<interface>*** in the ***<devices>*** section similar to the following (note the ***<model type='virtio'/>*** line is optional to include):

```
<interface type='network'>
  <source network='isolated'/>
  <model type='virtio'/>
</interface>
```

6. Shut down, then restart each of these guests.

The guests are now able to reach the host at the address 192.168.254.1, and the host will be able to reach the guests at the IP address they acquired from DHCP (alternatively, you can manually configure the IP addresses for the guests). Since this new network is isolated to only the host and guests, all other communication from the guests will use the macvtap interface.

## B.10. Could not add rule to fixup DHCP response checksums on network *'default'*

**Symptom**

This message appears:

```
Could not add rule to fixup DHCP response checksums on network
'default'
```

**Investigation**

Although this message appears to be evidence of an error, it is almost always harmless.

**Solution**

Unless the problem you are experiencing is that the guest virtual machines are unable to acquire IP addresses through DHCP, this message can be ignored.

If this is the case, refer to Section B.8, “PXE Boot (or DHCP) on Guest Failed” for further details on this situation.

## B.11. Unable to add bridge br0 port vnet0: No such device

**Symptom**

The following error message appears:

```
Unable to add bridge name_of_bridge port vnet0: No such device
```

For example, if the bridge name is *br0*, the error message will appear as:

```
Unable to add bridge br0 port vnet0: No such device
```

In **libvirt** versions 0.9.6 and earlier, the same error appears as:

```
Failed to add tap interface to bridge name_of_bridge: No such
device
```

Or for example, if the bridge is named *br0*:

```
Failed to add tap interface to bridge 'br0': No such device
```

**Investigation**

Both error messages reveal that the bridge device specified in the guest's (or domain's) **<interface>** definition does not exist.

To verify the bridge device listed in the error message does not exist, use the shell command **ifconfig *br0***.

A message similar to this confirms the host has no bridge by that name:

```
br0: error fetching interface information: Device not found
```

If this is the case, continue to the solution.

However, if the resulting message is similar to the following, the issue exists elsewhere:

```
br0         Link encap:Ethernet  HWaddr 00:00:5A:11:70:48
            inet addr:10.22.1.5  Bcast:10.255.255.255
Mask:255.0.0.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:249841 errors:0 dropped:0 overruns:0 frame:0
            TX packets:281948 errors:0 dropped:0 overruns:0
carrier:0
            collisions:0 txqueuelen:0
    RX bytes:106327234 (101.4 MiB)  TX bytes:21182634 (20.2 MiB)
```

**Solution**

**Edit the existing bridge or create a new bridge with virsh**

Use **virsh** to either edit the settings of an existing bridge or network, or to add the bridge device to the host system configuration.

**Edit the existing bridge settings using virsh**

Use **virsh edit *name_of_guest*** to change the **<interface>** definition to use a bridge or network that already exists.

For example, change **type='bridge'** to **type='network'**, and **<source bridge='br0'/>** to **<source network='default'/>**.

**Create a host bridge using virsh**

For **libvirt** version 0.9.8 and later, a bridge device can be created with the **virsh iface-bridge** command. This will create a bridge device *br0* with **eth0**, the physical network interface which is set as part of a bridge, attached:

```
virsh iface-bridge eth0 br0
```

Optional: If desired, remove this bridge and restore the original **eth0** configuration with this command:

```
virsh iface-unbridge br0
```

**Create a host bridge manually**

For older versions of **libvirt**, it is possible to manually create a bridge device on the host. Refer to Section 11.3, “Bridged Networking with libvirt” for instructions.

## B.12. Guest is Unable to Start with Error: warning: could not open /dev/net/tun

**Symptom**

The guest virtual machine does not start after configuring a **type='ethernet'** (also known as 'generic ethernet') interface in the host system. An error appears either in **libvirtd.log**, **/var/log/libvirt/qemu/*name_of_guest*.log**, or in both, similar to the below message:

```
warning: could not open /dev/net/tun: no virtual network
emulation qemu-kvm: -netdev tap,script=/etc/my-qemu-
ifup,id=hostnet0: Device 'tap' could not be initialized
```

### Investigation

Use of the generic ethernet interface type (**<interface type='ethernet'>**) is discouraged, because using it requires lowering the level of host protection against potential security flaws in **QEMU** and its guests. However, it is sometimes necessary to use this type of interface to take advantage of some other facility that is not yet supported directly in **libvirt**. For example, **openvswitch** was not supported in **libvirt** until *libvirt-0.9.11*, so in older versions of **libvirt**, **<interface type='ethernet'>** was the only way to connect a guest to an **openvswitch** bridge.

However, if you configure a **<interface type='ethernet'>** interface without making any other changes to the host system, the guest virtual machine will not start successfully.

The reason for this failure is that for this type of interface, a script called by **QEMU** needs to manipulate the tap device. However, with **type='ethernet'** configured, in an attempt to lock down **QEMU**, **libvirt** and SELinux have put in place several checks to prevent this. (Normally, **libvirt** performs all of the tap device creation and manipulation, and passes an open file descriptor for the tap device to **QEMU**.)

### Solution

Reconfigure the host system to be compatible with the generic ethernet interface.

**Procedure B.4. Reconfiguring the host system to use the generic ethernet interface**

1. Set SELinux to permissive by configuring **SELINUX=permissive** in **/etc/selinux/config**:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of
enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=permissive
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

2. From a root shell, run the command **setenforce permissive**.

3. In **/etc/libvirt/qemu.conf** add or edit the following lines:

```
clear_emulator_capabilities = 0
```

```
user = "root"
```

```
group = "root"
```

```
cgroup_device_acl = [
        "/dev/null", "/dev/full", "/dev/zero",
        "/dev/random", "/dev/urandom",
        "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
 "/dev/rtc", "/dev/hpet", "/dev/net/tun",
```

4. Restart **libvirtd**.

**Important**

Since each of these steps significantly decreases the host's security protections against **QEMU** guest domains, this configuration should only be used if there is no alternative to using **<interface type='ethernet'>**.

**Note**

For more information on SELinux, refer to the *Red Hat Enterprise Linux 6 Security-Enhanced Linux User Guide*.

## B.13. Migration Fails with `Error: unable to resolve address`

### Symptom

**QEMU** guest migration fails and this error message appears:

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
  error: Unable to resolve address name_of_host service '49155':
Name or service not known
```

For example, if the destination hostname is "newyork", the error message will appear as:

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
error: Unable to resolve address 'newyork' service '49155': Name
or service not known
```

However, this error looks strange as we did not use "newyork" hostname anywhere.

### Investigation

During migration, **libvirtd** running on the destination host creates a URI from an address and port where it expects to receive migration data and sends it back to **libvirtd** running on the source host.

In this case, the destination host (**192.168.122.12**) has its name set to *'newyork'*. For some reason, **libvirtd** running on that host is unable to resolve the name to an IP address that could be sent back and still be useful. For this reason, it returned the *'newyork'* hostname hoping the source **libvirtd** would be more successful with resolving the name. This can happen if DNS is not properly configured or **/etc/hosts** has the hostname associated with local loopback address (**127.0.0.1**).

Note that the address used for migration data cannot be automatically determined from the address used for connecting to destination **libvirtd** (for example, from **qemu+tcp://192.168.122.12/system**). This is because to communicate with the destination **libvirtd**, the source **libvirtd** may need to use network infrastructure different from that which **virsh** (possibly running on a separate machine) requires.

**Solution**

The best solution is to configure DNS correctly so that all hosts involved in migration are able to resolve all host names.

If DNS cannot be configured to do this, a list of every host used for migration can be added manually to the **/etc/hosts** file on each of the hosts. However, it is difficult to keep such lists consistent in a dynamic environment.

If the host names cannot be made resolvable by any means, **virsh migrate** supports specifying the migration host:

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
tcp://192.168.122.12
```

Destination **libvirtd** will take the **tcp://192.168.122.12** URI and append an automatically generated port number. If this is not desirable (because of firewall configuration, for example), the port number can be specified in this command:

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
tcp://192.168.122.12:12345
```

Another option is to use tunnelled migration. Tunnelled migration does not create a separate connection for migration data, but instead tunnels the data through the connection used for communication with destination **libvirtd** (for example, **qemu+tcp://192.168.122.12/system**):

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system --p2p --
tunnelled
```

## B.14. Migration Fails with `Unable to allow access for disk path: No such file or directory`

**Symptom**

A guest virtual machine (or domain) cannot be migrated because **libvirt** cannot access the disk image(s):

```
# virsh migrate qemu qemu+tcp://name_of_host/system
error: Unable to allow access for disk path
/var/lib/libvirt/images/qemu.img: No such file or directory
```

For example, if the destination hostname is "newyork", the error message will appear as:

```
# virsh migrate qemu qemu+tcp://newyork/system
error: Unable to allow access for disk path
/var/lib/libvirt/images/qemu.img: No such file or directory
```

### Investigation

By default, migration only transfers the in-memory state of a running guest (such as memory or CPU state). Although disk images are not transferred during migration, they need to remain accessible at the same path by both hosts.

### Solution

Set up and mount shared storage at the same location on both hosts. The simplest way to do this is to use NFS:

**Procedure B.5. Setting up shared storage**

1. Set up an NFS server on a host serving as shared storage. The NFS server can be one of the hosts involved in the migration, as long as all hosts involved are accessing the shared storage through NFS.

```
# mkdir -p /exports/images
# cat >>/etc/exports <<EOF
/exports/images    192.168.122.0/24(rw,no_root_squash)
EOF
```

2. Mount the exported directory at a common location on all hosts running **libvirt**. For example, if the IP address of the NFS server is 192.168.122.1, mount the directory with the following commands:

```
# cat >>/etc/fstab <<EOF
192.168.122.1:/exports/images  /var/lib/libvirt/images  nfs
auto  0 0
EOF
# mount /var/lib/libvirt/images
```

**Note**

It is not possible to export a local directory from one host using NFS and mount it at the same path on another host — the directory used for storing disk images must be mounted from shared storage on both hosts. If this is not configured correctly, the guest virtual machine may lose access to its disk images during migration, because the source host's **libvirt** daemon may change the owner, permissions, and SELinux labels on the disk images after it successfully migrates the guest to its destination.

If **libvirt** detects that the disk images are mounted from a shared storage location, it will not make these changes.

## B.15. No Guest Virtual Machines are Present when libvirtd is Started

### Symptom

The **libvirt** daemon is successfully started, but no guest virtual machines appear to be present.

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
#
```

### Investigation

There are various possible causes of this problem. Performing these tests will help to determine the cause of this situation:

#### Verify KVM kernel modules

Verify that KVM kernel modules are inserted in the kernel:

```
# lsmod | grep kvm
kvm_intel             121346  0
kvm                   328927  1 kvm_intel
```

If you are using an AMD machine, verify the **kvm_amd** kernel modules are inserted in the kernel instead, using the similar command **lsmod | grep kvm_amd** in the root shell.

If the modules are not present, insert them using the **modprobe *<modulename>*** command.

**Note**

Although it is uncommon, KVM virtualization support may be compiled into the kernel. In this case, modules are not needed.

#### Verify virtualization extensions

Verify that virtualization extensions are supported and enabled on the host:

```
# egrep "(vmx|svm)" /proc/cpuinfo
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv
svm_lock nrip_save
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv
svm_lock nrip_save
```

Enable virtualization extensions in your hardware's firmware configuration within the BIOS setup. Refer to your hardware documentation for further details on this.

#### Verify client URI configuration

Verify that the URI of the client is configured as desired:

```
# virsh uri
vbox:///system
```

For example, this message shows the URI is connected to the **VirtualBox** hypervisor, not **QEMU**, and reveals a configuration error for a URI that is otherwise set to connect to a **QEMU** hypervisor. If the URI was correctly connecting to **QEMU**, the same message would appear instead as:

```
# virsh uri
qemu:///system
```

This situation occurs when there are other hypervisors present, which **libvirt** may speak to by default.

### Solution

After performing these tests, use the following command to view a list of guest virtual machines:

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

## B.16. Unable to connect to server at 'host:16509': Connection refused ... error: failed to connect to the hypervisor

### Symptom

While **libvirtd** should listen on TCP ports for connections, the connections fail:

```
# virsh -c qemu+tcp://host/system
error: unable to connect to server at 'host:16509': Connection
refused
error: failed to connect to the hypervisor
```

The **libvirt** daemon is not listening on TCP ports even after changing configuration in **/etc/libvirt/libvirtd.conf**:

```
# grep listen_ /etc/libvirt/libvirtd.conf
listen_tls = 1
listen_tcp = 1
listen_addr = "0.0.0.0"
```

However, the TCP ports for **libvirt** are still not open after changing configuration:

```
# netstat -lntp | grep libvirtd
#
```

### Investigation

The **libvirt** daemon was started without the **--listen** option. Verify this by running this command:

```
# ps aux | grep libvirtd
root     27314  0.0  0.0 1000920 18304 ?       Sl   Feb16   1:19
libvirtd --daemon
```

The output does not contain the **--listen** option.

### Solution

Start the daemon with the **--listen** option.

To do this, modify the **/etc/sysconfig/libvirtd** file and uncomment the following line:

```
#LIBVIRTD_ARGS="--listen"
```

Then restart the **libvirtd** service with this command:

```
# /etc/init.d/libvirtd restart
```

## B.17. Common XML Errors

The **libvirt** tool uses XML documents to store structured data. A variety of common errors occur with XML documents when they are passed to **libvirt** through the API. Several common XML errors — including misformatted XML, inappropriate values, and missing elements — are detailed below.

### B.17.1. Editing Domain Definition

Although it is not recommended, it is sometimes necessary to edit a guest virtual machine's (or a domain's) XML file manually. To access the guest's XML for editing, use the following command:

```
# virsh edit name_of_guest.xml
```

This command opens the file in a text editor with the current definition of the guest virtual machine. After finishing the edits and saving the changes, the XML is reloaded and parsed by **libvirt**. If the XML is correct, the following message is displayed:

```
# virsh edit name_of_guest.xml

Domain name_of_guest.xml XML configuration edited.
```

**Important**

When using the **edit** command in **virsh** to edit an XML document, save all changes before exiting the editor.

After saving the XML file, use the **xmllint** command to validate that the XML is well-formed, or the **virt-xml-validate** command to check for usage problems:

```
# xmllint --noout config.xml
```

```
# virt-xml-validate config.xml
```

If no errors are returned, the XML description is well-formed and matches the **libvirt** schema. While the schema does not catch all constraints, fixing any reported errors will further troubleshooting.

**XML documents stored by libvirt**

These documents contain definitions of states and configurations for the guests. These documents are automatically generated and should not be edited manually. Errors in these documents contain the file name of the broken document. The file name is valid only on the host machine defined by the URI, which may refer to the machine the command was run on.

Errors in files created by **libvirt** are rare. However, one possible source of these errors is a downgrade of **libvirt** — while newer versions of **libvirt** can always read XML generated by older versions, older versions of **libvirt** may be confused by XML elements added in a newer version.

## B.17.2. XML Syntax Errors

Syntax errors are caught by the XML parser. The error message contains information for identifying the problem.

This example error message from the XML parser consists of three lines — the first line denotes the error message, and the two following lines contain the context and location of the XML code containing the error. The third line contains an indicator showing approximately where the error lies on the line above it:

```
error: (name_of_guest.xml):6: StartTag: invalid element name
<vcpu>2</vcpu><
-----------------^
```

**Information contained in this message:**

**(*name_of_guest.xml*)**

This is the file name of the document that contains the error. File names in parentheses are symbolic names to describe XML documents parsed from memory, and do not directly correspond to files on disk. File names that are not contained in parentheses are local files that reside on the target of the connection.

**6**

This is the line number in the XML file that contains the error.

**StartTag: invalid element name**

This is the error message from the **libxml2** parser, which describes the specific XML error.

### B.17.2.1. Stray < in the document

**Symptom**

The following error occurs:

```
error: (name_of_guest.xml):6: StartTag: invalid element name
<vcpu>2</vcpu><
-----------------^
```

**Investigation**

This error message shows that the parser expects a new element name after the **<** symbol on line 6 of a guest's XML file.

Ensure line number display is enabled in your text editor. Open the XML file, and locate the text on line 6:

```
<domain type='kvm'>
   <name>name_of_guest</name>
<memory>524288</memory>
<vcpu>2</vcpu><
```

This snippet of a guest's XML file contains an extra **<** in the document:

**Solution**

Remove the extra **<** or finish the new element.

### B.17.2.2. Unterminated attribute

**Symptom**

The following error occurs:

```
error: (name_of_guest.xml):2: Unescaped '<' not allowed in
attributes values
<name>name_of_guest</name>
--^
```

**Investigation**

This snippet of a guest's XML file contains an unterminated element attribute value:

```
<domain type='kvm>
<name>name_of_guest</name>
```

In this case, **'kvm'** is missing a second quotation mark. Strings of attribute values, such as quotation marks and apostrophes, must be opened and closed, similar to XML start and end tags.

**Solution**

Correctly open and close all attribute value strings.

### B.17.2.3. Opening and ending tag mismatch

**Symptom**

The following error occurs:

```
error: (name_of_guest.xml):61: Opening and ending tag mismatch:
clock line 16 and domain
</domain>
---------^
```

**Investigation**

The error message above contains three clues to identify the offending tag:

The message following the last colon, **clock line 16 and domain**, reveals that **<clock>** contains a mismatched tag on line 16 of the document. The last hint is the pointer in the context part of the message, which identifies the second offending tag.

Unpaired tags must be closed with **/>**. The following snippet does not follow this rule and has produced the error message shown above:

```
<domain type='kvm'>
  ...
    <clock offset='utc'>
```

This error is caused by mismatched XML tags in the file. Every XML tag must have a matching start and end tag.

**Other examples of mismatched XML tags**

The following examples produce similar error messages and show variations of mismatched XML tags.

This snippet contains an unended pair tag for **<features>**:

```
<domain type='kvm'>
 ...
 <features>
   <acpi/>
   <pae/>
 ...
 </domain>
```

This snippet contains an end tag (**</name>**) without a corresponding start tag:

```
<domain type='kvm'>
  </name>
  ...
</domain>
```

**Solution**

Ensure all XML tags start and end correctly.

### B.17.2.4. Typographical errors in tags

**Symptom**

The following error message appears:

```
error: (name_of_guest.xml):1: Specification mandate value for
attribute ty
<domain ty pe='kvm'>
-----------^
```

**Investigation**

XML errors are easily caused by a simple typographical error. This error message highlights the XML error — in this case, an extra white space within the word **type** — with a pointer.

```
<domain ty pe='kvm'>
```

These XML examples will not parse correctly because of typographical errors such as a missing special character, or an additional character:

```
<domain type 'kvm'>
```

```
<dom#ain type='kvm'>
```

**Solution**

To identify the problematic tag, read the error message for the context of the file, and locate the error with the pointer. Correct the XML and save the changes.

## B.17.3. Logic and Configuration Errors

A well-formatted XML document can contain errors that are correct in syntax but **libvirt** cannot parse. Many of these errors exist, with two of the most common cases outlined below.

### B.17.3.1. Vanishing parts

**Symptom**

Parts of the change you have made do not show up and have no effect after editing or defining the domain. The **define** or **edit** command works, but when dumping the XML once again, the change disappears.

**Investigation**

This error likely results from a broken construct or syntax that libvirt does not parse. The **libvirt** tool will generally only look for constructs it knows but ignore everything else, resulting in some of the XML changes vanishing after **libvirt** parses the input.

**Solution**

Validate the XML input before passing it to the **edit** or **define** commands. The **libvirt** developers maintain a set of XML schemas bundled with **libvirt** which define the majority of the constructs allowed in XML documents used by **libvirt**.

Validate **libvirt** XML files using the following command:

```
# virt-xml-validate libvirt.xml
```

If this command passes, **libvirt** will likely understand all constructs from your XML, except if the schemas cannot detect options which are valid only for a given hypervisor. Any XML generated by **libvirt** as a result of a **virsh dump** command, for example, should validate without error.

### B.17.3.2. Incorrect drive device type

**Symptom**

The definition of the source image for the CD-ROM virtual drive is not present, despite being added:

```
# virsh dumpxml domain
<domain type='kvm'>
  ...
  <disk type='block' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <target dev='hdc' bus='ide'/>
    <readonly/>
  </disk>
  ...
</domain>
```

### Solution

Correct the XML by adding the missing **<source>** parameter as follows:

```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source file='/path/to/image.iso'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
</disk>
```

A **type='block'** disk device expects that the source is a physical device. To use the disk with an image file, use **type='file'** instead.

# Appendix C. Revision History

| | | |
|---|---|---|
| **Revision 0.5-43** | **Mon June 24 2016** | **Jiri Herrmann** |
| Several post-release updates | | |
| **Revision 0.5-42** | **Mon May 02 2016** | **Jiri Herrmann** |
| Multiple updates for the 6.8 GA release | | |
| **Revision 0.5-41** | **Tue Mar 01 2016** | **Jiri Herrmann** |
| Multiple updates for the 6.8 beta release | | |
| **Revision 0.5-40** | **Thu Oct 08 2015** | **Jiri Herrmann** |
| Cleaned up the Revision History | | |
| **Revision 0.5-39** | **Wed Apr 29 2015** | **Dayle Parker** |
| Republishing for 6.7 Beta. | | |