KVM : Kernel Based Virtual Machine

Module # 5

Introduction

- Kernel-based Virtual Machine (KVM) is an open source virtualization technology built into Linux
- Specifically, KVM lets you turn Linux into a hypervisor that allows a host machine to run multiple, isolated virtual environments called guests or virtual machines (VMs).

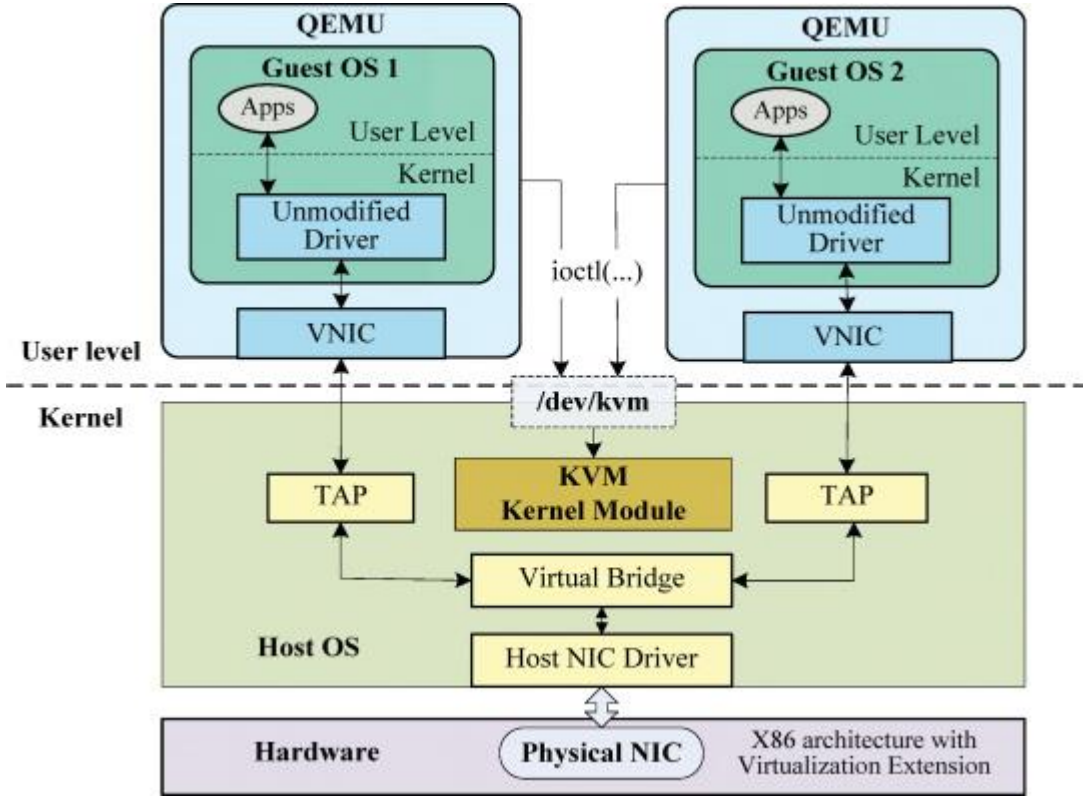
Introduction

- KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V).
- It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure and a processor specific module, kvm-intel.ko or kvm-amd.ko.
- KVM also requires a modified QEMU although work is underway to get the required changes upstream.

How it Works

- KVM converts Linux into a type-1 (bare-metal) hypervisor.
- All hypervisors need some operating system-level components—such as a memory manager, process scheduler, input/output (I/O) stack, device drivers, security manager, a network stack, and more—to run VMs. KVM has all these components because it’s part of the Linux kernel.
- Every VM is implemented as a regular Linux process, scheduled by the standard Linux scheduler, with dedicated virtual hardware like a network card, graphics adapter, CPU(s), memory, and disks.

KVM Architecture



QEMU

- QEMU is the software that actually creates the hardware which a guest operating system runs on top of.
- All of this virtual hardware is created by code, depending on the QEMU configuration.
- All the devices that the guest operating system sees, like a keyboard, a mouse, a network card, and so on, are instances of code within the QEMU project.
- In addition to mimicking real hardware, QEMU also creates some devices written specifically for virtualization use-cases.

QEMU

- Creates the machine
- Device emulation code
  - some mimic real devices
  - some are special: paravirtualized
- Uses several services from host kernel
  - KVM for guest control
  - networking
  - disk IO
  - etc.

Libvirt

- LIBVIRT is an open source API, daemon and management tool for managing platform virtualization. It can be used to manage KVM, Xen, VMware ESX, QEMU and other virtualization technologies.
- These APIs are widely used in the orchestration layer of hypervisors in the development of a cloud-based solution.
- Internals : LIBVIRT itself is a C library, but it has bindings in other languages, notably in Python, Perl, OCaml, Ruby, Java, JavaScript (via Node.js) and PHP. libvirt for these programming languages is composed of wrappers around another class/package called libvirtmod. libvirtmod’s implementation is closely associated with its counterpart in C/C++ in syntax and functionality.

Use Cases

- Server consolidation
- Virtual desktop infrastructure
- Compute cloud
- Embedded End-user virtualization

Features

- i386 and x86_64 UP and SMP guests
- Runs Linux, Windows, and many other OSes
- PCI pass-through
- Optional paravirtualized I/O
- Live migration including block migration
- Snapshot save/resume
- Guest swapping and memory dedup (KSM)

KVM kernel module

http://linux-kvm.org

- Part of mainline Linux kernel tree.
- Provides common interface for Intel VMX and AMD SVM hardware assist.
- Contains emulation for instructions and CPU modes not supported by hardware assist.
- Handles performance critical parts of timers and interrupts via in-kernel I/O emulation.

KVM Installation

Step 1: Verifying Hardware Support

Before installing KVM, it is important to verify if your CPU supports virtualization and if the virtualization technology has been activated on your CPU.

```
egrep -c '(svm|vmx)' /proc/cpuinfo
```

- An output of 1 or anything greater than that indicates that your CPU can be set-up for using the virtualization technology.
- An output of 0 indicates the inability of your system to run KVM. In my case, the output is 4 which verifies that I can install and run virtual machine(s).

KVM Installation

- The next thing to do is to check if the virtualization technology is enabled on your system or not as it is very important to run KVM. Use the following command in order to do so: **`sudo kvm-ok`**
- The output will either indicate that your CPU has not been configured to run virtualization or your system has virtualization enabled and this can be used for KVM acceleration.
- If it returns error, use the following package:**`sudo apt install cpu-checker`**

KVM Installation

Step 2: Installing the KVM Package

Now that you have verified the prerequisites for installing KVM on your system, use the following command to install KVM:

```
sudo apt update

sudo apt-get install qemu-kvm libvirt-bin bridge-utils virt-manager
```

Step 3: Adding Your User Account on KVM

You can use virtual machines on KVM only if you are a root user or if you are part of the libvirt and kvm group. So, add user to libvirt and kvm groups

```
sudo adduser ‘username’ libvirt

sudo adduser ‘[username]’ kvm
```

KVM Installation

Step 4: Verify the Installation

Confirm the installation was successful by using the `virsh` command:

```
virsh list --all
or
sudo systemctl status libvirtd
```

If the virtualization daemon is not active, activate it with the following command:

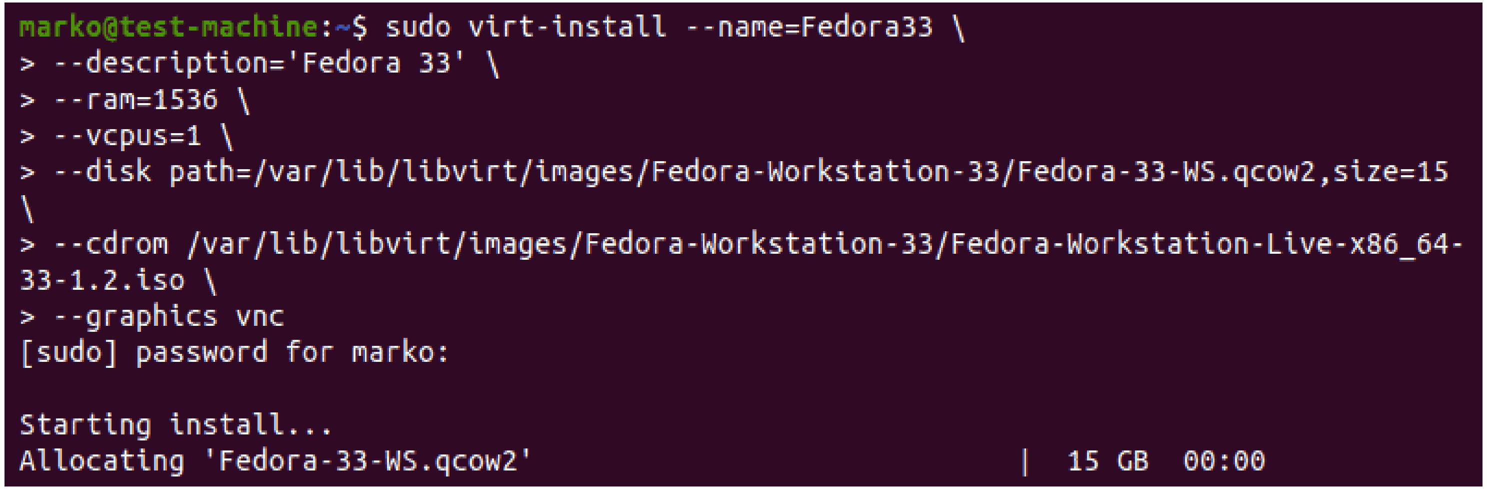
```
sudo systemctl enable --now libvirtd
```

Creating a Virtual Machine using KVM

Before you create a VM,, install virt-manager, a tool for creating and managing VMs:
`sudo apt install virt-manager`

Method 1: Using Command Line

1. Use the `virt-install` command to create a VM via Linux terminal.
   `virt-install --option1=value --option2=value` …



```
marko@test-machine:~$ sudo virt-install --name=Fedora33 \
> --description='Fedora 33' \
> --ram=1536 \
> --vcpus=1 \
> --disk path=/var/lib/libvirt/images/Fedora-Workstation-33/Fedora-33-WS.qcow2,size=15
\
> --cdrom /var/lib/libvirt/images/Fedora-Workstation-33/Fedora-Workstation-Live-x86_64-
33-1.2.iso \
> --graphics vnc
[sudo] password for marko:

Starting install...
Allocating 'Fedora-33-WS.qcow2'                              |  15 GB  00:00
```

Command Line Options

Option	Description
`--name`	The name you give to the VM
`--description`	A short description of the VM
`--ram`	The amount of RAM you wish to allocate to the VM
`--vcpus`	The number of virtual CPUs you wish to allocate to the VM
`--disk`	The location of the VM on your disk (if you specify a qcow2 disk file that does not exist, it will be automatically created)
`--cdrom`	The location of the ISO file you downloaded

More Commands

- To list running virtual machines: **`virsh list`**
- To start a virtual machine: **`virsh start <guestname>`**
- Similarly, to start a virtual machine at boot: **`virsh autostart <guestname>`**
- Reboot a virtual machine with: **`virsh reboot <guestname>`**
- The *state* of virtual machines can be saved to a file in order to be restored later. The following will save the virtual machine state into a file named according to the date: **`virsh save <guestname> save-my.state`**

More Commands

- A saved virtual machine can be restored using:

  ```
  virsh restore save-my.state
  ```
- To shutdown a virtual machine do: `virsh shutdown <guestname>`
- A CDROM device can be mounted in a virtual machine by entering:

  ```
  virsh attach-disk <guestname> /dev/cdrom /media/cdrom
  ```
- To change the definition of a guest virsh exposes the domain :

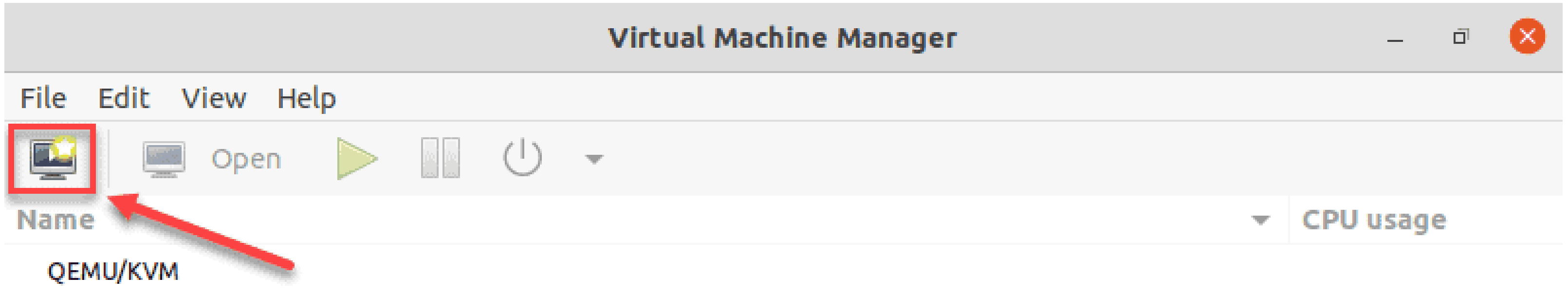
  ```
  virsh edit <guestname>
  ```

Creating a Virtual Machine using KVM

Method 2: Virt Manager GUI

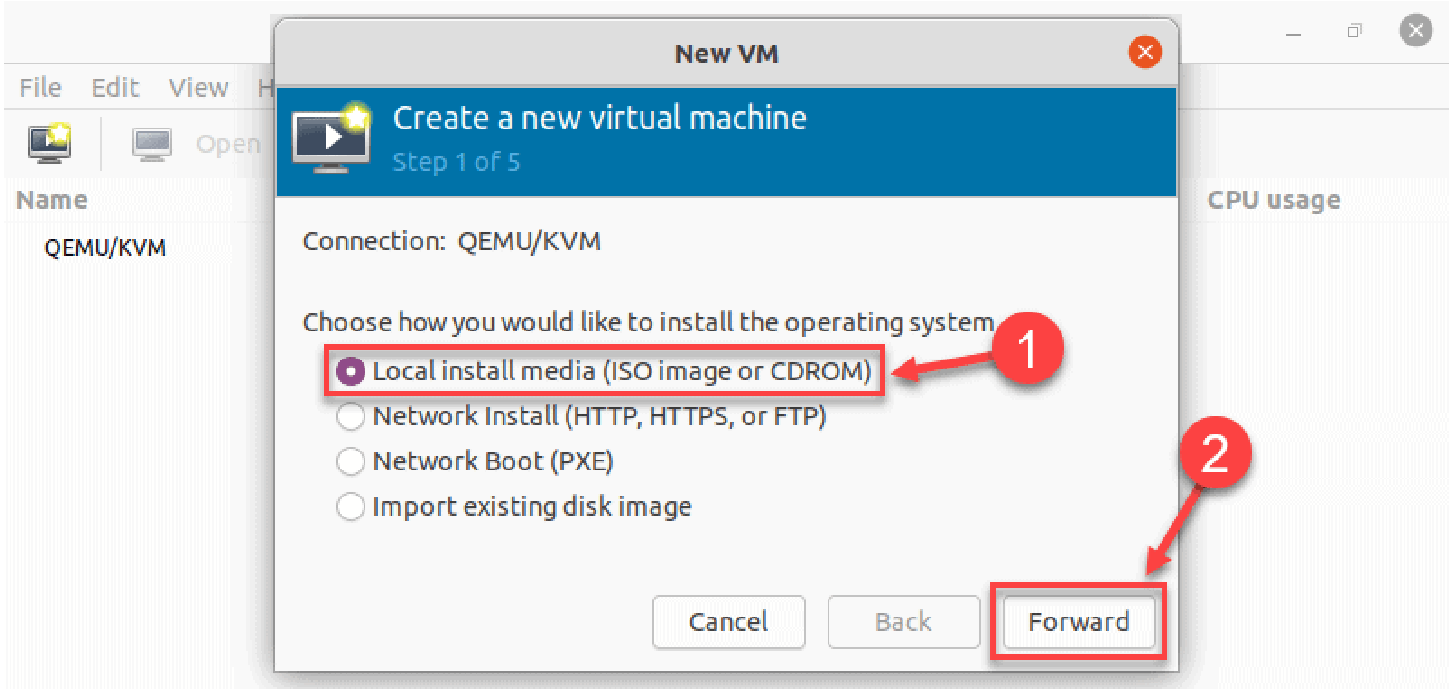
Start virt-manager with: `sudo virt-manager`

In the first window, click the computer icon in the upper-left corner.



Creating a Virtual Machine using KVM

In the dialogue box that opens, select the option to install the VM using an ISO image. Then click **Forward.**



Create a new virtual machine
Step 1 of 5

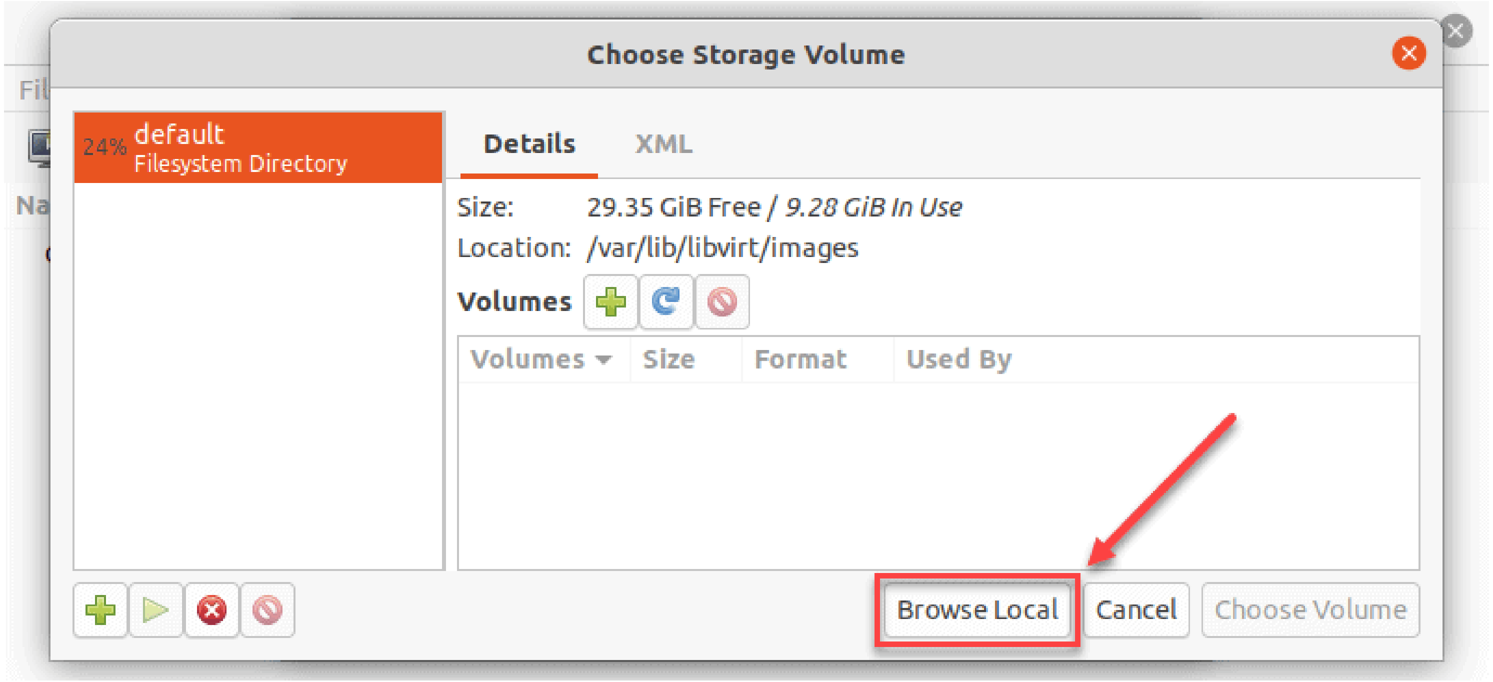
Connection: QEMU/KVM

Choose how you would like to install the operating system

[✓ Local install media (ISO image or CDROM)]
Network Install (HTTP, HTTPS, or FTP)
Network Boot (PXE)
Import existing disk image

Creating a Virtual Machine using KVM

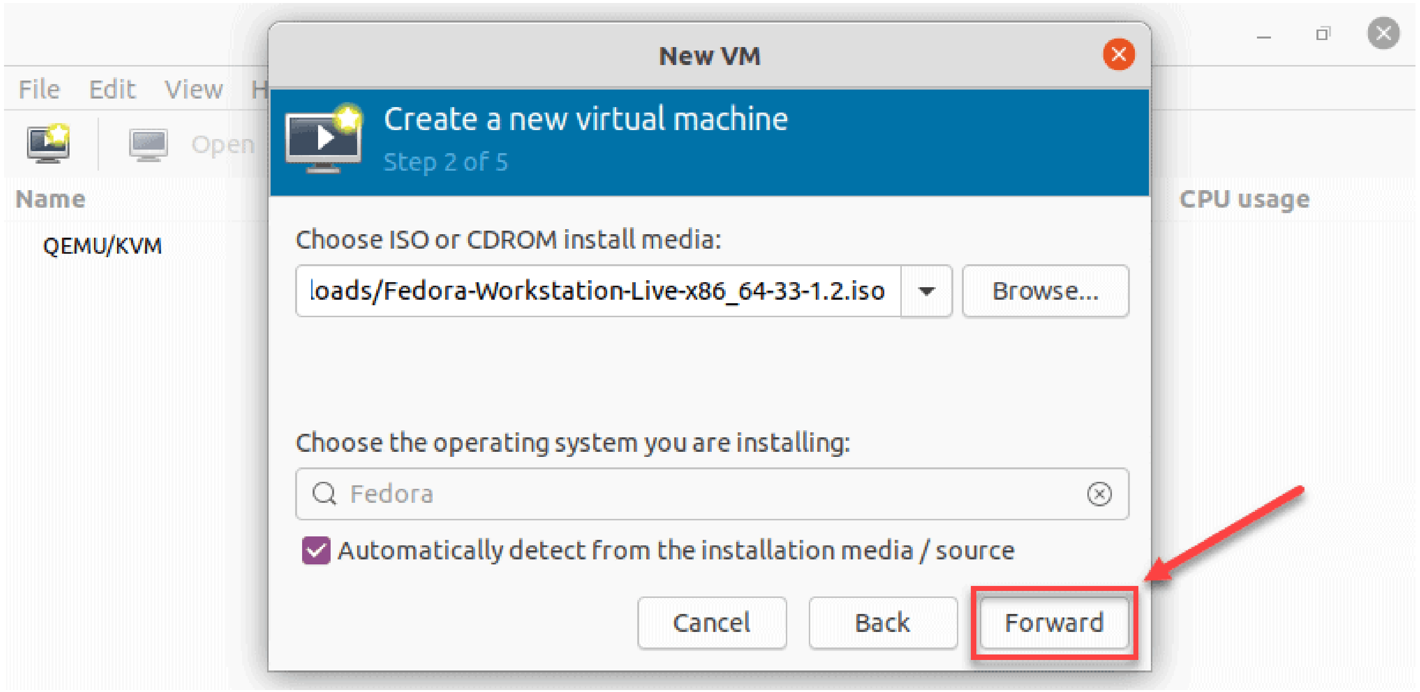
In the next dialogue, click Browse Local and navigate to the path where you stored the ISO you wish to install.



Size: 29.35 GiB Free / *9.28 GiB In Use*
Location: /var/lib/libvirt/images

Creating a Virtual Machine using KVM

The ISO you chose in the previous window populates the field in Step 2. Proceed to Step 3 by clicking **Forward.**



Choose ISO or CDROM install media:

loads/Fedora-Workstation-Live-x86_64-33-1.2.iso

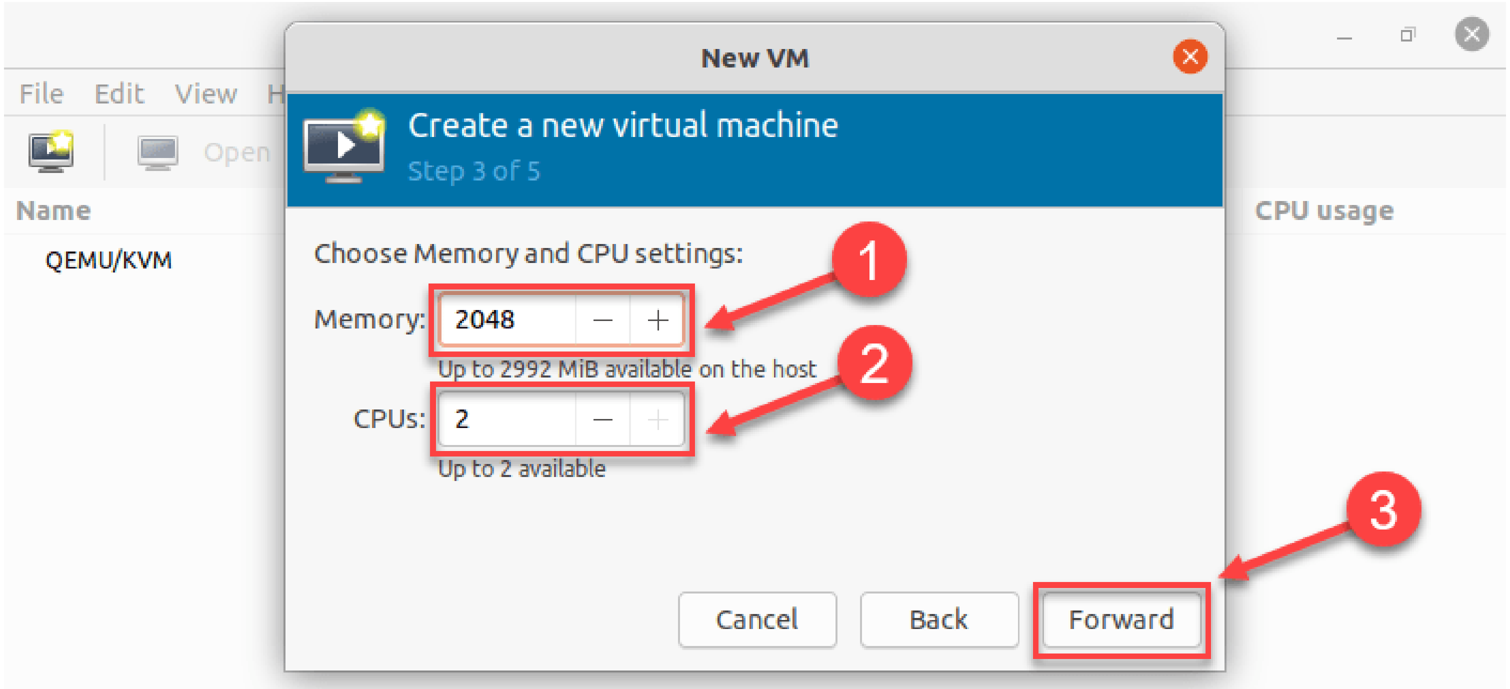
Choose the operating system you are installing:

Fedora

[✓] Automatically detect from the installation media / source

Creating a Virtual Machine using KVM

Enter the amount of RAM and the number of CPUs you wish to allocate to the VM and proceed to the next step.



New VM

Create a new virtual machine

Step 3 of 5

Choose Memory and CPU settings:

Memory: 2048

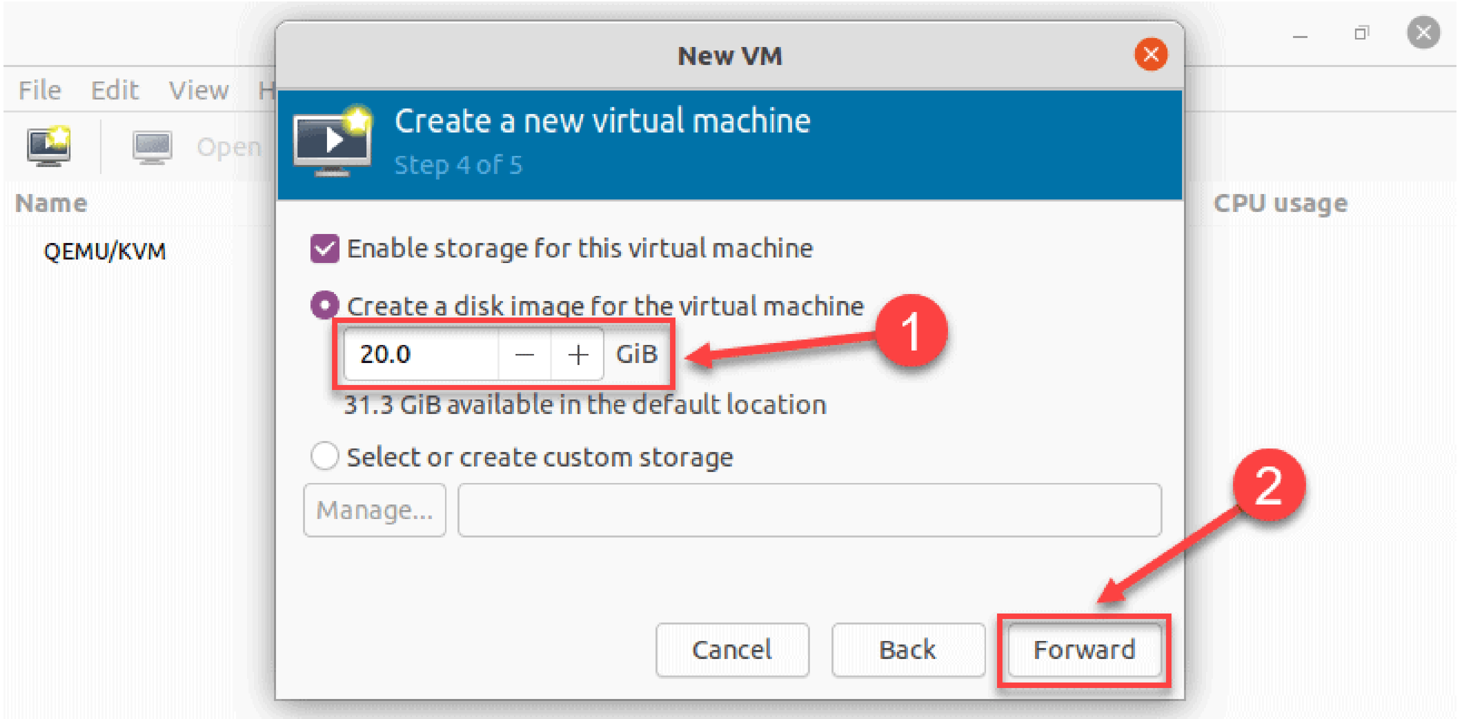
Up to 2992 MiB available on the host

CPUs: 2

Up to 2 available

Creating a Virtual Machine using KVM

Allocate hard disk space to the VM. Click Forward to go to the last step.


New VM

Create a new virtual machine

Step 4 of 5

[✓] Enable storage for this virtual machine

(●) Create a disk image for the virtual machine

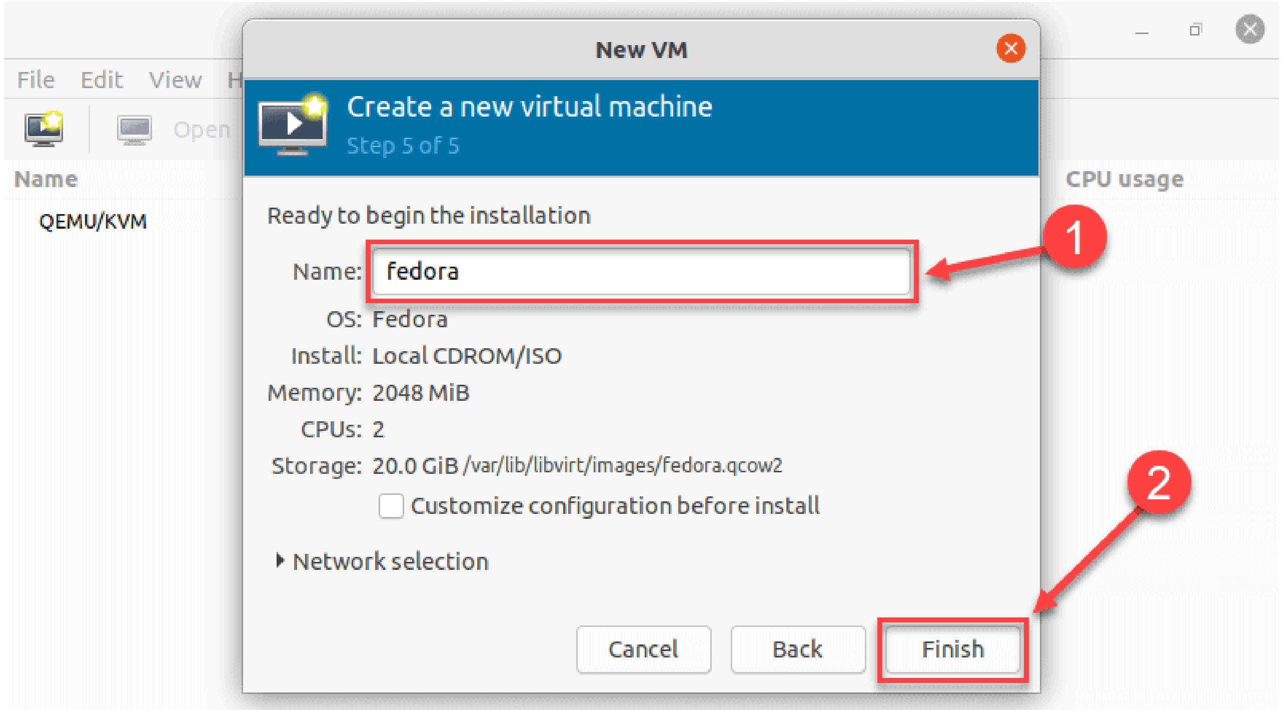
20.0 GiB

31.3 GiB available in the default location

( ) Select or create custom storage

Creating a Virtual Machine using KVM

Specify the name for your VM and click Finish to complete the setup.



Ready to begin the installation

Name: fedora

OS: Fedora

Install: Local CDROM/ISO

Memory: 2048 MiB

CPUs: 2

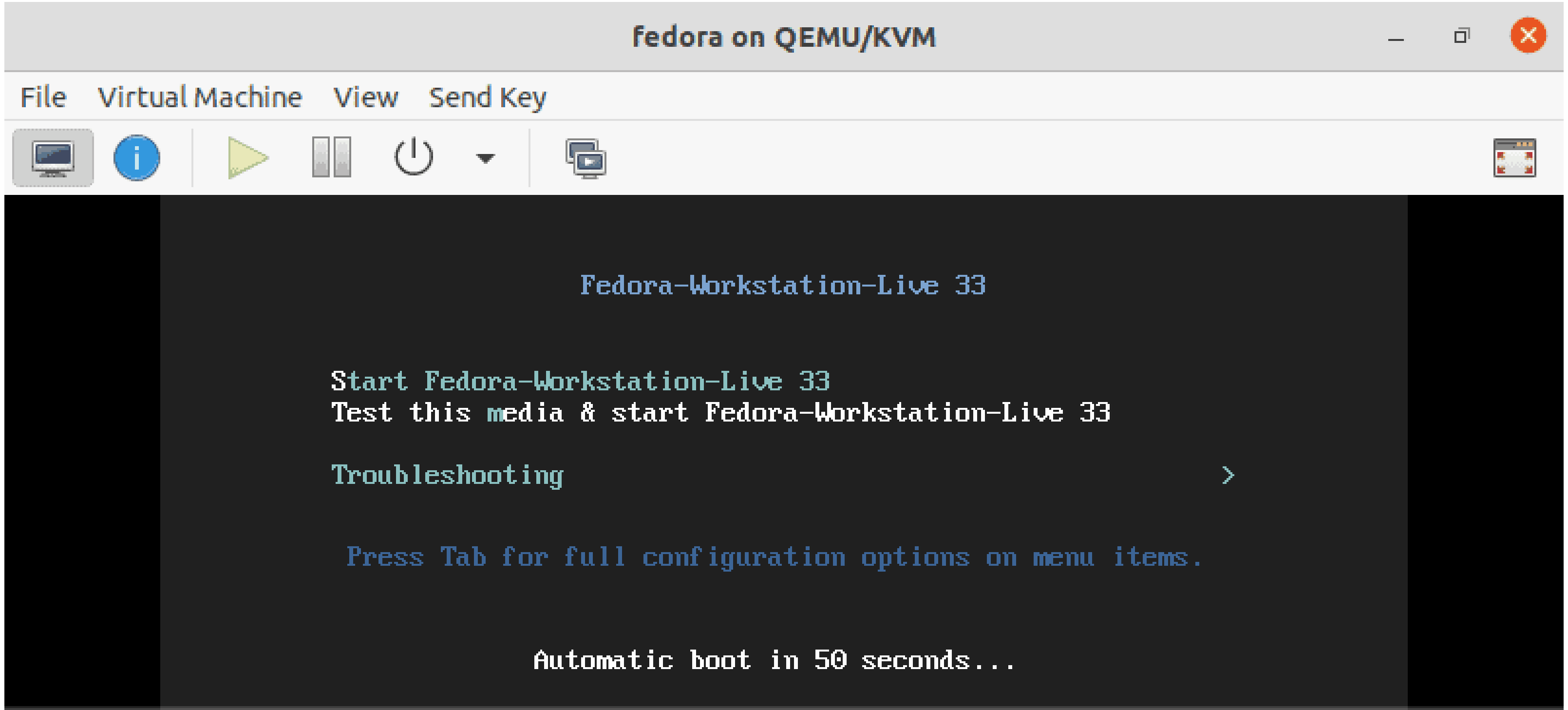
Storage: 20.0 GiB /var/lib/libvirt/images/fedora.qcow2

Customize configuration before install

Network selection

Creating a Virtual Machine using KVM

The VM starts automatically, prompting you to start installing the OS that’s on the ISO file.



```
Fedora-Workstation-Live 33

Start Fedora-Workstation-Live 33
Test this media & start Fedora-Workstation-Live 33

Troubleshooting                                    >

Press Tab for full configuration options on menu items.

Automatic boot in 50 seconds...
```

Live VM Migration

- Live VM Migration can be performed based on two basic network scenarios: Local Area Network (LAN) and Wide Area Network (WAN).
- Live Migration over LAN is easier for two reasons.
  - high- speed low-latency links in the LAN which makes migration comparatively quicker
  - the VM can retain its IP address (es) after migration since the hosts share the same IP address space.

Benefits of Live VM Migration

- **Consolidation:** Several underutilized small data centers can be replaced with few larger ones.
- **Load balancing:** It requires the transfer of VM's from an overloaded host to a light loaded ones.
- **Scaling:** Multiple sites need to be created at different geographical sites to scale up as the cloud grows.
- **Disaster recovery and reliability:** In times of catastrophe or any fault occurrence, VM's can be migrated to mirrored sites across cloud with minimal downtime.
- **Maintenance:** Applications and data can be migrated to another machine to free up the hardware for maintenance.

Process of Live Migration

Design Considerations

At a high level we can consider a virtual machine to encapsulate access to a set of physical resources. Providing live migration of these VMs in a clustered server environment leads us to focus on the physical resources used in such environments:

- Memory
- Network
- Disk

Migrating Memory

When a VM is running a live service it is important that this transfer occurs in a manner that balances the requirements of minimizing both downtime and total migration time. It is easiest to consider the trade-offs between these requirements by generalizing memory transfer into three phases:

1. Push phase : The source VM continues running while certain pages are pushed across the network to the new destination. To ensure consistency, pages modified during this process must be re-sent.

2. Stop-and-copy phase : The source VM is stopped, pages are copied across to the destination VM, then the new VM is started.

3. Pull phase : The new VM executes and, if it accesses a page that has not yet been copied, this page is faulted in (“pulled”) across the network from the source VM.

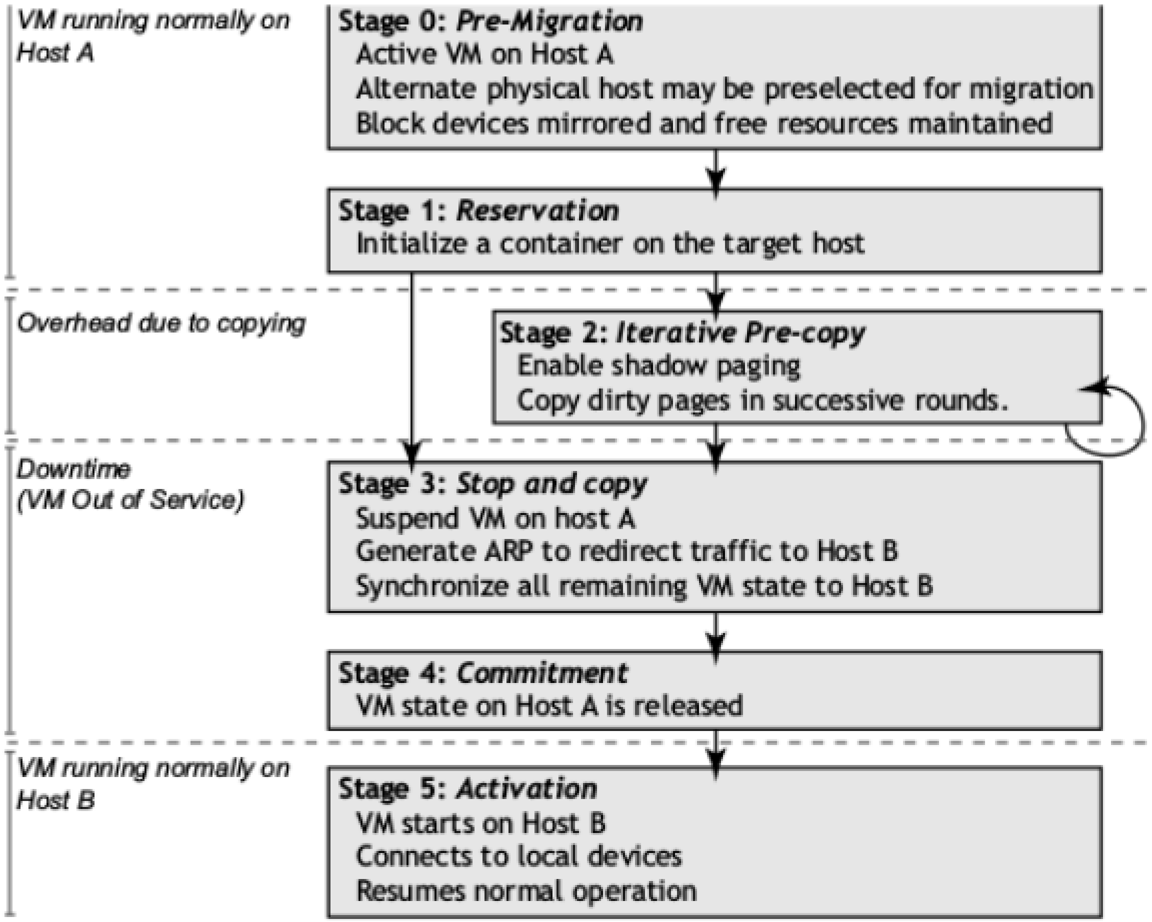
Local Resources

A key challenge in managing the migration of OS instances is what to do about resources that are associated with the physical machine that they are migrating away from. While memory can be copied directly to the new host, connections to local devices such as disks and network interfaces demand additional consideration.

Two Key problems :

- Network resources
- Local storage

How It Works


VM running normally on Host A
Stage 0: Pre-Migration
Active VM on Host A
Alternate physical host may be preselected for migration
Block devices mirrored and free resources maintained
Stage 1: Reservation
Initialize a container on the target host
Overhead due to copying
Stage 2: Iterative Pre-copy
Enable shadow paging
Copy dirty pages in successive rounds.
Downtime (VM Out of Service)
Stage 3: Stop and copy
Suspend VM on host A
Generate ARP to redirect traffic to Host B
Synchronize all remaining VM state to Host B
Stage 4: Commitment
VM state on Host A is released
VM running normally on Host B
Stage 5: Activation
VM starts on Host B
Connects to local devices
Resumes normal operation