

CLOUD SECURITY & MANAGEMENT
LAB

Name: Harsh Ranjan

SAP ID: 500097019

Roll no: R2142211262

Batch :B7

SUBMISSION TO:

Ms. Avita Katal

EXP 4: To install and configure virtualization using KVM (Kernel-based Virtual Machine) on a Linux system.

STEP 1: Check Hardware and Software Requirements:

- Ensure that your CPU supports hardware virtualization (Intel VT-x or AMD-V).
- Verify that virtualization extensions are enabled in the BIOS settings.
- Make sure your Linux distribution supports KVM and has the necessary kernel modules installed.
- Check Hardware Virtualization Support: Ensure that your CPU supports hardware virtualization. You can check this by running the following command:

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



STEP 2: Install KVM Packages:

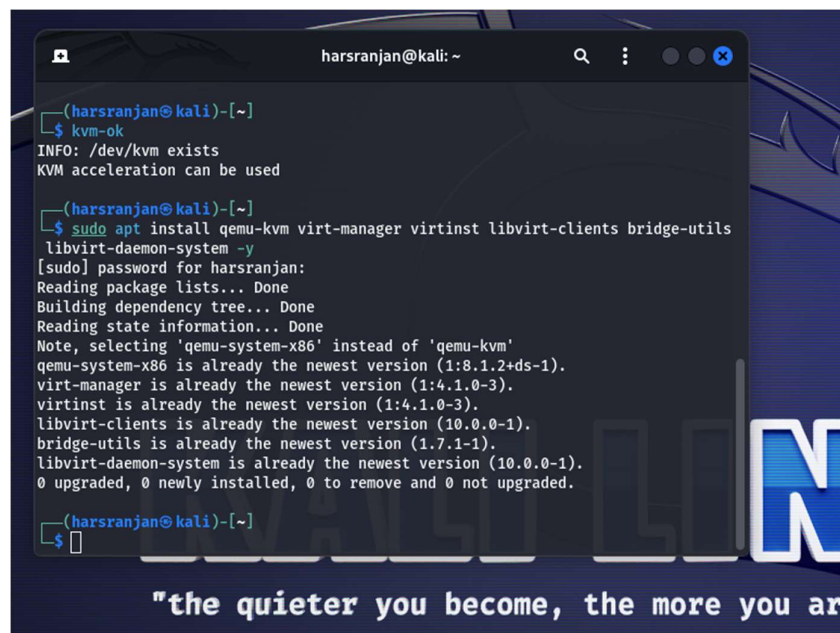
- Update your system's package repositories.
- Install the necessary KVM packages. On Debian-based systems (e.g., Ubuntu), you can use:

```
sudo apt update
```

```
sudo apt install qemu-kvm libvirt-clients libvirt-daemon-system bridge-utils virtinst  
libvirt-daemon
```

On Red Hat-based systems (e.g., CentOS, RHEL), you can use:

```
sudo yum install qemu-kvm libvirt libvirt-python libguestfs-tools virt-install
```

A terminal window titled 'harsranjan@kali: ~' showing the execution of 'kvm-ok' and 'sudo apt install' commands. The 'kvm-ok' command outputs 'INFO: /dev/kvm exists' and 'KVM acceleration can be used'. The 'apt install' command lists several packages as already being the newest version. At the bottom of the terminal window, there is a quote: "the quieter you become, the more you are".

```
(harsranjan@kali)-[~]  
$ kvm-ok  
INFO: /dev/kvm exists  
KVM acceleration can be used  
  
(harsranjan@kali)-[~]  
$ sudo apt install qemu-kvm virt-manager virtinst libvirt-clients bridge-utils  
libvirt-daemon-system -y  
[sudo] password for harsranjan:  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
Note, selecting 'qemu-system-x86' instead of 'qemu-kvm'  
qemu-system-x86 is already the newest version (1:8.1.2+ds-1).  
virt-manager is already the newest version (1:4.1.0-3).  
virtinst is already the newest version (1:4.1.0-3).  
libvirt-clients is already the newest version (10.0.0-1).  
bridge-utils is already the newest version (1.7.1-1).  
libvirt-daemon-system is already the newest version (10.0.0-1).  
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.  
  
(harsranjan@kali)-[~]  
$
```

STEP 3. Start and Enable Libvirt Service:

- Start the libvirtd service:

```
sudo systemctl start libvirtd
```
- Enable the libvirtd service to start automatically on boot:

```
sudo systemctl enable libvirtd
```

STEP 4. Verify Installation:

- After installing the packages, you can verify that KVM is working correctly by running:

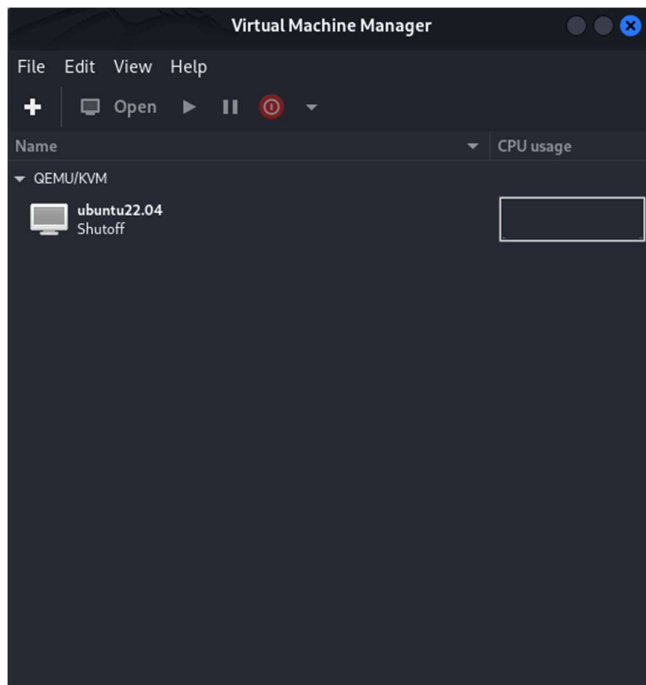
```
kvm-ok
```

This command should output "INFO: /dev/kvm exists" if everything is set up correctly.



STEP 5: Use tools like *virt-install* or *virt-manager* to create and manage virtual machines.

You can use a command-line interface or a graphical interface, depending on your preference.



STEP 6: Configure Storage and Advanced Settings

- Configure storage pools to store virtual machine disk images.
- Decide on the Type of Storage Pool:
Determine the type of storage pool you want to create based on your requirements. Common types include directory-based, logical volume-based, and network-based storage pools.
Create a Directory-Based Storage Pool:

Directory-based storage pools store disk images in a directory on the host

Create a directory-based storage pool

```
virsh pool-define-as mypool dir - - - "/path/to/storage"
```

Start the storage pool

```
virsh pool-start mypool
```

Autostart the storage pool

```
virsh pool-autostart mypool
```

Refresh the storage pool

```
virsh pool-refresh mypool
```

Verify the storage pool status

```
virsh pool-info mypool
```

```
(harsranjan@kali)-[~]
$ virsh pool-define-as harshpool dir - - - ~/Documents/Exp4
Pool harshpool defined

(harsranjan@kali)-[~]
$ virsh pool-start harshpool
Pool harshpool started

(harsranjan@kali)-[~]
$
```

```
(harsranjan@kali)-[~]
$ virsh pool-info harshpool
Name:          harshpool
UUID:          453a4764-0dfd-4466-ad8a-01ea8f72c043
State:         running
Persistent:    yes
Autostart:     no
Capacity:      86.40 GiB
Allocation:    36.07 GiB
Available:     50.33 GiB

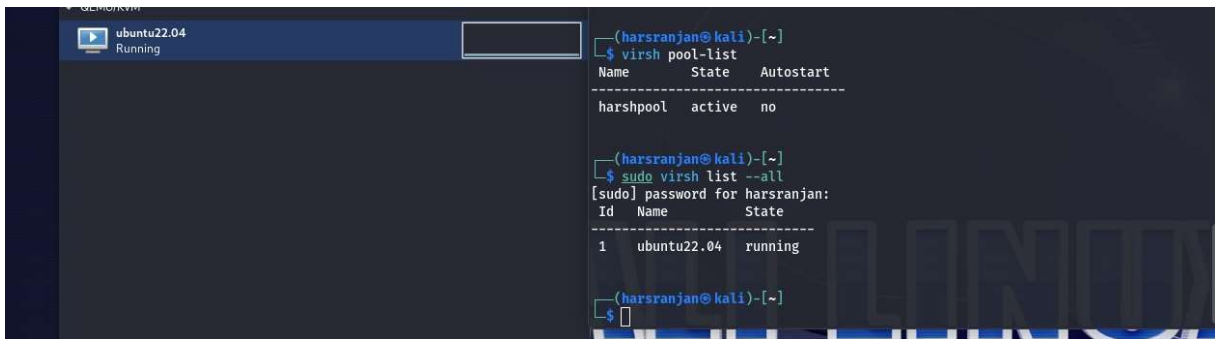
(harsranjan@kali)-[~]
$ virsh pool-refresh harshpool
Pool harshpool refreshed
```

STEP 7: Configure CPU Pinning:

You can configure CPU pinning either by directly editing the XML definition of each virtual machine or using the virsh command-line tool. Here's how to do it using virsh:

Get the UUID of the virtual machine

virsh list --all



```
(harsranjan@kali)~  
$ virsh pool-list  
Name      State    Autostart  
-----  
harshpool  active  no  
  
(harsranjan@kali)~  
$ sudo virsh list --all  
[sudo] password for harsranjan:  
Id Name      State  
--  
1  ubuntu22.04 running  
  
(harsranjan@kali)~  
$
```

Edit the XML definition of the virtual machine

virsh edit <domain-uuid>

This command will open the XML definition of the virtual machine in a text editor. Look for the `<vcpu>` element, which specifies the number of virtual CPUs allocated to the virtual machine. You can use CPU pinning by adding the `<vcpupin>` elements to specify the physical CPU cores to be used.

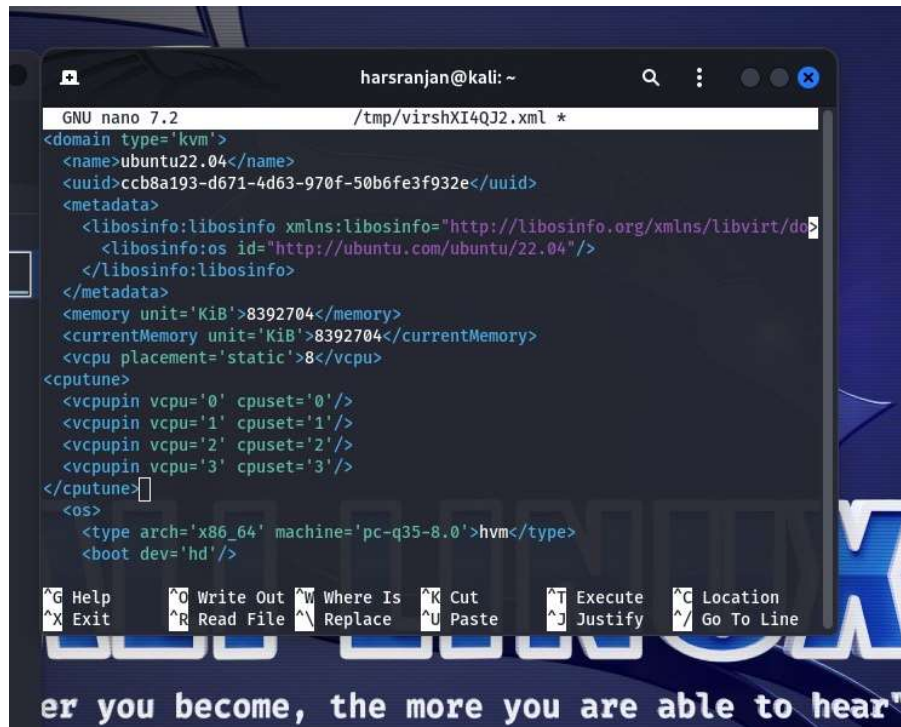
Example XML snippet for CPU pinning:

```
<vcpu placement='static'>4</vcpu>  
<cputune>  
  <vcpupin vcpu='0' cpuset='0'/>  
  <vcpupin vcpu='1' cpuset='1'/>  
  <vcpupin vcpu='2' cpuset='2'/>  
  <vcpupin vcpu='3' cpuset='3'/>  
</cputune>
```

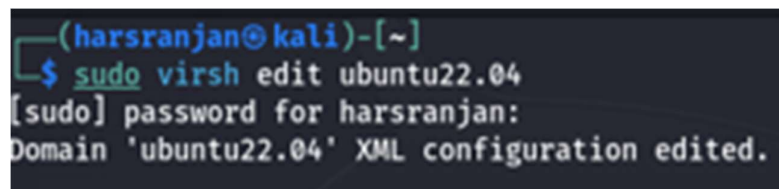
In this example, we have a virtual machine with 4 virtual CPUs, and each virtual CPU is pinned to a specific physical CPU core.

Apply Changes:

Save the changes to the XML definition and exit the text editor. Then, restart the virtual machine for the changes to take effect.



```
GNU nano 7.2 /tmp/virshXI4QJ2.xml *
<domain type='kvm'>
  <name>ubuntu22.04</name>
  <uuid>ccb8a193-d671-4d63-970f-50b6fe3f932e</uuid>
  <metadata>
    <libosinfo:libosinfo xmlns:libosinfo='http://libosinfo.org/xmlns/libvirt/domain/1.0'>
      <libosinfo:os id='http://ubuntu.com/ubuntu/22.04'>
    </libosinfo:libosinfo>
  </metadata>
  <memory unit='KiB'>8392704</memory>
  <currentMemory unit='KiB'>8392704</currentMemory>
  <vcpu placement='static'>8</vcpu>
  <cpu>
    <vcpupin vcpu='0' cpuset='0'>
    </vcpupin>
    <vcpupin vcpu='1' cpuset='1'>
    </vcpupin>
    <vcpupin vcpu='2' cpuset='2'>
    </vcpupin>
    <vcpupin vcpu='3' cpuset='3'>
    </vcpupin>
  </cpu>
  <os>
    <type arch='x86_64' machine='pc-q35-8.0'>hvm</type>
    <boot dev='hd'>
  </os>
</domain>
```



```
(harsranjan@kali)-[~]
$ sudo virsh edit ubuntu22.04
[sudo] password for harsranjan:
Domain 'ubuntu22.04' XML configuration edited.
```

Restart the virtual machine

virsh reboot <domain-uuid>

Verify CPU Pinning:

After restarting the virtual machine, you can verify that CPU pinning is applied correctly using tools like **top**, **htop**, or **lscpu** within the virtual machine to see which CPU cores are being utilized.


```
Applications Places Wof Feb 21 4:33:00 PM
harsranjan@kali: ~
168 root -51 0 0 0 0 5 4.3 0.0 0:23.93 irq/33-ACPI:Event
7525 harsran+ 20 0 781472 55796 42536 R 4.3 0.4 0:00.45 gnome-terminal-
5727 libvirt+ 20 0 14.0g 3.9g 30336 S 1.7 25.9 4:14.21 qemu-system-x86
15 root 20 0 0 0 0 0 5 0.9 0.0 0:00.12 ksofirqd/0
233 root -51 0 0 0 0 5 0.9 0.0 0:04.85 irq/45-PNPAC90:Re
338 root -2 0 0 0 0 0 5 0.9 0.0 0:00.10 gfx_low
5465 root 20 0 1713884 61248 24936 S 0.9 0.4 0:05.02 libvirt
6575 root 20 0 0 0 0 0 0 0.9 0.0 0:00.39 kworker/u32:2+events_unbound
7567 harsran+ 20 0 11988 5632 3456 R 0.9 0.0 0:00.05 top
1 root 20 0 22944 13024 9312 S 0.0 0.1 0:02.01 systemd
2 root 20 0 0 0 0 0 5 0.0 0.0 0:00.01 kthread
3 root 0 -20 0 0 0 0 1 0.0 0.0 0:00.00 rcu_gp
top - 16:32:57 up 1:04, 1 user, load average: 0.43, 0.49, 0.42
Tasks: 340 total, 2 running, 338 sleeping, 0 stopped, 0 zombie
%Cpu0 : 2.0 us, 1.3 sy, 0.0 ni, 96.0 id, 0.0 wa, 0.0 hi, 0.7 si, 0.0 st
%Cpu1 : 3.3 us, 2.0 sy, 0.0 ni, 94.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 2.2 us, 0.7 sy, 0.0 ni, 97.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu4 : 0.3 us, 1.0 sy, 0.0 ni, 98.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu5 : 1.3 us, 1.7 sy, 0.0 ni, 97.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu6 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu7 : 0.0 us, 5.9 sy, 0.0 ni, 94.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu8 : 0.3 us, 0.3 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu9 : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu10 : 0.0 us, 0.0 sy, 0.0 ni, 99.7 id, 0.3 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu11 : 0.0 us, 0.7 sy, 0.0 ni, 98.0 id, 1.3 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 15341.1 total, 353.8 free, 6844.8 used, 8520.0 buff/cache
MiB Swap: 9537.0 total, 9535.7 free, 1.2 used, 8496.2 avail Mem

PID USER PR NI VIRT RES SHR S CPU% MEM% TIME+ COMMAND
168 root -51 0 0 0 0 5 3.3 0.0 0:24.26 irq/33-ACPI:Event
1796 harsran+ 20 0 114448 150768 80368 S 5.3 1.0 1:09.16 Xorg
7525 harsran+ 20 0 772276 55796 42536 R 4.0 0.4 0:00.40 gnome-terminal-
2029 harsran+ 20 0 5007012 372752 136052 S 3.7 2.4 1:42.08 gnome-shell
5727 libvirt+ 20 0 13.9g 3.9g 30336 S 2.0 25.9 4:14.30 qemu-system-x86
233 root -51 0 0 0 0 5 1.0 0.0 0:04.91 irq/45-PNPAC90:Re
5465 root 20 0 1631916 108372 62476 S 0.3 1.2 0:13.72 virt-manager
6575 root 20 0 0 0 0 0 1 0.3 0.0 0:00.30 kworker/2:0+events
7567 harsran+ 20 0 11988 5632 3456 R 0.3 0.0 0:00.00 top
1 root 20 0 22944 13024 9312 S 0.0 0.1 0:02.01 systemd
2 root 20 0 0 0 0 0 5 0.0 0.0 0:00.01 kthread
3 root 0 -20 0 0 0 0 1 0.0 0.0 0:00.00 rcu_gp
4 root 0 -20 0 0 0 0 1 0.0 0.0 0:00.00 rcu_par_gp
```

```
top - 16:32:57 up 1:04, 1 user, load average: 0.43, 0.49, 0.42
Tasks: 340 total, 2 running, 338 sleeping, 0 stopped, 0 zombie
%Cpu0 : 2.0 us, 1.3 sy, 0.0 ni, 96.0 id, 0.0 wa, 0.0 hi, 0.7 si, 0.0 st
%Cpu1 : 3.3 us, 2.0 sy, 0.0 ni, 94.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 2.2 us, 0.7 sy, 0.0 ni, 97.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu4 : 0.3 us, 1.0 sy, 0.0 ni, 98.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu5 : 1.3 us, 1.7 sy, 0.0 ni, 97.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu6 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu7 : 0.0 us, 5.9 sy, 0.0 ni, 94.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu8 : 0.3 us, 0.3 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu9 : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu10 : 0.0 us, 0.0 sy, 0.0 ni, 99.7 id, 0.3 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu11 : 0.0 us, 0.7 sy, 0.0 ni, 98.0 id, 1.3 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 15341.1 total, 353.8 free, 6844.8 used, 8520.0 buff/cache
MiB Swap: 9537.0 total, 9535.7 free, 1.2 used, 8496.2 avail Mem
```

Q1. What is a kernel in OS?

ANSWER: The kernel is the core component of an operating system (OS) that manages system resources and provides essential services to higher-level software. It acts as an intermediary between software applications and the hardware of the computer. The kernel handles tasks such as memory management, process scheduling, device management, and system calls.

Q2. Define KVM and explain how it differs from other types of hypervisors.

ANSWER: KVM is an open-source virtualization technology for Linux that allows the Linux kernel to act as a Type 1 hypervisor. Unlike traditional hypervisors, which run as separate software on top of the host operating system (Type 2 hypervisor), KVM is integrated directly into the Linux kernel. This integration provides better performance and efficiency compared to Type 2 hypervisors.

Q3. Is it possible for the Windows kernel to function as a Type 1 hypervisor, and are there analogous solutions similar to

ANSWER: While the Windows kernel primarily functions as part of the operating system, Microsoft offers a solution called Hyper-V, which allows the Windows kernel to operate as a Type 1 hypervisor. Hyper-V enables running multiple virtual machines directly on top of the Windows kernel.

Q4. What are hardware virtualization extensions, and how do they enhance virtualization performance in KVM?

ANSWER: Hardware virtualization extensions, such as Intel VT-x and AMD-V, are features built into modern processors that enhance virtualization performance. These extensions allow virtual machines to run more efficiently by providing hardware-level support for virtualization, including features like nested paging and virtualization-assisted I/O.

Q5. Describe the role of QEMU in conjunction with KVM and how they work together.

ANSWER: QEMU (Quick Emulator) is an open-source emulator that provides hardware virtualization for various guest operating systems. It can be used in conjunction with KVM to accelerate virtual machine performance by leveraging hardware virtualization extensions. QEMU emulates hardware devices and provides virtualized interfaces to guest operating systems.

Q6. Explain the significance of KVM being integrated into the Linux kernel. How does this integration benefit virtualization on Linux?

ANSWER: The integration of KVM into the Linux kernel offers several benefits for virtualization on Linux:

- Improved Performance: KVM benefits from being tightly integrated with the kernel, resulting in better performance compared to standalone hypervisors.

- **Simplified Deployment:** Since KVM is included in the Linux kernel, there's no need to install additional software or modules to enable virtualization.
- **Tighter Integration:** KVM can take advantage of kernel features and optimizations, leading to greater efficiency and scalability.
- **Better Compatibility:** KVM supports a wide range of hardware platforms and Linux distributions, ensuring broad compatibility and ease of use.

Q7. What is Libvirt, and how does it contribute to the management of KVM-based virtual machines?

ANSWER: Libvirt is an open-source API and management tool for managing virtualization platforms, including KVM. It provides a common interface for performing tasks such as creating, configuring, and monitoring virtual machines. Libvirt abstracts the underlying virtualization technology, allowing administrators to manage virtual machines across different hypervisors through a unified interface.

Q8. Discuss the performance benefits of using KVM for virtualization. How does it achieve near-native performance for virtual machines?

ANSWER: KVM offers several performance benefits for virtualization, including:

- **Near-Native Performance:** KVM achieves near-native performance by leveraging hardware virtualization extensions and optimized kernel integration.
- **Minimal Overhead:** Since KVM is integrated into the Linux kernel, it incurs minimal overhead compared to standalone hypervisors.
- **Efficient Resource Utilization:** KVM dynamically allocates resources to virtual machines, optimizing resource utilization and maximizing performance.
- **Scalability:** KVM supports large-scale virtualization deployments with support for multi-core processors, large memory configurations, and advanced networking features.