

EXPERIMENT NO: 12

DATE:

HYPERVERSORS AND VIRTUAL MACHINE

AIM: FAMILIARISATION TO HYPERVERSORS AND VIRTUAL MACHINES

VIRTUAL MACHINE:

A virtual machine is a virtual representation, or emulation, of a physical computer. They are often referred to as a guest while the physical machine they run on is referred to as the host. Virtualization makes it possible to create multiple virtual machines, each with their own operating system (OS) and applications, on a single physical machine. A VM cannot interact directly with a physical computer. Instead, it needs a lightweight software layer called a hypervisor to coordinate between it and the underlying physical hardware.

HYPERVERSOR:

Hypervisor is a software program that manages multiple operating systems (or multiple instances of the same operating system) on a single computer system. The hypervisor manages the system's processor, memory, and other resources to allocate what each operating system requires. Hypervisors are designed for a particular processor architecture and may also be called virtualization managers.

HYPERVERSOR TYPES

Type 1: native (bare-metal) hypervisors

The Hypervisor runs directly on the host's hardware to control the hardware and to manage guest operating systems.

E.g., Xen, VMWare ESXi, Microsoft Hyper-V

Type 2: hosted hypervisors

These hypervisors run on a conventional operating system just as other computer programs do. Eg: VMWare Workstation, VirtualBox

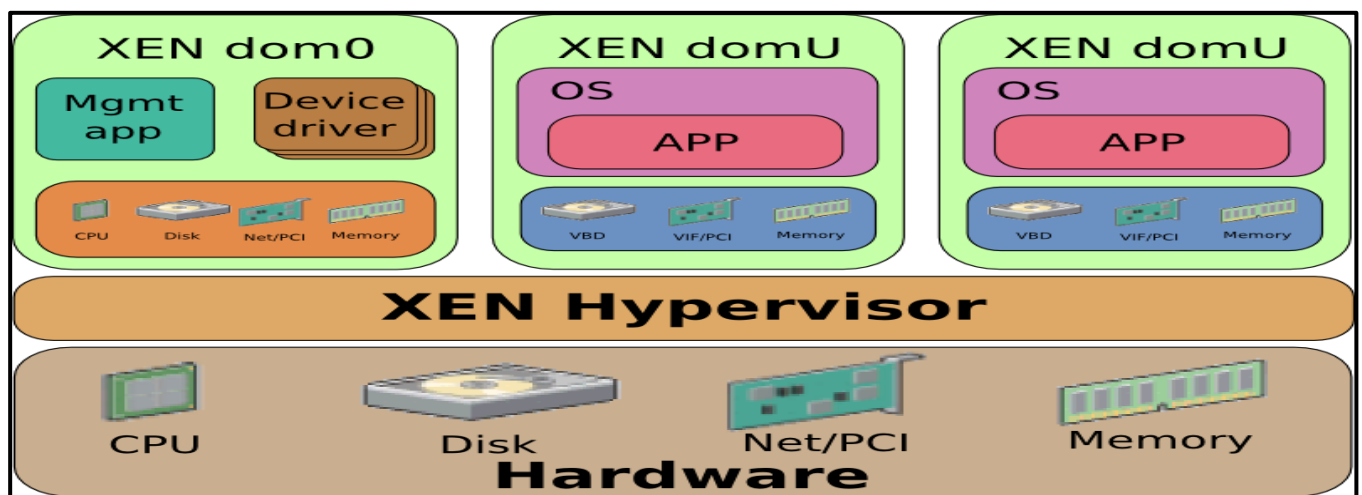
BENEFITS OF HYPERVISOR

- 1) Speed
- 2) Efficiency 3) Flexibility
- 3) Portability

- **XEN OR KVM**

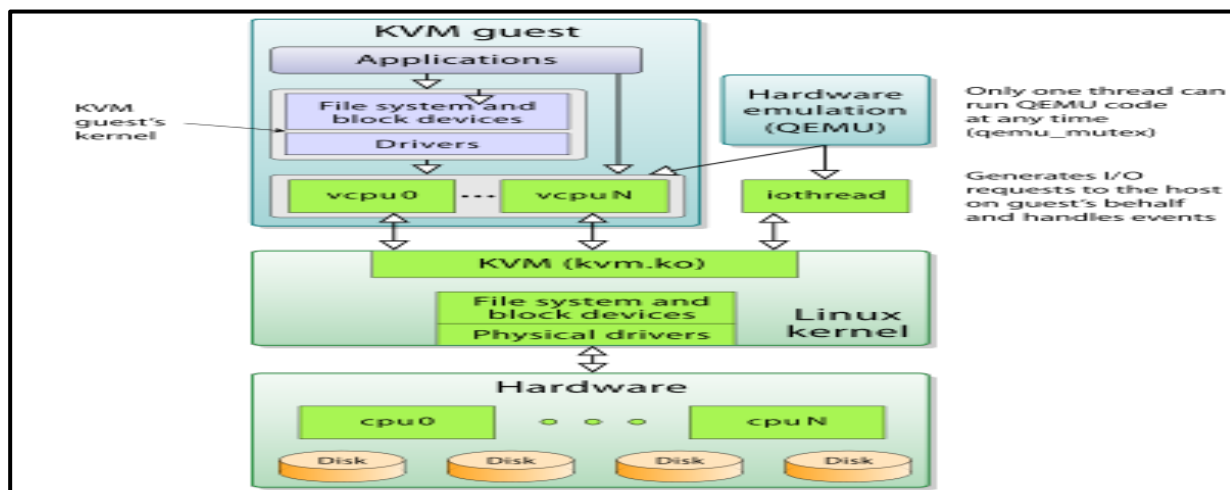
XEN

Xen is an open-source paravirtualization technology that provides a platform for running multiple operating systems in parallel on one physical hardware resource.



KERNEL-BASED VIRTUAL MACHINE (KVM)

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). 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.



DOCKER

Docker overview

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

What is a container?

- Docker provides the ability to package and run an application in a loosely isolated environment called a container.
- The isolation and security allow you to run many containers simultaneously on a given host.
- Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host.
- You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

What is a Docker image?

- A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template.
- Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.

The Role of Images and Containers



Docker Image

Example: Ubuntu with Node.js and
Application Code

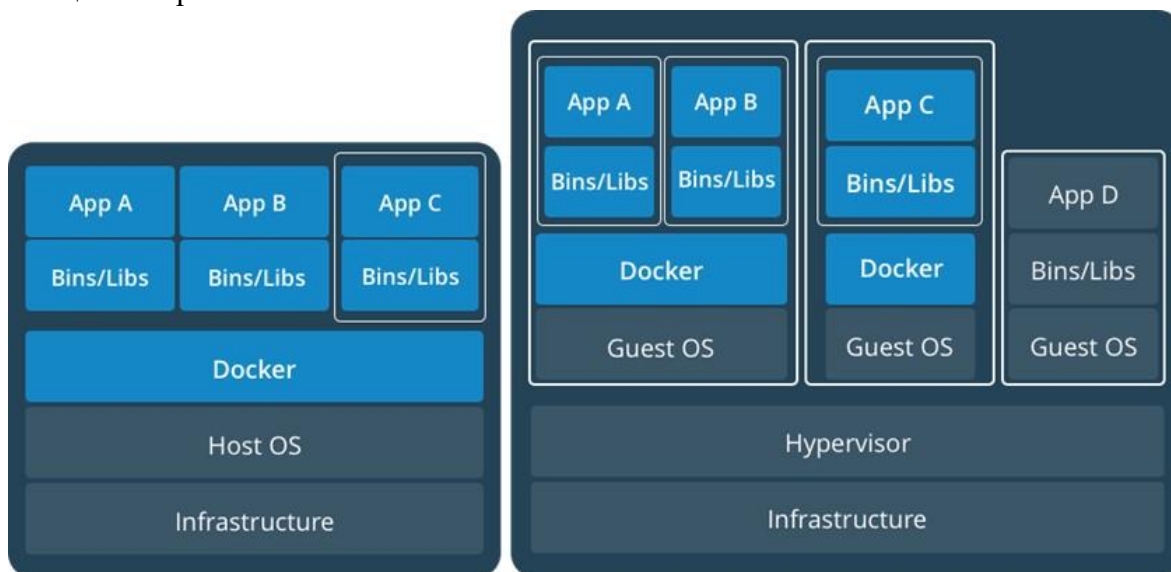


Docker Container

Created by using an image. Runs your
application.

Docker installation

---->\$ sudo apt install docker.io



```
user@user-HP-Laptop-15-da0xxx: ~  
user@user-HP-Laptop-15-da0xxx:~$ sudo apt install docker.io  
[sudo] password for user:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  bridge-utils containerd pigz runc ubuntu-fan  
Suggested packages:  
  ifupdown aufs-tools btrfs-progs cgroupfs-mount | cgroup-lite debootstrap  
  docker-doc rinse zfs-fuse | zfsutils  
The following NEW packages will be installed:  
  bridge-utils containerd docker.io pigz runc ubuntu-fan  
0 upgraded, 6 newly installed, 0 to remove and 373 not upgraded.  
Need to get 74.0 MB of archives.  
After this operation, 359 MB of additional disk space will be used.  
Do you want to continue? [Y/n] y  
Get:1 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 pigz amd64 2.4-1  
[57.4 kB]  
Get:2 http://in.archive.ubuntu.com/ubuntu focal/main amd64 bridge-utils amd64 1.  
6-2ubuntu1 [30.5 kB]  
Get:3 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 runc amd64 1.  
0.0~rc95-0ubuntu1~20.04.2 [4,087 kB]  
Get:4 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 containerd am  
d64 1.5.2-0ubuntu1~20.04.2 [32.9 MB]
```

```
user@user-HP-Laptop-15-da0xxx: ~  
Selecting previously unselected package ubuntu-fan.  
Preparing to unpack .../5-ubuntu-fan_0.12.13_all.deb ...  
Unpacking ubuntu-fan (0.12.13) ...  
Setting up runc (1.0.0~rc95-0ubuntu1~20.04.2) ...  
Setting up bridge-utils (1.6-2ubuntu1) ...  
Setting up pigz (2.4-1) ...  
Setting up containerd (1.5.2-0ubuntu1~20.04.2) ...  
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service →  
/lib/systemd/system/containerd.service.  
Setting up ubuntu-fan (0.12.13) ...  
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service →  
/lib/systemd/system/ubuntu-fan.service.  
Setting up docker.io (20.10.7-0ubuntu1~20.04.1) ...  
Adding group 'docker' (GID 135) ...  
Done.  
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /li  
b/systemd/system/docker.service.  
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/sy  
stemd/system/docker.socket.  
Processing triggers for systemd (245.4-4ubuntu3.11) ...  
Processing triggers for man-db (2.9.1-1) ...  
user@user-HP-Laptop-15-da0xxx:~$
```

Version check

----->\$ docker --version

```
user@user-HP-Laptop-15-da0xxx:~$ docker --version
Docker version 20.10.7, build 20.10.7-0ubuntu1~20.04.1
user@user-HP-Laptop-15-da0xxx:~$
```

Check whether it is running or not

----->\$ sudo systemctl status checker

If not active

----->\$ sudo systemctl enable --now docker

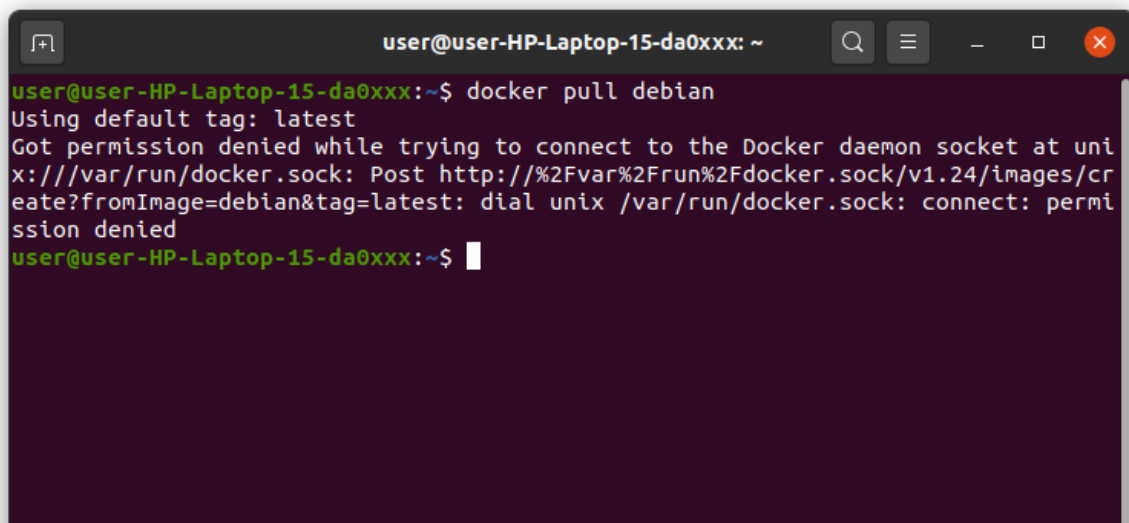
List all the images you have locally

----->\$ sudo docker images

```
user@user-HP-Laptop-15-da0xxx:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID   CREATED   SIZE
user@user-HP-Laptop-15-da0xxx:~$
```

Pull an image from the Docker registry

---->\$ sudo docker pull <image_name>:<tag>



```
user@user-HP-Laptop-15-da0xxx: ~
user@user-HP-Laptop-15-da0xxx:~$ docker pull debian
Using default tag: latest
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/create?fromImage=debian&tag=latest: dial unix /var/run/docker.sock: connect: permission denied
user@user-HP-Laptop-15-da0xxx:~$
```

List all the running containers

----->\$ sudo docker ps -a

```
user@user-HP-Laptop-15-da0xxx:~$ sudo docker ps -a
[sudo] password for user:
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
user@user-HP-Laptop-15-da0xxx:~$
```

Remove a container

----->\$ sudo docker rm <container id>

RESULT: Familiarised with Hypervisors and VMs, Xen or KVM, Containers,Docker