Flavin Daxey | 4NI20IS401| B Section

# devOps

## ASSIGNMENT 2

### 1. Comparison between Hypervisor and Docker.

Hypervisors and Dockers are not the same, and neither can be used interchangeably.

**Functioning Mechanism**

The most significant difference between hypervisors and Dockers is the way they boot up and consume resources.

Hypervisors are of two types – the bare metal works directly on the hardware while type two hypervisor works on top of the operating system.

Docker, on the other hand, works on the host kernel itself. Hence, it does not allow the user to create multiple instances of operating systems.

Instead, they create containers that act as virtual application environments for the user to work on.

**Memory Requirement**

Hypervisors enable users to run multiple instances of complete operating systems. This makes them resource hungry.

They need dedicated resources for any particular instance among the shared hardware which the hypervisor allocates during boot.

Dockers, however, do not have any such requirements. One can create as many containers as needed.

Based on the application requirement and availability of processing power, the Docker provides it to the containers.

**Number of Application Instances Supported**

A hypervisor allows the users to generate multiple instances of complete operating systems.

Dockers can run multiple applications or multiple instances of a single application. It does this with containers.

**Boot-Time**
As Dockers do not require such resource allocations for creating containers, they can be created quickly to get started.

One of the primary reasons why the use of Dockers and containers is gaining traction is their capability to get started in seconds.

A hypervisor might consume up to a minute to boot the OS and get up and running.

Docker can create containers in seconds, and users can get started in no time.

**Architecture Structure**
If we consider both hypervisor and Docker's architecture, we can notice that the Docker engine sits right on top of the host OS.

It only creates instances of the application and libraries.

Hypervisor though, has the host OS and then also has the guest OS further. This creates two layers of the OS that are running on the hardware.

If you are to run a portable program and want to run multiple instances of it, then containers are the best way to go. Hence you can benefit significantly with a Docker.

Dockers help with the agile way of working. Within each container, different sections of the program can be developed and tested.

In the end, all containers can be combined into a single program. Hypervisors do not provide such capability Security

Hypervisors are much more secure since the additional layer helps keep data safe.

One of the major differences between the two is the capability to run operating systems or rather run on operating systems.

**OS Support**
Hypervisors are OS agnostic. They can run across Windows, Mac, and Linux.

Dockers, on the other hand, are limited to Linux only. That, however, is not a deterrent for Dockers since Linux is a strong eco-system. Many major players are entering into the Dockers' fray.

## 2. Comparison between Containers and Virtual Machines.

The following are some of the similarities and differences of these complementary technologies.

### Isolation

Containers typically provides lightweight isolation from the host and other containers, but doesn't provide as strong a security boundary as a VM.

VM Provides complete isolation from the host operating system and other VMs. This is useful when a strong security boundary is critical, such as hosting apps from competing companies on the same server or cluster.

### Operating system

Runs the user mode portion of an operating system, and can be tailored to contain just the needed services for your app, using fewer system resources.

Runs the user mode portion of an operating system, and can be tailored to contain just the needed services for your app, using fewer system resources.

### Guest compatibility

Runs on the same  operating system version as the host

Runs just about any operating system inside the virtual machine

### Deployment

Deploy individual containers by using Docker via command line; deploy multiple containers by using an orchestrator such as Azure Kubernetes Service.

Deploy individual VMs by using Windows Admin Center or Hyper-V Manager; deploy multiple VMs by using PowerShell or System Center Virtual Machine Manager.

### Operating system updates and upgrades

Updating or upgrading the operating system files within a container is the same:

- Edit your container image's build file (known as a Dockerfile) to point to the latest version of the Windows base image.

- Rebuild your container image with this new base image.
- Push the container image to your container registry.
- Redeploy using an orchestrator.
  The orchestrator provides powerful automation for doing this at scale.

Download and install operating system updates on each VM. Installing a new operating system version requires upgrading or often just creating an entirely new VM. This can be time-consuming, especially if you have a lot of VMs.

**Persistent storage**

Use Azure Disks for local storage for a single node, or Azure Files (SMB shares) for storage shared by multiple nodes or servers.

Use a virtual hard disk (VHD) for local storage for a single VM, or an SMB file share for storage shared by multiple servers

**Load balancing**

Containers themselves don't move; instead an orchestrator can automatically start or stop containers on cluster nodes to manage changes in load and availability.

Virtual machine load balancing moves running VMs to other servers in a failover cluster.

**Fault tolerance**

If a cluster node fails, any containers running on it are rapidly recreated by the orchestrator on another cluster node.

VMs can fail over to another server in a cluster, with the VM's operating system restarting on the new server.

**Networking**

Uses an isolated view of a virtual network adapter, providing a little less virtualization–the host's firewall is shared with containers–while using less resources

Uses virtual network adapters.