# EXPERIMENT-12

**NAME:** SHAIKH MUBASHIRA TUFEL AHMED
**ROLL NO:** 612055 **COURSE:** ADVANCE DEVOPS(ITL504)
**BRANCH:** T.E. INFORMATION TECHNOLOGY (SEM 5)

## CASE-STUDY [Kubernetes]

### Q1. What is Kubernetes?

➔Kubernetes automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, monitoring your applications, and more making it easier to manage applications.

Kubernetes, or K8s for short, is an open-source container-orchestration tool designed by Google. It's used for bundling and managing clusters of containerized applications a process known as 'orchestration' in the computing world. The name Kubernetes originates from Greek, meaning helmsman or pilot.
Kubernetes is a portable, extensible, opensource platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

### Q2. How is Kubernetes related to Docker?

➔Kubernetes is open-source orchestration software that provides an API to control how and where those containers will run. It allows you to run your Docker containers and workloads and helps you to tackle some of the operating complexities when moving to scale multiple containers, deployed across multiple servers.

You can decide to use Kubernetes without Docker, or even Docker without Kubernetes for that matter (but we advise you to use it for different purposes than running containers). Still, even though Kubernetes is a rather extensive tool, you will have to find a good container runtime for it – one that has implemented CRI.

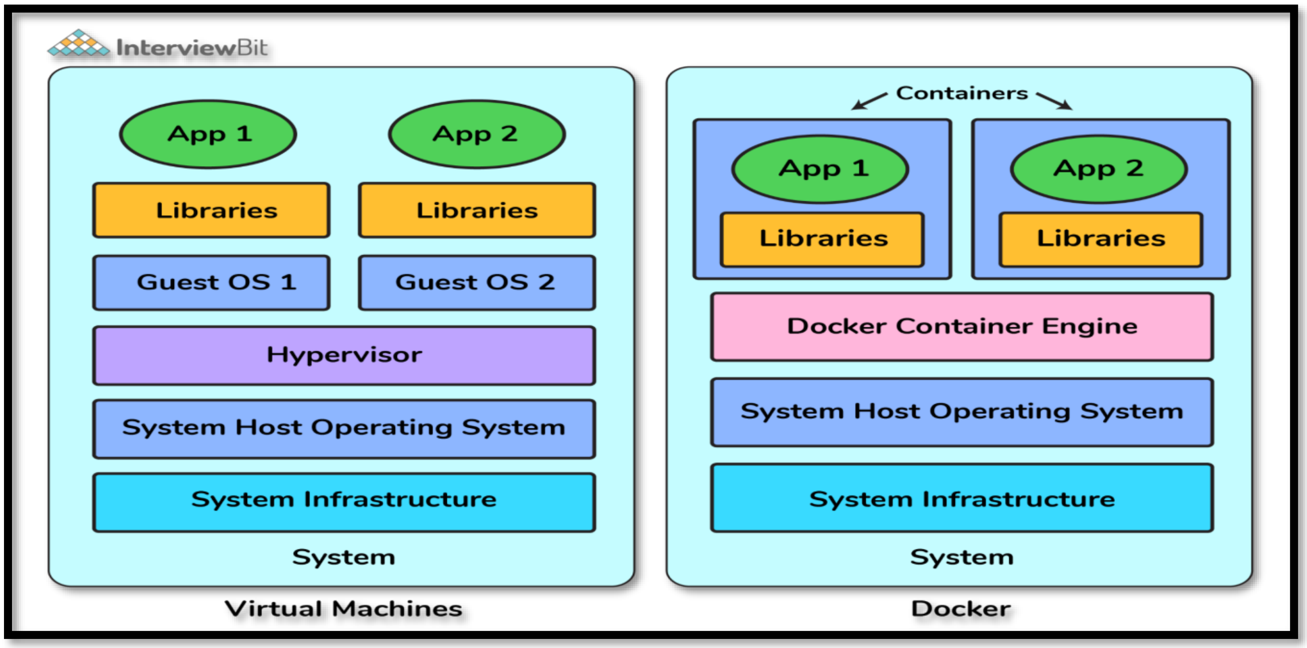
Kubernetes is most commonly used with Docker managed containers, although it doesn't strictly depend on it.

There are still problems with this architecture though, for example:
Stopping and starting instances is slow.
Communicating between nodes in this architecture can be quite complex.
If you're using a monolithic architecture over microservices, you may not need to deal with communication between nodes and so won't experience these issues. You will still
need to worry about instance startup time whenever introducing rolling upgrades or performing a failover. This can incur extra costs since the risk of losing requests becomes higher the longer a restart takes, so you're likely to want to have another instance to minimize that risk. This is where Kubernetes comes in. Kubernetes is a
platform for managing containerised services. This means that it's a tool made to
abstract away details such as separation of nodes, while automating things like rolling upgrades, failover, and scaling of services. The idea is that you should be able to deploy in a very similar way whether running locally or in the cloud.
This blog will walk through some of the basics of Kubernetes, and will setup a simple microservice application locally to demonstrate its use. This blog will assume you have the following tools installed and configured already:

- **Maven (v3+)**
- **JDK 8**

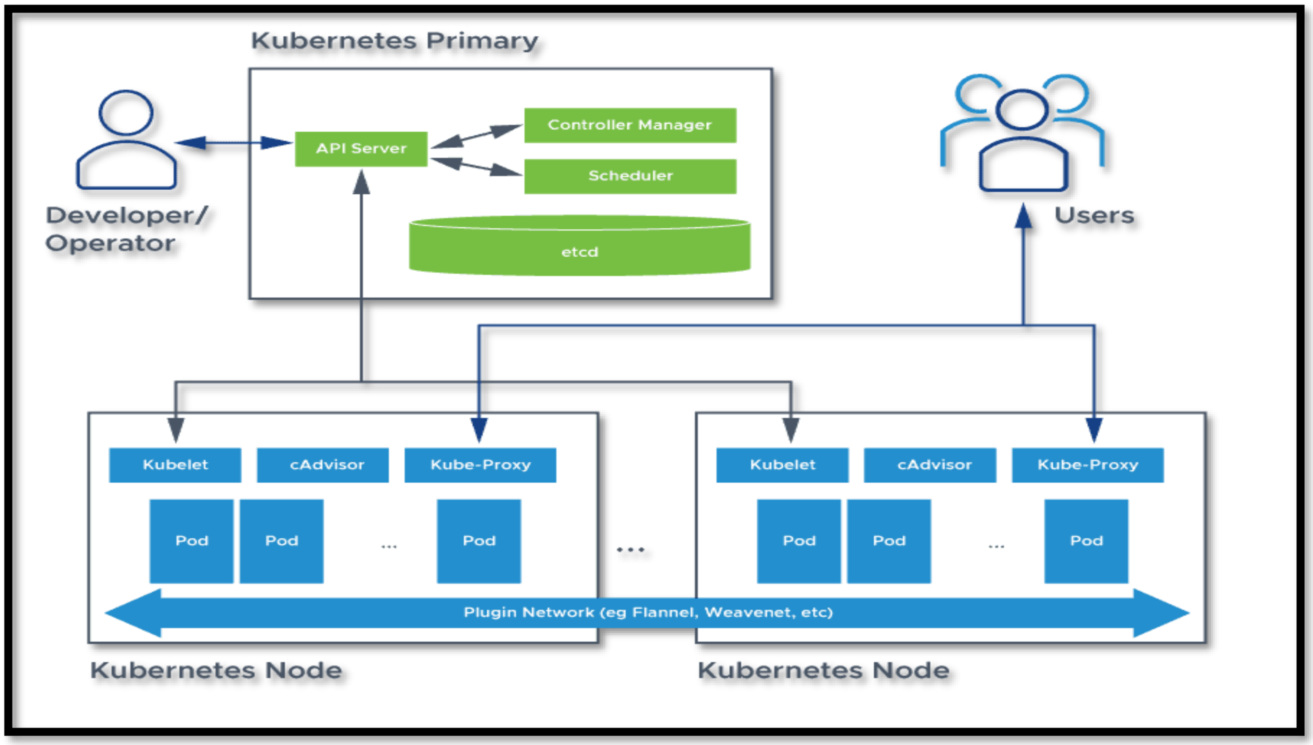
**Q3. What is the difference between deploying applications on hosts and containers?**

| Parameter | Virtual Machines | Containers |
|---|---|---|
| Guest OS | Each VM runs on virtual hardware and Kernel is loaded into in its own memory region | All the guests share same OS and Kernel. Kernel image is loaded into the physical memory |
| Communication | Will be through Ethernet Devices | Standard IPC mechanisms like Signals, pipes, sockets etc. |
| Security | Depends on the implementation of Hypervisor | Mandatory access control can be leveraged |
| Performance | Virtual Machines suffer from a small overhead as the Machine instructions are translated from Guest to Host OS. | Containers provide near native performance as compared to the underlying Host OS. |
| Isolation | Sharing libraries, files etc between guests and between guests hosts not possible. | Subdirectories can be transparently mounted and can be shared. |
| Startup time | VMs take a few mins to boot up | Containers can be booted up in a few secs as compared to VMs. |
| Storage | VMs take much more storage as the whole OS kernel and its associated programs have to be installed and run | Containers take lower amount of storage as the base OS is shared |



**Q6. Explain Kubernetes architecture with neat diagram and explain all the component of Kubernetes architecture.**

Basic Kubernetes architecture exists in two parts: the control plane and the nodes or compute machines. Each node could be either a physical or virtual machine and is its own Linux environment. Every node also runs pods, which are composed of containers.



**Control Plane Components:**

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied). Control plane
components can be run on any machine in the cluster. However, for simplicity, set up
scripts typically start all control plane components on the same machine, and do not run user containers on this machine. See Creating Highly Available clusters with kubeadm for an example control plane setup that runs across multiple machines.

**kube-apiserver:**

The API server is a component of the Kubernetes control plane that exposes the

Kubernetes API. The API server is the front end for the Kubernetes control plane. The main implementation of a Kubernetes API server is kube-apiserver. kube-apiserver is
designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance tra c between those instances.

**Etcd:**

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data. If your Kubernetes cluster uses etcd as its backing store, make sure you have a back up plan for that data. You can find in-depth information about etcd in the ocial documentation.

**kube-scheduler:**

Control plane component that watches for newly created Pods with no assigned node, and selects a Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, anity and antianity specifications, data locality, inter-workload interference, and deadlines. **kube-controller-manager:**

Control plane component that runs controller processes. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

- **Node controller:** Responsible for noticing and responding when nodes go down. Job controller: Watches for Job objects that represent one-o tasks, then creates Pods to run those tasks to completion.
- **EndpointSlice controller:** Populates EndpointSlice objects (to provide a link between Services and Pods).
- **ServiceAccount controller:** Create default ServiceAccounts for new namespaces.

**cloud-controller-manager:**

➔A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.

As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process.

You can scale horizontally (run more than one copy) to improve performance or to help tolerate failures.

The following controllers can have cloud provider dependencies:

- **Node controller:** For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- **Route controller:** For setting up routes in the underlying cloud infrastructure
- **Service controller:** For creating, updating and deleting cloud provider load balancers

**Node Components:**

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

**Kubelet:**

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

**kube-proxy:**

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.kube-proxy maintains network rules on nodes.These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the trac itself.

**Container runtime:**

The container runtime is the software that is responsible for running containers.
Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

**Addons:**

Addons use Kubernetes resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace. Selected addons are described below; for an extended list of available addons, please see Addons.

**DNS :**

While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it. Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.

**Web UI (Dashboard):**

Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster
itself.

**Container Resource Monitoring:**

Container Resource Monitoring records generic time-series metrics about containers in a central database, and provides a GUI for browsing that data. Cluster-level Logging A cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface.