Dharmsinh Desai University

MCA Department

Semester -II

Seminar Report

On

CONTANERIZATION - KUBERNETES

By

JAYMIN VALAKI D (MA075), 22MAP0G014

TANK RAJNIKUMAR R (MA068), 22MAPB075

MCA Department
Faculty of Management & Information Sciences,
Dharmsinh Desai University,
College Road, Nadiad-387001

Certificate

This is to certify that **JAYMIN VALAKI D (MA075), 22MAP0G014**
of MCA Semester-II of Dharmsinh Desai University, Nadiad, has worked on the seminar

“Containerization - KUBERNETES” under my guidance.

Guided By	**Dr. Narayan Joshi**
Prof./Purvi Jarods	Professor & Head, MCA Department
MCA Department	D.D.U University
Date:	Date:

❖ Preface: -

Containerization has transformed the way software applications are developed and deployed, making it faster, more efficient, and less prone to errors. Docker is a popular tool used to create and manage containers, allowing developers to easily package their applications with all their dependencies. Kubernetes, on the other hand, is a powerful orchestration tool that helps manage and scale containerized applications. It automates tasks like deployment, scaling, and management of containers, making it easier to manage complex applications.

Kubernetes is a widely used container orchestration platform that simplifies the deployment, scaling, and management of containerized applications. It has become an essential tool for modern application development and deployment, providing a platform-agnostic way of managing containers across different environments. Kubernetes offers a declarative approach to application management, making it more efficient and reducing the risk of human error.

Kubernetes offers features such as load balancing, service discovery, and rolling updates, which make application deployment and management more efficient.With its robust ecosystem of tools and integrations, Kubernetes is an essential tool for modern application development and deployment.

INDEX

Sr No	Name of Topic	Page No
1)	**What is Containerization?**	**6**
1.1	**Virtualization vs Containerization**	**7**
1.1	**Containerization Features**	**7**
2)	**Docker**	**8**
2.1	**Docker History**	**8**
2.2	**Architecture of Docker**	**9**
2.3	**Docker Command**	**10**
3)	**Kubernetes**	**12**
3.1	**Architecture of kubernetes**	**13**
3.2	**Kubernetes Install**	**15**
4)	**Kubernetes deployment**	**16**
4.1	**Pods and Minikube**	**16**
4.2	**Deployment**	**18**
5)	**References**	**20**

❖ Abstract: -

- ➢ Containerization is a deployment method that packages applications and dependencies into lightweight containers for portability and scalability. It provides a consistent runtime environment that is isolated from the underlying system for security. Containerization also enables efficient resource utilization by allowing multiple containers to run on a single host. Popular containerization technologies include Docker, Kubernetes, and container orchestration. Kubernetes is a popular container orchestration tool that automates the deployment, scaling, and management of containerized applications.
- ➢ Kubernetes is an open-source platform that facilitates the deployment, scaling, and management of containerized applications. It was originally developed by Google, and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes provides a highly scalable and fault-tolerant infrastructure for managing containerized workloads and services, which allows organizations to build and deploy applications faster and more efficiently.
- ➢ At the heart of Kubernetes is its container orchestration engine, which enables developers to package their applications and dependencies into self-contained, portable units known as containers. These containers can be easily deployed across different environments, such as public clouds, private data centres, or hybrid infrastructures. Kubernetes also provides a set of powerful features for managing containerized applications, such as automatic load balancing, rolling updates, and self-healing mechanisms.

  Overall, Kubernetes has become a widely adopted platform for container orchestration and management, and has helped to drive the adoption of container-based architectures and microservices. As organizations continue to embrace cloud-native technologies, Kubernetes is likely to remain a critical tool for building and deploying modern applications in a scalable and reliable manner.

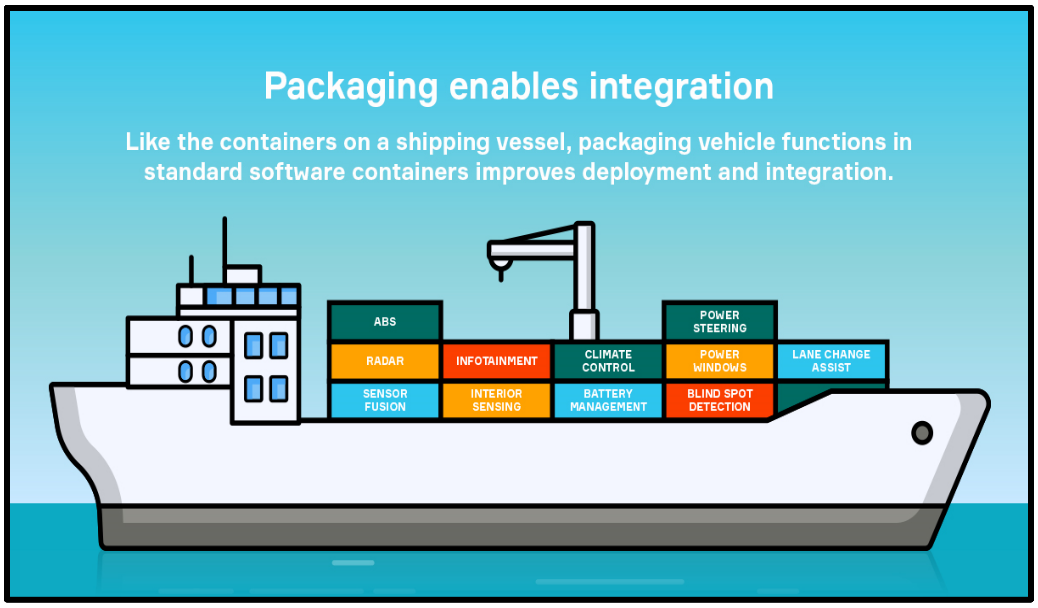
1. What is containerization?

- Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.[1]

- Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run.

- This single package of software or "container" is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.

- The concept of containerization and process isolation is actually decades old, but the emergence in 2013 of the open-source Docker Engine—an industry standard for containers with simple developer tools and a universal packaging approach—accelerated the adoption of this technology. Today organizations are using containerization increasingly to create new applications, and to modernize existing applications for the cloud. In an IBM survey (PDF, 1.4 MB), 61% of container adopters reported using containers in 50% or more of the new applications they built during the previous two years; 64% of adopters expected 50% or more of their existing applications to be put into containers during the next two years.[1]


Packaging enables integration
Like the containers on a shipping vessel, packaging vehicle functions in standard software containers improves deployment and integration.
ABS
POWER STEERING
RADAR
INFOTAINMENT
CLIMATE CONTROL
POWER WINDOWS
LANE CHANGE ASSIST
SENSOR FUSION
INTERIOR SENSING
BATTERY MANAGEMENT
BLIND SPOT DETECTION

1.1) Containerizations vs virtualization

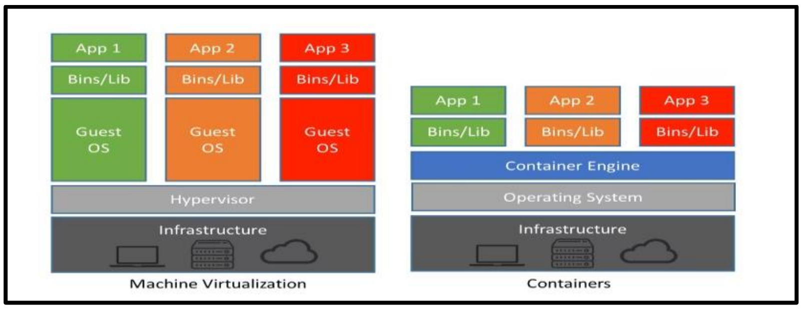
- “Virtualization is a technology that allows you to create multiple simulated environments or dedicated resources from a single, physical hardware system”.

- “Containerization is the packaging together of software code with all its necessary components like libraries, frameworks, and other dependencies so that they are isolated in their own "container".[1]

- The major differences are:

◆ In virtualization, the hypervisor creates an abstraction layer over hardware, so that multiple operating systems can run alongside each other. This technique is considered to be the first generation of cloud computing.

◆ Containerization is considered to be a lightweight version of virtualization, which virtualizes the operating system instead of hardware. Without the hypervisor, the containers enjoy faster resource provisioning. All the resources that are needed to run the application or microservice are packaged together, so that the app run everywhere.[1].



1.2) Feature of containerization

- Isolation: Containers provide a high level of isolation between applications and their dependencies, ensuring that changes made to one container do not affect others.[1]

- Portability: Containers can be easily moved between different environments, such as development testing, without any compatibility issues.

- Efficiency: Containers are lightweight and require fewer resources than traditional virtual machines, making them faster and more efficient to deploy.

- Security: Containers provide a more secure environment for applications, as they are isolated from the host operating system and other containers, reducing the risk of malware or other security breaches.

- Flexibility: Containers allow developers to use different programming languages, frameworks, and tools within the same environment, making it easier to build and deploy complex applications.

2)DOCKER

What is Docker?

- Docker is a Linux-based, open-source containerization platform that developers use to build, run, and package applications for deployment using containers. Unlike virtual machines, Docker containers offer:

- Docker is one among many container platforms that allows you to build such containers to test, build and scale applications rapidly and easily by running them in isolated environments.[2]

History of Docker

- Docker is a popular software containerization platform that allows developers to package and deploy their applications and services as containers. Docker was first introduced in 2013 by Solomon Hykes and his team at dot Cloud, a PaaS provider that was later rebranded as Docker Inc.

- Docker introduced a simple and powerful way to create, deploy, and manage containers that could run almost any application. Docker quickly gained popularity among developers and became the de facto standard for containerization. Docker also created an ecosystem of tools and services around it, such as Docker Compose, Docker Swarm, and Docker Hub.

- In 2015, Docker Engine, an open-source project that provided a runtime for containers, was released. Docker has revolutionized application deployment and management, making it easier for developers and DevOps teams to create, test, and deploy applications in a consistent and reliable manner. Docker is now owned by Merantis, a cloud computing company, and continues to be widely used by developers and organizations around the world.[2]

Different Vendors that provide Containerization service

1. DOCKER
2. KUBERNETS
3. AMAZON WEB SERVICES
4. DIGITAL OCEAN
5. FUZE and many more

Why use docker?

- Helps transport code faster
- Makes scaling, deploying and identifying issues in the application easier
- Saves up space as you don’t need to install the whole application locally

2.2) ARCHITECHURE OF DOCKER

- **Docker Client**

It is what we use to interact with Docker. You can think of it like the user interface for Docker. Whenever you type in a command, the client sends these to the daemon.

- **Docker CLI**

It allows users to issue commands to the Docker Daemon. Docker uses a client – server architecture.

- **Docker Engine**

It is responsible for the overall functioning of Docker platform. It is a client-server-based application with three components:

(i) Server – which runs the daemon

(ii) Rest API – deals with the interaction of applications with their server

(iii) Client – which is nothing but the command line interface (CLI)

- **Docker Daemon**

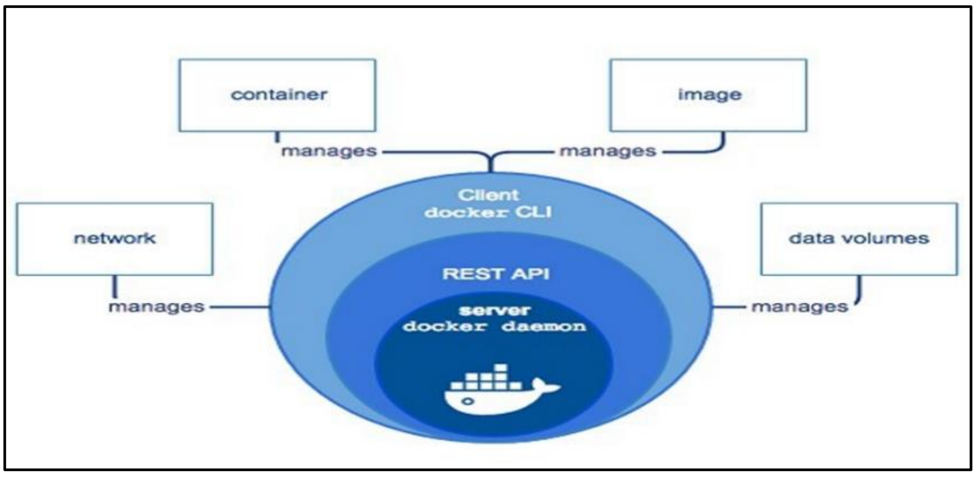
It is the heart of the Docker architecture, which does the crucial work of building, running, & distributing the containers. It also manages the Docker images and the container.

- **Docker Image**

The file that contains the source code, operating system files to run the application along with its other dependencies inside the container is called Docker image. A containerized application is the running instance of an image. Docker images are immutable.

- **Dockerfile**

It contains the list of instructions to create Docker images.



2.2) Docker Install

- **Install Docker Desktop**

- If you installed Docker Desktop/Toolbox for either Windows or Mac, you already have Docker Compose! Play-with-Docker instances already have Docker Compose installed as well. If you are on a Linux machine, you will need install docker compose from site: **https://docs.docker.com/desktop/install/windows-install/**
- After installation, you should be able to run the following and see version information.

```
$ docker compose version
```

2.2) Docker command

i. **docker build**: This command is used to build a Docker image from a Dockerfile. The Dockerfile contains instructions for building the image. When this command is executed, Docker reads the Dockerfile and creates a new image based on the instructions in the file.

ii. **docker run**: This command is used to run a Docker container from an image. The command pulls the specified image (if not already present) and starts a new container based on that image.

iii. **docker images**: This command is used to list all the Docker images available on the local machine. The output includes the repository, tag, and image ID of each image.

iv. **docker PS**: This command is used to list all the running Docker containers. The output includes the container ID, image, status, and name of each running container.

v. **docker stop:** This command is used to stop a running Docker container. The command sends a signal to the container to stop it gracefully.

vi. **docker rm**: This command is used to remove a Docker container. The command can be used with the container ID or name.

vii. **docker rmi**: This command is used to remove a Docker image. The command can be used with the image ID or name.

viii. **docker pull**: This command is used to download a Docker image from a registry. The command pulls the specified image from the registry and saves it locally.

ix. **docker push**: This command is used to push a Docker image to a registry. The command pushes the specified image to the registry so that it can be accessed by others.

x. **docker exec**: This command is used to execute a command in a running Docker container. The command can be used to start a new process inside the container or to run a command in an existing process.

xi. **Docker copy**: This command copies a file from docker to the local system. Here is how to use it.

xii. **docker swarm**: This command is used to create and manage a swarm of Docker nodes, enabling you to deploy and orchestrate containers across a cluster of machines.

3) KUBERNETES

➢ What is Kubernetes?

- With the widespread adoption of containers among organizations, Kubernetes, the container-centric management software, has become the de facto standard to deploy and operate containerized applications. Google Cloud is the birthplace of Kubernetes—
- originally developed at Google and released as open source in 2014.
- Kubernetes builds on 15 years of running Google's containerized workloads and the valuable contributions from the open-source community. Inspired by Google's internal cluster management system, Borg, Kubernetes makes everything associated with deploying and managing your application easier. Providing automated container orchestration, Kubernetes improves your reliability and reduces the time and resources attributed to daily operations.

Kubernetes defined

Kubernetes (sometimes shortened to K8s with the 8 standing for the number of letters between the "K" and the "s") is an open-source system to deploy, scale, and manage containerized applications anywhere.[3]

What is Kubernetes used for?

- Kubernetes is used to create applications that are easy to manage and deploy anywhere. When available as a managed service, Kubernetes offers you a range of solutions to meet your needs. Here are some common use cases.

1. **Increasing development velocity**
   Kubernetes helps you to build cloud-native microservices-based apps. It also supports containerization of existing apps, thereby becoming the foundation of application modernization and letting you develop apps faster.

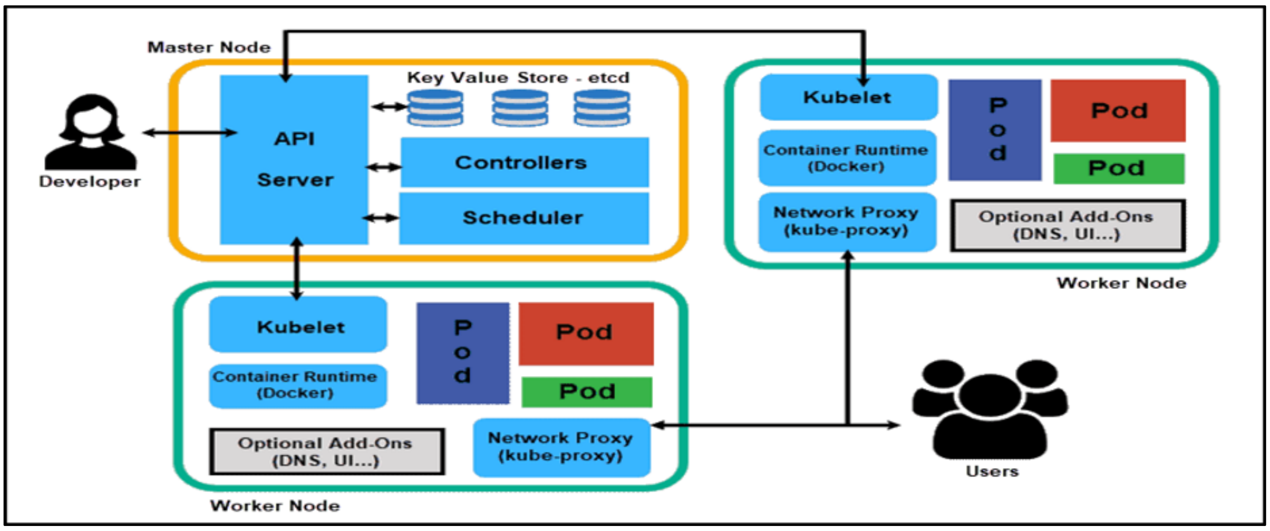
2. **Deploying applications anywhere**
   Kubernetes is built to be used anywhere, allowing you to run your applications across on-site deployments and public clouds; as well as hybrid deployments in between. So you can run your applications where you need them.

3. **Running efficient services**
   Kubernetes can automatically adjust the size of a cluster required to run a service. This enables you to automatically scale your applications, up and down, based on the demand and run them efficiently.

4. **Automated operations**
   Kubernetes has built-in commands to handle a lot of the heavy lifting that goes into application management, allowing you to automate day-to-day operations..

➢ KUBERNETES ARCHITECTUE

Kubernetes Architecture consists of master node which manages other worker nodes. Worker nodes are nothing but virtual machines / physical servers running within a data center. They expose the underlying network and storage resources to the application. All these nodes join together to form a cluster with providing fault tolerance and replication. These nodes were previously called minions.


Master Node
Key Value Store - etcd
API Server
Controllers
Scheduler
Developer
Kubelet
Container Runtime (Docker)
Network Proxy (kube-proxy)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Worker Node
Kubelet
Container Runtime (Docker)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Network Proxy (kube-proxy)
Worker Node
Users

➢ Main component of a Kubernetes

1. Master Node
   1.1. API Server
   1.2. Scheduler
   1.3. Control Manager

2. Master Node
   2.1. Kubelet
   2.2. Pods
   2.3. Containers

➢ Components of the Master Node

1.1) API server

- Gatekeeper for the entire cluster. CRUD operations for servers go through the API. API server configures the API objects such as pods, services, replication controllers () and deployments. It exposes API for almost every operation. How to interact with this API? Using a tool called kubectl aka Kub control. It talks to the API server to perform any operations that we issue from cmd. In most cases, the master node does not contain containers. It just manages worker nodes, and also makes sure that the cluster of worker nodes are running healthy and successfully.[5]

1.2. Scheduler:

- The scheduler is responsible for assigning workloads to nodes in the Kubernetes cluster. It watches for new workloads that need to be scheduled and selects the most appropriate node to run them on based on factors such as resource availability and workload constraints. The scheduler also takes into account factors such as affinity and anti-affinity rules to ensure that workloads are scheduled in a way that meets the desired configuration.[5]

1.3. Control Manager:

- The control manager is a collection of components that are responsible for maintaining the desired state of the Kubernetes cluster. This includes components such as the node controller, the replication controller, and the endpoints controller. The control manager watches for changes to the state of the cluster and takes action to ensure that the actual state matches the desired state.
- For example, if a node goes down, the node controller will detect this and take action to reschedule any workloads that were running on that node. The control manager also performs tasks such as managing security policies, managing storage resources, and performing automatic scaling based on workload demands.[5]

➢ Components of the Worker Node

2.1. Kubelet:

- The Kubelet is the primary agent running on each node in a Kubernetes cluster. Its main responsibility is to ensure that the containers running on the node are healthy and running as expected. The Kubelet communicates with the Kubernetes API server to receive instructions on what workloads to run, and it also communicates with the container runtime to manage the containers.

2.2. Pods:

- A pod is the smallest deployable unit in Kubernetes. It is a logical host for one or more containers, and it provides a shared namespace for those containers to communicate and share resources. A pod can contain one or more containers, and those containers are always scheduled together on the same node. Pods can be used to run applications or services, and they can be scaled horizontally by creating multiple replicas of the same pod.

2.4. Containers in Kubernetes:

- Containers in Kubernetes are the smallest unit of deployment that can be managed by Kubernetes. Containers are used to package and run applications and services in a way that is portable and easy to manage. Kubernetes can manage containers running on multiple nodes in a cluster, and it can automatically scale the number of containers based on workload demands.

- Kubernetes uses container runtimes such as Docker or container to manage the containers, and it provides a range of features such as networking, storage, and security to ensure that the containers are running as expected.

3.3) Installation

1. Minikube

It is used in case you want to install it on the system but have limited system resources. Minikube is all in one system, i.e., no multiple architectures of master and worker Node. The same system acts as a master as well as a worker node. It can be used for testing purposes. kubernetes.io/docs/setup/learning-environment. Install from site [4]
https://minikube.sigs.k8s.io/docs/start/

2. Installing Kubeadm, Kubelet and kubectl

You will install these packages on all of your machines:

i. Kubeadm: the command to bootstrap the cluster. **https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/**

ii. Kubelet: the component that runs on all of the machines in your cluster and does things like starting pods and containers.

iii. kubectl: the command line util to talk to your cluster.

- Kubeadm will not install or manage Kubelet or kubectl for you, so you will need to ensure they match the version of the Kubernetes control plane you want Kubeadm to install for you. If you do not, there is a risk of a version skew occurring that can lead to unexpected, buggy behaviour. However, one minor version skew between the Kubelet and the control plane is supported, but the Kubelet version may never exceed the API server version. For example, the Kubelet running 1.7.0 should be fully compatible with a 1.8.0 API server, but not vice versa.[4]

3) Cloud Platforms

- Various cloud services are being provided to run and manage Kubernetes. One can define a number of nodes in cluster, CPU and RAM configurations, etc. and the cloud will manage those resources. Some of the examples include Cavo, GCE, AWS, Azure, Cloud Stack, etc. For more information, check: kubernetes.io/docs/concepts/cluster-ad minis.

4) Play-with-k8s

- Imagine that, you want to quickly test something on your Kubernetes cluster. But it is not readily available. And you don’t want to set up Kubernetes cluster. Play-with-k8s provides with a Kubernetes playground which is similar to play-with-docker. A GitHub or Docker account is required. labs.play-with-k8s.com

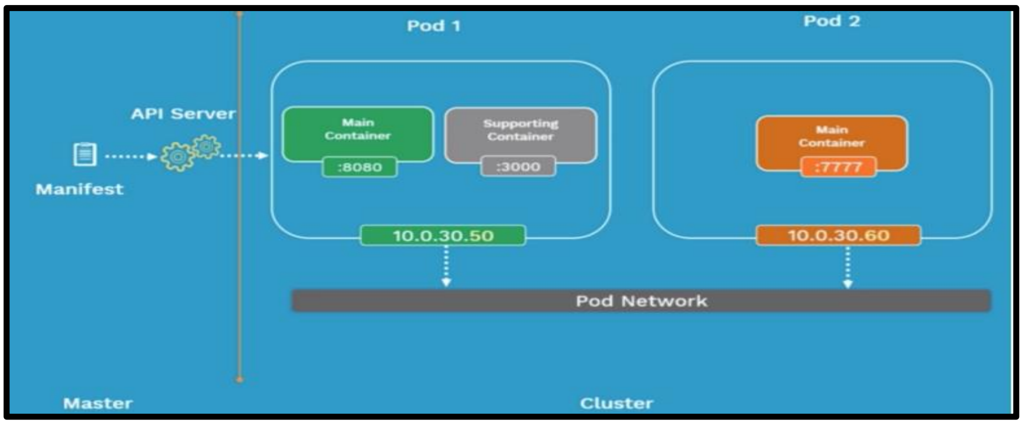
4) KUBERNETES DEPLOYMENT

4.1) More about Pods:

Every node inside a Kubernetes cluster has its unique IP address known as Node IP Address. In Kubernetes, there is an additional IP Address called Pod IP address. So once we deploy a Pod on the worker node, it will get its own IP Address. Containers in pods communicate with the outside world by network namespace. All the containers inside a pod operate within that same network namespace as the pod. Means all the containers in a pod will have the same IP Address as their worker node. There is a unique way to identify each container. It can be done by using ports. Note: Containers within the same pod, not only just share the same IP Address, but will also share the access to the same volumes, c-group limits, and even same IPC names.[5]

Pod Networking

- How do Pods communicate with one another?
- Inter-Pod communication: All the Pod IP addresses are fully routable on the Pod Network inside the Kubernetes cluster.



➢ **Pod Lifecycle**

- Define the pod configuration inside manifest file (explained ahead) in yaml/json. Submit the manifest file on the api server of the Kubernetes master.
- It will then get scheduled on a worker node inside the cluster. Once it is scheduled, it goes in the pending state. - During this pending state, the node will download all container images and start running the containers. It stays in the pending state until all containers are up and running.[5]
- Now it goes in the running state. When the purpose is achieved, it gets shutdown & state is changed to succeeded.
- Failed state: It happens when the pod is in the pending state and fails due to some particular reason. If a pod dies, you cannot bring it back. You can replace it with a new pod.

➢ Pod Manifest file

- A Pod Manifest file in Kubernetes is a YAML file that defines a Pod's configuration, including its containers, volumes, networking, and other properties.

- A Pod is the smallest deployable unit in Kubernetes, and it can contain one or more containers. The Pod Manifest file provides a declarative way of defining the desired state of a Pod and its associated resources.[5]

```
apiVersion: v1
kind: Pod
metadata:
   name: nginx-pod   //name of pod
   labels:
      app: nginx
      tier: dev
spec:
   containers:
     - name: nginx-container
       image: nginx
```

1. **ApiVersion**

- It defines the version of the Kubernetes API you're using to create this object.
- v1: It means that the Kubernetes object is part of first stable release of the Kubernetes API. So, it consists of core objects such as Pods, Replication Controller and Service.
- apps/v1: Includes functionality related to running apps in Kubernetes.
- batch/v1: Consists of objects related to bash processes and jobs like tasks.

2. **kind**
- It defines the type of object being created.

3. **metadata**
- Data that helps uniquely identify the object, including a name string, UID, and optional namespace.

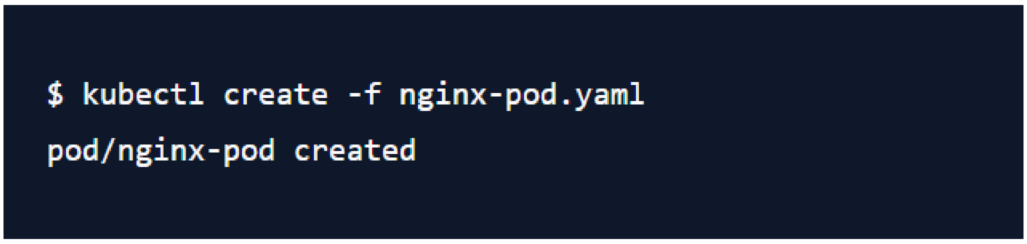
4. **spec**
- The precise format of the object spec is different for every Kubernetes object, and contains nested fields specific to that object

- Pod Manifest files can also define additional resources, such as volumes, environment variables, and other settings that configure the Pod's behavior. Once a Pod Manifest file is defined, it can be deployed to a Kubernetes cluster using the kubectl apply command, which ensures that the Pod and its associated resources are created and maintained in the desired state.

4.2) DEMO

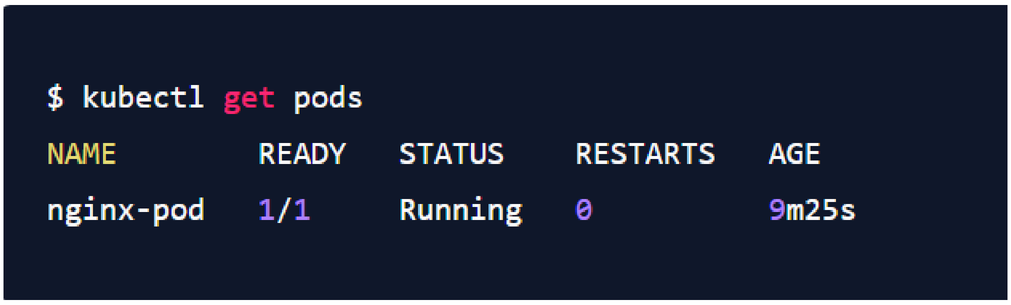
In this demo we will use the manifest file nginx-pod. Yaml described above.

- Create a new pod name it as a ngnix-pod. Yaml
- Syntax: **kubectl create -f <pod name>**



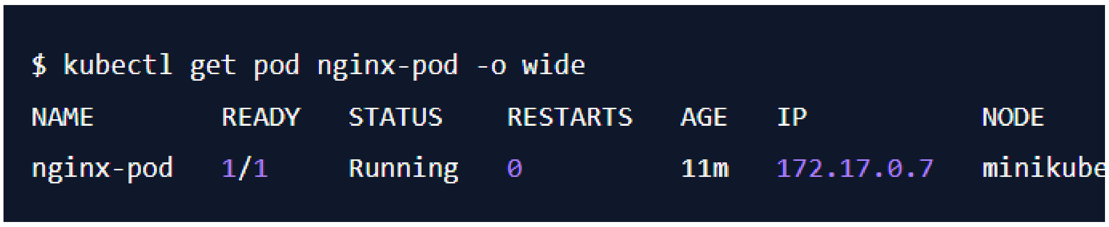
```
$ kubectl create -f nginx-pod.yaml
pod/nginx-pod created
```

- **To list all the pods, use the command**
  Syntax: kubectl get pods



```
$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
nginx-pod    1/1     Running   0          9m25s
```

- **Every pod has a unique IP address**



```
$ kubectl get pod nginx-pod -o wide
NAME         READY   STATUS    RESTARTS   AGE   IP           NODE
nginx-pod    1/1     Running   0          11m   172.17.0.7   minikube
```

- Get pod configuration in YAML format $ kubectl get pod nginx-pod -o Yaml
- Get pod configuration in JSON format $ kubectl get pod nginx-pod -o Json
- This will display details of the pod which includes list of all events from the time the pod is sent to the node till the current status of the pod $ kubectl describe pod nginx-pod
- Check if the pods are accessible: Verify if the connectivity from the master node to the pod is working by using the pod's IP address $ ping 172.17.0.7

- **Expose the pod using Node Port service**

```
$ kubectl expose pod nginx-pod --type=NodePort --port=80
service/nginx-pod exposed
```

- **Use the command kubectl describe svc nginx-pod to see details of a pods**

```
$ kubectl describe svc nginx-pod
Name:                     nginx-pod
Namespace:                default
Labels:                   app=nginx
                        tier=dev
Annotations:              <none>
Selector:                 app=nginx,tier=dev
Type:                     NodePort
IP:                       10.101.229.154
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30843/TCP
Endpoints:                172.17.0.7:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

- **Now let’s get inside the pod and execute some commands**

```
$ kubectl exec -it nginx-pod -- /bin/sh
# ls
bin  boot  dev etc  home  lib lib64  media  mnt  opt proc  root  run
# hostname
nginx-pod
# exit
```

References: -

- **https://www.ibm.com/inen/topics/containerization**
- **https://docs.docker.com/**
- **https://kubernetes.io/docs/home/**
- **https://minikube.sigs.k8s.io/docs/start/**
- **https://blog.wemakedevs.org/getting-started-with-kubernetes#heading-minikube**
- **https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/**