WHILE PRACTICE YOU FACE ANY ISSUE IN COPY AND PASTING BELOW COMMAND, FOR YOUR REFERENCE BELOW DOCUMENT IS AVAILABLE IN GOOGLE DRIVE ALSO

https://docs.google.com/document/d/1iy4xW2qQ7mUiDpk89-m5rbESbiVvybUB/edit?usp=share_link&ouid=105193179311738607734&rtpof=true&sd=true

CONTAINER

Sudo literally means SuperUser Do - it's a way to run commands as root user

APT stands for Advanced Packaging Tool

```
sudo su -
```

```
apt update
```

```
apt install docker.io -y
```

```
service docker status
```

Docker host is a physical or virtual server on which the Docker is installed

```
docker images
```

To create a container we need image

https://hub.docker.com/

Nginx is an open source web server

```
docker pull nginx:latest
```

```
docker images
```

Docker will store this image so you don’t need to download the image each time.

Create Docker Container

docker run -d --name my-nginx-container -p 80:80 nginx:latest

- `run` is the command to create a new container
- `-d` stands for detached mode, docker container will runs in the background of your screen you can continue to type command on your screen
- The `--name` is to specify the name of the container
- `-p` create a mapping between Dockerhost-port:Container-port without the port mapping, you wouldn't be able to access the Nginx application.
- `Nginx:latest` Nginx is the name of the image and latest is the version

```
root@docker-host:~# docker run  --name  my-nginx-container -p 80:80 nginx:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/12/05 12:48:44 [notice] 1#1: using the "epoll" event method
2022/12/05 12:48:44 [notice] 1#1: nginx/1.23.2
2022/12/05 12:48:44 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/12/05 12:48:44 [notice] 1#1: OS: Linux 5.15.0-1023-azure
2022/12/05 12:48:44 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/12/05 12:48:44 [notice] 1#1: start worker processes
2022/12/05 12:48:44 [notice] 1#1: start worker process 28
```

How many containers are running on Docker host

```
docker ps
```

Now access the webpage

Stop the Nginx Container

```
docker stop my-nginx-container
```

Start the Nginx Container

```
docker start my-nginx-container
```

let’s go inside the container

docker exec -it my-nginx-container bash

cd /usr/share/nginx/html

ls

apt-get update && apt-get install -y vim

vi index.html

<h1>This is a Container</h1>

Now access the webpage

CREATE A CUSTOM IMAGE FROM CONTAINER

docker commit my-nginx-container zameerm2526/my-nginx-container-image:v1

docker images → you will see your image here which you can push to docker hub

docker login

zameerm2526

docker push zameerm2526/my-nginx-container-image:v1

Now stop the existing container and delete the container

docker ps

docker stop my-nginx-container

docker rm my-nginx-container

docker ps

Now delete all the images

docker images

docker image rm nginx:latest

docker image rm zameerm2526/my-nginx-container-image:v1

docker images

DIFFERENCE BETWEEN VM AND CONTAINER

	VIRTUAL MACHINE	CONTAINER
1	Hypervisor Required to create VM	Docker Required to create Container
2	Each VM has it own OS and applications	Containers donot has OS in them they just have the application
3	It takes a few minutes for VMs to boot.	Boots in a few seconds.
4	VM Image can be created but there is no central place to keep the VM image and acessing vm image from anywhere is a challenge	Container Image can be created and stored it in docker hub and access from anywhere
5	VM image creation process is complex	Container image creation process is simple
6	VM images are in GB they are heavyweight	Container image are in MB, they are light weight

KUBERNETES

Kubernetes is a container orchestrator which means

1) Deployment of containers
2) Load balancing of containers
3) Auto-Scaling of containers
4) Rolling updates of containers when there is a change in image
5) Monitoring and health check of containers

Kubernetes follows the master/worker architecture. So, we have the master nodes and the worker nodes. The master nodes manage the worker nodes and together they form a cluster

Master Node

Master Node can be Physical or Virtual Server

On Master Node the Control Plane is installed and it coordinates all activities in your cluster, such as scheduling containers on Worker Nodes, scaling containers, and rolling out new updates.

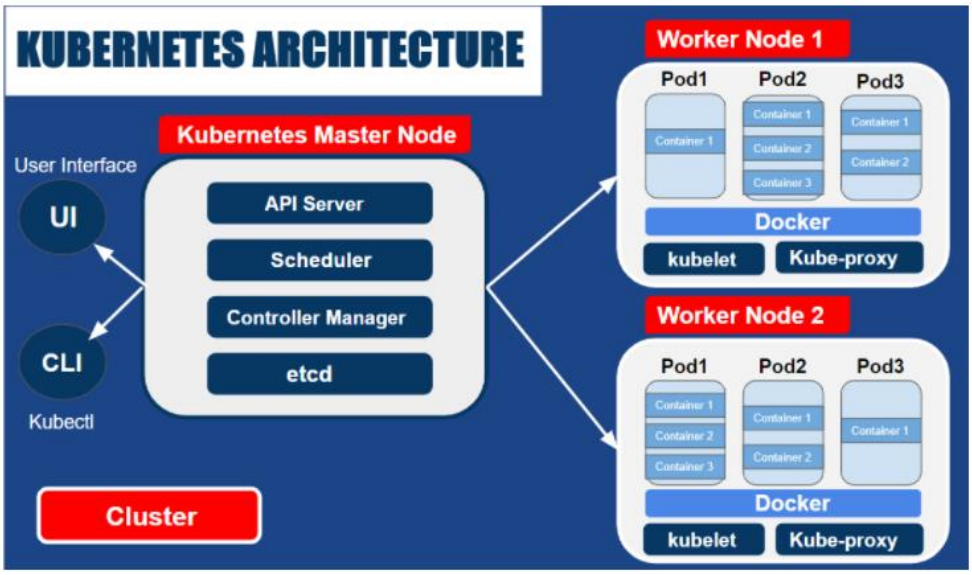
Worker Nodes

Each Worker Node has

1) Each Worker Node contains a kubelet, a tiny application that communicates with the Master Node.

2) Each Worker Node contains kube-proxy, which allows network communication

3) Each Worker Node contains a container runtime (like Docker) responsible for pulling the container image from a docker hub, unpacking the container, and running the container



POD

Container are not directly created on Worker Nodes

Pod are created on Worker Nodes and inside the POD the container is created

Container is just an application like NGINX working on port 80 to connect to this application you need IP Address and that IP Address is provided by POD, hence POD are needed

POD can consist of one or multiple containers

Azure AKS

AKS is Azure Kubernetes Services which is Microsoft's managed service for Kubernetes running in Azure

When you create a cluster, Microsoft manages the AKS control plane and you only pay for the worker nodes

The **kubectl** command line tool lets you control Kubernetes clusters

KUBERNETES

kubectl get nodes

---

vi myfirstpod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-pod
spec:
  containers:
  - name: my-nginx-container
    image: zameerm2526/my-nginx-container-image:v1
    ports:
    - containerPort: 80
```

kubectl apply -f myfirstpod.yaml

Pods is running on which node

kubectl get pods

In kubernetes, every pod gets assigned an IP address, and every container in the pod gets assigned that same IP address.

kubectl get pods -o wide

See Container inside Pod

kubectl get po -o jsonpath='{range .items[*]}{"pod: "}{.metadata.name}{"\n"}{range .spec.containers[*]}{"\tname: "}{.name}{"\n\timage: "}{.image}{"\n"}{end}'

IN DEPTH DETAILS ABOUT POD

kubectl describe pod my-nginx-pod

Kubernetes Master will assign the pod to Worker Node with name of **my-nginx-pod**

The worker node will pull the image from docker hub

And create a container with the name of **my-nginx-container** inside the POD

GO INSIDE CONTAINER

kubectl exec -i -t my-nginx-pod --container my-nginx-container -- /bin/bash

curl http:// 10.244.0.11

DELETE POD

kubectl delete pod my-nginx-pod

kubectl get pods

The pod and container inside the pod is deleted

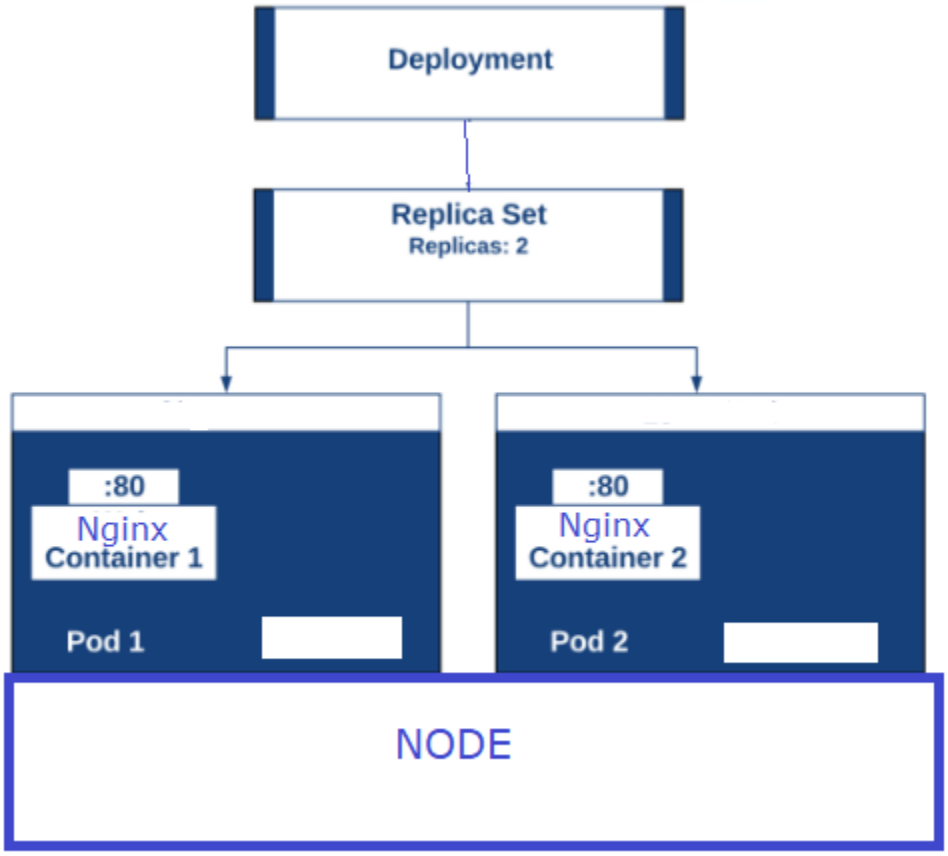
KUBERNETES ARCHITECTURE

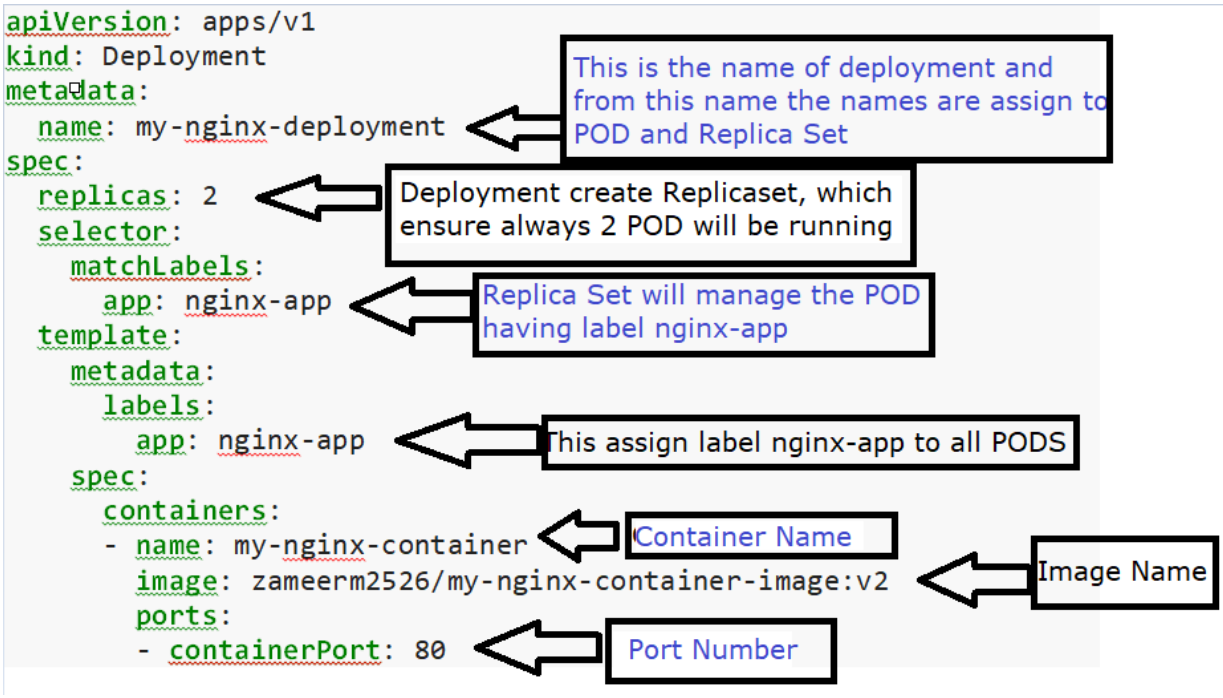
Deployments ensure that we can safely rollout new versions of our pods without outages. They also make it possible to rollback a deployment if there is some terrible issue with the new version.

So Deployments manage replica sets and replica sets manage pods and pods manage containers.

Deployment → perform upgrade of image rollout/rollback, scaling

Replicaset → replica set ensure that anytime the specified number of pods are running and provide High Avialability





Now Lets Create Deployment

vi nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
        app: nginx-app
    spec:
      containers:
      - name: my-nginx-container
        image: zameerm2526/my-nginx-container-image:v1

        ports:
        - containerPort: 80
```

kubectl apply -f nginx-deployment.yaml

kubectl get pods

kubectl get pods -o wide

kubectl get replicaset

kubectl get all

SEE CONTAINER INSIDE POD

kubectl get po -o jsonpath='{range .items[*]}{"pod: "}{.metadata.name}{"\n"}{range .spec.containers[*]}{"\tname: "}{.name}{"\n\timage: "}{.image}{"\n"}{end}'

DESCRIBE POD

kubectl describe pod my-nginx-deployment-6d4d96876b-8ntqf

pods are created on node

pods are controlled by replica set

pods have labels example app=nginx-app

pods have private IP

DELETE A POD

kubectl delete pod my-nginx-deployment-6d4d96876b-8l8qb

SEE POD WITH LABELS

kubectl get pods --show-labels

WHY WE NEED SERVICE

When we need to access our website from Internet then we need public ip and pods have private IP

So we need a service like load balancer which has a Public IP

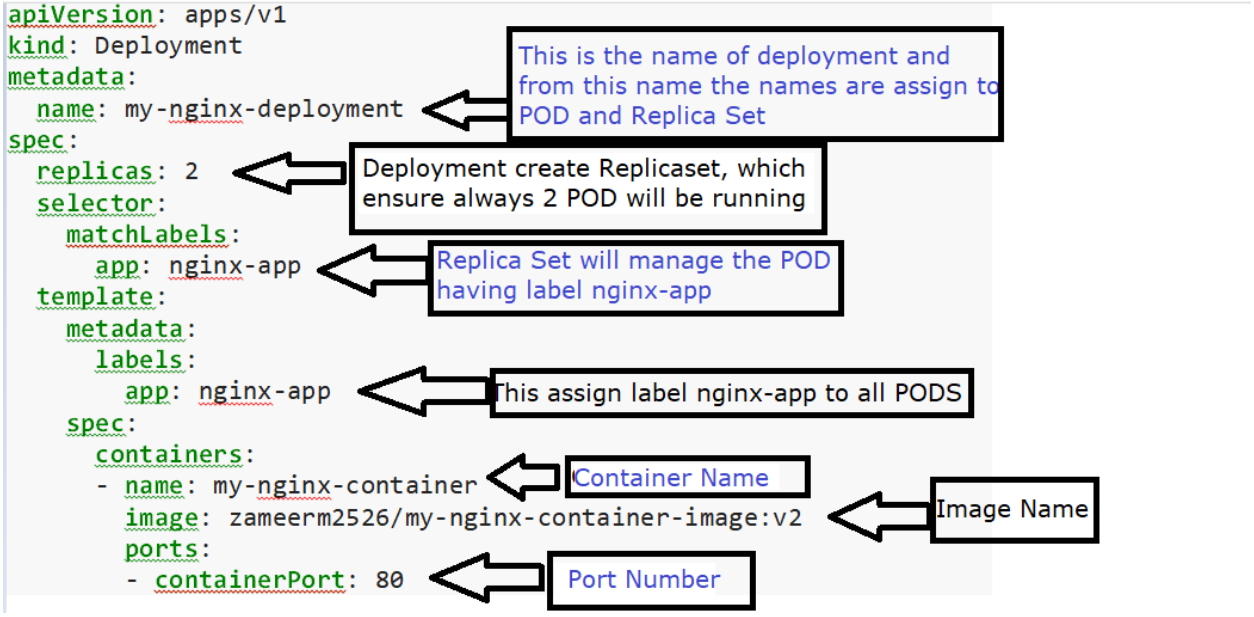
The incoming request will come to this load balancer IP

The Load balancer will forward this request to all PODS having label nginx-app

vi ngnix-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx-app
  ports:
    - port: 80
      targetPort: 80
  type: LoadBalancer
```

Below is deployment file which has label for pods



kubectl apply -f ngnix-service.yaml

kubectl get all

kubectl describe services nginx-service

Go Inside Container and See Load Balancing

kubectl get pods

kubectl exec -i -t my-nginx-deployment-85b9f6c66f-4wg9k --container my-nginx-container -- /bin/bash

cd /usr/share/nginx/html

vi index.html

<h1>1</h1>

DELETE Deployment

kubectl delete deployment my-nginx-deployment

kubectl delete svc nginx-service

kubectl get pods