**https://stor678.blob.core.windows.net/?sv=2019-12-12&ss=b&srt=sc&sp=rlcx&se=2020-11-27T20:29:12Z&st=2020-11-26T12:29:12Z&spr=https&sig=CZHNOwLgUaftUFQ8OHa4%2BrF0xC1aQgbe8kV%2FwHi5eDk%3D**

**What is Docker?**

Docker is a Platform, which packages an application and all its dependencies together in the form of containers.

Each and every app’s runs on separate containers and has its own set of dependencies and libraries

**What is Docker Engine?**

Docker Engine is simply the application that is installed on Host Machine. It works like a client-server application.

Rest API used for the comm’n between the command line and Docker daemon.

**What is Docker file?**

A Docker file is a text file which contains the all the commands, user can call the command line to assemble an image.

**What is Docker Image?**

The read only templates are the building blocks of a container.

**What is Container?**

It is a running instance of the Docker image, they hold the entire package to run the application.

**What is Containerization?**

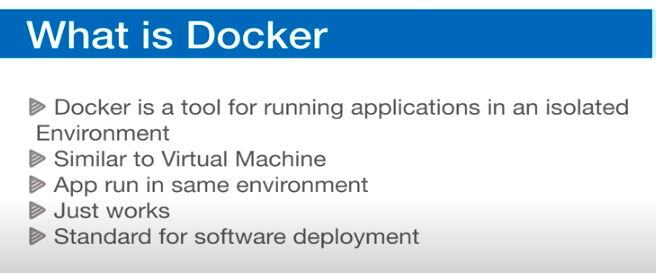
Containerization is a type of virtualization, which brings virtualization to the operating system level.

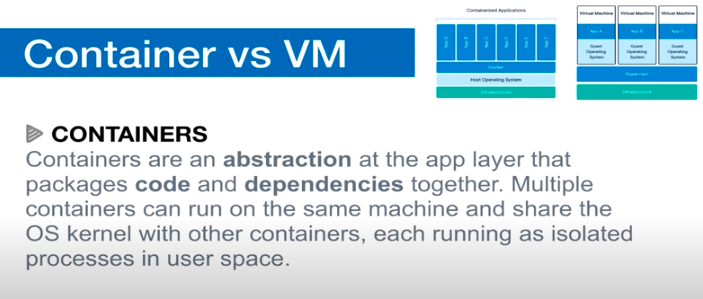
While virtualization brings abstraction to the hardware level.

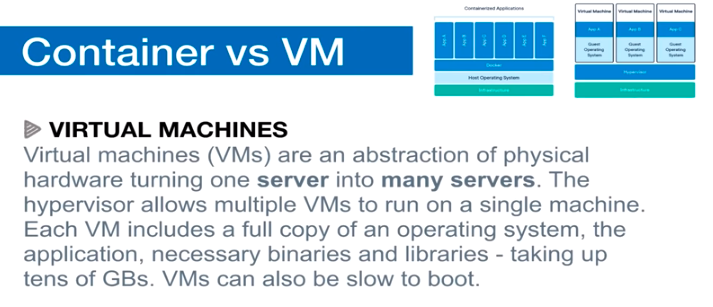
Containers have no guest o.s and use the same Host o.s. so they share the relevant libraries and resources when needed.

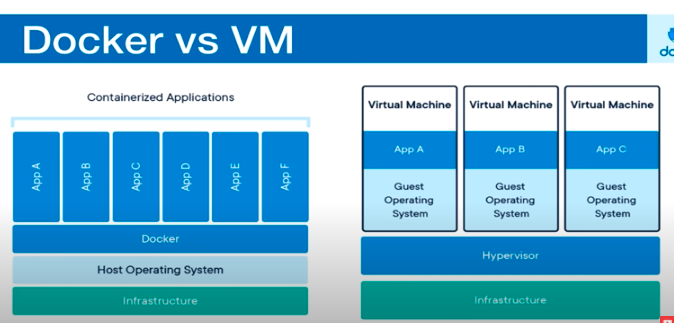
Processing and execution of applications are very fast.

Boot up a containers taking only a fraction of seconds.

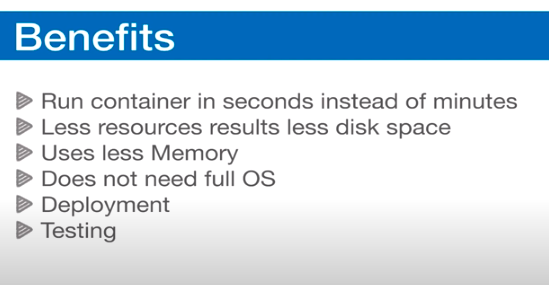


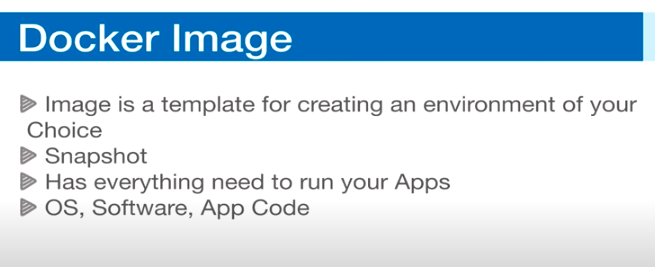


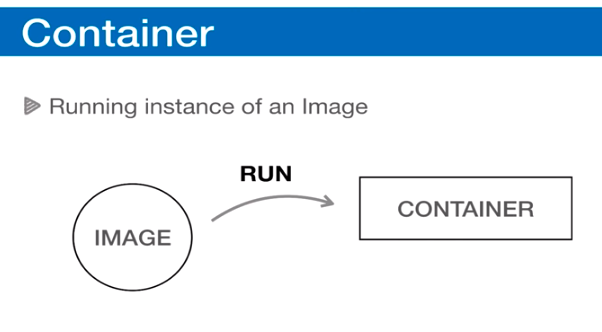




Docker Advantages







1. Pulling images from Docker Hub

$docker pull nginx

1. List of images

$docker images

1. Running Container from Image

$docker run -d nginx:latest

1. List of Containers

$docker ps or $docker container ls

1. Stop the Container

$docker stop <container id>

1. Exposing Port

$docker run -d -p 8080:80 nginx:latest

1. Exposing multiple Ports

$docker run -d -p 8080:80 -p 9090:80 nginx:latest

1. List the all container with stopped

$docker ps -a

1. Delete the containers

$docker rm <containerid>

1. To list the all container ids

$docker ps -aq

1. To delete all containers

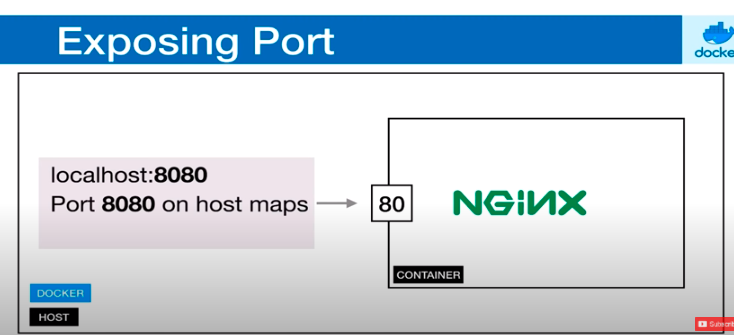
$docker rm $(docker ps -aq)

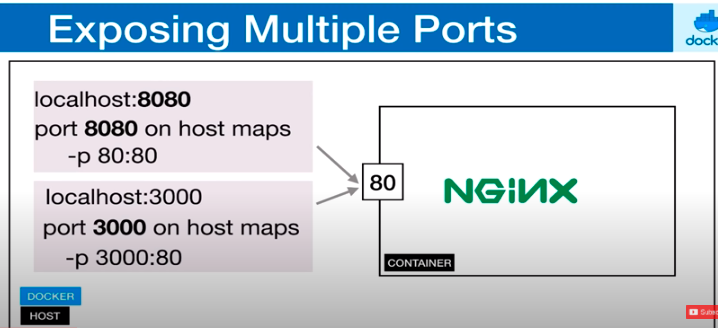
1. Naming the Container

$docker run --name website -d -p 8080:80 -p 9090:80 nginx:latest

Note: We cannot remove running container. If I want to remove container run below command

$docker rm -f $(docker ps -aq)





**Volumes**

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. Volumes are completely managed by Docker.

**Prerequisites: Create index.html file in local host**

1. To create Docker Volume (Copy the container path from the nginx image)

$docker run --name website -v $(pwd):/usr/share/nginx/html:ro -d -p 8080:80 nginx

1. Check the file data in browser
2. Enter the inside of the Container

$docker exec -it website bash

$ls -al

$cd /usr/share/nginx/html

1. Try to create files inside the container. It has only read only permissions. ro means read only so exit from the container.
2. Again create the new container with write permissions

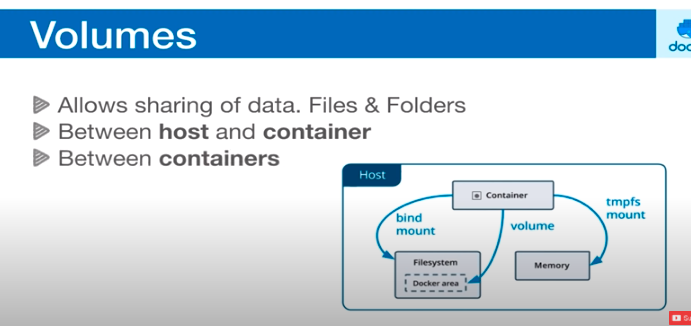
$docker run --name website -v $(pwd):/usr/share/nginx/html -d -p 8080:80 nginx

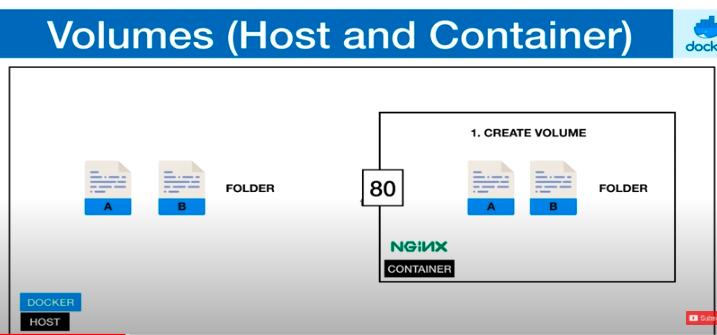
$docker exec -it website bash

$cd /usr/share/nginx/html

$touch test.html

1. So the created file appears in both(local host and container) the locations.

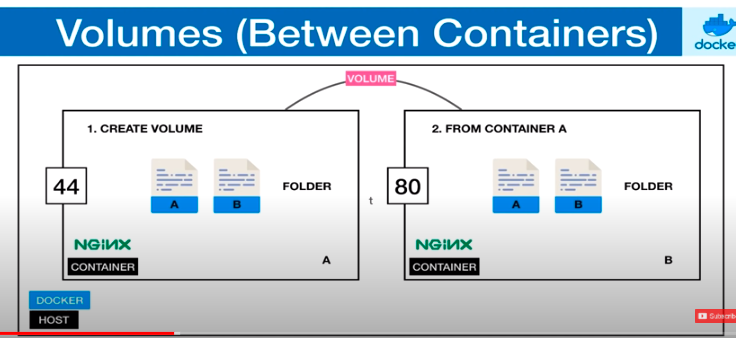


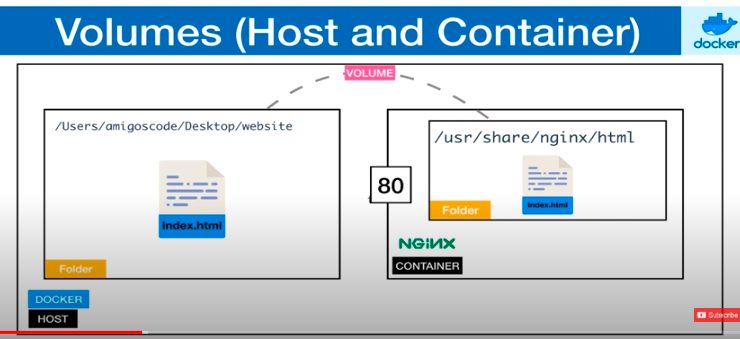


Volumes between Containers

1. Sharing the files from one container to another container

$docker run –name website-copy --volumes-from website -d -p 9090:80 nginx





What is Docker prune?

The basic usage of the command docker system prune is Remove unused data. Removes all unused containers, networks, images (both dangling and unreferenced), and optionally, volumes.

$ docker system prune

Docker File

**FROM:** Base image

**ADD:** Copy files from source on host into the to Destination of containers own filesystem

EX: ADD [Source Directory or URL] [Destination directory]

**COPY:** Copy files from a specific location into a Docker image.

EX: COPY [Source location][Destination location]

**WORKDIR:** Set the default directory for the Container (It is used to set where command defined with the CMD is to be executed)

EX: WORKDIR /app

**RUN:** It executes the command. It installs the packages.

EX: RUN apt-get update

**CMD:** CMD instruction can be override in run time. The CMD command​ specifies the instruction that is to be executed when a Docker container starts.

**ENTRYPOINT:** Entry point is not override, but it will append to previous command.

**ENV:** It is used to set the env variables. These values consists of “key value” pairs which can be accessed within the container by scripts and applications. I can define username and versions.

EX: ENV PATH=/usr/local/nginx/bin:$PATH

EX:ENV PG\_MAJOR=9.3

EX:ENV ADMIN\_USER="mark"

**Note:** CMD and ENTRYPOINT are used in run time.

DockerFile

Vi Dockerfile

FROM nginx:latest

ADD . /usr/share/nginx/html

$docker build --tag website:latest .

$docker images

$docker run --name website -d -p 8080:80 <imagename>(website:latest)

**Node js**

Install Nodejs

$nodejs -v

**Express**

Install Expressjs

$mkdir user-serviceapi

$cd user-serviceapi

$npm init

$npm install --save express

$vi index.js

<https://expressjs.com/en/starter/hello-world.html>

$node index.js (It tells the listening port)

DockerFile

FROM node:latest

WORKDIR /app

ADD <local dir> <container path>(ADD . .)

RUN npm install

CMD index.js

$docker build -t user-api:latest .

$docker images

**Running the Docker container for our API**

$docker run --name user-api -d -p 3000:3000 user-serviceapi:latest

$docker ps

Dockerignore

What is dockerignore?

The .dockerignore file allows you to exclude files from the context like a .gitignore file allow you to exclude files from your git repository.

Before the docker CLI sends the context to the docker daemon, it looks for a file named .dockerignore in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using ADD or COPY.

Node\_modules

.git

Dockerfile

$docker build -t user-api:latest .

Caching and layers

If i build the docker file multiple times, haven’t changed steps will use the cache by installing the cache packages.So the image will build fast.

$npm i -S react webpack gulp grunt

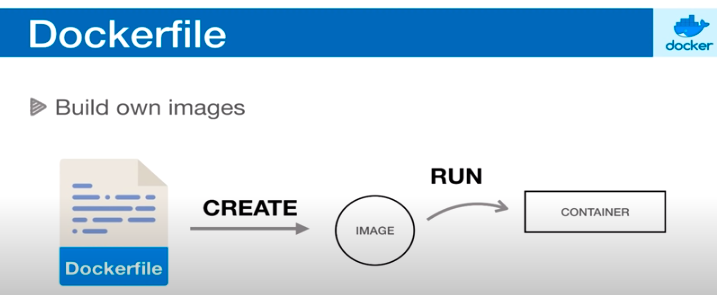
$docker build -t user-api:latest .

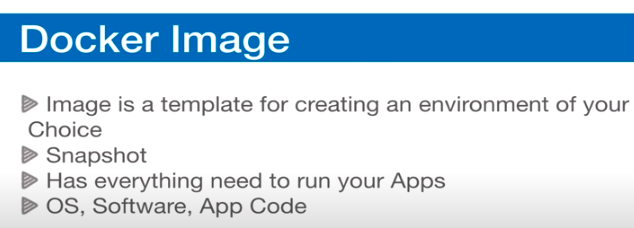
$docker run --name user-api -d -p 3000:3000 user-serviceapi:latest

$docker rm -f user-api

Need to run cache modules, because if I changed docker file then I built it.it will take the previous cache, it won’t create again.

CACHING AND LAYER PART2





Alpine

To reduce image size by using Linux Alpine distribution.

<https://alpinelinux.org/about/>

1. Download the node alpine image from the Docker hub

<https://hub.docker.com/_/node>

$docker images

$docker pull node:lts-alpine

The size is less compared to normal **node** image

1. Download nginx alpine image from the Docker hub

<https://hub.docker.com/_/nginx>

$docker images

$docker pull nginx:alpine

Using Alpine for custom Images

DockerFile

Vi Dockerfile

FROM nginx:alpine

ADD . /usr/share/nginx/html

$docker images

$docker build website:latest .

**Note:** Define the version in base image

**Docker File**

FROM node:lts-alpine

WORKDIR /app

ADD <local dir> <container path>(ADD . .)

RUN npm install

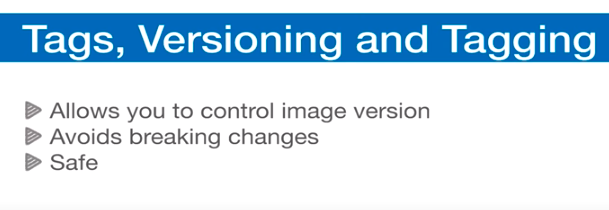
CMD index.js

**Note: Use alpine in base image**

$docker build -t userservice:latest .

$docker images

**Note:** The size will be decreased compared to previous



Using Tags with Versions

DockerFile

FROM nginx:1.18-alpine

ADD . /usr/share/nginx/html

$docker build -t website:latest .

$docker images

$docker run --name user-api -d -p 3000:3000 website:latest

**Note:** Define the version in base image. It won’t use the cache because we are changing version right.

**Docker File**

FROM node:10.22.1-alpine

WORKDIR /app

ADD <local dir> <container path>(ADD . .)

RUN npm install

CMD index.js

$docker build -t userservice:latest .

$docker images

$docker run --name user-api -d -p 3000:3000 user-service:latest

Tagging Override

**Docker File**

FROM node:10-alpine

WORKDIR /app

ADD <local dir> <container path>(ADD . .)

RUN npm install

CMD index.js

$docker build -t userservice:latest .

$docker images

Note: whenever we make the version changes use the same tag. It will override the image. Remove the no repo images.

Tagging Own Images

**Docker File**

FROM node:10-alpine

WORKDIR /app

ADD <local dir> <container path>(ADD . .)

RUN npm install

CMD index.js

$docker build -t new-userservice:latest .

$docker images

$docker tag new-userservice:latest new-userservice:1



If any changes applied in the files do the tag as 2. Tags specifying the versions

$docker build -t new-userservice:latest .

$docker images

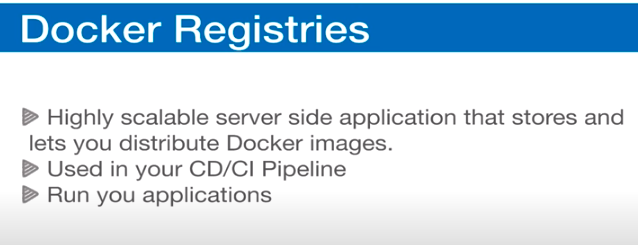
$docker tag new-userservice:latest new-userservice:2

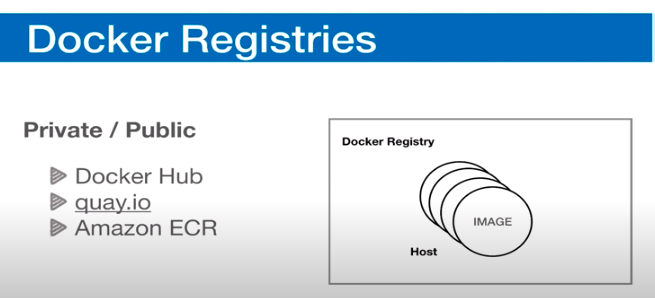
Running containers with Different Tags

$docker run --name newuserapi -d -p 8080:80 new-userservice:latest

$docker run --name newuserapi-1 -d -p 8081:80 new-userservice:1

$docker run --name newuserapi-2 -d -p 8082:80 new-userservice:2

Docker Registry



Create Docker Hub repo

1. Create repo in Docker Hub.
2. Pushing images to Docker Hub
3. First tag your images to created repo name

$docker tag new-userservice:1 19950330/website:1

$docker tag new-userservice:2 19950330/website:2

1. Login to the Docker Hub by using below command

$docker login

1. Provide Username and password
2. Push the images into Docker Hub.

$ docker push 19950330/website:tagname

$docker push 19950330/website:1

Pulling Own Images

$docker pull 19950330/website:1

**Docker Inspect**

If I know the details by running the below command

$docker inspect <image id>

$docker inspect <container id>

If I monitor the container run the below command

$docker logs <container id>

$docker logs -f help

$docker exec --help

Kubernetes

What is Kubernetes?

* Kubernetes is Open source Container orchestration tool
* Developed by Google.
* Kubernetes helps you to manage containerized applications in different environments

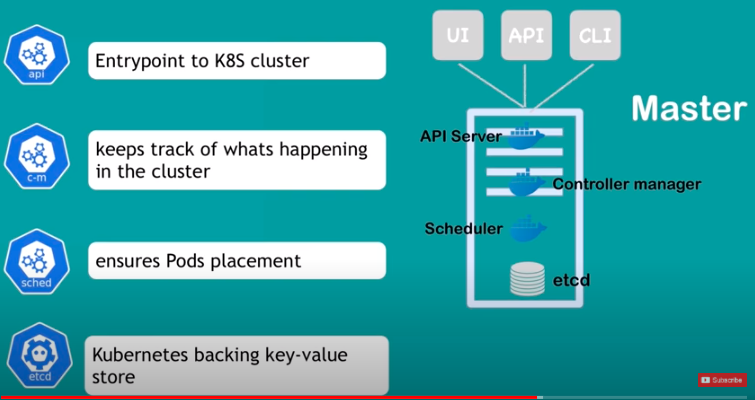
Need for container Orchestration tool

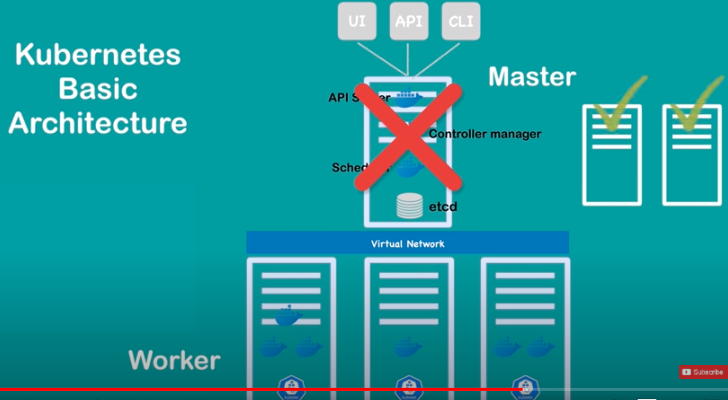
* Trend for monolith to Micro services

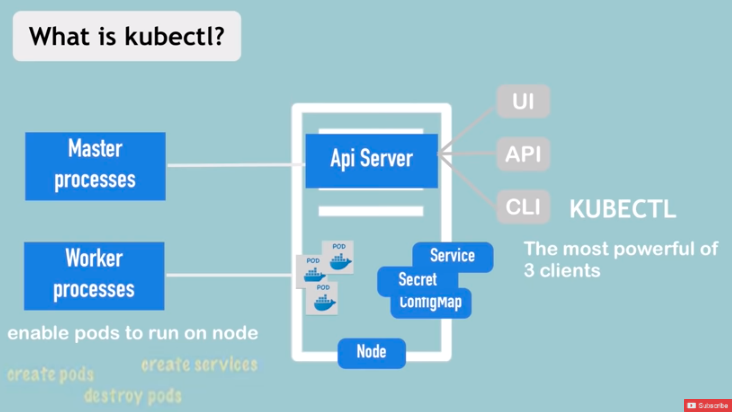
What features do Orchestration tool offer?

* High availability or no down time
* Scalability or High performance
* Disaster recovery – backup and restore

Architecture:







**Kube-proxy:**

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.

**Replica set and replication controller:**

Replica Set ensures how many replicas of pod should be running. ... The key difference between the replica set and the replication controller is, the replication controller only supports equality-based selector whereas the replica set supports set-based selector.

**Pod:**

* Smallest unit of k8s
* Abstraction over container
* Usually 1 application per pod
* Kubernetes offers out of the box Virtual network which means each pod gets own IP address.
* Each pod gets its own Ip address. Two pods can communicate each other using IP address. Which is internal IP address its not the public one.

**Service:**

* Service enables network access to a set of Pods in Kubernetes. Services select Pods based on their labels.
* A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them.
* If pod dies or crashes the IP address will change, comm’n will be not possible to both pods so service concept came.
* Service can have the Permanent IP address that can be attaches to each pod.
* Life cycle of Pod and Service not connected.
* If pod dies the service and it’s IP address will stay, you don’t have to change the endpoint anymore.
* App’s should access through browser, for this I need to create external service .
* External service is opens the comm’n from external sources like application in web browser.
* obsiously database shouldn’t open to the public requests. For that I need to create Internal Service.

**Ingress:**

* Ingress is an object that allows access to your Kubernetes services from outside the Kubernetes cluster.
* You configure access by creating a collection of rules that define which inbound connections reach which services.
* If I want to access my application with secure protocol(https) and domain name, go for Ingress. Instead of service request goes to Ingress

**ConfigMap:**

* A ConfigMap is an API object that lets you store configuration for other objects to use.
* For example Database url is stored in the configmap, this url is endpoint to the app. Because application access the database by using configmap.
* If I defined configuration in application pod, I need to do new build for pod. To avoid those steps I am creating configmaps.

**Secrets:**

* Kubernetes Secrets let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. Storing confidential information in a Secret is safer and more flexible than putting it verbatim in a Pod definition or in a container image.
* Don’t put credentials into configmap, plain text format should be insecure eventhough it’s an external configuration.
* Secrets stores the credentials as base64 encoded format.

**Volumes:**

* If the Database pod or container restarts the data would be gone so that problematic and inconvenience because I want my database data or log data persistent long time.
* It basically attaches a physical storage on a hard drive to your pod. That storage would be local machine meaning same server node on pod is running. Or remote storage, outside of the Kubernetes cluster.
* Kubernetes cluster doesn’t manage the data persistence.
* Kubernetes user or Administrator responsible for the backing up the data and replicating and managing it and making sure kept on proper hardware etc..

**Stateful set:**

* StatefulSet is the workload API object used to manage stateful applications.
* If my application pod dies when I have to restart because built a new image, basically I would have a down time where user can reach my application
* So I am replicating my application container to another node and the replica is connected to same service.
* Service is also a loadbalancer, it fetches the request and forward its to whichever pod in the list.
* For second you no need to create pod define blue print for the pod and specify how many replicas of pod you would like to run.

**Deployment:**

* Blue print is called deployment
* You are creating deployments where you are specifying the replicas for the pods and I can also scale up or scale down no.of replicas of pod .
* Pod is a layer and abstraction of pods.
* Deployment is another abstraction on top of pods
* If application pod will die, the service forwards request to another one
* DB can’t be replicated via deployment because database has a state meaning that if we have clones or replicas of DB there would all the same shared data storage there you would need some kind of mechanism that manages which pods are currently writing to that storage or which pods are reading from the storage and avoid data inconsistence. That mechanism ordered by the Statefulsets.

**Statefulsets:**

* This mechanism specifically for the Database applications.
* Like mysql, elastic search should be create it by statefulsets not deployments.
* Statefulsets just like deployments would take care of replicating the pods and scaling them up or down and making sure the DB reads and write are synchronized so that no database inconsistency are offered.

**Cluster Role:** Where give the permissions to the user is cluster role. Default it get the cluster-admin permissions, We can create custom roles in this roles

**Cluster Rolebinding:** We bind the cluster role with the user, will use the cluster role binding

Practice:

Creating Deployment

$Kubectl create deployment nginx-depl --image=nginx

$kubectl get deployment

$kubectl get pod

$kubectl describe pod <pod name>

$kubectl delete deployment <deployment name>

$kubectl create deployment mongo-depl --image=mongo

$kubectl get deployment

$kubectl get pod

$Kubectl logs <nginx pod name>

$kubectl delete deployment <deployment name>

Debugging Pod

$Kubectl logs <nginx pod name>

$Kubectl logs <mongo pod name>

$kubectl exec -it <pod name> -- bin/sh (Enter some commands and debug the container)

KUBECTL APPLY -F

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.16

ports:

- containerPort: 8080

$vi nginx.yaml

$kubectl apply -f nginx.yaml

$vim nginx-deployment.yaml

$kubectl apply -f nginx.yaml

$kubectl delete deployment <deployment name>

Deploy Mongo DB and Mongo Express

Mongo DB

$kubectl get all | grep mongodb

1. Create Secrets

$kubectl apply -f secret.yaml

$kubectl get secret

1. Create Mongo db deployment

$kubectl apply -f mongo.yaml

$kubectl get pod

$kubectl get pod -o wide

1. Create internal service

$kubectl apply -f internalservice.yaml

$kubectl get service

$kubectl describe service <service name>

Env variables

* Admin username
* Admin password

Mongo Express

1. Create ConfigMap

$kubectl apply -f ConfigMap. Yaml

$kubectl get secret

1. Create Mongo Express deployment

$kubectl apply -f mongoExpress.yaml

$kubectl get pod

$kubectl get pod -o wide

1. Create External service

$kubectl apply -f Externalservice.yaml

$kubectl get service

$kubectl describe service <service name>

Env variables

* Admin username
* Admin password
* Configmap