**Docker**: Docker is a Paas product and it uses container to run the apps.

* **To run the container, we will create the image**
* **We will have light weight operating system inside the container**

**What is container?**

It is a way to package application with all the necessary dependencies and configuration

It is a running instance of image

**Docker Images  
  
Check images:**   
  
 docker images

**To pull images:**

Docker pull <image-name>

**Docker image:** it is a ready only template that contains a set of instructions for creating a container that can run docker platform

* We use image template to create the contains and run them.
* Image is read only template to make it into running state we will use the containers
* When we try to add the data in the container that will be stored under writable layers, if we restart the containers that data will be lose. to avoid this, we use the concept of **volumes**.
* While creating the container docker will create the writable layer on top of the image.
* Any Docker image will have multiple layers, suppose that image is created longtime ago it will have more layers **but recently created image will have less layers**.
* Whenever we add data to the container that will be treated as another layer,
* To check data or additional libraries added to image we use this Docker **history <image-name>**

**To pull particular image version:**  
  
Docker <image-name>:version  
Docker pull mysql:5.7.7

**Remove image:**

Docker rmi <image-name>

**Docker commands:**

**Docker login: you should have the docker hub account  
  
Docker logout: to logout of the hub**

**Docker search <image-name> : it will show the images**

**Docker inspect image <image-name>: to see details of the pulled image**

**Docker images: to list out the images  
Docker ps -a: to list out all the containers  
  
To create the container and run it :  
  
Docker create <image-name>  
  
Docker start <container id>**

**Docker stop <container-name>  
  
To provide specific name to container:  
  
Docker create --name <container-name> <image-name>**

**Create,start, run the container:   
  
Docker run --name <container-name> <image-name>**once you execute the command container will come into running state and will be loggedin to it **To avoid container login while running the container we use detach mode**

**Docker run -d --name <container-name> <image-name>  
  
Shutdown the irresponsive container:  
  
docker kill <container-name>**

Loginto the container:  
  
Docker exec -it <container-name> <shell-name>

**Docker exec -it nginx-co /bin/bash**

Once you loginto the container only few commands will workout as it is lighweight operating system

**Note: you can’t delete the image if the container is running on it**

**Docker Networking:**

Suppose if a container is having backend application and another container is having frontend application. To have communication between two containers there should be some network establishment.

* 3 types of networking available in Docker  
    
  Bridge  
  Host  
  None

**Command to see networks available:**   
  
 **docker network ls**

**Bridge:** it is a default network inside the docker host whatever container you created it will be connected to bridge network only to have communication between containers and to have communication outside network also.

**Host:** if the container is created using host network the application inside the container will be shown as it is running in docker host  
  
**None:** if you don’t want any network communication to your container use this. (testing purpose)

**Bridge Network:**

**Syntax:** Docker run -d --name <container-name> -p <port-number> <image-name>

Docker run -d --name ngn -p 8080:80 nginx

This will be accessed by Docker host under bridge network

**Host Network:**

We don’t have option like port number in this network and it will be run on docker host only.

**None:** it is neither connect internally nor externally to the containers

**Docker Volumes**

To overcome the data lost in writable layers from Image we use docker volumes

**Syntax to check volumes and inspect:**

Docker volume ls

Docker volume inspect <volume-name/id>  
  
\***Default location for docker volumes**: **/var/lib/docker/volumes/**

**Create a volume:**docker volume create tst1 **to mount the volume to the newly creating container:  
  
Syntax: docker run -it –name=<container-name> --mount source=<volume-name>,destination=<path> <image-name>   
  
docker run -it –name=srv1 --mount source=tst1,destination=/data centos**

**docker run -it --name <container-name> -v <voume-name>:/data <image-**

**name>**

**Share the the existing volume container to newly creating container:**  
docker run -d -it --voulmes-from <volume-name> --name <container=name> <image-name> /bin/bash

**Docker run -d -it –volumes-from srv2 –name src nginx /bin/bash**

**To create the volume in present working directory use below commands:  
  
docker run -it --name <container-name> -v “$(pwd)”:/data <image-name>**

**To delete unused volumes:**

**Docker rm <volume-name/id>  
  
Docker volume prune :** this command used when any volume is not being used by atleast one container

**Note: when you delete the containers volumes still available**

**Dockerfile**

To build custom image as per the requirements we use Dockerfile for that.

**FROM:** refer to which is base image

**ADD:** usually developers build the code and generates the artifact, that artifact is being integrated to the image using **ADD or COPY** attribute.

**Difference between ADD & COPY:  
  
ADD is useful when the data is in some URL  
  
ADD is useful when the data is in the zip file format**

**COPY is not compatible with the above these**

**CMD:** which attributes to be executed for the application

**ENTRYPOINT:** Application startup path will be mentioned in this when container started.

**ENV:** if you want to define any env variables inside the container use this.

**EXPOSE**: which port need to be exposed of the container  
  
**MAINTAINER &LABEL:** who is maintaining the image

**RUN**: if you want to install any add on like notepad,vi editor on image while preparing the custom image

**USER:** sets the user which is to run the container **VOLUME:** will be used when any directory will be mounted as volume on the host machine to the container.

**WORKDIR:** sets the path where the command, defined with CMD to be executed

**Sample Dockerfile:**FROM ubuntu:18.04

LABEL maintainer=”shaiksaida843@gmail.com”  
  
RUN apt-get update && apt-get -y install apache2  
EXPOSE 80  
  
ENTRYPOINT [“/usr/sbin/apachectl”]

CMD [“-D”, “FOREGROUND”]  
  
  
**Build Command:**   
 **Syntax: docker build -t <image-repository\_name> .  
  
docker build -t saida/ubuntu .**

Docker run -dt –name <container-name> -p <portnumber> <image-name>

**Multi Container App using Docker :  
  
  
GitHub URL : https://github.com/narayanacharan/mern\_docker\_demo  
  
MongoDB -Container:**docker run -dt -p 27017:27014 -e MONGO\_INITDB\_ROOT\_USERNAME=admin -e MONGO\_INITDB\_ROOT\_PASSWORD=password -e

PWD=/ -v mongo-db:/data/db --name mern\_library\_nginx\_mongodb\_1 --net   
library-mern-api mongo **MongoDB-expres::**

Docker run -dt -p 8081:8081 -e ME\_CONFIG\_MONGODB\_ADMINUSERNAME=admin -e ME\_CONFIG\_MONGODB\_ADMINPASSWORD=password --net library-mern-api --name mern\_libraray\_nginx\_mongo\_express\_1 -e ME\_CONFIG\_MONGODB\_SERVER=mern\_library\_nginx\_mongodb\_1 -e ME\_CONFIG\_BASICAUTH\_USERNAME=admin -e ME\_CONFIG\_BASICAUTH\_PASSWORD=admin123456 mongo-express

**Docker Compose:**  
  
we usually create and run containers using run, create commands by using docker command line  
  
we can automate the these commands by using this **Docker Compose** ifyou have multiple containers  
  
Brief : it allows you to define and run multi container applications

It is also allows you to define services, networks, volumes that make up your application in a declarative YAML file.

`

**Troubleshooting containers:**  
  
docker logs <container-name>

Docker exec -it <container-name> /bin/bash

Docker rename <old-container-name> <new container>

**Docker Swarm:** when you have high load on the container once it is crashes you have to create another container and run it manully so it will be tough to run every time to **overcome this issue we have docker swarm.** It will create multiple replicas of the application  
  
  
To setup the docker swarm you should have the multiple docker hosts and one **host will be like swarm manager** among docker hosts and others will be slaves or worker nodes

**docker swarm init :** this command needs to run on one docker host to make it as swarm manager and for joining the other hosts as worker nodes you need to execute the this command : **docker swarm join –token <token>**

**docker service:** suppose if you want to run your application on multiple containers you use this.

**docker service create –replicas=3 <image-name> : it should be execute on manager cluster or host**

**docker service is a collection of multiple container defined inside the yaml code**

**Docker stack: it is group of services**

**Swarmpit: whatever we created volumes,container etc they will be shown in graphical user interface**