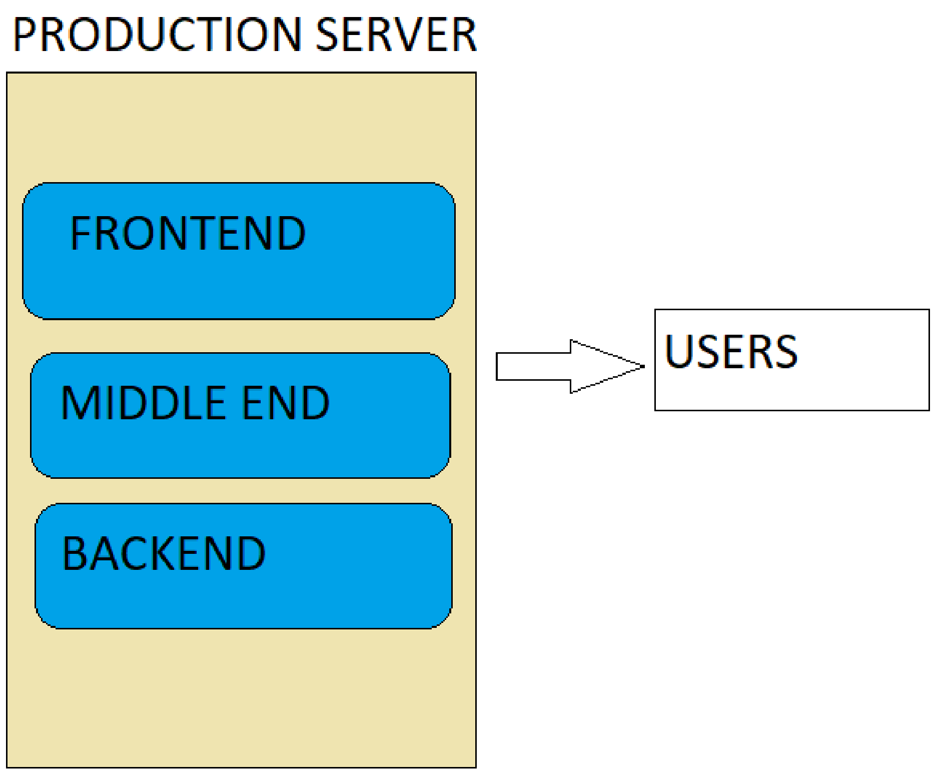
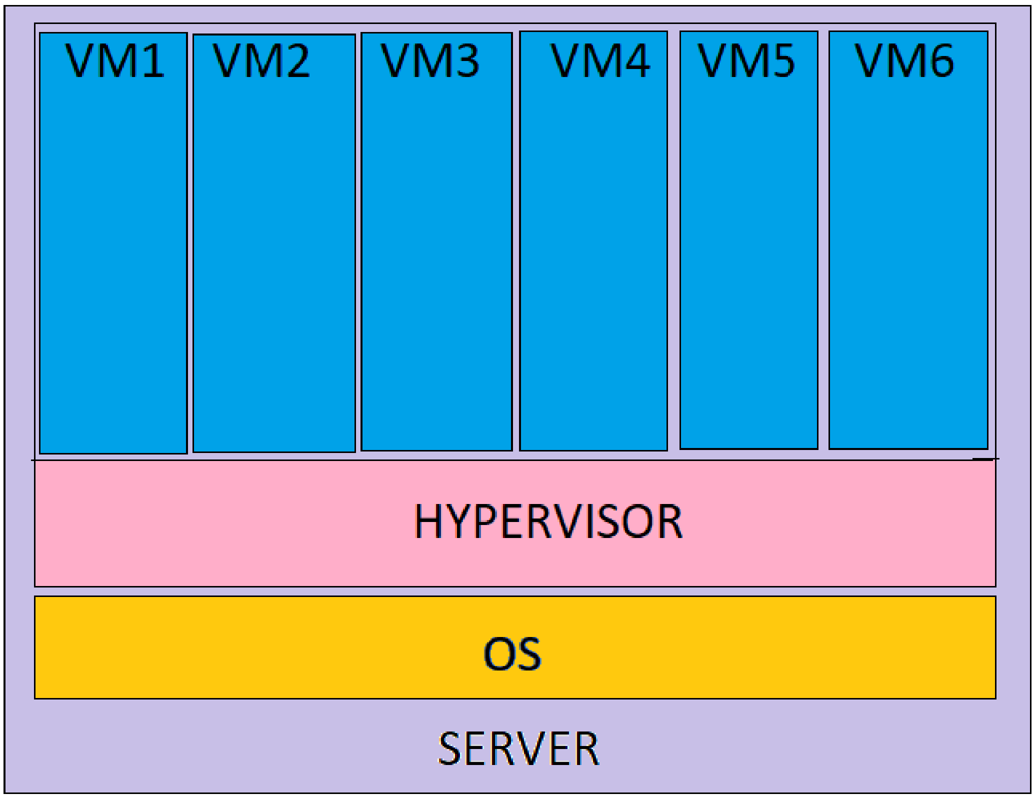
DOCKER

PROBLEMS BEFORE DOCKER



When the whole application is deployed in the production. It may not work fine and users may not access the application.

It is because a single production server may not support for three different technology i.e frontend, middle end and backend due to different dependencies and libraries that they support.



server: server is a computer that provides a service to another computer and its users.

OS: operating system is an interface between computer user and computer hardware.

Hypervisor: hypervisor is a software that can install virtual machines on your physical server.

Ex: vmware, xen, oracle-virtualbox

Virtual machine [VM]: virtual machines are virtual environments which function as virtual computer system, has their own cpu, memory and hardware.

To overcome the drawback of single production server we go for virtual machine's.

In these virtual machine’s we can deploy our application but the problem is here we are not using the resource of these virtual machine’s efficiently.

To overcome these problems we go for Docker.

Introduction to Docker

What is Docker?

Docker is a tool which is used to automate the deployment of applications in light-weight containers. So that applications can work efficiently in different environments.

It was invented by Solomon Hykes in 2013.

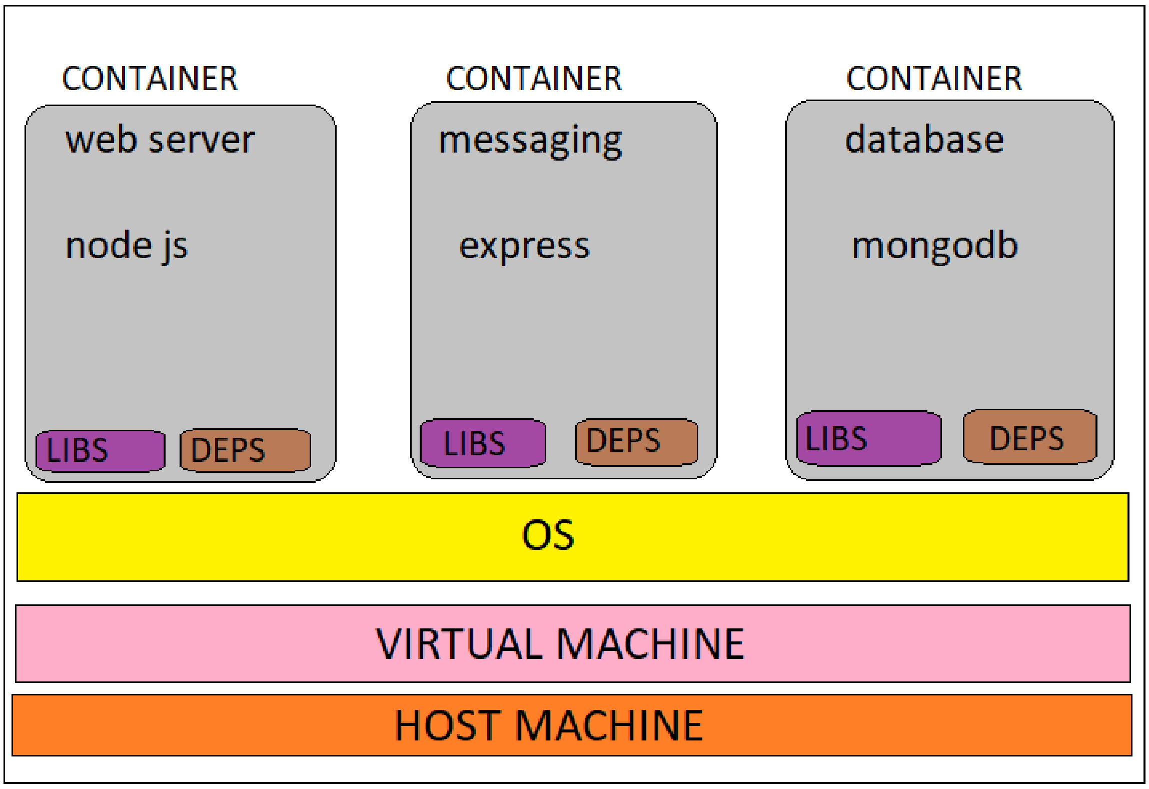
It is written Go-lang

Website: docker.com

Why we need Docker?

- Consistent and isolated environment
- Rapid application deployment
- Better portability
- Ensures scalability and flexibility

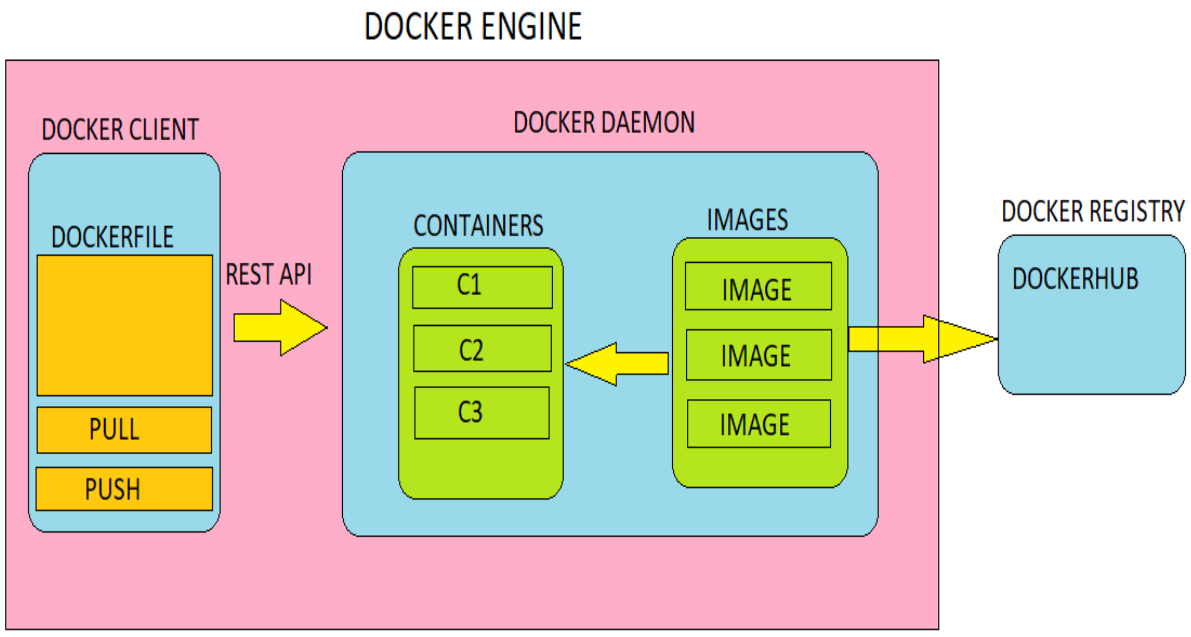
DOCKER CONTAINERS



Containers are light weight software packages that consists of all the dependencies required to run an application.

- They do not have complete OS.
- They use the resources from base OS/VM that they are running on.

STRUCTURE OF DOCKER



Docker engine or docker is the base engine installed on your host machine to build and run containers using docker components and services.

It uses a client server architecture.

Docker client and server communicate using REST API.

Docker client is a service which runs a command, the command is translated using REST API and is sent to the docker daemon. Then docker daemon checks the client request and indicates with the operating system in order to create or manage containers.

The images created in the docker daemon can be stored in the docker registry [dockerhub].

Components of Docker

1.Docker client and server

2.Docker images

3.Docker container

4.Docker registry

Docker client: Docker client uses **commands** and **REST APIs** to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.

Docker images: Docker images are the **read-only binary templates** used to create Docker Containers.

Docker containers: Container is a software that consists of all the dependencies required to run an application.
Multiple containers can run on the same hardware.

Docker registry: Docker Registry manages and stores the Docker images.

There are two types of registries in the Docker -

Pubic Registry - Public Registry is also called as **Docker hub**.

Private Registry - It is used to share images within the enterprise.

Dockerfile: Dockerfile is a script that contains a set of instructions for building a docker image.

Docker image are basis for containers, which are lightweight, portable, and self-sufficient environments.

Here’s a step by step explanation of writing a Dockerfile.

1.choose a base image:

Start by specifying the base image for your application. This is the starting point for your docker image.

FROM ubuntu

2.Update and install dependencies:

Update the package list and install any necessary dependencies for your application

RUN apt update\ apt install nginx -y\apt install git -y

3.Set the working directory:

Define the working directory inside the container where your application code will reside.

WORKDIR /app

4.Copy Application files:

Copy your application code into the container

Copy ./app

5.Expose ports:

Specify any ports that your application will expose.

```
Expose 80
```

6. Define Environment variables:

Set environmental variables if needed

```
ENV DEBUG=False
```

7.Run Applications:

Specify the command to run your application when the container starts.

```
CMD [“nginx”,”-g”,”daemon off;”]
```

8.Build the docker image:

Save the dockerfile and run the following command in the Terminal to build the docker image.

```
Docker build -t image_name:tag .
```

9.Run docker container:

After building the image, you can run a container based on that image

```
Docker run -itd -p 80:80 image_name:tag
```

Docker Volumes

The data doesn't persist when that container no longer exists and it is difficult to get back the data.

To overcome these problems that we had regarding the data of container we will go for docker containers.

Volumes: It is used to store the data of a container.

It is an independent file system entirely managed by docker and it exists as normal file or directory on the host machine.

Docker has two ways to store data of container on host machine

1.Bind mounts

2.Volumes

1.Bind mounts: this kind of volume will be attached to container by creating a directory on the host machine.

2.Volumes: It is managed by docker and these volumes will get stored in(var/lib/docker/volumes) inside volumes dir go to volume and the _data

Commands used in docker volumes

1.To create a volume

```
docker volume create volume_name
```

2. To list the volumes

```
docker volume ls
```

3. To delete the volume

```
Docker volume rm volume_name
```

DOCKER COMPOSE

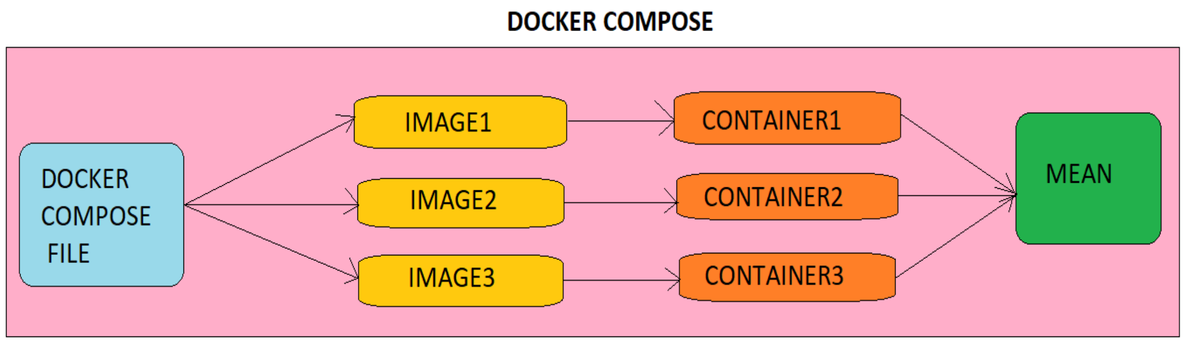
Docker compose is used to run multi-containers application.

Each container will run a stand-alone application and it can communicate with other containers present in the same host.

Docker-compose file will be written in YAML [Yet another markup language] language.

Extension of docker-compose file is docker-compose.yml

STRUCTURE OF DOCKER COMPOSE



M---> MONGO DB
E----> EXPRESS JS
A---> ANGULAR JS
N---> NODE JS

INSTALLATION OF DOCKER COMPOSE

1.Install the docker compose

curl -SL https://github.com/docker/compose/releases/download /v2.24.5/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose

2.To give execution permission

```
sudo chmod a+x /usr/local/bin/docker-compose
```

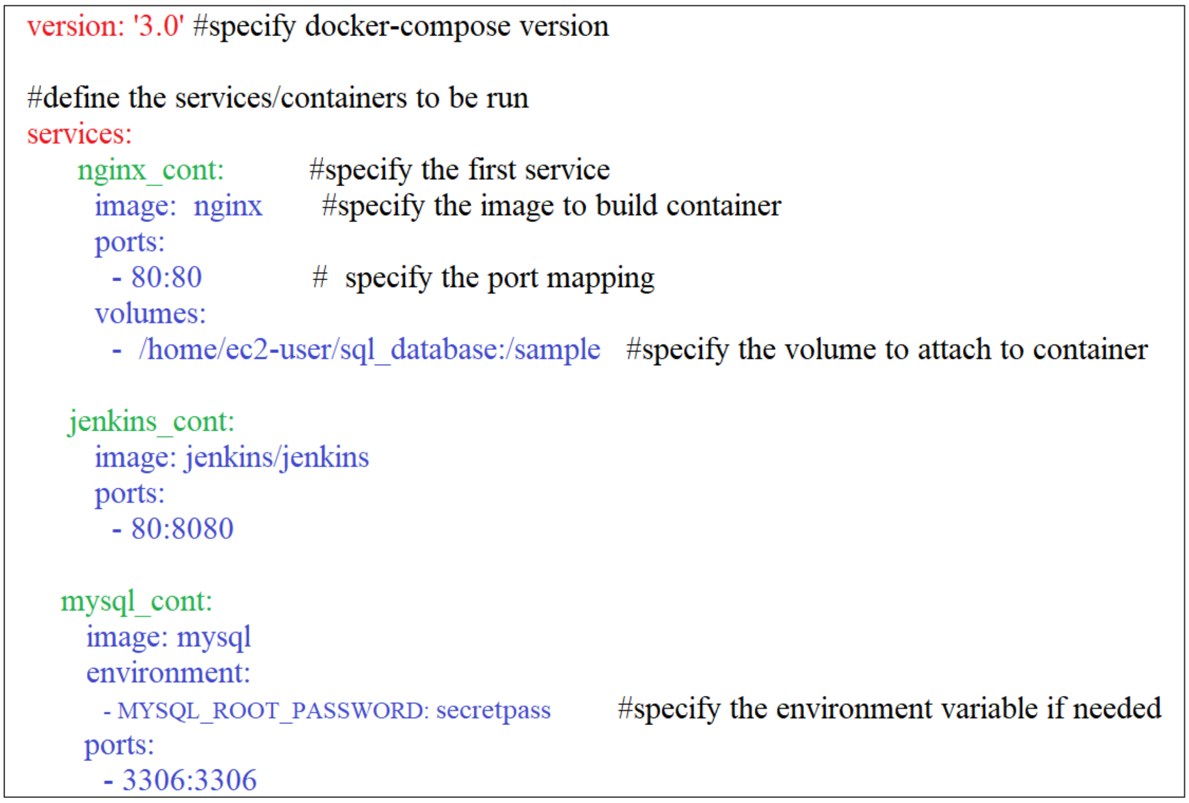
3. To check given execution permission

```
sudo ls -l /usr/local/bin/docker-compose
```

4.To check the version of docker-compose

```
docker-compose –version
```

Docker-compose file

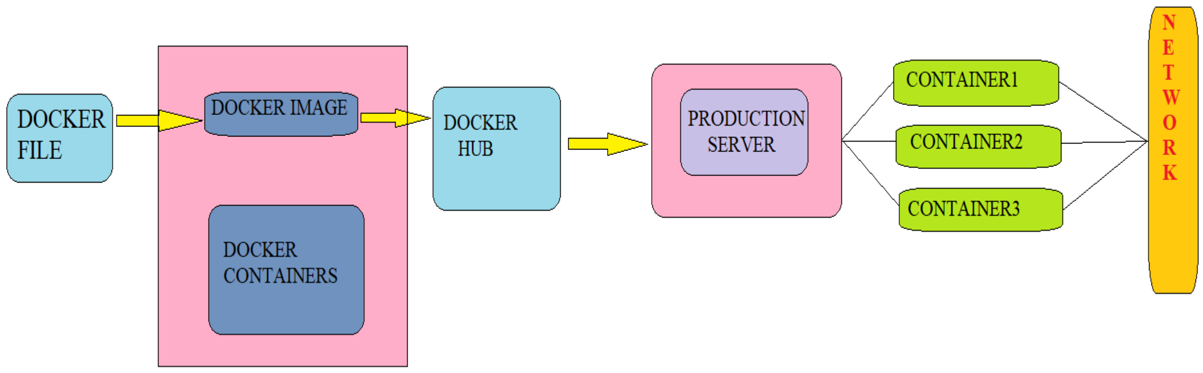


```
version: '3.0' #specify docker-compose version

#define the services/containers to be run
services:
      nginx_cont:         #specify the first service
        image:  nginx      #specify the image to build container
        ports:
          - 80:80           #  specify the port mapping
        volumes:
          -  /home/ec2-user/sql_database:/sample   #specify the volume to attach to container

     jenkins_cont:
        image: jenkins/jenkins
        ports:
          - 80:8080

    mysql_cont:
       image: mysql
       environment:
         - MYSQL_ROOT_PASSWORD: secretpass       #specify the environment variable if needed
       ports:
         - 3306:3306
```

5. Run the docker-compose file

```
docker-compose up -d
```

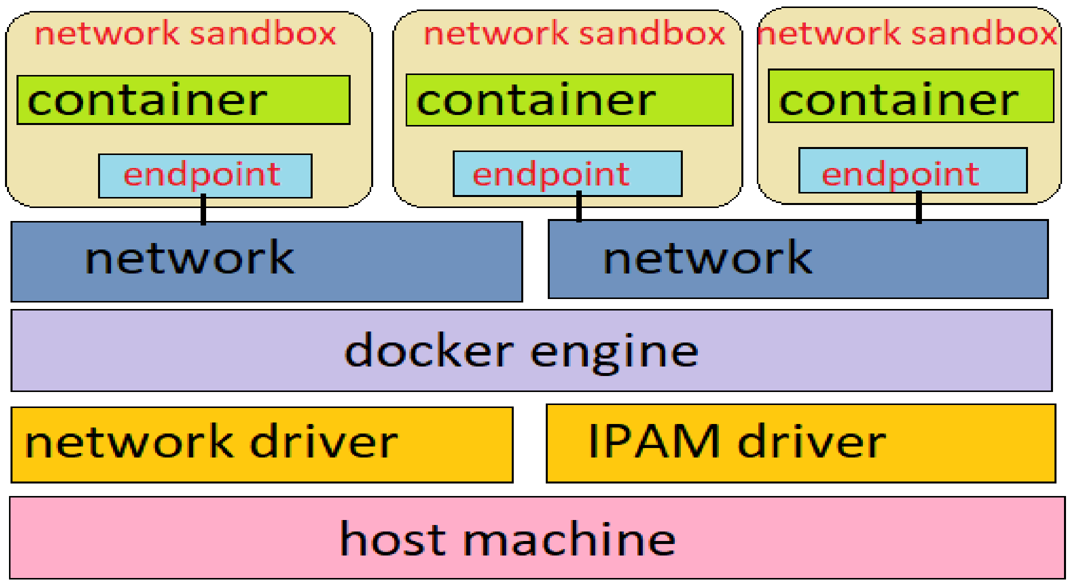
DOCKER NETWORKING



Network is a group of two or more devices that can communicate with each other either physically or virtually.

The docker network is a virtual network created by docker to enable communication between docker containers.

CONTAINER NETWORK MODEL



NETWORK DRIVERS

1.Bridge

2.Host

3.None

1.BRIDGE: If you build a container without specifying the kind of driver the container will be created in the bridge network.

2.HOST: containers will not have any ip address they will directly created in the system network which will remove the isolation between docker host and the container.

3.NONE: ip address won't be assigned to containers, these containers are not accessible to us from the outside or from any other container.

DOCKER NETWORKING COMMANDS

1.To list the network drivers

```
docker network ls
```

2.To create a network

```
docker network create - -driver <driver_name> network_name
```

3.To connect docker container to an existing network

```
docker network connect network_name container_name/id
```

4.To know the details of your network

```
docker network inspect network_name
```

5.To disconnect the container from the network

```
docker network disconnect network_name container_name/id
```

6.To delete the created network

```
docker network rm network_name
```

7.To remove all unused network

```
docker network prune
```

8.To connect a container to the created network

```
docker run -itd - -network=network_name - -name container-name image_name bash
```

9.To create a container and connect it over a network

```
docker run - -rm -itd - -net network_name - -name container_name image_name bash
```

rm: to remove the container as soon as it stops

--itd: stands for terminal interactive and run in background

--net: used to set the network of the container

--name: to give the name for the container

10.To enable port for the container

```
nc -lp port no
```

11.To connect to the container via port no

```
nc container port no
```

MULTI STAGE BUILD

Multi stage build allows you to use multiple images to build a final product.

In a multi stage build you have single dockerfile, you can define multiple images inside it to help to build the final product.

Dockerfile:

```
# use ubuntu as base image
FROM ubuntu  as build
#update the packages
RUN apt update
 # install java-17
RUN apt install openjdk-17-jdk
# create app directory
WORKDIR /usr/src/app
 # copy the source files
COPY . .


# use nginx as a second base image
FROM ngix
# copy build files to nginx
COPY --from=build  /usr/src/app  /usr/share/nginx/html
 # start the nginx service
CMD ["nginx","-g","daemon off;"]
```

Command:

To create images from above dockerfile

sudo docker build -t image_name .

Installation of docker desktop

Website: https://www.docker.com/products/docker-desktop/

Create Dockerhub account

Website: https://hub.docker.com/

Installation of Docker on Linux

1.update the packages

```
sudo yum update
```

2.Install docker

```
sudo yum install docker -y
```

3.Enable the docker

```
sudo systemctl enable docker
```

4.Start docker service

```
sudo systemctl start docker
```

5.Check the status of docker

```
sudo systemctl status docker
```