DOCKER

Docker is a set of platforms as a service (PaaS) products that use operating system level virtualization to deliver software in packages called containers.

Containers are isolated from one another and bundle their own software, libraries, and configuration files. All containers are run by a single operating system kernel and therefore use fewer resources than a virtual machine.

DIFFERENCE B/W DOCKER CONTAINERS & VIRTUAL MACHINES

DOCKER CONTAINERS

→ Docker containers contain binaries, libraries, and configuration files along with the application itself.

Date : / /
Page No.

→ They don't contain a guest OS for each container and rely on the underlying OS kernel, which makes the containers lightweight

→ Containers share resources with other containers in the same host OS and provide OS-level process isolation.

VIRTUAL MACHINES

→ Virtual Machines (VMs) run on hypervisors, which allow multiple virtual Machines to run on a single machine along with its own operating System.

→ Each VM has its own copy of an operating system along with the application and necessary binaries, which makes it significantly larger and it requires more resources.

→ They provide hardware-level process isolation and are slow to boot

Important Terminologies by Docker

(i) Docker Image

→ It is a file, comprised of multiple layers, used to execute code in a Docker container.

→ They are a set of instructions used to create docker containers.

(ii) Docker Container

→ It is a running instance of an image.

→ Allows developers to package applications with all parts needed such as libraries and other dependencies.

(iii) Docker File

→ It is a text document that contains necessary commands which on execution helps assemble a Docker image.

→ Docker image is created using a Docker file.

Date: / /
Page No.

(IV) DOCKER ENGINE

→ The software that hosts the containers is named Docker Engine.

→ Docker Engine is a client-server based application

→ The docker engine has 3 main components:

- **Server**: It is responsible for creating and managing docker images, containers, networks, and volumes on the Docker. It is referred to as a daemon process.
- **REST API**: It specifies how the applications can interact with the server and instructs it what to do.
- **Client**: The client is a docker command-line interface (CLI), that allows us to interacts with Docker using the docker commands.

(V) DOCKER HUB

→ Docker hub is the official online repository where you can find other Docker images that are available for use.

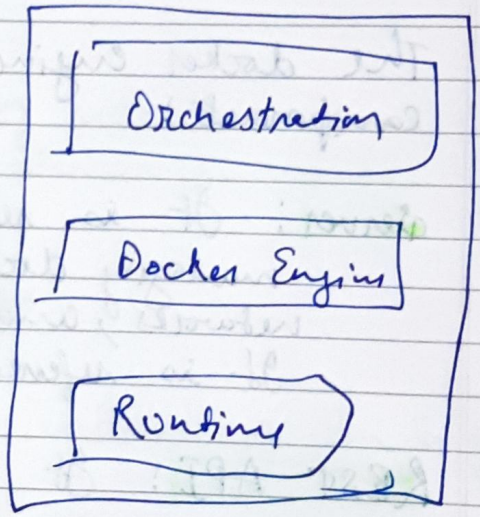
WSL → Windows Sub system for Linux

Date : / /
Page No.

→ It makes it easy to find, manage, and share containers images with others.

COMPONENTS OF DOCKER.

1. RUNTIME



Also Containers - runtime is a low-level component of a container engine that mounts the container and work with OS kernel to start and support the containerization process.
There are two types!

→ The most common runtime is RunC.
RunC is a low-level runtime.
Its role is to work with system and to start and stop our containers.

→ Containerd

Containerd is a container runtime that

manages the lifecycle of a container on a physical or virtual machine (host).

It is a daemon process that creates, starts, stops and destroys containers.

It is also be able to pull container images from container registries, mount storage, and enable networking for a container.

It also manages runC.

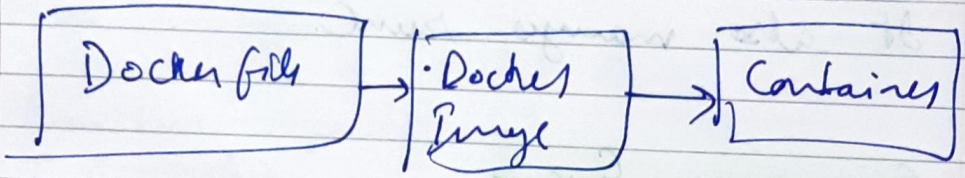
2. CONTAINER ENGINE

Docker engine is an open source containerization technology for building and containerizing your applications.
Docker engine acts as a client server application with:

- A server with a long-running daemon process i.e. dockerd.
- APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon. aka Docker Engine REST API
- A command line Interface (CLI) client i.e. docker.

→ **Docker Daemon (Dockerd)**

It is a persistent background process that manages Docker images, containers, networks, and storage volumes. The Docker daemon constantly listens for Docker API requests and processes them.



DOCKER COMMANDS

(i) **docker run hello-world**
(image name)

to check if the docker is installed or not

(ii) **docker images**

It will list all images download/stored on our local machine.

Date : / /
Page No.

(iii) **docker run -it Ubuntu**

It will run ubuntu in a container and we can use it.

~~ubuntu~~ i
it → It means interactive environment

(iv) **docker pull imagename.**

It will download the image from the registry.

Eg.

docker pull ubuntu (latest version if version not specified)

or

docker pull Ubuntu:16.04 (for specific version)

(v) **docker ps**

to check which containers are running.

(vi) docker container ls

This will also ~~show~~ list the containers.

(vii) docker container exec -it container_id bash

this will attach the bash shell to the running container.

Note: the container mentioned in command should be running.

Eg.

docker container exec -it 54799351d94c bash

(viii) docker stop container_id

It will stop the container.

first use docker ps then copy container id from there and use this command.

Date : / /
Page No.

(ix) docker ps -q

It will list all the containers including stopped containers as well.

(x) docker rm container ID

It will remove the container

(xi) docker container prune -f

It will delete all the containers that are stopped.

(xii) docker rmi imagename -f

It will remove docker img mentioned in the command.

(xiii) docker run -d -p newport: default port container name

this command is used to access a container locally on the computer

this also known as port forwarding

Eg.

docker run -d -p 8080:80 nginx

d → detach mode

p → port

(xiii) docker commit -m "message" container ID new image name: version Tag

to save changes in a container as a new image so that when the image is shared to someone the changes will be there.

Eg.

docker commit -m "added file.txt" 062a437064f3 name.ubuntu:1.0

(xv) docker images -q

It will only list the ID's of the images

(xvi) docker rmi $(docker images -q) -f

It will delete all the images at once.

Note: all the containers should be stopped first.

Date : / /
Page No.

LAYERS / DOCKER IMAGE LAYERS

Docker Images consist of layers.

→ Each layer is an image itself.

→ Each layer stores the changes compared to image its based on.

→ Each instruction in a Dockerfile results in a layer.

→ Layers are used to avoid transferring redundant information and skip build steps which have not changed.

HOW TO CREATE A DOCKER IMAGE.

→ First create a Dockerfile using the following command.

touch Dockerfile

→ Edit the Dockerfile with text editor and write in it.

~~FROM Ubuntu~~ → ~~Base image~~

FROM ubuntu → Base image

MAITAINER : MOHD IMRAN <email id>

RUN apt-get update → (write command to run)

CMD ["echo", "Hello world"] → (this specifies what executable will run when container starts)

→ Now build an image using the following command.

docker build -t myImg:1.0 path to Dockerfile

myImg → Image name

1.0 → version

(xvii) docker login

Command to login to dockerhub account from terminal