**DOCKER**

**Docker Environment:-**

Docker Engine (Most Important)

-Docker CLI (Interactive Interface) =>Do Request by user

-Docker API (Messanger b/w CLI & Daemon) =>Accept Request and transfer to the Daemon

-Docker Daemon(Actual Processor) =>Process the Request by API

Docker Object (Created & Managed by Docker Engine)

-Docker Image =>Blueprint/ Rules to create container

-Docker Container

-Docker Volume

-Docker Networks

-Docker Nodes/Services

Docker Registry

Docker Compose

Docker Swam

**Docker Common Commands:-**

Pull the ubuntu Image=>

#sudo docker pull ubuntu

Get all the docker images pulled on your system=>

#sudo docker images

Run the Docker Image=>

#sudo docker run -it(Interactive mode) -d(detached mode) ubuntu(Image\_name)

Get the process status to check weather the Container is created or image is running or not=>

#sudo docker ps

Give the desirable name to the container=>

#sudo docker run -it -d --name mycontainer(desired\_name) ubuntu

Give the exposible port to the container for host system=>

#sudo docker run -it -d --name myubuntu -p 80:80 ubuntu

Open the container in your hosts system in bash shell=>

#sudo docker exec(executable) -ti(interactive mode) mycontainer(container\_name) bash

Stop the running container in docker=>

#sudo docker stop 66789023(Container\_ID)

View all the container running or stopped=>

#sudo docker ps -a(all)

Stop all the containers at one time

#sudo docker container stop $(docker container ls -aq)

Start the stopped container=>

#sudo docker start mycontainer(container\_name)

For forcible stop container in docker=>

#sudo docker kill mycontainer(container\_name)

For restarting the container=>

#sudo docker restart 678902334(container\_ID)

Remove the container by stopping before removing=>

#sudo docker rm 678902334(conatiner\_ID)

Forcible removal of the container=>

#sudo docker rm -f 678902334

Remove all the container at one time=>

#sudo docker container rm $(docker container ls -aq)

Save the changes done on the container even after stopping it=>

#sudo docker commit 678902334 custom-image(image\_name)

See all the images on host system of the docker=>

#sudo docker images

Give the image name in specific format to push it on docker hub=>

#sudo docker tag custom\_image(image\_name) ayushson/ubuntu-ngix-image(name\_to\_push)

Command to login to docker hub=>

#sudo docker login

Push the image on the docker hub=>

#sudo push ayushson/ubuntu-nginx-image

Container logging can be done by=>

#sudo docker run -it mycontainer /bin/bash

Get the container logging in docker=>

#sudo docker logs 66789990(container\_ID)

**Docker Volumes:-**

The storage provided to the container is not persistance and is removed after the container removal but for safe & easy storeage like hard drives we use the docker volumes.

Single Docker Volume can be attached with multiple containers. Volume drivers enhance the abilities of the docker volumes.

**Docker Networks:-**

Establish the connection b/w 1 or more containers by creating the network. Easy to manage & isolate the container by using docker network.

**Docker Registry:-**

It is the storage location for various docker images and also versioning of images. It can be private & public as well.

Examples=> Docker Hub, JFrog Artifactory, ECR(Elastic Container Registry)

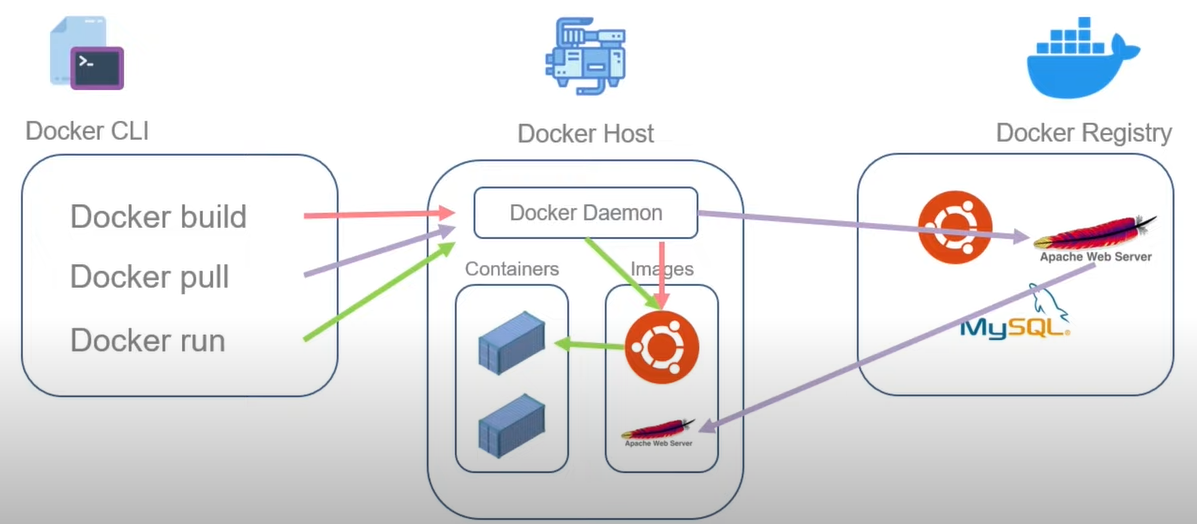
**Docker Compose:-**

Docker Compose helps to launch the multiple container at the same time by using the YAML file

**Docker Swarm:-**

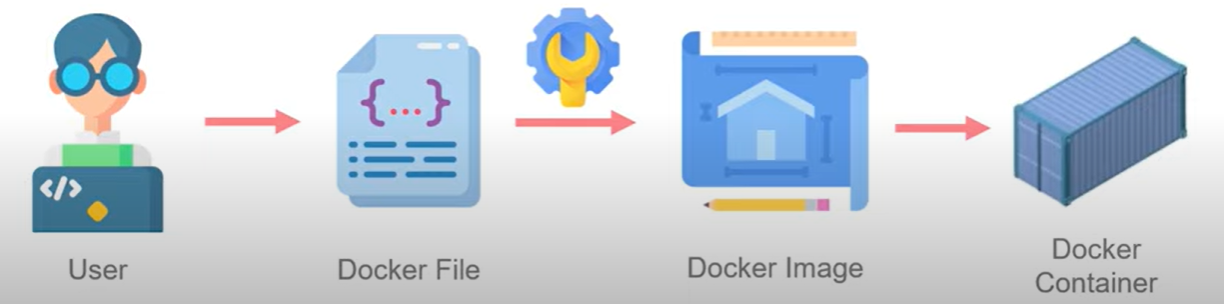
It is the orchestration service used in docker that manage multiple docker container by creating the master node & the worker nodes.

**Docker Archiecture**



**Docker File**

Docker Files are script to create a Docker Image that in result run in the container.



**DOCKER FILE - Format**

FROM

Syntax=> FROM <image\_name> (Inherit image)

ADD

Syntax=> ADD <source><destination>

COPY

Syntax=> COPY <source><destination>

Note:- Only difference between ADD & COPY is add can include the URL of file system but copy can’t.

RUN

Syntax=>RUN <command> (To run any command on the container)

WORKDIR

Syntax=>WORKDIR <directory> (specify the working directory of the container)

CMD

Syntax=>CMD <command> (Tell the container to run command at the time of container started)

VOLUME

Syntax=>VOLUME <path> (Mount point with given name)

EXPOSE

Syntax=>EXPOSE <port> (Publish the service on specific port in container)

ENTRYPOINT

Syntax=> ENTRYPOINT <command> <parameter1><parameter2>

NOTE:- ENTRYPOINT is same as CMD but only difference is you can give parameter to the command & this command overrides all other commands even CMD command too & ENTRYPOINT command cannot be override.

LABEL

Syntax=>LABEL <key>=<value> (Add the metadata to the instructions)

Note:- Every instruction creates a new label. So, it’s important to optimize the instructions as much as possible.

Example DockerFile:-

FROM ubuntu:latest

WORDIR /app

ADD . /app

RUN apt update && apt install python -y

CMD python /app/main.py

LABEL color=red

Command to run the dockerfile:-

#sudo docker build -t custom-python-image .(dockerfile present in same directory)

**Docker Images:-**

It is made of multiple layer, Each instruction form each layer & these layers execute sequentially.

Similarly Below Dockerfile has 6 layers.

FROM ubuntu:latest

WORDIR /app

ADD . /app

RUN apt update && apt install python -y

CMD python /app/main.py

LABEL color=red

Build the dockerfile with given docker\_image name=>

#sudo docker build -t <image\_name> .

NOTE:- Layers are immutable and so image but the container is mutable and any change can be done at container layer only.

Images(Read-Only)

Containers(Read-Write) It comprises of the reference of an image and a writable layer.

**Docker Layers:-** Docker layer is the command specified inside the docker image, the instruction FROM ubuntu itself container series of layers, Check-on for this:-

For finding how many layers contains in an image=>

#sudo docker image history <image\_name/image\_ID>

Image Child commands:-

* Within docker image inspect command, we can use format argument to get the value of that =>

#sudo docker image inspect ngnix –format={{.id}}

* Docker image prune Command is used for removing the dangling images, it refers to the images that doesnot associated with any tag and container=>

#sudo docker image prune -a

* Docker allows to achieve the image then load on any other system=>

#sudo docker save nginx-env > ngnix.tar

# docker load < ngnix.tar

* Docker allows to flatten the image by reducing its size & also lowering the no. of layers in the image=>

#docker export <container\_name> > container.tar

#cat container.tar | docker import <image\_name>

**Docker Storage**

Default Docker story is Containers Writable layer but it’s not an efficient approach as it results in data loss when we remove the container and lower the performance of the container.

So, we use the different types of docker storage to achieve -Persistent Data

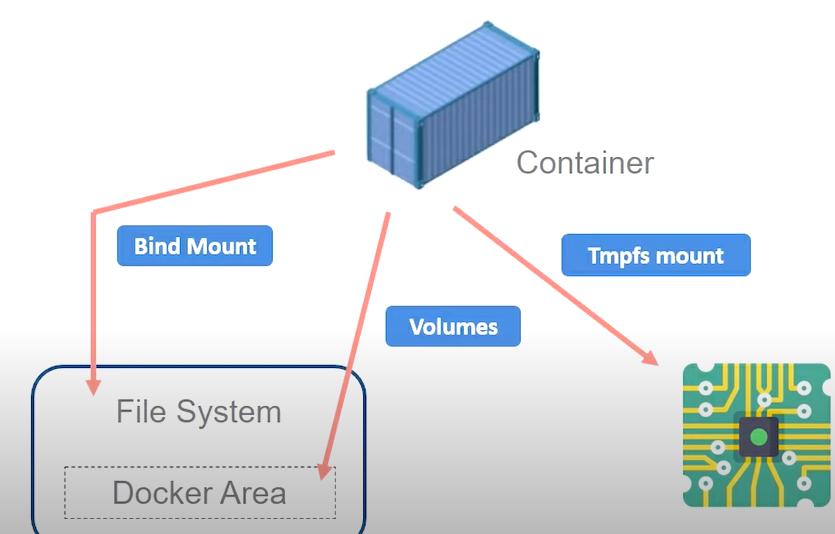
-Easily Data Transfer

-Increasing Container Performance

**Docker Storage Types:-**

1. Volume (Used with Docker application)
2. Bind Mount (Can be used standalone)
3. Tmpfs Mount (Temporary Data storage)

Where these Storage is resides & Accessible to ?



**Docker Volume:-**

-Managed by Docker and resides specifically on the docker file system.

-It can be attached and detached from one container to another container.

-Most used Docker storage types.

-It can be backup also

Command & Syntax:-

To create a Docker volume=>

#sudo docker create <volume\_name>

To get all the Docker volumes=>

#sudo docker volume ls

Get all the details of a volume=>

#sudo docker volume inspect <volume\_name>

How to remove the docker volume=>

#sudo docker volume rm <volume\_name>

Remove all the volume at once=>

#sudo docker volume prune

Attach a volume to a container and create if not already created

**Two ways:-**

1)Using Mount (Can be used for all 3 types of storage)

#sudo docker run -it -d –name conA –mount source=<volume\_name>,target=<mount\_location> ubuntu

2)Using Volume

#sudo docker run -it -d –name conB –volume <volume\_name>(source):<mount\_location>(target) ubuntu

Verify this by

#sudo docker container inspect conA/conB (look for the mount parameter in the inspect section)

Or

#sudo docker volume ls

**Binds Mount:-**

Not managed by docker and are mapped in some hosts directory.

Syntax:-

Create a Mount storage & attach it to any container=>

#sudo docker run -it -d –name ConC –mount type=bind,source=$(pwd),target=/apps ubuntu

Read only mount creation=>

#sudo docker run -it -d –name ConC –mount type=bind,source=$(pwd),target=/apps,readonly ubuntu

**Tmpfs Mount**(Non-persistence Data storage):-

Use & throw storage data type. Temporary Storage for making container performance efficient. It cannot be mount with multiple containers, one container mount (only for linux).

Syntax:-

Create a tmpfs storage and attach it to the container

#sudo docker run -it -d –name ConD –type=tmpfs,target=/app ubuntu

Verify by:-

#sudo docker container inspect ConD

**Storage Drivers:-**

Maintain control over the storage of Docker’s writable layer.

**Docker Networks**

Docker Networks are used to establish communication between two or more containers, along with this it results in the isolation of the container network from other container network.

Types of Docker Network:-

1. Bridge
2. Host
3. Overlay
4. Macvlan
5. None

**Note:-** Every Docker container is able to communicate with other container by default with bridge network, if want to isolate the container with rest assign it any network type or none network.

**Bridge Network:-**

Docker Container is connected to another container by custom bridge network and can achieve the isolation & abstraction from same.

Syntax:-

Create bridge Network=>

#sudo docker network create –driver bridge <name>

Get all the networks in docker=>

#sudo docker network ls

Get all the details of the specific Network=>

#sudo docker network inspect <name> (check for the <containers> parameter)

**Host Network:-**

In this network, container do not have any IP address they share the namespace of the host & used in very specific use cases.

Syntax:-

Create a host network & attach it to the container=>

#sudo docker run -it -d –network host –name <container\_name> ngnix

Verify it:-

#sudo docker container inspect <container\_name> (search for the networks parameter inside that IP\_Address i.e. Blank)

**Overlay Network:-**

This network is used when we have multiple containers on multiple servers so, we connect different docker host to each other by the help of docker swarm over overlay network.

Brief about Docker Swarm:-

To initialize the docker swarm=>

#sudo docker swarm init (The node become the master node & the next connected node is the worker node to this master).

**None Network:-**

It is used to create a container isolation that doesnot communicate with any other node/network/container. This type of container has no IPaddress.

Syntax:-

Create a none container=>

#sudo docker run -it -d –name noncon –network none ubuntu

Verify it=>

#sudo docker container inspect noncon (check for networks parameter no IPaddress & none type of network).

**Docker Compose**

It is just the service within docker to help multiple containers to launch at the same time.

Commands:-

Create a compose.yaml file configuring network,services & container details.

Run the docker compose file=>

#sudo docker-compose up (It start running the container with specified services & network)

For stopping all the services=>

#sudo docker-compose down (It stop the services & remove the containers, networks)

**Restart policies**

Flags:-

No=>Don’t restart the container automatically

0n-failure=>restart the container if it exit with non-zero error

Unless-stopped=>restart the container it get stopped & exited

Always=> restart every time when container stops.

Syntax:-

#docker run -it -d –restart unless-stopped ubuntu

Docker exec command is used to execute the new command in the container where the primary process is in running state.

Note:- If the primary process of the docker container stopped, then the container also stop working & this process not restarted after every restart.

Importance of -it flag:-

Every process in linux has three 3 parts=> stdin,stdout & stderr

For stdin=> -i flag(interactive) & For stdout=> -t flag(pseudo-TTY)

Overriding the cmd command specified in dockerfile=>

#sudo docker container run -d ngnix <sleep 20>(this is overriding & now the primary process will be this after it get stop the container also stops).

Default IP Address of the Docker Hosts:- 172.17. 0.0/16

What command is used for remove all stopped containers, unused networks, build caches, and dangling images:- #docker system prone

The memory space occupied by docker:-

#docker system df

How to automatically remove the container when it get exited.

#docker container run -it -dt -rm(automatically container removes) –name testcontainer ubuntu ping -c10 google.com

Copy the file with specific no. of count

#dd if=<inputfile\_directory>(/dev/zero) of=<file\_name> bs=1M count=500

To verify if file created & size of it=>

#du -sh <file\_name>

**Docker Swarm**

It is the docker-orchestration tool help in deploying containers on different nodes & managing it with manager node, all other nodes were worker nodes.

Swarm cluster:-

Initialise the docker swarm by manager node=>

#docker swarm init

Connect the node with this manager node=>

#docker swarm join –token <token\_ID>

Service creation=>

#docker service create <image\_name>

Get list of services=>

#docker services ls

Details of a specific service=>

#docker service ps <service\_name>

Docker with certain no. of replicas=>

#docker service create –name webserver –replicas 1 <image\_name>

Docker service can be scaled up & down=>

#docker service scale webserver=5 (This will create the 5 replicas/containers of same service)

OR

#docker service update –replicas 5 mywebsserver

Removing the service=>

#docker service remove <service\_name>

We can also scale multiple services by same command=>

#docker service scale service01=4 service02=3 (It cannot be done by service update command)

Two types of services:-

Replicated Services=> This create the identical services to different nodes.

Global Services=> This create a single service and run it on different nodes.

Syntax for creating a Global Services=>

#docker service create –name antivirus –mode global -dt nginx

Draining the nodes by docker swarm=>

#docker node update –availability drain <node\_name> (All the task on the respective node shutdown)

Get the status of Node=>

#docker node ls

Again changing the availability to Active=>

#docker node update –availability active <node\_name> (All the task again stated on the node)

Inspect the service=>

#docker service inspect <service\_name> --pretty

Inspect the node=>

#docker node inspect <node\_name> -pretty

We can specify the port for service=>

#docker service create –name myserver –replicas 2 –publish 8080:80 nginx

**Docker-Compose**

Docker-compose is use for starting multiple containers at same time, it can be achieve by docker-compose.yaml file that includes the configuration regarding each container to get started.

Default compose.yaml file or Docker-compose.yaml files are used for the execution:-

1. version: '3.3'
3. services:
4. db:
5. image: mysql:5.7
6. volumes:
7. - db\_data:/var/lib/mysql
8. restart: always
9. environment:
10. MYSQL\_ROOT\_PASSWORD: somewordpress
11. MYSQL\_DATABASE: wordpress
12. MYSQL\_USER: wordpress
13. MYSQL\_PASSWORD: wordpress
15. wordpress:
16. depends\_on:
17. - db
18. image: wordpress:latest
19. ports:
20. - "8000:80"

After writing the file we can activate all the container at once by=>

#docker-compose up

Check for the containers by=>

#docker ps -a

Remove all the containers by the single command=>

#docker-compose down