DOCKER

DOCKER:

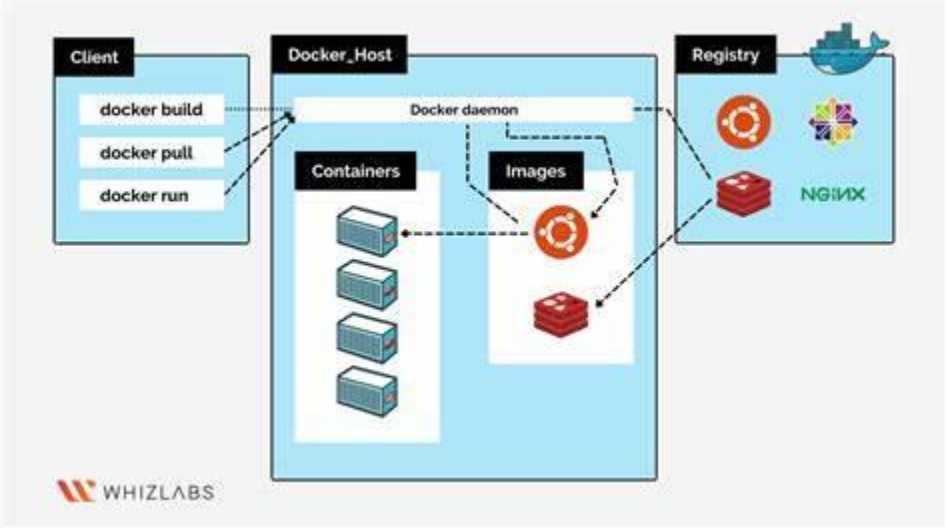
- It's free & opensource tool.
- It is platform independent.
- It is used to create, run & deploy applications on containers.
- Here we write files on YAML.

CONTAINERS:

- It is same as a server/VM-Instance.
- It will not have any operating system. OS will be on images.

(SERVER=AMI, CONTAINER=IMAGE)

Docker – Architecture:



Client: it will interact with User. User gives commands and it will be executed by docker client

Docker-Daemon: Manages the docker components (images, containers, volumes)

Docker-Host: where we install docker (eg: Linux, windows, macOS)

Registry: Manages the images (Image Library).

Basic Commands:

- Pull image from Registry: docker pull ubuntu
- To see list of images: docker images
- To see list of Containers: docker ps -a
- To see list of Container ID: docker ps -q
- To create a container: docker run it --name $CONTAINER_NAME $IMAGE

Example:

docker run -it --name cont1 ubuntu

[-it (interactive terminal) - to go inside a container]

To exit from container: docker p q

docker run -itd --name cont1 ubuntu

[-itd (interactive terminal Detach) - not go inside container]

To go inside container: docker attach CONTAINER_NAME

- To start container: docker start $CONTAINER_NAME
- To stop container: docker stop $CONTAINER_NAME
- To pause container: docker pause $CONTAINER_NAME
- To un-pause container: docker pause $CONTAINER_NAME
- To get complete info of a container: docker inspect $CONTAINER_NAME
- To Delete Container: docker rm $CONTAINER_NAME
- To Create Image from Container: docker commit $CONTAINER_NAME $IMAGE:TAG

Example: docker commit cont1 image:v1

DOCKERFILE:

- It is an Automation way to create Image.
- Here we use components to create images.
- In Dockerfile D must be Capital (vim Dockerfile).
- Here we can create images directly without container help.

COMPONENTS:

FROM	: used to pull base image
RUN	: used to run Linux commands (During image creation)
CMD	: used to run Linux commands (After container creation)
ENTRYPOINT	: high priority than commands
COPY	: to copy local files to container
ADD	: to copy internet files to container
WORKDIR	: to open required directory
LABEL	: to add labels for docker file
ENV	: to set env variables (inside container)
ARGS	: to pass env variables (outside containers)
EXPOSE	: to give port number

COMMAND:

To build Image from Dockerfile: docker build -t IMAGE:TAG .

Example: docker build -t image:v2 .

DOCKER – COMPOSE:

- It's a tool used to manage multiple containers
- We can create, start, stop, and delete all containers together.
- We write container information in a file called a compose file.
- The compose file is in YAML format.
- Inside the compose file we can give images, ports, and volumes info of containers.
- We need to download this tool and use it.

Installation: Run this in VM Instance Terminal

- curl -SL https://github.com/docker/compose/releases/download/v2.27.0/docker-compose-linux-x86_64 -o /usr/local/bin/docker-compose
- sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
- sudo chmod +x /usr/local/bin/docker-compose
- docker-compose version

Basic Docker-Compose file:

vim docker-compose.yml

```
version: '3.8'
services:
 movies:
  image: $IMAGE:TAG
  ports:
   - "81:8080"
 train:
  image: $IMAGE:TAG
  ports:
   - "82:80"
```

COMMANDS:

docker-compose up -d	: to create and start all containers
docker-compose stop	: to stop all containers
docker-compose start	: to start all containers
docker-compose kill	: to kill all containers
docker-compose rm	: to delete all containers
docker-compose down	: to stop and delete all containers
docker-compose pause	: to pause all containers
docker-compose unpause	: to un-pause all containers
docker-compose ps -a	: to list the containers managed by compose file
docker-compose images	: to list the images managed by compose file
docker-compose logs	: to show logs of docker compose
docker-compose top	: to show the process of compose containers
docker-compose restart	: to restart all the compose containers
docker-compose scale train=10	: to scale the service

NOTE: By default, the docker-compose will support the following names

docker-compose.yml, docker-compose.yaml, compose.yml, compose.yaml

CHANGING THE DEFULT FILE:

mv docker-compose.yml abc.yml (change compose.yml to abc.yml)

docker-compose up -d: throws an error

docker-compose -f abc.yml up –d (to create and start all containers)

Docker-Hub:

- Docker Hub is a repository service, and it is a cloud-based service
- We can push Docker Container Images and pull the Docker Container Images from the Docker Hub anytime or anywhere via the internet.
- It provides features such as you can push your images as private or public.

image (local) -- > dockerhub (internet) = to access by others

Pre-requisites:

Create a dockerhub account and create a repo in it.

COMMANDS:

Login to Docker Hub: docker login

-- username:

-- password:

Tag the Image before pushing into Docker hub:

docker tag $IMAGE_NAME:TAG $USER_NAME/$REPO_NAME

Eg: docker image:v1 saravana2002/movies

To Push the image to Docker Hub:

docker push $USER_NAME/$REPO_NAME

Eg: docker push saravana2002/movies

To Pull Image from Docker Hub

docker pull $USER_NAME/REPO_NAME:LATEST

Eg: docker pull saravana2002/movies:latest

Docker-Swarm:

- It is an orchestration tool for containers.
- Used to manage multiple containers on multiple servers.
- Here we create a cluster (group of servers).
- in that cluster we can create same container on multiple servers.
- Here we have master node and worker node.
- Master node will distribute the container to worker nodes.
- Worker node main purpose is to maintain the container.
- Without a docker engine we can't create the cluster.

DOCKER-SWARM-SETUP:

Step1 – Create 3 VM Instance and Install docker and start the service

Step2 – Set Hostname to master and worker nodes

- hostnamectl set-hostname manager (MASTER-VM-INSTANCE)
- hostnamectl set-hostname worker-1 (WORKER-VM-INSATACE-1)
- hostnamectl set-hostname worker-2 (WORKER-VM-INSATACE-2)

Step3 – Generate token to add worker node in Cluster

- docker swarm init **(run this in mater-vm-instance)**

-- > copy-paste the token to worker nodes

- docker node ls – To list nodes are added to cluster **(run this in mater-vm-instance)**

Step4 – Create a Service

SERVICE: it's a way of exposing and managing Multiple Containers.

Note: individual containers are not going to be replicated. If we create a service, then only containers will be distributed.

Docker-Service -Commands:

To Create a Service:

docker service create --name $SERVICE_NAME --replicas 3 -p 81:80 $IMAGE:TAG

Eg: docker service create --name paytm --replicas 3 –p 81:80 iamge:v1

docker service ls	: to list services
docker service inspect paytm	: to get complete info of paytm service
docker service ps paytm	: to list the containers of paytm
docker service scale paytm=10	: to scale in the containers
docker service scale paytm=3	: to scale out the containers
docker service rollback paytm	: to go previous state
docker service logs paytm	: to see the logs of paytm service
docker service rm paytm	: to delete paytm services.

Note: if we delete a container indocker swarm cluster, it will recreate automatically itself. It is called as self-healing.

CLUSTER ACTIVIES – COMMANDS:

docker swarm leave (run in worker node)	: to make node inactive from cluster
docker node rm node-id (run in master)	: to delete worker node which is on down state
docker swarm join-token manager (master)	: to generate the token to join
docker node inspect node_id (master)	: to get comple info of worker node

Note:

we can’t delete the node, which is ready state, we need to remove the node from cluster and then delete it.

if we want to join the node to cluster, again we need to paste the token on worker node

DOCKER-STACK:

- Docker stack is used to create multiple services on multiple hosts.
- i.e., it will create multiple containers on multiple servers with the help of compose-file.
- To use the docker stack we have initialized docker swarm, if we are not using docker swarm, docker stack will not work.
- Once we remove the stack automatically all the containers will get deleted.
- We can share the containers from master to worker according to the replicas

Eg: Let's assume if we have 2 servers which is, master and worker node, if we deployed a stack with 4 replicas. 2 are present in master and 2 are present in worker node.

- Here master node will divide the work based on the load on a server

DOCKER-STACK-COMMAND:

- TO DEPLOY: docker stack deploy --compose-file docker-compose.yml $STACK_NAME
- TO LIST: docker stack ls
- TO GET CONTAINERS OF A STACK: docker stack ps $STACK_NAME
- TO GET SERVICES OF A STACK: docker stack services $STACK_NAME
- TO DELETE A STACK: docker stack rm $STACK_NAME

DOCKER NETWORKING:

Docker networks are used to communicate between the multiple containers running on same or different docker hosts.

We have different types of docker networks:

Bridge Network

Host Network

None network

Overlay network

BRIDGE NETWORK: It is a default network that containers will communicate with each other within the same host.

HOST NETWORK: When you Want your container IP and ec2 instance IP same then you use host network

NONE NETWORK: When you don't Want The container to get exposed to the world, we use none network. It will not provide any network to our container.

OVERLAY NETWORK: Used to communicate containers with each other across the multiple docker hosts.

Docker Network – Commands:

- To create a network: docker network create $Network_Name
- To see the list: docker network ls
- To delete a network: docker network rm $Network_Name
- To inspect: docker network inspect $Network_Name
- To connect a container to the network: docker network connect $Network_Name container_id/name
- command to install ping checks: apt install iputils-ping -y
- To disconnect from the container: docker network disconnect $Network_Name container_name
- To prune: docker network prune $Network_Name