Docker Is a containerization software using which we can create containers.

Docker is for Devlepors, Admins(DevOps) to build and run applications as contianers.

Docker Editions:

Docker CE --> Comunity Edition --> Open Source(Free)

Docker EE --> Enterprise Edition --> Commercial

Type: Containerization

Vendor: Docker INC

O.S --> Cross Platform(Docker can be installed in any O.S)

Docker Can Be Installed in Linux, Windows OS ,mac OS

Docker CE Can be installed in Most of the linux except redhat.

Docker EE can installed in all O.S including redhat.

Docker CE --> OpenSouce Free

Docker EE --> Commerical

DTR --> Docker Trusted Registry(Private Repo to main docker images)

UCP --> Universal Controll Pane --> It's GUI for managing Docker Machines

Docker,CoreOS,Rockert --> Containerzation Platforms/Softwares.

Dockerfile --> Dockerfile is file which contains instructions to create an image. Which contains

Docker Domain Specific Key Words to build image.

DockerImage --> It's a package which contains everything(Softwares+ENV+Application Code) to run your application.

DockerContainer --> Run time instance of an image.If you run docker image container will be created

that's where our application(process) is running.

DockerRegistriy/Repository

DockerRepo/Registry. --> We can store and share the docker images.

Public Repo --> Docker hub is a public reposotiry. Which contains all the open source softwares as

a docker images. We can think of docker hub as play store for docker images.

Private Repo(Nexus,JFrog,D.T.R(Docker Trusted Registory)),AWS ECR --> We can store and share the docker images with in our company

network using private repo

Docker Enigine/Daemon/Host --> It's a software or program using which we can create images & contianers.

Docker is cross platform.

Docker CE

Docker CE will not be supported by Redhat.

Docker EE

Docker EE will be support most of the os including redhat.

First Create Account in docker hub

https://hub.docker.com

What is docker hub?

It's a public repository for docker images. You can think as play store for

docker images.

Install Docker on AWS Ubuntu

############################

sudo apt update -y

sudo apt install docker.io -y

sudo service docker start

sudo docker info

# Check docker is installed or not

docker info

# You will get permison denied error as regular user dosn't have permisions to execute docker commands.Add user to docker group.

sudo usermod -aG docker $USER

or

sudo usermod -aG docker ubuntu

# Exit From Current SSH Terminal & SSH(Login) again .Then execute

docker ps

# Amazon Linux

================

sudo yum update -y

sudo yum install docker -y

sudo service docker start

Add Regural user to dockergroup

sudo usermod -aG docker <username>

ex:

sudo usermod -aG docker ec2-user

Once you add user to group exit from the server and login again.

# Get docker information

docker info

#Install Docker in Linux (Works for most of linux flavors).

sudo curl -fsSL get.docker.com | /bin/bash

Docker Home/Working Dir:

/var/lib/docker

Install Docker on AWS RHEL (Offcially No Support)

############################

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo dnf install docker-ce-3:18.09.1-3.el7 -y

sudo systemctl enable docker

sudo systemctl start docker

sudo docker info

# Check docker is installed or not

docker info

# You will get permison denied error as regular user dosn't have permisions to execute docker commands.Add user to docker group.

sudo usermod -aG docker $USER

or

sudo usermod -aG docker ec2-user

# Exit From Current SSH Terminal & SSH(Login) again .Then execute

docker ps

How many containers we can run in on system/server?

It dependes on your system resources(CPU,RAM).

# List Images

docker images

# Sample DockerFile Content

FROM tomcat:8-jdk8-corretto

COPY target/maven-web-application\*.war /usr/local/tomcat/webapps/maven-web-application.war

# Build Image

Defautl Docker file Name: Dockefile

docker build -t <imageName> .

If you have docker file with custom name using -f <fileName> while building docker image.

docker build -f DockerfileMaven -t <imageName> .

Note: Image name should have repository details along with name and version.

Public Repo (Docker Hub)

docker build -t <registryName>/<RepoName>:<version> .

Note: If we don't mention version information. By defualt it will use 'latest' as version

ex:

docker build -t dockerhandson/maven-web-application .

Private Repo (Nexus/JFrog/DTR)

docker build -t <imageName> .

docker build -t <IP/HostNameOfRepo>:<RepoPort>/<repoName>:<version> .

ex:

docker build -t 178.90.34.12:8083/maven-web-application .

Authenticate with repo

# Public Repo

docker login -u <userName> -p <password>

ex:

docker login -u dockerhandson -p password

Priavate Repo

docker login -u <username> -p <password> <URL>

ex:

docker login -u admin -p admin123 178.90.34.12:8083

Push Docker Image to Repo

docker push <imageName>

Public Repo

docker push dockerhandson/maven-web-application

Private Repo

docker push 178.90.34.12:8083/maven-web-application

=============

# List Images

docker images

docker image ls

# Will return only ids.

docker images -q

# Sample DockerFile Content

FROM tomcat:8-jdk8-corretto

COPY target/maven-web-application\*.war /usr/local/tomcat/webapps/maven-web-application.war

# Build Image

Defautl Docker file Name: Dockefile

docker build -t <imageName> .

If you have docker file with custom name using -f <fileName> while building docker image.

docker build -f DockerfileMaven -t <imageName> .

Note: Image name should have repository details along with name and version.

Public Repo (Docker Hub)

docker build -t <registryName>/<RepoName>:<version> .

Note: If we don't mention version information. By defualt it will use 'latest' as version

ex:

docker build -t dockerhandson/maven-web-application .

Private Repo (Nexus/JFrog/DTR)

docker build -t <imageName> .

docker build -t <IP/HostNameOfRepo>:<RepoPort>/<RepoName>:<version> .

ex:

docker build -t 178.90.34.12:8083/maven-web-application .

Authenticate with repo

# Public Repo

docker login -u <userName> -p <password>

ex:

docker login -u dockerhandson -p password

Priavate Repo

docker login -u <username> -p <password> <URL>

ex:

docker login -u admin -p admin123 178.90.34.12:8083

Push Docker Image to Repo

docker push <imageName>

Public Repo

docker push dockerhandson/maven-web-application

Private Repo

docker push 178.90.34.12:8083/maven-web-application

# Downlod Image from repo

docker pull <imageName>

Public Repo

docker pull dockerhandson/maven-web-application

Private Repo

docker pull 178.90.34.12:8083/maven-web-application

Inspect Docker Image

==================

docker image inspect <imageId/Name>

docker inspect <imageId/Name>

How to list only layers of an image?

docker history <imageId/Name>

Delete Image

docker rmi <imageId/Name>

docker rmi -f <imageId/Name>

Note: We cann't remove images if there are running container for the image.We cann't force delete images if there is running container.

If container is in stopped(exited) state we can force delete image for the stopped container.

what is dangling images in docker?

The image which doesn't have repository mapping or tag.

How to delete all the images?

docker rmi -f imageId imageId imageId

docker rmi -f $(docker images -q)

docker system prune

Will delete all stopped containers , unused docker networks and dangling images.

docker image prune

Will delete angling images.

We can tag image with repo.

# We can use docker tag to tag images with multiple repo.

docker tag <ImageId/ExistingImageName> <ImageName>

What is working directory of docker?

/var/lib/docker

How can we move/copy images from one server to another server with out repo?

In Source Server(where you have image)

# It save image(All the layers) as a tar file

docker save -o <fileName>.tar <imageName/Id>

Then SCP tar file from Source Server to Destination Server

# In destination server

docker load -i <fileName>.tar

List Dangling images

docker images -f dangling=true

Remove Dangling Images

docker rmi $(docker images -f dangling=true -q)

docker system prune

This will remove:

- all stopped containers

- all networks not used by at least one container

- all dangling images

docker image prune

This will remove:

- all dangling images

docker contianer prune

This will remove:

- all stopped containers

docker network prune

This will remove:

- all networks not used by at least one container

Create ECR in AWS.

ECR

===

docker build -t 935840844891.dkr.ecr.ap-south-1.amazonaws.com/maven-web-app

# Authentication with ECR

aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 935840844891.dkr.ecr.ap-south-1.amazonaws.com

Note: Create IAM Role with required policy and attach to EC2 Servers.

# IAM Policiy to autheticate and pull & Push image.

AmazonEC2ContainerRegistryFullAccess

# IAM Policiy to autheticate and pull image.

AmazonEC2ContainerRegistryReadOnly

Container Commands:

===================

How to create a contianer?

docker run or docker create

docker create --name <containerName> -p <hostPort>:<containerPort> <imageName>

docker run --name <containerName> -p <hostPort>:<containerPort> <imageName>

# Create a container in dettached mode

docker run -d --name <containerName> -p <hostPort>:<containerPort> <imageName>

what is the difference b/w docker run and docker create?

docker create will only create a container but it will not start the container.

docker run will create a container and start the container.

what is port publish or port mapping in docker ?

If We have to access application which is running as container from out side of docker we can't access using continerIP & ContainerPort. We can publish contianer port using host port using -p or --publish.

So that we can access using HostIP(docker server IP) and Host Port from outside docker.

docker run -d -p 8080:8080 --name mavenwebapp dockerhandson/maven-web-application

Access Application which is running Using Docker Server IP & Host Port.

http://<DOCKERSERVERPUBLICIP>:<HOSTPORT>/maven-web-application

# How to create container in interactive mode?

docker run -it --name <nameofthecontainer> <image>

List Running Containers

=======================

docker ps

docker container ls

List All Containers

==================

docker ps -a

docker container ls -a

List only running container ids

==============================

docker ps -q

docker container ls -q

List all container ids

==============================

docker ps -aq

docker container ls -aq

Start the container

===================

docker start <containerId/Name>

Restart Container

docker restart <containerId/Name>

Stop Container

docker stop <containerId/Name>

Kill container

docker kill <containerId/Name>

What is the difference b/w docker stop & docker kill?

docker stop will first send SIGTERM then SIGKILL it will kill process with grace period. Docker kill send SIGKILL it will kill process with out any grace period.

Can we have/run more than one process in a container?

Yes Can we have. But it's not suggestable.

Pause contaier process.

docker pause <containerId/Name>

docker unpuase <containerId/Name>

Inspect container

docker inspect <containerId/Name>

docker container inspect <containerId/Name>

It will container if it is stopped.

docker rm <containerId/Name>

Force Remove If container is runing we can force remove

docker rm -f <containerId/Name>

How to delete only stopped containers

docker rm $(docker ps -aq --filter status="exited")

How to delete all containers

docker rm -f $(docker ps -aq)

How to trouble shoot or debug application which is running as a container?

docker logs <containerId/Name>

docker logs --tail <NoOflines> <containerId/Name>

# It will display process details which is runing inside a container.

docker top <containerId/Name>

# It will display resource(RAM,CPU) consumtion details.

docker stats <containerId/Name>

# Execute commands on a runinging container.

docker exec <containerId/Name> <cmd>

ex:

docker exec javawebapp ls

docker exec javawebapp pwd

How to go inside a container?

docker exec -it <containerId/Name> /bin/bash

or

docker exec -it <containerId/Name> /bin/sh

# Docker attach will attach container process or shell to host server

docker attach <containerId/Name>

If we have to come out with out stoping the process cntl p+q.

How to copy files from container to host system or host system to container?

docker cp

Container to the system

docker cp <containerName>:</pathOftheContainerFile> <SystemPath>/<fileName>

docker cp javawebappone:/usr/local/tomcat/logs/catalina.2020-04-23.log javawebappone.log

system to the Container

docker cp <SystemPath>/<fileName><containerName>:</pathOftheContainerFile>

docker cp /home/ubunut/test.log javawebappone:/usr/local/tomcat/logs/test.log

docker rename <ContainerId/NameOld> <NewName>

What is docker commit?

Using docker commit we can create image from the continer.

docker commit <containerId/Name> <imageName>

Can we set CPU,RAM limit for the containers while creating?

Yes We set using options while creating a container.

# Search images in repos

docker search <imageName>

docker search tomcat

Image is package (AppCode+ Softwares)

Cotainer running process of an image.

If you have to crate docker image we need a dockerfile.

Dockerfile --> Dockerfile is file which contians instructions(Docker Domain Specific KeyWords) to create an image.

Docker Daemon will process these instruction from top to bottom.

EX:

FROM tomcat:8.0.20-jre8

COPY target/java-web-app\*.war /usr/local/tomcat/webapps/java-web-app.war

DockerImage --> It's package which contains application code + all it's dependencies(Software+ENV Varibles + Config Files) together.

Dockerfile keywords

===================

FROM

MAINTAINER

COPY

ADD

RUN

CMD

ENTRYPOINT

WORKDIR

ENV

EXPOSE

USER

VOLUME

LABEL

ARG

FROM --> FROM indicates the image base image which we are using to build our own image.

Syntax:

FROM <ImageName>

Ex:

FROM tomcat:8.0.20-jre8(Software)

FROM openjdk:8-alpine

MAINTAINER --> It's will be used as commnets to describe author/owner who is maintaning docker file.

MAINTAINER MithunTech <devopstrainingblr@gmail.com>

COPY --> Using COPY we can copy files/folders to the image. Files/Folders will be copied while creating an image.

It will copy local files from host server(docker server)from where we are building image to the image while creating a image.

SYTNAX:

======

COPY <source> <destination>

ServerFile/FolderPath PathInsideImage

EX:

COPY target/java-web-app.war /usr/local/tomcat/webapps/java-web-app.war

# Below also valid it will copy all the files/folder from HOST Machine current working

directory to Image working dirctroy.

COPY . .

ADD --> ADD also can copy files to the image while creating image. ADD can copy local files from host server and also it can download files from remote HTTP/S locations while creating a image.

ADD <URL> <destination>

ADD <source> <destination>

EX:

# File from http(s) location

ADD https://downloads.apache.org/tomcat/tomcat-8/v8.5.54/bin/apache-tomcat-8.5.54.zip /opt/

# Local file

ADD target/java-web-app.war /usr/local/tomcat/webapps/java-web-app.war

Note: If it's tar file ADD will copy file and also it will extract tar file.

RUN --> RUN instruncation will execute commands .RUN commands or instructions will be executed while creating an image. Next to run you can mention any command based on base os of image.

We can have n number RUN instructions in a docker file all the RUN instructions will be exectued one after the other from top to bottom.

Syntax:

#Shell Form

RUN <commond with args>

#Executable Form

RUN ["commond" , "Arg1","Arg2"]

EX:

RUN mkdir -p /opt/app

RUN tar -xvzf /opt/apache-tomcat-8.5.54.tar.gz

CMD --> CMD instruncation will execute commands. CMD commands or instructions will be executed while creating a container.CMD insturction can be used to start the process inside the container.

#Shell Form

CMD <commond with args>

#Executable Form

CMD ["commond" , "Arg1","Arg2"]

# Shell Form

CMD java -jar springapplication.jar.

# Executable form

CMD ["java", "-jar" , "springapplication.jar"]

What is difference b/w RUN & CMD?

RUN instructions will be executed while creating a image. CMD Instructions will be executed while creating a

container.We can have more than one RUN keyword in a docker file. All the RUN keywords will be processed while creating an image in the defined order(top to bottom).

Can we have more than one CMD in dockerfile?

Yes you can have. But only the last one/recent one in the order will be proccessed while creating a container.

ENTRYPOINT --> ENTRYPOINT instruncation will execute commands. ENTRYPOINT commands or instructions will be executed while creating a container.

ENTRYPOINT java -jar springapplication.jar. # Shell Form

ENTRYPOINT ["java", "-jar" , "springapplication.jar"] # Executable form

What is the diffrence b/w CMD & ENTRYPOINT ?

CMD command/instruction can be overridden while creating a container. ENTRYPOINT command/instruction can't be

overridden while creating a container.

Can we have both CMD & ENTRYPOINT in docker file?

Yes we can have both in a docker file. CMD instructions will not be executed if we have both CMD & ENTRYPOINT.CMD instructions will be passed as an arguments for ENTRYPOINT.

FOR Example:

CMD ls

ENTRYPOINT ["echo", "Hello"]

IT Will be executed as below

/bin/echo HELLO ls

# Out Put

Hello ls

Requirement always we have to execute sh catalina.sh . But argument by default it has to execute "start". But dynamically i should a option to pass different argument while creating a container.

CMD start

ENTRYPOINT ["sh", "catalina.sh"]

Docker Networks

What is network ?

Group of servers will be connected to each other in a specific network. If Servers

are in same network each one can talk to another server.

Docker network

If One Container has to talk to another Container in Docker. Both has to created under

same docker network.

If Containers are in two different networks. They can't accees each other.

In which docker network the container will be created if we don't mention network name while

creating a container ?

Containers will be created in a default bridge network.

If we don't mention network name while creating a container.

How to list networks in docker?

docker network ls

Docker will have 3 networks by default.

bridge(default)

host

none/null

docker run -d --name javawebapp -p 8080:8080 dockerhandson/java-web-app

docker run -d --name mavenwebapp dockerhandson/maven-web-app

If containers are created in a default bridge network. Communcation will happen only

with IP Address of container. Communcation will not happen using containerName(hostName).

To Check Go inside javawebapp container and ping mavenwebapp container using name & ip. When we ping using ip it will work it will not able to communicate using name.

Developers should not code the connectivity based on the IP in case of contianers. Since IP address of cotnainers will be dynamic.

IP will keep changing.

How to create a custom bridge network ?

# Create Network

Syntax: docker network create -d <driver> <networkName>

Ex:

docker network create -d bridge flipkartnetwork

# Inspect network

docker network inspect <networkNameOrId>

If containers are created in custom bridge network. Each container can access other using containerName/ContainerIP.

# Delete Containers which are running in default bridge or create container with different name.

docker run -d --name javawebapp -p 8080:8080 --network flipkartnetwork dockerhandson/java-web-app

docker run -d --name mavenwebapp --network flipkartnetwork dockerhandson/maven-web-app

Create both containers in same network and try to ping mongo contaner with name & IP from sprinapp container or vice versa it will work with both.

Another Application

docker run -d --name mongo -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234 --network flipkartnetwork mongo

docker run -d -p 8080:8080 --name springapp -e MONGO\_DB\_HOSTNAME=mongo -e MONGO\_DB\_USERNAME=devdb -e MONGO\_DB\_PASSWORD=devdb1234 --network flipkartnetwork dockerhandson/spring-boot-mongo

# Remove unused networks

docker network prune

# Remove Network

docker network rm <networkNameOrId>

Docker Host Network.

If we create contaienrs in host network. Container will not have IP Address. Container will be created

in a system network.

But we can't create more than one cotnainer with same container port in host network.We no need to do port publish to access

containers.

Docker none/null network

If we create contaienrs in none/null network. Container will not have IP Address.We can't

accees these contianers from out side or from any other cotnainer.

We connect container to more than one network by using docker connect.

docker network connect <networkName/Id> <containerName/Id>

docker network disconnect <networkName/Id> <containerName/Id>

Docker Volumes

Volumes:

=======

1) Create docker network using below commond(If it's not created already)

docker network create -d bridge springappnetwork

2) Create a mongo contianer with out volume in above network

docker run -d --name mongo -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234 --network springappnetwork mongo

3) Create Spring Application Container in above network & which will talk to mongo data base container

docker run -d -p 8080:8080 --name springapp -e MONGO\_DB\_HOSTNAME=mongo -e MONGO\_DB\_USERNAME=devdb -e MONGO\_DB\_PASSWORD=devdb1234 --network springappnetwork dockerhandson/spring-boot-mongo

4) Access Spring application & insert data it will be inserted to mongo db. Delete and recreate mongo container

what ever you have inserted will no longer be availbale. As once we delete contaienr data also will be deleted

in container.

To take data backup from container we have to use volunmes

Bind Mounts:

Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

# Volumes Using bind mount

docker run -d --name mongo -v ~/mongobkp:/data/db -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234 --network springappnetwork mongo

Docker Volumes

Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux).

Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.

5) Create a volume a Local Volume(Execute docker volume ls to check existing volumes)

docker volume create dbbackup

# To list volumes

docker volume ls

6) Use above volume while creating container.

docker run -d --name mongo -v dbbackup:/data/db -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234 --network springappnetwork mongo

7) Access Spring application & insert data it will be inserted to mongo db. Delete and recreate mongo container

with same volume mapping. You can see the data back.

Volume Driver Plugin --> It's a piece of code or software which is responsible for creating a storage and attaching the storage to the container.

===== Network Volumes Using AWS EBS==========

1) Create IAM User with EC2 Full Access and user access key & Secret Key of the same. Replace your access key & secret below. Or Use Your root aws account access Key & Secret Key.

docker plugin install rexray/ebs EBS\_ACCESSKEY=<ACCESSKEY> EBS\_SECRETKEY=<SECRETKEY>

EX:

docker plugin install rexray/ebs EBS\_ACCESSKEY=AKIAJRVS26WY3UKXG57Q EBS\_SECRETKEY=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

docker volume create --driver rexray/ebs --name ebsvolume

docker run -d -p 27017:27017 -v ebsvolume:/data/db --name mongo -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234 --network springappnetwork mongo

Map Volumes As Read Only using below option>

-v <volumeName/BindMount>:<containerPath>:ro

Docker Compose

==============

Docker Compose is a tool for defining and running multicontainer applications.

With out compose to deploy above applications which has only 2 images we executed below commnads.

docker network create -d bridge springappnetwork

docker volume create -d local mongobkp

docker run -d --name mongo -v mongobkp:/data/db -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234 --network springappnetwork mongo

docker run -d -p 8080:8080 --name springapp -e MONGO\_DB\_HOSTNAME=mongo -e MONGO\_DB\_USERNAME=devdb -e MONGO\_DB\_PASSWORD=devdb1234 --network springappnetwork dockerhandson/spring-boot-mongo

With Docker Compose we deploy/create all of the above 4 with single command using compose file.

With Compose

Install docker compose using below command:

sudo apt install docker-compose

We will define all the serivces(cotainers) details in compose file using compose file we can deploy multi container applications.

Defautl name : docker-compose.yml or docker-compose.yaml

Example 1: (Volumes & Networks also will be created by docker compose)

version: '3.1'

services:

springboot:

image: dockerhandson/spring-boot-mongo:latest

restart: always # This will be ignored if we deploy in docker swarm

container\_name: springboot

environment:

- MONGO\_DB\_HOSTNAME=mongo

- MONGO\_DB\_USERNAME=devdb

- MONGO\_DB\_PASSWORD=devdb1234

ports:

- 8080:8080

working\_dir: /opt/app

depends\_on:

- mongo

deploy: # This will be considered only in docker swarm.

replicas: 2

update\_config:

parallelism: 1

delay: 20s

restart\_policy:

condition: on-failure

delay: 10s

max\_attempts: 5

networks:

- springappnetwork

mongo:

image: mongo

container\_name: springboot-mongo

environment:

- MONGO\_INITDB\_ROOT\_USERNAME=devdb

- MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234

volumes:

- mongobkp:/data/db

restart: always

networks:

- springappnetwork

volumes:

mongobkp:

driver: local

networks:

springappnetwork:

driver: bridge

Commands

# Syntax Check

docker-compose config

# Create Services/Contianers

docker-compose up -d

# Remove Services/Contianers

docker-compose down

Example 2: (Volumes & Networks will not be created by docker compose.As we set volumes and networks as external)

==========

version: '3.1'

services:

springboot:

image: dockerhandson/spring-boot-mongo:latest

restart: always # This will be ignored if we deploy in docker swarm

container\_name: springboot

environment:

- MONGO\_DB\_HOSTNAME=mongo

- MONGO\_DB\_USERNAME=devdb

- MONGO\_DB\_PASSWORD=devdb1234

ports:

- 8080:8080

working\_dir: /opt/app

depends\_on:

- mongo

deploy: # This will be considered only in docker swarm.

replicas: 2

update\_config:

parallelism: 1

delay: 20s

restart\_policy:

condition: on-failure

delay: 10s

max\_attempts: 5

networks:

- flipkartnetwork

mongo:

image: mongo

container\_name: springboot-mongo

environment:

- MONGO\_INITDB\_ROOT\_USERNAME=devdb

- MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234

volumes:

- mongodb:/data/db

restart: always

networks:

- flipkartnetwork

volumes:

mongodb:

external: true

networks:

flipkartnetwork:

external: true

If docker compose file with custom name.

docker-compose -f <CustomeComposeFileName>.yml <command>

Ex:

docker-compose -f docker-compose-springapp.yml config

docker-compose -f docker-compose-springapp.yml up -d

docker-compose -f docker-compose-springapp.yml down

Docker Compose Commands:

config Validate and view the Compose file

create Create services

down Stop and remove containers, networks, images, and volumes

exec Execute a command in a running container

help Get help on a command

images List images

kill Kill containers

logs View output from containers

pause Pause services

port Print the public port for a port binding

ps List containers

pull Pull service images

push Push service images

restart Restart services

rm Remove stopped containers

run Run a one-off command

scale Set number of containers for a service

start Start services

stop Stop services

top Display the running processes

unpause Unpause services

up Create and start containers

version Show the Docker-Compose version information

# In Normal(Standalone) Docker Server We can use below command to create a containers.

docker-compose up

# In docker swarm we will use below command to deploy services using docker compose.

docker stack deploy --compose-file docker-compose.yml <stackName>

Containerization Tools: docker,rocker(rkt),coreos

Container Orchestration Tools: docker swarm,kubernetes,openshift ..etc

H.A --> HighAvailability

F.T --> Fault Tolarence

Scalability

L.B

# In Normal(Standalone) Docker Server We can use below command to create a container.

docker run -d -p <hostPort>:<containerPort> --name <containerName> <imageName>

# In docker swarm we will use below command to deploy service.

docker service create --name <serviceName> -p <hostPort>:<containerPort> <imageName>

# In Normal(Standalone) Docker Server We can use below command to create a containers from compose.

docker-compose up

# In docker swarm we will use below command to deploy services using docker compose.

docker stack deploy --compose-file docker-compose.yml <stackName>

Note: Make Sure You Open Required/All Ports in AWS Security Groups.

======================================================================

# Initialize docker swarm cluster by exeuting below command on docker server which you want make it as Manager

# Initialize docker swarm on privateIP

docker swarm init

# Initialyze Docker swarm with Public IP

Note: Don't use below(If restart your systems public ip will change will break your cluster) use above commond to initilaze cluster.

# Intilaize docker swarm cluster on public IP

docker swarm init --listen-addr=eth0 --advertise-addr $(curl http://169.254.169.254/latest/meta-data/public-ipv4)

# Get worker token

docker swarm join-token worker (Get the token in manager & exeute in nodes)

Docker Swarm has two modes

Global --> All the nodes (3 servers 1 Manager + 2 Workers)

Replicas --> It will deploye based on replicated number.

What is serivce in docker or docker swarm?

Serivce is nothing but a collection of one or more replicas(contianers) of same type(Image).

What is stack in docker or docker swarm?

Stack is nothing but a collection of one or more serivces of some application.

# Default Mode Replica and it will create a service with 1 replica

docker service create -p 8080:8080 --name javawebapp dockerhandson/java-web-app

docker service ls

docker service ps <servcieNAme>

ex:

docker service ps javawebapp

docker service scale <serviceName>=<noOfReplicas>

docker service scale javawebapp=3

docker service scale javawebapp=2

# While creating service we can mention number of replicas as below

docker service create -p 9090:8080 --name mavenwebapp --replicas 2 dockerhandson/maven-web-app

# User constriants to create containers in specific docker hosts based on condtion

docker service create -p 8080:8080 --name javawebapp --replicas 2 --constraint 'node.role==worker' dockerhandson/java-web-app

# Add labels to node

docker node update --label-add key=value <nodeid>

Ex: docker node update --label-add type=appServer qmdh9tgvdef99sryhbezswfl9

#Use above label in constrainsts

docker service create -p 8080:8080 --name javawebapp --replicas 1 --constraint 'node.labels.type==appServer' dockerhandson/java-web-app

# Drain Nodes in Cluster(Swarm will not create containers in drained nodes)

docker node update --availability drain <NodeID>

# Make Node Active in Cluster

docker node update --availability active <NodeID>

# Create a service with a rolling update policy

docker service create --replicas 2 --name javawebapp --update-delay 30s --update-parallelism 1 dockerhandson/java-web-app:1

# Update service image without down time.

docker service update --image dockerhandson/java-web-app:2 javawebapp

docker stack deploy --compose-file docker-compose.yml springmongo

docker stack ls

docker stack rm <stackName>

version: '3.1'

services:

springboot:

image: dockerhandson/spring-boot-mongo:latest

restart: always # This will be ignored if we deploy in docker swarm

container\_name: springboot

environment:

- MONGO\_DB\_HOSTNAME=mongo

- MONGO\_DB\_USERNAME=devdb

- MONGO\_DB\_PASSWORD=devdb1234

ports:

- 8080:8080

working\_dir: /opt/app

depends\_on:

- mongo

deploy: # This will be considered only in docker swarm.

replicas: 2

update\_config:

parallelism: 1

delay: 20s

restart\_policy:

condition: on-failure

delay: 10s

max\_attempts: 5

networks:

- flipkartnetwork

mongo:

image: mongo

container\_name: springboot-mongo

environment:

- MONGO\_INITDB\_ROOT\_USERNAME=devdb

- MONGO\_INITDB\_ROOT\_PASSWORD=devdb1234

volumes:

- mongodb:/data/db

restart: always

networks:

- flipkartnetwork

volumes:

mongodb:

external: true

networks:

flipkartnetwork:

external: true

# To come out of swarm execute below commond in worker node

docker swarm leave

# Remove node from Manager

docker node rm <nodename>

# Use private repo's.

#Send registry authentication details to swarm agents using --with-registry-auth

If you are using insecure(http) registry. Make sure you have flow below step in all servers (jenkins,docker swarm master,workers). Below Steps from 1 to 6 are not requried in real time since we will have secure(https) repositories like ECR or nexus with https.

1. Login as root user

2. Go to /etc/docker

cd /etc/docker

3. Then create a file called daemon.json

vi /etc/docker/daemon.json

4. Write these script in daemon.json

{

"insecure-registries": [ "<IPOfPrivateRepo>:<dockerRepoPort>" ]

}

ex:

{

"insecure-registries": [ "172.31.45.81:8083" ]

}

Note: Replace with your nexus ip instead of 172.31.45.81. Make Sure You Opened 8083 port in nexus server security

group.

(Here we are allowing our docker daemon to access the Nexus Hosted Repo)

5. Save the file

6. Restart docker

sudo systemctl restart docker

Before Pushing Image From Jenins execute docker login

docker login -u <username> -p <password> <URL>

Ex:

docker login -u admin -p admin123 172.31.45.81:8083

Execute docker login only in master

docker login -u <username> -p <password> <URL>

docker login -u admin -p admin123 172.31.45.81:8083

# While Creating a Service Or Stack send registry authentication details to swarm agents using --with-registry-auth

docker service create -p <hostPort>:<containerPort> --name <serviceName> --replicas 1 --with-registry-auth <imageName>

docker stack deploy --compose-file docker-compose.yml --with-registry-auth <stackName>