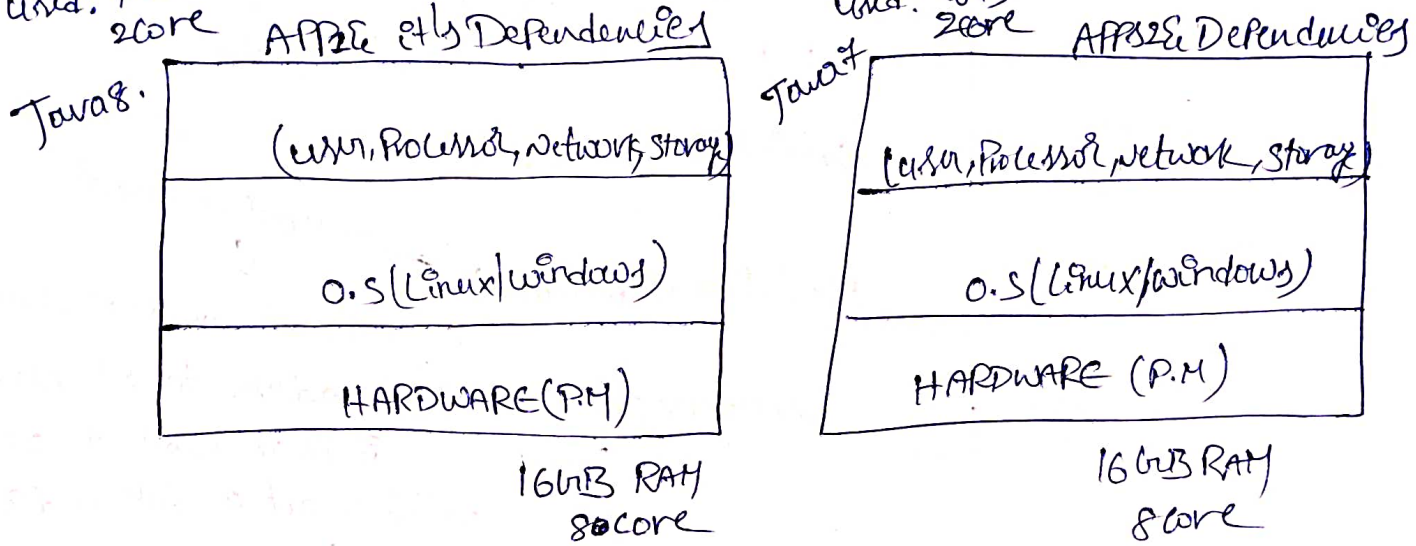
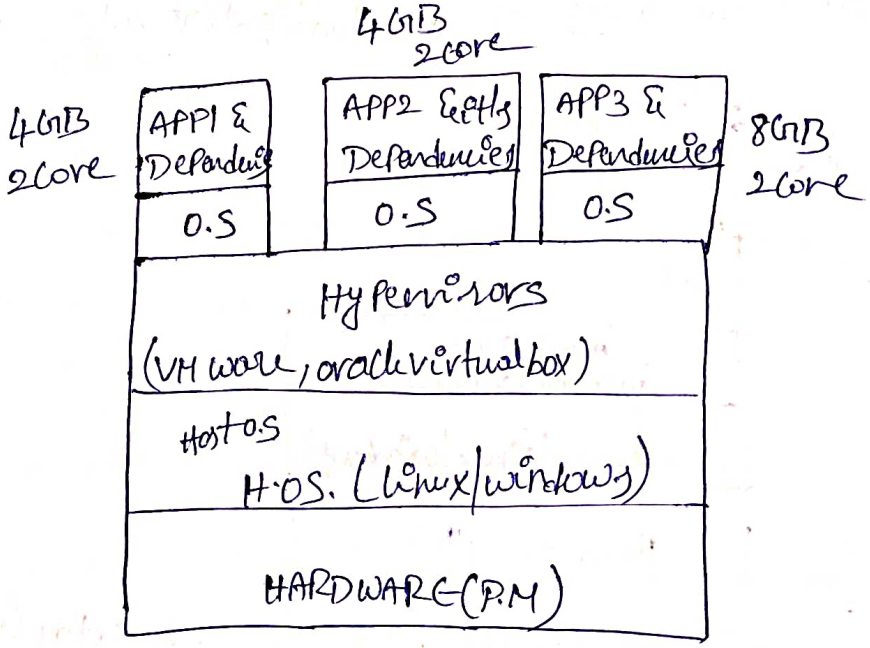
Docker

Previous way Deploy application:-

↳ there is no isolation here for deploying application, if you want deploy then Again you need to setup one more m/c.

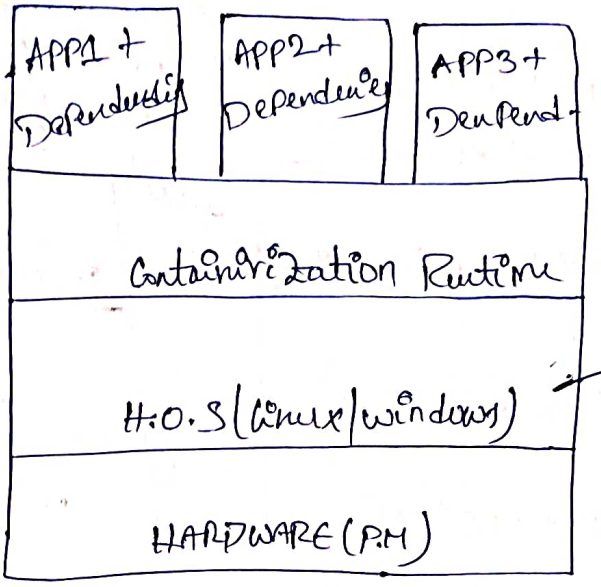


Here there is no isolation & wasting our resources also. so later peoples using vertualization process instead of this.



Here there is isolation based on resources of physical machine. we can deploy application application Easily, if something went wrong with one apps there is no effect on another application. but this is the time Consuming Process for installation. The Drawback with vertualization is

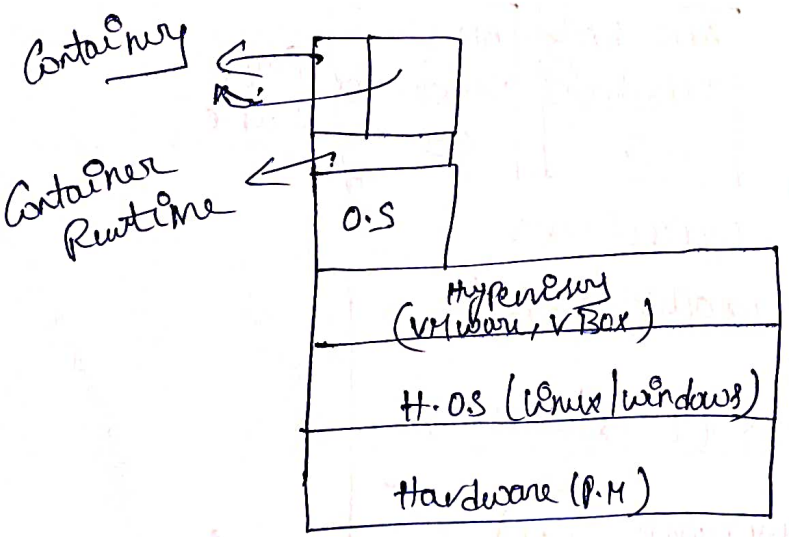
if you miss any configuration or software in any one environment the application may not work. That's why Containers are coming to the picture.



→ Host OS kernal
↓
It will converts human readable to M/c readable
It is an heart of OS which is act as intermediate b/w hardware & software.

if something went wrong with any application, we don't effect on another application.

This container can Run in virtualization m/c also.



Container:- It is light weight, standlone, executable Package of software that includes Everything needed to ~~make~~ run an application (code, dependencies, system tools).

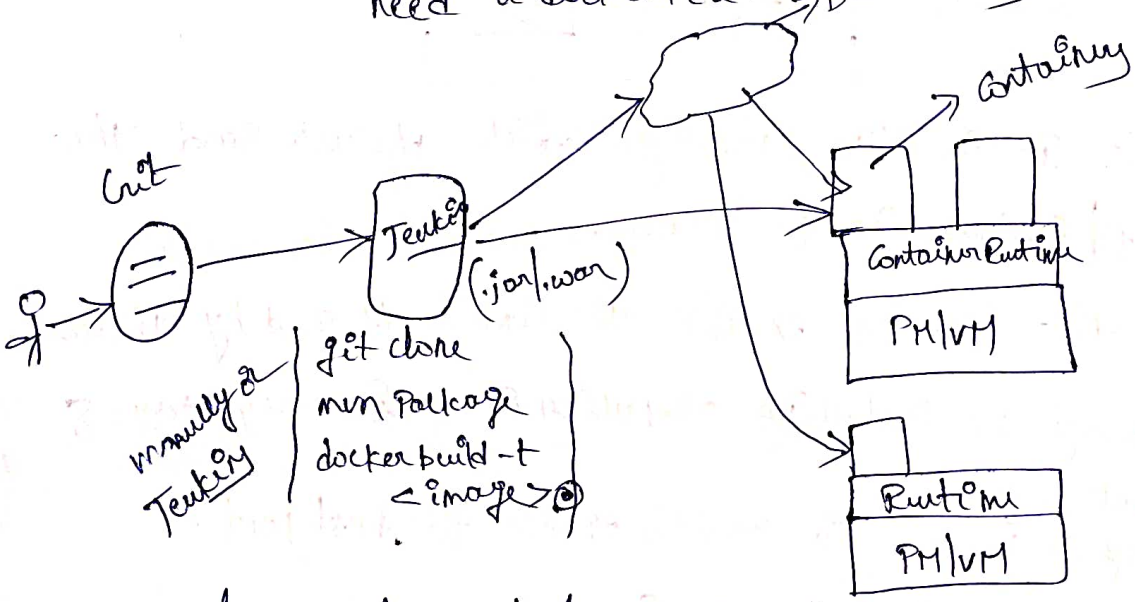
Docker Container:- Run time instance/Process of docker image.

Docker:- It is an opensource platform. Docker is a container managment platform for developing, deploying & running application. Docker is light weight in nature as it doesn't require hypervisor.

Docker image:- It's Package which contains application code & dependencies. It is portable I can move from one server to another with the help of registry store.

Dockerfile:- Dockerfile is a file which contains instructions to create a image.

If you want containirization your application you need a dockerfile.

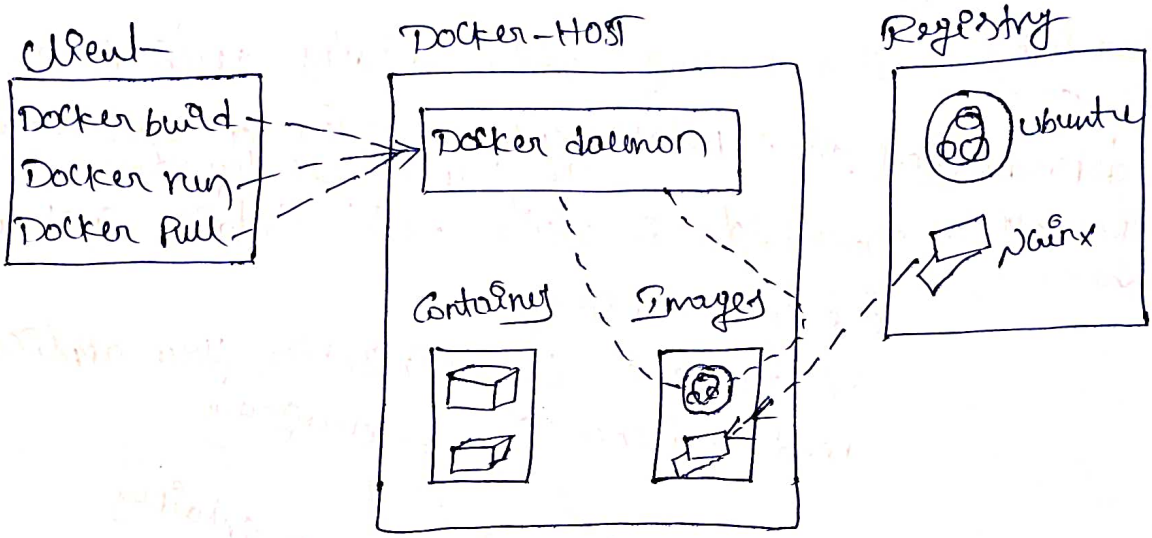


Directly without deploy (.jar/.war) in Deployment Server (tomcat installed Server). here create one image with the help of dockerfile & that image can store in registry then use that image run as Docker Container in diff. Enviments.

Docker Hub:- It is "A Servic Provided by Docker for finding & sharing Containers image with your teams". it is a storing Container ~~and~~ images.

Docker Architecture :-

When we are installing docker software in server, we are getting client & Dockerdaemon.



Docker client :- It is CLI through which the user send the commands / instruction to docker.

Docker Daemon :- It is exeute the commands sent by the user.

Commands may be installing OS, puling image from registry & running containers.
Docker Proxy is used to forward the trafic to host port.

I) Docker Community Edition (CE) → free

II) Docker Entreprise Edition (EE) → Commercial
   i) UCP - universal control Pane → GUI to manage / monitor Docker cluster
   ii) DTR - Docker Trusted Registry

Image Creation Command :-

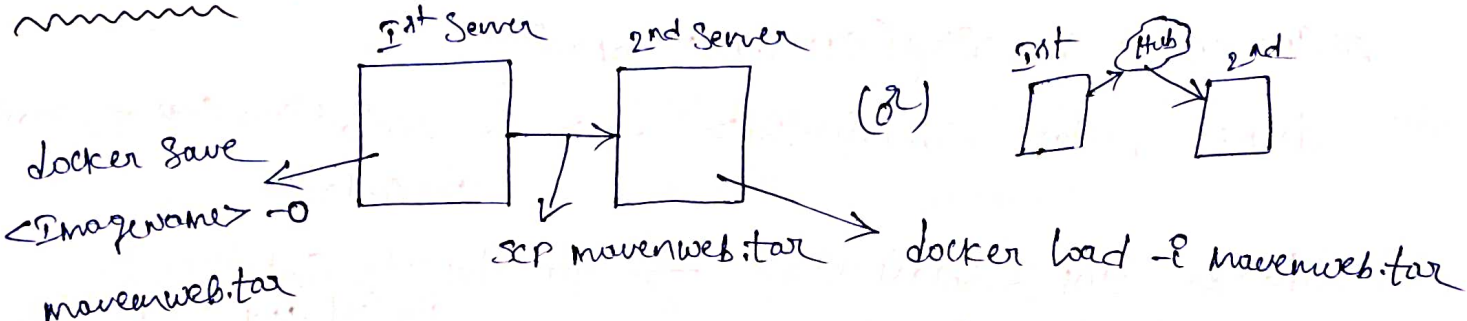
↳ docker build -f <customfilename> -t bavisettiram/mavenwebapp .

↳ docker build -t bavisettiram/maven-web-app . → for Dockerhub

↳ " " " 179.89.89.8:8083/maven-web-app . → for nexus & Jfrog.
nexus.tcs.com

↳ " " ecr.124123132.amazon.com/maven-web-app . → ECR login

Commands :- i) I want to copy docker image from one server to another.



1st create one.tar file for that image in 1st server & copy that file to another server & execute the that command the image can formed from that command.

Container Commands :-

i) docker Create --name <Containername> <Imagename> → for just only creating the container

ii) docker start <Containername|Id> → After creating the container execute this command for starting container.

iii) docker run --name <Containername> <Imagename> → it will creates & start the container.

iv) docker PS (or) docker Container ls → only Runing Containers shows.

v) docker PS -a (or) docker Container ls -a → All Containers shows.

v) docker inspect <Containername|Id> → for more info of Container

vi) docker rm <Containername> → for removing Container
rm -f " → for forcefully.

vii) docker run -d --name mavenwebapp -P 8080:8080 <Imagename>
-d → detachedmode
8080 → hostPort
8080 → ContainerPort.

vii) we can create no of ~~Commands~~ Containers based on Server resources & Capacity (CPU & RAM Capacity).

viii) docker restart <ContainerId> → for restarting (stop & start) the Container.

ix) docker Stop <ContainerId> → for stop the Container grace Period stop the Process.

x) docker Kill <ContainerId> → for ~~forcefully~~ forcefully kill the Process

xi) docker rename <ContainerId/NAME> <Newname> → for renaming the Container.

xii) docker Pause <ContainerId> → for pause the Process that means sleeping mode application not accessable.

xiii) docker unpause <ContainerId> → unpause sleeping mode apps.

xiv) docker top <ContainerId> → what Process is running in Container showing.

xv) docker run --memory="512mi" --CPU="0.5" → ~~re~~resources limit giving while creation of Container.

xvi) docker stats <ContainerId> → CPU & memory Consumption for Container showing.

xvii) docker logs <ContainerId> → logs information what happens.

xviii) docker exec <ContainerId> ls → Container files shows

xix) docker exec -it <ContainerId> bash → going inside the Container.

xx) docker attach <ContainerId> → for attaching the Process to current shell that means whts going on inside Container here directly showing on server shell).

xxi) we can copy files from Docker host to Container & Container to docker host.

↳ docker cp <sourcefile> <ContainerNAME>:/DestinationPath>

↳ docker cp <ContainerID>:/sourcefile> <DestinationPath>

COPY!- It is used in dockerfile to create an image. Copy diffrent & docker cp is different.

Dockerfile!- It is a file which will have instructions to create an image. Docker will Process these Commands from top to bottom.

FROM!- Indicates Base image on top it we can create our own image.

FROM <imagename>

Ex!-

FROM tomcat:jdk8.0.1

↓ this image for base image is another → another →

MAINTAINER!- Author/owner of docker file Image.

Ex!-

MAINTAINER vamo60895

COPY!- It is used to copy local files/folders to image while creating an image.

Ex!-

COPY <Source> <Destination>

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

ADD!- ADD also copy files to image. ADD can copy local files & also it can download files from remote location. if you are downloading tar files & will add & also extract that file in image.

Ex!-

ADD <Source> <Destination>

ADD <URL of tomcat> . → current directory of an image.

RUN:- Using RUN we can excutes commands on top of base image. RUN instruction will be executed while creating an image. RUN instruction can be used to install/setup required softwares/configuration while creating an image.

Ex:-
```
RUN mkdir /app
RUN yum install java
RUN curl
```

CMD Ex:-
```
CMD["echo", "welcome CMD"]
```

CMD:- CMD instruction/command will be executed while creating a container. it can be used to start the process (application) inside container only while creating a container.

ENTRYPOINT:- ENTRYPOINT instruction also will be executed while creating a container. we can set an entry point (command) for our container which we want to execute.

CMD commands can be overridden while creating a container. where is Entrypoint can't be overridden while creating container. this CMD & ENTRYPOINT can process/execute the MOST Recent CMD/ENTRYPOINT when no. of CMD & ENTRYPOINTS.

NOTE:- 1) If container Run then it will prints later suddenly it is going to exited state (docker PS -a). because here there is no Software Test echo & Run statments are printing.
2) you can pass both ENTRYPOINT & CMD in one file also based on requirement.

WORKDIR:- Using this we can set working directory for an image/container. it is like cd.

Ex:- WORKDIR /usr/local/tomcat

ENV:- We can set environment variables using ENV. these variables can be accessable in image (while creating image) & also we can access in container.

Ex:- ↳ ENV <key> <value>

↳ ENV CATALINA_HOME /usr/local/tomcat

↙ refering ENV as variable

LABEL:- LABELS are key value pair which we can add to the image. It is like meta-data. it's data about data.

Ex:- LABEL branchname develop

if you are inspect you container, this data will shows like which container is code of github branch.

ARG:- ↳ ARG branch=develop

LABEL branchname $branch

(or)

you can pass runtime also these args (interactive way).

→ docker build -t <imagename> --build-arg branch=develop .

It is like a variable which we can define & reffer in dockerfile. At runtime (while creating image) also we can pass Argments.

'ENV' variable can accessable while creating an image & inside the container oswell. but 'Args' variable can accessable while creating image but not in ~~side~~ inside container.

USER:- We can set an user for an image or container. The instruction will be executed as whatever user we have set using USER.

Ex:- USER <username>

USER jenkins

↓

Jenkins user for container application inside.

EXPOSE:- EXPOSE indicates on which port our container is listening. It's like documentation using this we understanding what port is used.

Ex:- EXPOSE 9941

VOLUME:-

Ex:- volume /var/jenkins_HOME

[/var/lib/docker/volumes/ ls]

~~multiple~~

multistage Dockerfile:-

with multistage builds, you use multiple 'FROM' statements in your Dockerfile.

Ex:- vi Dockerfile

```
#git
FROM alpine/git as repos
MAINTAINER vasu
WORKDIR /app
RUN git clone <HTTPSURL>

#maven
FROM maven:3.5-jdk-8-alpine as build
WORKDIR /app
COPY --from=repos /app/maven-web-application /app
RUN mvn install
#tomcat
FROM tomcat:8.0.20-JRE8
COPY --from=build /app/target/maven-web-app.war /usr/local/tomcat
                                    /webapps/maven-web-app.war.
```

what is shell form & executable form in Docker:-

RUN, CMD, ENTRYPOINT instructions can be defined in shell form & executable form.

Shell form:-

```
RUN <CMD> <ARGS>
CMD <CMD> <ARGS>
ENTRYPOINT <CMD> <ARGS>
```

If I use shell form it will executed as something like this in internally

/bin/<shell> -c <command>

(/bin/bash /bin/sh)

Ex:-

RUN mkdir -p /usr/test
/bin/bash -c mkdir -p /usr/test → Internally it will execute when we give upper command. My later commands also same.
CMD java -jar app.war
/bin/bash -c java -jar app.jar

CMD sh test.sh
/bin/bash -c sh test.sh
ENTRYPOINT java -jar app.jar
/bin/bash -c java -jar app.jar

Executable form:-

RUN ["CMD", "ARG1", "ARG2"]
CMD ["CMD", "ARG1", "ARG2"]
ENTRYPOINT ["CMD", "ARG1", "ARG2"]

Ex:-
RUN ["mkdir", "-p", "/usr/test"]
/bin/mkdir -p /usr/test
CMD ["java", "-jar", "app.jar"]
/bin/java -jar app.jar
CMD ["sh", "test.sh"]
ENTRYPOINT ["java", "-jar", "app.jar"]
/bin/java -jar app.jar

which one Preferable for CMD & ENTRYPOINT:-

Any
↳ Executable

Docker volume :-

frontEnd → Spring Boot mongo → Stateless

Backend → mongo (Image from hub) → statefull

Run as Container for Both application :-

for this you create a network,

↳ docker network create <networkname>

for Spring boot mongo :- (frontEnd) :-

↳ docker run -d -P 8080:8080 --name SPRINGAPP -e MONGO_DB_HOSTNAME=mongo -e MONGO_DB_USERNAME=devlab

-e MONGO_DB_PASSWORD=dev@123 --network <networkname> <Imagename>

Same name give for Container of mongo.

↑

↳ Environment variables in Raw code.

for mongo (Backend Connect to frontEnd for maintaing data) :-

↳ docker run -d --name mongo -e MONGO_INITDB_ROOT_USERNAME=devlab -e MONGO_INITDB_ROOT_PASSWORD=

dev@123 --network <networkname> <Imagename>

→ Same name for Hostname.

whatever data is coming from users that data will stores in mongo Container. If Container is deletes or down that data will not Consistency. So to overcome this dockervolumes are coming.

Docker volume!- Docker volume are file systems mounted on Docker Containers to Persist data generated by running Container. It is a share file systems. if Container dies also the data will Safe & with this data you can Put & create another Container also.

mutable!- Anything which can be modified.

Immutable!- It can't be modified. docker is a immutable.

there two types volumes.

1) local volumes
   i) Bind Mounts → It's file/folder from dockerhost. which is mounted with Container directry.
   ii) Docker Persistent volumes
2) External volumes (network volumes).

for Bind Mounts while creation of Container!-

↳ mkdir mongodb → for creating directry which mounts on Container.

Path → /home/ubuntu/mongodb

I^st Run statelessapps then stateful is below,

↳ docker run -d --name mongo -e MONGO_INITB_ROOT_USERNAME=devdb -e MONGO_INITB_ROOT_PASSWORD=dev@123 -v /home/ubuntu/mongodb:/data/db --network <networkname> <Imagename>

→ for mongo Image Run as Container.

mounting Path in location.

this mongo Image volumes store in this location, if you want your Image in hub & check details of path.

↓

ls mongodb/

↓shows all files of mongo Container.

If you delete the docker, do you know what is directory of container. we don't know because this only for bind mounts. if you are inspect the container or volume then only you knows which file/folder is mounted.

Recommended way use the persist volume because if you want migrate docker from one server to another. you need all volumes & network & containers .. etc so if you use persist volume you are able to achieve this.

Persistent volume :- the storage can access later point of time.

docker volume create <volname>

↓

docker volume ls (shows list of volumes)

↓

docker inspect volume <volname> [for more info. /var/lib/docker/volumes/vol1/_data]

↓

if container dies the data will safe.

↓

1st Run Stateless & Stateful,

↳ docker run -d --name mongo -e MONGO_INITDB_ROOT_USERNAME: dwdb -e MONGO_INITDB_ROOT_PASSWORD: dw@123

-v <volname>:/data/db --network <networkname> <Imagename>

volumename here (this is mounts on home Dir of Docker).

↳ docker run -itd --name nginx2 --Privileged=true --volumes-from nginx1 -P 8122:80 nginx

- nginx2 → new container creating with same vol of Prev Container.
- nginx1 → Prev container name
- nginx → Image name

Install ebs volume for the container so you need to install one plugin, (rexray/ebs plugin)

↳ docker plugin install rexray/ebs EBS_ACCESSKEY=<accesskey> EBS_SECRETKEY=<secretkey>
↳ for installing plugin

↓

docker plugin ls → Shows installed plugins.

↓

docker volume create -d rexray/ebs <volname>
↳ Type ↳ for creating volume

↓

docker volume ls (shows volumes here)

↓

use this volume & create Container. whatever name your given for this volume under creation (rexray/ebs) same volume give for that Container. So all data will Storing in aws Ebs volume.

↳ docker run - - - —

↓

docker inspect <volname>

what is readonly volumes how can we attach volumes as readonly?

Same Command as above,

↳ docker run -d --name mongo -e MONGO_INITB_ROOT_USERNAME=devdb -e MONGO_INITB_ROOT_PASSWORD=dev@123 -v ~~mongodbe~~<volname>:/data/db:ro --network <networkname> <Imagename>.

readonly

Docker-Compose:-

Docker-Compose is a tool, which runs as single/multiple containers. You can Put & write through a command by in 'yaml' file.

Install Docker-Compose:-

Docker-Compose file:-

```
version: "3.2"
services:
springapp:  ──→ stateless app name
  image: <Imagename>
  ports:
  - 8080:8080
  networks:
  - <networkname>
  environments
  - <MONGO Hostname>
  - <MONGO USERNAME>
  - <MONGO PASSWORD>
  Containername: mongo  ──→ Same name give for mongodb container.

mongo:
  image: <Imagname>
  environments
  - MONGO_INITB_ROOT_USERNAME=dwdb
  - "  "  "  - PASSWORD=dwdb@123
  volumes:
  - mongodbvol: /data/db  ──→ volumes creates
  networks:
  - <networkname>

volumes:
  mongodbvol:  ──→ volume name
    driver: rexray/ebs  } ? If you don't mention this then default volume takes.

networks:
  <networkname>
    driver: bridge.
```

} Here volumes & networks Creates by docker-Compose.

↓

docker-Compose Config → for validate the file

↓

docker-Compose up -d → for runs the Compose file.

↓

docker-compose PS

↓

docker-compose start & stop & restart

↓

docker-compose down → for deleting the Compose.

if you have a volumes & network the if you want to use those. then modify some changes in file,

```
services:
  springapp:
    :
    :
  mongo:
    :
    :
volumes:
  mongodbvol:
    external: true
networks:
  <networkname>
    external: true
```

make it as "external is true" when if you have already volumes & networks are available.

Docker networking :-

It is Primarly used to establish communication b/w Docker Containers & the outside word via host m/c. there are diff types of networks.

1) Bridge network
2) overlay network
3) macvlan network.

Bridge network means when we Run's any container that Container Consisting a default network that network called as bridge network. while creating of any network the driver option is needed if you are not give driver option then bridge network form.

1) Creating network :-

↳ docker network Create <networkname>

2) more info about network :-

↳ docker network inspect <networkname>

3) list of all networks :-

↳ docker network ls

4) Remove network :-

↳ docker network rm <networkname & Id>

5) network attaching while creation of container :-

↳ docker run --name <Containername> --network <networkname> <Imagename>

6) If two Containers are Same network then Communication happens.

↳ ping IPAddress (pvt) for checking

7) network attaching for running container:-

↳ docker network connect <networkname> <Containername & Id>

8) network detaching for running container:-

↳ docker network disConnect <networkname> <Containername>.

9) Going to particular Container:-

↳ docker attach <Containername>

NOTE:- you can attach the different network for any single Container.