Docker

=> It is containernization platform
=> Containernization means code + environment or code + dependencies packing

Where code is our app/project code and env/dependencies are like OS , JVM , web server s/w , DB , jar files /libraries and etc..

=> Docker is platform independent tool i.e the code can be in any language and env.. can be there in any setup

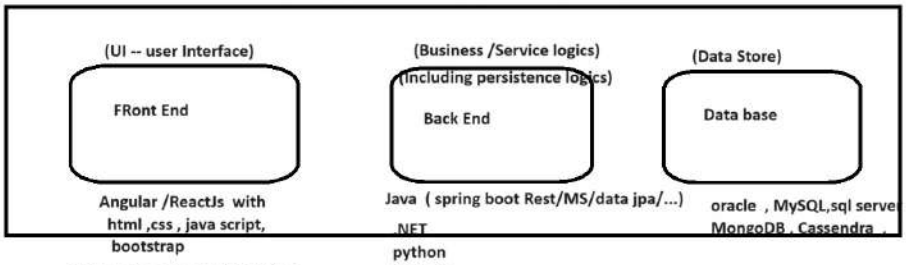
=> Docker Tool makes our code easily portable and deployable across the multiple machines

In Docker Tool

=> Application Stack
=> your Life with out Docker
=> Your life with Docker
=> Docker definitation
=> Docker architecture
=> Docker Installation (Linux VM - AWS)
=> Docker commands
=> Docker file
=> Docker image
=> Docker Network
=> Docker Volumes
=> Docker Compose
=> Docker swarm (failed alternate to K8S-Kubernates for orchestration platform)

Orchestration is managing the docker container by enabling feature called autoscaling i.e increases or decreases docker containers creation as needed

PRoject Application stack (Typical new projects)



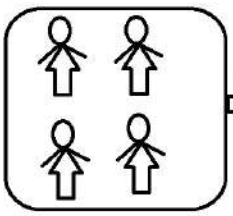
Angular /ReactJs with html ,css , java script, bootstrap
(Spring boot mvc with jsp files is bit outdated front end)

Java (spring boot Rest/MS/data jpa/...)
.NET
python
Node Js

oracle , MySQL,sql server , PostgreSQL
MongoDB , Cassendra ,

Profiles / Environments in Project development to Release to Production

=> Setting up profile is all about keeping the env /setup ready that is required for the Project execution
=> Using these setups of different profiles/env.. we generally develop/ test the code at various levels to make sure code is executing is smoothly



Project Code

Code Repository /VCS Software
GIT Hub Repo

Profiles/ Env.. For Testing Project

Dev Profile /env
-> jdk 11
-> tomcat 10
--> Window OS
-> MySQL
-> .....
(For software Company)

(SIT Profile /env)
Test Profile /env
-> jdk 11
-> tomcat 10
--> windows
-> mySQL
(For software Company)

=> For every Profile , we need to keep one Linux VM Instance of AWS ready to develop, deploy and test project
=> Dev profile VM is for both development and unit testing
=> Test /UAT /Pre Prod profile VMs are purely for Testing at various levels
=> Prod profile VM is pure for keeping the Project in Live code for endusers

note:: The Tests that happens at client organization after releasing the Project is called UAT (User acceptance Test)

Prod Profile /env
-> jdk 11
-> wildfly
-> Linux OS
-> Oracle
(For Client Org)

Pre-Prod Profile /env
-> jdk 11
-> wildfly
-> Linux OS
-> Oracle
(For Client Org)

UAT Profile /env
-> jdk 11
-> tomcat 10
-> Linux OS
-> mySQL
(For client org)

Life with out Docker

=> Dev Ops team is responsible to keep these Profiles /env.. related machines ready to develop/test the project in different env...
=> if Docker is not there , the dev Ops Team needs to keep all the profiles/envs.. very much ready manually which is complex process

=> Some times software company machines setups of development and Test/SIT (System Integration Test) may not match with Client Org's machines setup (Cloud or outside Cloud) (we generally observe the following mismatches)

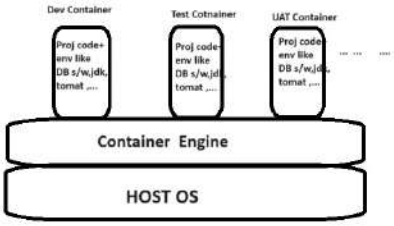
Company (Client org)
=> software uses window machines/mac mac machines for development but they use Linux/ unbuntu machines
=> Software company might have developed the java code using java 11 setup , But the client org may provide java 8 setup
=> Software company might have used Oracle 11g , but the client organization may provide oracle 19c setup
and etc..

(all these indicates compitablity uses to run the Project code in different profile/env.. machines)

To solve these problems take the support containerization using the Docker tool

Life with Docker

=> Docker automates the env/profile setup process by using the concept called containerization which perform$ code + env packing



HOST the OS will change based on setup of the profile

=>

Docker Containernization

=>Docker logo is ship carrying comps indicating docker makes the code as easily shippable code

=> Once we keep Project code + env setup in the Docker container , we can make that docker container working any machine for any developer or enduser

=> Docker passes the statement to Programmers/Developers u just keep Application code ready becoz the Docker it self arranges all s/ws , OS and etc.. that are required to execute the code in different profiles /environments

Docker Definitation
==========

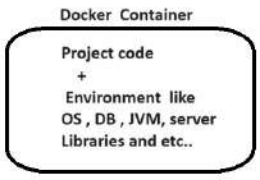
Docker is a tool or platform for packaging , deploying and running the Applications

Advantages with docker
==========

a) Docker enables the developers / devops team to separate our application/project and the intrastructure (hardware setup) so that we can deliver software project quickly

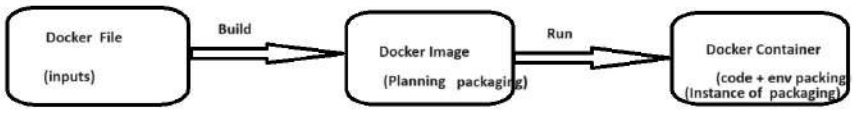
b) Docker packages the software/project and its env/dependencies into units called containers that have every thing that the project needs to run including libraries(jar files), tools , code and Run time setup

c) By using the docker style packaging, deployment , execution and shipping which is quick process and which reduce the time delay beween developing the code in Dev and running the code prod Profile



profile in

Docker Architecture (High Level)
==========



Main Comps of Docker Architecture
==========

=>Docker file (Set of instructions to build the docker Image)
=> Docker Image (Package which contains the planning of Source code + env packing)

=> Docker Registry (The registry where docker images are available : eg:Docker HUB , Aws ECR and etc..)
=> Docker Container (An Instance of Docker image using which we can run our application code in certain env.. /profile that is specified in the docker image)

=> Docker Client (An implicit comp in the docker tool or software who is actually resposible to execute the Docker commands)

Procedure to install Docker in Amazon Linux VM of Aws Cloud
==========

step1) create Linux VM Machine of type t2.micro in Aws cloud

step2) Launch mobxterm tool connecting to the above VM

step3) Perform the following commands execution to keep Docker ready in our machine

update the yum package manage

```
$ sudo yum update -y

# install the docker
$ sudo yum install docker -y

# start the docker server
  $ sudo service docker start


# Add the current user (ec2-user) to the docker group
  $ sudo usermod -aG docker ec2-user

      # To get information about docker tool
$ docker info       (gives access permission error)


# Restart the Mobxterm session
    $ exit
```

```
Session stopped
    - Press <Return> to exit tab
    - Press R to restart session
    - Press S to save terminal output to file
```

"R"

```
  # To get information about docker tool
$ docker info

    ..
    ...   Gives info about docker tool
    ...
    ..
```

Docker Commands
==========

```
# To get info about docker tool
  $ docker info

# To display list of docker images

   $ docker images      (Initiall shows no images)

#  To pull the docker image fom docker hub (docker registry)

       syn :: $ docker pull <image id>/<image name>

 # To pull the ready made "hello-world" image

         $ docker pull hello-word
```

```
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY    TAG       IMAGE ID   CREATED   SIZE
[ec2-user@ip-172-31-0-222 ~]$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:1408fec50309afee38f3535383f5b09419e6dc0925bc69891e79d84cc4cdcec6
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    d2c94e258dcb   14 months ago   13.3kB
```

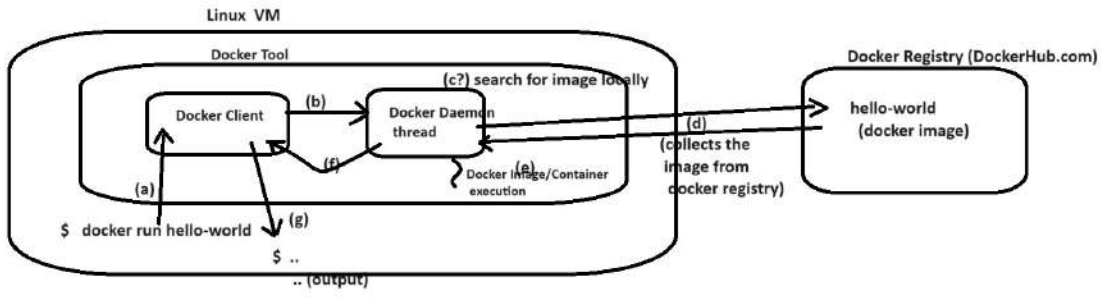
```
  # To delete the Docker image

      syn :: $ docker rmi <image id/image name>

          $ docker rmi hello-world
```

```
[ec2-user@ip-172-31-0-222 ~]$ docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:1408fec50309afee38f3535383f5b09419e6dc0925bc69891e79d84cc4cdcec
Deleted: sha256:d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a
Deleted: sha256:ac28800ec8bb38d5c35b49d45a6ac4777544941199075dff8c4eb63e093aa81e
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

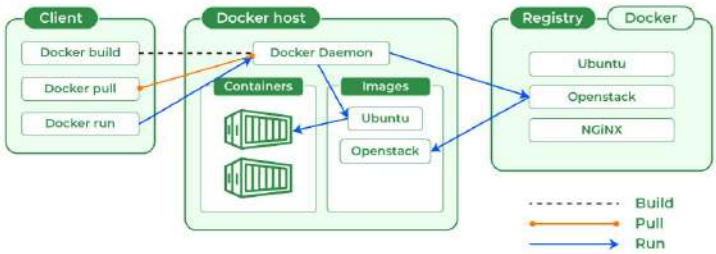
```
  # Run the Docker container       (By dunning docker image we can get docker container)

          $ docker run hello-world
```

(Running the docker container is nothing but running the App code that is there inside the docker container using the env.. /dependencies packed in the docker container)

```
[ec2-user@ip-172-31-0-222 ~]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:1408fec50309afee38f3535383f5b09419e6dc0925bc69891e79d84cc4cdcec6
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```



Complete Docker Architecture
==========



# To get all active /running docker containers
$ docker ps

To get all docker containers (both active and stopped)

$ docker ps -a

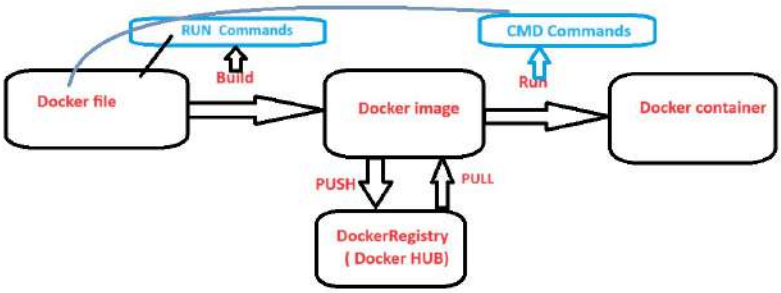
```
[ec2-user@ip-172-31-0-222 ~]$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
a202ad7fc4c4   hello-world   "/hello"   4 minutes ago    Exited (0) 4 minutes ago              vigor
f004d23f495d   hello-world   "/hello"   23 hours ago     Exited (0) 23 hours ago               xenod
```

To remove the docker container

$ docker rm 454546adff
(container id)

```
[ec2-user@ip-172-31-0-222 ~]$ docker  rm  f004d23f495d
f004d23f495d
```

Understanding the process of developing the docker file using which the docker image will be bult and later can run that image to create docker container
==========



Docker File development
==========

=> In Docker File , we place commands/instructions that are required to build docker image
=> The instructions in docker file will be given using DSL (Domain specific language based on java and groovy)
=> Any <filename> can be taken as the docker file name
=> The default docker file name is "Dockerfile"

=> We generally use the following commands in the development of Docker File

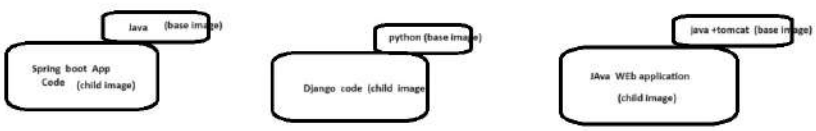
FROM
MAITAINER
COPY
ADD
RUN
CMD
ENTRYPOINT
ENV
LABEL
USER
WORKDIR
EXPOSE
VOLUME
and etc..

note :: DockerFile commands are not case-sensitive

FROM
=====

=> It allows us to specify the base image on which our app should run be
=> On the top of base image /images our application image will be created and will executed

=> The base images can be the OS name , language name , base software name using which env our code executes



Syntax :: FROM <image-name>

eg :: FROM java: jdk-1.8.0
FROM tomcat: 10.0.1
FROM mysql

note:: The commands kept in docker file are sequential commands i.e they execute top to bottom

note:: Using docker build command that "DockerFile" as the input one docker image will be created representing our App/Project .. Inside that image of our App/Project .. what are the other images that we should include will be

as base images will be decided by using "FROM" command

MAINTAINER
==========

=> It allows us to specify the the "author" or "developer" of the Docker File using the Docker image will be created

eg:
MAINTAINER nataraz <natarazjavaarena@gmail.com>

COPY
=====

=> This command allows us to copy the content(files/folders) from the current machine FileSystem to docker image while creating the Docker Image

Syntax:
COPY <Source Location> <Destination -Location>

Copy target/first-java-webapp.war /usr/local/tomcat/webapps

note:: Using this only Copy Operation is possible , but downloading the content from the specifiled URL is not possible

ADD
=====

=> It is also useful to copy file/folders from one location to another location while creating the Docker image but it is also capable downloading the content from the URL to given the Destination folder

Syntax ::
a) ADD <source file/folder> <Destination>
b) ADD <source url> <Destination>

What is difference between COPY AND ADD Commands?

=> COPY can take only Current Machine File System Location as the Source Location
i.e it can not be used for downloading the content

=> ADD can take Current Machine File System Location and Internet URL as the Source Location
i.e it can be used even for downloading the content

RUN
====

=> It is useful to execute commands (like Linux commands) while creating the docker image (as part of docker build operation)
=> we can write multiple RUN commands /instructions in a docker file
=> if multiple RUN commands are placed in a Docker file then all commands will execute from top to bottom

syntax :: RUN <command syntax>

RUN mkdir workspace1

RUN YUM install git

RUN yum install maven

=> if we place multiple RUN commands in the docker file then all the Run commands will execute in the given sequence

CMD
=====

=> Useful to execute command during the process of Docker container creation (Running the docker image to create the Docker container)
=> if we place multiple "CMD" commands in a docker file , only last command will execute i.e technically we can place multiple "CMD" commands in a docker file ,but only the last command will execute in that file

syntax: cmd <command to execute>

eg: CMD JAVA -jar App1.jar

PRocedure sample Docker Image and Docker Container using Docker File
==========

step1) create docker file having name '"Dockerfile" and place the following instructions

$ vi Dockerfile

```
FROM ubuntu
MAINTAINER nataraz <natarazjavaarena@gmail.com>
RUN echo "1st run "
RUN echo "2nd run"
CMD echo "1st cmd"
CMD echo "2nd cmd"
RUN echo "3rd run"
CMD echo "4th cmd"
```

```
[ec2-user@ip-172-31-0-222 ~]$ docker build -t dock-img1 .
[+] Building 6.7s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 282B
 => [internal] load metadata for docker.io/library/ubuntu:latest
 => [internal] load .dockerignore
```

step2) build the docker image from the Dockerfile

$ docker build -t dock-img1 .

image name — image location

```
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
dock-img1    latest    4e3f51965067   3 minutes ago    78.1MB
```

```
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/ubuntu:latest@sha256:2e863c44b718727c860
=> => resolve docker.io/library/ubuntu:latest@sha256:2e863c44b718727c860
=> => sha256:35a88802559dd2077e584394471ddaa1a2c5bfd16893b829ea57619301e
=> => sha256:9c704ecd0c694c4cbdd85e589ac8d1fc3fd8f890b7f3731769a5b169eb4
=> => sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5cd9058fc4362
=> => sha256:c920ba4cfca05503764b785c16b76d43c83a6df8d1ab107e7e6610000d9
=> => extracting sha256:9c704ecd0c694c4cbdd85e589ac8d1fc3fd8f890b7f37317
=> [2/4] RUN echo "1st run "
=> [3/4] RUN echo "2nd run"
=> [4/4] RUN echo "3rd run"
=> exporting to image
=> => exporting layers
=> => writing image sha256:4e3f51965067e6995ecc8ba6ab6daccf53b7883253baa
=> => naming to docker.io/library/dock-img1
```

step3) Run the image to create the container

$ docker run docker-img1

```
[ec2-user@ip-172-31-0-222 ~]$ docker run dock-img1
4th cmd
```

note:: Using one Dockerfile we can create multiple docker images , similarly using one docker image we can create multiple docker containers

$ docker build -t dock-img2 .

```
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
dock-img1    latest    4e3f51965067   11 minutes ago   78.1MB
dock-img2    latest    4e3f51965067   11 minutes ago   78.1MB
```

$ docker run dock-img1

```
[ec2-user@ip-172-31-0-222 ~]$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS
84120402b28    dock-img1   "/bin/sh -c 'echo \"4…"   46 seconds ago   Exited (0)
svesvaraya
e1e3979ca78f   dock-img1   "/bin/sh -c 'echo \"4…"   4 minutes ago    Exited (0)
own
```

Using other than default name for the Dockerfile

=> For this we need to use "docker build -f " option

syn: $ docker buid -f <docker file name> -t <image name> .

step1) copy content of Dockerfile to nat-dock-file

$cp Dockerfile nat-dock-file

step2) build the docker image

$ docker build -f nat-dock-file -t imageone .

step3) run the docker image to create docker container

$ docker run imageone

Procedure to keep docker image in docker hub (dockerhub.com)

step1) make sure that u r docker image is ready

```
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
dock-img1    latest    4e3f51965067   35 minutes ago   78.1MB
dock-img2    latest    4e3f51965067   35 minutes ago   78.1MB
dock-img3    latest    4e3f51965067   35 minutes ago   78.1MB
imageone     latest    4e3f51965067   35 minutes ago   78.1MB
```

step2) login to docker hub from VM terminal

```
[ec2-user@ip-172-31-0-222 ~]$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head
 over to https://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and
 is required for organizations using SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: nataraz@gmail.com
Password:
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```


step3) tag the docker image with logical name

$ docker tag imageone nataraz/imageone

image name — tag name

step4) Push the image to docker registry by specifying the tag name

$docker push nataraz/imageone

note:: To pull the docker image from the docker hub or docker registry

$docker pull nataraz/imageone

=> Run the image to create container

$ docker run nataraz/imageone

EntryPoint

=> Entry POINT is given to execute the command while creating the docker container from docker image

=> we can override CMD instructions where as we can not override ENTRYPOINT instructions by passing values as cmd line args

eg::

```
ENTRYPOINT ["echo","welcome to docker"]
ENTRYPOINT ["java","-jar", "App1.jar"]
```

Example on "CMD" and "ENTRYPOINT" difference

natDocfile

```
FROM ubuntu:latest
MAINTAINER nataraz <natarazjavaarena@gmail.com>
RUN echo "1st run "
RUN echo "2nd run"
CMD echo "1st cmd"
CMD echo "2nd cmd"
RUN echo "3rd run"
CMD echo "4th cmd"
ENTRYPOINT ["echo","welcome to docker"]
```

$ docker build -f natDocfile -t imagethree .

```
[ec2-user@ip-172-31-0-222 ~]$ docker run  imagethree
welcome to docker /bin/sh -c echo "4th cmd"
```

Giving both last command and entry point output

```
[ec2-user@ip-172-31-0-222 ~]$ docker run  imagethree how are u
welcome to docker how are u
```

Overriding "CMD" output with cmd line arg , but the same is not possible for "ENTRYPOINT" output

WORKDIR

=> It is useful to set an working directory for an docker image/container

eg:: WORKDIR <DIRNAME>

note:: Once we place WORKDIR instruction in dockerfile , the next instructions of the same docker file will start executing from "WORKDIR" location

In Dockerfile

```
WORKDIR /home/nataraz/app1
RUN sh "git clone <url>"
RUN sh "maven clean package"
```

Both these commands execution takes place from /home/nataraz/app1 folder

ENV

=> It is useful to set env.. variable values like PATH ,CLASSPATH and etc...

syn: ENV <key> <value>

eg: ENV "PATH" "d:/java/jdk1.8.0_1/bin"

LABEL (For documentation and For display)

It is useful to represent the data in the form of key - value pairs

eg: LABEL branch-name release (It is like a variable holding a value)

<key> <value>

EXPOSE

=> It is also a documentation command that is useful to expose /display the port number on which the docker container is running

syn:: EXPOSE <port>

eg:: EXPOSE 8181

USR

=>It is useful to set username for creating Docker image /container

eg:: USR root

ARG

It is useful to avoid hardcoded values in the dockerfile i.e it support coding of value by collecting the values as the cmd line arg values

in Dockerfile

```
eg:: ARG branch
     RUN sh 'git clone -b $branch <repo url>'
```

$ docker -build -t <imagename> -build-arg branch=netbanking

Here we are suppluing the arg value

Volume

=> It is useful to specify the storage location for our docker container

eg :: Volume /user/app1

Commands in Dockerfile

- FROM
- MAITAINER
- ADD
- COPY
- RUN
- CMD
- ENTRYPOINT
- VOLUME
- ARG
- LABEL
- WORKDIR
- USR
- EXPOSE

To get All docker images

$ docker images

For docker login

$ docker login

To tag the docker image

$ docker tag imagethree nataraz/image3

TO push docker image into docker hub

$ docker push nataraz/image3

To pull the docker image

$ docker pull nataraz/image3

To run the docker image

$ docker run nataraz/image3

To remove the specific docker image

$ docker rmi <image -name /image -id>

To build the docker image by taking Dockerfile as the file name

$docker build -t <image -name> .

To build the docker image by taking any file name as doker file

$docker build -f <filename> -t <image -name> .

To remove the specific docker image though its docker container running

```
$ docker rmi -f <image-name/image-id>
```

#To display all running containers

```
$ docker ps
```

#To display all containers (both stopped and active)

```
$ docker ps -a
```

To stop the docker container

```
$ docker stop <container id>
```

To remove the docker container

```
$ docker rm <container id>
```

To remove all the stopped contianers and unused images

```
$ docker system prune -a
```

```
[ec2-user@ip-172-31-0-222 ~]$ docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
a162fddf0822c4551eec21b6a4a068b8e35978aa3f8f66b7cdba26f64d484181
e2fb7bb912519ee08503634ff27cfe04ab58b54775ead0a6e3acea76813b1073
31c2abd156cec29989cf6f6e3700bf66e5fe36a4cbb0acfb454e521575b590a2
aeeff06514c100fb48fdb22ff5b1398458ec8d33af025d002b4a4aae9b509209
5f712403a75b0a25b82cc771334451fcfd0e91d814343a3eb6885fd5d66ef4fa
e28902be4a725a299bf2714ea6d198cd67a934426a97cc21462cf471216e43ae
016fea46108c822b00e6d7cc9fb587427e02b4db0e2624bb30e44a29cb9e2bb9
788bdacb728e6413b0278ca0920c8c5a83f9a6aee7ea291fd3f46e6f93786139
841200402b28ee37734d97404d45963c0ca577b6d98dcffb3a738d39f5b8ecc5
e1e3979ca78f5ce35d929be9d4700a04307b4066137c0bf15ab940cfe5773771

Deleted Images:
untagged: dock-img1:latest
untagged: dock-img2:latest
untagged: dock-img3:latest
untagged: nataraz/imageone:latest
untagged: nataraz/imageone@sha256:5b5b419d0dda4f723e13f1826367a05fa817f48eabbc312c532
deleted: sha256:4e3f51965067e6995ecc8ba6ab6daccf53b7883253baa1e68dc8c3d68ba4142f
deleted: sha256:63594d2971dc083a8211f5da0a1f62982ad77b10d5b3d19f1d52249b204a81c9
untagged: imagethree:latest
untagged: imagetwo:latest
deleted: sha256:9ff9a6bccf716204128b842a7b1c5863a865a2f279c2414f53c17c6b1bec8338
```

Containerizing the java web application (with out spring/Spring boot)

==========

=>Docker container is an instance of docker image that contains App code + dependencies
So that docker container can be executed in any machine

As of now java web applications are developed in two ways

a) Java web application using servlet , jsp technologies (No Spring/Spring boot) (just 20%)

b) Java web application using Spring /Spring boot (80%)

=> In option1 , we need to take web server s/w seperatly becoz servlet,jsp apps do not give built-in servers
=> In option2 (especially in sprng boot) , we need not to take web server s/w seperatly becoz spring boot gives web server s/w as the embedded server

Containerizing the java web application (with out spring/Spring boot)

=>Docker container is an instance of docker image that contains App code + dependencies
So that docker container can be executed in any machine

As of now java web applications are developed in two ways

a) Java web application using servlet , jsp technologies (No Spring/Spring boot) (just 20%)

b) Java web application using Spring /Spring boot (80%)

=> In option1 , we need to take web server s/w seperatly becoz servlet,jsp apps do not give built-in servers
=> In option2 (especially in sprng boot) , we need not to take web server s/w seperatly becoz spring boot gives web server s/w as the embedded server

Dockerizing the normal Java web application (servlet ,jsp based web application)

Application Code = Java web application using servlet ,jsp and build with maven tool
Dependencies = Java + Tomcat

Sample Dockerfile

```
FROM tomcat:10.1-jre11                (Gives both tomcat and java images)
COPY target/docker-webapp.war /usr/local/tomcat/webapp/my-docker-webapp.war
EXPOSE 8080
```

END to END PRocedure

step1) use eclipse IDE to develop servlet,jsp based web application and test that local tomcat server

a) create maven project using maven-archetype-webapp archetype

b) add Jakarta servlet api dependency to the pom.xml

```
<!-- https://mvnrepository.com/artifact/jakarta.servlet/jakarta.servlet-api -->
<dependency>
  <groupId>jakarta.servlet</groupId>
  <artifactId>jakarta.servlet-api</artifactId>
  <version>5.0.0</version>
  <scope>provided</scope>
</dependency>
```

c) change the java version to 11 in pom.xml file and also perform maven update operation

d) Change project's dynamic webmodule version to 5.0

Project --->properties ----> Project facets ---> dynamic web module 5

e) Develop the application code

index.jsp

```
<h1 style="color:red;text-align:center"> Docker Testing web application </h1>

<a href="dateurl"> show date and time</a>
```

DateServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/dateurl")
public class DateServlet extends HttpServlet {
```

```
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
          //get PrintWriter
        PrintWriter pw=res.getWriter();
        //set response content type
        res.setContentType("text/html");
        //   write b.logic
         Date d=new  Date();
         //write the response
         pw.println("<h1 style='color:red'> Date and Time::"+d+"</h1>");

         //add home hyperlink
         pw.println("<br> <a href='index.jsp'> home </a>");

         //close the stream
         pw.close();
        }

    public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        doGet(req, res);
    }

}
```

web.xml
=======

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

step2) Test the web application in Local Tomcat server

step3) perform maven package operation to generate war file

Right click on the Project ----> run as ----> build .. --> goals :: package

Gives "Docker-webapp.war" file in maven's target folder

step4) Push the code to GIT Code Repository

step5) Using Mobaxterm terminal perform GIT Clone operation

$ sudo yum install git

$ git init

$ git clone https://github.com/natarazworld/JRTP701Repo-EclipseApps.git

$ cd JRTP701Repo-EclipseApps

```
[ec2-user@ip-172-31-0-222 JRTP701Repo-EclipseApps]$ ll
total 0
drwxr-xr-x. 5 ec2-user ec2-user 97 Jul 19 02:37 Docker-webapp
drwxr-xr-x. 5 ec2-user ec2-user 97 Jul 19 02:37 MavenProj01
```

$ cd Docker-webapp/

step6) prepare the "Dockerfile"

note:: This file can be created in eclipse IDE itself , so that this file can be placed in the GIT code repository (In The Project folder of the Eclipse IDE)

$vi Dockerfile

```
FROM tomcat:10.1-jre11
COPY target/Docker-webapp.war /usr/local/tomcat/webapps/My-Docker-WebApp.war
EXPOSE 8080
```

step7) Build the Docker image

$ docker build -t docker-webapp-image .

$ docker images -a

$ docker images -a

```
[ec2-user@ip-172-31-0-222 Docker-webapp]$ docker  images -a
REPOSITORY            TAG       IMAGE ID       CREATED          SIZE
docker-webapp-image   latest    e58fd615cabb   28 seconds ago   274MB
```

step8) add protocol 8080 To inbound rules of Aws Linux Machine's security group

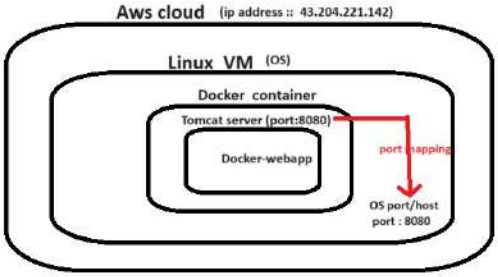
step9) The following way is wrong of running docker container app by creating it from the docker image

$ docker run docker-webapp-image

open the browser and use this url for testing

http://43.204.221.142:8080/ Docker-webapp (gives error)

The reason for this error is tomcat server is running on port number 8080 inside the docker container of Linux VM of Aws cloud i.e tomcat server is not running the port number 8080 of Linux VM of aws cloud directly To solve this problem we need to docker container tomcat server's port number with Linux Host machine any port number (this called port mapping)



http://43.204.221.142:8080/ My-Docker-WebApp (gives errors)

Becoz it is looking for tomcat server running on port number 8080 directly in Linux VM

solution:: map docker container tomcat server port number 8080 with any other port number of Linux OS

8080 : 8080
Linux
(docker port) (host port)

So wrong syntax to create docker container from docker image

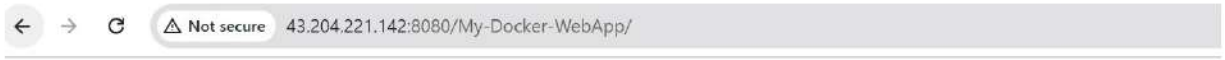
$docker run docker-webapp-image (wrong becoz no port mapping with Liunx HOST VM)

Correct way of creating docker container from docker image

$ docker run -p 8080:8080 docker-webapp-image

port mapping
second 8080 is docker app's port number (tomcat server of docker container)
First 8080 is Linux Host port number (It can be any number)

URL to test the application
http://43.204.221.142:8080/My-Docker-WebApp/



Docker Testing web application

show date and time

How to run docker container in the background with out blocking the shell prompt of the Linux VM?

Ans) use -d option in "docker run" command

$ docker run -d -p 8080:8080 docker-webapp-image

```
[ec2-user@ip-172-31-0-222 ~]$ docker  run  -d  -p  8080:8080 docker-webapp-image
59ec8efcb2ad68277bce3d4e993a1890d722ae01d9e8547b394ba554e2de3a17
```

Pushing the the above docker image to the dockerhub.com (docker registry)

==========

```
[ec2-user@ip-172-31-0-222 ~]$ docker tag  docker-webapp-image  nataraz/docker-webapp
[ec2-user@ip-172-31-0-222 ~]$ docker push nataraz/docker-webapp
```

Dockerizing the Spring boot App (web application/ restful App)

=> Spring boot web application/ restful app can be executed by deploying in the embedded server itself that comes when run spring boot app as the stand alone app

=> Normal java web applications(servlet,jsp) must be deployed in external server by taking the app as the war file

=> if the spring boot app is developed as the war file then it can be deployed in external server or also can be deployed in Embedded server

=> if the Spring boot app is developd as the jar file then it must be deployed only in embedded tomcat server

=> To run the spring boot app that is packed as the jar file , we can use
$ java -jar <jar file name>

(To run this app as docker container instance we need "java" image as the base image , but we do not need tomcat image here)

sample docker file

```
FROM openjdk:17

COPY target/springboot-rest-app.jar /usr/app/

WORKDIR /usr/app/

ENTRYPOINT ["java", "-jar","springboot-rest-app.jar"]
```

STeps for implementation

step1) develop the spring boot Rest App by taking the packing type as the "jar" file

a) set server port number

in application.properties

```
server.port=4041
```

b) develop the Rest Controller class

```
//ActorOperationsController
package com.nt.rest;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/actor-api")
public class ActorOperationsController {


    @GetMapping("/wish/{name}")
    public ResponseEntity<String> showWishMessage(@PathVariable String name){
        return new ResponseEntity<String>("Good Moring::"+name,HttpStatus.OK);
    }

}
```

c) specify jar file in pom.xml

```
<build>
        <finalName>springboot-rest-app</finalName>
```

step2) Test the Application Locally

step3) Pack the app using maven goals "clean" "package"

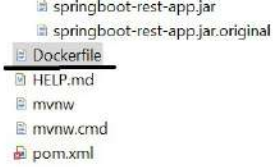
Right click on the Project ----> run as ---> maven build .. ---> goals :: clean package

step4) add Dockerfile to the root folder of the project

Dockerfile

```
FROM openjdk:17

COPY target/springboot-rest-app.jar /usr/app/

WORKDIR /usr/app/

ENTRYPOINT ["java", "-jar","springboot-rest-app.jar"]
```

```
FROM openjdk:17
COPY target/springboot-rest-app.jar /usr/app/
WORKDIR /usr/app/
ENTRYPOINT ["java","-jar","springboot-rest-app.jar"]
```



step5) push the code to GIT Code repository

Right click on the Project --->team ---> share project ----> .... ---> .....

note:: remove "target" folder line and other lines where target folder word is there from the .gitignore file (line 2,4,5) before pushing/sharing the project to GIT repository .. For this we need to open .gitignore file from the Project Explorer

step6) Pull the code to Linux VM instance through mobxterm window

```
$cd JRTP701Repo-EclipseApps
$ git pull
```

```
$ cd BootRestApp
```

```
[ec2-user@ip-172-31-0-222 JRTP701Repo-EclipseApps]$ cd   BootRestApp/
[ec2-user@ip-172-31-0-222 BootRestApp]$ ll
total 28
-rw-r--r--. 1 ec2-user ec2-user   134 Jul 22 02:09 Dockerfile
-rw-r--r--. 1 ec2-user ec2-user 10666 Jul 22 02:09 mvnw
-rw-r--r--. 1 ec2-user ec2-user  6913 Jul 22 02:09 mvnw.cmd
-rw-r--r--. 1 ec2-user ec2-user  1768 Jul 22 02:09 pom.xml
drwxr-xr-x. 4 ec2-user ec2-user    30 Jul 22 02:09 src
```

step7) Build the docker image using Dockerfile

```
$ docker build -t restapp-image .
```

step8) configure u r choice port number in the security group as the inbound rule

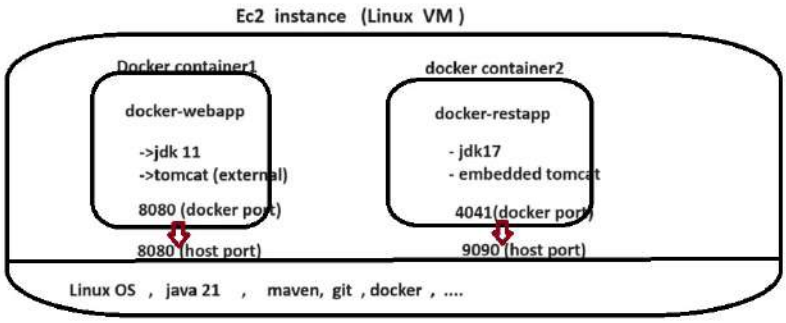
step9) run the docker image to create the docker container and indirectly the rest app using embedded tomcat server

```
$ docker run -d -p 9090:4041 restapp-image
```

```
[ec2-user@ip-172-31-0-222 BootRestApp]$ docker  run  -d -p 9090:4041 restapp-image
bab754bb7d4b7446a9af64e6894aa00b0db58a8fd8219ebf6cfbb60449381764
```

step10) test the application in the browser

http://43.204.221.142:9090/actor-api/wish/raja



docker-webapp (web application)

base image :: tomcat :jre11
docker container port:: 8080
VM host port:: 8080
code :: Servlet, jsp code (web application)

docker -restapp

base image :: OpenJDK:17
docker container port : 4041
VM host port :: 9090
Code :: Spring boot Rest Code (restfull web service)

To push the Docker images to Linux VM

```
[ec2-user@ip-172-31-0-222 BootRestApp]$ docker tag   restapp-image  nataraz/boot-rest-app-image
[ec2-user@ip-172-31-0-222 BootRestApp]$ docker push  nataraz/boot-rest-app-image
Using default tag: latest
The push refers to repository [docker.io/nataraz/boot-rest-app-image]
5f70bf18a086: Mounted from nataraz/docker-webapp
ac73e3491d83: Pushed
dc9fa3d8b576: Mounted from library/openjdk
27ee19dc88f2: Mounted from library/openjdk
c8dd97366670: Mounted from library/openjdk
latest: digest: sha256:42ba3ed24d5dfcdf2a0abfb0e2c263f816f2595d28a2b365f5bd4db6a8888662 size: 1372
[ec2-user@ip-172-31-0-222 BootRestApp]$
```

Docker commands
==========

$ docker info (basic info docker software)
$ docker images (gives all docker images)
$ docker build -t <image-name> . (To create docker image by "Dockerfile" as the docker file name)
$ docker build -f <docker filename> -t <image-name> . (To create docker image by given name as the docker file name)
$ docker rmi <image-id> (to remove docker image)
$ docker pull <image-id/name> (To get docker image from docker hub)
$ docker run <image -name> (To run the docker image)
$ docker tag <image-name> <tag-name> (To provide tag name for the docker image)
$ docker login (To login to docker hub)
$ docker push <image/tag name> (To keep the docker image in the docker hub)
$ docker run -d -p hostport:containerport <image name> (To run the docker container in detached mode with port mapping)

$ docker ps (to get all active docker containers)
$ docker ps -a (To get all active and inactive docker containers)
$ docker stop <container id> (To stop the docker container)
$ docker rm <container id> (To remove the docker container)
$ docker rm -f <container id> (To remove the docker forcefully though it is in running mode)
$ docker rm -f $(docker ps -a -q) (To remove all the docker containers forcefull that are given by docker ps -a -q command)
$ docker system prune -a (deletes all inactive containers and unsused images)
$ docker logs -f <container- id> (To get log messages of the docker container that is started)

$ docker exec -it <container-id> bash (To get into docker container)

```
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
restapp-image                 latest    deae8d25495f   23 hours ago    492MB
nataraz/boot-rest-app-image   latest    deae8d25495f   23 hours ago    492MB
docker-webapp-image           latest    e58fd615cabb   3 days ago      274MB
nataraz/docker-webapp         latest    e58fd615cabb   3 days ago      274MB
[ec2-user@ip-172-31-0-222 ~]$ docker ps -a
CONTAINER ID   IMAGE                 COMMAND                  CREATED         STATUS                     PORTS
                     NAMES
bab754bb7d4b   restapp-image         "java -jar springboo…"   23 hours ago    Up 23 hours                0.0.0.0:9090->404
1/tcp, :::9090->4041/tcp   admiring_pasteur
59ec8efcb2ad   docker-webapp-image   "catalina.sh run"        3 days ago      Up 3 days                  0.0.0.0:8080->808
0/tcp, :::8080->8080/tcp   unruffled_mclean
5d65b40a9fff   docker-webapp-image   "catalina.sh run"        3 days ago      Exited (130) 3 days ago
                     hardcore_vaughan
e8409b555fd5   docker-webapp-image   "catalina.sh run"        3 days ago      Exited (130) 3 days ago
                     sleepy_curran
[ec2-user@ip-172-31-0-222 ~]$ docker logs -f bab754bb7d4b

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.3.2)

2024-07-22T02:41:19.189Z  INFO 1 --- [BootRestApp] [           main] com.nt.BootRestAppApplication            : Starting
 BootRestAppApplication v0.0.1-SNAPSHOT using Java 17.0.2 with PID 1 (/usr/app/springboot-rest-app.jar started by root i
```

```
[ec2-user@ip-172-31-0-222 ~]$ docker rm -f $(docker ps -a -q)
bab754bb7d4b
59ec8efcb2ad
5d65b40a9fff
e8409b555fd5
[ec2-user@ip-172-31-0-222 ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[ec2-user@ip-172-31-0-222 ~]$
```

```
[ec2-user@ip-172-31-0-222 ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[ec2-user@ip-172-31-0-222 ~]$ docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: docker-webapp-image:latest
untagged: nataraz/docker-webapp:latest
untagged: nataraz/docker-webapp@sha256:19e09091e10aeda0b4811ee9beafd2cfe77fef2dead06a738141d754cd0c4cd5
deleted: sha256:e58fd615cabbdd164984ab5e4a1b9a7b21d3dac6163147273eca01c089193d87
untagged: restapp-image:latest
untagged: nataraz/boot-rest-app-image:latest
untagged: nataraz/boot-rest-app-image@sha256:42ba3ed24d5dfcdf2a0abfb0e2c263f816f2595d28a2b365f5bd4db6a8888662
deleted: sha256:deae8d25495fc493eda2e7e779c4ebc49cc50d0746e77ffdd7dd4ce6ac9686a0

Deleted build cache objects:
vnexzrfy9d9s06hl30yvxnzt4
xms9qda64cqzl2b51ysre3ci0
0v2lb7a8ou67wahdh7jsc03jw
tlg9mhuhbr08b65kytyvaninp
75u11i8fy3gu6jx7sdz97zn0m
```



```
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
[ec2-user@ip-172-31-0-222 ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[ec2-user@ip-172-31-0-222 ~]$
```

DOCKER NETWORK
==========

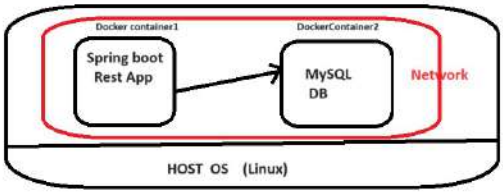
=> Every Project is combination of front end , backend and Database

eg:: front end ======> Back end ========> DB
(angular/reactjs) (java,.net,python,node js) (oracle,mysql,MongoDB)

=> Docker network is all about get communication b/w docker containers

=> Docker network is useful to provide isolated network for docker containers i.e using docker network that containers of the same network can participate in the communication

eg::



AWS Ec2 instance (Linux VM)

=> By default , the docker tool will give 3 networks

a) none
b) host
c) bridge

```
[ec2-user@ip-172-31-0-222 ~]$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
a03fca57a701   bridge    bridge    local
260644dbce7d   host      host      local
2cd34d0aeb3e   none      null      local
```

In docker , we have 5 networks drivers

a) bridge ----> it is the default network driver
b) host
c) None
d) OverLay
e) Macvlan

The network driver acts as the bridge b/w docker containers , to make the communication b/w docker containers happening effectively

Bridge

=> It is default network driver
=> It useful , when we are running standalone docker container
=> It assigns separate IP Address for every docker container

note:: The container with out having dependency with other container is called standalone docker container

HOST

==> same as Bridge but separate IP address will not be given for docker container i.e docker container runs in the same IP address where of HOST machine

==> This container is also veryful for standalone docker containers

None

=> None means No network will be provided by Docker containers

Overlay

=> This network driver is used for orchestration (scaling for docker containers)
=> Docker swaram will use this Overlay network

Macvlan

=> This network driver assings separate MAC address for a container i.e makes the communication easy
=> By getting MAC address , the docker container becomes the physical Container

Commands

To get all docker networks

$ docker network ls

# To create docker network
$ docker networks create nat-network

# To remove the docker network
$ docker network rm <network-id>

```
[ec2-user@ip-172-31-0-222 ~]$ docker network create nat-work
40e1cc1514f15e88ec716254f956a39208143925e1f97e7f6f4734d67980687
[ec2-user@ip-172-31-0-222 ~]$ docker network ls -a
unknown shorthand flag: 'a' in -a
See 'docker network ls --help'.
[ec2-user@ip-172-31-0-222 ~]$ docker network ls
NETWORK ID     NAME       DRIVER    SCOPE
a03fca57a701   bridge     bridge    local
260644dbce7d   host       host      local
40e1cc1514f1   nat-work   bridge    local
2cd34d0aeb3e   none       null      local
[ec2-user@ip-172-31-0-222 ~]$
```

To inspect the docker network

```
[ec2-user@ip-172-31-0-222 ~]$ docker inspect nat-work
[
    {
        "Name": "nat-work",
        "Id": "40e1cc1514f15e88ec716254f956a39208143925e1f97e7f6f4734d679806875",
        "Created": "2024-07-23T02:33:36.851595285Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
```

```
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

=>Simple demo of docker containers communication by keeping them in the same docker network using the support nginx server (web server)

# pull nginx docker image from docker hub
$ docker pull nginx

# create docker containers for nginx docker image by keeping the in the same docker network whose name " nat-work"
$ docker run - -name webserver1 -d --network nat-work -p 8080:80 nginx
$ docker run - -name webserver2 -d --network nat-work -p 9090:80 nginx

```
[ec2-user@ip-172-31-0-222 ~]$ docker  pull  nginx
Using default tag: latest
latest: Pulling from library/nginx
f11c1adaa26e: Pull complete
c6b156574604: Pull complete
ea5d7144c337: Pull complete
1bbcb9df2c93: Pull complete
537a6cfe3404: Pull complete
767bff2cc03e: Pull complete
adc73cb74f25: Pull complete
Digest: sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
[ec2-user@ip-172-31-0-222 ~]$ docker imaages
docker: 'imaages' is not a docker command.
See 'docker --help'
[ec2-user@ip-172-31-0-222 ~]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    ffffc90d343    4 weeks ago   188MB
[ec2-user@ip-172-31-0-222 ~]$ docker run --name webserver1 -d  --network nat-work -p 8080:80 nginx
d0e076d545a744c8014d228364605120feb7a9813cb911ca2ac89cde49be2d2a
[ec2-user@ip-172-31-0-222 ~]$ docker run --name webserver2 -d  --network nat-work -p 9090:80 nginx
1d424c94b511e4a2eb1c55892b278001c3d45a4c0/ae1db1e6eb//12a45432e9
[ec2-user@ip-172-31-0-222 ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
   NAMES
1d424c94b511   nginx     "/docker-entrypoint.…"   10 seconds ago   Up 9 seconds    0.0.0.0:9090->80/tcp, :::9090->80/tcp
   webserver2
d0e076d545a7   nginx     "/docker-entrypoint.…"   28 seconds ago   Up 27 seconds   0.0.0.0:8080->80/tcp, :::8080->80/tcp
   webserver1
```

Talking: NareshIT

inspect the network to get IP addresses of the docker containers that are linked to a docker network

$ docker network inspect nat-work

```
    "Containers": {
rofile      141 Bytes  2023-01-28 00:00  ec2-user  ec2-user  -rw-r--r--.7712a45432e9": {
            "Name": "webserver2",
            "EndpointID": "4665d41409340a08bb524d3c3ce14928d06867325f5de5c8ad103066fca7e681",
            "MacAddress": "02:42:ac:12:00:03",
            "IPv4Address": "172.18.0.3/16",
            "IPv6Address": ""
        },
        "d0e076d545a744c8014d228364605120feb7a9813cb911ca2ac89cde49be2d2a": {
            "Name": "webserver1",
            "EndpointID": "80e8ce4891d2d4021476e1af14254d1cc32f5d9c574fa5e88c0fe9244bd1ba58",
            "MacAddress": "02:42:ac:12:00:02",
            "IPv4Address": "172.18.0.2/16",
            "IPv6Address": ""
        }
```

nginx webserver1 ip address :: 172.18.0.2
nginx webserver1 ip address :: 172.18.0.3

to
Connect web server1 and ping web server2

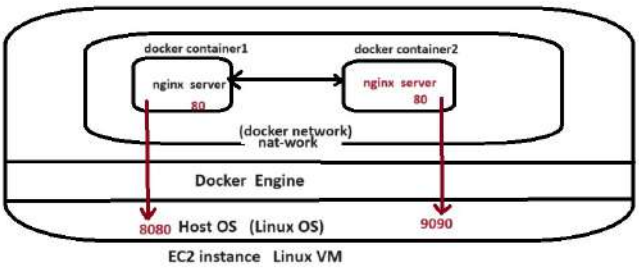
$ docker exec -it webserver1 /bin/bash

update all packages
#apt-get update

install ping command
apt-get install iputils-ping

ping 172.18.0.3
..
.. messages
..

=>ctrl+c for stopping this ping
exit (to come back host Linux prompt)



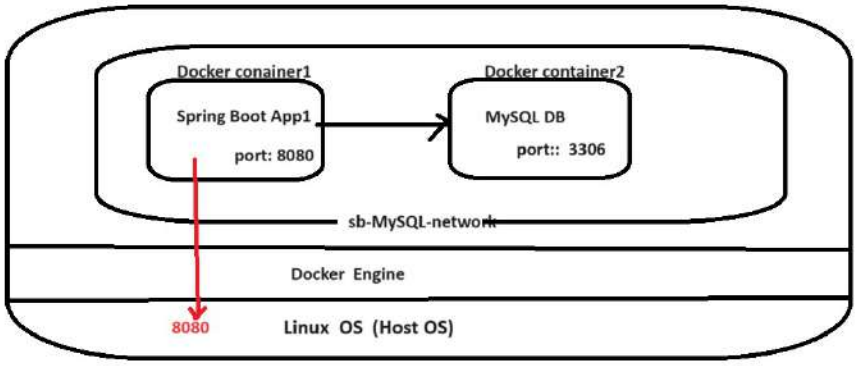
To inspect the docker network and to get ip addresses

$ docker network inspect nat-work

web server1 (nginx server) ip address:: 172.18.0.2
web server2 (nginx server) ip address:: 172.18.0.3

Dockerizing Spring Boot App interacting with MySQL Database

==> For this we need to take same docker network to get communication between Spring Boot app Docker container and MySQL DB Docker Container



Aws Ec2 Instance Linux VM

=>Since we want to allow the enduser to access the MySQL DB only through spring boot App , not directly So there is no need of performing port mapping for the MySQL container that is running as the Docker container But port mapping is mandatory for the spring boot App container

Procedure for implemenation

step1) keep Spring boot MVC App with MySQL App ready
(take any old spring boot MVC App and change the properties as required for MySQL interaction)

step2) Test the above application once locally

step3) add jar file name in pom.xml using <finalName> tag

```
<finalName>Sb-Mysql-App</finalName>   (under <build> tag)
```

step4) add file Dockerfile to the Project root folder

keeping the <packaging> as the war file in pom.xml

Dockerfile

```
FROM openjdk:17
EXPOSE 4041
COPY target/Sb-MySQL-App.war Sb-MySQL-App.war
ENTRYPOINT ["java","-jar","/Sb-MySQL-App.war" ]
```

note: if we pack the maven web application as the jar file then it will not pack the .jsp files
note: if we pack the maven web application as the war file then it will also pack the .jsp files

step5) build the app using maven "clean","package" goals to get the war file in target folder

Right click on the Project ---->run as ---> maven build ... ---> goals :: clean package

step6) push the code to GIT Repository

Right click on the Project ----> team --->share Project ----> ....

step7) PULL the Project Linux VM of AWS ec2 instance using mobaxterm

$ git pull

$ cd BootMVCPro-MiniProject-CURDOperations

step8) pull MySQL docker image from docker registry

$docker pull mysql:8.4

step8) create the MySQL docker container in deatched mode

```
[ec2-user@ip-172-31-88-108 BootMVCPro-MiniProject-CURDOperations]$ docker run --name mysqldb --network sb-mysql-network
 -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=jrtp701db -d mysql:8.4
```

$ docker ps -a

$ docker logs -f <container id>

```
mysql> show databases
    -> ;
+--------------------+
| Database           |
```

step9) Get into MySQL db Docker container and check the db availability

$ docker exec -it <containerid > bash

connect MySQL DB s/w
mysql -u root -p
password :: root

mysql> show database;

```
+--------------------+
| information_schema |
| jrtp701db          |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)
```

step10) make sure that the MySQL container name is specified in the jdbc url of the application.properties file

```
spring.application.name=BootMVCProj08-MiniProject-CURDOperations

# Data Source configuration
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://mysqldb:3306/jrtp701db
spring.datasource.username=root
spring.datasource.password=root      (docker container name)

#Embedded server port
server.port=4041

#  JPA properties
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update

#View Reoslver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

step11) build the spring boot MVC Application using maven commands

$ mvn clean package note:: Remove @SpringBootTest class from src/test/java folder

step12) create docker image representing our spring boot app

$ docker build -t sb-mysql-img .

step13) create docker container for the above img by keeping same network of MySQL container

$ docker run --network sb-mysql-network --name sb-mysql-app -p 4041:4041 -d sb-mysql-img

$ docker ps -a

step14) make sure that both spring boot app and MySQL contajners are placed in the same docker network

$ docker inspect sb-mysql-network


```
"Containers": {
    "7275bd3e029e00f6f7ddb26c40de5918387c130d5edc15e1b24b648e95d7e7b1": {
        "Name": "sb-mysql-app",
        "EndpointID": "0bec56502ddde7b86bb14e5c2efd972f2f3b6a97c6e49ab32cb3372775034fc7",
        "MacAddress": "02:42:ac:13:00:03",
        "IPv4Address": "172.19.0.3/16",
        "IPv6Address": ""
    },
    "a9245e7fcf6aa8cc724bbb2639394309dacb3648787dcd5ea0f6d227fdedd226": {
        "Name": "mysqldb",
        "EndpointID": "b3507e131b9b65938cc3bb9a556daa2471421b515380ea35255ff9de6021b158",
        "MacAddress": "02:42:ac:13:00:02",
        "IPv4Address": "172.19.0.2/16",
        "IPv6Address": ""
    }
},
```


step15) add 4041 port number to aws ec2 instance security group inbound rules

step16) access the spring boot App from the browser

http://18.212.49.140:4041/

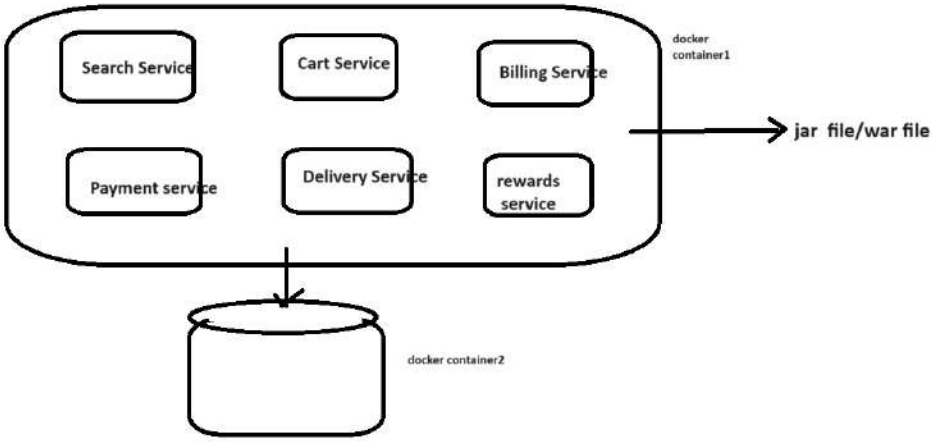
Docker Compose

Need of Docker Compose

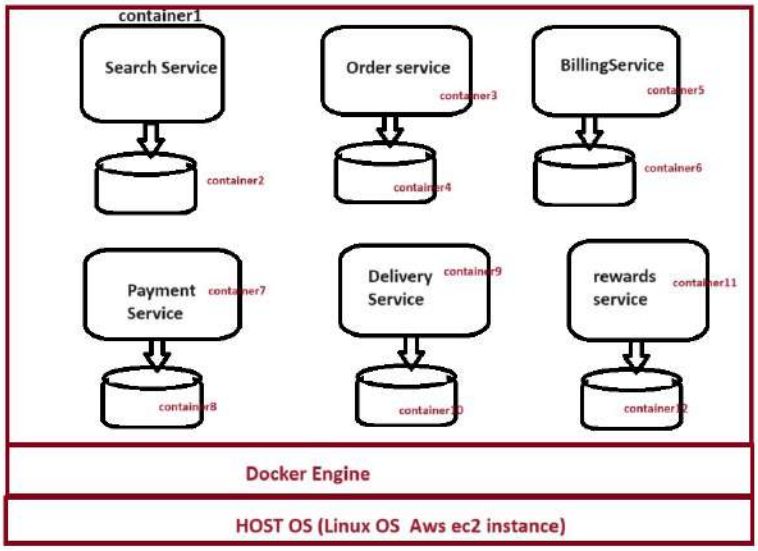
=> Monolith architecture App means single app that will be having all the functionalities together packed in a jar file or war file

=> Microservice architecture App means collection of apis where each api will be developed as the separate project or App

E-commerce App in Monolith Architecture



E- Commerice App in MicroService Archictecture



=> Here we are assuming every MS(MicroService) is using separate DB s/w So for DB of each microservice we need to create separate docker container

=>if all McroSErvcies(MS) are using the same DB then we can manage with one Docker container of DB

=> In MicroService architecture App /project , every API /MicroService should be container as separate docker container

=> So creating multiple docker containers by remembering their order of creation is difficulr process/taks , To solve this problem use docker compose

Docker compose

=>Docker compose is a tool (extension of docker tool) which is used to manage multi docker container based applications,

=> very useful in microservice architecture based app development

=>While working with docker compose , we give information about multiple containers ,docker network and other details using one yml file (default name is docker-compose.yml)

=> Docker compose tool needs separate installation though it is extension of docker

=> any <filename>.yml can be taken as the docker compose file , the default file name is docker-compose.yml

=> docker compose yml file contains the following information

version:

services:

network:

volumes:

docker compose commands

To create and up the docker container from docker compose file

$docker-compose up (takes default docker compose file docker-compose.yml)

$docker-compose -f <filename> up (takes the given file name as the docker compose file)

To list out containers that are created by docker compose tool

$docker-compose ps

$ docker-compose ps -a

To list out images manged by docker compose tool

$ docker-compose images

To stop and remove docker containers created by docker compose tool

$ docker-compose down

DOCEKR COMPOSE SETUP

==========

To download docker compose tool

$ sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

Give execute permission to docker-compose tool

$ sudo chmod +x /usr/local/bin/docker-compose

To check wheather docker compose is installed or not

$ docker-compose --version

Spring boot with MySQL Application development and execution using Docker compose tool

==========

step1) keep MySQL 8.4 and spring boot with MySQL app related docker images ready

$ docker pull mysql:8.4

$ cd cd JRTP701Repo-EclipseApps/BootMVCPro-MiniProject-CURDOperations/

$ docker build -t sb-mysql-img .

step2) prepare docker-compose.yml file

```
version: "3"
services:

 application:
  image: sb-mysql-img

  depends_on:
     mysqldb:
       condition: service_healthy        (our sprjng boot app will not be started until MySQL db service health is OK (started effectively)

  ports:
   - "4041:4041"
  networks:
   - sb-mysql-network

 mysqldb:
  healthcheck:
     test: ["CMD-SHELL", "mysqladmin ping -h localhost"]
     interval: 10s                                              tries to keep the MySQL DB up and ready
     timeout: 5s
     retries: 3
  image: mysql:8.4
  environment:
   - MYSQL_ROOT_PASSWORD=root
   - MYSQL_DATABASE=jrtp701db

  networks:
   - sb-mysql-network
```

```
networks:
  sb-mysql-network:
```

step5) run docker compose command to create the docker containers and network

```
$docker-compose up -d        ( creates the docker containers in detached mode)
$ docker-compose up
```

step6) Access the application

Mini Project-- Home Page

Show Report

step7) Execute the other commands

```
# To delete containers and network created by docker compose tool
$docker-compose down

# To see all the docker container created by the docker compose
$ docker-compose ps
$ docker-compose ps -a
```

Docker Volumes

=> Docker containers are stateless by default that means we can not persist the data given by Docker container Apps across the multiple creations and executions of the docker container

note:: Once the docker container is removed we will loose data that is stored in the container by the app execution but in real pratices we should not loose App data enen the docker container is removed

Problem praticals

$ docker-compose up (creates the docker containers)

=> Access the app and provide data (http://<host ip address>:4041)

$ docker-compose down (removes the docker containers)

=> Access the app , but we will find no previous data (This makes docker conainers as the stateless containers)

=> To solve the above problem take the support of docker volumes which needs to be specified in the docker-compose.yml file along with other details like services ,network information

(The App Data the is stored in the DB should available across the creation and removal of the docker containers)

=> Docker Volumes are given to makes the Docker containers created in the Docker compose env.. as the statefull

Stateless:: not remembering the data across the multiple executions
Stateful:: remembering the data across the multiple executions

=> Docker Volumes are given to store Docker container based App generated Data across the multiple creations and removals of the docker containers

3 types of Docker Volumes

a) Anonymous Volumes (Volume with out name) (bad)
b) Named Volumes (Volume with name) (best)
— These two types of Volumes stores App Data in Docker container itself

c) Bind Mounts (bad) — This Concept stores the App data in HOST machine (In our case that is Ec2 instance Linux VM)

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you store volumes on remote hosts or cloud providers, encrypt the contents of volumes, or add other functionality.
- New volumes can have their content pre-populated by a container.
- Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windov hosts.

In addition, volumes are often a better choice than persisting data in a container's writable layer, because volume doesn't increase the size of the containers using it, and the volume's contents exist outside the lifecycle of a given container.

(i.e though docker container is removed , the docker volume content will not be removed)

a) Anonymous Volumes

=> These are the volumes with out names
=> they can be given in the docker-compose file as shown below

docker-compose.yml

```yaml
mysqldb:
  healthcheck:
       test: ["CMD-SHELL", "mysqladmin ping -h localhost"]
       interval: 10s
       timeout: 5s
       retries: 3
  image: mysql:8.4
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=jrtp701db

  networks:
    - sb-mysql-network
  volumes:
    - /var/lib/mysql    (anonymous Volume)
```

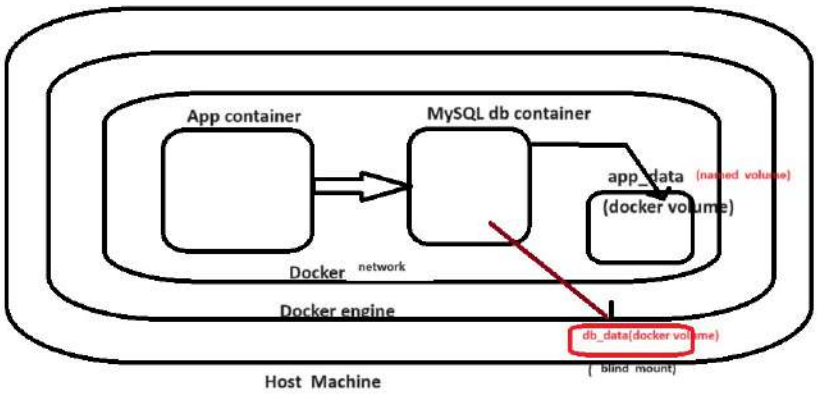
Why the Anyomous Volumnes are bad?

Ans) these volumes do not have any name ... so their access becomes bit complex and some times they inaccssiable or dangling volumes

What is dangling Volume?

Ans) The Volume that is created , but not associated with any docker container is called dangling volume

```
# To delete all the dangling volumes
$ docker volume rm $(docker volume ls -q -f dangling=true)

# To create docker volume
$ docker volume create <volume-name>

# To inspect docker volume
  $ docker volume inspect <voume-name>

# To delete docker volume
  $ docker volume rm <vol name>

# delete all docker volumes
  $docker system prune --volumes
```



Named Volumes

==========

=> This docker volume contains name
=> these are mostly used volumes in real-time
=> Both anomymous and named volumes allocates memory inside the docker container/engine but they are not going be part of docker life cycle i.e though docker containers are removed the docker volumes will not be removed

```yaml
mysqldb:
  healthcheck:
       test: ["CMD-SHELL", "mysqladmin ping -h localhost"]
       interval: 10s
       timeout: 5s
       retries: 3
  image: mysql:8.4
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=jrtp701db
  volumes:
    - app_data:/var/lib/mysql .
                              NAmed Voume Utilisation
  networks:
    - sb-mysql-network


networks:
  sb-mysql-network:
volumes:
  app_data:
  NAmed Voume creation
```

NAmed Volume praticals

==========

step1) update docker-compose.yml as shown above having named volumes

step2) create the container

$ docker-compose up

step3) access the app and insert data

step4) remove the docker conainer
docker-compose down

note:: In latest versions of docker compose
there is no need of placing "version" in the docker compose file

step5) Re create docker containers
& access the application

$docker-compose up

Confirming the named Voume creation
==========

get Container infomation

```
[ec2-user@ip-172-31-25-179 BootMVCPro-MiniProject-CURDOperations]$ docker-compose ps -a
NAME                                                IMAGE          COMMAND                  SERVICE       CREATED
    STATUS                  PORTS
bootmvcpro-miniproject-curdoperations-application-1   sb-mysql-img   "java -jar /Sb-MySQL…"   application   2 minutes ag
o   Up 2 minutes            0.0.0.0:4041->4041/tcp, :::4041->4041/tcp
bootmvcpro-miniproject-curdoperations-mysqldb-1       mysql:8.4      "docker-entrypoint.s…"   mysqldb       2 minutes ag
```

Get into the container

```
[ec2-user@ip-172-31-25-179 BootMVCPro-MiniProject-CURDOperations]$ docker exec -it bootmvcpro-miniproject-curdoperations
-mysqldb-1 bash
```

#Execute the following commands

```
bash-5.1# pwd
/
bash-5.1# cd var/lib
bash-5.1# ll
bash: ll: command not found
bash-5.1# ls -l
total 20
drwxr-xr-x. 2 root  root     34 Jul  8 17:17 alternatives
drwxr-xr-x. 1 root  root     80 Jul 23 00:06 dnf
drwxr-xr-x. 2 root  root      6 Jan 10  2022 games
drwxr-xr-x. 2 root  root      6 Jan 10  2022 misc
drwxrwxrwt. 8 mysql mysql 16384 Aug  2 03:00 mysql
drwxr-x---. 2 mysql mysql     6 Jul 12 20:50 mysql-files
drwxr-x---. 2 mysql mysql     6 Jul 12 20:50 mysql-keyring
drwxr-xr-x. 1 root  root     74 Apr  9 20:07 rpm
drwxr-xr-x. 2 root  root      6 Jan 10  2022 rpm-state
drwxr-xr-x. 3 root  root     17 Jul  8 17:17 selinux
-rw-r--r--. 1 root  root    102 May  2 15:31 supportinfo
bash-5.1# cd mysql
bash-5.1# ls -l
```

```
bash-5.1# cd jrtp701db
bash-5.1# ls -l
total 224
-rw-r-----. 1 mysql mysql 114688 Aug  2 02:50 emp.ibd
-rw-r-----. 1 mysql mysql 114688 Aug  2 02:50 empno_seq1.ibd
```

Docker Swaram

==========

=>Docker is containerization tool i.e it is capable of packing code +env..

=> Docker swarm is Orchestration Tool (To create and manage the Docker containers)

note:: Docker swarm is not good orchestration tool becoz it does not have auto scaling feature
i.e we need to decide the no.of instances/replicas of docker containers manually , docker swarm can do that automatically

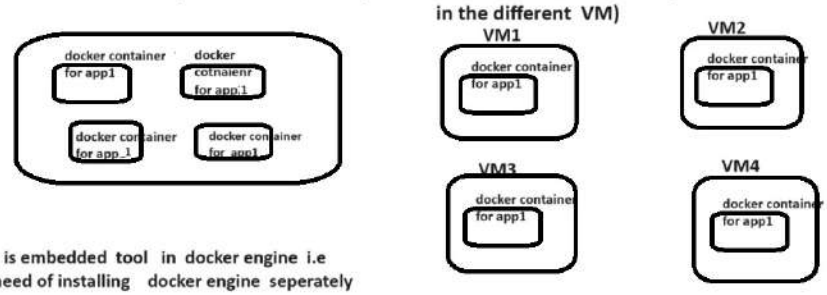
=> Other orchestration tools
kubernates (k8s) (best) ---> It has got auto scaling feature
open shift
Mesos
Rancher and etc..

Docker swarm

==========

=> It is a Container orchestration tool or software
=> The English meaning or orchestration is managing the processes
(docker swarm actually manages the docker containers)

=> Docker swarm is used to setup docker cluster (set of docker containers running in same VM or in the different VM)

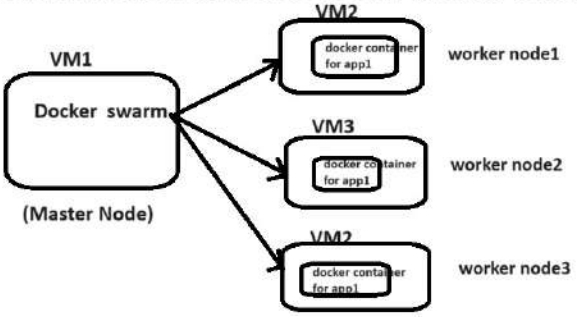


Cluster means set/group of same items or similar items
eg:: IT cluster , Text tile shops cluster

=> Docker swarm is embedded tool in docker engine i.e there is no need of installing docker engine seperately

=> In docker swarm Cluster means group of servers or docker containers running in the same VM or in the different VMs

=> Using Docker swarm cluster concept , we need to setup Master and Worker nodes



=>Master Node will schedule the tasks (containers) and manages the nodes and node failure.. It simply connects to worker nodes gives instructions to worker nodes to create manage desired no.of docker containers

=> Worker nodes perform the actions as per master node directions
(Docker containers creation and execution)

Docker swarm features

==========

=> Cluster management
=> Decentrailized Design
=> Declarative service model
=> Scaling
=> Multi Host network
=> Serivce Discovery
=> Load Balancing
=> Secure by default
=> Rolling updates

note:: For orchestration , the real companies prefer using unbuntu (flavor of Linux) OS Ec2 Instance
(So far we have used amazon Linux)

Setup

========

step1) create 3 Ec2 Instances having Ubuntu as the OS (1 master node and 2 worker nodes)

=> aws ec2 instance page -----> lanuch instance ----> name:: Docker Swarm ----> select ubuntu OS
---> select t2.medium ----> give the old security pem file ----> create new security group ---> no.of instances 3 ---> ... ...

=> Rename the instances 1) Master --Docker swarm VM
2) Node1 --Docker swarm VM
3) Node2 --Docker swarm VM

☐	Master -docke...	i-091f083fefdc489f7	Running	t2.medium	Initializing	View alarms +
☐	Node1-docker ...	i-021d0bdbc30453f8b	Running	t2.medium	Initializing	View alarms +
☑	Node2-doc...	i-0d3536bcc0019ff59	Running	t2.medium	Initializing	View alarms +

step2) Enable 2377 port number for docker swarm Cluster communication

Inbound rules Info

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info	
sgr-0c2eb0ab202474544	Custom TCP	TCP	2377	Cus... 0.0.0.0/0		Delete
sgr-078acd154e7dd2b71	SSH	TCP	22	Cus... 0.0.0.0/0		Delete

step3) open 3 Mobaxterm windows 1 for master node and 2 for worker nodes

Connect 1 mobxterm window to 1 VM having username as the ubuntu
Connect 2 mobxterm windows to 2 VMs having username as the ubuntu

step4) keep docker software ready in all the 3 windows

$ sudo apt update -y

$ curl -fsSL https://get.docker.com -o get-docker.sh

$ sudo sh get-docker.sh

step4) Perofrm the following operations from the Master window to activate the docker swarm service

$ sudo docker swarm init --advertise-addr 172.31.25.10

```
ubuntu@ip-172-31-25-10:~$ sudo docker swarm init  --advertise-addr 172.31.25.10
Swarm initialized: current node (eh6pgc1k1pe1d4oipt531tovv) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1at7s5di3pnlthcuzab242pvlxxpewi17irs0onqr7cd67y13l-a7cljktvflamsyprltddy48go 172.
31.25.10:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

step5) Make other two windows as the worker nodes

From Node1 window

```
Last login: Mon Aug  5 02:06:22 2024 from 120.138.12.112
ubuntu@ip-172-31-31-248:~$ sudo  docker swarm join --token SWMTKN-1-1at7s5di3pnlthcuzab242pvlxxpewi17irs0onqr7cd67y13l-a
7cljktvflamsyprltddy48go 172.31.25.10:2377
```


From node2 window
==========

```
Last login: Mon Aug  5 02:06:11 2024 from 120.138.12.112
ubuntu@ip-172-31-30-195:~$ sudo  docker swarm join --token SWMTKN-1-1at7s5di3pnlthcuzab242pvlxxpewi17irs0onqr7cd67y13l-a
7cljktvflamsyprltddy48go 172.31.25.10:2377
```


what is the docker swarm manager quarm?

Ans) we can take multiple docker swarm managers .. to get high availability
It always recomanded to take odd numbers of docker swarm managers for high availability

Formulae :: (n-1)/2

if we take 2 servers as the masters
2-1/2 = 0.5 (It can be master server)
3-1/2 = 1 (It can be leader when main leader is down)

note:: In Docker swarm we need to our app as the service

note:: In Docker swarm we need to run app as the service

Docker swarm service
==========
=>Docker swarm service is a collection of one or more docker containers of the docker image

Two docker swarm services
a) Replica (default node)
b) Global

step7) get java web application docker image

$ sudo docker pull nataraz/docker-webapp

step8) create docker service representing the above docker image

```
ubuntu@ip-172-31-25-10:~$ sudo docker service  create --name swarm-docker-webapp -p4041:4041 nataraz/docker-webapp
m3e21x8r0bcrymnxhv61cs659
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service m3e21x8r0bcrymnxhv61cs659 converged
ubuntu@ip-172-31-25-10:~$
```


step9) Add 4041 port number Ec2 instance security group as the inbound rule

step10) Access the application

Docker Testing web application

show date and time

To scale up/down no.of containers in the docker service

$ sudo docker service scale <service name>= <no.of replicas>

eg:: $ sudo docker service scale swarm-docker-webapp=10

# To get details about specific docker service
$ sudo docker service ps swarm-docker-webapp

# To inspect about specific docker service
$ sudo docker service --pretty swarm-docker-webapp

This kind of cluster of docker containers are required
for large apps like gmail ,google ,amazon and etc..
for giving high availability for them

To get all docker swarm services

$ sudo docker service ls

To remove one from docker cluster
--> from node VM
$ sudo docker swarm leave

```
ubuntu@ip-172-31-30-195:~$ sudo docker swarm leave
Node left the swarm.
```

To remove docker swarm service

$ sudo docker service rm <service id>