Docker

\_\_\_\_\_\_\_\_\_\_\_\_\_

-> Resolves Compatibility problem using containerization

-> Docker will help us to install our software on different server without worrying about compatibility issue

-> To deploy same application on multiple servers for load balancing we need not manually install technology stack (like Angular 18v, java 17V etc) manually one by one if docker is not used

-> Software upgrades can be done easily on multiple machines using docker

Note:

✔ Docker Engine acts as a bridge between the container and the host OS

✔ Containers don’t have their own kernel, they use the host’s kernel

✔ Docker Engine manages containers but does not create them—it runs & controls them

Note:

Linux VM--->Install Docker Engine--->Docker Engine acts as bridge between Linux Kernel and docker Container-->Provides isolated enviroment to run Docker Image.

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

#

# docker container

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

#

# dOCKER ENGINE

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

#

# lINUX vm

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

Docker Architecture:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Docker File: Here we will provide Intructions to create docker image

Docker Image: Application Package with dependencies of it

Docker Registry: Its a hub to store docker images

Docker Container: Isolated environment to run docker image

What is containerization?

containers package an application along with its dependencies (libraries, configuration files, etc.), ensuring that it runs consistently across different computing environments.

######################### ######################### #################### @@@@@@@@@@@@@@@@@

Dockerfile %%%%%%%%

Container1

(Contains instructions ------> Docker Image ------------> Docker hub/registry ----------> %%%%%%%%%

to download dependencies (build) (Application Code + (Store) (Collection of images) Container2

Dependencies to run that) %%%%%%%%%%

)

########################## ########################## ##################### @@@@@@@@@@@@@@@@@

Linux Server

Installing Docker:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For Amazon Linux use the following commands

Install Docker In Amazon Linux VM

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

sudo yum update -y

sudo yum install docker -y

sudo service docker start

sudo usermod -aG docker ec2-user

exit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For Ubuntu use the following commands

Install Docker In Ubuntu VM

sudo apt update

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

sudo usermod -aG docker ubuntu

exit

Verify docker installation

Use this command to check version of docker installed

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker -v

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

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

For practise pull the sample image from docker hub repository of pankaj sir academy: docker pull psait/pankajsiracademy:latest

For Practise pull docker official image: docker pull hello-world

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

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

Important Docker Commands

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

docker pull : download docker image from hub

docker pull [image-name]

docker run : run docker image - this will create container (Isolated Enviroment to run docker image- This is not a OSs)

docker run [image-name / image-id]

docker ps :To display docker containers that are running

docker ps

display stopped containers:

docker ps -a

docker stop :To Stop docker container

docker stop [container-id]

docker start : Start docker container

docker start [container-id]

docker rm : will remove stopped docker container

docker rm [contianer-id]

docker rmi : Will Remove docker image

docker rmi [image-name / image-id]

To remove all stopped containers and un-used docker images we can use below command

docker system prune -a

How to create Docker Image and run that to access from browser?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

run Docker Image from docker hub using the command

Docker run [image-name] - This command when executed we will not be able to access our container in browser because it requires port mapping.

As Container is running inside linux VM. We will have to map linux vm (host port) to container(Container port) this is called as port mapping. To do this perform the following

Docker run -p host-port:container-port [image-name]

Example: docker run -p 9090:9090 [image-name]

Example: docker run -d -p 9090:9090 [image-name] will run the cintainer in background

Now you can access our application using the url http://public-ip:host-post(linux-vm)

Note

1. Enable Inbound rule in security group custom ip IPv4 anywhere with host port number.

2. If you run Multiple Containers in same linux vm then host port number should be different for every container

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Install jekins using docker image name - docker run -d -p 8080:8080 jenkins/jenkins

Note Enable Inbound rule in security group custom ip 8080 IPv4 anywhere

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If Docker not used then we have to do the following to install jekins in linux vm

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Step - 1 : Create Linux VM with Ubuntu, use t2.micro as jekins is heavy software & requires good configuration server

Create Ubuntu VM using AWS EC2 (t2.medium)

Enable 8080 Port Number in Security Group Inbound Rules

Connect to VM using ssh cleint

Step-2 : Install Jdk as jekins is java application and requires jdk to run jekins

sudo apt update

sudo apt install fontconfig openjdk-17-jre

java -version

Step-3 : Install Jenkins

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install jenkins

Step-4 : Start Jenkins application using the commands given below

sudo systemctl enable jenkins

sudo systemctl start jenkins

Step-5 : Verify Jenkins

sudo systemctl status jenkins

Step-6 : Open jenkins server in browser using VM public ip. Jekins runs on port 8080 by default

http://public-ip-address:8080/

Step-7 : Copy jenkins admin pwd

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Step-8 : Create Admin Account & Install Required Plugins in Jenkins to start your CI/CD pipeline learning journey

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Note Docker will simply our journey to get required softwares in server through single line -

Example download Sonarqube using docker image. If not use linux command to do the same.

docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube:lts-community

Example download nexus using docker image, If not use linux command to do the same.

docker run -d -p 8081:8081 --name nexus sonatype/nexus3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How to create docker file? Create Docker Image? push that to docker hub?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In order to build a docker file as a developer you should know when to use the following keywords.Dockerfile instructions (FROM, CMD, ADD, etc.) are case-sensitive and must be written in uppercase.

1. FROM

\_\_\_\_\_\_\_\_\_\_\_

->The FROM instruction in a Dockerfile specifies the base image for your container or it will be used to specify what softwares should be downloaded to run our app. It is always the first instruction in a Dockerfile because it defines the environment where your application will run

FROM <image>:<tag>

<image> → Name of the base image (e.g., ubuntu, node, python)

<tag> → (Optional) Specifies the image version

Example:

FROM python:3.9

FROM openjdk:17

FROM tomcat:9.0

2. MAINTAINER

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-> The MAINTAINER instruction was used in older Docker versions to specify the author of the Dockerfile.

Example of old version - MAINTAINER Your Name <your-email@example.com> (This is deprecated)

Example for new version - LABEL maintainer="Your Name <your-email@example.com>"

3. RUN

\_\_\_\_\_\_\_\_\_\_

-> The RUN instruction in a Dockerfile is used to execute commands during the image build process.

Example :

RUN 'git clone <repo-url>'

RUN 'mvn clean package'

Note: If you specify multiple RUN instructions in Dockerfile, then those will execute in sequential manner.

4. CMD

\_\_\_\_\_\_\_\_\_\_\_\_

-> The CMD instruction in a Dockerfile specifies the default command that runs when the container starts

CMD "java -jar myapp.jar"

Note: when we write multiple CMD instructions in dockerfile, docker will execute only last CMD instruction only.

Let us create our first docker file:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Step 1: vi dockerfile

FROM openjdk:17

MAINTAINER Pankaj

RUN echo 'run-1'

RUN echo 'run-2

CMD echo 'cmd-1'

CMD echo 'cmd-2'

Step 2: create docker image using the command

-> docker build -t pankajsiracademy/image1 .

we have used dot (.) in above command to mention that dockerfile in present in same working directory

Step 3: To check image created use the command

--> docker image

Step 4: to delete the image

--> docker rmi image-id

Step 5: You can again create the image

--> -> docker build -t pankajsiracademy/image1 .

Step 6: Run your docker image

--> docker run [image-id]

Note:

1. Only last CMD instruction will run

2. docker run [image-id] echo 'hello world', you will notice now hello world will execute and not the last CMD of docker file

To over come the above said instruction we can use

5. ENTRYPOINT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-> Alternative command for CMD but the advantage is we cannot override this command

Example-1 : CMD "java -jar app.jar"

Example-2 : ENTRYPOINT ["java", "-jar", "app.jar"]

6. COPY

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

--> COPY instruction will copy the files from source to destination.

--> It is used to copy application code from host machine to container machine.

--> Here Source means "HOST Machine" and Destination means "Container machine"

Example : COPY target/your-app.jar /usr/app/

7. ADD

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

--> ADD instruction will copy the files from source to destination same as COPY, But in addition it can extract your compressed tar file

--> Here Source can be host machine

--> ADD cannot download from HTTP/S3 URLs—this is a misconception. (Be carefull in interviews)

Example:

ADD target/app.jar /usr/app/

ADD <http-url> /usr/app/ (Wrong)

8. WORKDIR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

--> The WORKDIR instruction sets the working directory (Like cd command )

--> If the directory does not exist, Docker will automatically create it

Example: WORKDIR /path/to/directory

COPY target/your-app.jar /usr/app/

WORKDIR /usr/app/

CMD "java -jar your-app.jar"

9. EXPOSE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-> EXPOSE command is used to specify application is running on which PORT number. Using this you cannot change the port number of application. You are just mentioning that our application is running on the port number 9090.

-> If our application is running on port number 8081 you are mentioning expose 9090 then it is wrong.

-> It is optional to use

Example : EXPOSE 9090

Now time for practicals guys, Let us do Dockerizing Spring boot application

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-> Spring Boot: framework which is used to develop enterprise based applications.

-> Spring Boot applications will be packaged as a jar file for deployment.

-> To run the jar file we will use command : java -jar <file-name.jar>

-> To run springboot application jar file we will use tomcat server as "embedded server".

-> By default spring boot application will run on port number 8080

Step 1: Create Docker File

FROM openjdk:17

COPY target/demo-app.jar /usr/app/

WORKDIR /usr/app/

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "demo-app.jar"]

Step 2: Note (Install maven and git in linux VM first)

1) Clone git repo in docker host machine (linux): git clone <http-url>

2) Point to project root folder: cd <app-name> and run to generate jar: mvn clean package

3) Create docker image and check that:

-> docker build -t psait/pankajsiracademy:<tag> .

(

Here

-> psait is username

-> repositoryname of docker hub

-> <tag> can

a. prod-v1 or prod-v2v2 for production

b. dev-v1 for development environment

c. test-v1 for testing environment

d. staging-v1 for staging environment

)

-> docker images

4) run docker container: docker run -d -p 8080:8080 --name psa psait/your-app

5)See the image is running or not: docker ps

6) docker logs <container-id>

5) Access application URL in browser: http://public-ip:8080/

Steps to push docker image to docker hub

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-> login into docker hub account from bash

docker login

Enter username: psait

Enter password

-> push docker image

docker push psait/pankajsiracademy:<tag>