**Docker**

1. Docker is a containerization software used to package applications and all its dependencies. With docker applications runs seamlessly in all Environments.

Docker is a software use to create docker images and run containerized apps. Docker is used by Developers, Administrator and others to build and run applications as containers.

app.war = application code

tomcat/java = dependencies

1. Docker Editions: available in
   1. Docker CE --> Comunity Edition --> Open Source (Free) it doesn’t work with redhat
   2. Docker EE --> Enterprise Edition --> Commercial
      1. DTR --> Docker Trusted Registry(Private Repo to keep docker images)
      2. UCP --> Universal Control Plane --> It's GUI for managing Docker Machines
2. Installation:
   1. O.S --> Cross **Platform** (Docker can be installed in any O.S)
      1. Linux -- (distribution) CentOS / ubuntu / redhat
      2. Windows --
      3. mac OS
   2. SaaS =
      1. DockerHub
      2. New Relic,
      3. GitHub,
      4. SonarCloud
   3. IaaS
      1. EC2
3. What is your experience in Open Source technology:
   1. SOFTWARES ( installed in operating system): java / tomcat / nexus / maven / sonar / git / github / jenkins / Docker-CE
   2. Linux OS distribution: ubuntu / centOS
4. Docker file --------image --------container
   1. **Dockerfile:** File that has a simple and descriptive set of instructions to create an image.
   2. docker image = A read-only snapshot of a container that is stored in Docker Hub or in a private repository; you use an image as a template for building containers. app + dependencies.

Docker image = application + dependencies

App.war + tomcat

Myapp.jar + java

App.ear + Jboss

* 1. **Container:** The standard unit where the application service is deployed or running. Docker Container--> It's a runtime instance of a docker image, where our application is running.

1. Explanation
   1. Code are managed by SCM = GitHub
   2. Packages (app.war) are uploaded and managed in Nexus
   3. Docker images are stored and/or managed in dockerhub/Registries
   4. engine/daemon --- That performs tasks and Instructions from docker CLI
   5. Docker Containers --- This is a running instance of docker images
2. Docker Repo/Registry. --> We can store and share docker images.
   1. Public Repo --> Docker hub : It is generally a public repo which contains all the open source software as a docker images. We can think of docker hub as play store for docker images.
   2. Private Repo (Amazon ECR, Nexus, JFrog, D.T.R(Docker Trusted Registory)) --> We can store and share the docker images with in our company network using private repos
3. What is docker hub? It's a public repository for docker images. You can think as play store for docker images.
   1. First Create Account in docker hub <https://hub.docker.com>
   2. In docker hub/registry we have repo and image is transfer there with docker push
   3. Deployment server **pull** image from registry
4. **Docker commands:**
   1. docker info = all info about docker processes
   2. docker --version
   3. docker create = create a docker container
   4. docker ps = list running containers
   5. docker ps -a = list all running containers and stopped/EXITED containers
   6. docker login -u mylandmarktech

**Docker image commands**

* 1. docker build = to build/create docker image
  2. docker images = list docker images
  3. docker rmi imageID/name:tag = delete an image
  4. docker rmi -f imageID/name:tag = force delete an image ( cant del a running container)
     1. docker rmi -f c8c2b947edda
     2. Cant stop a running docker so = docker stop imageName ( gotten from docker ps) then can remove it
  5. Docker pull
  6. Docker image inspect <imageid/Name>
  7. docker images -q = list image ids
     1. To forcefully delete all images

docker rmi -f  $(docker images -q)

**Docker container commands**

* 1. docker run = will create and start a container ( pull,create and start)
  2. docker push = push images to Registry / dockerhub
  3. docker start containerName= to restart a container that stopped running
  4. docker push = push images to Registry / dockerhub
  5. docker pull = deploy server pull images from registry
  6. docker scan
  7. docker stop | start = to stop or start a running container
  8. docker tag = assigns repository name to images
  9. docker rm -f containerName
  10. docker system/container/image prune -a = will remove all dangling container

docker system prune

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

An image that is not tagged to any repo is call Dangling images.

* 1. docker container ls =
  2. curl -L public IP:container port
     1. Curl -L 52.60.79.66:3000/ =
  3. l

**Explanation**

1. **Docker file**  --- list of Instructions used to create docker images. E.g of a simple dockerfile in a java project repo in github - <https://github.com/sonarlove/java-web-app>.
   1. Content = application codes (app.war) and softwares / dependencies

Example1 :

FROM tomcat:8.0.20-jre8

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

Example 2:

FROM jboss/wildfly

COPY target/\*.war /opt/jboss/wildfly/standalone/deployments/app.war

Example 3:

FROM nginx:alpine

COPY index.html /usr/share/nginx/html

* + 1. From = use for minimal based image ( from dockerhub) dependencies, eg - FROM tomcat:jdk17-temurin-focal
    2. Copy = file name ( web-app.war) found in location ( target dir) , tomcat /~ = {/usr/local/tomcat/ ( in dominion tomcat home dir)} = /opt/tomcat9), in tomcat deployment takes place in (webapps)
  1. Dockerfile Keywords / INTRUCTIONS :
     1. MAINTAINER = Author's details -
     2. FROM = Define minimal base images (tomcat)which contains required software + dependencies (java - jre8)
     3. COPY = This will copy files from the docker-host to the container when creating a docker image. Executes on host
     4. ADD = this will copy files from the docker-host to the container when creating a docker image. It can copy files from external sources(nexus) like https also . Similar to COPY except url .

ADD https://github.com/LandmakTechnology/k8s-ansible /usr/local/tomcat/webapps

* + 1. RUN = This will execute commands when creating a docker image, docker build [RUN instructions/commands are executed]

RUN mkdir -pv /opt/app

RUN touch /opt/app/test.java

RUN tar -xvzf /opt/app/tomcat-8.tar.gz

* + 1. CMD = This will execute commands when creating a docker container, It will execute the last CMD listed on the DockerFile. instructions are use to pass commands [docker run/ docker start]. execute entry point linux command
    2. ENTRYPOINT = this will execute commands when creating/starting a docker container. instructions are use to pass commands [docker run]
    3. WORKDIR
    4. ENV, environment variables
    5. EXPOSE - port
    6. USER ---
    7. VOLUME
    8. LABEL
    9. ARG
  1. what is difference b/w COPY and docker cp?
     1. docker cp = copy files and directories from docker-host to container and vice versa
     2. COPY is a keyword in the docker file use to copy files into the image at creation.
  2. What is the difference between CMD and ENTRYPOINT?

We can overwrite CMD but we cannot overwrite ENTRYPOINT

* 1. Not sure
     1. FROM centos
     2. RUN mkdir /app
     3. RUN touch /app/test.sh
     4. RUN echo mylandmarktech >> /app/test.sh
     5. COPY . .
     6. CMD ["echo" ,"From CMD DevOps 1st"]
     7. CMD ["echo" ,"From CMD DockerFile last"]

1. **Dockerfile keywords/instructions contd.**
   1. Maven BuildServer creating artifacts [jar/war/ear]
      1. Web applications ie war artifacts are deployed in Tomcat/JBoss servers
      2. standalone applications [jar] are deploy in boxes[VM] with java installed
      3. enterprise applications [ear] are deployed in JBoss/wildfly servers
   2. Maven BuildServer creating artifacts [jar/war/ear]
      1. Artifacts are stored securely in artifactory JFrog/Nexus [jar/war/ear]
      2. https://45.0.2.88:8081/repository/td-releases/app.war
   3. Docker BuildServer creating artifacts [docker images]
      1. To build a docker image a Dockerfile is required.
      2. Docker images generally contained application code + dependencies, softwares
   4. FROM tomcat:8.0.20-jre8
      1. ADD https://45.0.2.88:8081/repository/td-releases/app.war /usr/local/tomcat/webapps
      2. Containers are meant to be lightweight, to be isolated, having more than one process run in a container can cause issues ( like scaling problem)if an app is to be scaled up, we will have to scale both because they will be running on a single contain which can result to waste of resources. eg

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

ADD target/springapp.war /usr/local/tomcat/webapps

* 1. RUN =used to run commands while creating image. E.g creating an important file <touch readme.md>
  2. CMD = e.g to start tomcat <sh startup.sh>
     1. CMD sh catalina.sh start = shell form ( not recommended)
     2. CMD ["sh", "catalina.sh", "start"] = executable form
  3. ENTRYPOINT sh catalina.sh run
     1. ENTRYPOINT ["sh", "catalina.sh", "start"]
     2. ENTRYPOINT ["sh", "catalina.sh", "run"]
  4. Instruction execution
     1. docker build [RUN, FROM, ]
     2. docker run [CMD, ENTRYPOINT ]

1. shell form vs executable form:
   1. shell form: When using the shell form, the specified binary is executed with an invocation of the shell using /bin/sh -c, with shell form application runs as a child process and It can be killed unnoticed.app outage < sh catalina.sh>
      1. ENTRYPOINT sh catalina.sh start
      2. CMD sh catalina.sh start
   2. executable form in docker: When instruction is executed in exec form it calls the executable directly, and shell processing does not happen. When instruction is executed in shell form it calls /bin/sh -c <command> under the hood and normal shell processing happens.

The following snippet in Dockerfile-

* + 1. ENTRYPOINT ["sh","catalina.sh"]
    2. CMD ["sh","catalina.sh"]

Application runs as a parent process, It cannot be killed unnoticed

1. ENV, environment variables ( to rep a path) e.g
   1. It can rep/ set diff env for diff environment. Or if I want to deploy to diff environment
      1. DEV / CATALINA\_HOME /usr/local/tomcat
      2. UAT / CATALINA\_HOME /opt/tomcat
      3. PROD CATALINA\_HOME /app/tomcat
   2. From docker file:
      1. Example 1:

ENV CATALINA\_HOME /usr/local/tomcat

ENV PATH $CATALINA\_HOME/bin:$PATH

RUN mkdir -p “CATALINA\_HOME”

WORDIR $CATALINA\_HOME

* + 1. Example 2:

ENV JAVA\_HOME /USER/JAVA/OPENJDK-20

ENV PATH $JAVA\_HOME/bin:$PATH

1. ‘run’ in dockerfile = each ‘run’ instruction create a layer in the docker image, Each run instruction increases the size of the docker image. Which means- The more Layers the heavier the docker image so recommendation
   1. Avoid multiple run instruction
   2. Separate run instruction with “&& separator “
      1. RUN tar xvfz apache\*.tar.gz \
      2. && mv apache-tomcat-9.0.65/\* /opt/tomcat/. \
      3. && yum -y install java \
      4. && java -version
2. **Multi Stage Docker file**: it let you write Dockerfiles with multiple FROM statements. This means you can create images which derive from several bases, which can help cut the size of your final build (image size). to check layers = docker history image name/ID:tag

Docker images are created by selecting a base image using the FROM statement, You then add layers to that image by adding commands to your Dockerfile. With multi-stage builds, you can split your Dockerfile into multiple sections. Each stage has its own FROM statement so you can involve

more than one image in your builds. Stages are built sequentially and can reference their predecessors. so you can copy the output of one layer into the next. very large image that will impact the performance of applications running in the container

1. **Docker image**
   1. Docker home dir = /usr/local

To deploy in docker: we need application code;

* + 1. applications code: web-app.war
    2. dependencies: tomcat:8.0.20-jre8

Jre8 = java run time version 8

* 1. **docker build -t myimage:1** . = building from default name

docker build -t myimage:1 -f Dockerfile-jboss . = customized name of the file

1. -t = tag, myimage = name, 1= version, . = current working dir (pwd), -f = flag
2. **Docker Containers -**
   1. docker run (image name ) - deploy app
      1. docker run image:1
      2. docker run --name containerName -d -p hostPort:containerPort imageName:v
      3. docker run --name sonarjb -d -p 8000:8080 myimage:2
      4. docker run --name sonartom -d -p 9000:8080 myimage:1

--name = assigned container name, -d = detachable format/mode, -p = port( host port : container port), v= version,

* + 1. <http://3.99.247.124:9000/web-app/> = to view the application
  1. A host port **cannot** be used twice

1. **docker push to move image to dockerhub**
   1. docker tag myimage:1 sonar/mwapp:5
   2. To give connect with dockerhub
      1. Access to dockerhub = docker login -u Username (dockerhub username)
      2. then it will ask for password
      3. docker tag myimage:1 sedoten/sonar-app:5
      4. docker push sedoten/sonar-app:5
      5. Tag:8 = hello,Tag:7= springapp, 6=
   3. F
2. Scanning a docker file is recommended after pulling from dockerhub before running for vulnerable purposes.

docker pull mylandmarktech/maven-web-app:12

  docker scan mylandmarktech/maven-web-app:12

  docker run -d -p 3000:8080 mylandmarktech/maven-web-app:12

1. Docker/jenkins- can either
   1. Run a container directly
      1. docker run --name sonar -d -p 6000:9000 sonarqube
   2. Pull dockerfile from dockerHub = docker pull sonatype/nexus3
2. docker exec containerName = In docker, to run command inside container
   1. To get nexus container password = docker exec containername command
      1. docker exec nexus cat **/nexus-data/admin.password**
3. Docker mode can be run in ;
   1. Interactive mode = it runs a bash shell inside an image
      1. Docker run -it ubuntu /bin/bash
   2. Detachable mode =
      1. Docker run --name mywildfly -d -p 8080:8080 jboss/wildfly
4. List of containers
   1. only stopped container
      1. docker ps -aq --filter status="exited"
      2. docker ps -a --filter status="exited"
   2. List Running Containers
      1. docker ps -a --filter status="running"
      2. docker ps
      3. docker container ls
   3. List All Containers
      1. docker ps -a
      2. docker container ls -a
   4. List only running container ids
      1. docker ps -q
      2. docker container ls -q
   5. pause/unpause
      1. docker pause <containerId/Name>
      2. docker unpause <containerId/Name>
5. How to resolves issues occurring in a docker container??
   1. docker ps to ensure the container is running
   2. docker ps -a to ensure that the container exist
   3. docker inspect <name/id> =it shows configuration, it show details like metadata in short, on docker images
      1. workingdir,
      2. entry point,
      3. Architecture
      4. Exposed port
      5. date created,
      6. os <linux>
      7. ID
   4. docker exec <containerId/Name> and command you want to run = This is use to execute commands inside a container
      1. docker exec containerId commands
      2. docker exec app ls
      3. docker exec -it <containerId/Name> /bin/bash
      4. docker exec -it app /bin/bash
   5. docker top = to check resources / Display the running processes of a container
   6. docker logs <containerId/Name> insufficient / Limited resources
      1. docker logs myapp | grep errors
      2. docker logs myapp | grep exception
      3. docker logs --tail <NoOflines> <containerId/Name>
      4. docker logs --tail 10 app | grep errors
      5. docker logs --tail 5 app | grep errors
   7. docker stats = to check resources and consumption details
      1. e0845ab3308e app 97% cpu 98% mem
   8. docker attach <on container>
   9. check: Environments variables and ExposedPorts ?
6. How to copy files from container to host system or host system to container?
   1. docker cp app:/usr/local/tomcat/logs . = from container to host system
   2. docker cp test28 app:/usr/local/tomcat/ = host system to container
      1. app = container name
      2. /usr/local/tomcat/logs = container location (docker exec app pwd)
      3. Test28 = file created in host
7. The default setup/configuration of my container has change, How can I apply those changes for future application deployment.
   1. answer: Use docker commit to apply the changes to my container images
8. What is docker commit?Using docker commit we can create image from the container which will include changes effected
   1. docker commit <containerId/Name> <imageName>
      1. docker commit myapp newimage:1
   2. docker tag newimage:1 sedoten/sonar-app:10
   3. docker push

**Docker Network**

1. How micro service/ containers talk to each other = docker network. docker containers can ONLY communicate if they are in the same network by using ip/names

How can docker containers communicate with each other???

   docker containers communicate Via docker Networking

   docker containers communicate can ONLY communicate if they are in the same network

* 1. applications need to talk to the database:
     1. Networking
     2. Environmental varaibles
  2. Application required database I.e stateless app require stateful app
  3. Default bridge network - container communicate using container ips ONLY
  4. Custom bridge network - container communicate using container ips and host names/container name
  5. Commands
     1. docker network ls
     2. docker network create fintech -d bridge < -d = driver>
     3. docker run --name spingapp -d -p 80:8080 --network fintech \

Notes

* 1. If not pass the name of the driver, it will create default bridge

ubuntu@buildServer:~$ docker network ls

NETWORK ID     NAME      DRIVER    SCOPE

456c2263e715   bridge    bridge    local

d95852f9ebfb   ebay      bridge    local

c3ff56962fc2   fintech   bridge    local

a505adf682e2   host      host      local

fd5308ed7730   none      null      local

* 1. Network type
     1. bridge [ default, custom]

docker run -d --network fintech -p 800:8080 mylandmarktech/maven-web-app <custom>

* + 1. host ( no port is assigned so it will operate on docker port and it doesn’t be in use check, docker inspect) trouble

docker run --name myapp -d --network host mylandmarktech/maven-web-app

* + 1. null

Container created in the null network cannot communicate with other containers

Container created in the null network doesn't have IPs

1. **Application under network**
   1. **deploy stateless applications:**
      1. **springapp**

**docker run --name springapp -d -p 80:8080 --network fintech -e MONGO\_DB\_USERNAME=devdb -e MONGO\_DB\_PASSWORD=dev@123 -e MONGO\_DB\_HOSTNAME=mongodb mylandmarktech/spring-boot-mongo**

* 1. **deploy statefull applications:** 
     1. **mongo database:**

**docker run -d --name mongodb --network fintech -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=dev@123 mongo**

-e = passing environment variable for username and password

* 1. **Pass environment variables**
     1. docker run --name spingapp -d -p 80:8080 --network fintech \

-e MONGO\_INITDB\_ROOT\_USERNAME=devdb \

-e MONGO\_INITDB\_ROOT\_PASSWORD=dev@123 \

-e MONGO\_DB\_HOSTNAME=mongodb mylandmarktech/spring-boot-mongo

* + 1. docker run --name mongodb -d --network fintech \

-e MONGO\_INITDB\_ROOT\_USERNAME=devdb \

-e MONGO\_INITDB\_ROOT\_PASSWORD=dev@123 mongo

* 1. Notes-
     1. each container has an ip address and hostname. Either can be used to communicate with each other.
     2. Problem - containers are ephemeral <they die> I.e dynamic, they cant get a fix ip address.
     3. Docker inspect will show container network, variables

1. Database
   1. RDS/ SQL
      1. oracle
      2. mySQL
      3. PostgreSQL
      4. MariaDB
      5. Aurora
   2. In RDB

STORES DATA IN ROWS AND COLUMNS (TABLES)

   ID      Name      AGE

   ID24    SIMON     55

   ID27    Evet      66

* 1. Non- RDS
     1. mongo
     2. casandra
     3. dynamoDB
     4. REDSHIFT
  2. In non - RDB

stores data in JSON

   {

    name: 'simon'

    id: 'id20145'

    age: 55

   }

* 1. Database can be
     1. Self managed
        1. create an EC2 Server and install mySQL / PostgreSQL / oracle

mongo / casandra

* + - 1. create DB username and password

username = admin

password = admin123

* + - 1. login into the db using
    1. Provided by cloud services provider e.g AWS. just launch

1. Java based web -app:
   1. spring Boot framework --> It's a java based framework. It will eliminate the boilerplate code.

Using this framework Developers build projects very quickly, It also eases the process of developing, building, testing and deploying java-based application

* 1. deployment:
     1. app
     2. Db - <stateful>

1. Volume
   1. Bind Mounts: Bind mounts may stored data anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time. It does have persistent data
      1. docker run --name mongodb **-v** /tmp/mydata:/data/db -d mongo ( check docker doc)
      2. docker run -d --name mongodb --network fintech -e MONGO\_INITDB\_ROOT\_USERNAME=devdb -e MONGO\_INITDB\_ROOT\_PASSWORD=dev@123 **-v** /tmp/mydata:/data/db mongo
      3. Notes -
         1. If container is deleted data will be stored in /tmp/mydata
   2. Persistent volume: This is a piece of storage managed by the docker service
      1. docker volume create volumeName -d driver
      2. docker volume create data28

**Docker best practices**

1. Containers are meant to be lightweight, to be isolated, having more than one process run in container can cause issues
2. IQ: Explain a problem you faced in your Project recently? Read more
   1. new hires modify our CMD instruction - shell form vs executable form:
3. Best practices in writing Dockerfiles:
   1. 1. Use Linux lightweight alpine images were applicable/possible
      1. redhat / ubuntu / centos / alpine (lightest)
   2. Reduce the number of layers by minimizing the number of RUN instructions in your Dockerfile. Multiple run instructions affects the performance of containers.
      1. RUN yum install java -y \
      2. && yum install maven -y \
      3. && yum install git wget unzip tree nano zip -y \
      4. && git clone github/maven-web-app
   3. Use only official images in docker hub for base image <docker inc. Guaranteed, updated etc. >
   4. Avoid downloading unnecessary software and packages
   5. Use multistage Dockerfile were applicable.
   6. scan base images before deployment
   7. Use executable form over shell form for CMD and/or ENTRYPOINT instructions
   8. Developers should not hard code
4. Explain a project you are proud of ??
   1. I migrated jenkins running on a Linux VM {EC2 INSTANCE} TO Jenkins running as a docker container using Job Import Plugin
5. To know used port in docker = netstat -nutlp < network tools has to be installed first = sudo apt install net-tools>
6. Port range - 0 - 65355
7. Other notes
   1. **cat /nexus-data/admin.password = to get nexus password** in a nexus container

**To create Dockerfile**

FROM python:3.9

ADD   script.py .

RUN pip install --no-cache-dir --upgrade sqlalchemy==1.3.16

CMD [ “python”, “./script.py”]

**To build the image**

docker build -t python-db .

**To deploy the  container**

docker run --name mypython-db -d  python-db

Questions

* 1. Monolithic architecture and micro services- docker network ??? Experience in automation of VMs, Containers and packaging Java artifacts.

How to resolves issues occuring in a docker container??, docker pest practices

1. Why run in detachable ?
2. A time you troubleshoot a problem. What you did you do and how did you resolve it.
3. How you provision.
4. Can you run jenkins as a container
5. Read more on docker file
6. To move from monolith to micro service decoupled ?

Notes

1. Benefit of devops
   1. Reliability
   2. Improved collaboration
   3. Security
   4. Scale
   5. speed
2. Benefit of micro services ?????????
3. Pipeline benefit
   1. Visualization of the process
   2. Simplification and standardization
   3. Safer and faster
4. AWS code pipeline ??????
5. S
6. S
7. S
8. S