**Docker Commands**

**Docker Engine** -> Docker CLI + Rest API + Docker Deamon.

**docker login** -> login docker cli to docker account

**docker run nginx** -> pull either from local repository or remote image repository and run it.

**docker run image\_name/tag** -> by deafult tag considered as latest

**docker run -d ngnix** -> pull either from local or remote and run it in a detached mode and will return the container\_id

**docker attach container\_id** -> it will again attach cmd to the docker container

**docker run -p 5000:80 nginx** -> so nginx on port 80, on docker internal network will be connected to externally port 5000. http://localhost:8080 or http://host-ip:8080 in your browser.

**docker inspect container\_id/container\_name** -> to get more information about the image, networks, environment variables and all other things are there

**docker logs container\_id/container\_name** -> to get logs for the container

**docker pause conainer\_id** -> To pause a running container.

**docker stop container\_name/container\_id** -> to stop a running container gracefully.

**docker rm container\_name/container\_id** -> to remove a running container after stopping

**docker exec -it container\_name/container\_id /bin/bash** -> to run bash of a running container

**docker ps** -> Show the running containers

**docker ps -a** -> show all the containers

**docker pull nginx** -> pull either from local repository, it will not run it.

**docker images** -> to get all available images and its size

**docker rmi image\_name** -> to delete an image

**docker image prune -a** -> for deleting all images

**docker exec** -> Run a command in a running container

**docker exec -it ubuntu\_bash bash** -> This will create a new Bash session in the container ubuntu\_bash.

**docker run -it ubuntu bash** -> it will run ubuntu image and then run bash in interactive mode in the same command prompt where ‘i’ mean interactive and ‘t’ means same terminal

**docker run --env MYSQL\_ROOT\_PASSWORD=100997 mysql** -> we can specify environment variable using –env

**docker volume create data\_volume** -> it will create a data\_ volume folder under the volume directory of "var/lib/docker" directory

**docker run -v data\_volume:var/lib/mysql mysql:latest** -> this will mount the data\_volume directory inside the docker

**docker run -v /opt/tempMysql:/var/lib/mysql --env MYSQL\_ROOT\_PASSWORD=100997 mysql** -> -v means it map docker internal directory to external file storage

**docker run -v external\_path:var/lib/mysql mysql:latest** -> if we want to use external directory

**docker run –mount type=bind,source=C:/Users/ghosh/OneDrive/Desktop/devops/data/mysql,target=/var/lib/mysql mysql ->** docker uses the storage driver to do all volume and storage related things.docker uses the different type of storge drivers

**docker volume prune -a** -> for deleting all volumes

**docker build -t repo\_name/image\_name:version dockerfile\_location ->** build a docker image from Dockerfile ( example : docker build -t abhishek1009/arc-reactor-digital-test:1.0.0 . )

**docker build -t repo\_name/image\_name:version -f <Dockerfile\_name> dockerfile\_location ->** build a docker image from Dockerfile with different name. (example: docker build -t abhishek1009/arc-reactor-digital-test:1.0.1 -f Dockerfile.optimized . )

**Theory**

container is meant to run a specific task like a server or a process. the container will be running if process inside in it will be in a living stage. Docker is not meant to run an OS like ubuntu. If we try to run Ubuntu then it just run once then it will be stopped, so to run we must start any process inside it.

So, we can append a command into the process like this ->

**docker run ubuntu sleep 10** -> it will run ubuntu image and sleep for 10 sec. when 10 sec is completed then ubuntu will be stopped as it has no running process

**docker run ubuntu cat /etc/"os-release"** -> ubuntu version

**Docker compose**

docker build . -t voting-app

docker build . -t worker-app

docker build . -t result-app

docker run -d --name=redis redis

docker run -d -e POSTGRES\_PASSWORD=10091997 --name=db postgres:9.4

docker run -p 5000:80 --link redis:redis voting-app

docker run -p 5001:80 --link db:db result-app

docker run --link redis:redis --link db:db worker-app

docker-compose up --build with docker-compose.yml

**Single stage not optimized dockerfile**

FROM openjdk:11-jre-slim

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} app.jar

EXPOSE 8080

ENTRYPOINT ["java","-jar","-Xms256m", "-Xmx512m","/app.jar"]

**Multi stage optimized dockerfile**

# =========== BUILD STAGE =====================

FROM maven AS build

WORKDIR /workspace/app

# build maven .m2 cache as layer for reuse

COPY pom.xml pom.xml

RUN mvn dependency:go-offline -B

# build application as fat executable JAR

COPY src src

RUN mvn package -DskipTests

# explod fat executable JAR for COPY in RUN stage

RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../\*.jar)

# =========== RUN STAGE =====================

#FROM openjdk:alpine

FROM openjdk:11-jre-slim

VOLUME /tmp

ARG DEPENDENCY=/workspace/app/target/dependency

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib

COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF

COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app

EXPOSE 8080

ENTRYPOINT ["java","-cp","app:app/lib/\*","com.github.typicalitguy.DigitalLayerApplication"]