**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 stop $(docker ps -a -q)** -> Stops all the containers

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

**docker rm $(docker ps -a -q)** ->to delete all the container

**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 commit <container\_name> <image\_name>** -> to take a snapshot of a running container

**docker history <image\_name>** -> to show all the layer details of the image

**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"]

Maven has a shortcut to build spring boot image ->

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.6.7</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.github.typicalitguy</groupId>

<artifactId>ARC\_REACTOR\_DIGITAL</artifactId>

<version>1.0.0</version>

<name>ARC\_REACTOR\_DIGITAL</name>

<description>Digital layer ARC\_REACTOR Microservice</description>

<properties>

<java.version>11</java.version>

<spring-cloud.version>2021.0.1</spring-cloud.version>

</properties>

.

.

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<image>

<name>abhishek1009/${project.artifactId}:${project.version}</name>

</image>

<pullPolicy>IF\_NOT\_PRESENT</pullPolicy>

</configuration>

</plugin>

</plugins>

</build>

</project>

Build image by maven -> mvn spring-boot:built-image

Build image with skip test -> mvn spring-boot:built-image -DskipTests

**Image layers**

* To build a good image we must divide the docker file into different stages: 1. Base Image, 2. Working directory, 3. Build, 4. Run and Expose
* In the first line of docker build docker sends the context to the docker daemon. We can reduce the context by adding dockerignore file.

For reference: go to <https://github.com/boot-services/metadata-service/tree/with-mongodb>

Check docker files in this order ->

* Dockerfile
* Dockerfie.optimised.d
* Dockerfile.multistage.d
* Dockerfile.multistage.optimized.d

**CMD and ENTRYPOINT in docker:**

If we see any dockerfile there we can find CMD, it defines the program that will be run when the container starts.

For nginx image -> **CMD[“nginx”]**

Then there is another keyword **ENTRYPOINT**. Here can specify whatever program we want to start at the start of the container and if we add anything in the command line then it will be appended with the **ENTRYPOINT**. If we may have any default value, then we can pass that in **CMD** and if we specify more than one value then we must provide list of string on **CMD** or **ENTRYPOINT**.

#Ubuntu-sleeper image: it will start a ubuntu os and then start sleep process for 10 seconds

FROM UBUNTU #Base image

ENTRYPOINT [“sleep”] #Command on start

CMD [“10”] #default value

If we want to modify the ENTRYPOINT during runtime like sleep to sleep-v2 then we can do like this:

Previous version: docker run ubuntu-sleeper

New version: docker run --entrypoint sleep-v2 ubuntu-sleeper 20

Docker is one of the implementations of the virtualization technology.

Before docker we have hypervisor.

Hypervisor based architecture is like

1. Physical layer
2. Host operating system
3. Hyepervisor (VMware, Virtual Box)
   1. Guest OS
   2. Binaries/libraries
   3. Applications

Benefits:

1. Cost efficient
2. Easy to scale

Virtualization happens on physical layer.

Limitations:

1. Kernal Resource Duplication
2. Application portability issue
3. Runtime isolation. (like running JRE 7 and JRE8 in same system)

Container based architecture is like

1. Physical layer
2. Host operating system
3. Container Engine
   1. Binaries/libraries
   2. Applications

Benefits:

1. Cost efficient
2. Fast time for boot up and deployment.
3. Portability

Virtualization happens on operating system layer.

Docker client server architecture

**Images:**

• Images are read only templates used to create containers.

• Images are created with the docker build command, either by us or by other docker users.

• Images are composed of layers of other images.

• Images are stored in a Docker registry.

**Containers:**

• If an image is a class, then a container is an instance of a class - a runtime object.

• Containers are lightweight and portable encapsulations of an environment in which to run applications.

• Containers are created from images. Inside a container, it has all the binaries and dependencies needed to run the application.

**Registries and Repositories:**

• A registry is where we store our images.

• You can host your own registry, or you can use Docker’s public registry which is called DockerHub.

• Inside a registry, images are stored in repositories.

• Docker repository is a collection of different docker images with the same name, that have different tags, each tag usually represents a different version of the image.

Docker run busybox:1.24 echo “Hello world”

|  |  |  |
| --- | --- | --- |
|  | Run Container in  Foreground | Run Container in  Background |
| Description | Docker run starts the process  in the container and attaches  the console to the process’s  standard input, output, and  standard error. | Containers started in  detached mode and exit when  the root process used to run  the container exits. |
| How to specify? | default mode | -d option |
| Can the console be  used for other  commands after the  container is started  up? | No | Yes |

Docker run --rm busybox:1.24 sleep 1 : Remove the container as soon as it exited.

Docker image layers: