

Content List

[Content List 1](#_Toc165390856)

[DOCKER 1 1](#_Toc165390857)

[Virtualization 1](#_Toc165390858)

[Containerization (Container Deployment) 1](#_Toc165390859)

[Docker Architecture 3](#_Toc165390860)

[Installing Docker 3](#_Toc165390861)

[Installing Docker in Ubuntu 4](#_Toc165390862)

[Advantages of Docker 5](#_Toc165390863)

[DOCKER 02 6](#_Toc165390864)

[DockerHub 6](#_Toc165390865)

[Amazon Elastic Container Registry (Amazon ECR) 7](#_Toc165390866)

[Docker image commands: 7](#_Toc165390867)

[Push to AWS ECR 8](#_Toc165390868)

[Deploying a Docker Application 9](#_Toc165390869)

[Stopping a running container 9](#_Toc165390870)

[Accessing a running container online 9](#_Toc165390871)

[Docker container commands: 9](#_Toc165390872)

[Deploying Jenkins image on dockerHub to demonstrate: 10](#_Toc165390873)

[\*\*Avoid address/port binding\*\* 10](#_Toc165390874)

[Deploying SonarQube image to demonstrate: 11](#_Toc165390875)

[Docker 03 12](#_Toc165390876)

[Dockerfiles 12](#_Toc165390877)

[Dockerfile keywords: 13](#_Toc165390878)

[Pulling and containerizing a jboss application... 13](#_Toc165390879)

DOCKER 1

**Def:** Docker is a containerization multi-platform software used to create, build, ship, share and deploy our applications as containers

* Build applications
* Ship application
* Deploy applications

**Q: How many environments are you supporting in your role?**

* Dev environment
* Testing/QA env
* Production env

Traditionally, multiple applications were deployed on a physical system, running say one Host OS. If one app was having issues it would likely affect the other, talk less of the physical machine having a break down.

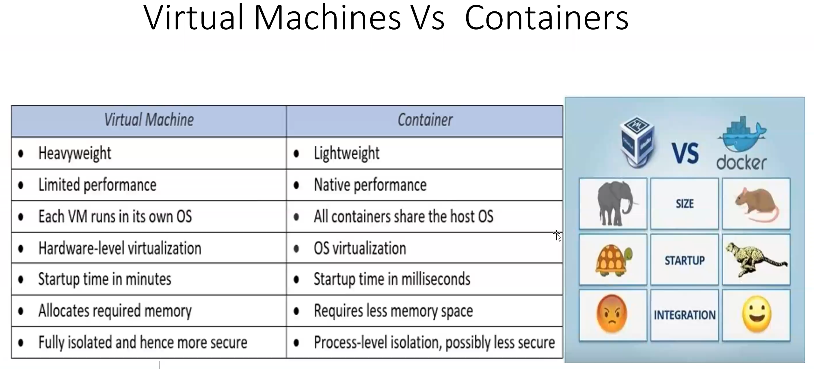
Virtualization

With a **Hypervisor** - VMware/Zen/VirtualBox etc, more Guest OS's can be installed and run on the Host OS. Hence, each application can be run on its own OS, say redhat8, ubuntu etc.

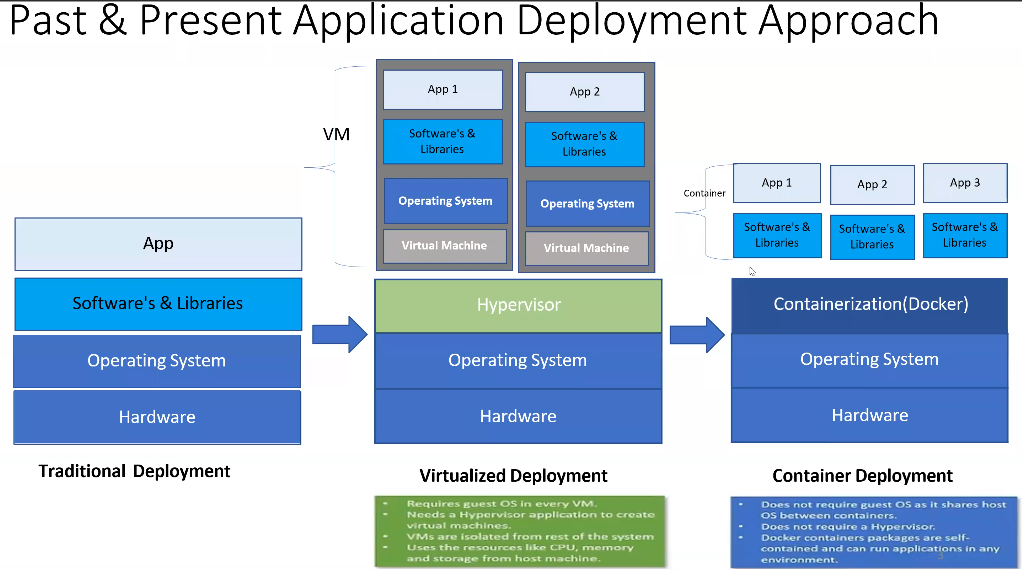
Containerization (Container Deployment)

With containerization software like **Docker**, **Pod Man**, **CRI-O**, **Rocket**, **Core-OS**, **Container-D** the guest OS is able to host more than one app. These apps share the kernel of the Guest OS and each is isolated from the other.

|  |  |
| --- | --- |
| Traditional way on physical servers | Virtualization |



|  |  |
| --- | --- |
|  |  |



1. Traditional/Physical Deployment – 1990s
2. Virtualized Deployment (Hardware level Isolation)
3. Containerization Deployment (OS/Process level Isolation)

A container contains everything (App files-code, dependencies-software and libraries), Env vars and other config files) which are required for a piece of application/process.

**For example**, if we have app.war, deployed in a virtual machine, it requires java and Tomcat installed and configured.

App code = app.war + dependencies (openjdk11+ + tomcat9)

App code = app + dependencies (npm)

Docker is available in 2 Editions:

1. Docker CE: 🡪 Community Edition (free to use)
2. Docker EE: 🡪 Enterprise Edition (Commercial/Licensed)

Which is CE +

1. Vendor support = docker.io
2. Additional features like DTR (Docker Trusted Registry)
3. UCP (Universal Control Plane) GUI

Docker is developed in Go Lang.

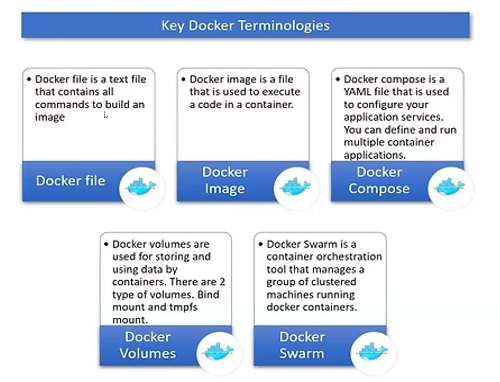
Docker Architecture

* **Docker Client** – CLI for writing user commands/instructions to docker
* **Docker Daemon** – Executes the commands sent by user, eg installing OS, pulling registry image
* **Docker Registry** – Central repo maintained by Docker known as Docker Hub. Contains various public images that can be used. Users can create and upload images in registry either as private or public

Installing Docker

Docker installation on local desktops – use the link 🡪 <https://docs.docker.com/engine/install/#desktop>

Docker can be installed on all Linux flavours (except on RedHat for Docker CE)



Installing Docker in Ubuntu

* Login to your AWS account and launch a server (ubuntu), call it dockerServer for instance

(default vpc, t2.micro, keypair, security group)

* SSH into the server - Note that default user for ubuntu is ‘ubuntu’

**Commands**:

#run script  
sudo apt update -y

sudo apt install docker.io -y

sudo service docker start

sudo docker info

sudo hostnamectl set-hostname docker

sudo su - ubuntu

* It is not a good practice to run docker as root user, in effect, regular users cannot even run docker commands, eg at this point, try running ‘docker info’ command and you’ll receive permission denied.
* Add user to docker group

sudo usermod -aG docker $USER

or

sudo usermod -aG docker ubuntu

sudo su – ubuntu

* Exit from current SSH Terminal & SSH(Login) again. Then execute

docker ps

* Alternatively, installing docker in Amazon Linux use the following script

#script

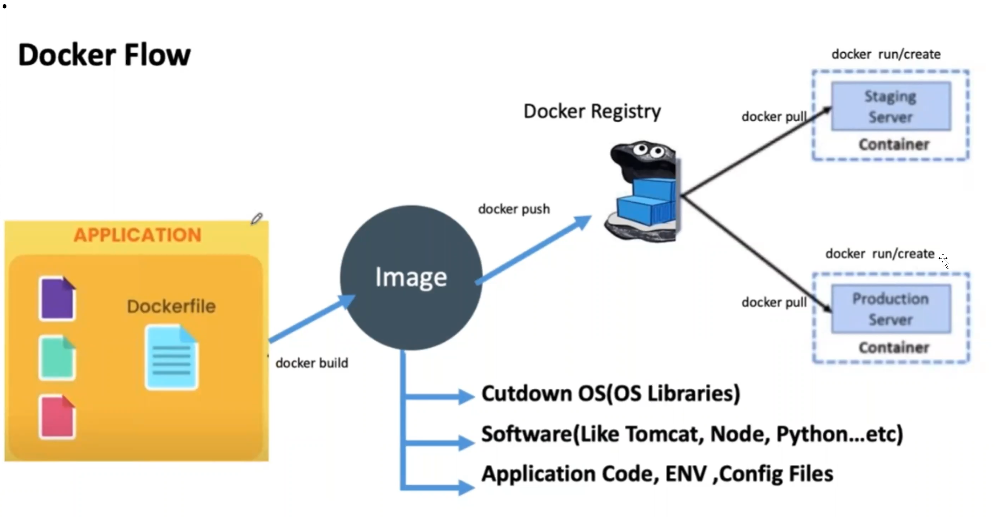
sudo yum update -y

sudo yum install docker -y

sudo service docker start

#add regular user to docker group

sudo usermod -aG docker ec2-user



Advantages of Docker

* Rapid application deployment
* Portability across machines
* Version control and component reuse
* Sharing
* Lightweight footprint and minimal overhead
* Simplified maintenance

**Some key words:**

**Dockerfile –** a file which contains instructions needed to create an image.

It contains specific keywords used to build images by the Domain Daemon.

**Docker Image** – a package which contains everything (Libraries+Softwares+ENV+Application Code) to run your application

**Docker Containers** – Runtime instance of a docker image.

*If you execute*: ‘docker run’, a container is created from the image

**Cloning a github maven web app**;

git clone [https://github.com/DevOps-Team-Cam/maven-web-app](https://github.com/DevOps-Team-Cam/maven-web-app/)

ls

mv maven-web-app/ web

cd web/

cat Dockerfile

vi Dockerfile

#replace the names of the *maven-web-app.war* with *tesla.war,* as below:

FROM tomcat:8.0.20-jre8

#### Good stuff

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

Install maven:

sudo apt install maven -y

#sudo yum install maven -y in redhat

mvn package

Run:

docker images

# to verify if any images exist.

Then run:

docker build -t teslafewa .

# to build images

Then do *docker images* command to see the images. This image now has to be deployed in order to run, and to do so, run the following command;

docker run --name teslafewaapp -d -p 8080:8080 teslafewa

#run docker ps to see the app running

docker ps

To verify whether you can connect to the app, run the curl command like below;

curl -v localhost:8080/tesla

#you can also curl using the ip address to see if you can access the app through the web

If you can’t access using the browser, ensure you opened port 8080 in your security group

**Note**! That git wasn’t installed because the AMI (Amazon Machine Image) ubuntu comes with git pre-installed.

DOCKER 02

DockerHub (https://hub.docker.com/)

* Docker images are built and saved online on the registry DockerHub which can be found at dockerhub.com
* These images are both private and public, one can get and use public images for free
* You need to create an account on dockerhub in order to use the service (This is more of a version control system like GitHub but for container images)
* After creating an account, just like github, you have an option to create a repo with the name of your app, eg teslawebapp.

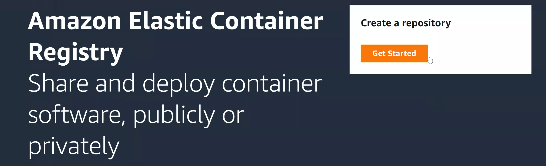
Pushing an image to dockerhub you have a docker *push* command with parts as below:

|  |  |  |
| --- | --- | --- |
| S/N | Part | Example |
| 1 | Dockerhub username | fewaitconsulting |
| 2 | Registry name | fewa-web-app |
| 3 | Tagname | latest version, mlt-v4, v1.2, etc |

Hence,

docker push fewaitconsulting /fewa-web-app:mlt-v4

docker push fewaitconsulting/fewa-web-app1:tagname

Amazon Elastic Container Registry (Amazon ECR)

An equivalent of DockerHub by Amazon for both public and private container repos

Docker image commands:

* docker build -t <imageName><tagname> -f <dockerfileName> /<path/to/Dockerfile>

for example (*Do not run these examples now*)

docker build -t webapp:1 -f dockerfile\_webapp /home/ubuntu/web/

docker build -t webapp:1 -f dockerfile\_webapp . #case where file is found in pwd

docker build -t webapp:1 -f dockerfile\_webapp ~ #case where file is found in home directory

*Note that default name is always dockerfile*

Now let’s run:

docker images

docker rmi -f tesla24 # remove an existing image

cd web/

mvn clean package #server does not have enough resources, so we clean old files

ls Dockerfile

cat Dockerfile

mv Dockerfile df

docker build -t fewa:1 -f /home/ubuntu/web/df /home/ubuntu/web/

More commands:

docker images #list all docker images

docker image ls #does same

docker image --help #gives you more commands and their functions

Once you have an account and logged into dockerHub, you can push or pull images from there using:

For example:

#docker tag = tag the local image with repository details in order to push successfully

docker tag fewa:1 fewaitconsulting/fewa-web-app:1

Note: we need authentication to push images to both public and private repos/registries

Hence, to login run the commands:

docker login -u fewaitconsulting #your password will be requested after this

docker push fewaitconsulting/fewa-web-app:1

*Note: you need a docker client (CLI), to run docker commands*

To delete an image you have locally, run the docker rmi command like so:

docker rmi -f <imageName>:<tag>/<image-id> #or just

docker rmi <image-ID>

docker rmi -f <imageName>:<tag>/<image-id> # -f, for force delete

e.g.

docker rmi fewaitconsulting/fewa-web-app:1

docker rmi 91070ab9015a #in case of image-ID

Now we can do docker pull using this command:

docker pull fewaitconsulting/fewa-web-app:1

docker images #to see the image download locally

Push to AWS ECR

To push the image to Amazon ECR, you need to install AWS on your server, create an IAM role and get authenticated to it;

Then you should look around on your ECR dashboard for “*View push commands*”, you’ll see auth commands, tag and push commands for your account (and various OS’s-macOS, Linux and Windows). Example command to login;

aws ecr get-login-password –region us-east-1 | docker login –username AWS –password-stdin 287784533479.dkr.ecr.us-east-1.amazonaws.com

Then tag the image to get it ready for pushing, for example;

docker tag mylandmarktech/landmark-web-app:1 287784533479.dkr.ecr.us-east-1.amazonaws.com/landmark-web-app:1

Then push as follows;

docker push 287784533479.dkr.ecr.us-east-1.amazonaws.com/ landmark-web-app:1

Deploying a Docker Application

Commands;

docker ps #displays running applications

#next we do docker run

docker run --name myapp -d -p 80:8080 fewaitconsulting/fewa-web-app:1

#now run docker ps to see the app runnning

Pulling some more images, eg nginx, alpine, ubuntu...

docker pull nginx

docker pull alpine

docker pull ubuntu

docker images #to see the images

*Note: you cannot delete an image with a running container*

Stopping a running container

docker stop myapp #to stop the container

docker rmi -f 91070ab9015a #for forced delete

docker ps -a # lists all containers plus those not running

docker start myapp #restarts a stopped container

Listing only docker image ID’s;

docker images -q

Accessing a running container online

curl -v 54.156.124.67:80/tesla #to check if it’s running on port 80 as was deployed

docker system prune #removes stopped containers, dangling images and build cache

Docker container commands:

🡪A container is a running process/instance of a docker image. So in other words, a container is created from and image.

docker container --help

docker pull = pulls docker image from registry

docker create = use the image to create a container

docker start = start the container

docker stop = stop the container

docker run = docker create + docker start

Deploying Jenkins image on dockerHub to demonstrate:

docker run -d jenkins/jenkins # pulls and runs the Jenkins image-in detached mode

docker ps -a #to see running or exited images

docker run --name jenkins -d -p 7000:8080 jenkins/jenkins

So the general command looks like so:

docker run –name appName -d -p hostPort:containerPort image:tagName

* the Jenkins initial admin passwork is found in

/var/jenkins\_home/secrets/initialAdminPassword

So, we have our Jenkins container running now...

Note: *docker exec* is used to execute commands inside a container

Hence, the general command looks like so:

docker exec <containerName or containerID> <command>

* to get the adminPassword, we there:

docker exec jenkins cat /var/jenkins\_home/secrets/initialAdminPassword

to interact with a running container’s shell:

docker exec -it app sh #for bourne shell

docker exec -it app bash #for bash shell

more commands:

docker ps - #list running containers

docker container ls #list running containers

docker ps -q #list running container IDs

docker ps -a #list running and stopped/exited containers

docker ps -aq #list running stopped/exited container IDs

##to copy files into the container from the docker host and vice versa

docker cp <fileName> containerID:</location/path>

docker cp app:/usr/local/tomcat/logs . #copy logs from container to host

\*\*Avoid address/port binding\*\*

This means that you should always be weary of the ports you are using for new deployments in case they might coincide with already used ports. You can run *netstat -tulpn* to see the ports you have used so far.

The command stands for, *network statistics (netstat), tcp udp listening port/process number (tulpn)*

Deploying SonarQube image to demonstrate:

free -m # to check available server resources

docker run --name sonar -d -p 4000:9000 sonarqube

docker container ls #to check if sonarqube container is running

curl -v localhost:4000 #to access the running container

Now you can access SonarQube on the browser by using http://<ipAddress:4000>/

docker ps -a --filter status=“exited” #filters and lists containers that are exited

docker rm $(docker ps -aq --filter status=“exited”)#delete all exited containers

docker rm -f $(docker ps -aq --filter status=“running”)#force delete running containers

you can use other keywords like “running” etc

let us deploy back our webapp and practice more commands with:

docker run --name webapp -d -p 80:80 fewaitconsulting/fewa-web-app

docker stop webapp

docker ps -a #container will be exited

docker start webapp

docker ps #container will be up and running

docker pause webapp #you can pause a container to do some troubleshooting etc

docker ps #container will be up but paused

docker unpause webapp

How can we troubleshoot or debug applications running as docker containers?

docker rmi -f $(docker images -q) #force delete all docker images

docker images #to verify

docker ps -aq

vi deploy.sh #create and edit a shell script (deploy.sh)

type the commands below into deploy.sh

#Deploy.sh

docker run --name javaapp -d -p 7000:8080 fewaitconsulting/fewa-web-app:1

docker run --name app -d -p 8000:8080 mylandmarktech/myapp

docker run --name myapp -d -p 8080:8080 mylandmarktech/spring-boot-mongo

Note: the below commands can go with the container name or ID

docker ps -a #verify its running, paused or exited

#if exited, docker start, if paused, docker unpause

docker top app #check resources used by app

docker stats app #check statistics, how much cpu, memory consumed etc

docker logs app #check the logs

docker logs app | grep error #checks for any errors

docker exec app pwd #checks the directory

docker exec app ls logs # summarized logs

docker inspect app #inspects everything from state to ports etc

docker attach app #atach to the local shell to inspect say logs

netstat -tulpn #used to display a list of all TCP and UDP connections that the system is currently listening on, along with the associated process IDs

Docker 03

When a container is created form and image and is running, there are some changes that will occur in the running instance. To see these changes, use the *docker diff* command, like so:

docker diff <containerName/ID>

docker diff app

We might want to at any point in time maintain these changes and use them in future images, and thence comes in handy the docker commit command;

docker commit <containerID/Name> <newImageName>:tag

for example;

docker commit javaapp fewa-web-app:2 #creates new image(fewa-web-app) from running container (javaapp)

#Note that new image name preferably should match a repo you already have on dockerhub for subsequent pushing

Now if we wish to push the image to DockerHub, we have to tag the image with username and repo name as follows:

docker tag image2:1 fewaitconsulting/fewa-web-app:2 # tags the image for subsequent push to dockerhub

docker login # to login to dockerhub account

docker push fewaitconsulting/fewa-web-app:2 #push the image to repo

* Summary of some docker container commands:

docker run/create/start/stop/kill/inspect/logs/top/stats/

docker diff/update/cp/commit/pause/unpause/ps/ps -a/exec/

docker rm/rm-f/rm/ps -q/

* Summary of some docker images commands:

docker build/login/push/pull/tag/images/images prune/

docker rmi/rmi -f/system prune

Dockerfiles

Dockerfiles are input used to build docker images

Docker images are input needed/used to create containerised applications

Example of a Dockerfile is:

FROM tomcat:8.0.20-jre8

# This is a .war application

COPY target/\*war /usr/local/tomcat/webapps/tesla.war

**DockerImage** 🡪 It is a package which contains application code + all it’s dependencies (Software+ENV Variables + Config Files) together.

Dockerfile keywords:

FROM, MAINTAINER, COPY, ADD, RUN, CMD, ENTRYPOINT, WORKDIR, ENV, EXPOSE

Pulling and containerizing a jboss application...

docker pull jboss/wildfly #pulling the image from dockerHub

cd web

docker images #to see the just pulled image

docker inspect <imageID> #to see details and how we can use the image, port number etc

vi Dockerfile\_JB #in the file, keep the two lines below

FROM jboss/wildfly:latest

ADD target/\*war /opt/jboss/wildfly/standalone/deployments/fewa.war

ls target/ #list target directory files

cat Dockerfile\_JB # to see the content

docker build -t jbfewa:1 -f Dockerfile\_JB . #to build and image

docker images #to see the image jbfewa in this case

#delete image cos it’s no longer needed

#always delete images not needed in order to clear memory and cpu resourses

df -h # see server resources and their usage

docker ps #to see running applications

#now let us deploy as below

docker run -d -p 6000:8080 --name jbapp jbfewa:1 #deployed at port 6000

docker ps #to see that it is running

Multiple applications can be deployed from the same container image to different ports numbers and ensure different app names too. Examples are shown below;

# docker run -d -p 8888:8080 --name jbapp2 jbfewa:1

# docker run -d -p 8887:8080 --name jbapp3 jbfewa:1

# docker run -d -p 8111:8080 --name jbapp4 jbfewa:1

We can then curl to check if our app is running;

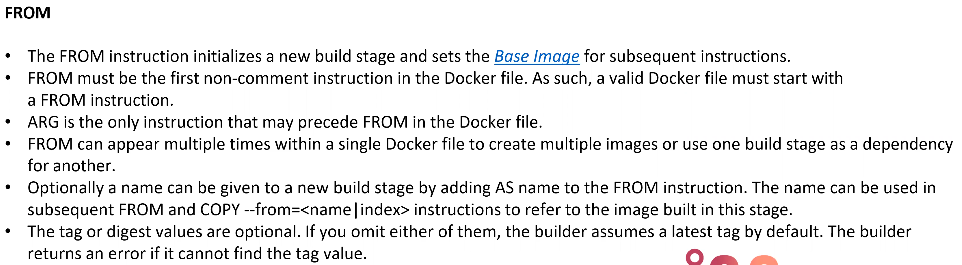
curl -v localhost:6000/fewa

curl -v 44.211.75.85:6000/fewa

docker exec jbapp pwd #to check the running home directory

docker exec jbapp ls /opt/jboss #list files in jboss directory

docker exec jbapp ls /opt/jboss/wildfly/standalone/deployments #see fewa.war.deployed

Base image generally contains Libraries and dependencies.

FROM keyword

Usage:

FROM <image>

FROM <image>:<tag>

FROM <image>@<digest>

Examples;

Java-standalone applications

FROM openjdk11

Java-web applications

FROM tomcat:9.0.7-openjdk11

FROM jboss/wildfly

**Hence**: the keyword FROM is used to determine the base image

Examples:

FROM openjdk:8-alpine

FROM nginx

FROM node

FROM python

MAINTAINER keyword

It will be used as comments to describe author/owner who is maintaining the docker file. This instruction allows you to set the Author/Owner of the image.

MAINTAINER <name>

MAINTAINER FewaITConsulting [fewaitconsulting@gmail.com](mailto:fewaitconsulting@gmail.com)

MAINTAINER Docker Community

COPY keyword

Using COPY we can copy file/folders to the image.

Files/Folders will be copied to the image while building an image.

Syntax;

COPY <source> <destination>

ServerFile/FolderPath PathInsideImage

Examples;

COPY target/tesla.war /usr/local/tomcat/

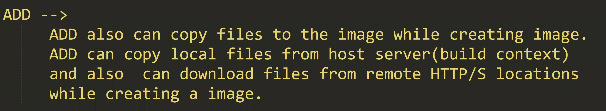
When copying, if you have a file or folder you want to ignore, you can add to dockerignore as so;

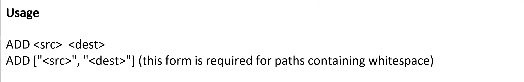
touch .Dockerignore #then add the filename in to it

vi .Dockerignore #in here, add the name of the file or folder to be ignored

cat .Dockerignore #to see the content

ADD keyword



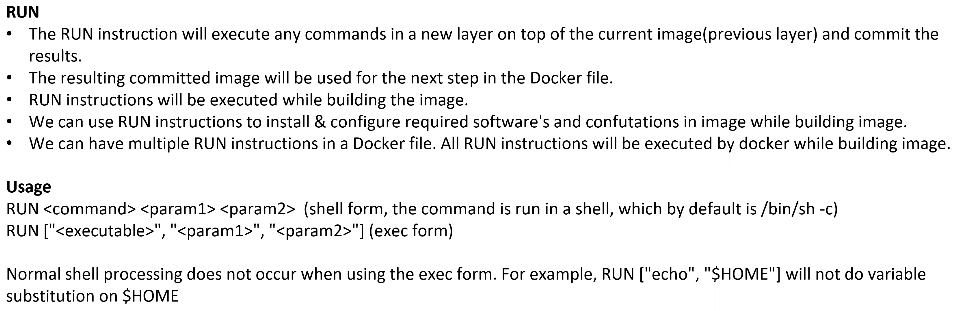


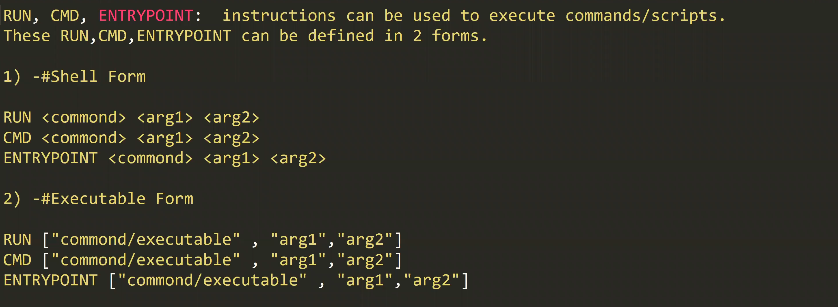
Example (script);

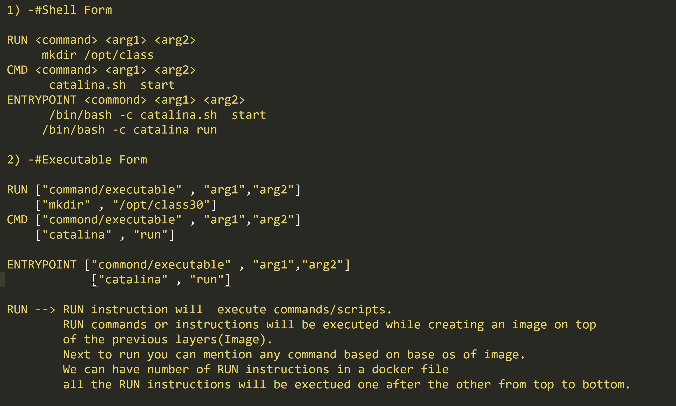
FROM ubuntu

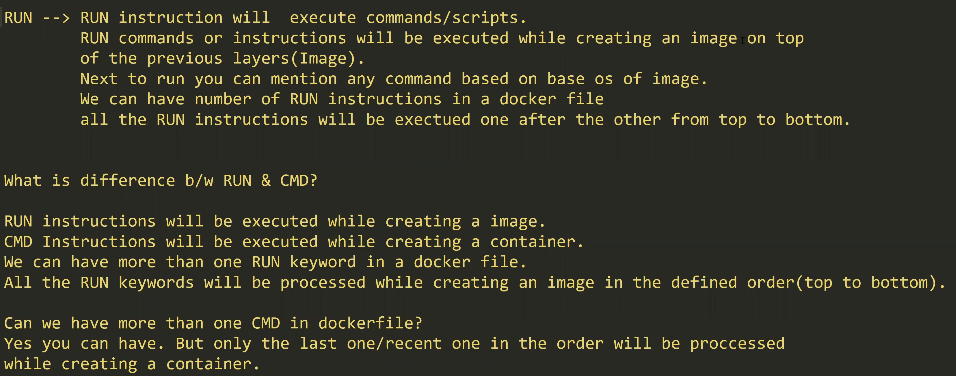
ADD [https://dlcn.apache.org/tomcat/tomcat-9/v9.0.71/bin/apache-tomcat-9.0.71.tar.gz /opt/](https://dlcn.apache.org/tomcat/tomcat-9/v9.0.71/bin/apache-tomcat-9.0.71.tar.gz%20/opt/)

RUN keyword









Examples;

Let us vi into a script and name it Dockerfile\_RUN, the following content is typed inside (note that there should exist an image named imageone:2)

FROM imageone:2

RUN mkdir /opt/class01

RUN touch /opt/class01/jobs

ADD . /opt/class01

Save the file and then run:

docker build -t imageone:3 -f Dockerfile\_RUN .

docker history imageone:3

Note that shells are used to execute docker commands;

echo $SHELL

/bin/bash -c pwd # runs a Bash shell (/bin/bash) inside a Docker container and executes the pwd command

mkdir laps

cd laps/

vi Dockerfile\_CMD

#Write the following lines in the file

# Use Ubuntu as the base image

FROM ubuntu

# Create a directory named "class30" in /opt

RUN mkdir /opt/class01

# Create a file named "jobs" inside /opt/class30 directory

RUN touch /opt/class01/jobs

# Add the content of the current directory into /opt/class30 directory

ADD . /opt/class01

# Print the current working directory as the command to execute when the container starts

CMD pwd

# Print "I am rich" as the command to execute when the container starts

CMD ["echo", "I am rich"]

Then build;

docker build -t image2 -f Dockerfile\_CMD .

ENTRYPOINT keyword

ENTRYPOINT configures a container that will run as an executable. It is a command or script that is executed when you run the docker container.

Example;

Create file Dockerfile\_EP:

FROM ubuntu

RUN mkdir /opt/class30

RUN touch /opt/class30/jobs

ADD . /opt/class30

ENTRYPOINT ["echo", "I am rich from ENTRYPOINT"]

Build with;

docker build -t image -f Dockerfile\_EP .

Then do;

docker run image3 # to see the ENTRYPOINT text printed



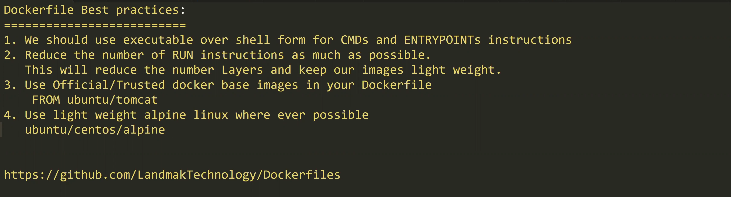
Try ;

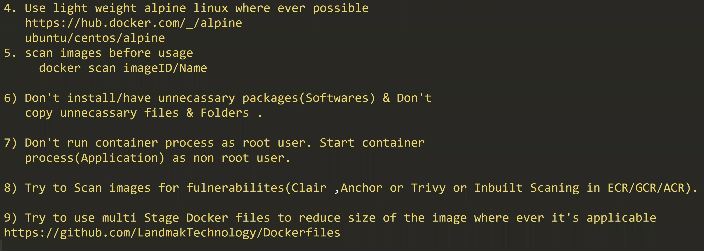
Docker run image3

Docker run image3 whoami #output not overridden

Docker run image2

Docker run image2 whoami #notice the output is overridden





Visit or clone this repo:

<https://github.com/Landmark-Technologies/Dockerfiles>

Multistage Dockerfiles

Example: vi into multi-stage\_DF and paste the following code

# Stage 1: Clone the repository

FROM alpine/git as repo

LABEL maintainer="fewaitconsulting@gmail.com"

WORKDIR /app

RUN git clone https://github.com/Landmark-Technology/maven-web-application

# Stage 2: Build the Maven project

FROM maven:3.5-jdk-8-alpine as build

WORKDIR /app

COPY --from=repo /app/maven-web-application /app

RUN mvn install

# Stage 3: Deploy to Tomcat

FROM tomcat:8.0.20-jre8

COPY --from=build /app/target/\*.war /usr/local/tomcat/webapps/maven-web-app.war

Then do:

docker build -t image4 -f multi-stageDF .

#then run docker images to see the image

docker images

docker system prune command;

