**ocker commands:**

1. [docker –version](https://www.edureka.co/blog/docker-commands/#version)
2. [docker pull](https://www.edureka.co/blog/docker-commands/#pull)
3. [docker run](https://www.edureka.co/blog/docker-commands/#run)
4. [docker ps](https://www.edureka.co/blog/docker-commands/#ps)
5. [docker ps -a](https://www.edureka.co/blog/docker-commands/#psa)
6. [docker exec](https://www.edureka.co/blog/docker-commands/#exec)
7. [docker stop](https://www.edureka.co/blog/docker-commands/#stop)
8. [docker kill](https://www.edureka.co/blog/docker-commands/#kill)
9. [docker commit](https://www.edureka.co/blog/docker-commands/#commit)
10. [docker login](https://www.edureka.co/blog/docker-commands/#login)
11. [docker push](https://www.edureka.co/blog/docker-commands/#push)
12. [docker images](https://www.edureka.co/blog/docker-commands/#images)
13. [docker rm](https://www.edureka.co/blog/docker-commands/#rm)
14. [docker rmi](https://www.edureka.co/blog/docker-commands/#rmi)
15. [docker build](https://www.edureka.co/blog/docker-commands/#build)

1.docker --version

2. **docker pull <image name>**

**docker run -it -d <image name>**

**docker ps**

**docker ps –a**

**docker exec -it <container id> bash**

**docker stop <container id>**

**docker kill <container id>**

This command kills the container by stopping its execution immediately. The difference between ‘docker kill’ and ‘docker stop’ is that ‘docker stop’ gives the container time to shutdown gracefully, in situations when it is taking too much time for getting the container to stop, one can opt to kill it

**docker commit <conatainer id> <username/imagename>**

**docker login**

**docker push <username/image name>**

**docker images**

**docker rm <container id>**

**docker system df**

**docker system df –v**

**docker stats**

**docker rmi <image-id>**

**docker build <path to docker file>**

**sudo docker tag nginx:latest rammohan478/dev:ramu**

**sudo docker push rammohan478/dev:ramu**

**docker volumes:**

|  |  |
| --- | --- |
| **Command** | **Description** |
| docker volume create | Create a volume |
| docker volume inspect | Display detailed information on one or more volumes |
| docker volume ls | List volumes |
| docker volume prune | Remove all unused local volumes |

**Start a container with a volume**

If you start a container with a volume that doesn’t yet exist, Docker creates the volume for you. The following example mounts the volume myvol2 into /app/ in the container.

docker run -d --name ram --mount source=myvol2,target=/app ubuntu:latest

docker inspect ram

docker container stop ram

docker container rm ram

docker volume rm myvol2

# **Docker – Compose**

https://www.tutorialspoint.com/docker/docker\_compose.htm

**Docker Compose** is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.

## Docker Compose ─ Installation

*curl -L "https://github.com/docker/compose/releases/download/1.10.0-rc2/dockercompose*

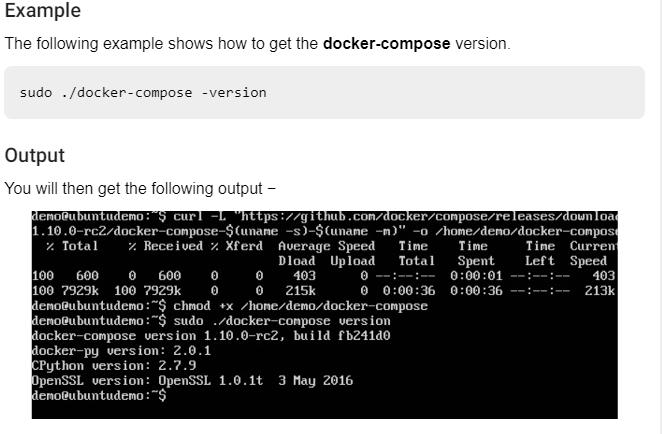
*-$(uname -s) -$(uname -m)" -o /home/demo/docker-compose*

The above command will download the latest version of Docker Compose which at the time of writing this article is **1.10.0-rc2**. It will then store it in the directory **/home/demo/**.

Next, we need to provide **execute privileges** to the downloaded Docker Compose file, using the following command –

*chmod +x /home/demo/docker-compose*

*docker-compose version*

**



* The **database** and **web** keyword are used to define two separate services. One will be running our **mysql** database and the other will be our **nginx** web server.
* The **image** keyword is used to specify the image from **dockerhub** for our **mysql** and **nginx** containers
* For the database, we are using the ports keyword to mention the ports that need to be exposed for **mysql**.
* And then, we also specify the environment variables for **mysql** which are required to run **mysql**.

Now let’s run our Docker Compose file using the following command −

sudo ./docker-compose up

This command will take the **docker-compose.yml** file in your local directory and start building the containers.

And when you do a **docker ps**, you can see that the containers are indeed up and running.

## Use a volume with Docker Compose

Here’s a+

-n example of a single Docker Compose service with a volume:

services:

frontend:

image: node:lts

volumes:

- myapp:/home/node/app

volumes:

myapp:

Running docker compose up for the first time creates a volume. The same volume is reused when you subsequently run the command.

### **start a service with volumes**

When you start a service and define a volume, each service container uses its own local volume. None of the containers can share this data if you use the local volume driver. However, some volume drivers do support shared storage.

The following example starts an nginx service with four replicas, each of which uses a local volume called myvol2.

docker service create -d --replicas=4 --name devtest-service --mount source=myvol2,target=/app nginx:latest

Use docker service ps devtest-service to verify that the service is running:

docker service ps devtest-service

You can remove the service to stop the running tasks:

docker service rm devtest-service

## Child commands

Command Description

[docker service create](https://docs.docker.com/engine/reference/commandline/service_create/) Create a new service

[docker service inspect](https://docs.docker.com/engine/reference/commandline/service_inspect/) Display detailed information on one or more services

[docker service logs](https://docs.docker.com/engine/reference/commandline/service_logs/) Fetch the logs of a service or task

[docker service ls](https://docs.docker.com/engine/reference/commandline/service_ls/) List services

[docker service ps](https://docs.docker.com/engine/reference/commandline/service_ps/) List the tasks of one or more services

[docker service rm](https://docs.docker.com/engine/reference/commandline/service_rm/) Remove one or more services

[docker service rollback](https://docs.docker.com/engine/reference/commandline/service_rollback/) Revert changes to a service’s configuration

[docker service scale](https://docs.docker.com/engine/reference/commandline/service_scale/) Scale one or multiple replicated services

[docker service update](https://docs.docker.com/engine/reference/commandline/service_update/) Update a service

https://docs.docker.com/storage/volumes/

create tag and upload file from local docker hub

sudo docker tag nginx:latest rammohan478/dev:rahul

sudo docker push rammohan478/dev:rahul

# **Networking overview**

https://docs.docker.com/network/network-tutorial-standalone/

Commands

docker network ls

docker run -dit --name alpine1 alpine ash

docker run -dit --name alpine2 alpine ash

docker container ls

docker network inspect bridge

docker attach alpine1

ip addr show

## What is Docker?

Docker is a configuration management tool that is used to automate the deployment of software in lightweight containers. These containers help applications to work efficiently in different environments.

**What is Docker?**

Docker is a configuration management tool that is used to automate the deployment of software in lightweight containers. These containers help applications to work efficiently in different environments.

Features of Docker:

* Easy and faster configuration
* Application isolation
* Security management
* High productivity
* High scalability
* Infrastructure independent

Docker container is a software package that has all the dependencies required to run an application.

Note: A Docker Image is a template of instructions used to create containers

## What is a Dockerfile?

Before we discuss what is a Dockerfile, it is important to know what a Docker image is.

### **Docker Image:**

A Docker Image is a read-only file with a bunch of instructions. When these instructions are executed, it creates a Docker container.

### **Dockerfile:**

Dockerfile is a simple text file that consists of instructions to build Docker images.

Sample dockerfile:

FROM ubuntu

MAINTAINER ram [mohanlucky7777@gmail.com](mailto:mohanlucky7777@gmail.com)

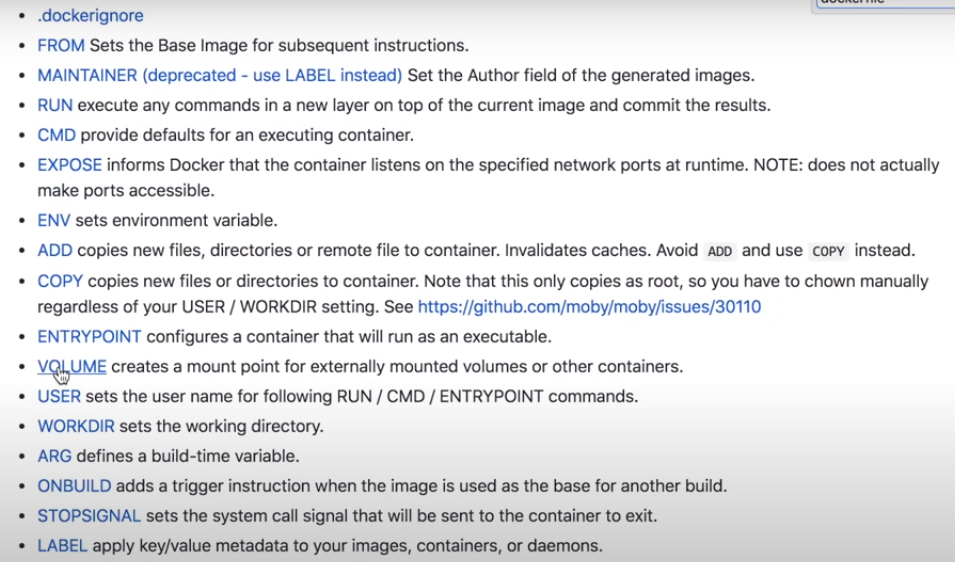
RUN apt-get update

CMD [“echo”,”hello world..!from my first docker image”]

Docker build –t Ubuntu:2.0 .

Docker run imageid

FROM



FROM :sets the base image for subsequent instructions

MAINTAINER: set the author field of the generated fields

RUN execute any commands in a new layer on top of the current image and commit the result

CMD provide defaults for an executing container

EXPOSE informs docker that the container listen on the specified network ports at runtime

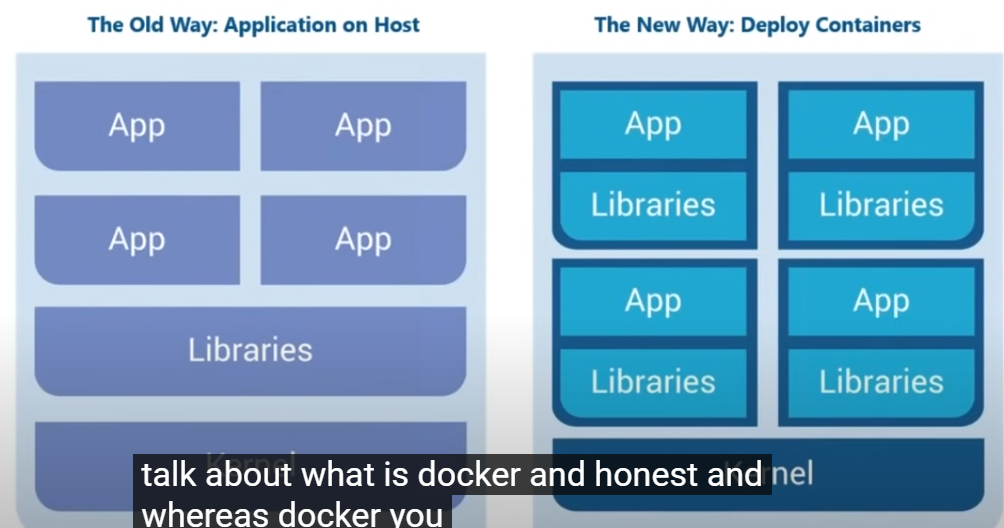
NOTE does not actually make ports accessible

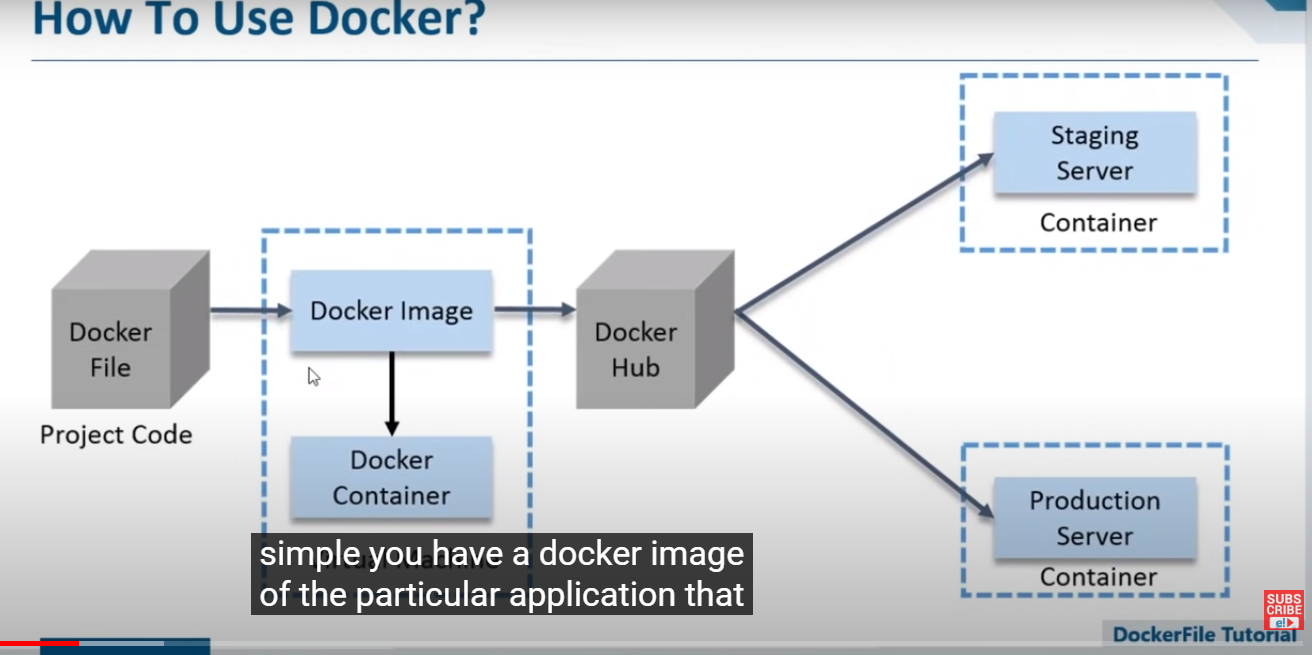
COPY; copy the files from source pathe to destination path

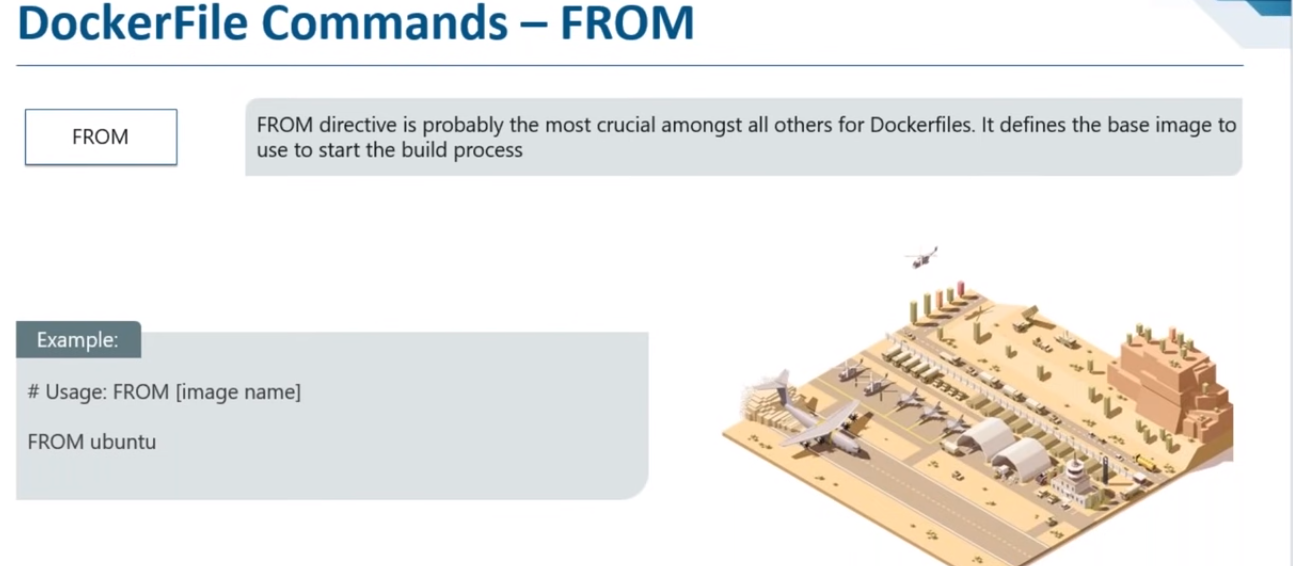
ADD:it can also works like copy command and inaddition it can add the remote files like artifacts download from internet

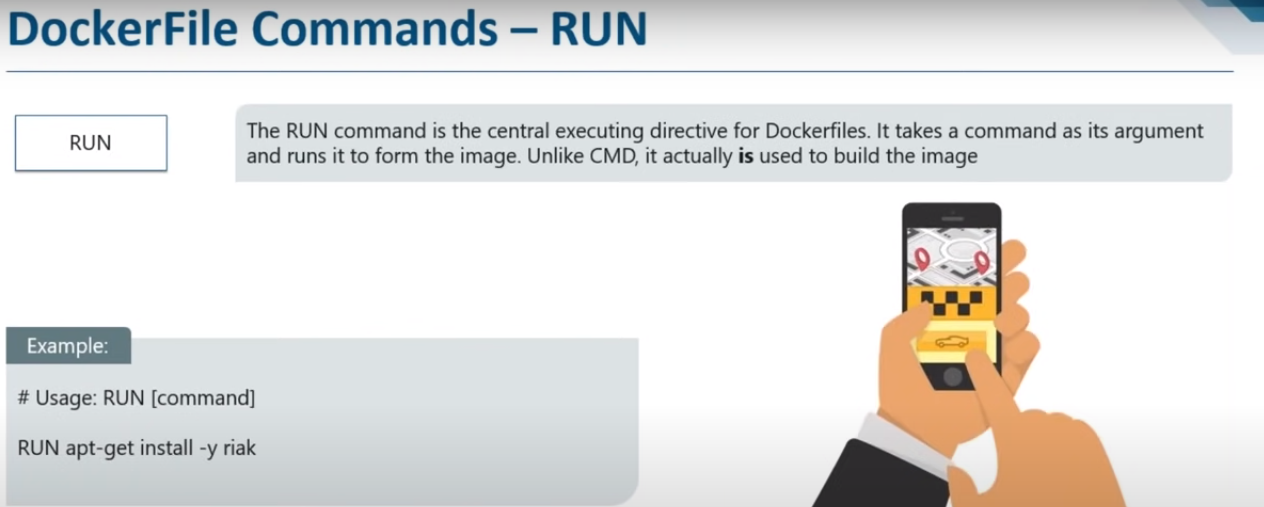
One more difference is it can extart the downloaded files

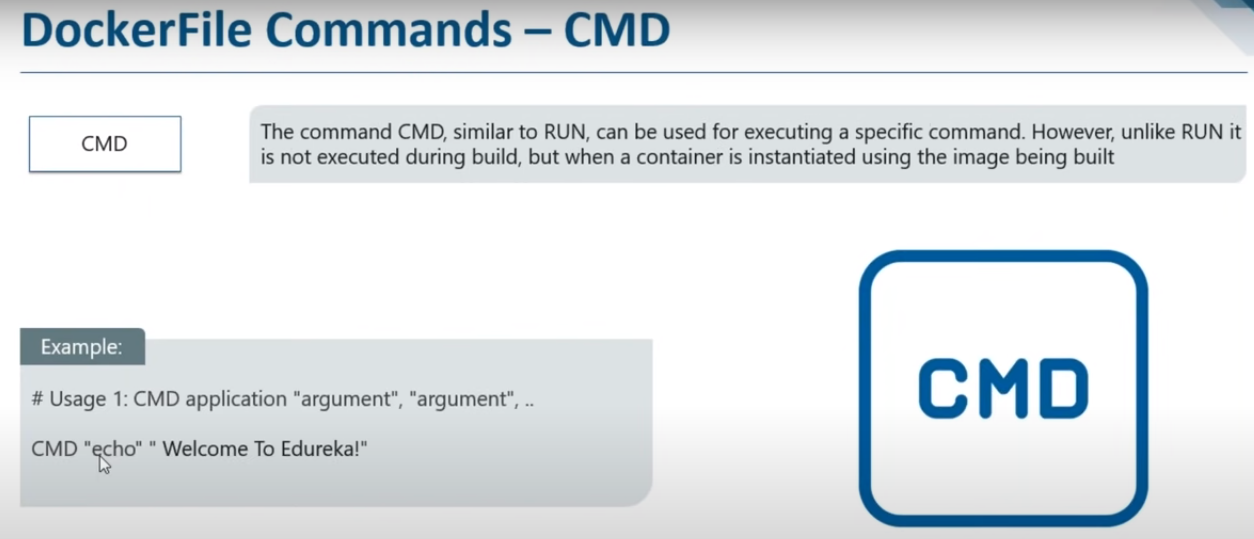
Run instruction uses during the build process only

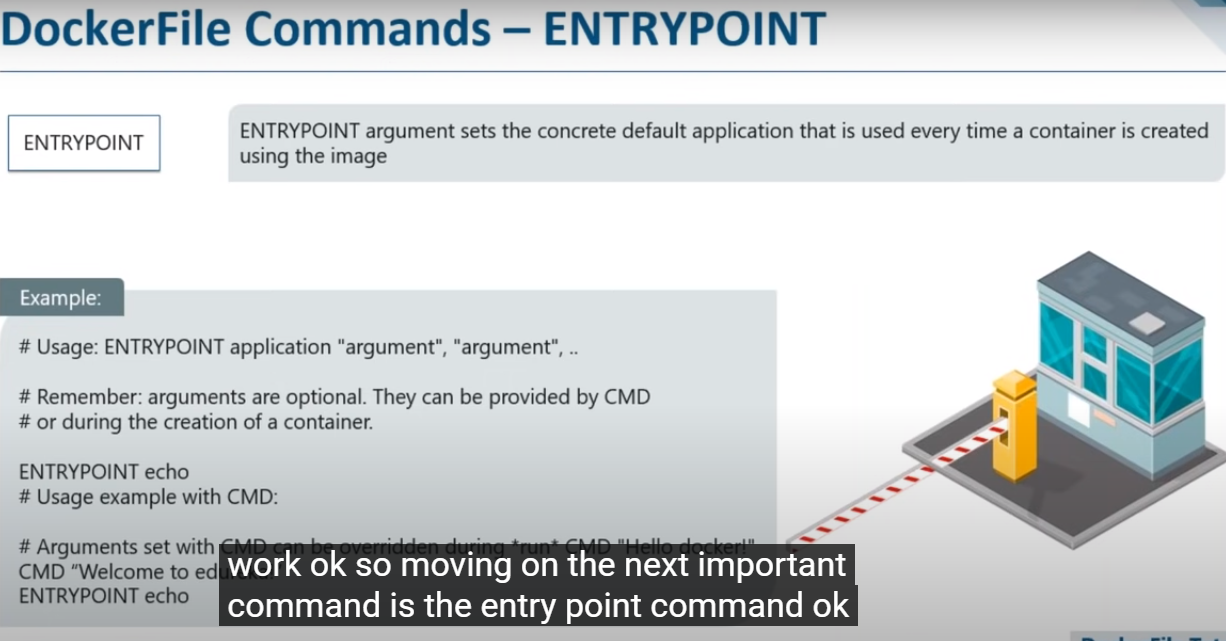


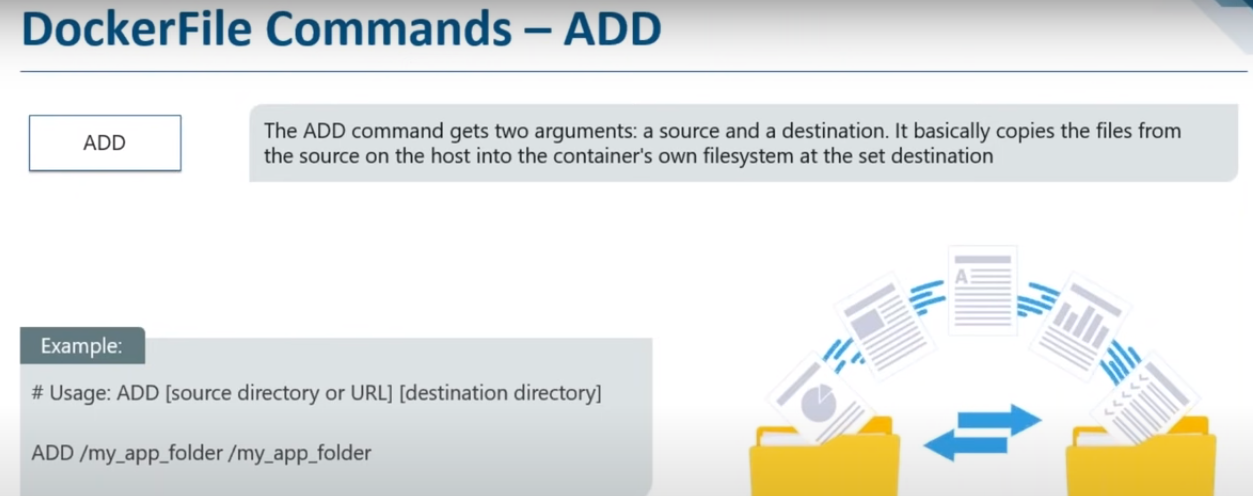




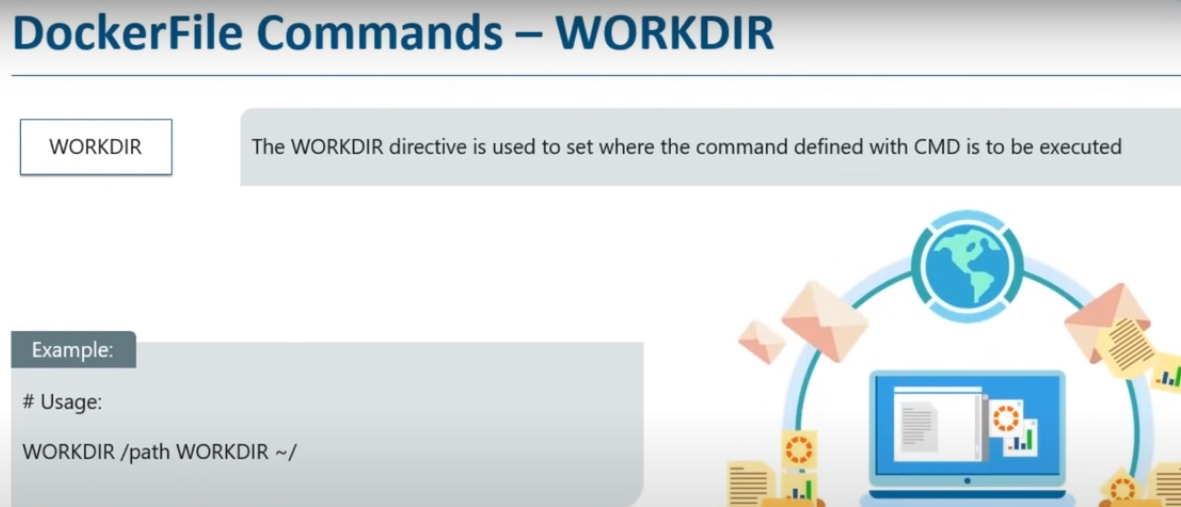


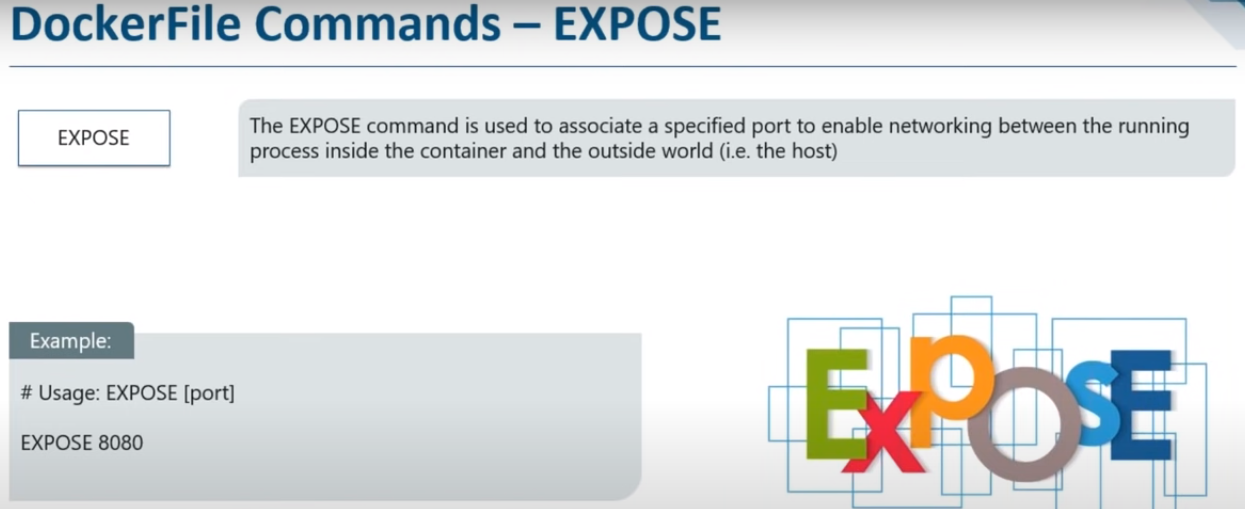




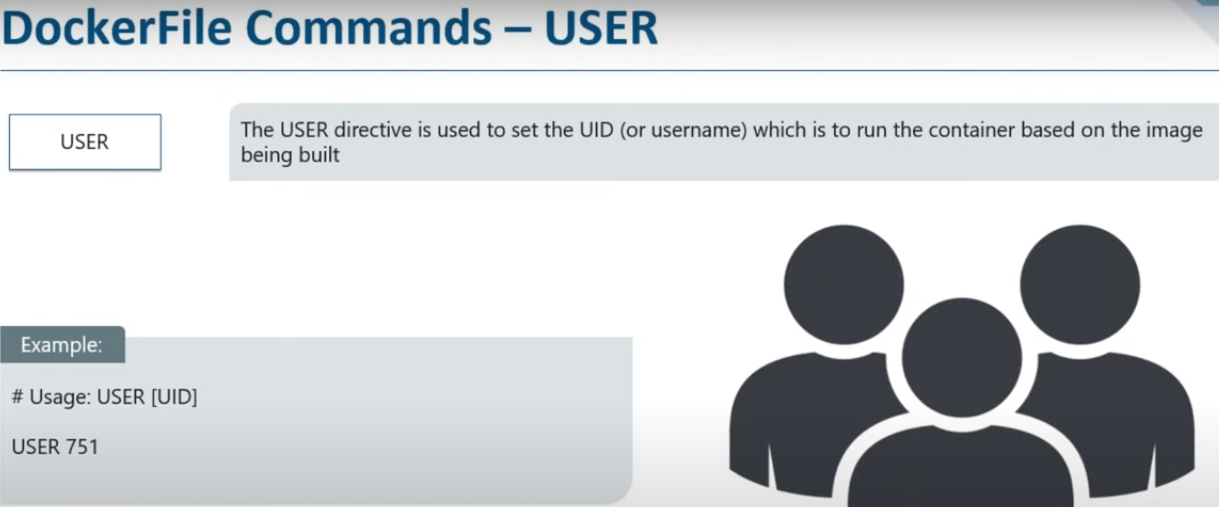


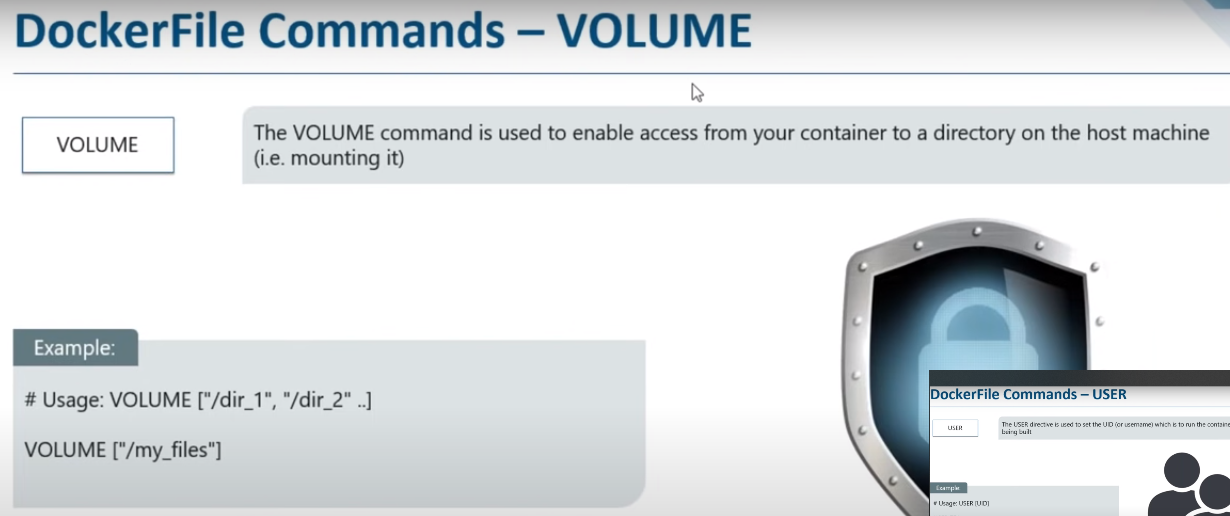


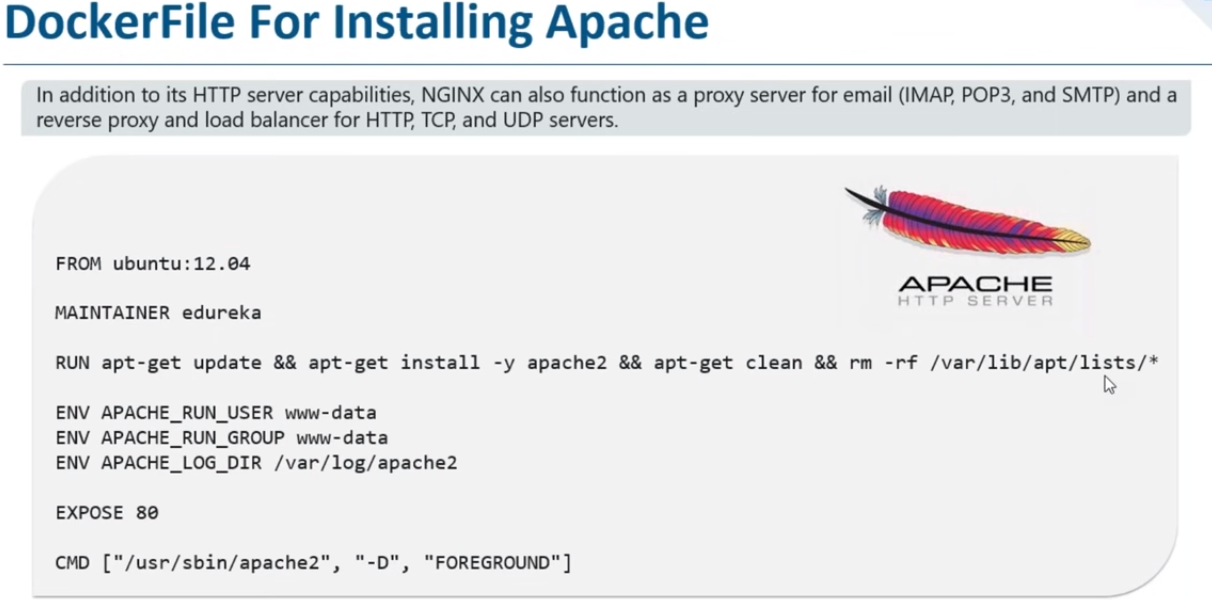












FROM ubuntu 12.04

MAINTAINER ram

RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm rf /var/lib/apt/lists/\*

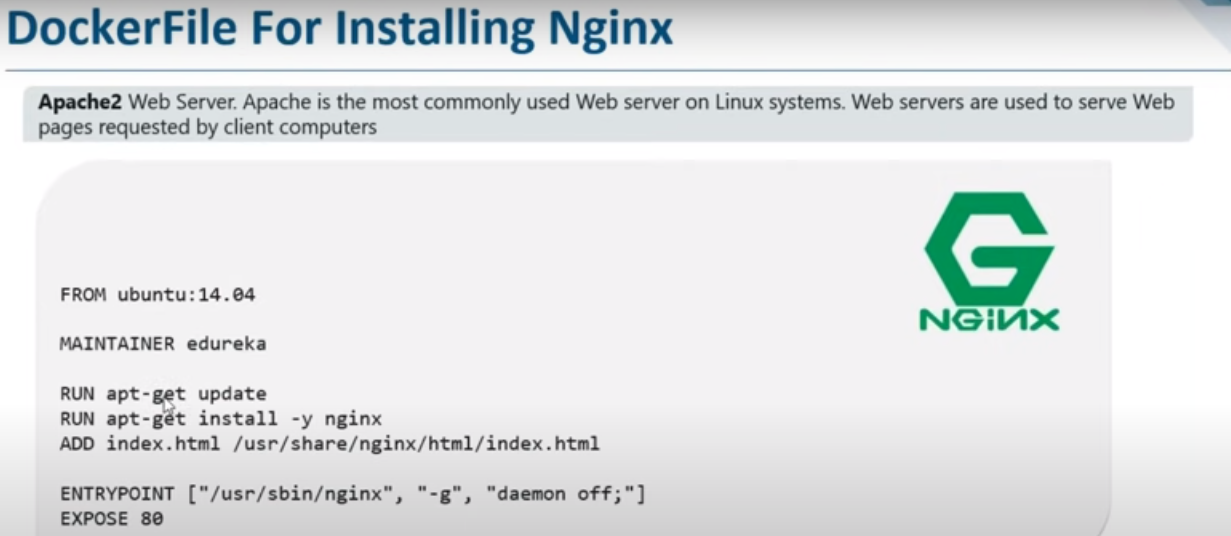
ENV APACHE\_RUN\_USER www-data

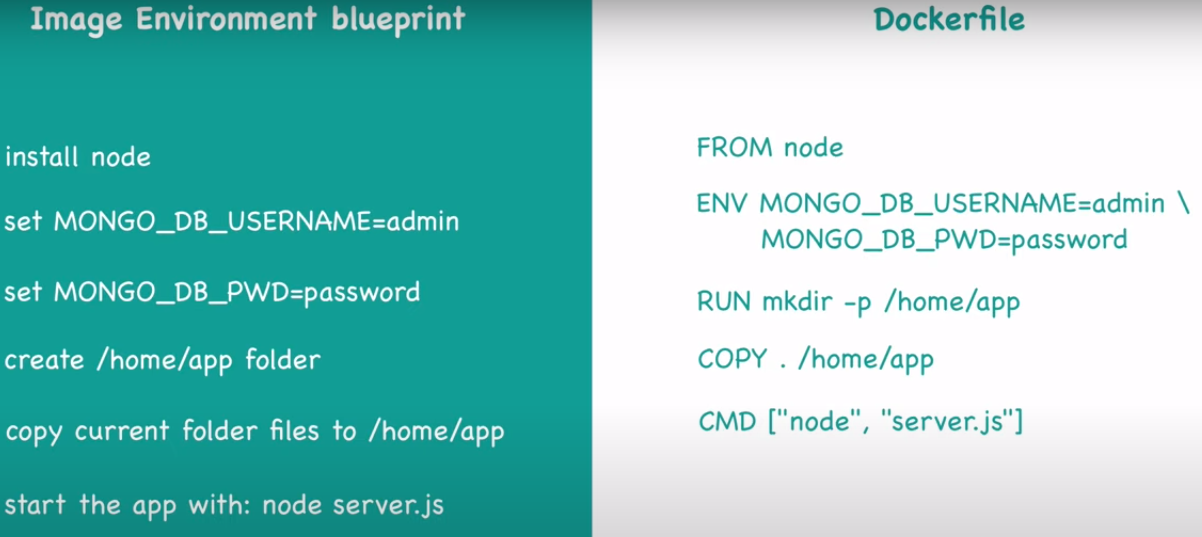
APACHE\_RUN\_GROUP www-data

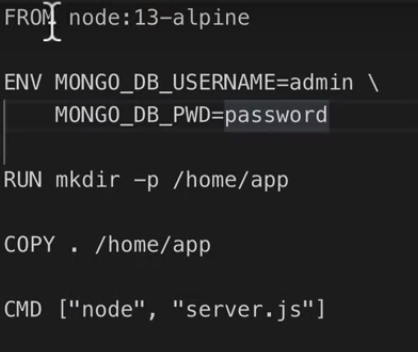
ENV APACHE\_LOG\_DIR /var/log/apache2

EXPOSE 80

CMD ["/user/sbin/apache2", "-D", "FOREGROUND"]







FROM node:13-alpine

ENV MONGO\_DB\_USERNAME=admin \

MONGO\_DB\_PWD=passwd

RUN mkdir –p /home/app

COPY . /home/app

CMD [“node”,’server.js”]

**DOCKERFILE COMMADS**

**1.**FROM: 2.WORKDIR 3.COPY 4.RUN 5.EXPOSE 6.CMD 7.ADD 8.ENV 9.ENTRY POINT