docker logs , docker stats , docker history(shows you layers)..
touch .dockerignore
open the file from the navigator and type inside it:
[b|B]in //b small or capital
[o|O]bj //o small or capital

That will ignore any bin or obj file

what is docker multi stage and when to use it ?
Docker multi-stage builds are a feature in Docker that allow you to create more efficient and smaller Docker images by using multiple stages or phases in your Dockerfile. This is particularly useful when building complex applications or services where you need certain tools or dependencies to build your application, but you don't want those tools and dependencies to be included in the final production image. Multi-stage builds help reduce the size of your Docker images and make them more secure and efficient.

Here's how Docker multi-stage builds work and when to use them:

1. **Multiple Stages**: In a multi-stage build, you define multiple "stages" in your Dockerfile. Each stage represents a separate build environment with its own set of instructions and base image. Each stage can be used to perform a specific task or to build a specific part of your application.
2. **Intermediate Images**: The intermediate images created in each stage are discarded after that stage is completed, except for the final stage. This means that you can use one stage to build your application and another stage to package it for production, without carrying over unnecessary files or dependencies.
3. **Reduced Image Size**: By using multi-stage builds, you can significantly reduce the size of your final Docker image. The final image only contains what you copy or extract from the intermediate stages, which typically includes only the application binary and required runtime dependencies, excluding build tools and artifacts.

When to use Docker multi-stage builds:

1. **Compile and Build Steps**: When you need to compile your application or build artifacts during the Docker image build process, but you don't want to include the build tools or intermediate files in the final image. For example, you may build a Go, Java, or C++ application within one stage and then copy the resulting binary to the final image.
2. **Multi-Component Applications**: When your application consists of multiple components (e.g., a front-end and a back-end), and you want to build them separately in their own stages before combining them into a final image.
3. **Dependency Management**: When you need to fetch or install dependencies during the build process but want to avoid including the package manager and downloaded files in the final image.
4. **Security and Efficiency**: When you want to improve the security and efficiency of your Docker images by minimizing their size and reducing the attack surface. Smaller images are faster to transfer and deploy, and they have fewer vulnerabilities.

Here's a simplified example of a Dockerfile using multi-stage builds for a Node.js application:

Dockerfile

```
# Stage 1: Build the application
FROM node:14 AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: Create the production image
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Notes on above code:
`COPY --from=build /app/build /usr/share/nginx/html`: This line copies the build artifacts from the "build" stage (the first stage) into the `/usr/share/nginx/html` directory in the current stage. The build artifacts typically include the static files generated by your Node.js application, such as HTML, CSS, JavaScript, and other assets.

In this example, the first stage builds the Node.js application and its dependencies, while the second stage creates the final production image using only the built artifacts from the first stage. This results in a smaller and more efficient Docker image for running the application in a production environment.

Difference between CMD and ENTRYPOINT:
ENTRYPOINT (The Oven's Default Function):

- `ENTRYPOINT` is like the default function of a microwave oven, which is heating food. It's what the oven primarily does.
- You set the default operation of the oven. For a microwave, this is heating food.
- In a Dockerfile, `ENTRYPOINT` sets the primary command that your container will run when it starts.

Example Dockerfile:

Dockerfile

```
ENTRYPOINT ["python", "app.py"]
```

In this analogy, the oven's default function is to heat, and the container's default function is to run `python app.py`.

CMD (Providing Optional Instructions):

- `CMD` is like providing additional instructions for the microwave's default operation. You can add settings like power level and cooking time, but you don't change the fundamental heating function.
- In a Dockerfile, `CMD` provides default arguments that can be overridden when you use the container.

Example Dockerfile:
Dockerfile

```
CMD ["--power=50%", "--time=2 minutes"]
```

In this analogy, you're providing default settings for the microwave's default heating operation.

Combining ENTRYPOINT and CMD (Customizing the Operation):

- Combining `ENTRYPOINT` and `CMD` is like having a microwave oven with its default heating function but allowing you to customize the settings for each use.
- You set the primary function (heating) with `ENTRYPOINT` and then use `CMD` to provide options that can be changed based on what you're cooking.

Example Dockerfile:
Dockerfile

```
ENTRYPOINT ["microwave"]

CMD ["--power=50%", "--time=2 minutes"]
```

1. In this analogy, the oven always heats (`ENTRYPOINT`), but you can adjust the power and time (`CMD`) depending on what you want to cook.

So, ENTRYPOINT is like the primary function, and CMD provides optional customization. When you run a Docker container, you can specify additional instructions (arguments) that override the defaults provided by CMD, just as you can adjust the settings on a microwave oven each time you use it.

PEOPLE OFTEN USE CMD THAN ENTRYPOINT BECAUSE IT IS MORE FLEXIBLE

========
docker

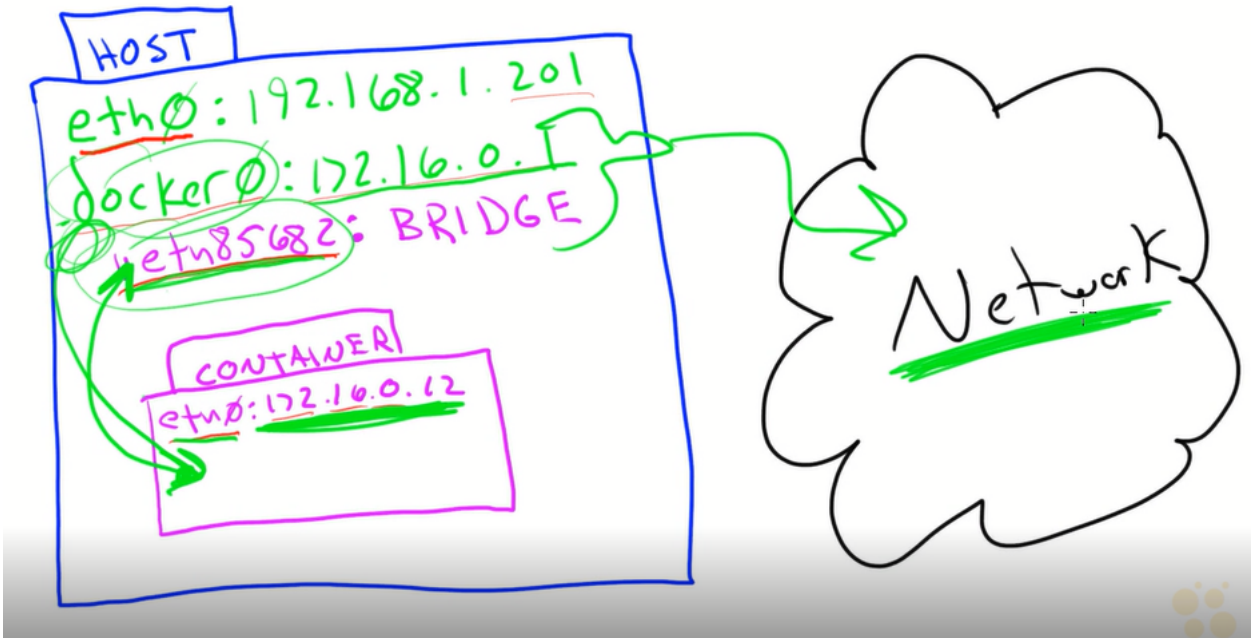
This bash script app counts from 1 to infinity:
/bin/bash -c 'while true;do X=$[$X+1]; echo $X; sleep 1;done'

To contain this script app in a docker container:
sudo docker run -d centos /bin/bash -c 'while true;do X=$[$X+1]; echo $X; sleep 1;done'
—

Docker networking:

About virtual interfaces,bridged networking,Finding IP of container.

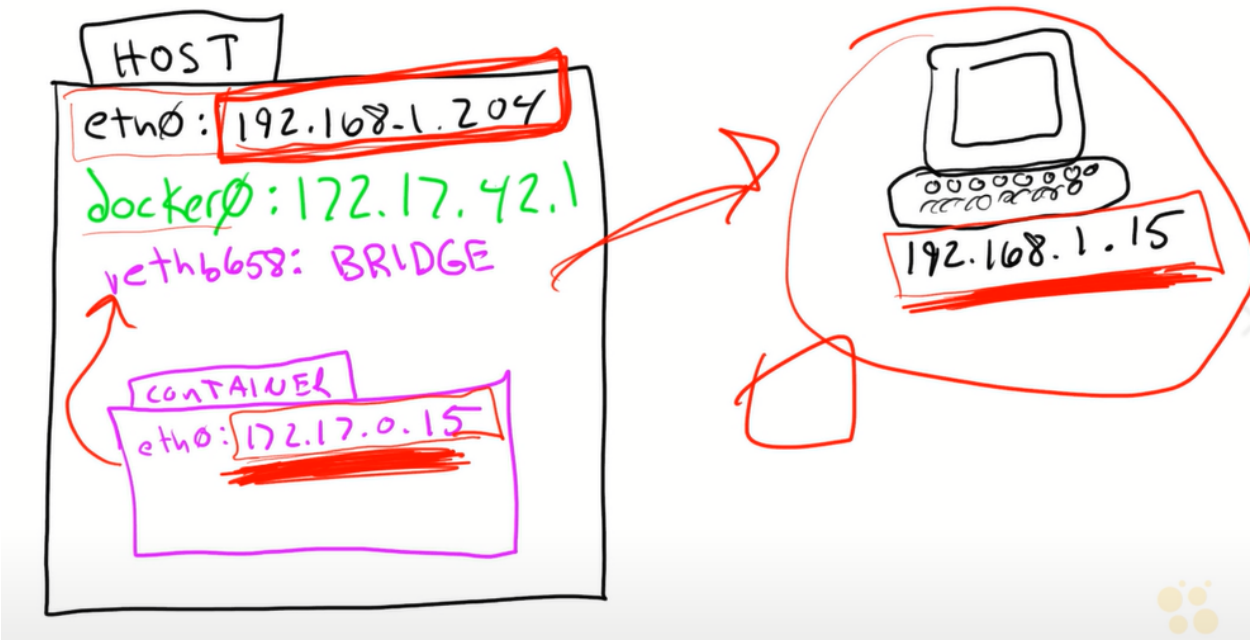
eth0 is by default on my linux that is your normal host address, A docker0 is virtual interface created on installation"a private IP". When a contatiner is created it will have its own eth0 address. The virtual eth0 on the container creates a virtual bridge on the local machine with name veth(same address).



Advanced Networking:

Port mapping,port redirection,host networking.

Computer outside can not directly talk to the container as they are totally different IP'S. so we need to map port the address.



Port mapping vs port redirection vs host networking in docker

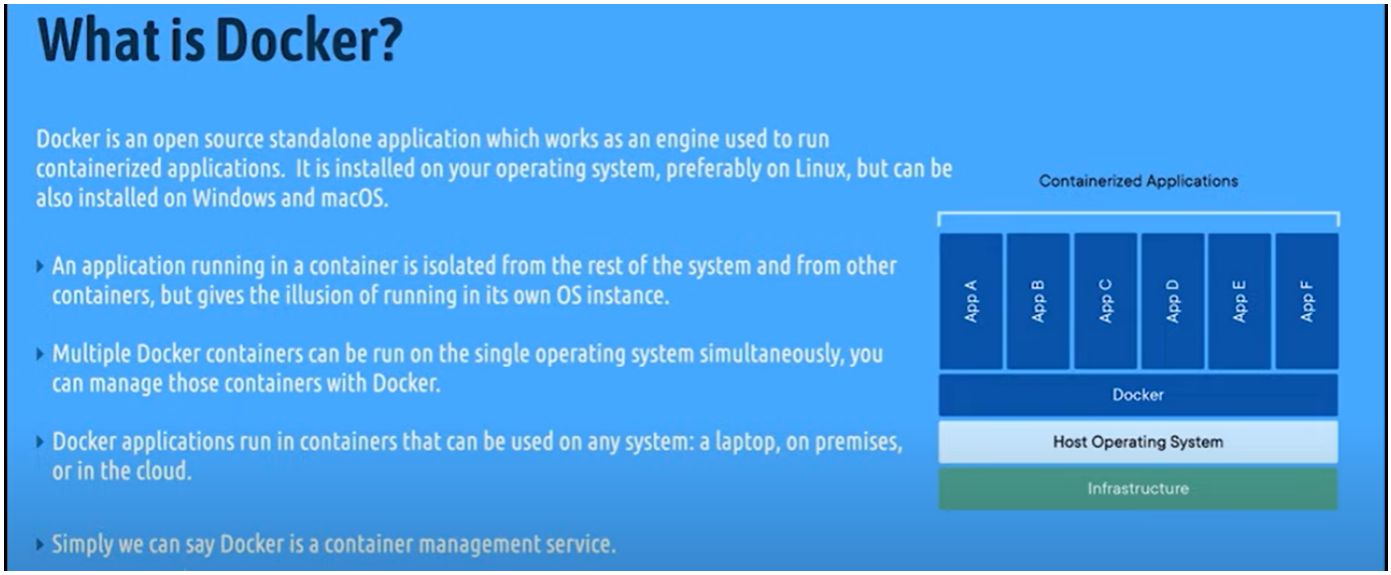
docker run -d -p 8080:80 nginx //port mapping you assign port

docker run -d -P nginx //port redirection , port assigned dynamically

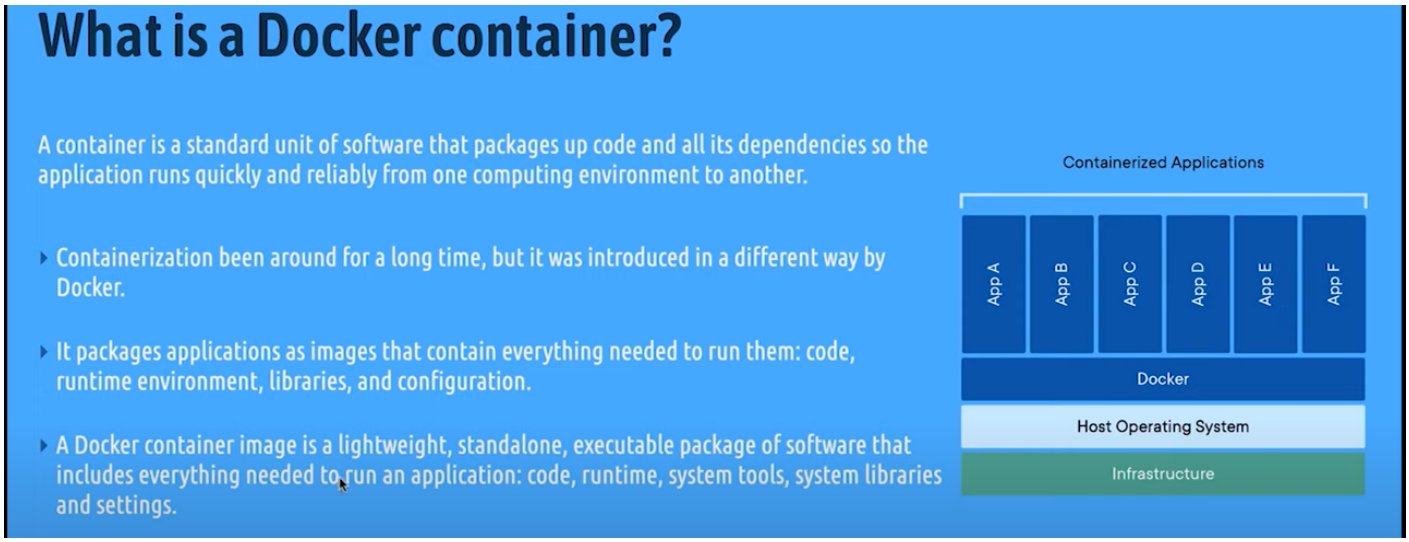
docker run -d --network host nginx //Host networking, shares network with localhost

Docker:

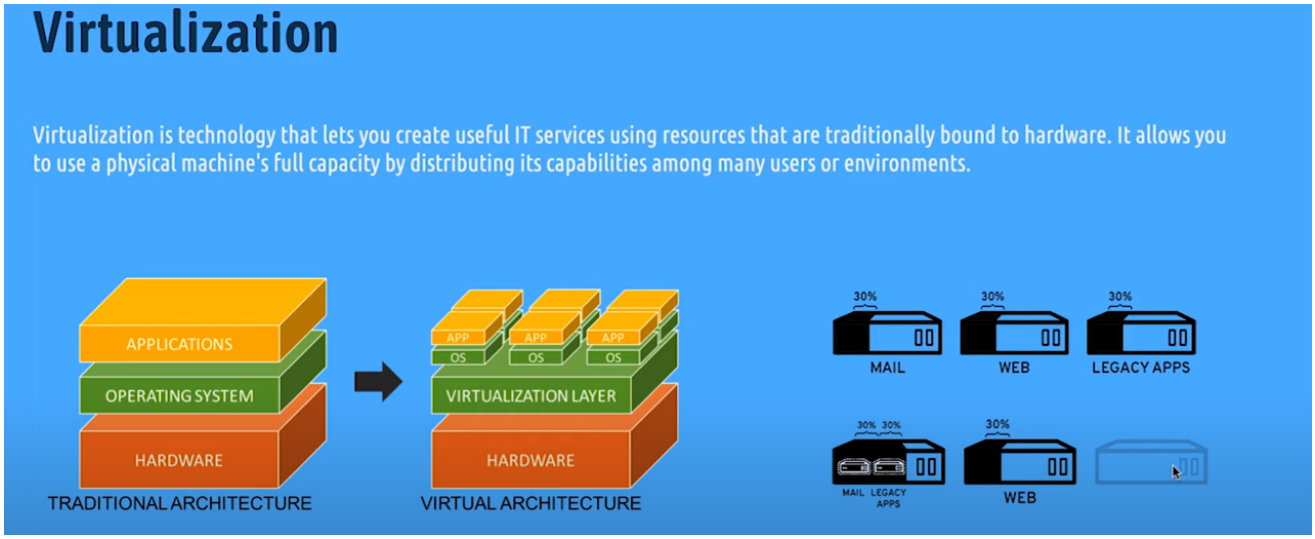
What is docker ?
What is a container ?
What is virtualization ?
What is hypervisor ?
Containerization vs virtualization ?
What is docker for ?
A.



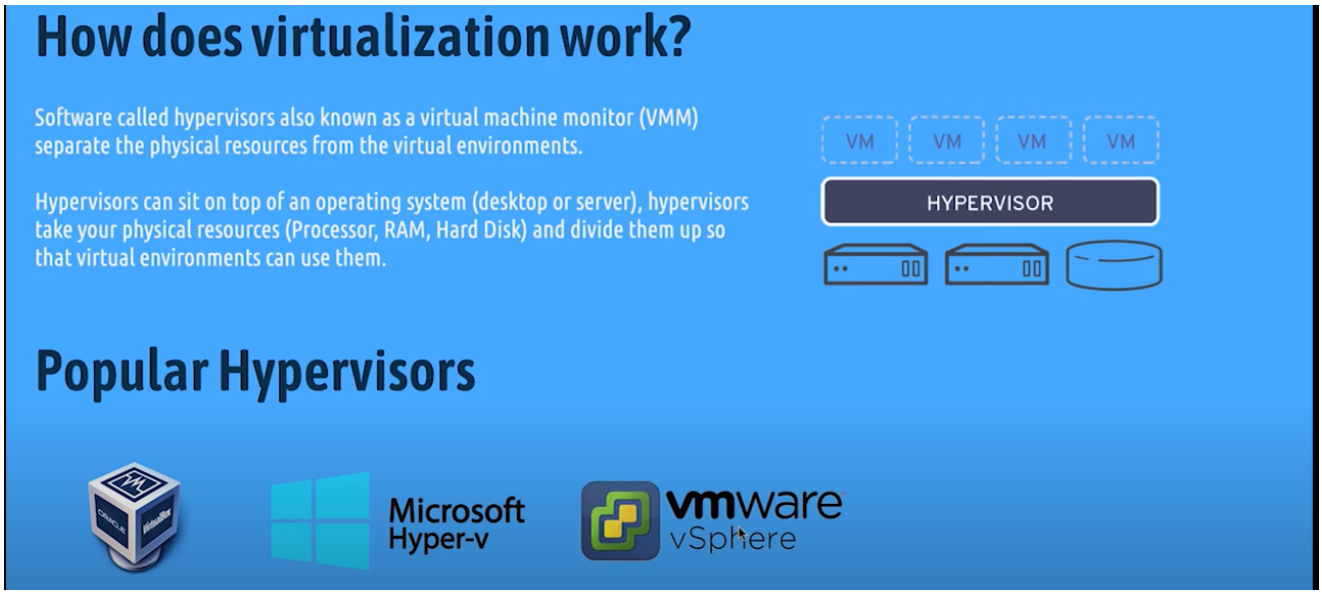
B.



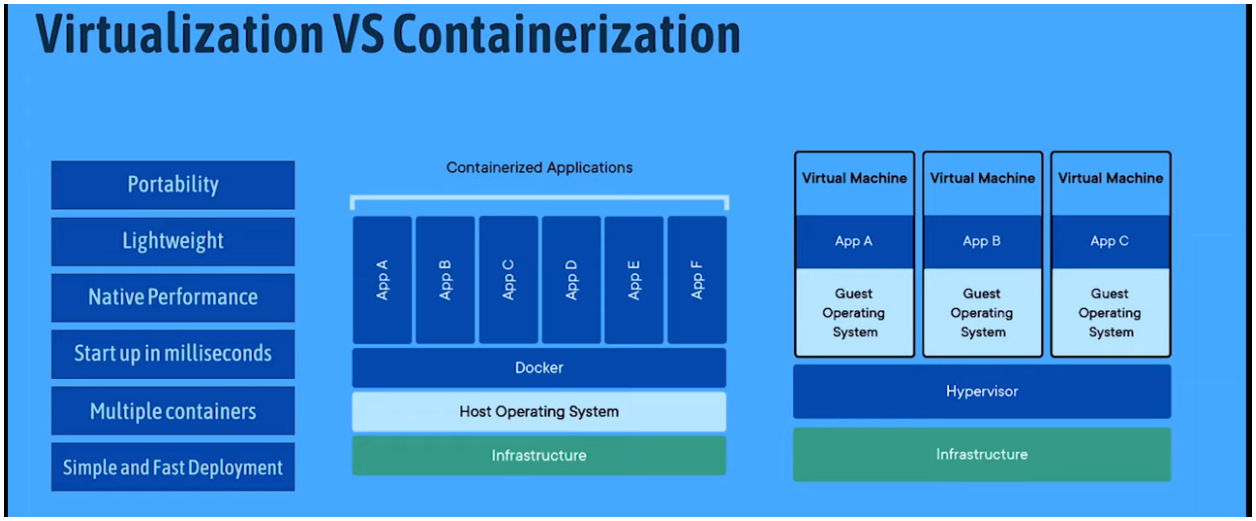
C.


Virtualization
Virtualization is technology that lets you create useful IT services using resources that are traditionally bound to hardware. It allows you to use a physical machine's full capacity by distributing its capabilities among many users or environments.
APPLICATIONS
OPERATING SYSTEM
HARDWARE
TRADITIONAL ARCHITECTURE
APP
OS
VIRTUALIZATION LAYER
HARDWARE
VIRTUAL ARCHITECTURE
30%
MAIL
WEB
LEGACY APPS
LEGACY APPS
WEB

D.

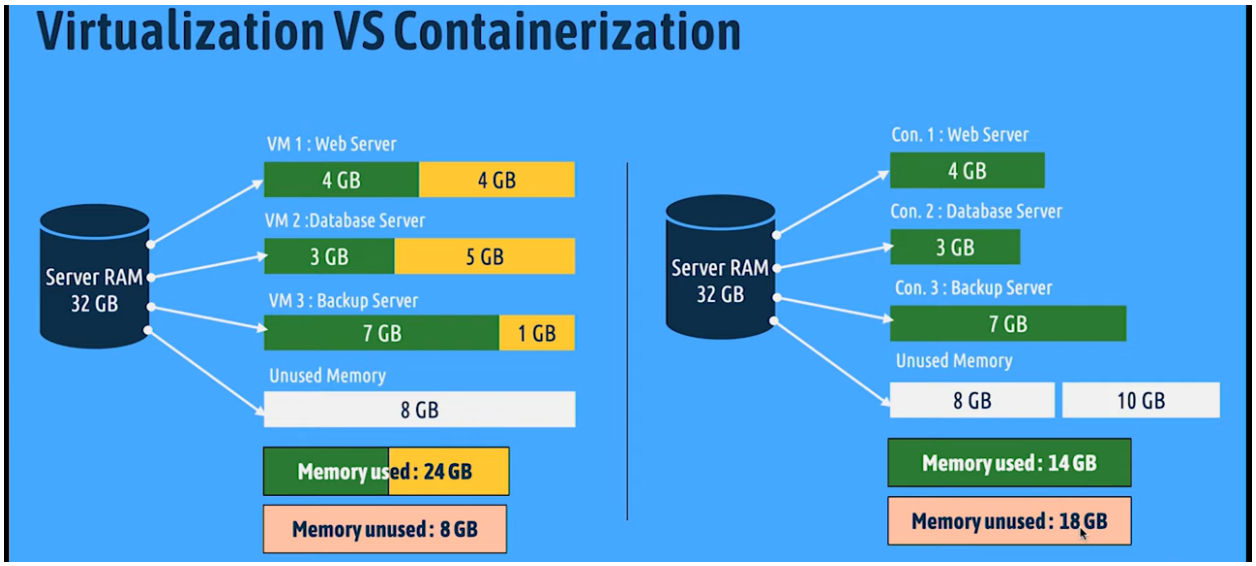

How does virtualization work?
Software called hypervisors also known as a virtual machine monitor (VMM) separate the physical resources from the virtual environments.
Hypervisors can sit on top of an operating system (desktop or server), hypervisors take your physical resources (Processor, RAM, Hard Disk) and divide them up so that virtual environments can use them.
VM
VM
VM
VM
HYPERVISOR
Popular Hypervisors
Microsoft
Hyper-v
vmware
vSphere

E.

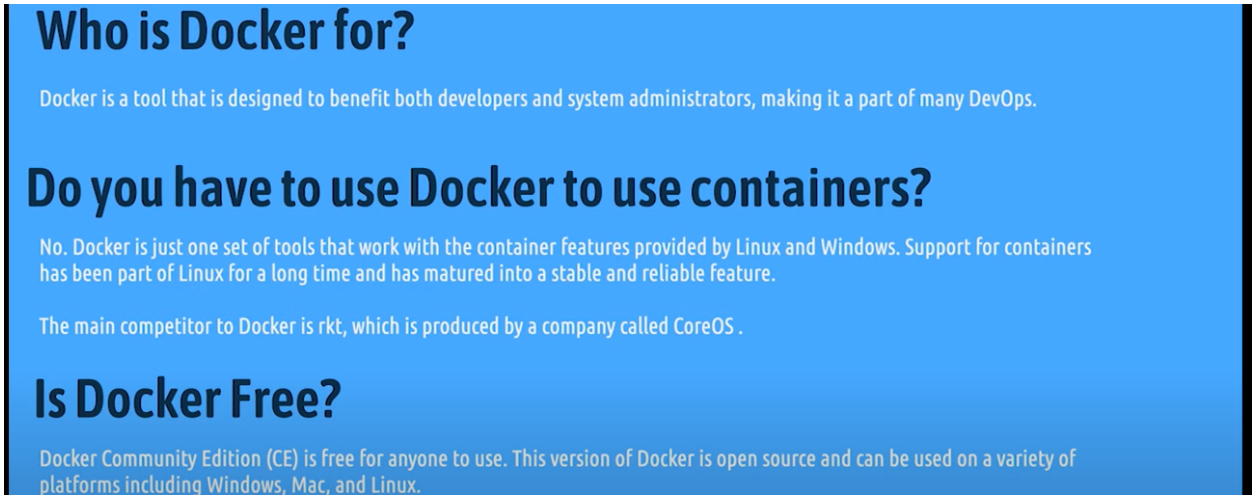


Adv of containers:Portability means that it can be moved easily as it have images.
Also it is lightweight and Native performance as in vms it closes completely when you reach maximum size until you modify it, while in container you can set it to automatically expand size for example.

F.

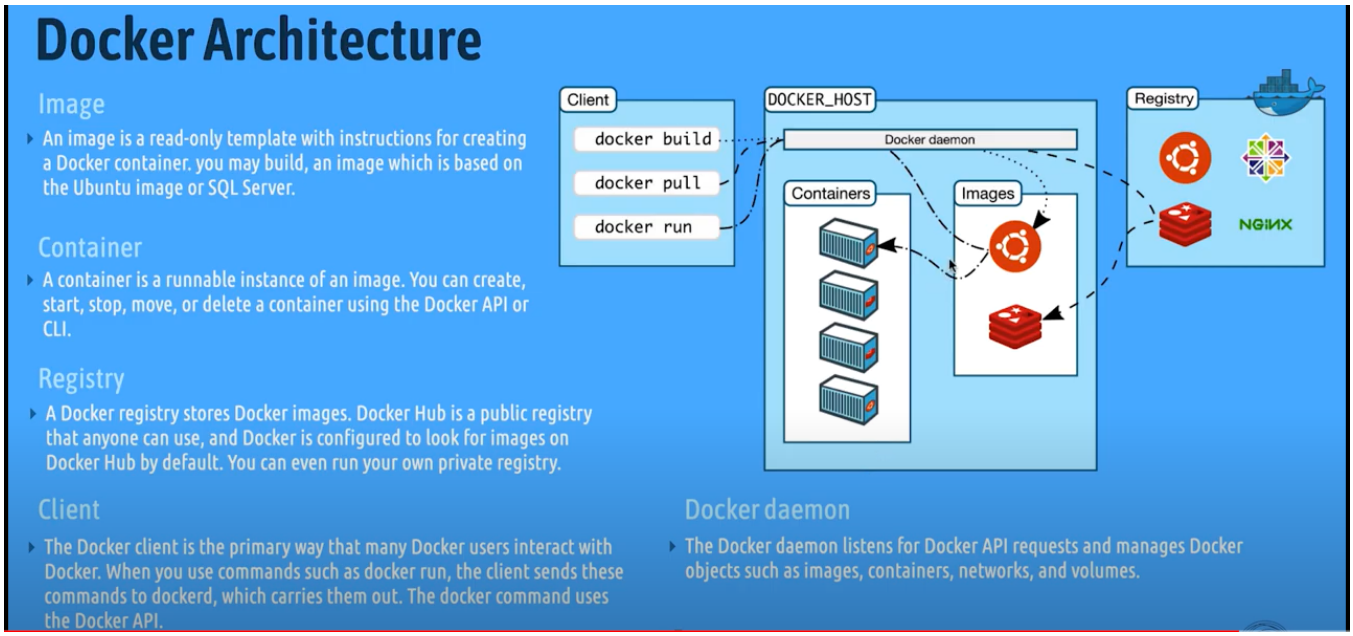


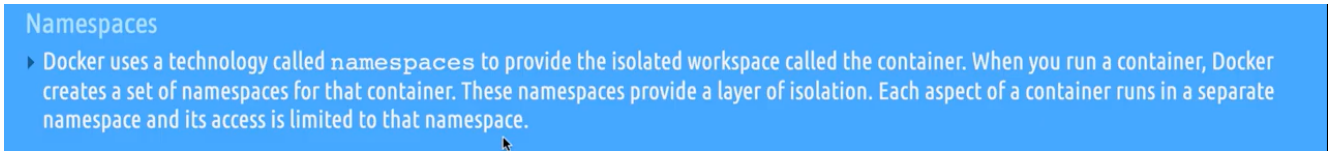
G.



Part TWO:

What is a Docker image ?
What is a Docker container ?
What is a Docker registry ?
What is a Docker client ?
What is a Docker daemon ?
What is a Docker namespace ?





Part THREE:

Docker installation:

1.Go to docker.com – Get started – Docker Desktop -- Windows
2.Install it from downloads
3.To check it is installed correctly – open powershell: docker version

Part Four:

Docker Settings:

Open docker desktop – You will see containers moving until it starts in task bar – right click on it “about” to check version ,,same as in powershell **docker version**.
right click on it –settings—General:check Start on desktop login. Auto update.
Resources:You can modify it if needed.
Command Line:Enable experimental features. To enable features that are not completely legit.
Kubernetes: For later on. By the way it orchestrates and manages your containers all together.

Right click on it switch to windows container: By default it is linux and it is better to leave it on linux.

Right click on it documentations for tutorials.

Right click on it docker hub to login to docker hub. Docker CE “Community edition”, EE “Enterprise edition”

Right click on it –Dashboard – To check all your running containers.

Checkbox of seetings general →

Expose daemon on tcp://localhost:2375 without TLS TO BE CHECKED

Part Five:

Basic Docker Commands:

from docker hub – Hello world by docker official images.
On powershell:

docker run hello-world	**//It will check your pc then goes auto to docker hub**
docker ps -a	**//To only containers you have**
docker container ls -a	**//Shows container and image you have**

-a means “all”

//If you did right click and check the dashboard you will find the container and the image itself,
You can also manage the container from here also from the dashboard.

--Centos and fedora are dist of linux.
on powershell:
docker run fedora

docker container ls -a	//You will find the container for fedora.
Docker images	//Will show you all images you have

--Docker container shows containers only and images show images only.

Docker rm 55c1j3reewf	//container will be removed
docker image rm fedora	//To remove image itself

Part SIX:

We will work with redis it is a very light database.

Docker container run redis //same as docker run,but it is now more clear
After we did run it we can not type anything in the powershell and if window closed container will close

Solution:

Docker container run -d redis //-d for detach
docker container ls //You will find it running
docker run redis:alpine //this is different version than previous redis
docker run -d redis:alpine
docker container ls //They are 2 different images
docker image ls //Image ID is also different

Docker container run -d redis
docker image ls //It will be the same as before as this image already existed

---

Docker container ls //Now we have 3 containers
docker image ls //Same as before two images alpine and normal one

Docker inspect redis //It will provide you with info on this image

Docker inspect redis:alpine

docker logs d82dufcs7 //container ID is mentioned, will show you all logs for it.start-end-connect etc..
or
docker **logs** optimistic_wescoff //name of container

docker **stats** optimistic_wescoff //**How much resources** it is taking from your PC

docker **info** //**will show you all containers running and images ,who is on and who is off**
and //cpus and total memory,you can check this from the interface app “logs,inspect and stats”

To get IP of container:
docker inspect -f "{{ .NetworkSettings.IPAddress }}" charming_pascal

for linux will be: **docker inspect charming_pascal | grep '"IPAddress"' | head -n 1**

//It will output the IP of this container

Part Seven:
Nginx is a web server and load balancer.
We will get nginx from docker hub.

In powershell:
docker conatainer ls //No containers running
docker container run –detach –publish 80:80 –name n1 nginx
//publish to use this port and expose it on browser,gave it name of n1

Open any browser and type localhost “Welcome to nginx”

Docker container stop n1 //Now it will not open in the browser
docker container start n1

docker inspect nginx //You will find exposed ports 70/tcp and 80/tcp
//They are already configured in the image to be used.

docker inspect d04 //It is for sql image and have exposed ports:1433/tcp
//you can here publish on 1433 for example

We will give commands to the nginx itself –as it have a bash inside it.
docker exec -it n1 bash //execute container name and command
//Now you can type inside the bash of the nginx
examples written in bash of nginx:
service nginx status
service nginx reload
service nginx stop

docker help run //Will give you every letter stands for what -I -t etc…

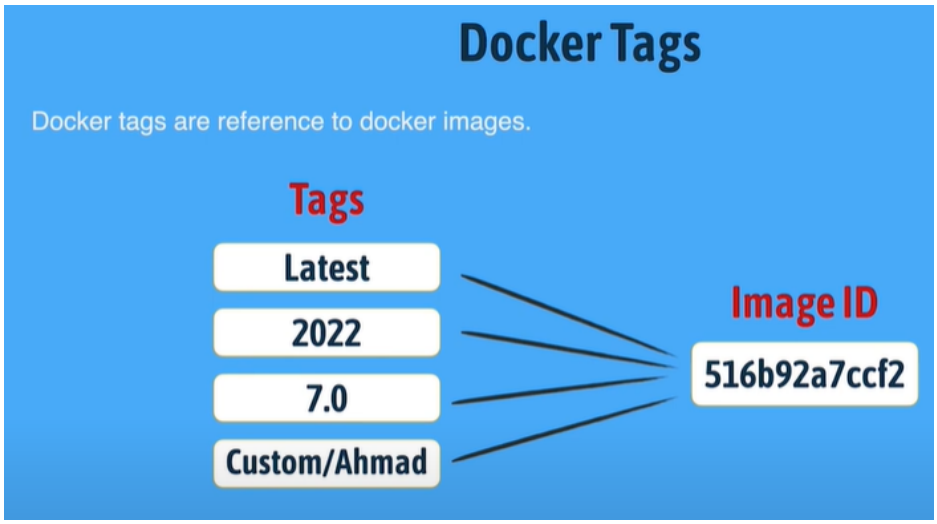
docker logs n1 //Will show you everything happened to nginx

PART EIGHT:

How to deal with files and folders:
docker container exec -it n1 bash
//Typing now in bash of nginx
cd etc
cd nginx

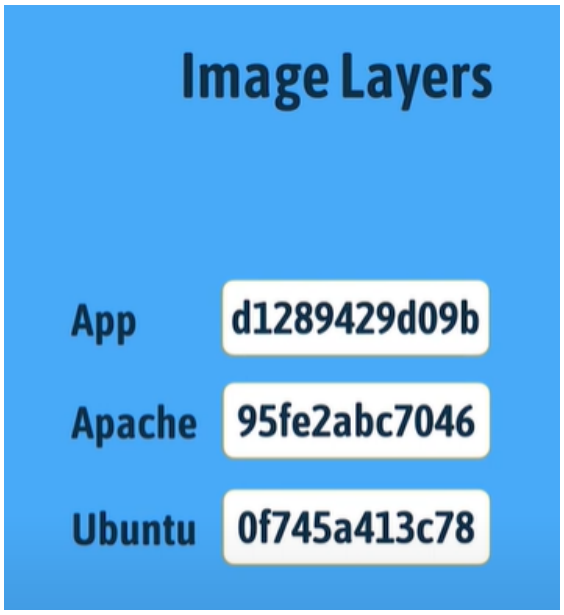
ls
cd .. //To return back
cd / //To home
mkdir MyFiles
mkdir content
touch default.html
touch style.css
cp default.html content
rm default.html

PART NINE:
Tags, Image layers, DockerFile.

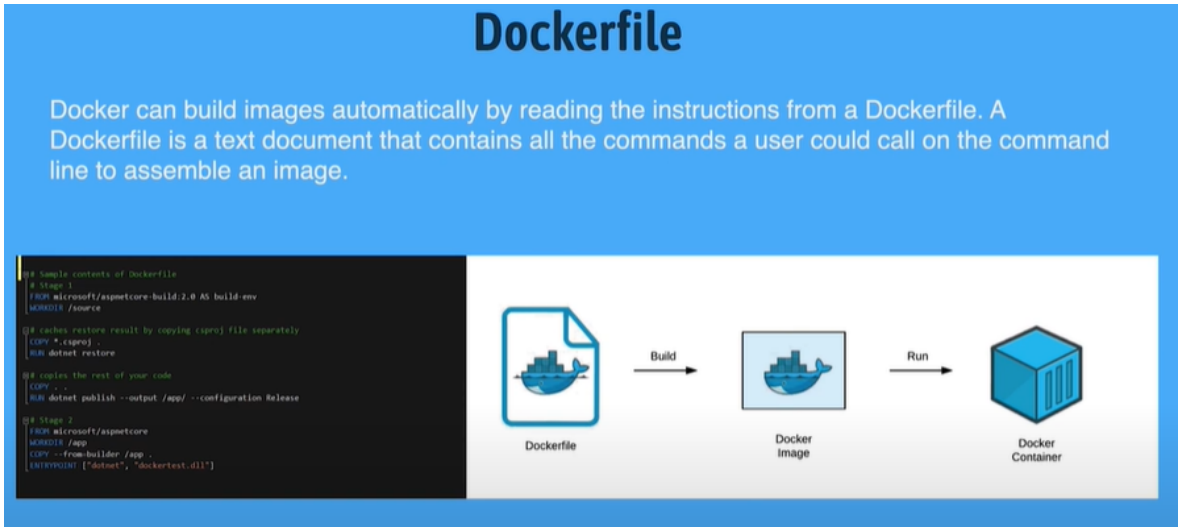


Latest does not mean the last, may mean default. You can call image with tag instead of ID.

Any change done on an image is considered a layer.
And the layer can be considered an image alone.



The above can be combined using a dockerfile to a single image.



On docker hub search for redis latest for example click on layers and you will see the commands written in the dockerfile.

Part TEN:

Docker history redis **//shows you the layers**
docker history nginx

We will create custom tag and upload it to docker hub:
docker tag //It is as help
//create docker hub account first:abdelrahmanmagdy1997 is my repo name
docker tag redis abdelrahmanmagdy1997/redis
docker image ls
//You will find abdelrahmanmagdy1997/redis and it will take tag **"latest"**

docker tag redis abdelrahmanmagdy1997/redis:devabdo
//Now it will take the tag name**"devabdo"** – This one and previous one will point to same image

docker push abdelrahmanmagdy1997/redis
//denied access

Solution:You need to login first.
docker login //Then enter your username and password
docker push abdelrahmanmagdy1997/redis

Now go to docker hub to your repo and you will find it there.

docker push abdelrahmanmagdy1997/redis:devabdo //It will be uploaded but will point to same
//image ID

docker logout //To remove credentials from powershell to dockerhub

---

Part ELEVEN:
Building docker image from Docker File.
mkdir dockerimages
using vscode.—New file:

FROM alpine //Here you put Operating system image -alpine is linux
CMD ["echo","Hello Abdo,I am custom image"] //cmd ["executable"."parameter"]

//Click on plain text in bottom right and click that it is a docker file

Terminal New terminal

cd /
cd dockerimages
docker build . //Dot to be build here, you can type another path to save in
docker image ls //It will have name none
docker container run 8we2e3333 //Hello Abdo,I am custom image

Note : To make it have a name:
docker build –tag abdo . //From the beginning on the build

#base image :alpine used for small size //Comments
FROM alpine
#packages or dep
RUN apk add –update redis //After run comes a command.apk is package manager
#commands to run on image
CMD [“redis-server”]

//on terminal
docker build –tag our_redis c:\dockerimages
docker image ls
docker container run our_redis
docker container ls

0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

PART TWELVE:
Dockerize ASP.NET core app.
we will need to install .net core 5 sdk from dotnet.microsoft – Run and install.

creating project 🙂–Vscode –terminal –powershell
cd /
mkdir dockerize
cd dockerize
dotnet new mvc –name hrapp –output dockerhrapp //.net core commands to create project
//name of app is hr app and output is name of folder,mvc is your design pattern

Cd dockerhrapp
ls //you will find root , views,controllers etc…..

on vscode
open folder dockerize Hrapp
in views—home—index – type your comment
<h1 class="display-4">Welcome abdooooo</h1>

In terminal"powershell":
dotnet build
ls
touch Dockerfile //So it will be created between project files

open docker file itself from navigation menu on the left in vscode and type inside it:

```
#First-stage base image
FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /source
#copy csproject and restore dependencies
COPY *.csproj .
RUN dotnet restore
#copy and publish application file
COPY . .
RUN dotnet publish -c release -o /app
#final stage image
FROM mcr.microsoft.com/dotnet/asp.net:5.0
WORKDIR /app
COPY –from=build /app .

ENTRYPOINT ["dotnet","hrapp.dll"]
```

NOTES ON ABOVE:
LINE 1 : Went to .NET page on docker hub –featured repos--.net sdk – We used version 5 "just take the link" to the base image
LINE 2: Set working directory on the previous image"sdk" that you will work from. Add,copy,run etc…
LINE 3: Copy any file csproj from here to the workdirect which is /source
LINE 4:run inside the image a command called dotnet restore, it will get any dependency needed for the project
LINE 5:Copy everything
LINE 6:Run dotnenet publish -c to publish release as it is finished it may be debug if it is not ready and will be tested first, -o for where to output
LINE 7: We will get ASP.NET CORE runtime image from .NET page also to be able to run project
LINE 8:Workdir for that image

LINE 9: copy all of the above work in previous container into the app folder
LINE 10:which app will run first

0000000000000
We will create a dockerignore file for the above app.
It tries to minimize size app as much as possible. When it is being build which files are not needed.

Terminal New terminal
touch .dockerignore
open the file from the navigator and type inside it:
[b|B]in //b small or capital
[o|O]bj //o small or capital

That will ignore any bin or obj file

Now we will build the image then run it.
Stand on workdirectory of the dockerhrapp which is inside dockerize and go to powershell terminal:
docker build –tag dockerhrapp .

docker image ls //you will find dockerhrapp
docker run -d -p 8080:80 –name hrapp dockerhrapp .

Open browser on localhost:8080 opened on browser 🙂

You can edit the index.cshtml file “Heloo abdo, edited” and save

powershell terminal:
dotnet build
//stop previous container first
docker container ls
docker container rm -f hrapp
docker build –tag dockerhrapp . //build image again
docker run -d -p 8080:80 –name hrapp dockerhrapp .

Open browser on localhost:8080 opened on browser 🙂

PART THIRTEEN:
Dockerize Angular Application.

First download nodejs from nodejs website “LTS”{Long term support } and install it.
When installing choose npm package manager.

Vscode terminal –new terminal PS:
node -v //To make sure it is installed
nopm -v //To make sure it is installed

TO INSTALL ANGULAR:
npm install -g @angular/cli //To install angular
ng –version //To make sure it is installed

Cd /
mkdir dockerizeangular
cd dockerizeangular

Create new angular app.
ng new helpdesk //New app with name helpdesk
would you like to add angular routing ? n
which stylesheet format would you like to use ? css

cd helpdesk
ng serve //compile success for the app
//open your browser with the given URL and you will find it running

In vs code—open folder—helpdesk
open terminal again and type ng serve
as changing opened folders closed the terminal

navigate to src—app—app.component.html
title app is running replace it with abdo is running
As you save it reflects directly to the app

Close terminal node and go to powershell again
touch dockerfile

on docker hub website-node official image- node alpine 3.4
open dockerfile from navigator itself:

```
#First stage of building angular image
FROM node:alpine3.14 as build
RUN mkdir -p /app                                    //-p if the folder exist it does not create it
WORKDIR /app
COPY package.json /app/
RUN npm install

COPY . /app/
RUN npm run build –prod

#final stage
FROM nginx:alpine                                    //nginx docker hub
COPY –from=build /app/dist/helpdesk /usr/share/nginx/html
//The above copies the file from helpdesk to the distination of nginx
```

00000000000000 in helpdesk folder
touch .dockerignore

open the file itself and type:
node_modules
npm-debug.log
.git
.gitignore

000000 close node app and only run from docker
docker build –tag helpdeskapp .

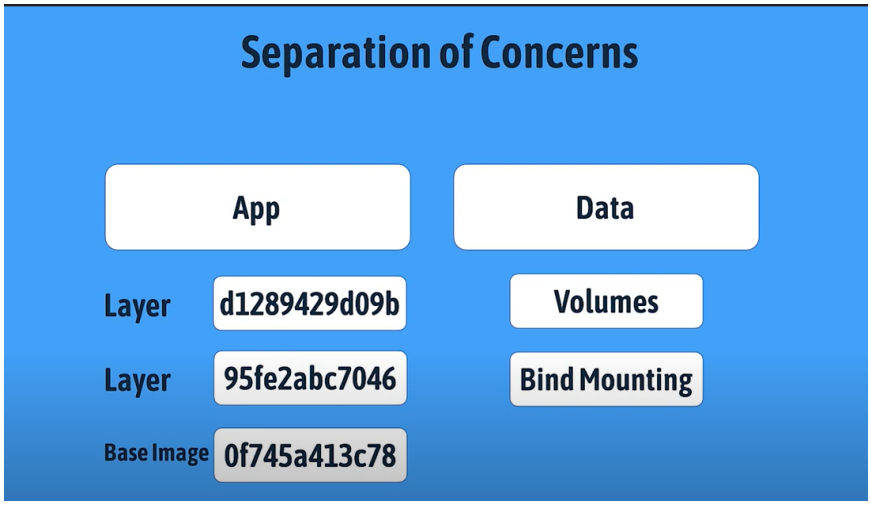
docker image ls   //helpdeskapp
docker run -d -p 4200:80 –name helpdesk helpdeskapp

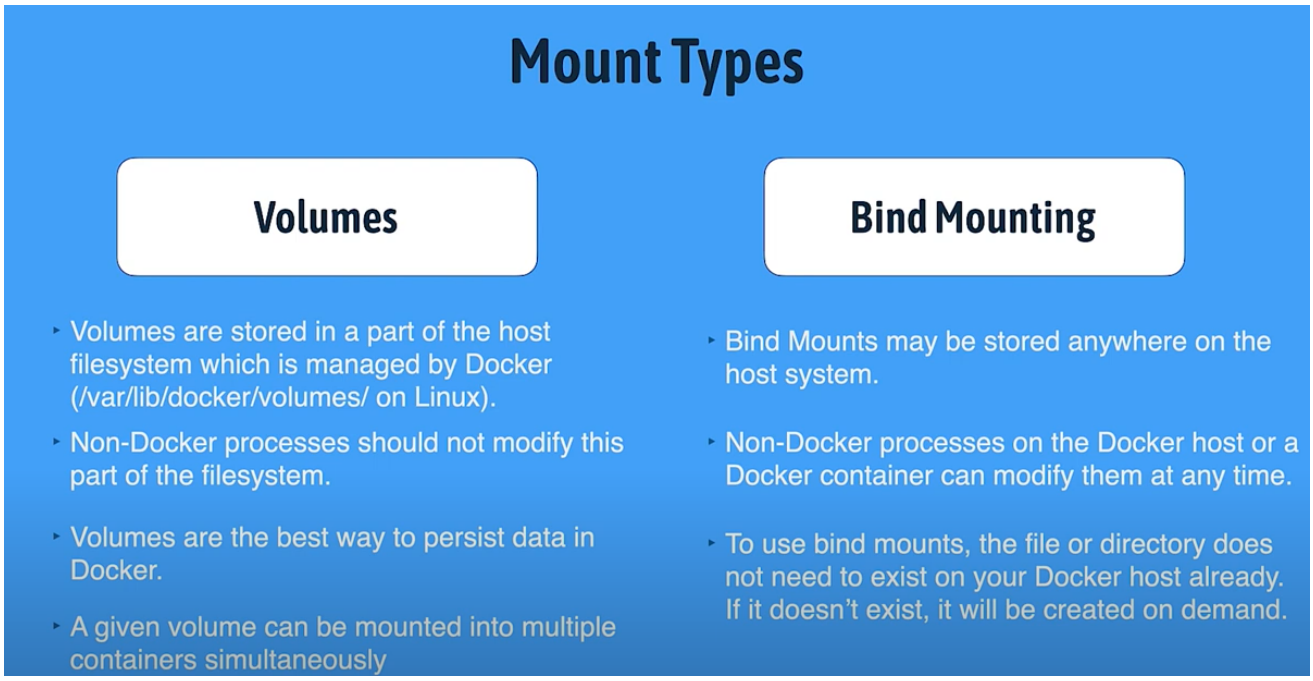
on browser localhost:4200

docker stop helpdesk

PART FOURTEEN:
Data Management in DOCKER:

Once a container is deleted all data is lost.
maybe users are uploading data to that container.
For data you can choose to work with volumes or work with bind mounting.





PART FIFTEEN:
Docker Volumes:

docker inspect redis //You will find that there is volumes /data where data is inside in this image

docker pull mariadb
docker inspect mariadb //volumes:/var/lib/mysql where data is inside in this img

docker run -d –name mymariadb -e MARIADB_ROOT_PASSWORD=1234 mariadb //password set

docker container ls //mymariadb is running

docker inspect mymariadb //Mounts:type=”volumes” and is taking name =**a big number** and source is /var/lib/docker/volumes and destination is /var/lib/mysql

/////**Source means on my host where it is “linux containers” using hyper-v or WSL , Destination is on container where is it**

docker volume ls //list all volumes available “you can not understand it”
So we will give volume a name to be friendly

docker run -d –name mariadbnew -e MARIADB_ROOT_PASSWORD=1234 **-v mariavolume:/var/lib/mysql mariadb**

BEFORE THE : IS THE NAME OF THE SOURCE THAT WILL BE SET AND AFTER THE : IS THE DESTINATION TO BE SET

Notes: -v mariavolume:/var/lib/mysql This part of the above command changes the name of the volume -v means give the volume a name and mariavolume will be the name for volume instead of any random number in var/lib/mysql

docker container ls //mariadb and mariadbnew

docker inspect mariadbnew //mounts, name:mariavolume , source and dest as //before

docker volume ls //you will find mariavolume

000000000000 Creating volume without creating container
docker volume //Acts as help
docker volume create maria-vol //created
docker volume ls //You will find it

docker run -d –name mariadbprd1 -e MARIADB_ROOT_PASSWORD=1234 -v maria-vol:/var/lib/mysql mariadb

In the above method I used an already made volume

docker container ls //mariadbprd1 , mariadbnew , mariadb

We will create database inside maria-vol
First : Enter bash that is inside the container.
docker exec -it mariaprd1 bash //now you are on bash
mysql –version //make sure it is there

Second: To write sql syntax
mysql -u root -p //To open db itself , it will ask for root password 1234 😊

show databases;

Third:What we will do is that we will create a databases empty, and take that volume to another container.

create database HRDATABASE;
create database HELPDESKDATABASE;
show databases;
exit //now on bash

exit //now on powershell

now mariadbprd1 have data in its volume ,databases created
Now , I will create a new container and will give it the same volume

docker run -d –name mariadbprd2 -e MARIADB_ROOT_PASSWORD=1234 -v maria-vol:/var/lib/mysql mariadb

maria-vol is now attached to it, which have databases created.
Now prd1 and prd2 have same attached volume

To make sure:
docker exec -it mariadbprd2 bash

mysql -u root -p //password 1234
show databases; //They are there

PART SIXTEEN:
BIND MOUNTING IN DOCKER:

mkdir data
cd data
touch data01.txt

//Idea of bind mounting: As shared folder in windows. Many people can access,we will check place on house where containers can share.

docker container run -it - -name alpine -v ${pwd}:/mounted_folder alpine
The above did not work , so I entered the path myself instead of pwd and it worked…

//Here after v , I will inform him with path that I will share, then :/ The name I want to name it. Opposite as before
//pwd is written if path is long
//Inside container as we did -it
ls //Will show you what is inside the container “alpine linux files and mounted_folder”

Cd mounted_folder
ls //data01.txt from inside container

-----new powershell terminal 2
cd data
ls //data01.txt
echo ‘This file is created on host’ > data02.txt
ls ///data01 and data02

---Back to the previous powershell 1 which is inside my container

ls //data01 and data02
cat data02.txt //This file is created on host

WHAT IF WE DID WRITE HERE IN ThE CONTAINER WILL IT REFLECT ON THE FOLDER ON HOST ?? YES

echo ‘This file is created inside container ’ > data03.txt
ls //data01,data02.data03
cat data03.txt // This file is created inside container
exit

------To terminal1 powershell which is outside container again

ls //data01,data02.data03
cat data03.txt // This file is created inside container

0000000000000000000000000000000000000000000000000000000000
On power shell again.
cd data
//create another container and change shared folder name but point to same path

So it will be 2 different folder names with same shared data

docker container run -it –name alpine2 -v ${pwd}:/container_shared alpine
ls //container shared and other linux folders

cd container_shared
ls //data01,data02,data03

0000000000000000000000000000000000000000000000000000000000
We will create a webpage and share its data.
startbootstrap.com //choose template and download

I will open this webpage in a container with nginx.
Had an issue to unzip and that was the solution → https://linuxize.com/post/how-to-unzip-files-in-linux/

The above link to unzip .zip files,,,,
First copy and paste the downloaded files to data folder—open index.html and it will open “but not hosted on server”

Second: Back to powershell.
docker container run -d - -name nginx_dashboard -p 8080:80 -v ${pwd}:/usr/share/nginx/html nginx
//again pwd did not work and entered the path manually

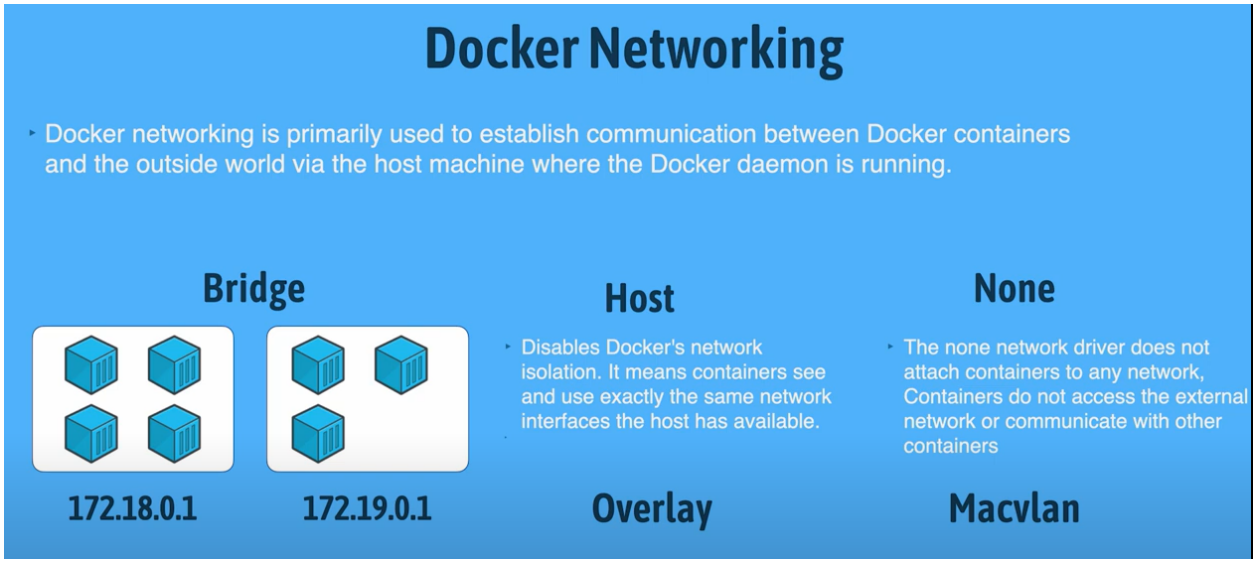
open browser localhost:8080

open data folder on vscode and edit. Index.html anywhere “hello abdo”
save and refresh and it will reflect 😊

Part seventeen: Docker Networking Basic Concepts:

Bridge network makes container on same bridge work and take IP. Default settings done once you create a container. Gets an automatic IP.
Can you create another bridge network than the default one by docker itself ?
Yes,And that satisfies that some containers see each other. And another subnet taking other contaners.
But the default will not see your custom and vice versa.

There are many other types than bridge as host,None,Overlay,Macvlan.



Part Eighteen: Applying Docker Networking | Bridge Network.

On powershell:
docker network ls //shows bridge,host,none,my_network
docker network inspect bridge //You will see details and subnet mask:172.0.0.0 ,Gateway:172.0.0.1
//any container created on this default bridge will take next IP 1,2,3 etc..
//Example:
docker run -d -it ubuntu
docker container ls //Hungry_panda
docker container inspect hungry_panda
//IP of it is 172.17.0.2
docker run -d -it ubuntu
docker container ls //Hungry_abdo

docker container inspect hungry_abdo
//IP of it is 172.17.0.3

Docker network inspect bridge //You will see containers attached to this bridge

Lets create another bridge or another network:

docker network create –driver bridge n1 //of type bridge and name n1
docker network ls //You will find n1 – bridge

docker network inspect n1 //subnet:172.19.0.0 Gateway:172.19.0.1

docker run –network=n1 -d -it ubuntu //Connecting it to my network
docker container ls //you will find it
docker container inspect boody //IP:172.19.0.2

docker run –network=n1 -d -it ubuntu //Connecting another container it to my network
docker network inspect n1 //2 containers now on my bridge .172.19.0.2 and //172.19.0.3

To stop one of them
docker container stop booody
docker network inspect n1 //only one container now

/// Creating a bridge network but this time with assigning manual subnet IP

Docker network create –driver bridge –subnet 172.25.0.0/16 n2
docker network ls //created 😊

//To test:
docker network inspect n2 //correct IP

docker run –network=n2 -d -it ubuntu
docker container ls //created

docker container inspect bodz //Ip:172.25.0.2

//You can move a container from bridge network to another. Example :move bodz from n2 to n1
docker network disconnect n2 bodz //must type network and container name

docker network connect n1 bodz
docker container inspect bodz //now it is moved to 172.19.0.3

Docker network inspect n1 ///We have now 172.19.0.2 and 172.19.0.3

// We want to make sure that containers on same network can reach each others:
We will try to ping 😊

Docker run –network=n1 -it alpine ash //ash is for running shell
ping 172.19.0.2 //Reply is there
ping 172.19.0.3 //Reply is there

From another powershell:
docker network inspect n2
docker run –network=n2 -it alpine ash
ping 172.19.0.3 //No reply at all – As it is another subnet mask
ping 172.19.0.2 //Same issue

Must be connected on same network.

Part Nineteen:Docker Backend Hyper-V vs WSL 2.

We have 2 choices when we are installing docker on windows ,to be working on Hyper-v as default or change it to WSL.

Before windows 10 version 1903 and windows 11 It worked on Hyper V.
If you opened Hyper-V manager you will find DockerDesktopVM,If you increases docker resources ,it will increase.

After those versions it can work on WSL 2 “Windows sub system for linux”—Means from windows you can run linux without any type of VM.
Click on docker icon in tool bar Settings Checkbox: USE THE WSL2 BASED engine and

Turn windows features on or off—check “windows subsystem for linux”
Then in docker settings –Resources –WSL integration--you will find alpine and ubuntu. To add or get other linux destributions:
Go to Microsoft store –Search Debian –Get – Then open it, It will open Debian terminal.
Now you can , ls , cd etc……..

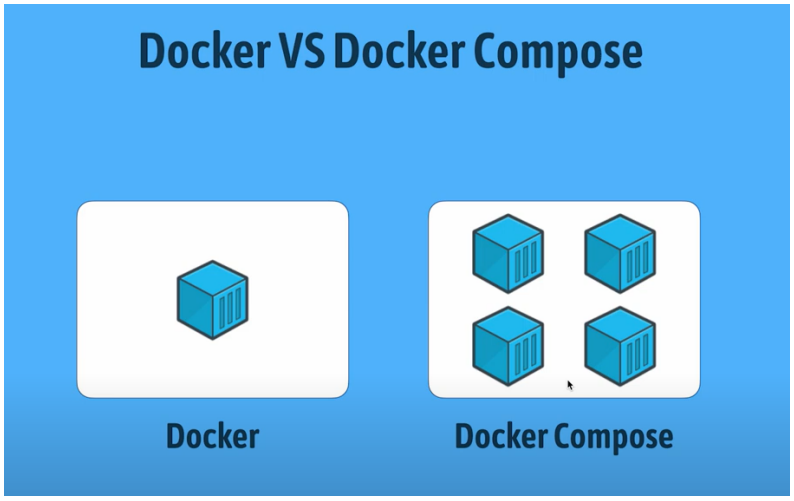
Now get back to docker settings—resources –WSL Integration-- choose Debian for example.
If you opened file explorer you will find linux files

WSL 2 as performance is better.

Part TWENTY: Docker Backend WSL 2 Based Installation:

Installing compose on ubuntu:
https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-22-04
Open docker.com on docker desktop download page.
Download and install choose configuration whether to work WSL2 or Hyper-v based on your windows version.

Part Twenty-One: Docker Compose:



Only if you are working on linux you will have to install it next to docker otherwise it will be installed automatically with docker.

Part Twenty-Two: Docker compose.

on powershell: docker-compose -v //It will provide you with the version
cd ..
cd .. //Go to drive c

mkdir WordPressCompose
cd WordPressCompose
touch docker-compose.yml //This name is a must
//yaml stands for “yet another markup language”
Open visual studio – open the folder on c then go to the yml file and open it.

There will be wordpress ,first layer needed is database of anytype. So layer 1 is mysql and layer 2 wordpress.
So they are 2 containers wordpress and mysql.
version written is version of the compose file.
services means containers.

```
//yaml file/////////////////–search for mysql,wordpres,phpmyadmin on dockerhub you will find templates for yaml
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
version: “3”                                    //version of compose file
services:                                       //Containers
mysql_database:                                 //name of container as you want
        image: mysql                            //image name from docker hub
        restart: always                         //If container is closed for any reason restart it
        environment:                    //root password,database user and pass. You can know env from docker hub
                MYSQL_ROOT_PASSWORD:1234                //admin
                MYSQL_DATABASE: wp_db                   //wp_db I named it here
                MYSQL_USER: wp_user                     //wp_user I named it here
                MYSQL_PASSWORD:1234                     //db password
        volumes:
                - mysql:/var/lib/mysql                  //where data is stored, - to take multiple items
wordpress:
        depends_on:                             //Do not work unless mysql is working
                - mysql_database                //depend on this container as I named it above
        image: wordpress:latest
        restart: always
        ports:
        - “8000:80”
        environment:                                    //Get it from docker hub also
        WORDPRESS_DB_HOST: mysql_database:3306          //name of my container above with port
        WORDPRESS_DB_USER: wp_user                      //all db users and pass as in container of db
        WORDPRESS_DB_PASSWORD: 1234                     //To connect with it.
        WORDPRESS_DB_NAME: wp_db
      volumes:
        [“./:/var/www/html”]

 phpMyAdmin:
        depends_on:
        -mysql_database                 //Do not start before this container
        image: phpmyadmin/phpmyadmin    //From docker hub
        restart:always
        ports:
        -“8080:80”
        environment:                    //From docker hub also
        PMA_HOST: mysql_database
        MYSQL_ROOT_PASSWORD:1234

///Top level volumes------- Always written at the END.
volumes:
        mysql:{}
```

//any – in the above code means accept more than one argument, The tab also is very important
Back to powershell:
docker compose up -d

In browser: http://localhost:8000

//Added phpMyAdmin in yml file
docker compose up -d //3 containers running

In browser: localhost:8080 //PhpMyAdmin with wp_db

In powershell" docker container ls //all 3 containers
docker network ls //wordpress_compose_default bridge created by compose
docker network inspect wordpresscompose_default
//You will find the 3 containers connected on it wordpress,phpMyAdmin,mysql

docker compose down //To switch it off with the network off also

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Part Twenty-Three:
Containerizing Nodejs Application With Docker Compose:
We will create a website which counts the visits to it. But here website is build by us not as wordpress.

on powershell:
cd .. //Till you get to C
mkdir WebsiteVisits
cd WebsiteVisits
touch Dockerfile
touch docker-compose.yml
touch package.json //For nodejs
touch home.js //For nodejs
code . //To open visual studio

////On visual studio:
open package.json

{

```
"dependencies":{
"express":"*",                    //Any version ,express is framework used with node
"redis":"3.1.2"                   //version of redis that you will connect with
},
"scripts":{
"start":"node home.js"            //start with node then go to home.js
}
}
```

Open home.js File in visual studio:

```
const express = require ('express');
const redis = require ('redis');

const app= express();             //new instance of framework

const client = redis.createClient();   //new instance of redis
client.set('visitsCounter',0);          //set initial value of counter to zero
app.get ('/',(req,res)=> {              //request and response
client.get('visitCounter',(err,visitsCounter)=>
{
res.send('visits Counter :' + visitsCounter);          //when you get a request,respond with visits counter
client.set('visitsCounter',parseInt(visitsCounter)+1);          //increment it
})
})
app.listen(8080,() => {
console.log('listening on port 8080');
})
```

Open docker file on vs:

```
FROM node:alpine
WORKDIR /app
COPY package.json   .
RUN npm install
COPY .  .

CMD ["npm","start"]
```

0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

open docker-compose.yml in vs:

```
version: '3'
services:
        redis:
```

image:’redis’
node: //For the node build it from the docker file and dot means that is inside the c
build: .   //build the docker file as here it is made by me not an image on docker hub
ports:
-“8080: 8080”
0000000000

Now, back to home.js file and we will replace the line”to let it know in home.js from where to get redis”:
const client = redis.createClient();
with the below:
const client = redis.createClient({
host:’redis’,   //the name of redis that I did in the yml file or any other name if you did
port:6379
});

Go to websitevisits folder – right click open powershell window here:
docker compose up

open browser localhost:8080   //with each refresh it will increment

If you did modifications to node file , you will need to remove container first from docker app on desktop.
///As you will need to build the same container again.
Then you will go back to powershell
docker compose up

docker container ls   //you will find 2 containers redis and node

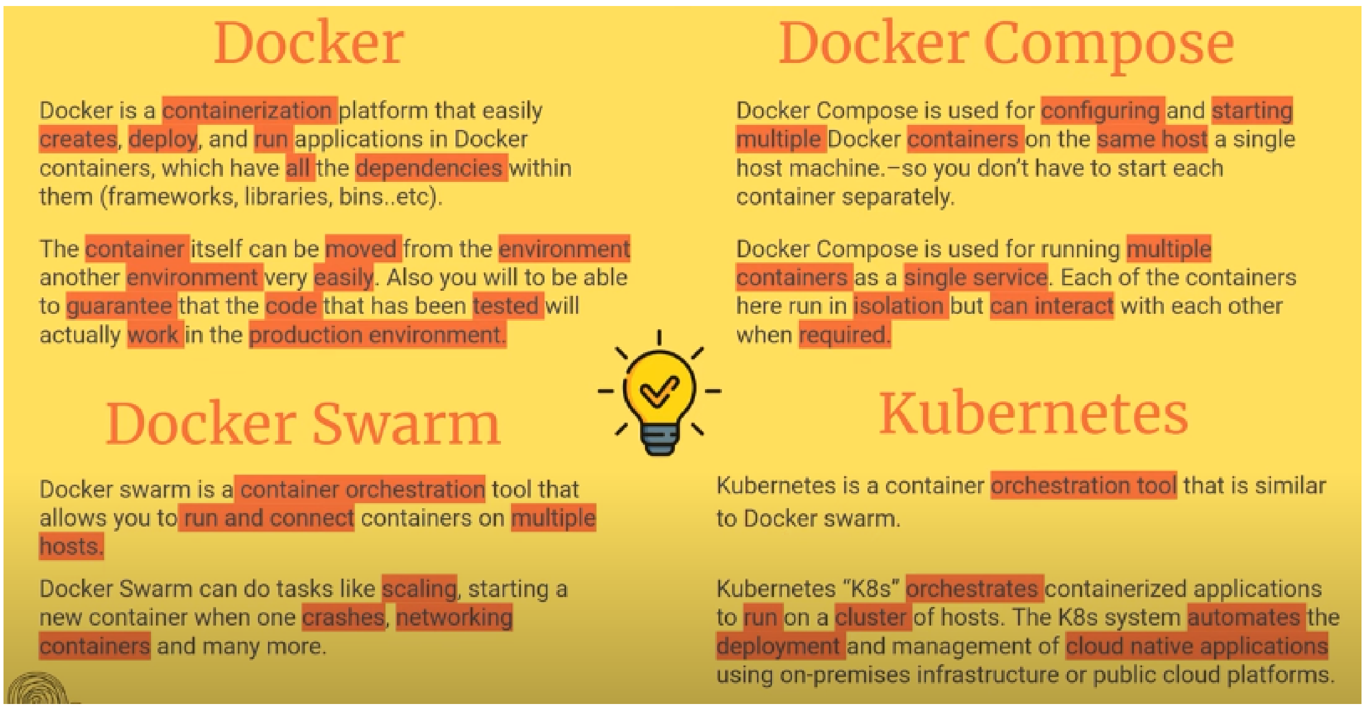
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Installing docker engine on linux:
on terminal:
docker   //output from terminal “not installed”
sudo apt-get update
sudo apt install docker.io
docker   //Installed 😊
docker –v   //Inform you with the version
sudo systemctl status docker   //To make sure that service is running

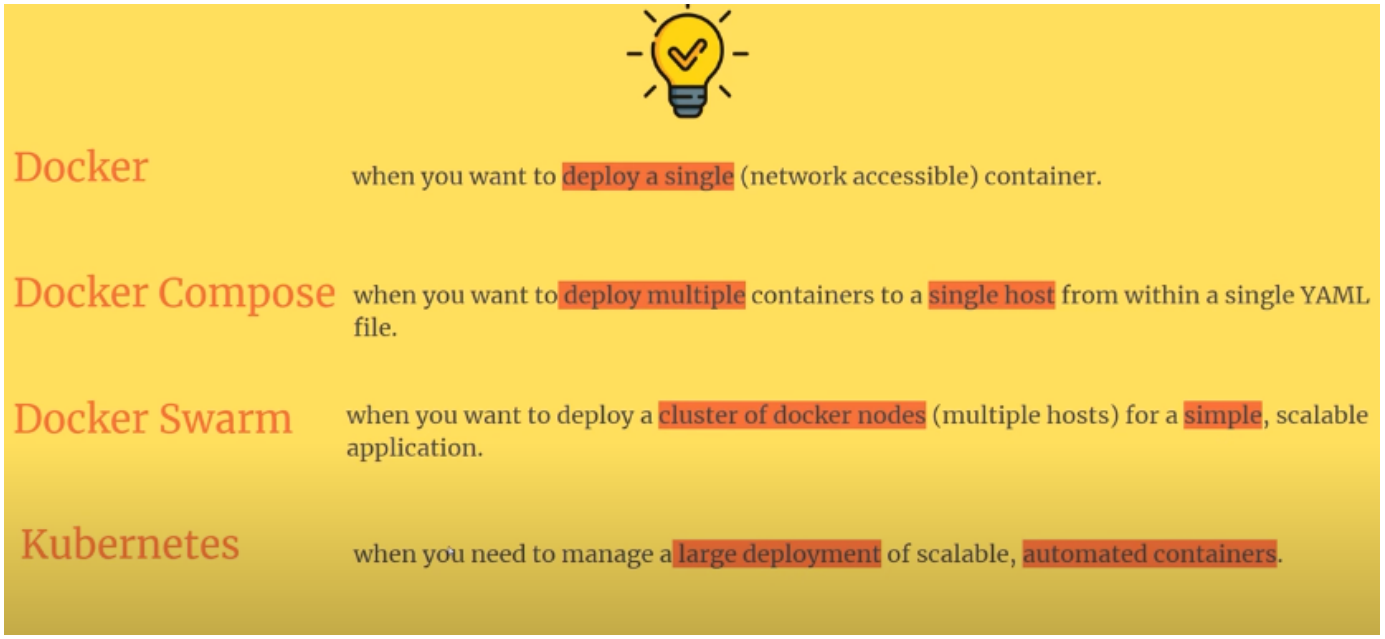
000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000

Docker vs Docker Compose vs Docker swarm vs Kubernetes

Docker only with itself deals with the container as individual on same host.
What if I want to deal with many containers and they are related to each other. Then I will need Docker compose on same host. In Docker compose they are all deal as single service. Wordpress needs mysql, we can let wordpress container communicate with itself for example.
Docker swarm: Acts with lots and lots of containers and guides them how to work and on more than one host.
Kubernetes”K8s”: Stronger than swarm.



When to Use:



—-------

Notes:

Remove all containers at once

docker rm $(docker container ls -a -q)

Remove all images at once

docker rmi $(docker images -a -q)