<https://github.com/bstashchuk/docker>

--------------------------------------------------------------------

**Commands-Notes**

* Section 3: Basic Docker Containers (Ubuntu, Busybox, Alpine)

docker run hello-world

docker run ubuntu //when we run this image, we download the image but the container cannot run because there were no commands (no process) that were executing inside this container

Every time that we are executing the command “docker run” a new container is created

Docker run -it ubuntu //we are entering the bash shell by using the -it (interactive)

In order to create a Linux container with the smallest size possible you can use the Busybox image.

Docker run busybox

Docker run -it busybox

Docker pull alpine //alpine is a Linux distribution that is small in size and secure

Docker run -it alpine

Cat /etc/os-release //we can see some information about our system (OS etc.)

* Section 4: Port and Volume Mapping in the Docker Containers

Nginx is a web server.

Docker pull nginx

Docker kill nostalgic\_wu

Docker run nginx

Docker stop <container\_id>

Docker run -p 8080:80 nginx //-p stands for port, now we exposed the port 8080 on my computer. The 80 port is the port that is exposed inside the container

docker run -p 8081:80 -v ${PWD}:/usr/share/nginx/html nginx //assigning the port to expose the web server, creating the volume and we are specifying our local path where the files, writing the image’s name

* Section 5: Docker Containers Management (Ubuntu, NGINX)

Docker history alpine

Docker run -i alpine //with -i we have access into the input of the container

Docker run -p 8080:80 -d nginx //run the container in the background so from now on we can’t see the logs in the terminal and the terminal window is not reserved

Docker logs <container\_id>

Docker run -t alpine //we are creating a pseudo-tty/pseudoterminal but we don’t have access to the input that’s why we combine -i and -t

Docker run -it ubuntu

We can run multiple containers of the same image by using different terminals!

docker run -p 5555:80 -v ${PWD}:/usr/share/nginx/html --name nginx1 nginx

docker run -p 5556:80 -v ${PWD}:/usr/share/nginx/html --name nginx2 nginx

docker start <container\_name> or <container\_id> //in order to start a specific container then I must use the command docker start

docker stop <container\_name> or <container\_id>

docker container prune //delete every stopped container

docker container rm <container\_id> //delete a container

* Section 6: Running Python Applications in Docker

Docker pull python

Docker run -it python

docker run -it -v ${PWD}:/app python python3 /app/hello-world.py //we have to make a volume for our files and we have to specify the version of the python and to insert the name of the file and the path of the file inside the container

docker run -it -v ${PWD}:/app -w /app python python3 hello-world.py //instead of specifying also the path inside the container we can assign the working directory inside the container by using the -w parameter

* Section 7: Running Node.js Applications in Docker

Docker pull node

Docker run -it node

docker run -v ${PWD}:/app -w /app node node hello.js //again we have to specify the working directory inside the container and also we have to specify the command that is going to run inside the container (letters with red color)

npm i install

docker run -v ${PWD}:/app -w /app -it node npm init

docker run -v ${PWD}:/app -w /app -it node npm i express

docker run -v ${PWD}:/app -w /app -p 3000:3000 -it node node index.js

* Section 8: Running MongoDB Containers

Docker pull mongo

Docker run mongo

Docker run mongo

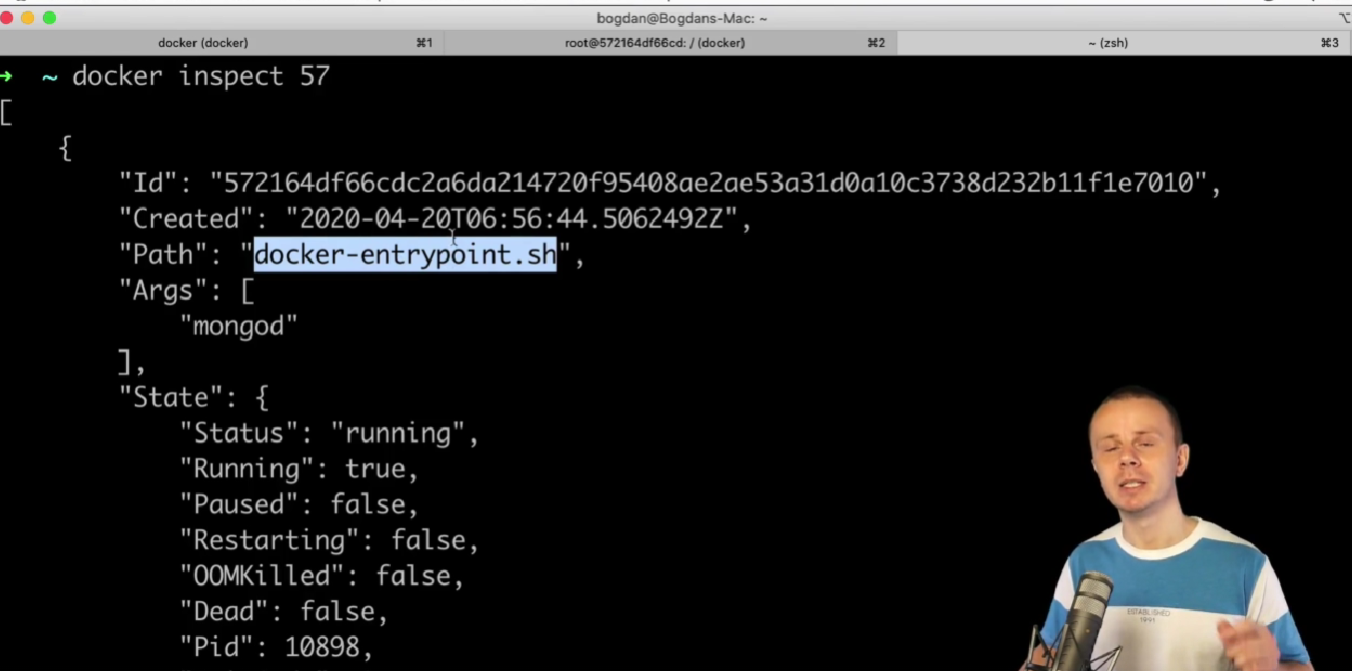
And in a new window I’m running the following commands:

Docker exec -it <container\_id> bash //in order to start a new process in a running container we have to use the command exec

mongosh

Docker inspect <container\_id> //this command gives us details about the running container

This container was started using the script in Path and the script used the argument “mongod”



Ls /usr/local/bin

Cat /usr/loca/bin/docker-entrypoint.sh

Ls /usr/bin

You are able to connect in a mongod process inside a container using shell process inside the same container

Docker exec -it <container\_id> mongo //we started a process for mongo shell in the same container

Show dbs

Use test

Db

Db.animals.insert({“animal”:”cat”})

Db.animals.find()

exit

**Persistent databases:**

Docker exec -it <container\_id> bash

ls

ls data

ls data/db

We want to expose a path of our computer to the container in order to save there the database that we are creating inside the container

Docker run -d -v ${PWD}/db:/data/db mongo //make sure that you are logged in the following path C:\Users\johnk\Desktop\Different Technology Notes\Docker\Docker and Kubernetes Udemy course\containers\mongodb

Mongosh

Use test2

Db.animals.insert({“animals”:”cat”})

Db.animals.insert({“animals”:”dog”})

Db.animals.insert({“animals”:”monkey”})

Db.animals.find()

**And by this we can use the same database in different containers!**

* Section 9: Communication between Containers and Environment Variables (MySQL, phpMyAdmin)

Docker pull wordpress

Docker run wordpress

Docker run -d -p 8080:80 wordpress //we have to create also a container for mysql database in order to setup the wordpress correctly

**Default bridge network and communication between containers**:

Docker run -it busybox

Docker run -it busybox

Hostname -i //check the ip address of these 2 containers

The containers are able to communicate. Containers in the same network could communicate with each other using IP addresses

Docker run mysql

Docker run -e MYSQL\_ROOT\_PASSWORD=1234 mysql //with -e we can initialize values in the environment variables

docker exec <container\_id> env //see the variables of a specific environment

docker pull phpmyadmin/phpMyAdmin

docker run -p 8080:80 phpmyadmin/phpMyAdmin

**Connecting phpMyAdmin to mysql container**: use the Usage with external server (<https://hub.docker.com/r/phpmyadmin/phpmyadmin>)

docker run -p 8080:80 -e PMA\_HOST=<container\_IP> phpmyadmin/phpMyAdmin

But it’s not good to use the ip of a container because it’s going to be changed

* Section 10: Default and Custom Bridge Networks in Docker (Wordpress, MySQL)

Docker run -it busybox

Hostname -i

Hostname

Docker run -it –name busybox1 -h busybox-one busybox

In the default bridge network containers can’t communicate using hostnames or names.

In order for the containers to communicate with each other by using their hostnames, we have to create a custom bridge network and then we have to create in this network the containers that we want

Docker network –help

Docker network ls

Docker network inspect bridge //we can see the range of the subnet for our containers, we can see also the containers that we have in this network, also we can see their MacAddress, IP addresses

Docker network create custom

Docker network ls

Docker network inspect custom

**Creating a network and assigning it in a specific network**: docker run -it –network custom busybox

docker run -it –network custom busybox

ping <container\_name> //run the command hostname in order to learn the name of a container

So by using a custom network, two containers can communicate by using the hostname

Docker run -it –network custom –name busybox2 busybox //by assigning a custom name into our container now the containers can communicate with their custom names

//Example

//Custom bridge network

docker network create mysql

START MY SQL SERVER WITH CUSTOM ROOT PASSWORD

docker run --network mysql --name mysql -e MYSQL\_ROOT\_PASSWORD=1234 -d mysql

START MYPHPADMIN WITH PMA\_HOST VARIABLE

docker run --network mysql -p 8080:80 -e PMA\_HOST=mysql -d phpmyadmin/phpmyadmin

**Wordpress with mysql and phpMyAdmin**:

//Example

docker network create wordpress

START MY SQL SERVER WITH CUSTOM ROOT PASSWORD

docker run --network wordpress --name mysql -e MYSQL\_ROOT\_PASSWORD=1234 -e MYSQL\_DATABASE=wordpress\_db -e MYSQL\_USER=kampe -e MYSQL\_PASSWORD=12345 -d mysql:5.7

START WORDPRESS

docker run --network wordpress -p 8080:80 --name wordpress -d wordpress:5.4

START MYPHPADMIN WITH PMA\_HOST VARIABLE

docker run --network wordpress --name phpmyadmin -p 8081:80 -e PMA\_HOST=mysql -d phpmyadmin/phpmyadmin

* Section 11: Additional Containers - Elasticsearch, Redis, Httpd

Docker run -it alpine

docker pull appropriate/curl

docker run -it appropriate/curl google.com

docker run -it appropriate/curl sh //inside the container run curl --help, so the curl is available inside this container

docker network create elasticsearch

docker run --network elasticsearch --name elastic -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:7.5.2

docker run --network elasticsearch --name curl -it appropriate/curl sh //check if we can ping the elastic node (ping elastic)

curl -XPUT <http://elastic:9200/my-index> //create a new index

curl -XGET <http://elastic:9200/_cat/indices?v> //get information about existing indexes in elastic search server

curl -XPOST http://elastic:9200/my-index/cities/1 -H 'Content-Type: application/json' -d '{"city":"New York"}'

curl -XPOST http://elastic:9200/my-index/cities/2 -H 'Content-Type: application/json' -d '{"city":"Paris"}'

curl -XPOST http://elastic:9200/my-index/cities/3 -H 'Content-Type: application/json' -d '{"city":"Peru"}'

curl -XGET http://elastic:9200/my-index/\_mapping?pretty

curl -XGET <http://elastic:9200/my-index/cities/2?pretty>

curl -XGET <http://elastic:9200/my-index/_search?q=city:new>

**Redis: database for real time data. Real fast db**

Docker run redis

docker exec -it <container\_id> redis-cli //INFO, SET key1 “Hey there”, GET key1

docker network create redis

docker run –name redis –network redis -d redis

docker run –name redis-commander –network redis -p 8081:8081 -e REDIS\_HOST=redis -d rediscommander/redis-commander

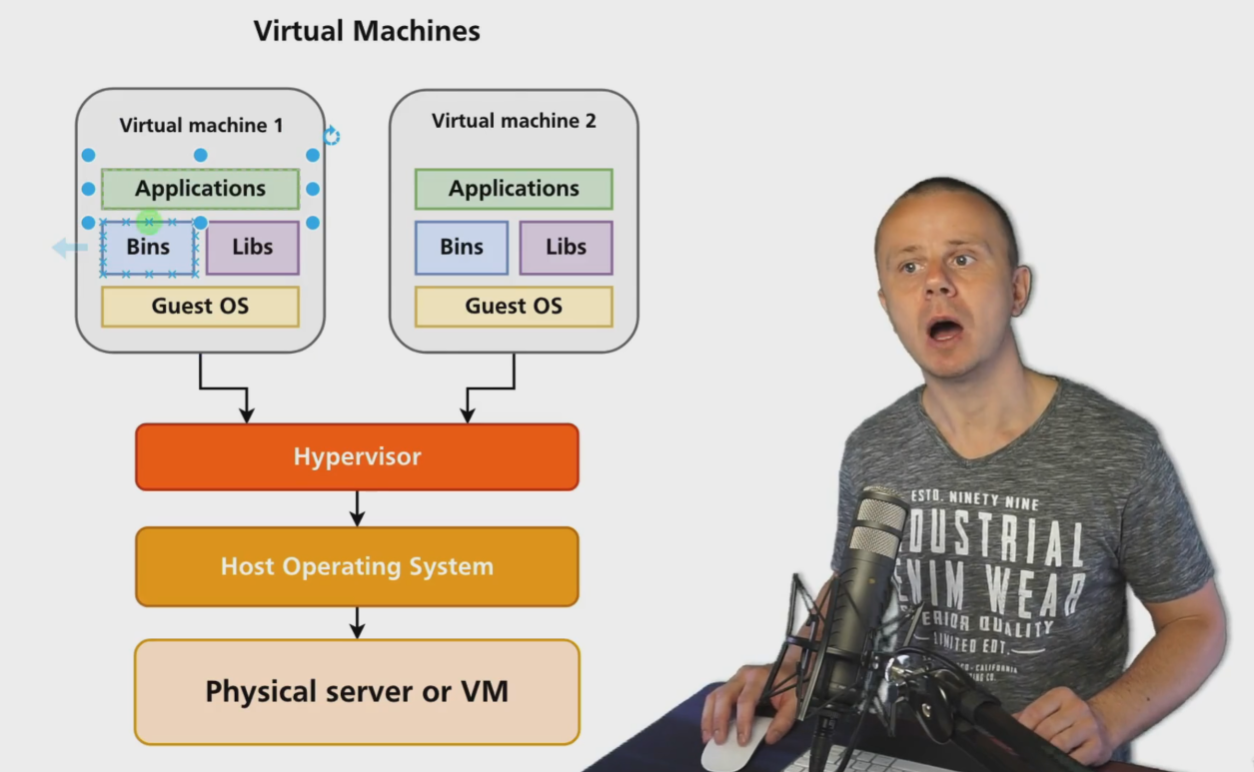
**httpd apache http server**

docker run -p 8080:80 -d httpd

* Section 13: What is Docker and Docker Components

VMs vs Docker Container

Virtual machines need a Hypervisor to be installed into our computer. Our computer has it’s own OS.



In Docker Containers we don’t have Guest OS and we don’t have a Hypervisor, we are using just a Docker Engine. The Docker Engine is the intermediate that helps the containers to use the OS of the current system that they are using.

Communication between Host OS and Docker Containers.

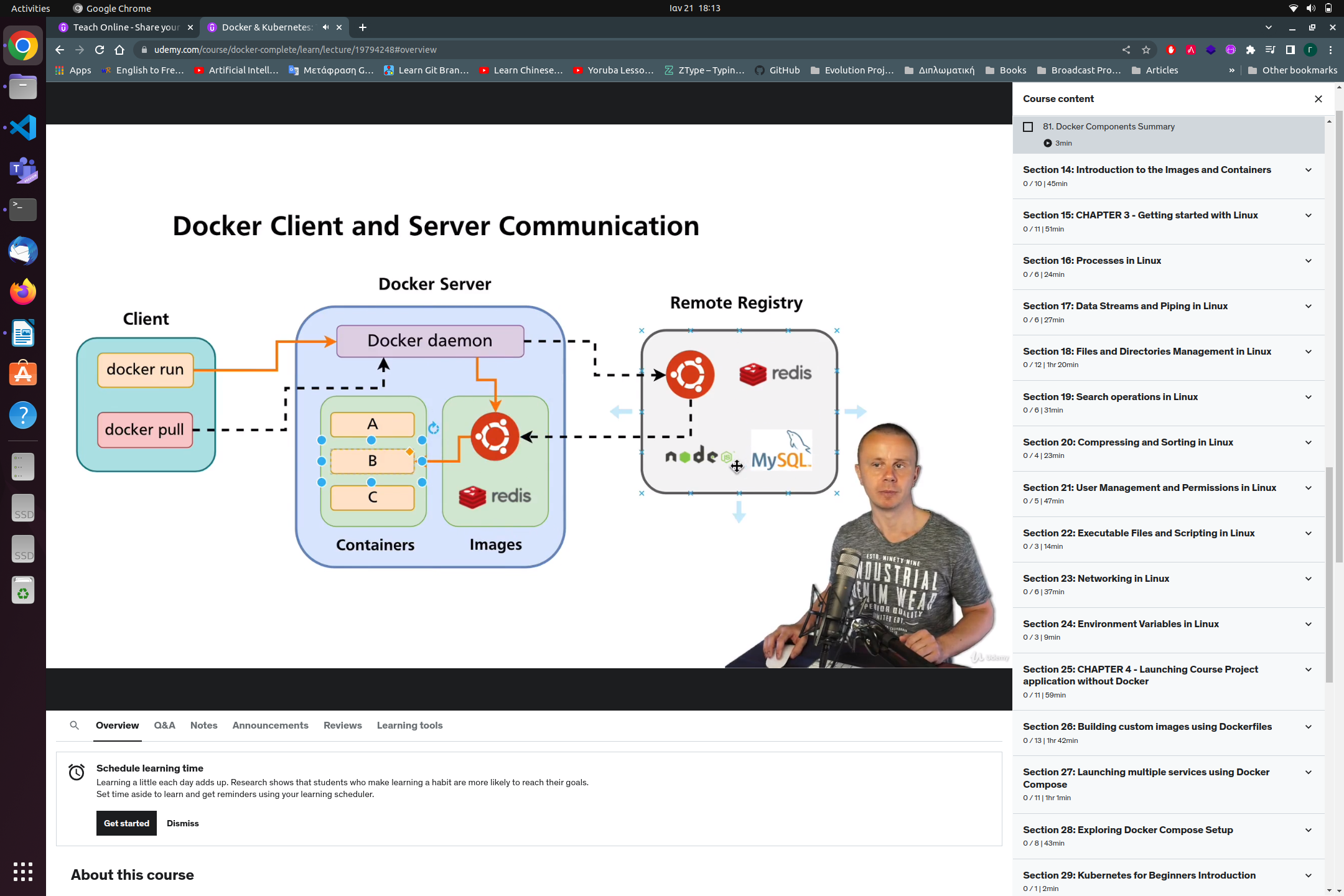
Docker containers run in a Linux VM in the other OSs like Windows and Mac

Namespaces in Linux allow isolation of computer resources between processes. We can limit the resources per process by using the cgroups.

Th best practice is to create a new container for every need that we have. Example: 1 for elasticsearch, 1 for mysql (every single one of them is going to run on it’s own process because we can manage them more easily with this technique [**Single Responsibility Principle**], Single purpose containers)

**Docker components Overview:**

* **Docker Client** = when I’m using the docker commands it means that I’m using the docker client process. It is separated from the docker server and it is used for execution of commands on docker server and getting output of it
* **Docker Server** = is a set of different processes and docker daemon is one of them
* **Docker Host** = the processes of the server are running in Docker Host. The docker host is the OS that our PC is using. Now the docker host is my ubuntu linux. In Windows OS I don’t have immediate access at the Docker Host, But in Linux I do (127.0.0.1)
* **Docker Image** = are read only files where we can create containers. A set of files, file system layers. Can’t modify the files that It has
* **Docker Container** = We can modify the files that it has. In a container we can modify and create files. We can create containers with the use of the images
* **Docker Repository** = it’s like git repository but in a docker repository we can have different versions of a docker image that is saved in the docker hub
* **Docker Registry** = It’s a place where we host the docker images. the most popular docker registry it’s the docker hub where we can save our docker images and share them

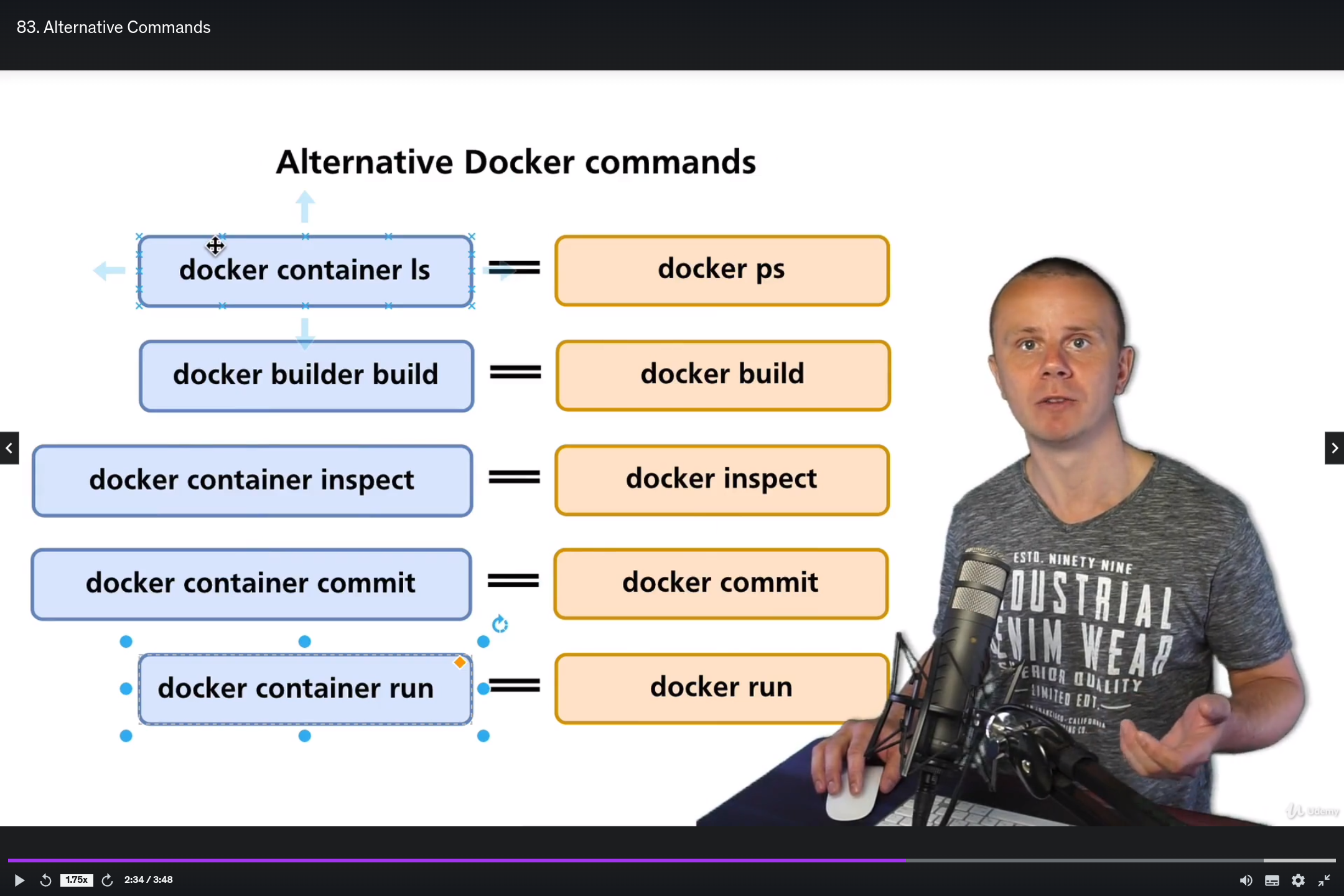


- **Section 14: Introduction to the Images and Containers**

**Management commands** = using docker image we can manage the images that we have installed

**Commands** (**regular commands**) = docker images we can list the images that we have installed, ps we can list the containers. Docker ps --help, docker container –help, docker container wait –help,

**Alternative commands** =



**Basic container and image commands** =

docker run = start a container

docker stop = stop container

docker kill = kill container

docker rm = remove a container

docker ps = list of running containers

docker ps -a = list all containers (including stopped containers)

docker pull = download an image from docker hub

docker images = list of downloaded images

docker rmi = remove images

docker build = create a new custom image

**Cleaning up my Docker setup** = docker container prune , docker image prune -a

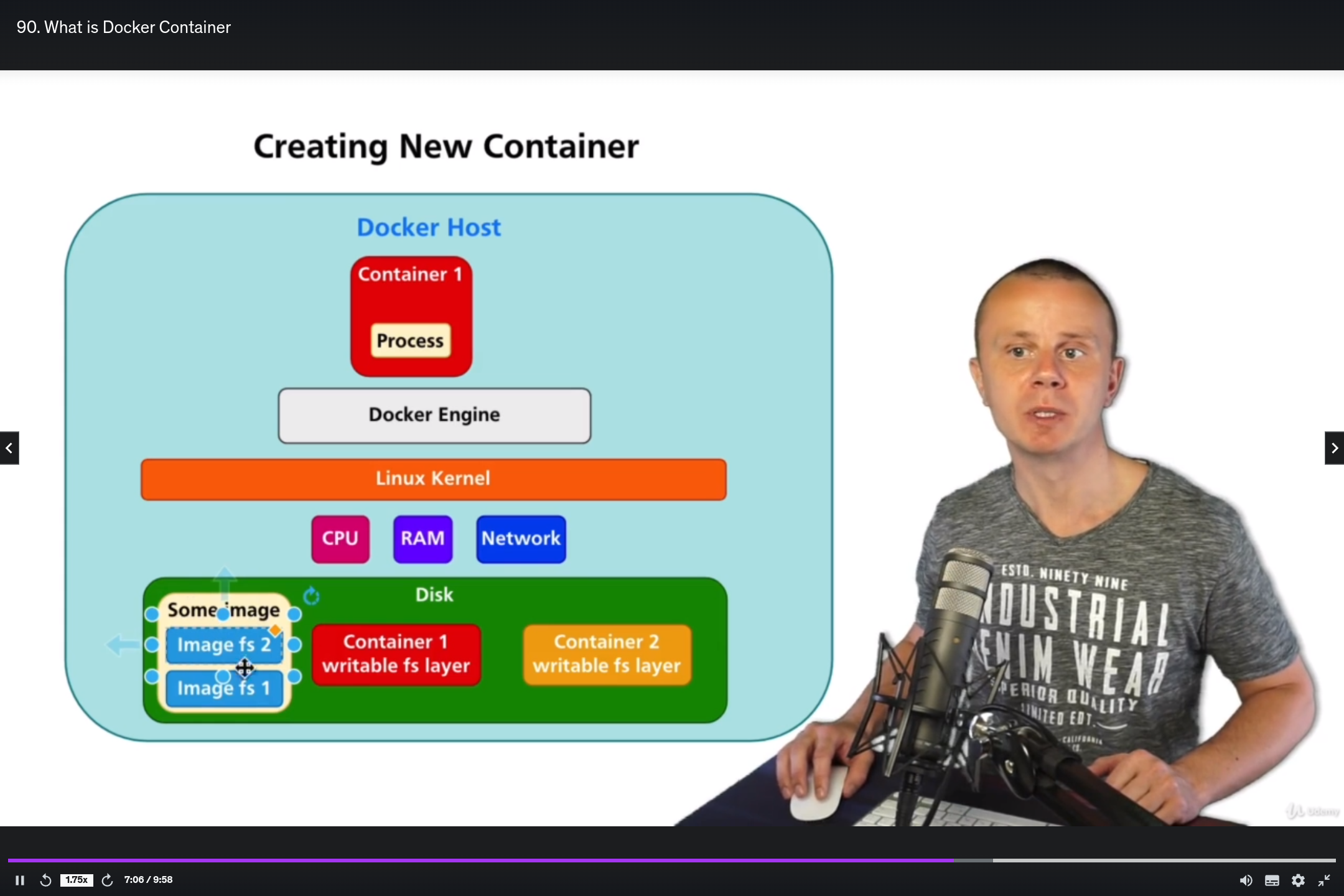
**Pulling images from the Docker Hub** = docker pull, docker images, docker rmi, docker build .

**What is Docker image** = they are a set of files (read-only), fs layers (file system layers) different multiple distinct layers in order to reuse files from other images etc., every fs layer is a set of files and folders, they are read-only and we can modify their content

**Creating new Container from the image** = docker run hello-world

**What is CMD in the Docker image** = the CMD shows which file is going to run when we are going to create a container from a specific image. Every layer has instructions on what commands the terminal is going to run

**What is Docker container** =



**Summary** = Every docker image is read-only and consists of multiple fs layers and those fs layers could be reused between different images also when we create a new container from a specific image docker creates a new writable fs layer for this container on the disk and also docker starts a new process(es) based on the CMD instructions located in the image. As soon as all processes in the container are stopped docker stops container as well. Container share resources of the Docker host such as RAM, CPU, Hard Drive and network. Requests to the resources are made via common Linux Kernel

**- Section 25: CHAPTER 3 – Getting started with Linux**

**Why you need to know Linux while studying Docker** = docker host is using a Linux Kernel so we must learn how to use Linux because Docker utilizes linux in order to work properly

**Introduction to the Linux section** = we must learn to use the terminal in Linux

**What is Shell, Terminal and command** = Bash is linux shell, Shell is the interpreter of Linux commands

- **Section 16: Processes in Linux**

**List running processes** = ps command lists the running processes

**Starting additional processes and killing processes** = with docker exec -it <container\_id> bash I can start a new process

**Installing htop package using apt-get** = install additional package using apt-get utility. Apt-get update, apt-get install htop

**Using htop utility** = htop utility shows processes in a more good looking way

Htop -u john

Section 17: Data Streams and Piping in Linux

**Data streams of the process** = STDIN, STDOUT, STDERR

**Redirecting STDOUT and STDERR to the file** = ls > stdout.txt, cat stdout.txt

Bash has a process and from this processes stdin we are sending to the commands process data then we are getting the stdout of the process the data to the bash process and then we are getting the results into our screen

**How to send data to STDIN and redirect STDOUT and STDERR** = 0 – STDIN, 1 – STDOUT, 2 – STDERR, cat 0< stderr.txt, ls 1> stdout.txt 2> stderr.txt, mkdir 1> stdout.txt 2> stderr.txt, if I want to append text into my file I have to do this mkdir 1>> stdout.txt 2>> stderr.txt, cat absent-file.txt 1>> stdout.txt 2>> stderr.txt

**Piping** = echo “Hello world” | cat > hello.txt, send the stdout of one process to the stdin of another process using piping

Section 18: Files and Directories Management in Linux

**Linux file system structure and navigation** = it’s like a tree which starts from /, pwd, cd / I’m located in root directory, in order to go back to my locked in user I just have to use the command cd, cd . represents the current directory

**Creating and removing directories and files** = mkdir -p test\_folder/test2 (create also the test\_folder directory which doesn’t exist), rm -r test2, touch file1.txt, echo “Hello World” > file2.txt, cat file2.txt

-rf = recursive force

**Creating new files** = touch file.txt , echo “Hello” > file.txt , cat file.txt

**Editing files using Vim and Nano editors** = apt-get install vim nano

**Copy and move files and directories** = cp prototype.txt copy-of-prototype.txt

cp *etc/*libaudit.conf libaudit-backup.conf , cp r /etc/\* etc-backup/ , mv file4.txt file5.txt (rename specific file)

**Reading files** = cp > lfile.txt (**press enter and then paste the content and in order to end the input process press Ctrl+C**) ,

head lfile.txt (**read the first 10 lines of a file**)

tail lfile.txt (**read the last 10 lines of a file**)

**Filtering text using grep command** = ls -l | grep hostname , cat file5.txt | grep ridiculus , ls -l | grep ^l , ls -l | grep ch$

**Soft vs Hard links** = ls -l | grep “\->”

the soft link has low size and it points into another file , hard link = inode

**Creating hard and soft links** = if I want to reuse a file then I;m going to create a soft link , ln -s file5.txt file5-softlink.txt , ln file6.txt file6-hardlink.txt , hard link = 2 or more files are pointing into the same inode memory so that’s why when a file changes then the other file has the same changes (same size, same date modified etc.)

**Section 19: Search operations in Linux**

find . -name file-one.pdf , current directory

find .. -name file-one.pdf , parent directory

find / -name file-one.pdf , relative path

find . -type f

find . -type d

find . -type l

find . -type f -ls

find . -type f -empty

find . -type f -size +10M -ls

find / -type f -perm 777 -ls

find . -name “\*.txt” -exec cat {} \;

ls | xargs cat

find / -name kernel -type d -exec ls -l --color {} \;

find / -name kernel -type d | xargs ls -l --color

**Section 20: Compressing and Sorting in Linux**

tar brings some files into an archive, gzip zips the files and unzips them

tar -czf ../archive.tat.gz .

Tar -xvzf /*archive*.tar.gz

ls -l --sort=size

sort animals.txt

sort -o sorted.txt animals.txt

**Section 21: User Management and Permissions in Linux**

useradd john , create john user

su john , login as john

passwd john , set password

userdel john

useradd --help | grep home

useradd -m john

adduser john , with adduser we can assign from the start the password we can create also a home directory for the user etc.

chown john:john /home/john/2.txt

nano ~/tmp/3.txt

cat tmp/3.txt

chmod 755 /home/john/new-file.txt

ls -l /home/john

chmod -x /home/john/new-file.txt

chmod +x /home/john/new-file.txt

**Section 22: Executable Files and Scripting in Linux**

touch script.sh

nano script.sh

first line of the script , #!/bin/bash echo “Hello World!”

/home/john/script.sh , ./script.sh , ~/script.sh

**Adding path to the scripts into the PATH variable** = mv script.sh bin/script.sh

export PATH=$PATH:/home/john/bin

**Section 23: Networking in Linux**

docker run -it alpine

hostname -i

ip address

ip address | grep inet

ip route

nslookup google.com

traceroute google.com

whois facebook.com

docker run -it -p 2222:22 ubuntu

apt-get update

apt-get install openssh-server

service ssh status

service ssh start

adduser john

ssh -p 2222 john@localhost

**using curl and wget utilities** = apt-get install curl

curl google.com

Section 24: Environment Variables in Linux

Env

Export MY\_NAME=John

Touch name.sh

Section 25: CHAPTER 4 - Launching Course Project application without Docker

We are going to make our life easier by dockerizing our microservices

Section 26: Building custom images using Dockerfiles

* Initializing a local repository for our project and then we are pushing the changes to GitHub
* Creating Dockerfile for the python API service

FROM python:3.9

WORKDIR /app

EXPOSE 5050

COPY Pipfile Pipfile.lock ./

RUN pip install pipenv

RUN pipenv lock --requrements > requirements.txt

RUN pip install -r requirements.txt

COPY . ./

CMD [“python”, “main.py”]

Docker build . (in this current folder that we are docker finds the dockerfile, if we want to specify another path then we have to write the command like this docker build ῀/Desktop/images…)

In order to insert a name to our image then we have to build the dockerfile again like this: docker build . -t <image\_name>

Docker run -p 5050:5050 images-gallery-api

Docker exec -it frosty\_babbage bash

.dockerignore

Creating a docker file for the front-end application,

FROM node:15.14-alpine

WORKDIR /app

EXPOSE 3000

COPY package.json package-lock.json ./

RUN npm install --silent

COPY . ./ --copy every other file and insert it into the working directory of the container (app)

CMD [“npm”, “start”]

Building docker image for the frontend app,

docker build .

Running docker container for the front-end service, docker run -p 3000:3000 images-gallery-frontend

Docker exec -it confident-bose sh

Section 27: Launching multiple services using Docker Compose

Create a new file named docker-compose.yml where we are going to start multiple images with a single command

Version: ‘3’

Services:

Frontend:

Build: ./frontend

Ports:

- ‘3000:3000’

Api:

Build: ./api

Ports:

- ‘5050:5050’

Docker-compose up, in order to run the docker-compose.yml file

Docker-compose build, only re-builds necessary images

Docker-compose up --build, re-builds images and creates new containers

Docker-compose down //stops and removes the images that we created in the folder that we are

In order to see the changes that we make locally into out website, we have to make volumes meaning that we have to map the folders that contain our code in our local machine with the container files so that the container files are going to be overwritten by our files

Version: ‘3’

Services:

Frontend:

Build: ./frontend

Ports:

- ‘3000:3000’

Volumes:

- /app/node\_modules //we are using this because we don’t want to be overwritten by frontend files ofr our local machine

- ./frontend:/app

Api:

Build: ./api

Ports:

- ‘5050:5050’

Docker-compose down

Docker-compose up

Docker exec -it images\_gallery\_frontend\_1 sh

If we won’t see the changes in the website which means that the changes are not reflected in the container. So we have to insert the following value inside out docker-compose file

Version: ‘3’

Services:

Frontend:

Build: ./frontend

Ports:

- ‘3000:3000’

Volumes:

- /app/node\_modules //we are using this because we don’t want to be overwritten by frontend files ofr our local machine

- ./frontend:/app

Environment:

- CHOKIDAR\_USEPOLLING=true

Api:

Build: ./api

Ports:

- ‘5050:5050’

Volumes:

- ./api:/app

Docker-compose up -d

If we are going to have a failure inside a docker container, then the container is going to bet stopped and then we have to re-run it. We can do this automatically by using a parameter inside docker-compose file.

Version: ‘3’

Services:

Frontend:

Restart: always

Build: ./frontend

Ports:

- ‘3000:3000’

Volumes:

- /app/node\_modules //we are using this because we don’t want to be overwritten by frontend files ofr our local machine

- ./frontend:/app

Environment:

- CHOKIDAR\_USEPOLLING=true

Api:

Restart: always

Build: ./api

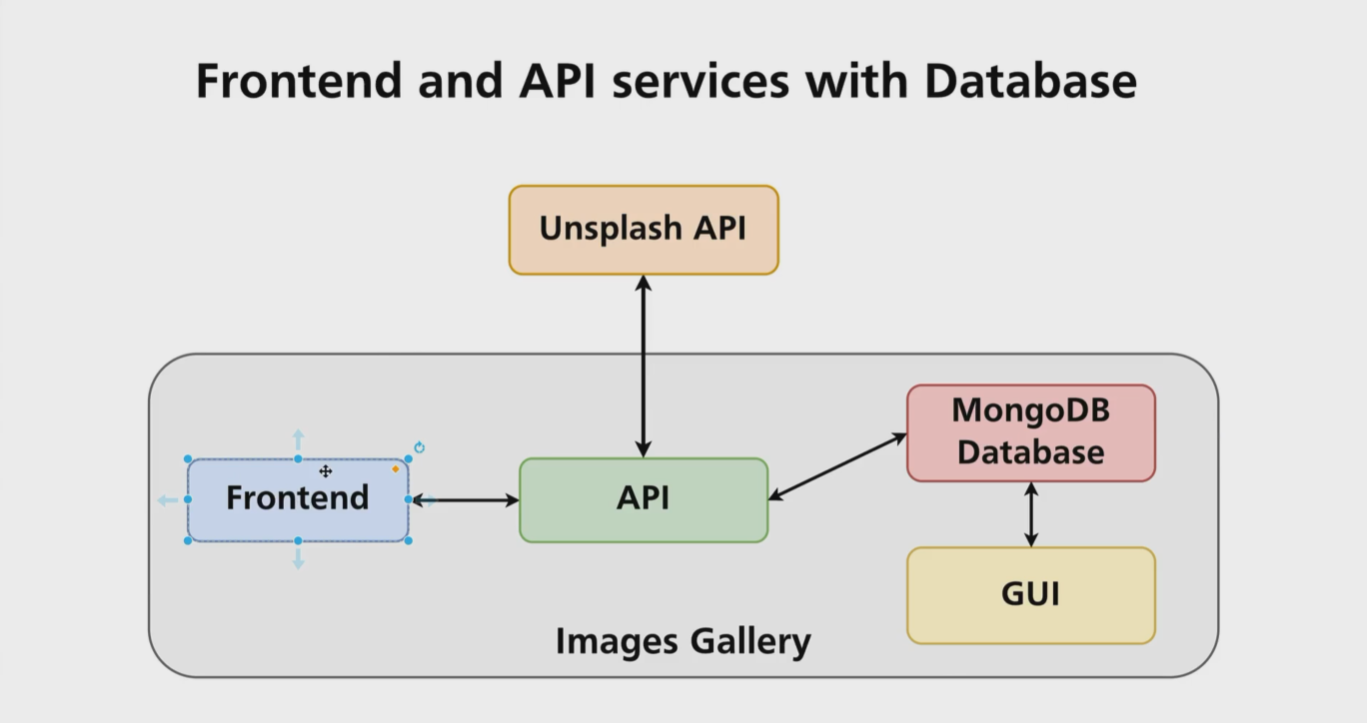
Ports:

- ‘5050:5050’

Volumes:

- ./api:/app

Incorporating mongo and mongo express services



Docker has internal DNS server and it automatically creates DNS mappings for all containers defined in the docker-compose.yml file. It means mongo container could be reached in the network by its name **mongo**

Version: ‘3’

Services:

Frontend:

Restart: always

Build: ./frontend

Ports:

- ‘3000:3000’

Volumes:

- /app/node\_modules //we are using this because we don’t want to be overwritten by frontend files ofr our local machine

- ./frontend:/app

Environment:

- CHOKIDAR\_USEPOLLING=true

Api:

Restart: always

Build: ./api

Ports:

- ‘5050:5050’

Volumes:

- ./api:/app

mongo:

image: mongo

restart: always

environment:

MONGO\_INITDB\_ROOT\_USERNAME: root

MONGO\_INITDB\_ROOT\_PASSWORD: example

mongo-express:

image: mongo-express

restart: always

ports:

- 8081:8081

environment:

ME\_CONFIG\_MONGODB\_ADMINUSERNAME: root

ME\_CONFIG\_MONGODB\_ADMINPASSWORD: example

ME\_CONFIG\_MONGODB\_SERVER: mongo

Depends-on:

- mongo //with this configuration, the mongo-express is going to start when the mongo db starts, **mongo-express is going to start after mongo**

Section 28: Exploring Docker Compose Setup

In order to login in to the mongo container you have to use the following command:

Docker exec -it images-gallery\_mongo\_1 mongo –username root –password very-strong-db-password

Docker-compose up creates new containers so our previous data are going to be deleted, so somehow we have to make persistent databases.

Version: ‘3’

Services:

Frontend:

Restart: always

Build: ./frontend

Ports:

- ‘3000:3000’

Volumes:

- /app/node\_modules //we are using this because we don’t want to be overwritten by frontend files ofr our local machine

- ./frontend:/app

Environment:

- CHOKIDAR\_USEPOLLING=true

Api:

Restart: always

Build: ./api

Ports:

- ‘5050:5050’

Volumes:

- ./api:/app

mongo:

image: mongo

restart: always

environment:

MONGO\_INITDB\_ROOT\_USERNAME: root

MONGO\_INITDB\_ROOT\_PASSWORD: example

Volumes:

- Mongodb\_data:/data/db

mongo-express:

image: mongo-express

restart: always

ports:

- 8081:8081

environment:

ME\_CONFIG\_MONGODB\_ADMINUSERNAME: root

ME\_CONFIG\_MONGODB\_ADMINPASSWORD: example

ME\_CONFIG\_MONGODB\_SERVER: mongo

Depends-on:

- mongo //with this configuration, the mongo-express is going to start when the mongo db starts, **mongo-express is going to start after mongo**

Volumes:

Mongodb\_data:

***Docker-compose creates bridge network automatically***

**Section 30: Kubernetes Overview**

Kubernetes or K8S, automatic deployment of the containerized applications across different servers, distribution of the load across multiple servers, auto-scaling, Orchestration of the containers

What is a Pod, containers are created inside a pod which containers share volumes and IP addresses. The most common case is one container per pod

Kubernetes cluster and nodes, inside a Kubernetes cluster we have nodes that are servers bare metal or virtual and inside a node we have pods and inside a pod we have containers. My job is to create the clusters and the nodes. Master node and worker nodes.

Kubernetes services of worker nodes and master node, container runtime (runs containers inside a Kubernetes node), kubelet (communicates with the API server of master node, the API server is needed for the internal communication of every worker node), kube-proxy (we need this for the inter communication inside a node and between nodes)

What is kubectl, we can connect to a specific Kubernetes cluster and manage it remotely

**Section 31: Minikube Installation**

Minikube creates Kubernetes cluster with single node locally

**Section 32: Kubernetes Cluster**

Minikube status

Minikube start --driver=hyperv

With minikube ip we can find the ip of the virtual node (Kubernetes node) in order to connect with SSH in the server. Ssh docker@<the ip of my Kubernetes node> with this command I can connect to the server via ssh. Minikube node user credentials = docker , tcuser.

Kubectl cluster-info

Kubectl get nodes

Kubectl get pods

Kubectl get namespaces

**Section 33: Kubernetes Pod Creation**

Kubectl run nginx –image=nginx

Kubectl get pods

Kubectl describe pod nginx

Kubectl get pods -o wide

Kubectl delete pod nginx

Use Git Bash in order to create aliases example: alias k=”kubectl”, k get pods

**Section 34: Kubernetes Deployment Creation**