Our app/software used to run on a single server in olden times. Whenever we needed to scale our app or users load used to increase on our app we then had to increase the no. of server eventually, but this was not a pretty good idea as it would have increased the cost of buying new hardware , cost of storing the hardware and also maintaining it as well.

This problem was solved by a company named VMWare. How=> Virtual Machines

**What are virtual machines and what they do?**

When we talked about one application on one server. VMs solved this problem.

The problem here is Vms require their own Operating system. Even though it was far better than one app on one system, still os was a problem in this scenario.

**Flaws of virtual machine**

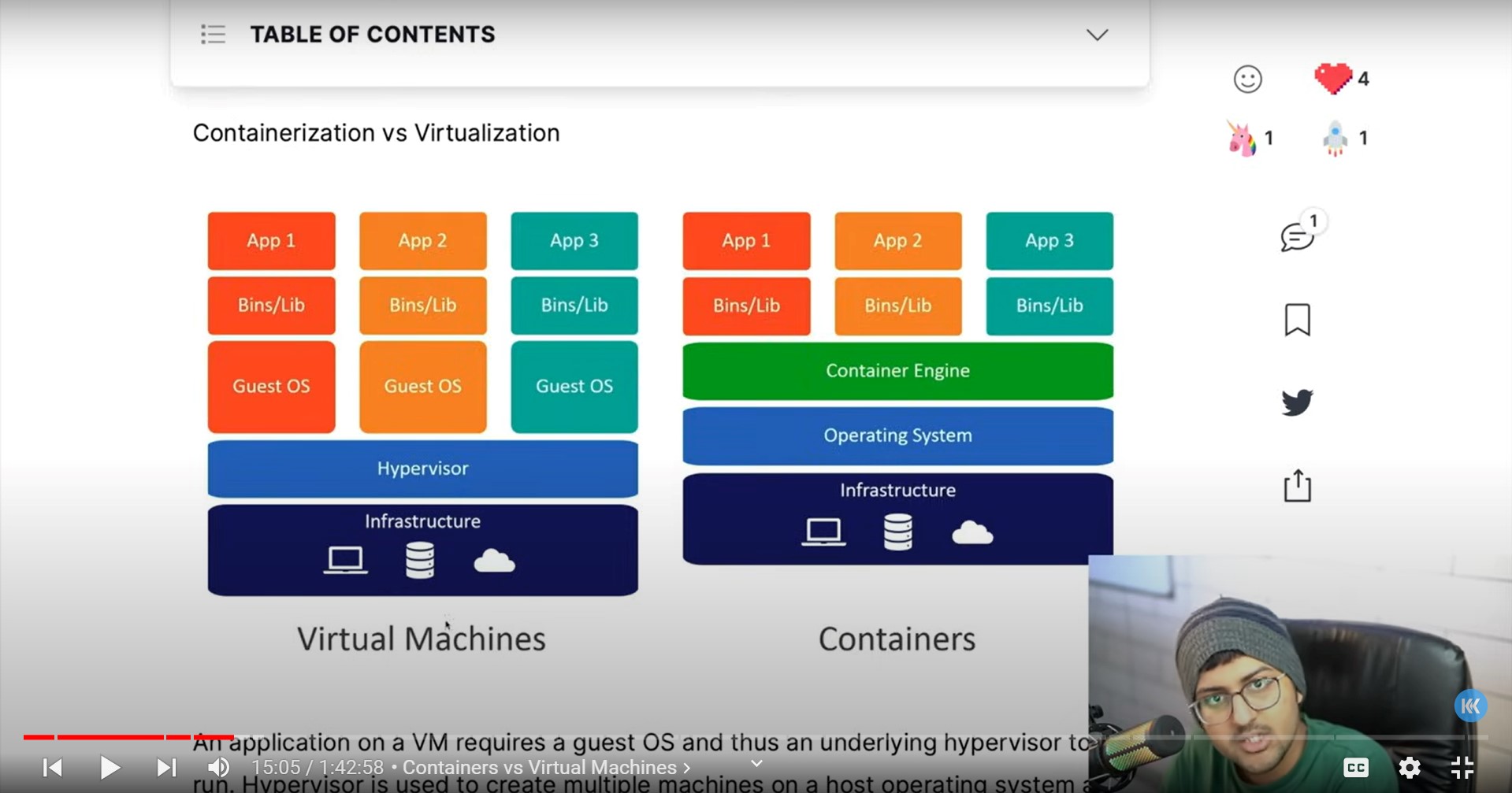
1. Having multiple projects thus operating systems => now we would have windows as well as an linux based os stored on the hard disk.
2. Slow speed of virtual machines. They were not superfast.
3. Dedicated cpu , ram for particular OS.

Dual booting - Dual booting is the process of installing and running two different operating systems (OS) on a single computer, allowing users to choose which OS to use when starting up the computer

Docker made popular containers popular for companies like google and all , they were useing containers prior to that as well but docker made it more popular.

**Difference between containers and VMs**

Containers are like sort of what VMs are doing but we need not require additional Os for that.



Microsoft worked with docker to run docker on windows

To run docker on windows

We need to install docker desktop

Most of the containers previously that became famous were linux based.

If we have a windows app that is containerized that will not run on a linux based docker/ container and vice-versa. Thus windows container will require windows and for linux app linux.

If we are using windows we need to use WSL.

For Mac: VMs & Docker Desktop

For Linux: As it is.

Now let us learn about what is docker how it helps us

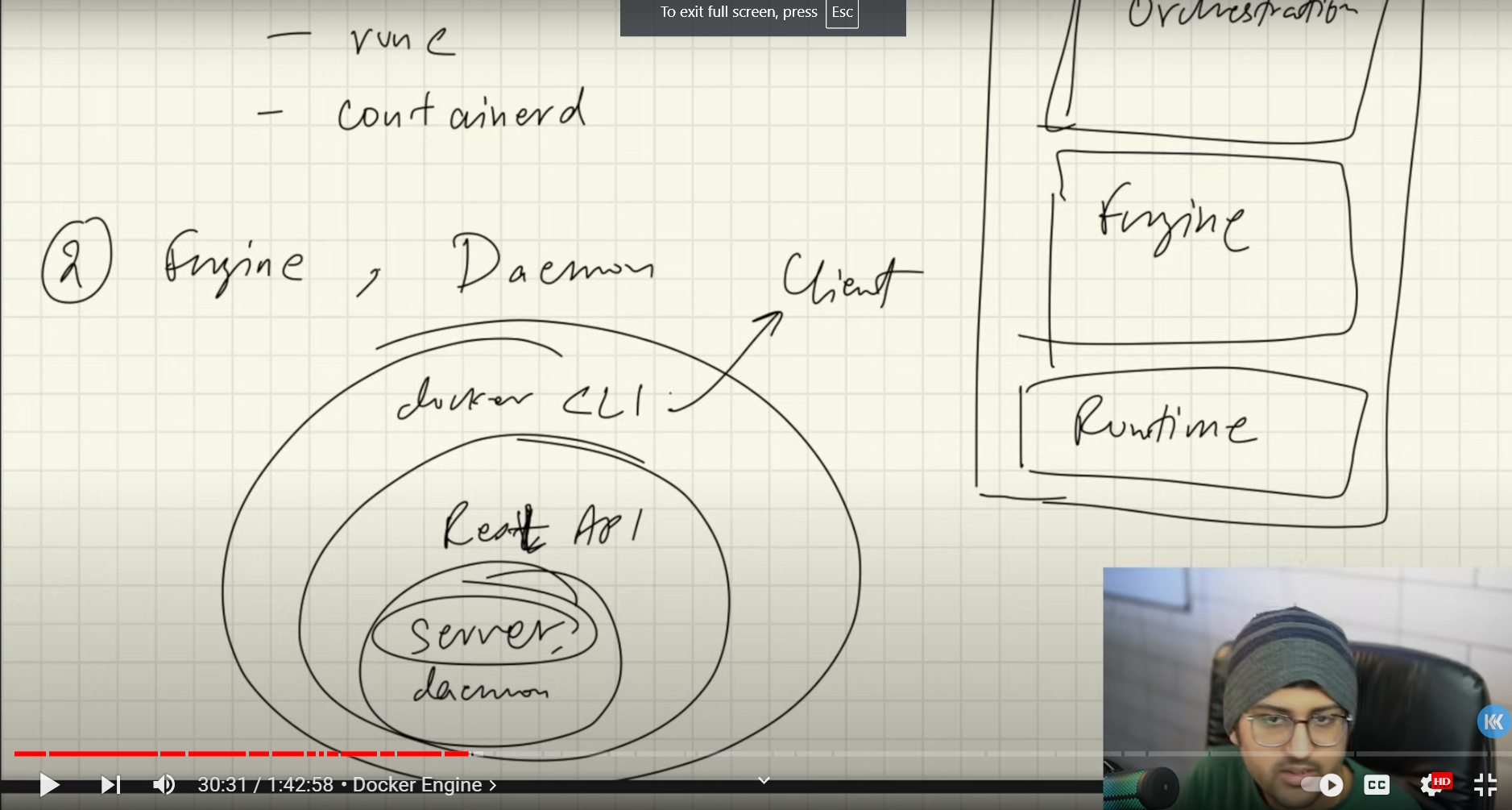
Docker helps us too run our app in isolated environments, means container will not know what is happening outside it even in the OS.

Docker allows us to do all these things.

1. Docker runtime => runtime basically allows us to start and stop containers. Runtime is of two types.

a. low level runtime - nc => to work with OS and starts and stops the container.

b. Container d => lies on higher level of hierarchy, manages the run, how to interact your container with the network, how to get data from the internet, this kind of stuff is done by this. Pulling data connecting with the network are done by container D.

2. Engine =>something that we use to interact with docker. Example – docker Daemon. It used client server architecture. See the below image for reference. We send commands through the docker CLI (command line interface) now what happens is APIs are triggered and then a request to server (daemon) is generated. Note -We can use CLI and also GUI (Graphical User Interface) as well for the same.

1. Orchestration => let’s say the app is running on 5000 containers and we want to scale it up, we can easily do it.

Let’s say half of the containers got ruined, we want to restart those containers.

Suppose we have many containers running with version 1 of the app. We can update the versions of the app in all of the containers. That’s where comes into picture orchestration engines.

Note=> Kubernetes is very much more than an orchestration engine, but it’s one of the functionality that it provides.

In easy words, orchestration engines provide functionality to manage docker engines.

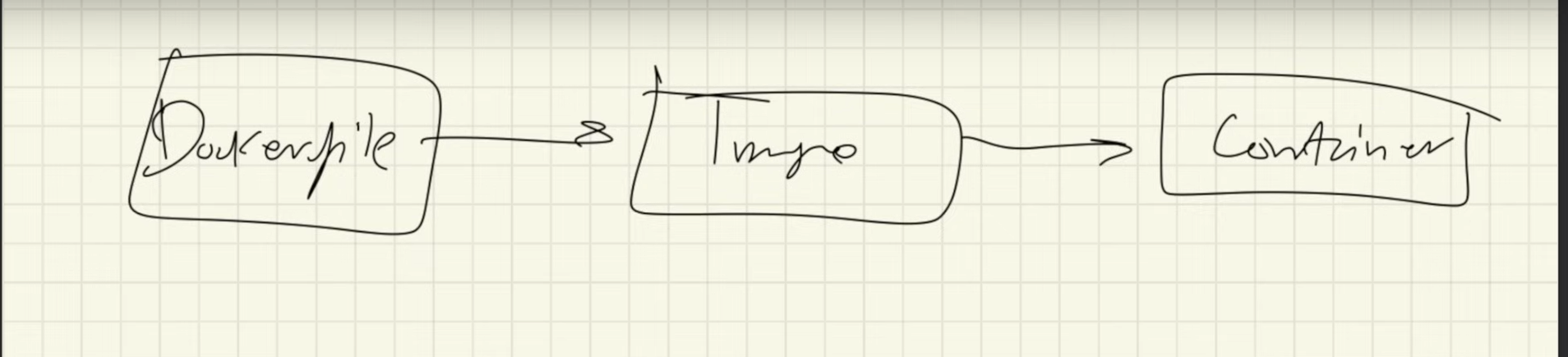
Docker image/ container image => If I want someone to run my app. I can share the docker images with them, when thy run the image, the file will start.

Image is nothing but a file that contains all the instructions.

Container is nothing but a running instance of the images. We can also create docker images, using something called docker files.

Docker files

Docker files contains a list of instructions.



Running an dockerfile we will get image and running an image we will get a docker container.

When we want to containerize our app we can create it’s dockerfile and then it will create docker image. This image can be share with others.

Difference between dockerfile and image

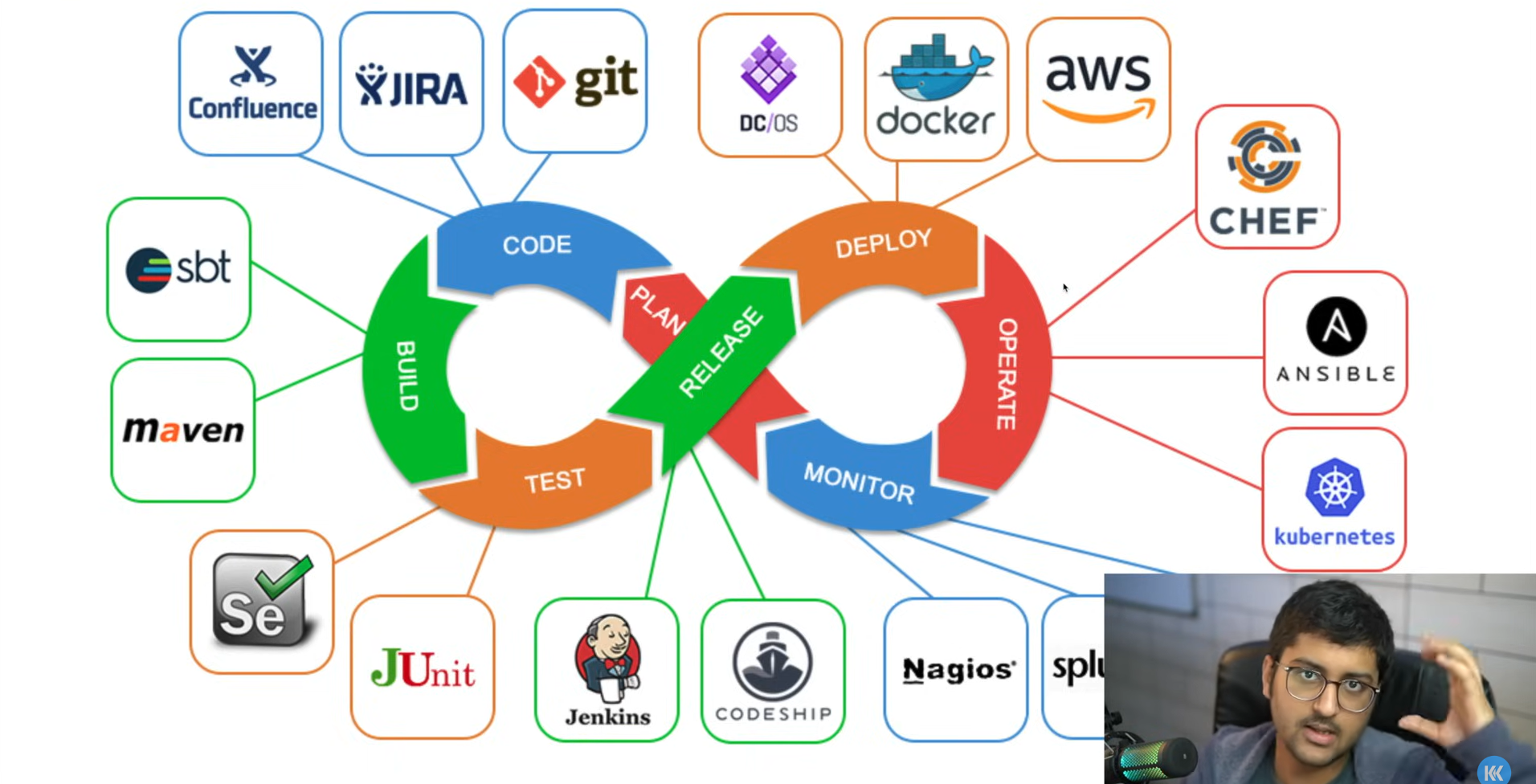
We can refer to image as class and containers as object and dockerfile can be something that is used to create image.

Open container initiative

So many runtimes were present in the market that people got confused as to which runtime to use, also there was confusion whether we can use multiple runtimes altogether or not? So open container initiative was launched. Here is the blog for that. They started two things runtime specs and image specs.

Devops – what is devops

Devops constitue of two words dev and ops. Development and operations. It refers to dev – and app is built/developed and then ops – after the app is built the app is maintained, it is tested, monitoring and maintenance on the app is done.this is like the cycle of devops along with various tools that are used.



Docker is used in devops.

Some hands-on

Docker run hello-world

Decoding the command it is on docker cli run here means run an image to create a container. The keyword following run states the image name.

Response from the docker CLI

Unable to find image ‘hello-world’ locally states that we do not have any image named the same in our local machine. So it is being downloaded from docker registry (online). Pull states downloaded the image. Then it runs the image and the image is run.

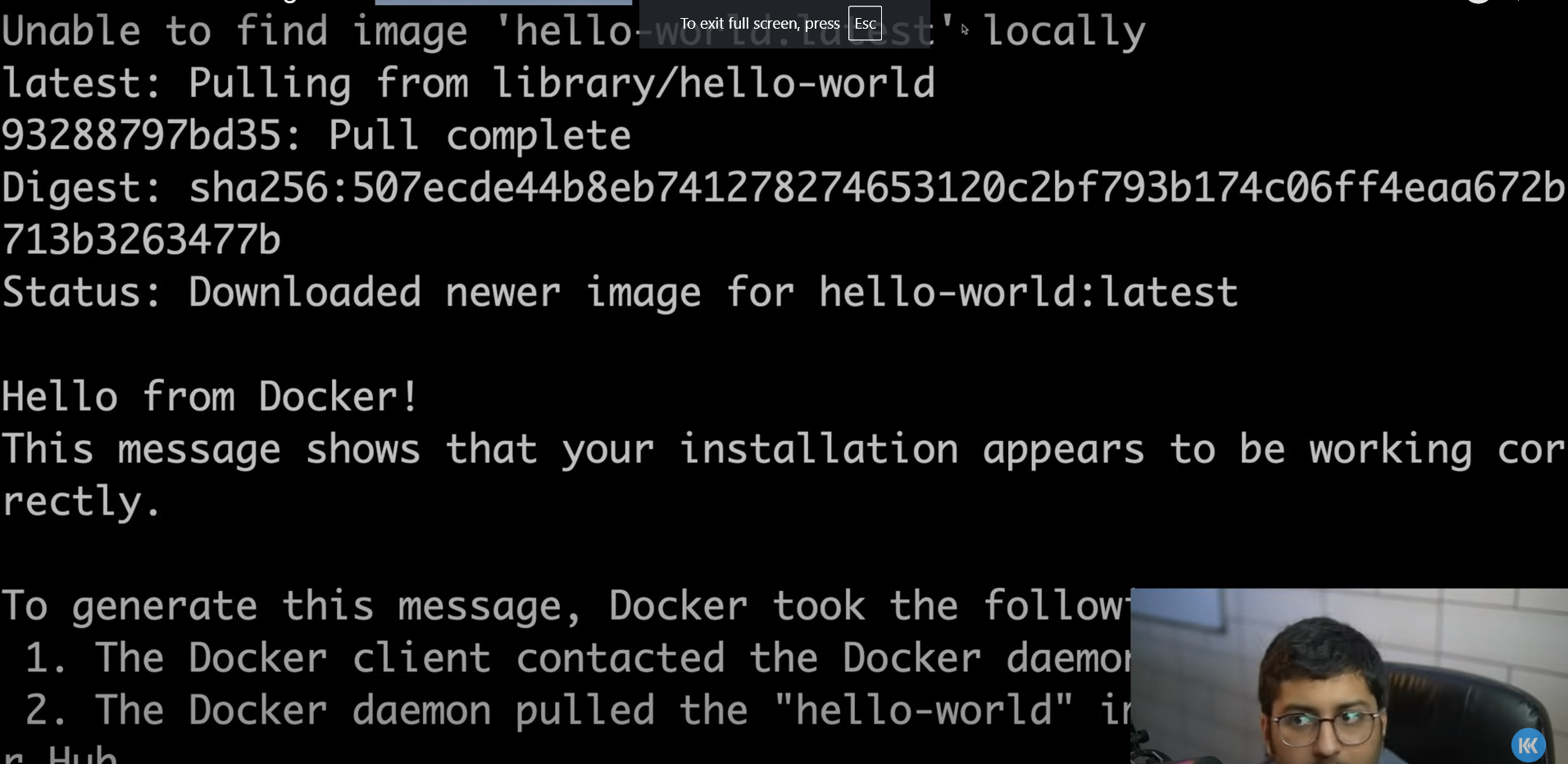
Docker images also contains operating systems files and the dependencies for that application as well.

Example => docker pull ubuntu

This will download the ubuntu for us from the docker registry. If we do not provide any particular version it will download the latest version available.

Advanced topics

How to reduce the size of docker images as well.



Docker run -it ubuntu

Decoding: docker run ubuntu in an interactive environment. (interactive environment means don’t exit out of this unless said so).

Docker images

Decoding: list of all the images downloaded in our local machine.

Note => when we download an image for the very first time then we will see it runs pretty slow for the first time, but after some time it will work pretty fast.

Docker ps

Decoding: gives list of all the docker images that are running at this time.

Docker container ls

Decoding: this will show me the list of all the containers present.

Docker container exec -it <id\_of\_container> bash

Decoding: This will execute the command to run the docker container which has a particular given id. This id must of the container that is working. Also it means the bash shell must be attached to this particular container.

Docker run mongo

Docker stop <id\_of\_container>

Decoding: To stop the container

Docker ps -a

Decoding: shows all the containers that are stopped.

Docker rm <id>

Decoding: to remove a container.

Docker inspect <id>

Decoding: gives all of the info about the container including the docker image info as well.

Docker container prune -f

Decoding: deleting all the containers that are stopped.

Docker logs –since 5s 9694

Decding: gives all the logs of the containers that have been generated in the last 5 seconds.

Docker run -d -p 8080:80 nginx

Decoding: will download the nginx image from the docker registry for the latest version if not present. After download then it will run on the port provided in the command i.e. 8080 on the local machine. Here the default port for nginx is 80.

By this I can access the nginx sever that is running on the containers on my local machine on the port number 8080 as well. Nginx works on port 80. So we can say a tunnel is created between different ports of docker and local machine.

Images: immutable, collection of files and directories.

To create an image from the dockerfile we need to build the docker file first. To build it we use this command:

Docker build -t myImage:1.01 .

Decoding: build docker -t <name\_of\_image>:<version\_number> <path of the dockerfile>

Note: here we gave the path of the docker file as (.) because it is located in current directory.

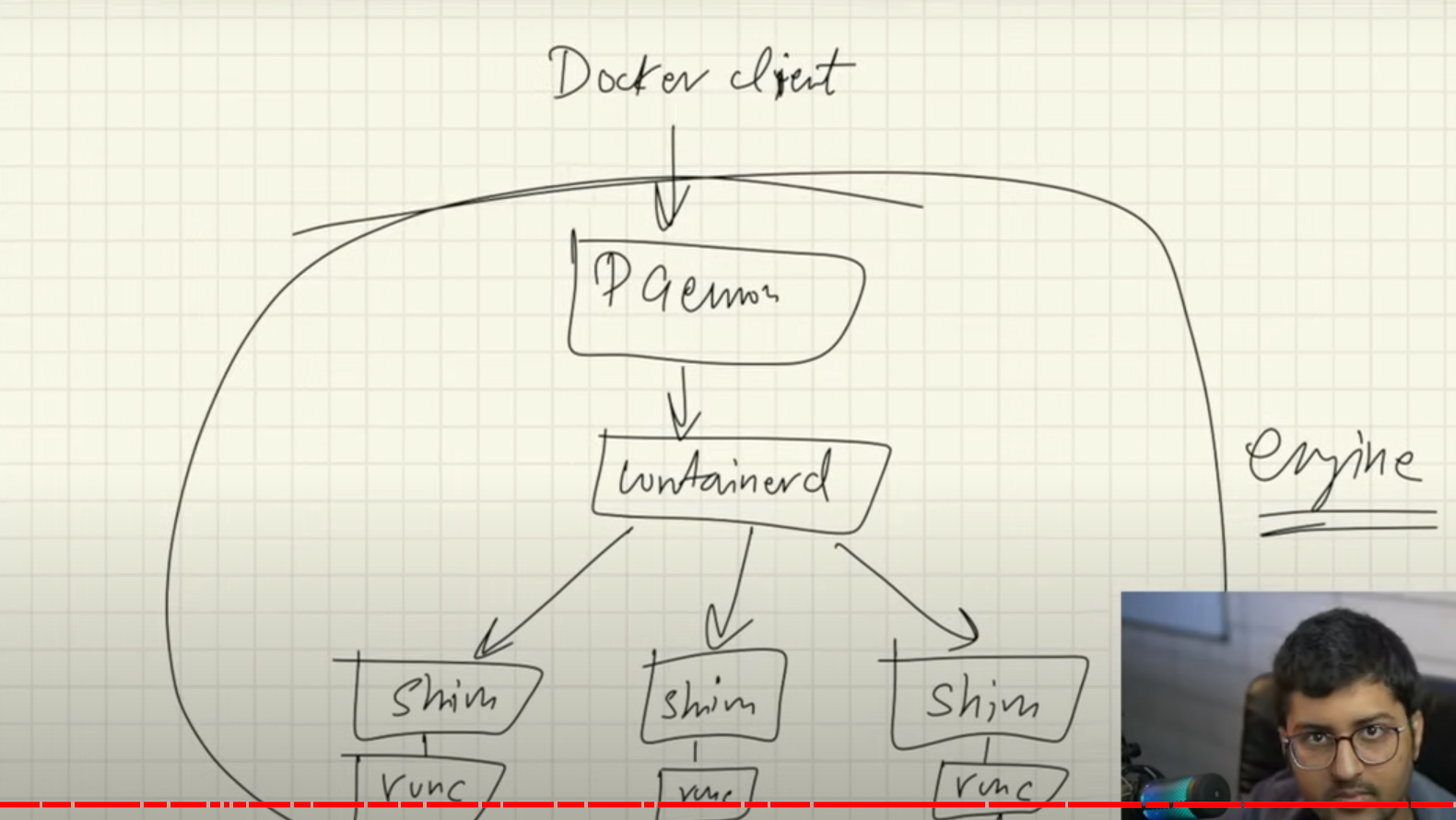
Now what if we want to share our docker iamges with someone.

Follow the steps for that:

1. All the images are stored in dockerhub. Create an account on docker hub and we can upload our image over there.
2. We can do is login docker hub account from terminal. Use command docker login. Then it will ask for Username and Password of dockerhub. Enter that.
3. Navigate to the dockerfile and build the dockerfile to create it into image using this command. docker build -t myImage:1.0. Note: this is used to give name and assign a version to the image we create. If not like that we can create an image using command docker build . Note: In the directory we are building dockerfile using the command (docker build .) we need to take care that all the files and directories in that folder are going to be built and sent to daeymon. So good practice is to move dockerfile into an empty directory and then build it there.
4. Verify that the image is created by using the command docker images. Moreover we can also try and run our image.

# **Architecture of docker**

Docker client



Runc: runtime that allows us to start/stop container. Works on lower level.

Containerd: to connect to network, store and push images, manages runc as well.

Earlier what used to happened was whenever we used to make any changes in doemon suppose we need to update it. The containers connected would automatically stop. This was a problem. So, that’s when containerd and shim were introduced.

There are also binary names as well for these files. To know what binary files are: Hint – it is related to linux.

Now when we talk about docker server and client they use Client- Server architecture. Port (2375) is used. Foor suitable connection, connection only that are suitable are allowed due to security concerns . connections secured by TLS are only allowed.

Future topics:

1. more about iamges
2. how to move our images into dockerhub
3. how to containerize applications,docker volumes
4. image layers, networking, docker compose
5. swam, docker security
6. docker file best practices.

Reference - <https://www.techwithkunal.com/blog/getting-started-with-docker>