⇒ Custom Images using Dockerfile :-

1:02:35

⇒ Suppose, we created a website using php, js, html, css, bootstrap. And to run that website we need Apache Server, MySQL, set a .conf file, and installed other dependencies that are necessary. Due to them, the website runs fine in our system.
Now, suppose we uploaded the website source code to github. Anyone, wanting to run our website in their system, will clone the repo.
Also, he needs to install all the req. softwares & dependency to run the website. Doing this too, chances are high that website may not run in his system, due to all req. dependency not prop. installed & set up.

⇒ To solve the above portability issue, we will create a Dockerfile.
We can download a, say, Ubuntu 18 image, and using that image, we created basic Ubuntu container, and in that, installed all our req. softwares & dependencies, and created its custom image. Using this image, anyone just can create a container, and in that container our website will run just fine, Absolutely Guranteed !

⇒ In the Dockerfile, we will write, after downloading the Ubuntu 18 base image, and what changes we made — step-by-step, to obtain our custom image, in which our website runs just fine.

The Dockerfile will contain commands & info of all steps, to obtain our custom image, starting from Ubuntu 18, here in this case!

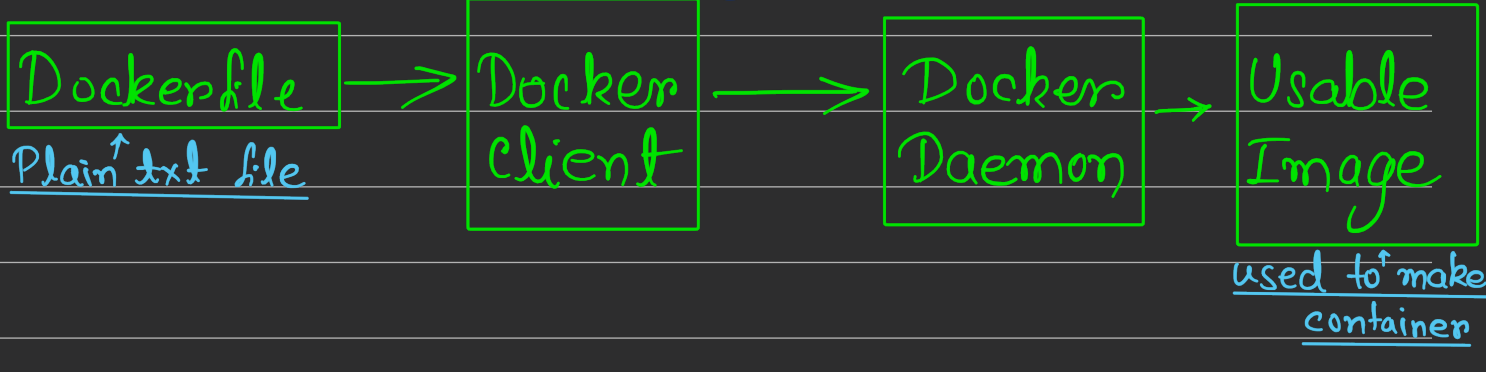
Then, we either push that custom image to Docker Hub, or give that image to the one wanting to run our website. He can the build that image, & run that container.

Dockerfile :- contains all the info for running our software in a base container,
Ubuntu 18 in our example above.

1:07:50

⇒ Steps to convert Dockerfile → Image:

⇒ The image that we will create from Dockerfile, will be the custom image for our specific software.



How to create a custom image?

1) Choose a base image.
2) Run commands for your software.
3) Specify the base command (startup command)

Creating Dockerfile :-

We will take alpine as our base image: Alpine is a very lightweight linux-distro, with very minimal softwares installed.

(1) First, we need to open terminal in the directory in which we want to work

(2) Create a file named Dockerfile (→ capital)
In it we will code now / put the commands / info.

(3) Put the following code into the Dockerfile :-

```
# Specify the base image
FROM alpine
# Download & install dependencies
RUN apk add --update redis
# Setup the startup command
CMD ["redis-server"]
```

- `#` lines → comments
- `FROM` → keywords; `alpine` → base-image
- `RUN apk add --update redis` → installs redis inside alpine container

With the above code, the Dockerfile is ready, and capable of making a custom image.
Save it & close

(4) Now, to build the image, run the following command in terminal, in same directory as of the Dockerfile :-

```
docker build .
```
(. → dot.)

We can see the commands being executed one-by-one
At last, it will build our image, and will give us the respective image id.

(5) We need to copy the image id obtained in the last step (say f39b98144cfe), and run it:-

```
docker run f39b98144cfe
```

Understanding the dockerfile code :-

- FROM command is used to specify the base image. When built, it will be downloaded from dockerhub, and will be ready for changes.

- RUN command takes the command specified after it, and executes it in the intermediate container (base that we selected previously).

After the command is executed, it makes apt changes to the file system snapshot of the base container.

⇒ CMD is used to specify / run the startup command in the container.

⇒ When executing commands of Dockerfile while building, container is spin in one step, changes are made into it running commands, and then the changed file system snapshot is taken and saved, and running container is removed.

With the new file system snapshot, a new container is being made, and in it the next command is run. Due to it, again the FS snapshot changes. It is saved (new one), and prev. container is removed, and so on the process continues till last step, where startup command is added.

⇒ Tagging Images :-

⇒ Till now, after building our image, we got an image id that identifies our image, and helps to create container of it.

We can give name to our images, such that they are identified with that image name. This is called Image Tagging.

⇒ Command to tag images :-

`docker build -t debanjan/redis:latest .`

(dot)

- `-t` : flag for tagging image
- `debanjan/redis:latest` : tag that we want to set for our image.

Thus, we need to set the tag/name of our image, while building the image using docker build

⇒ The tag can be set to any name/string.
The above tag is given according to convention :-

`... debanjan/redis:latest ...`

- `debanjan` : username of user who made the image
- `redis` : the main software being added/present in the image
- `latest` : Version of the image

These naming convention helps others to understand what the image contains, and version.

⇒ We can then containerise the above image as :-

`docker run debanjan/redis`

By default pulls the latest image.

To containerise any specific version :-

`docker run debanjan/redis:2.0`

(2.0 : version to pull)

docker commit command :-

We first ran a shell inside the alpine container as follow :-

```
[ docker run -it alpine sh
```

This command will attach our terminal input output to alpine containers input output, & will give us a shell running.

In it, we installed redis as follow :-

```
[ apk add --update redis
```

Now, we perform a commit command as follows :-

```
[ docker commit -c 'CMD "redis-server"' abef842d
```

(abef842d : docker id of the running alpine container in which we installed redis)

This command states that take the file system snapshot of how the container is currently, and commit it with the provided startup command (as string)

It will return a container id.

CONTAINER is an isolated environment that stores all our setup files (related to a product) and give it to our client for testing / deployment purpose.