Class 8 - Writing Docker file

### Container

- An isolated application uses an engine to run itself as a fully operable program utilizing a shared resource.

- Container engine runs one or more containers allocating different processes/networks for each container.

- We can create multiple containers using one image. Each container is an instance of the image.

> If an image isn’t in the local server, docker will pull the image from docker hub at first. Nginx run at port 80 by default. If docker is installed locally we can’t access port 80 from host machine.

>

`docker run nginx`

`docker run -p 5000:80 nginx`

> [localhost:5000](http://localhost:5000) forwarded to port 80 of the docker container. To see the currently running container -

>

`docker ps`

`docker ps -a` To see all containers.

> ContainerID → unique id assigned to a container, Image → Image name that is being used, the last column is the Container name.

>

![Screenshot from 2024-03-12 15-01-13.png](https://prod-files-secure.s3.us-west-2.amazonaws.com/f3198d70-ba65-49f1-915e-f719238b300e/1a86f5c7-86cd-4908-975c-1d5d5bcbda54/Screenshot\_from\_2024-03-12\_15-01-13.png)

> When we dockerize an application, we create a docker image. Docker images isn’t in the running mode. We have to run the image to create an instance(container) out of it.

>

`docker images`

- See all the images stored in local repository

- If we use an image from docker hub and don’t mention its tag, it will pull the latest version of that image.

- When we create a docker image of our application, we can add a tag and assign a name to it\*\*.\*\*

`docker run -p 5000:80 --name server1 nginx:tag`

- In production, be specific to an image. Never use the latest tag or without a tag, it may crash the application later because of incompatibility issues.

`docker ps | grep -i server1`

- We can’t delete an image if it's any instance(container) is running.

`docker rmi imageName/Id`

- `docker rm containerName/Id` removes a container

- Enter into the container-

`docker exec -it server1 bash`

> We know from the documentations of Nginx, the homepage we browsed is in /usr/share/nginx/htm/ folder and index.html is the homepage. If we changed this page then the new page will be shown in the browser.

>

- Create a folder `mkdir docker-demo` and `cd docker-demo` . Now create two files named index.html and Dockerfile into it. `touch index.html Dockerfile` write whatever into the index.html

## Dockerfile

- Dockerfile has some predefined directives.

- `From ImageName` using an Image

- `Copy fileOfRootFolder toWhateverDirectory`

- As a DevOps, I will communicate with those who have been working on the project for the following processes of the application -

- Install Dependencies (command)

- Build App (command)

- Run App (command)

- `Run install command`

### Create a simple Dockerfile

- Write in the docker file

```docker

From nginx:latest

Copy index.html /usr/share/nginx/htm/index.html

```

- `docker build -t demo:v1 .`

- docker build is now creating a custom image named demo with tag v1 which will look for the dockerfile from (.) current folder

- After the build is done, run `docker run -p 5000:80 demo:v1` means we are using demo:v1 image to create a container and mapping nginx 80 port with [localhost](http://localhost) 5000.

- We can use the demo:v1 image to create another instance(container) similarly. `docker run -p 8000:80 demo:v1`

- Browse [localhost:5000](http://localhost:5000) and [localhost:8000](http://localhost:8000)

### Dockerfile for JS project

- We made a react project and want to dockerize it. Write the following into the Dockerfile.

```docker

FROM node:20.01 #see node stable version

WORKDIR /react # create react folder and cd into it within docker container

COPY . . # copy from currentDir to docker workDir

RUN npm install # installing dependencies

RUN npm run build # build the code

CMD ["npm","run","start"] # expose the app port when running the container

```

- run `docker build -t react:v0 .` to build the image. if you have multiple docker file in the current directory then you can specify a docker file say dockerfile.test

- `docker build -t react:v1 -f Dockerfile.test .`

> If you use same docker file to build multiple images it uses cache of same steps to reduce building time and only do process for the newly steps.

>

- `docker run -p 6000:3000 --name react-client react:v1`

> If you mistakenly run the container but forget about port forwarding, then kill the process. `docker kill react-client` or stop it `docker stop react-client` then delete it.

>

> Docker file system utilise caching for each layer. Any layer that doesn’t change, won’t go through build process while building it next time.

>

```docker

#version 1

FROM alpine:latest # lightweight os for Linux

CMD ["echo", "hello user"] #this command will be executed only \

when running the container, not along the building process

```

`docker run version1` Can’t pass any argument, If so, it will show an error.

```docker

#version 2

FROM alpine:latest

ENTRYPOINT ["echo", "hello user"] #this command does the same thing,\

additionally, append an argument also.

```

`docker run version2 world` return will be - hello user world

```docker

#version 3

FROM alpine:latest

ENTRYPOINT ["echo", "hello"]

CMD ["user"]

```

`docker run vesion3 world` return - hello world. but `docker run version3` return - hello user. Guess what happens!

> Build each file before running. If you want to ignore any file/folder while build process write it within `.dockerignore`