docker run -p 5000:80 nginx

docker ps

docker ps -a

process – when a program is in the running state, it is called process.

sudo docker images

When a docker is pulled from the docker hub, it downloads the latest one. But same docker image can be present with different tag. So, when to delete a docker image, if there is more than one docker image version for the same image (for example, multiple images for nginx), I have to mention the tag, which I want to delete.

[A specific version of an image can be pulled from the docker hub. In this case, multiple images with different version will be present. e.g. 'docker run nginx:1.19.0', it will run , if not present, it will first pull and then run nginx 1.19.0]

sudo docker rmi nginx:latest

[for multiple version , use 'image name: tag', if tag is not mentioned, the image with 'latest' tag will be deleted.]

sudo docker rmi nginx

[List the docker images

sudo docker image ls

Inspect an image-

sudo docker image inspect 'image id or name'

sudo docker image inspect nginx

In the output, there is 'Data' which shows the directory.

'/var/lib/docker' is the directory. There is a number of directories inside it.]

sudo docker logs 'container name or id'

sudo docker logs first

will show logs of the docker container. The previous command will show all the current logs. **'logs -f'** will show and be in ready state to show if any new logs are generated.

Custom docker image can be created from the docker file. The reason is, whatever change is made in the docker container is not persistent, so a docker file with the required features will create a docker image to serve necessary purpose. Now create a docker file. I have given the file name **'Dockerfile'**.

FROM nginx:latest

[‘FROM’ keyword is used in the dockerfile to specify the base image for building a new docker image. In this context, I am using the latest nginx image as the base image.]

COPY index.html /usr/share/nginx/html/index.html

[The ‘COPY’ command in a dockerfile is used to copy new files or directories from a source on a local system to the filesystem of the container. In docker, it needs to mention in which name the file will be saved.]

So all the lines are –

FROM nginx:latest

COPY index.html /usr/share/nginx/html/index.html

I have kept the file in /home/amirul/dockerfile directory.

This time, I shall build the image from the docker file.

sudo docker build -t second:v1

[‘build’ command will create the image from the dockerfile.

-t : tag; if it is not mentioned docker will provide a name without version.

second:v1 ‘name:version’

. : directory location of the docker file. In my case, it’s the location in the current directory. So I use ‘.’ (dot)]

[By default, docker looks for the filename ‘Dockerfile’. If the filename is different, then ‘-f’ option is used. In the example, the dockerfile name is ‘test’. So the complete command is

sudo docker build -f /home/Amirul/files/test -t try:v1 /home/Amirul/files]

Now deploy a react in a docker container. The deployment of an app in general has 3 steps

a) Install Dependencies (e.g. npm install etc.)
b) Build app (e.g. npm build etc.)
c) Run App/Serve App (e.g. npm run start, npm run preview etc.)

The location of the app in the host machine **is D:\Devops(poridhi.io)\Module3\calss-01\multi-step-dockerfile\react-v0**]

A devops engineer to deploy an app in a container has to consult with the software engineer about the above mentioned procedures.

In this docker container, I will

node:latest
Copy all the source code to the container

npm install – install dependecnies

npm build – build the code

npm run preview – run the code

[npm is the node package manager]

The philosophy of docker is, at one time one application will run. Now I will build the image without running the code.

sudo docker build -t react:v0 -f Dockerfile.test .

[first time, the command above will take some time]

Now, apply

sudo docker run react:v0

Nothing will happen, as I didn't run the code in the 'Dockerfile.test'. If the image is built once, for later if it will use same dockerfile, it will cache the output of every line. After the first time, it will use the cache, so the build process will be faster. (This is called Layered Architecture.) This statement is true if no change has been made in the dockerfile.

Another side is, I am copying all the files from 'react-v0' directory. If there is any change in any file, it will again start building. In this example, upto 'WORKDIR /app', it will use the cache, then from 'COPY . /app', it will start building again.

Now again, make the change in the 'Dockerfile.test', add the following line –

RUN npm run preview

In the CLI

sudo docker build -t react:v0 -f Dockerfile.test .

The build process is going on without stopping. The reason is, I am running the app while building the docker, So, instead of build completion, it is trying to run the app and it is going on forever. The process has to be, the app will run when a container will be created from the docker image.

Modify the last line of the 'Dockerfile.test'

CMD ["npm", "run", "preview"]

[The line starts with 'CMD', executes the command when a container is run from the docker image.]

Now, again build the docker image –

sudo docker build -t react:v0 -f Dockerfile.test .

This time, the build process is complete.

sudo docker run react:v0

Now, ‘kill’ the docker container and with port forwarding, the application is listening on docker port 4173.

sudo docker kill ‘container name’

sudo docker run -p 7000:4173 react:v0

The ‘Dockerfile.test’ is the following –

FROM node:latest

#WORKDIR will mkdir app and cd app

WORKDIR /app

 # copy all the source code from the current directory to the container

COPY . /app

 # npm install dependencies

RUN npm install

 # npm build – build the code

RUN npm run build

 # npm run preview – run the code

CMD [“npm”, “run”,”preview”]

[Instead of ‘CMD’, ‘ENTRYPOINT’ can also be used.]

Now, create two docker files name Dckerfile.v1 and Dockerfile.v2 respectively.

In Dockerfile.v1

FROM alpine:latest

CMD [“echo”, “Hey User”]

In Dockerfile.v2

FROM alpine:latest

ENTRYPOINT [“echo”, “Hey User”]

sudo docker build -t demo:v1 -f Dockerfile.v1 .

sudo docker run demo:v1

Output:

Hey User

Now , again run demo:v1 image with an argument

sudo docker run demo:v1amirul

Output: docker: ……..

exec:”amirul”:executable file not found in $PATH:unknown

Now build the docker image using Dockerfile.v2.

sudo docker build -t demo:v2 Dockerfile.v2 .

sudo docker run demo:v2

sudo docker run demo:v2 amirul

Output: Hey User amirul

“ENTRYPOINT” takes the argument and append it to the output.

sudo vim Dockerfile.v3

FROM alpine:latest

ENTRYPOINT [“echo”, “Hey”

CMD[“User”]

sudo docker build -t demo:v3 -f Dockerfile.v3

sudo docker run demo:v3

Output: Hey User

sudo docker run demo:v3 amirul

Output: Hey amirul

Here, if no argument is passed, it will use the default value. Meaning, ‘ENTRYPOINT’ will use the value from CMD. Here it is “User”. If argument is passed, the ‘ENTRYPOINT’ will append the argument.

In ‘ENTRYPOINT’, the first command has to be executable. The command has to have a meaning. Here the first command is ‘echo’ which prints whatever is written after it.

During docker build, by default docker will read every file and every directory. In the example, I am using a dockerfile and modify it several times. So everytime, I build an image, it will read all the files and directories of ‘react-v0’. The unnecessary files and directories can be included in the ‘.dockerignore’ file.

The lower the image size, the better.

If there is any change in the file, let's say due to any simple modification, total process will start again. In the present scenario, if I make any change in the html file, it will again start installing npm. This can be very time consuming.

One way can be, first figure out dependency tracking files. Here those are package-lock.json, package.json. So first copy those files. Add the following the line in the docker file.

COPY package*.json .

Then install the dependency which takes longer time. So, here place the line 'RUN npm install' before copying the files. (Check it in the Dockerfile.test.)

Node modules take a lot of time, so add it to the .dockerignore file. It is the best practice to install node modules inside the docker container. If this procedure is not followed, many issues can arise like permission issue. When a docker image is built, 'dist' folder is generated automatically.

In the first line

FROM node:latest

The 'latest' will download the latest version of the node. The problem happens, if the application depends on a particular node version. Every time the docker will build a new image downloading the latest node. The app may not work properly.

Alpine version of the node image can be used as it is smaller in size.

FROM node:20-alpine