Assignment-Week 4

Task 4: Create a docker image from multiple methods likes Dockerfile, running containers.

1. Introduction

Docker images are the blueprint for running containers. They include everything needed to run an application—code, runtime, libraries, and dependencies. This document explores two common methods to create Docker images:

- Using a Dockerfile
- Committing changes from a running container

2. Method 1: Using Dockerfile

A Dockerfile is a script with step-by-step instructions on how to build a Docker image.

Steps to Create an Image Using Dockerfile

1. Create a Dockerfile
   Example for a simple dockerfile:

```
# getting base image ubuntu

FROM ubuntu

LABEL maintainer="ajay ingle <ajayingle3878@gmail.com>"

#Update package list

RUN apt-get update

# default command

CMD ["echo", "Hellow World..!"]
```

```
MINGW64:/d/DevOps/Docker/dockerfiles
# getting base image ubuntu
FROM ubuntu

LABEL maintainer="ajay ingle <ajayingle3878@gmail.com>"

#Update package list
RUN apt-get update

# default command
CMD ["echo", "Hellow World..!"]

~
```

2. Build the Image
   $ docker build -t my-app .

```
ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$ vim Dockerfile

ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$ docker build -t my-app .
[+] Building 33.5s (6/6) FINISHED                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                       0.1s
 => => transferring dockerfile: 230B                                       0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest           0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => CACHED [1/2] FROM docker.io/library/ubuntu:latest                      0.0s
 => [2/2] RUN apt-get update                                              32.9s
 => exporting to image                                                     0.3s
 => => exporting layers                                                    0.2s
 => => writing image sha256:7c9df8748fef5c07fb4a8d7fdef502b6af5657375ac8b  0.0s
 => => naming to docker.io/library/my-app                                  0.0s
```

3. Run the Container
   $ docker run -d -p 3000:3000 my-app

```
ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$ docker run -d -p 3000:3000 my-app
2efddaf0fe59a26653a8afc1386c6e7d3584b6ad6d63afa11d3e3fd25c569e08

ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$
```

✅ **Advantages**

- Reproducible builds
- Easy to version and share
- Automation-friendly for CI/CD pipelines

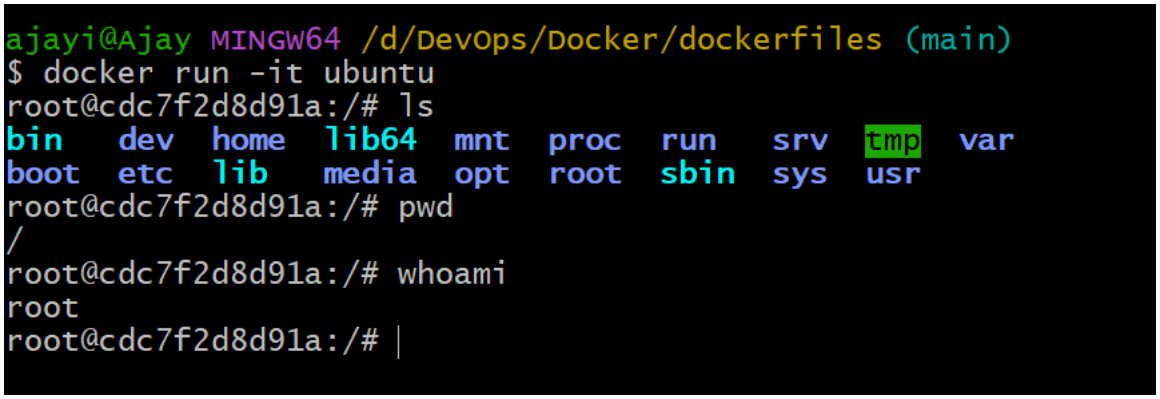
3. Method 2: Commit from Running Container

This method involves starting a container, modifying it, and then committing it to an image.

Steps to Create an Image from a Running Container

1. Start a Base Container

   $docker run -it ubuntu



2. Install Packages or Modify the Container
   Inside the container:

   $ apt update && apt install -y nginx

```
root@cdc7f2d8d91a:/# $ apt update && apt install -y nginx
bash: $: command not found
root@cdc7f2d8d91a:/# apt update && apt install -y nginx
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:3 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages
 [1566 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [19.3 MB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [
1110 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages
 [22.1 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1159
 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [331 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble/restricted amd64 Packages [117 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble/main amd64 Packages [1808 kB]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages
```

3. List Running Containers and Commit

   $ docker ps -a

```
ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$ docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS
      PORTS                    NAMES
7529f278c873   registry:2   "/entrypoint.sh /etc…"   58 minutes ago   Up 58 minu
tes   0.0.0.0:5000->5000/tcp   myregistry

ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$
```

   $ docker commit <container_id> my-custom-image

```
ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$ docker commit 7529f278c873  my-cutome-image
sha256:a30b324f0b7055af439813be3a8d9dd23fa228a1745fc7ac097372cc58a44e8d

ajayi@Ajay MINGW64 /d/DevOps/Docker/dockerfiles (main)
$
```

4. Run the Committed Image

   $ docker run -d -p 80:80 my-custom-image

✅ Advantages

- Quick for prototyping
- No need to write a Dockerfile for minor changes

⚠️ Limitations

- Not reproducible or scalable
- Harder to track or document changes

4. Conclusion

Both methods serve different needs:

- Use **Dockerfile** for consistent, scalable, and maintainable image creation.
- Use **container commit** for quick, ad hoc changes or experimentation.

Understanding these approaches empowers developers to choose the right method for each phase of development and deployment.