Name : Ali Ahmed Ali Mohamed

Answer Task Two
DevOps Assignment 2: Docker

1-Answer_Theory Questions

A - Docker Fundamentals

- Docker_container:

- An open platform for developing, shipping, and running applications.
- Docker separates your applications from your infrastructure

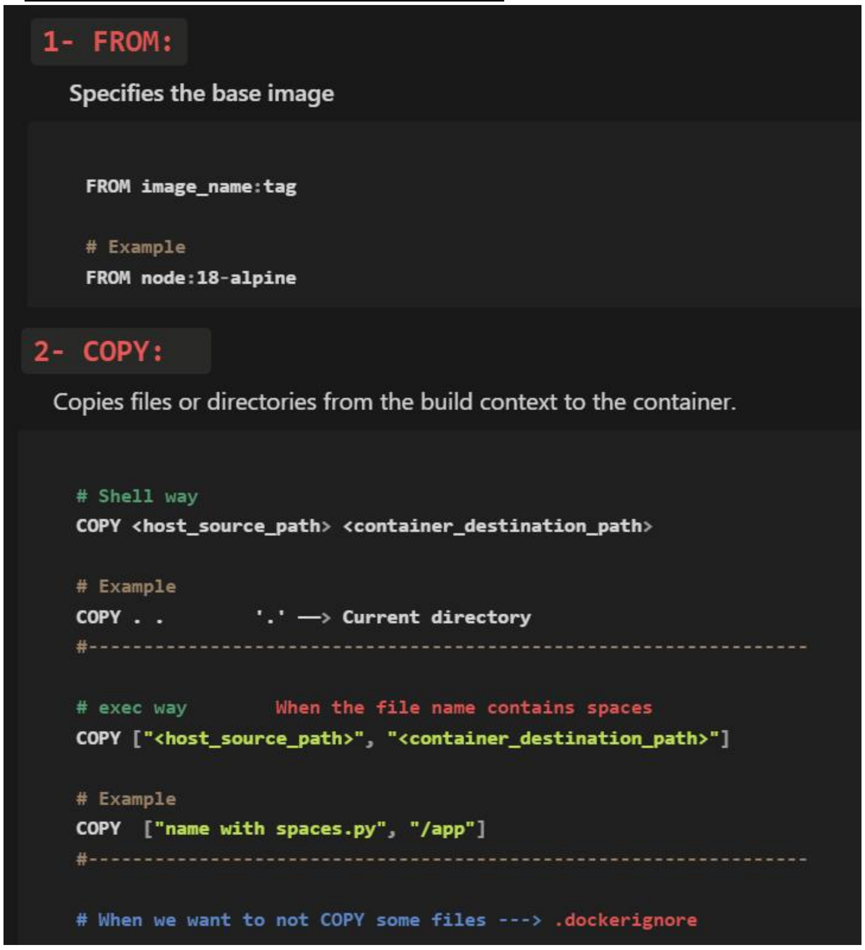
- Different from a virtual machine (VM)

- Virtual are not lightweight
- Virtual machines package the entire guest OS.
- Docker uses the host kernel and a minimal OS that can be shared between containers

- Purpose of a Dockerfile

- A Dockerfile is a text file used to automate the creation of Docker images. It contains a set of instructions that define how an image is built, including:

- FROM, COPY, RUN, and CMD



1- FROM:

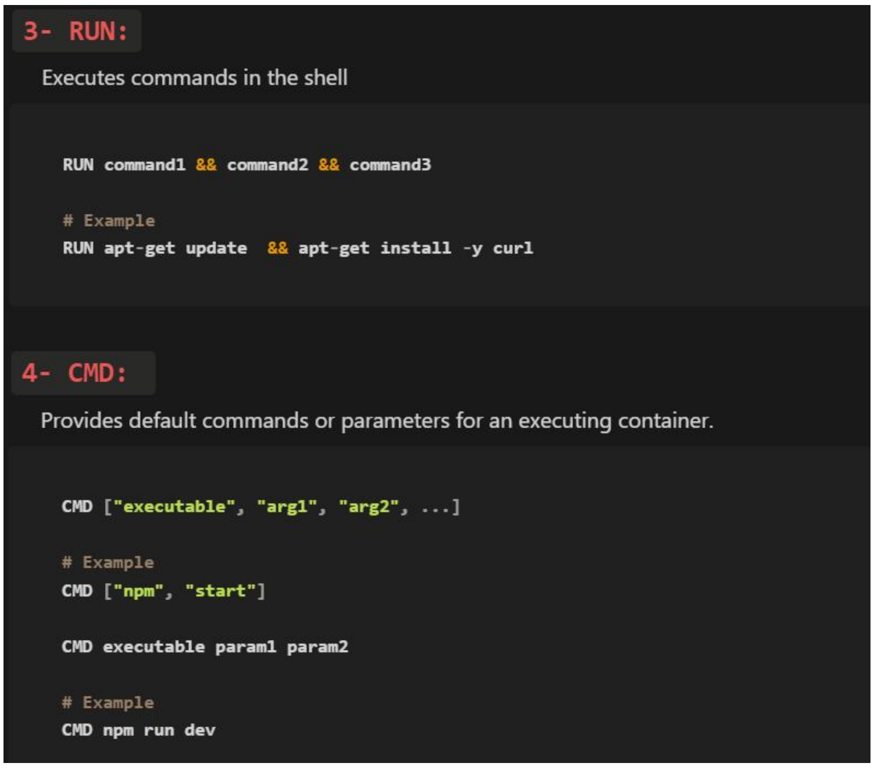
Specifies the base image

```
FROM image_name:tag

# Example
FROM node:18-alpine
```

2- COPY:

Copies files or directories from the build context to the container.

```
# Shell way
COPY <host_source_path> <container_destination_path>

# Example
COPY . .          '.' —> Current directory
#------------------------------------------------------------------

# exec way         When the file name contains spaces
COPY ["<host_source_path>", "<container_destination_path>"]

# Example
COPY  ["name with spaces.py", "/app"]
#------------------------------------------------------------------

# When we want to not COPY some files ---> .dockerignore
```


3- RUN:

Executes commands in the shell

```
RUN command1 && command2 && command3

# Example
RUN apt-get update  && apt-get install -y curl
```

4- CMD:

Provides default commands or parameters for an executing container.

```
CMD ["executable", "arg1", "arg2", ...]

# Example
CMD ["npm", "start"]

CMD executable param1 param2

# Example
CMD npm run dev
```

B - Image Management

- the layers of a Docker image

Key Characteristics of Docker Layers:

- Read-Only Layers: Each layer is immutable and read-only. When a new layer is added, it doesn't modify previous layers but overlays changes on top of them.
- Union File System: Docker uses a union file system to merge layers into a single cohesive view during runtime.

Optimization Benefits:

- **Space Efficiency:** Layers are reused across images. If multiple images share a base layer (e.g., FROM Ubuntu:20.04), it's downloaded and stored only once. Only the layers that change need to be updated or re-downloaded.

- **Build Caching:** Docker caches layers based on their instructions in the Dockerfile. If a layer doesn't change (e.g., RUN apt-get install remains the same), Docker skips rebuilding it, speeding up builds.

- **Layer Sharing:** When running containers, Docker shares layers between them, reducing the storage footprint and memory usage on the host system.

- Docker volumes

- Volumes mount a directory on the host into the container at a specific location
- Can be used to share (and persist) data between containers
- Directory persists after the container is deleted

C - Networking in Docker

- Docker handles networking

through network drivers that manage container communication. By default, containers use the bridge network, allowing isolated communication between containers on the same network.

- **Bridge:** Containers communicate via a virtual subnet ; external access requires port mapping.
- **Host:** The container shares the host’s network stack, eliminating isolation but improving performance.
- **None:** Completely disables networking; the container has no external or internal network access.

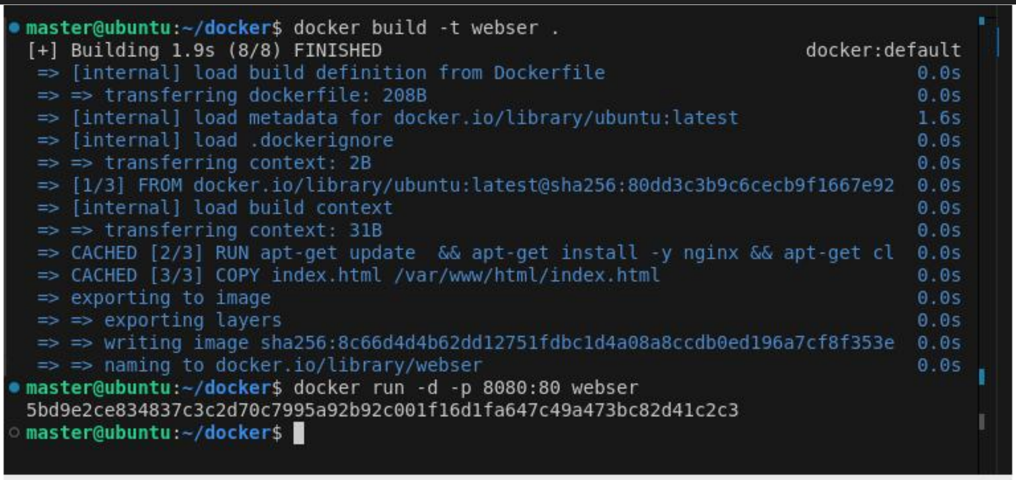
- Describe How you would configure container-to-container communication within a Docker network.

1- Create a custom Docker network using docker network create my-network.
2- Run containers with --network my-network to connect them to the same network.
3- Docker’s internal DNS lets containers communicate by name (**e.g**.,http://app2:port).
4- This ensures secure, container-to-container communication without host exposure.

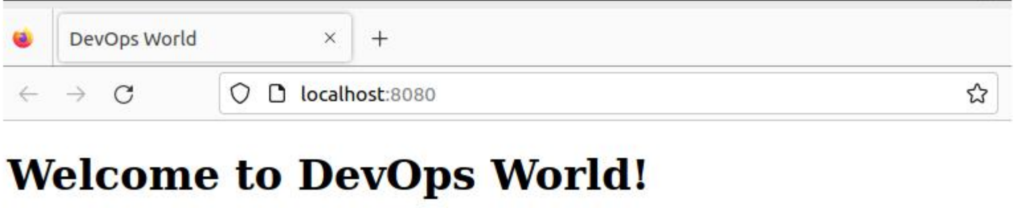
2-Answer_Practical Task

(a) Dockerfile Creation :

```
# Dockerfile
FROM ubuntu

RUN apt-get update  && apt-get install -y nginx && apt-get clean

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

EXPOSE 8080

CMD ["nginx", "-g", "daemon off;"]
#---------------------------------------------------------------------------
# index.html
<!DOCTYPE html>
<html>
<head>
  <title>DevOps World</title>
</head>
<body>
  <h1>Welcome to DevOps World!</h1>
</body>
</html>
#----------------------------------------------------------------------------
# Commands
# $ docker build -t webserver .
# $ docker run -d -p 8080:80 webserver
```

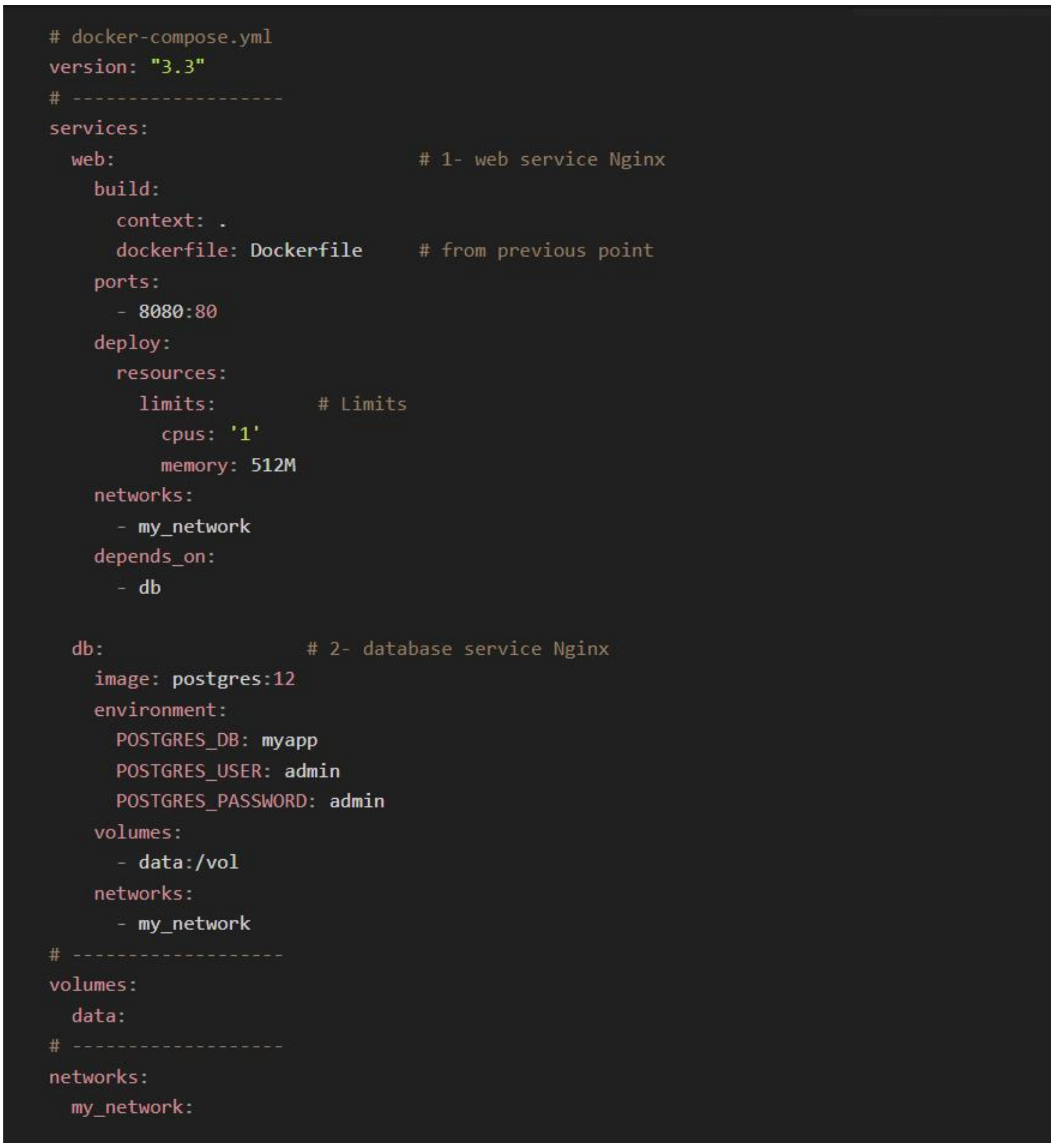


```
master@ubuntu:~/docker$ docker build -t webser .
[+] Building 1.9s (8/8) FINISHED                                         docker:default
 => [internal] load build definition from Dockerfile                              0.0s
 => => transferring dockerfile: 208B                                              0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                  1.6s
 => [internal] load .dockerignore                                                 0.0s
 => => transferring context: 2B                                                   0.0s
 => [1/3] FROM docker.io/library/ubuntu:latest@sha256:80dd3c3b9c6cecb9f1667e92   0.0s
 => [internal] load build context                                                 0.0s
 => => transferring context: 31B                                                  0.0s
 => CACHED [2/3] RUN apt-get update  && apt-get install -y nginx && apt-get cl   0.0s
 => CACHED [3/3] COPY index.html /var/www/html/index.html                         0.0s
 => exporting to image                                                            0.0s
 => => exporting layers                                                           0.0s
 => => writing image sha256:8c66d4d4b62dd12751fdbc1d4a08a8ccdb0ed196a7cf8f353e   0.0s
 => => naming to docker.io/library/webser                                         0.0s
master@ubuntu:~/docker$ docker run -d -p 8080:80 webser
5bd9e2ce834837c3c2d70c7995a92b92c001f16d1fa647c49a473bc82d41c2c3
master@ubuntu:~/docker$ 
```

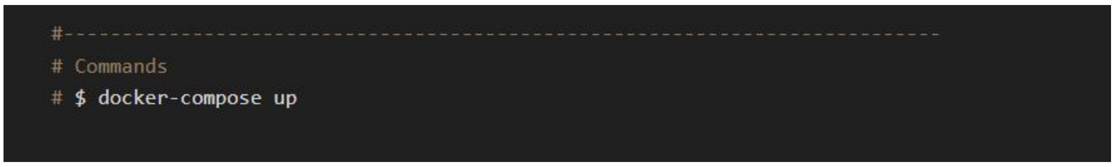


Welcome to DevOps World!

(b) Multi-Container Setup :



```yaml
# docker-compose.yml
version: "3.3"
# ------------------
services:
  web:                          # 1- web service Nginx
    build:
      context: .
      dockerfile: Dockerfile     # from previous point
    ports:
      - 8080:80
    deploy:
      resources:
        limits:         # Limits
          cpus: '1'
          memory: 512M
    networks:
      - my_network
    depends_on:
      - db

  db:                      # 2- database service Nginx
    image: postgres:12
    environment:
      POSTGRES_DB: myapp
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin
    volumes:
      - data:/vol
    networks:
      - my_network
# ------------------
volumes:
  data:
# ------------------
networks:
  my_network:
```

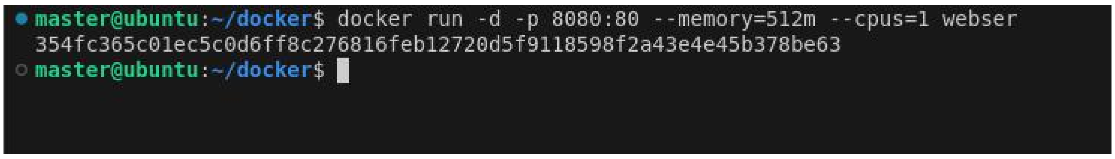


```
#---------------------------------------------------------------------------
# Commands
# $ docker-compose up
```

(c) Resource Limiting

Method One >> Create in docker-compose file “deploy”

Method Two >> Command line



```
● master@ubuntu:~/docker$ docker run -d -p 8080:80 --memory=512m --cpus=1 webser
  354fc365c01ec5c0d6ff8c276816feb12720d5f9118598f2a43e4e45b378be63
○ master@ubuntu:~/docker$ 
```