Name: Sownthari R P
Date: 07.08.2024

1.Mention the actions of following comments:

- **git remote add origin "http://github/a.git"**: This command adds a new remote repository with the name "origin" and the URL "http://github/a.git". The remote repository is typically where your code is stored, and "origin" is the default name for a remote.
- **git pull origin master**: This command fetches and merges changes from the "master" branch of the remote repository named "origin" into the current branch. It combines the git fetch and git merge commands.
- **git push origin dev**: This command pushes the commits from the local "dev" branch to the remote repository named "origin" and updates the "dev" branch there.

2. What are the functions of following Docker objects and key components:

- **Dockerd**: Dockerd is the Docker daemon, which runs on a host machine. It listens for Docker API requests and manages Docker objects like images, containers, networks, and volumes. It handles the creation, running, and stopping of containers.
- **Dockerfile**: A Dockerfile is a script containing a series of instructions on how to build a Docker image. It includes commands to set up the environment, copy files, install dependencies, and define default behaviors like running a specific command when a container starts.
- **docker-compose.yaml**: This file is used with Docker Compose to define and run multi-container Docker applications. It specifies the services, networks, and volumes required for the application, allowing you to manage the entire application stack with a single command (docker-compose up).
- **Docker Registries**: Docker registries are repositories where Docker images are stored. The most common registry is Docker Hub, but there are others like

Amazon ECR, Google Container Registry, and private registries. They allow you to share, store, and distribute Docker images.

- **DockerHost**: The DockerHost refers to the physical or virtual machine on which Docker is installed and running. It can be a local machine, a remote server, or a cloud-based environment where Docker manages containers.

3. What's the isolation in Docker container?

Isolation in Docker containers refers to the separation of applications and their dependencies into independent units. Each container runs in its own isolated environment with its own file system, network interfaces, and process space. Key aspects of Docker's isolation include:

- **Filesystem Isolation**: Each container has its own filesystem, provided by Docker images, ensuring that changes in one container do not affect others.
- **Process Isolation**: Containers run their own processes independently. The processes inside a container are isolated from those running on the host and in other containers.
- **Network Isolation**: Containers can have their own network interfaces and IP addresses, allowing for isolated network environments. Docker provides bridge networks, overlay networks, and other options for container networking.
- **Resource Limiting**: Docker can limit the amount of CPU, memory, and other resources that containers can use, preventing a single container from consuming all the host’s resources.

Docker Examples:

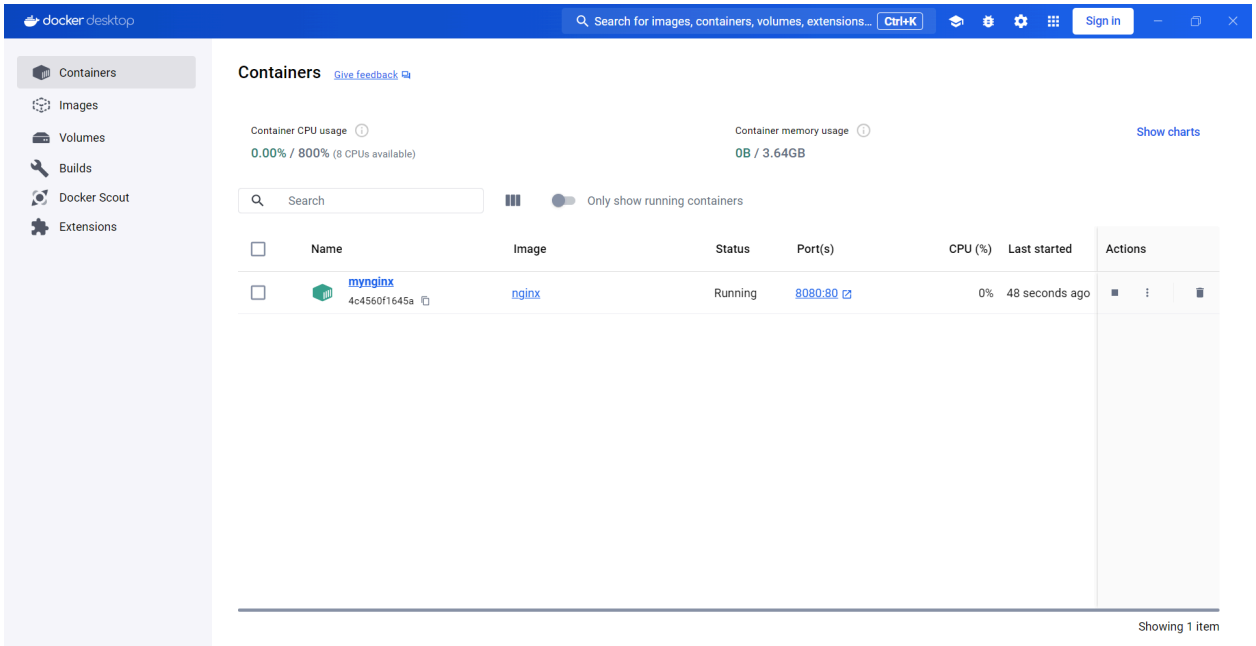
1.Pull image from Docker hub:

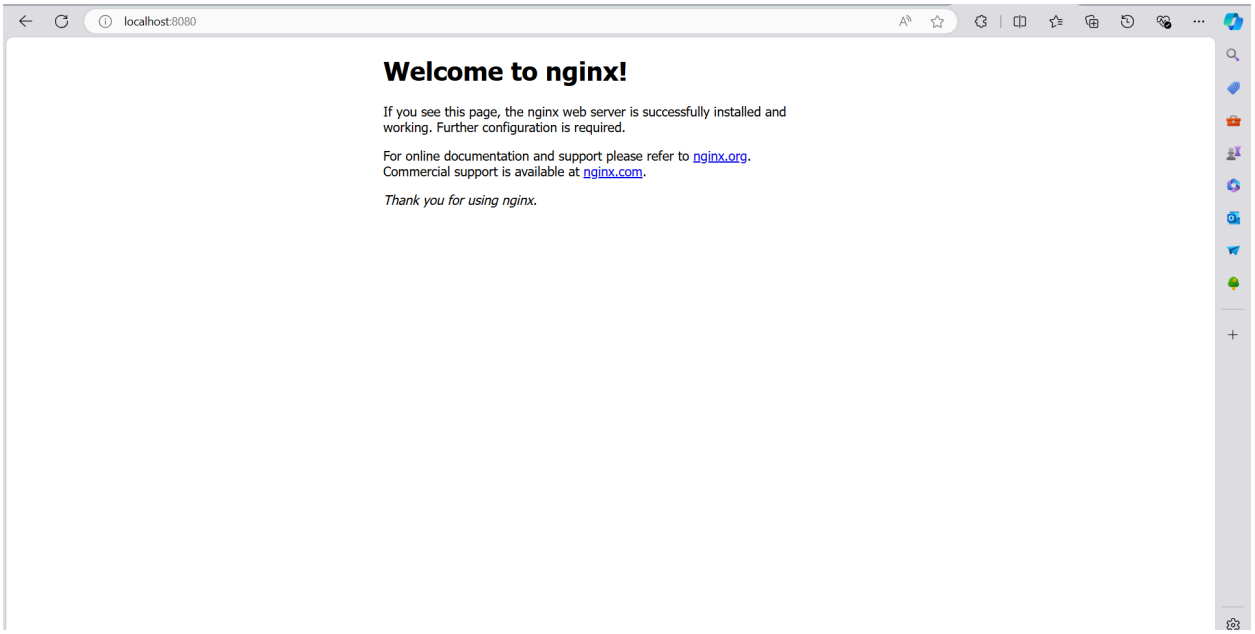
```
C:\Users\SOWNTHARI.RP>docker pull node:latest
latest: Pulling from library/node
ca4e5d672725: Pull complete
30b93c12a9c9: Pull complete
10d643a5fa82: Pull complete
d6dc1019d793: Pull complete
81bff076e6cf: Pull complete
1171ed9a56f6: Pull complete
fe9b706f3e3d: Pull complete
512b19417822: Pull complete
Digest: sha256:72314283e7a651d65a367f4e72fde18ec431a73ccfc87977f81be5dfc99c1c94
Status: Downloaded newer image for node:latest
docker.io/library/node:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview node:latest
```

2. Running a nginx Web Server in a Container

docker run -d -p 8080:80 --name mynginx nginx (or)

can be started directly from docker desktop.





3. Creating own image for python application

1. Create a Python script (app.py).
2. Create a Dockerfile to define the image.
3. Build the Docker image with docker build.
4. Run the Docker container with docker run.

1.app.py

```
def dog_facts():
    facts = [
        "Dogs have a sense of time and miss you when you're gone.",
        "Dogs' sense of smell is at least 40x better than humans'.",
        "Dogs can hear frequencies that are out of the range of human hearing.",
        "A dog’s nose print is unique, much like a person’s fingerprint.",
        "Some dogs are incredible swimmers."
    ]
```

```
    return facts
def print_dog_facts():
    facts = dog_facts()
    print("Here are some fun facts about dogs:")
    for fact in facts:
        print(f"- {fact}")
if __name__ == "__main__":
    print_dog_facts()
```

2. Dockerfile

```
# Official Python runtime as a parent image
FROM python:3.8-slim
# Set the working directory in the container
WORKDIR /usr/src/app
# Copy the current directory contents into the container at /usr/src/app
COPY . .
# Run the application
CMD ["python", "./app.py"]
```

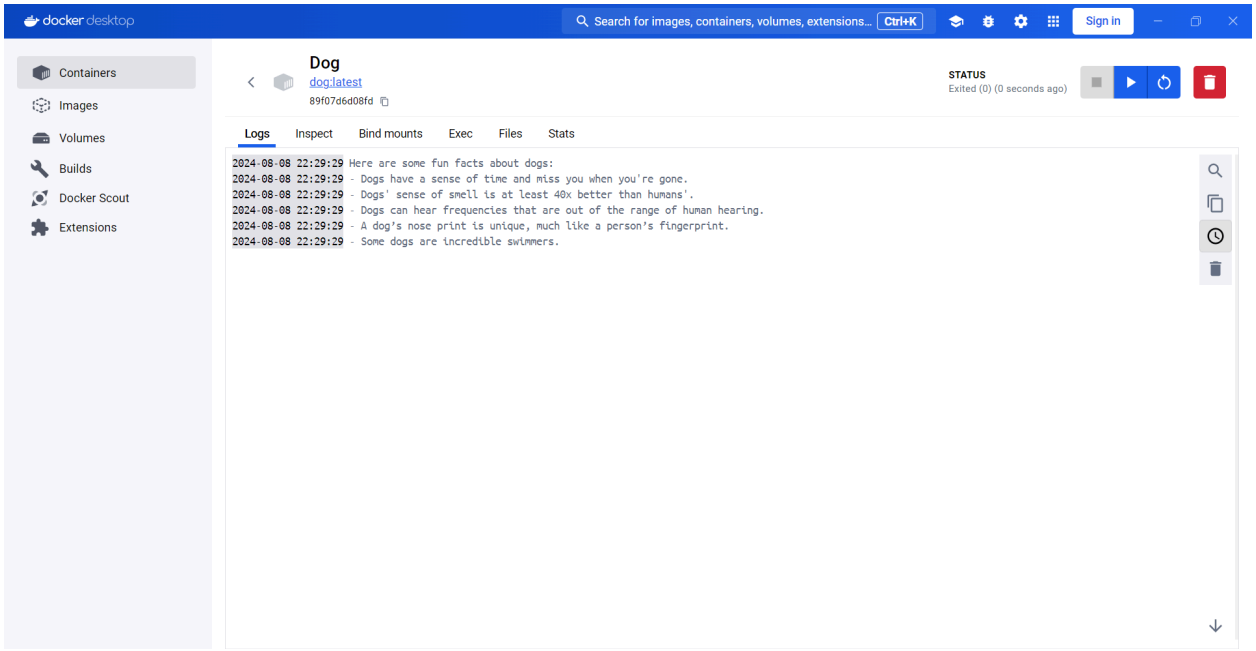
3.Build image for the application

docker build -t dog .

```
D:\Python\dog>docker build -t dog .
[+] Building 3.1s (8/8) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                0.1s
 => => transferring dockerfile: 313B                                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                                                  0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [1/3] FROM docker.io/library/python:3.8-slim                                                                    0.1s
 => [internal] load build context                                                                                   2.8s
 => => transferring context: 973B                                                                                   2.4s
 => [2/3] WORKDIR /usr/src/app                                                                                      0.0s
 => [3/3] COPY . .                                                                                                  0.1s
 => exporting to image                                                                                              0.1s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:975366b8d664e8b447642ae090b7365200042b05984fd9d6c09ef3cc844dee97                        0.0s
 => => naming to docker.io/library/dog                                                                              0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/jhjrjp6ufs27tetofdobrkbsw

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

4. Run dog image using command or docker desktop

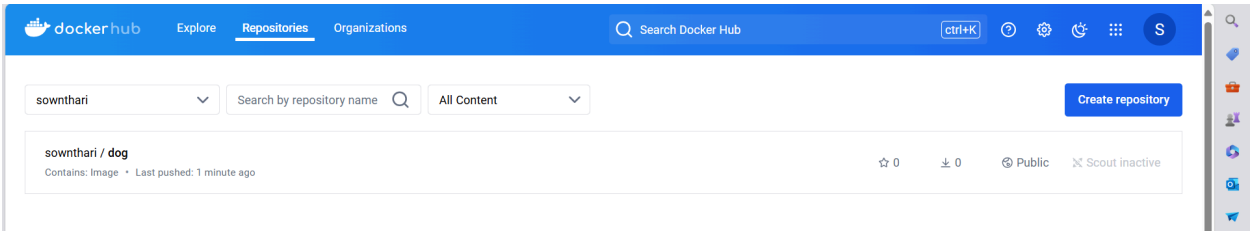
docker run dog



This image can be pushed to the docker hub.

5. Push the created image into docker hub.

```
D:\Python\dog>docker push sownthari/dog
Using default tag: latest
The push refers to repository [docker.io/sownthari/dog]
f11ed98ff595: Pushed
46178716d2b0: Pushed
0cfc6ead4554: Mounted from library/python
1109f5b27710: Mounted from library/python
cad3599a3016: Mounted from library/python
e7817ba7b646: Mounted from library/python
e0781bc8667f: Mounted from library/python
latest: digest: sha256:97bf10a8577b24477c36d27e915bab3bc6ae95eda2c43cbfa6140e9a1d366562 size: 1784
```



This image can be pulled from docker hub like all the other images.

4. Multi-container Docker application using docker compose with flask and redis images.

1. app.py: A simple Flask web application that interacts with Redis.
2. Dockerfile: Instructions to build the Python application image.
3. requirements.txt: Specifies Python dependencies.
4. docker-compose.yml: Defines the services, networks, and volumes.

1.app.py

```
import redis

from flask import Flask

app = Flask(__name__)

# Connect to Redis

r = redis.Redis(host='redis', port=6379)
```

```
@app.route('/')
def hello():
    # Increment the number of visits
    visits = r.incr('counter', 1)
    return f"Hello, World! This page has been visited {visits} times."
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

2.Dockerfile

```
# Official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the current directory contents into the container at /usr/src/app
COPY . .

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV FLASK_APP=app.py

# Run app.py when the container launches
CMD ["flask", "run", "--host=0.0.0.0"]
```

3.Requirements.txt

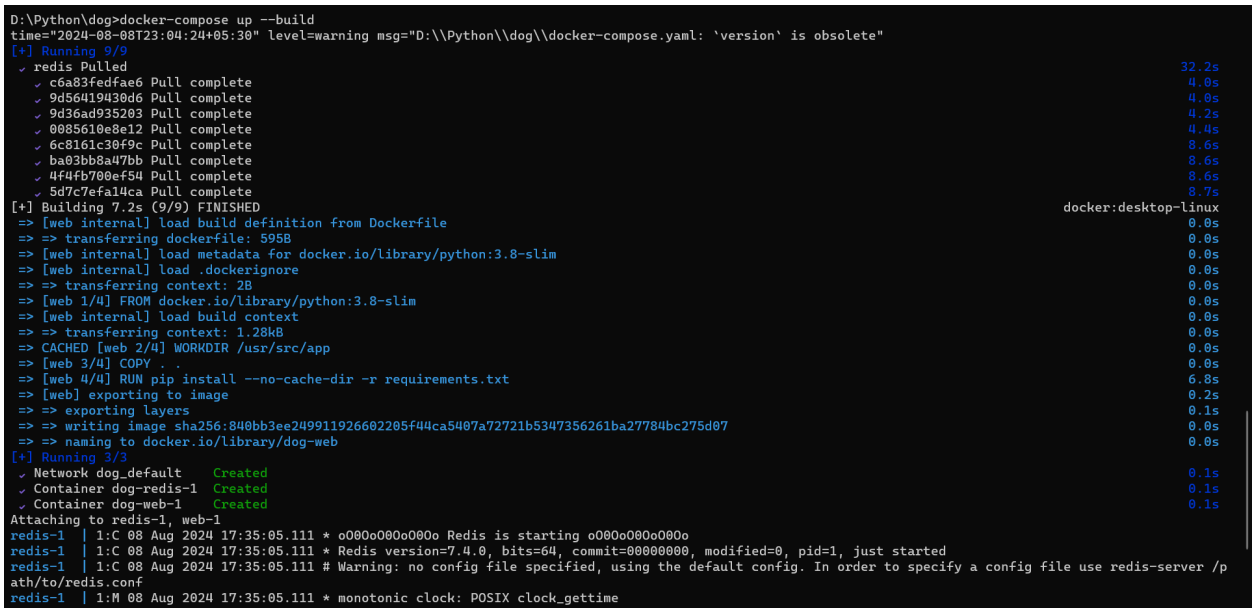
```
flask
redis
```

4.docker-compose.yaml

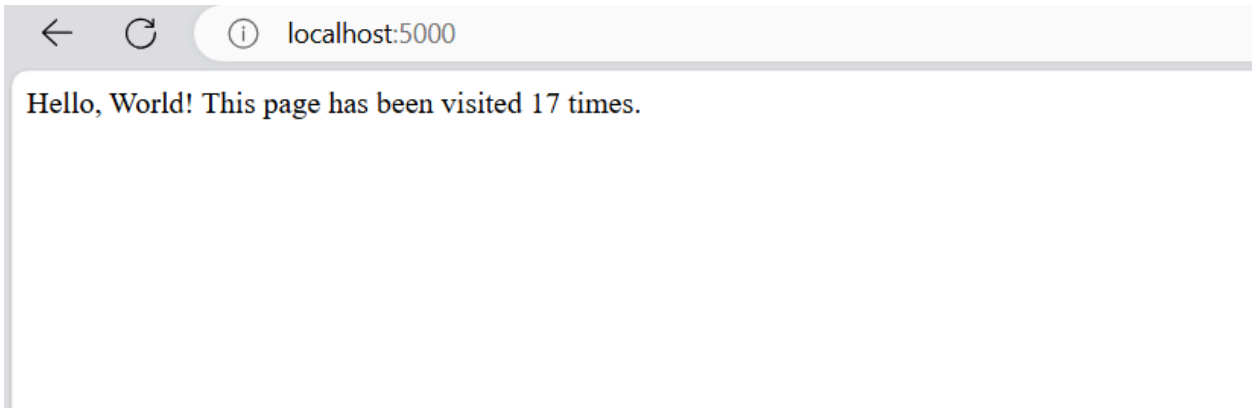
```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - redis

  redis:
    image: "redis:alpine"
```

Running the docker-compose

docker-compose up -- build



Output:


localhost:5000
Hello, World! This page has been visited 17 times.