Name: Siddarth S

Date: 07.08.2024

1.Mention the actions of following comments:

- **git remote add origin "http://github/siddarthh.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:

```
D:\Python\sidd>docker run --name sidd_container sidd
Hello, Sidd!

D:\Python\sidd>docker pull python
Using default tag: latest
latest: Pulling from library/python
ca4e5d672725: Already exists
30b93c12a9c9: Already exists
10d643a5fa82: Already exists
d6dc1019d793: Already exists
c3f22050b6d2: Pull complete
f326fa8e10c6: Pull complete
f4576d8c5700: Pull complete
cea52a973b36: Pull complete
Digest: sha256:c7862834f921957523cc4dab6d7795a7a0d19f1cd156c1ecd3a3a08c1108c9a4
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

2. 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

```python
# app.py

def hello_world():

    return "Hello, Sidd!"

if __name__ == "__main__":

    print(hello_world())
```

2. Dockerfile

```
# Use an 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"]
```

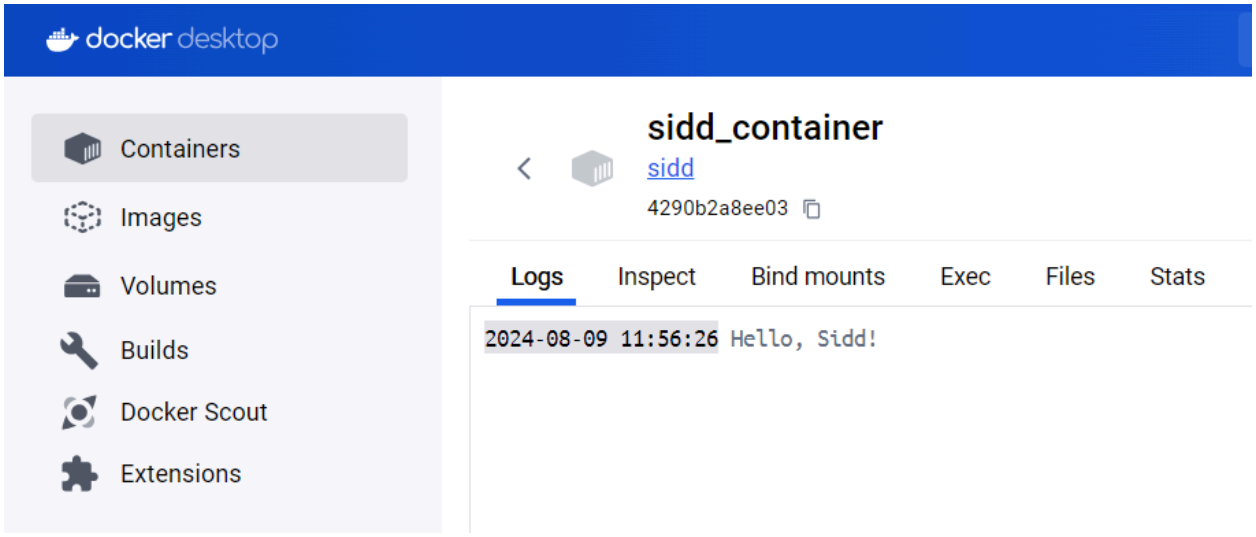
3.Build image for the application

docker build -t sidd .

```
D:\Python\sidd>docker build -t sidd .
[+] Building 0.1s (8/8) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                              0.0s
 => => transferring dockerfile: 322B                                                                                              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                                                                                  0.0s
 => [internal] load build context                                                                                                 0.0s
 => => transferring context: 57B                                                                                                  0.0s
 => CACHED [2/3] WORKDIR /usr/src/app                                                                                             0.0s
 => CACHED [3/3] COPY . .                                                                                                         0.0s
 => exporting to image                                                                                                            0.0s
 => => exporting layers                                                                                                           0.0s
 => => writing image sha256:567898659d21eb51e8afda10e8c9511cfae4c11dbee09fe1aede90db4b0b99b0                                      0.0s
 => => naming to docker.io/library/sidd                                                                                           0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/zf4k9d7v2d6mrujagpl6z4hin
```

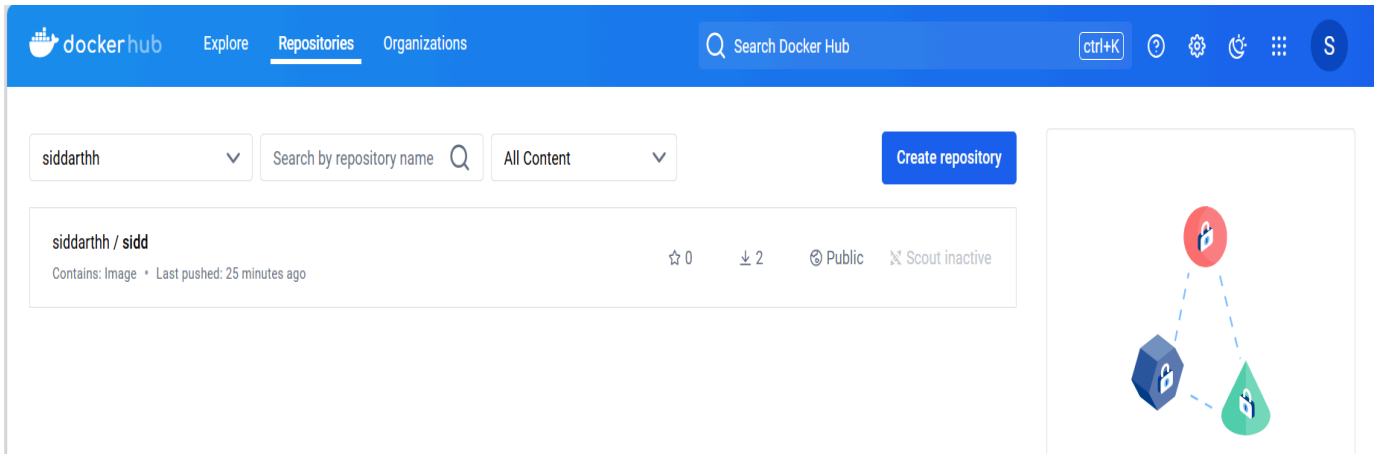
4. Run image using command or docker desktop



This image can be pushed to the docker hub.

5. Push the created image into docker hub.

```
D:\Python\sidd>docker push siddarthh/sidd
Using default tag: latest
The push refers to repository [docker.io/siddarthh/sidd]
baa3821afe63: Layer already exists
46178716d2b0: Layer already exists
0cfc6ead4554: Layer already exists
1109f5b27710: Layer already exists
cad3599a3016: Layer already exists
e7817ba7b646: Layer already exists
e0781bc8667f: Layer already exists
latest: digest: sha256:1f4d2181705e11aa903589ed6e2ef120f82cf87881b5c79a671d8fa56d708060 size: 1784
```



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