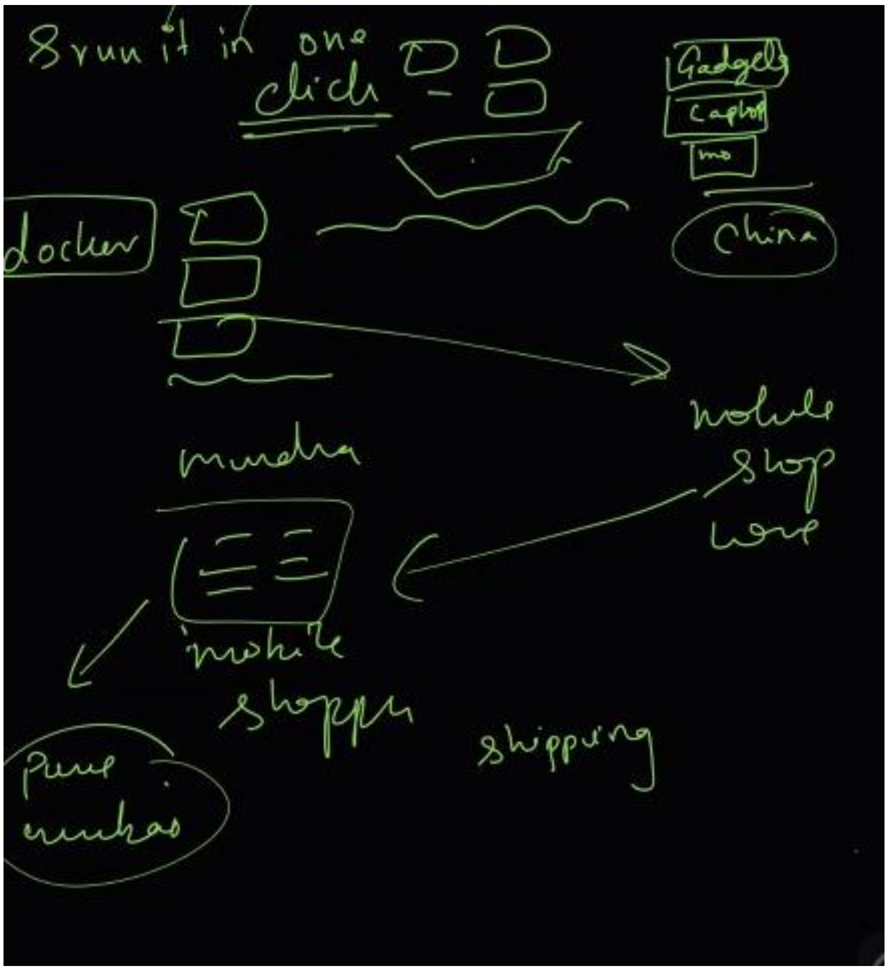
SRE day -11 (DOCKER):

This is done using concept of CI, known as continuos integration . when one can contribute to the other person's code using docker and it can be refered to shiping a container of software.

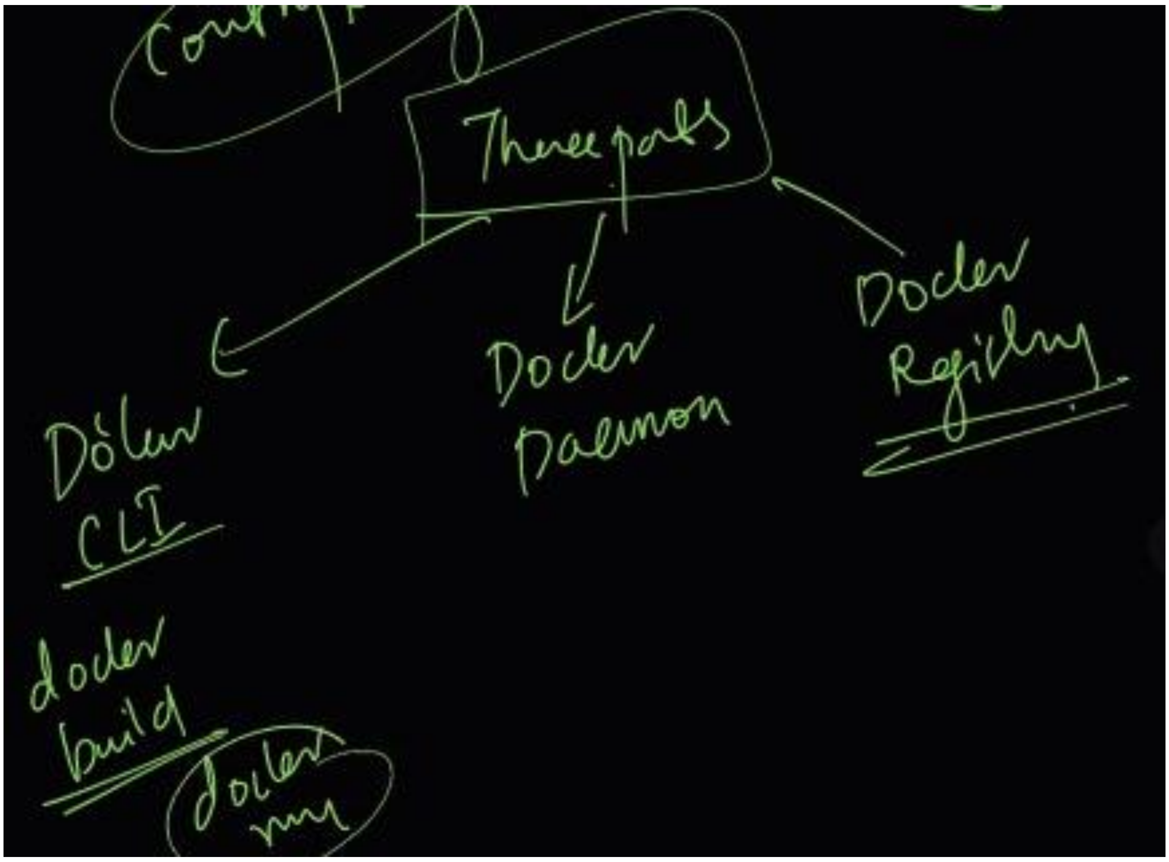


It creates a package of our application into a container, and each container is isolated and application stored in docker can run on any machine.

what is a docker, how it is useful ?

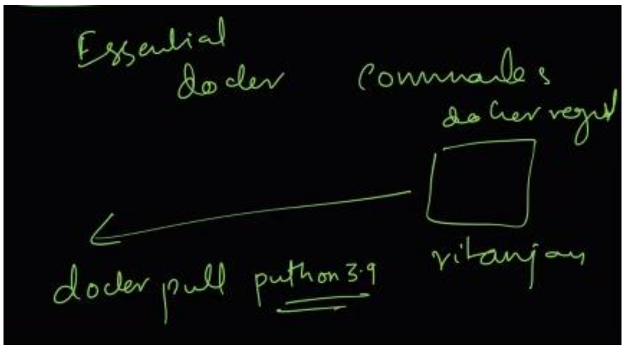
when we have a code and in addition a list of operations to perform, to make our code run (and it is known as image). and when we run it in docker , and it comes to life then it known as container and when it isn't implemented yet and is a theory, then it is known as image.

There are three main parts in docker architechture:



- **CLI:** it refers to accessing it from command prompt terminal from ubuntu wsl.
- **Dock daemon:** when we execute docker and it runs in background, then it is known as docker daemon.
- **Dock registery:** A store for docker images(or code theories) is known as docker registery. its like a GitHub for docker images, it contains all the theoretical codes of docker within it.

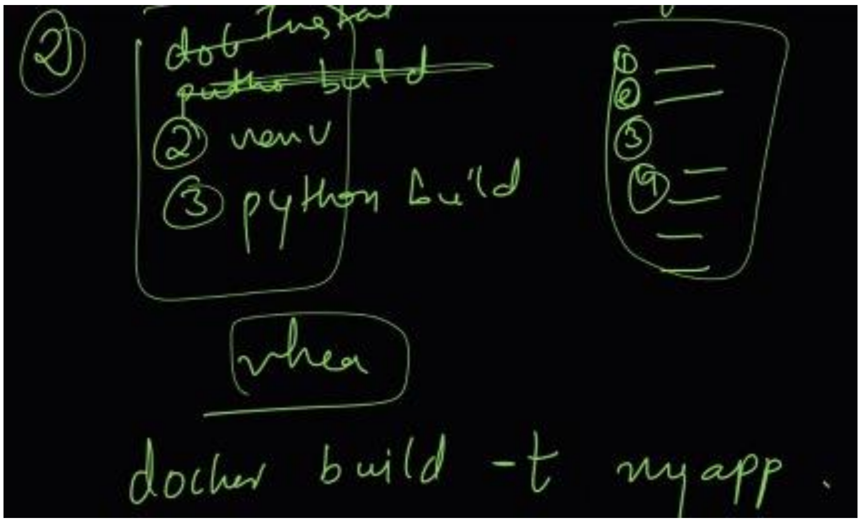
How to pull image from docker to our own device:



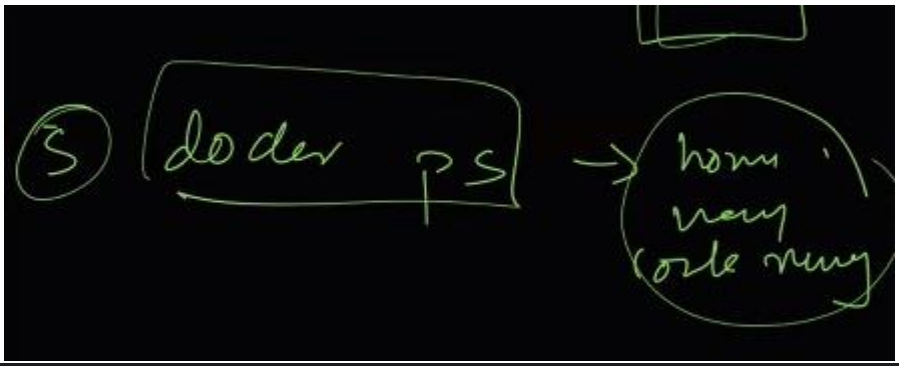
For this we use :
Docker pull command. and it brings the image from docker registery to that machine and it would do all the commands in the image one by one automatically.

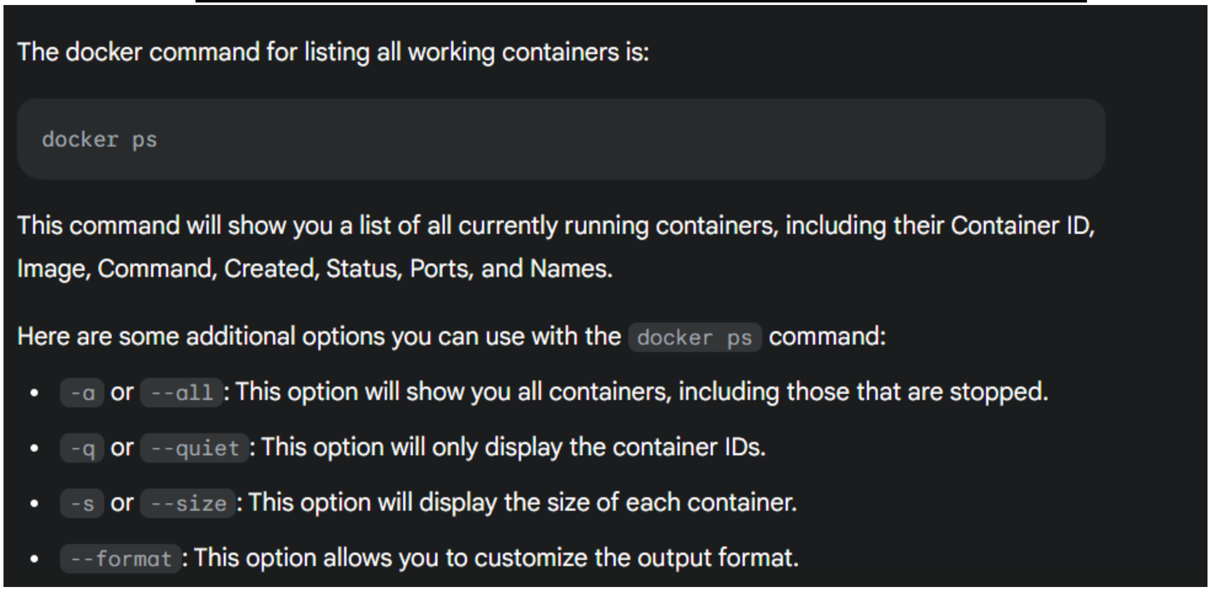
Now if to build a image into our registery, we use -t flag which stands for tag. And it is used to name the docker application:

Docker build -t <app name or image name>



Docker ps:




The docker command for listing all working containers is:

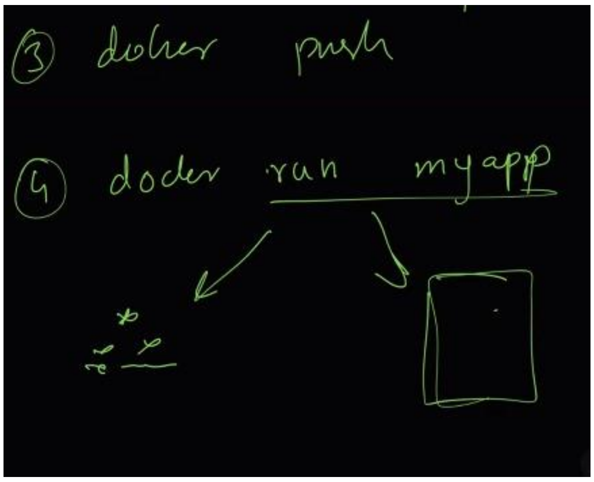
```
docker ps
```

This command will show you a list of all currently running containers, including their Container ID, Image, Command, Created, Status, Ports, and Names.

Here are some additional options you can use with the `docker ps` command:

- `-a` or `--all`: This option will show you all containers, including those that are stopped.
- `-q` or `--quiet`: This option will only display the container IDs.
- `-s` or `--size`: This option will display the size of each container.
- `--format`: This option allows you to customize the output format.

Docker push:



if we want to push the code into the registery or image. so basically when we push our docker file, it is converted into image and then it is pushed into docker. (basically we are packaging our application into a regitery, so that any person can run the registery and simply run our application).

(if we want to run a docker file, we use docker run <registery name>)

To install Docker:

# Update package list

```
sudo apt update
```

# Install prerequisites

```
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
```

# Add Docker's official GPG key

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```

# Add Docker repository

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list
```

# Update package list again

```
sudo apt update
```

# Install Docker

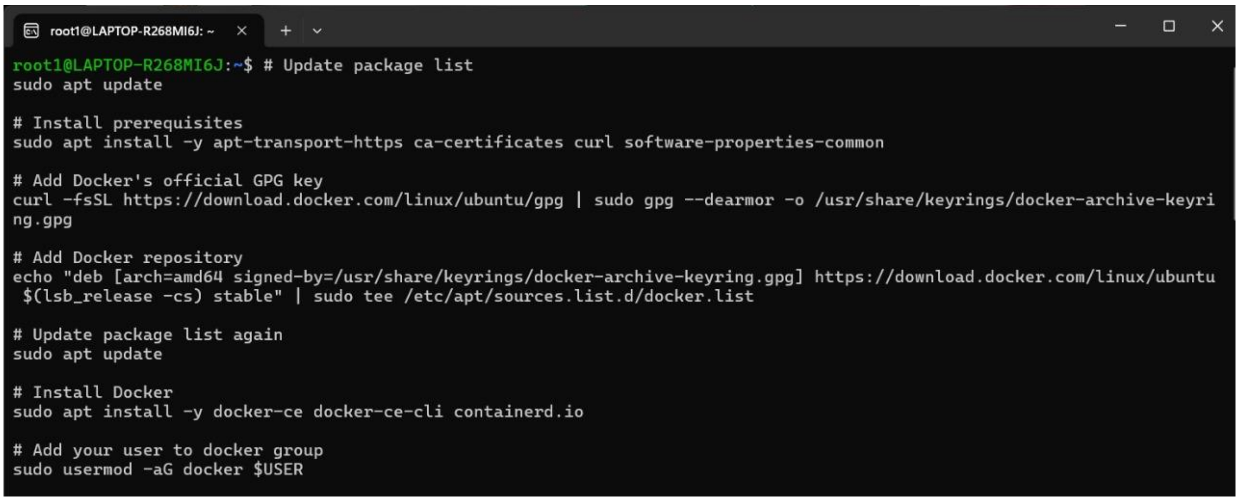
```
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

# Add your user to docker group

```
sudo usermod -aG docker $USER
```

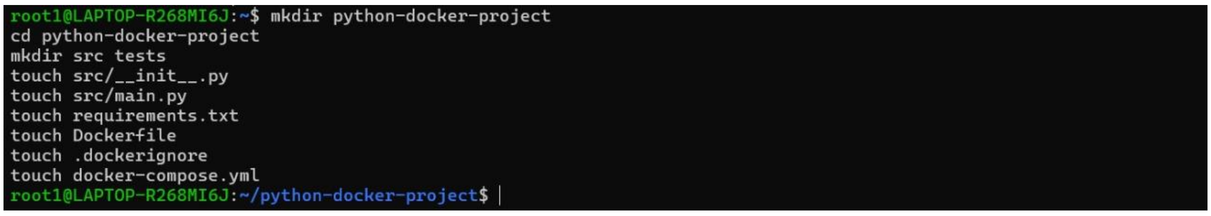
# Verify installation

```
docker --version
```

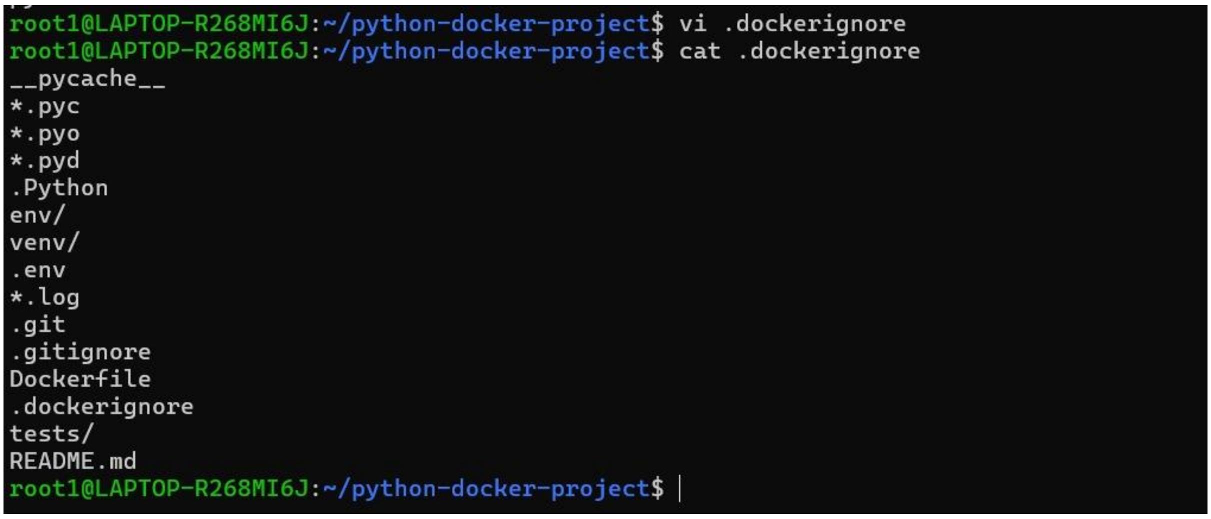


```
root1@LAPTOP-R268MI6J:~$ # Update package list
sudo apt update

# Install prerequisites
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyri
ng.gpg

# Add Docker repository
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu
 $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list

# Update package list again
sudo apt update

# Install Docker
sudo apt install -y docker-ce docker-ce-cli containerd.io

# Add your user to docker group
sudo usermod -aG docker $USER
```

And then run following command to create a specific:

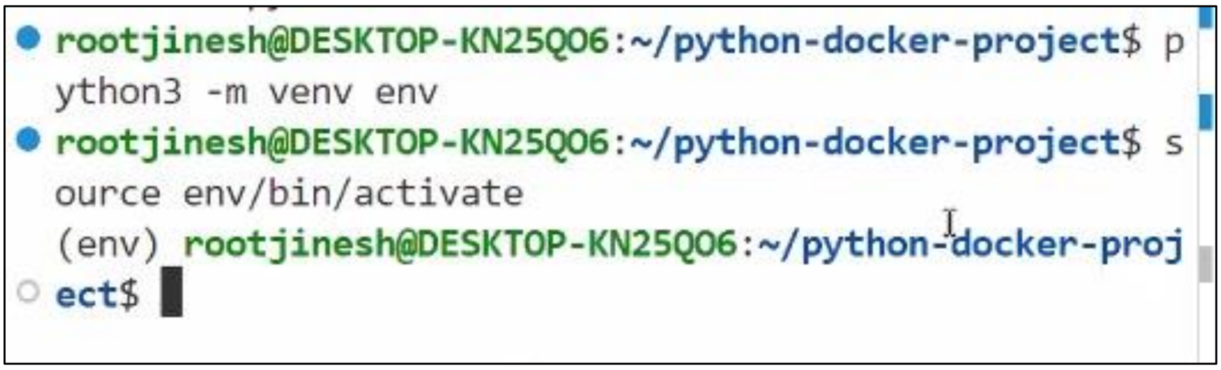


```
root1@LAPTOP-R268MI6J:~$ mkdir python-docker-project
cd python-docker-project
mkdir src tests
touch src/__init__.py
touch src/main.py
touch requirements.txt
touch Dockerfile
touch .dockerignore
touch docker-compose.yml
root1@LAPTOP-R268MI6J:~/python-docker-project$ |
```



```
root1@LAPTOP-R268MI6J:~/python-docker-project$ vi .dockerignore
root1@LAPTOP-R268MI6J:~/python-docker-project$ cat .dockerignore
__pycache__
*.pyc
*.pyo
*.pyd
.Python
env/
venv/
.env
*.log
.git
.gitignore
Dockerfile
.dockerignore
tests/
README.md
root1@LAPTOP-R268MI6J:~/python-docker-project$ |
```

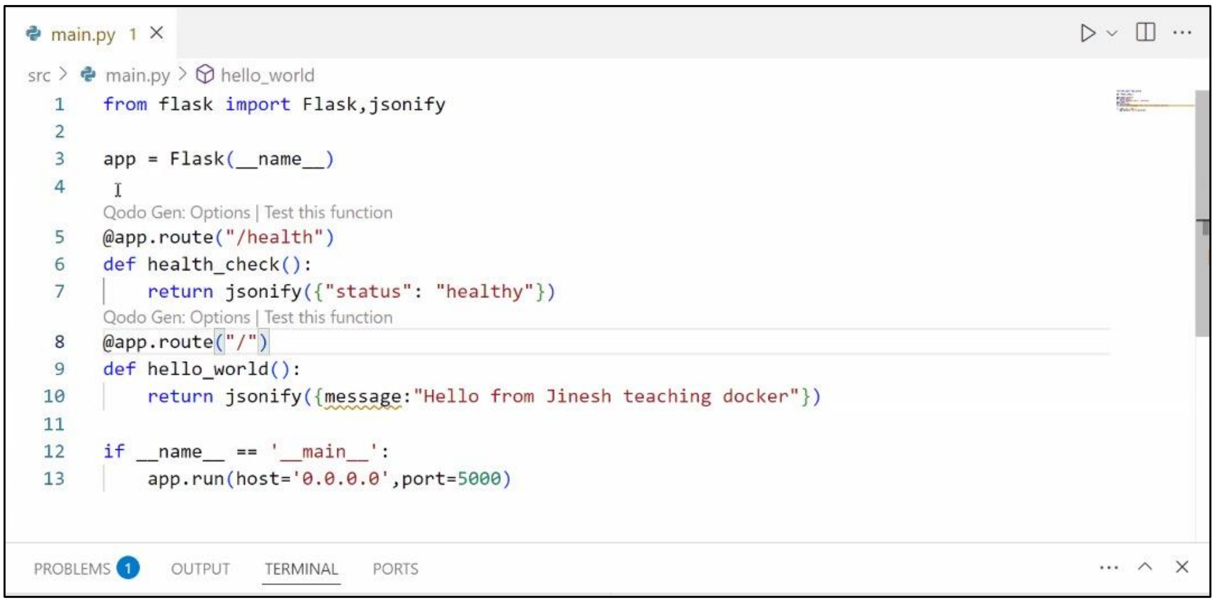
and then just simply write code " . " to open visual studio code for this, and then we create a virtual environment. and then we activate virtual environment.



```
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ p
ython3 -m venv env
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ s
ource env/bin/activate
(env) rootjinesh@DESKTOP-KN25Q06:~/python-docker-proj
ect$
```

and then we install pip install requirement.txt, **flask is used to create UI in python:**

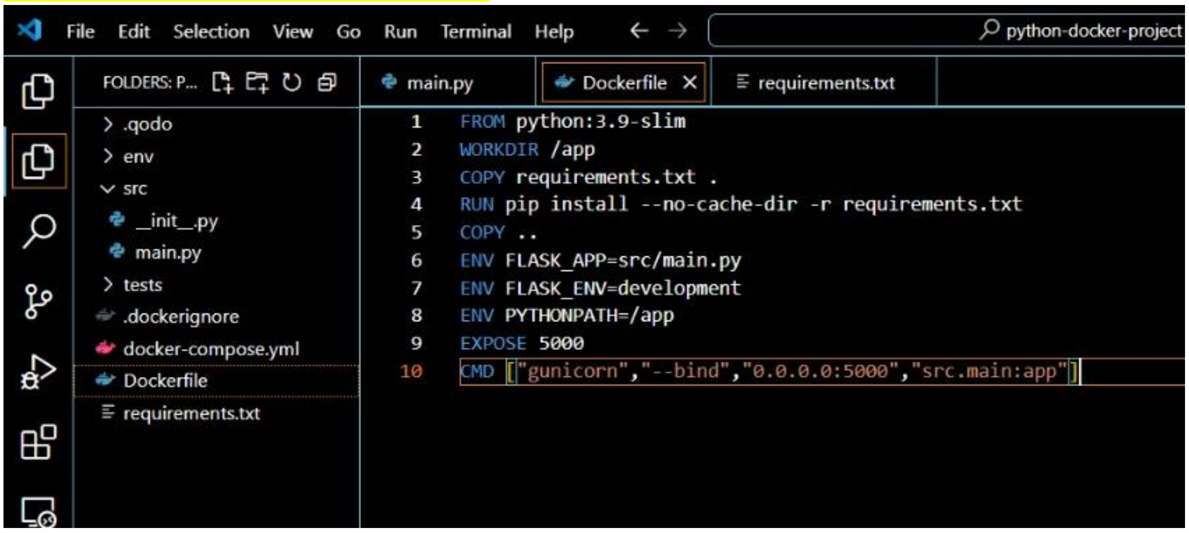
```
from flask import Flask,jsonify
app = Flask(_name_)
@app.route("/health")
def health_check():
   return jsonify({"status": "healthy"})
@app.route("/")
def hello_world():
   return jsonify({message:"Hello from Jinesh teaching docker"})
if _name_ == '_main_':
   app.run(host='0.0.0.0',port=5000)
```



```
from flask import Flask,jsonify

app = Flask(__name__)

@app.route("/health")
def health_check():
    return jsonify({"status": "healthy"})
@app.route("/")
def hello_world():
    return jsonify({message:"Hello from Jinesh teaching docker"})

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5000)
```

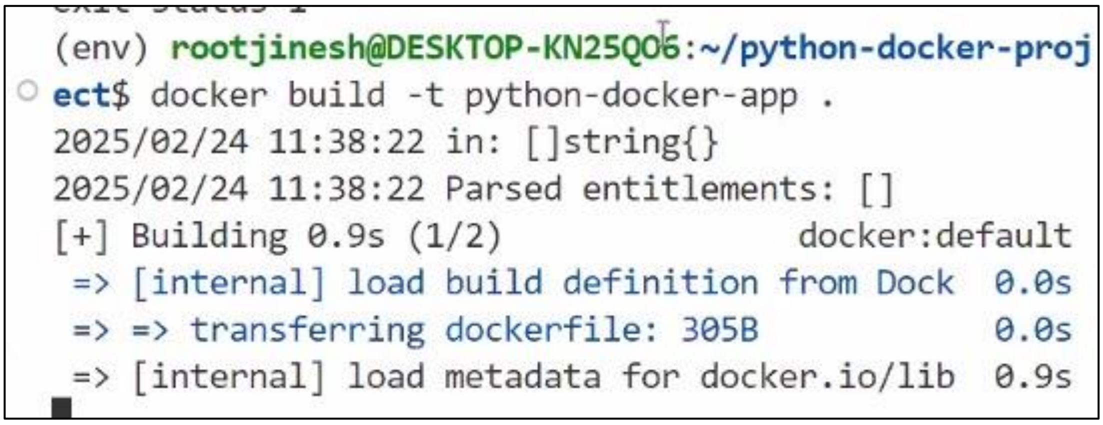
And then we simply run our file and our server would b live.

How to create a docker file:



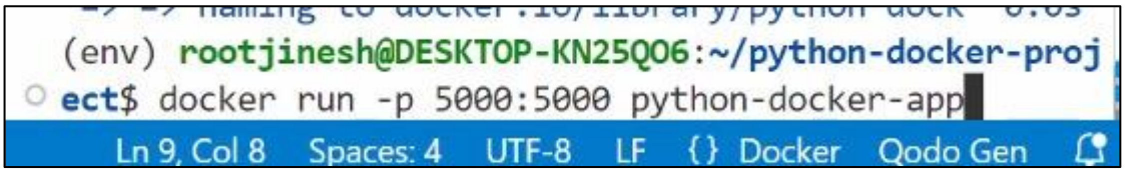
```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY ..
ENV FLASK_APP=src/main.py
ENV FLASK_ENV=development
ENV PYTHONPATH=/app
EXPOSE 5000
CMD ["gunicorn","--bind","0.0.0.0:5000","src.main:app"]
```

and then to run docker file, firstly we build it with tag flag for naming our app:



```
(env) rootjinesh@DESKTOP-KN25Q06:~/python-docker-proj
ect$ docker build -t python-docker-app .
2025/02/24 11:38:22 in: []string{}
2025/02/24 11:38:22 Parsed entitlements: []
[+] Building 0.9s (1/2)                    docker:default
 => [internal] load build definition from Dock   0.0s
 => => transferring dockerfile: 305B             0.0s
 => [internal] load metadata for docker.io/lib   0.9s
```

And then we write the run command:

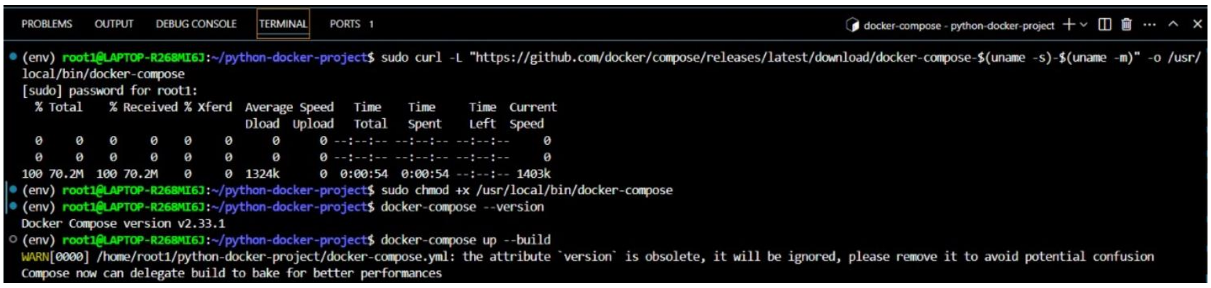


```
(env) rootjinesh@DESKTOP-KN25Q06:~/python-docker-proj
ect$ docker run -p 5000:5000 python-docker-app
```

Now say if we don't want to write two commands one for building and other one for running then we can simply write and use the "compose command".

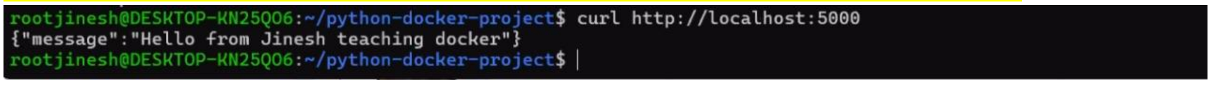
But for it we need to install compose and for that we write:

- sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
- sudo chmod +x /usr/local/bin/docker-compose
- docker-compose –version



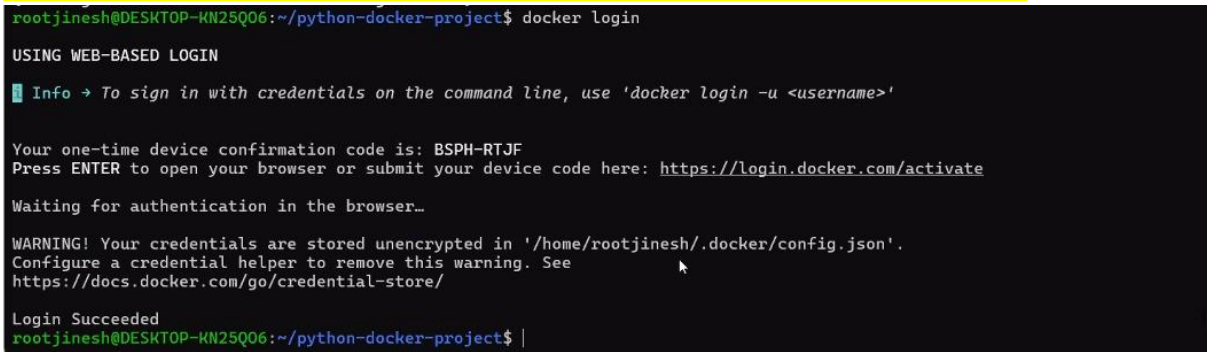
```
(env) root1@LAPTOP-R268MI6J:~/python-docker-project$ sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/
local/bin/docker-compose
[sudo] password for root1:
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 70.2M  100 70.2M    0     0  1324k      0  0:00:54  0:00:54 --:--:-- 1403k
(env) root1@LAPTOP-R268MI6J:~/python-docker-project$ sudo chmod +x /usr/local/bin/docker-compose
(env) root1@LAPTOP-R268MI6J:~/python-docker-project$ docker-compose --version
Docker Compose version v2.33.1
(env) root1@LAPTOP-R268MI6J:~/python-docker-project$ docker-compose up --build
WARN[0000] /home/root1/python-docker-project/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose now can delegate build to bake for better performances
```

To run from local host we can also simply paste the route:



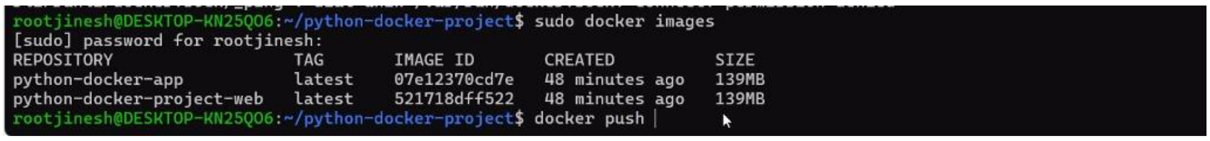
```
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ curl http://localhost:5000
{"message":"Hello from Jinesh teaching docker"}
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ |
```

Now to use docker from ubuntu wsl terminal, we need to do login from docker page and then perform following commands:



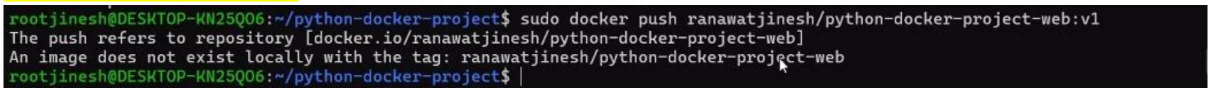
```
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ docker login

USING WEB-BASED LOGIN

i Info → To sign in with credentials on the command line, use 'docker login -u <username>'


Your one-time device confirmation code is: BSPH-RTJF
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser…

WARNING! Your credentials are stored unencrypted in '/home/rootjinesh/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ |
```

To verify if connection working or not, we can see if our previously created files are there or not:



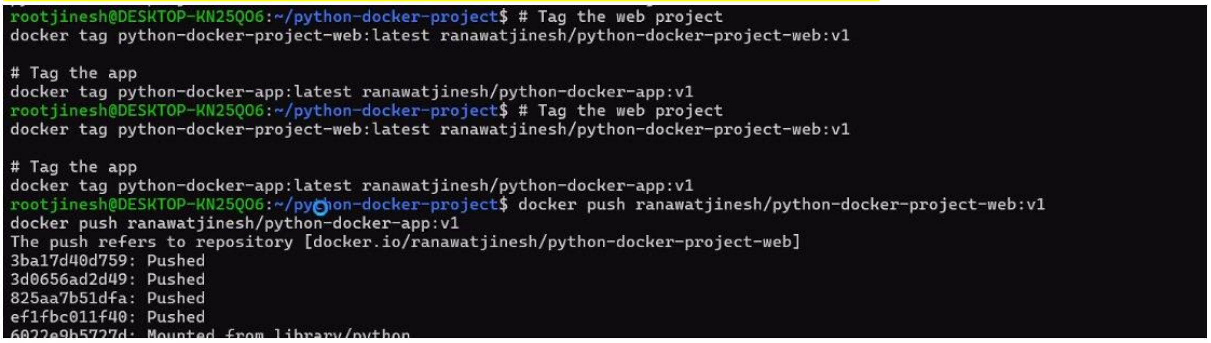
```
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ sudo docker images
[sudo] password for rootjinesh:
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
python-docker-app           latest    07e12370cd7e   48 minutes ago   139MB
python-docker-project-web   latest    521718dff522   48 minutes ago   139MB
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ docker push
```

To push our app:



```
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ sudo docker push ranawatjinesh/python-docker-project-web:v1
The push refers to repository [docker.io/ranawatjinesh/python-docker-project-web]
An image does not exist locally with the tag: ranawatjinesh/python-docker-project-web
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$
```

And to push "image: we need to perform tagging:



```
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ # Tag the web project
docker tag python-docker-project-web:latest ranawatjinesh/python-docker-project-web:v1

# Tag the app
docker tag python-docker-app:latest ranawatjinesh/python-docker-app:v1
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ # Tag the web project
docker tag python-docker-project-web:latest ranawatjinesh/python-docker-project-web:v1

# Tag the app
docker tag python-docker-app:latest ranawatjinesh/python-docker-app:v1
rootjinesh@DESKTOP-KN25Q06:~/python-docker-project$ docker push ranawatjinesh/python-docker-project-web:v1
docker push ranawatjinesh/python-docker-app:v1
The push refers to repository [docker.io/ranawatjinesh/python-docker-project-web]
3ba17d40d759: Pushed
3d0656ad2d49: Pushed
825aa7b51dfa: Pushed
ef1fbc011f40: Pushed
6022e9b5727d: Mounted from library/python
```

COMMAND FOR RUNNING DOCKER FROM WSL:

#tag web project

sudo docker tag python-docker-project-web:latest Ritanjay/python-docker-project-web:v1

# Tag the app

sudo docker tag python-docker-app:latest Ritanjay/python-docker-app:v1

# Tag the web project

sudo docker tag python-docker-project-web:latest Ritanjay /python-docker-project-web:v1

# Tag the app

sudo docker tag python-docker-app:latest Ritanjay/python-docker-app:v1

sudo docker push Ritanjay/python-docker-project-web:v1

sudo docker push Ritanjay/python-docker-app:v1