Setup docker in ubuntu: end to end docker process:

I have created one aws ec2 instance with instance type c7iflex.large --> availability zone -> eu-north-1b.

And connected to above server by git bash using below commands

ssh -i /c/Users/kasev/Downloads/aws_key_manju_mail.pem.pem ubuntu@16.171.36.240

Step 1: Install Docker on Ubuntu

Run these commands one by one in your EC2 terminal 👇

```
# Update your packages
sudo apt update -y

# Install Docker engine
sudo apt install docker.io -y

# Enable and start Docker service
sudo systemctl enable docker
sudo systemctl start docker

# Check Docker version
docker --version
```

✅ You should see output like:

```
Docker version 24.x.x, build ...
```

🧑‍💻 Step 2: Add your user to the Docker group

This lets you run `docker` commands without using `sudo` every time.

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

Then **logout and login again**, or run:

```
newgrp docker
```

To verify:

```
docker ps
```

If it shows no errors → good to go ✅

📁 Step 3: Create the Project

Create a simple directory for our app:

```
mkdir docker-flask-app
cd docker-flask-app
```

🧩 Step 4: Create Flask App Files

1️⃣ Create `app.py`

```
nano app.py
```

Paste this code:

```
from flask import Flask
```

```
app = Flask(name)

@app.route('/')

def hello():

    return "Hello Everyone, Thanks for watching this video!! please
like this video and subscribe my channel!!"

if __name__ == "__main__":

   app.run(host='0.0.0.0', port=5000)
```

:wq!

2️⃣ Create `requirements.txt`

```
flask
```

3️⃣ Create `Dockerfile`

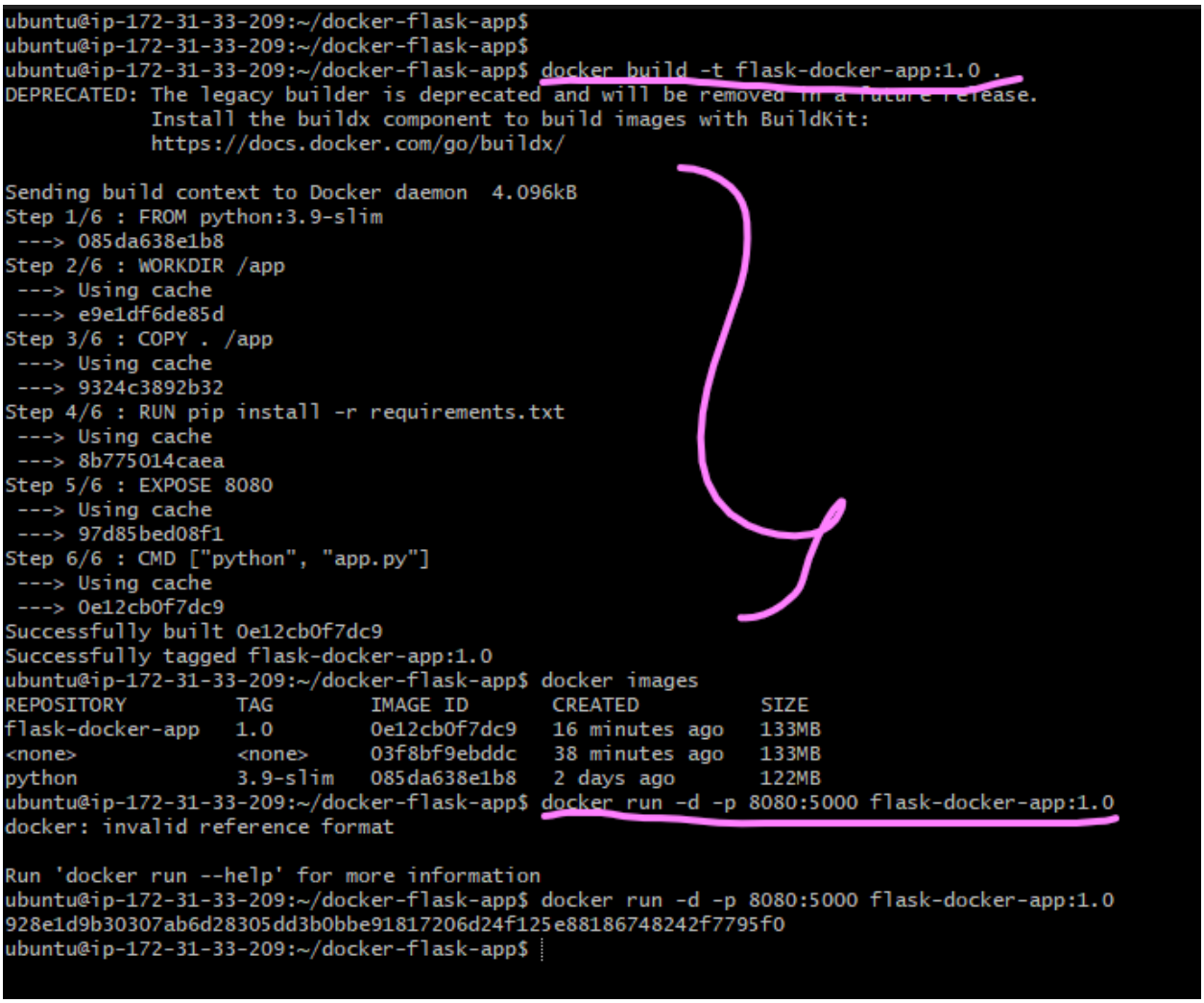
```
nano Dockerfile
```

Paste this:

```
FROM python:3.9-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 8080
CMD ["python", "app.py"]
```

Save and exit.

Step 5: Build Docker Image



`docker build -t flask-docker-app:1.0 .`

Step 6: Verify the Image

`docker images`

You should see `flask-docker-app:1.0` in the list.

☑ What Happened

- Docker pulled the **Python 3.9 base image.**

- Installed **Flask** and its dependencies (click, werkzeug, jinja2, etc.) inside the image.
- Successfully completed the **RUN pip install -r requirements.txt** step.
- Finally executed the CMD ["python", "app.py"] instruction — meaning when you run the container, it'll start your Flask app automatically.
- The image flask-docker-app:1.0 is now ready.

The **warning about "running pip as root"** is harmless inside containers — you can safely ignore it since containers are isolated environments.

Step 7: Run Container

docker run –d -p 8080:5000 flask-docker-app:1.0 -- to create and start the new container from the flask-docker-app image

or

docker run –p hostpoart:containrport <imagename>

Now your container is running!

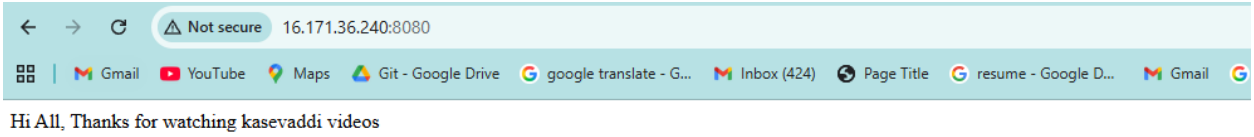
Check with:

docker ps

Step 7: Test Your App

Go to your AWS instance's **Public IP**, open in browser:

http://<your-public-ip>:8080



You should see:

Hello, Docker! 🚀 Flask app running inside AWS EC2 container.

the above content we can edit in app.py file.

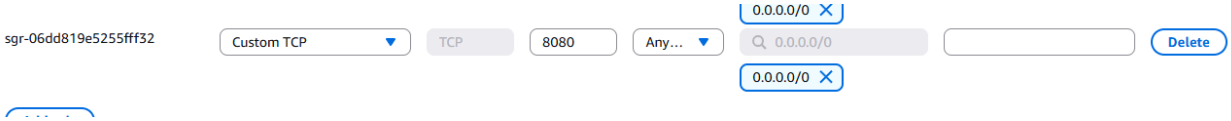
⚠️ If You Don’t See It:

Make sure your **EC2 Security Group** allows **Inbound Rule: Port 8080 (TCP).**

To fix:

- Go to **AWS Console → EC2 → Security Groups → Inbound Rules → Edit**
- Add:
  - Type: Custom TCP
  - Port Range: 8080
  - Source: 0.0.0.0/0

Then reload your browser.



✅ **Congratulations!**
You just completed:

Docker Build → Run → Deploy → Access via Public IP on AWS EC2

NOTE: IF YOU MAKE ANY CHANGES IN APP.PY WE SHOULD RUN BELOW COMMANDS

2. Rebuild your Docker image (so the fixed app.py is used):

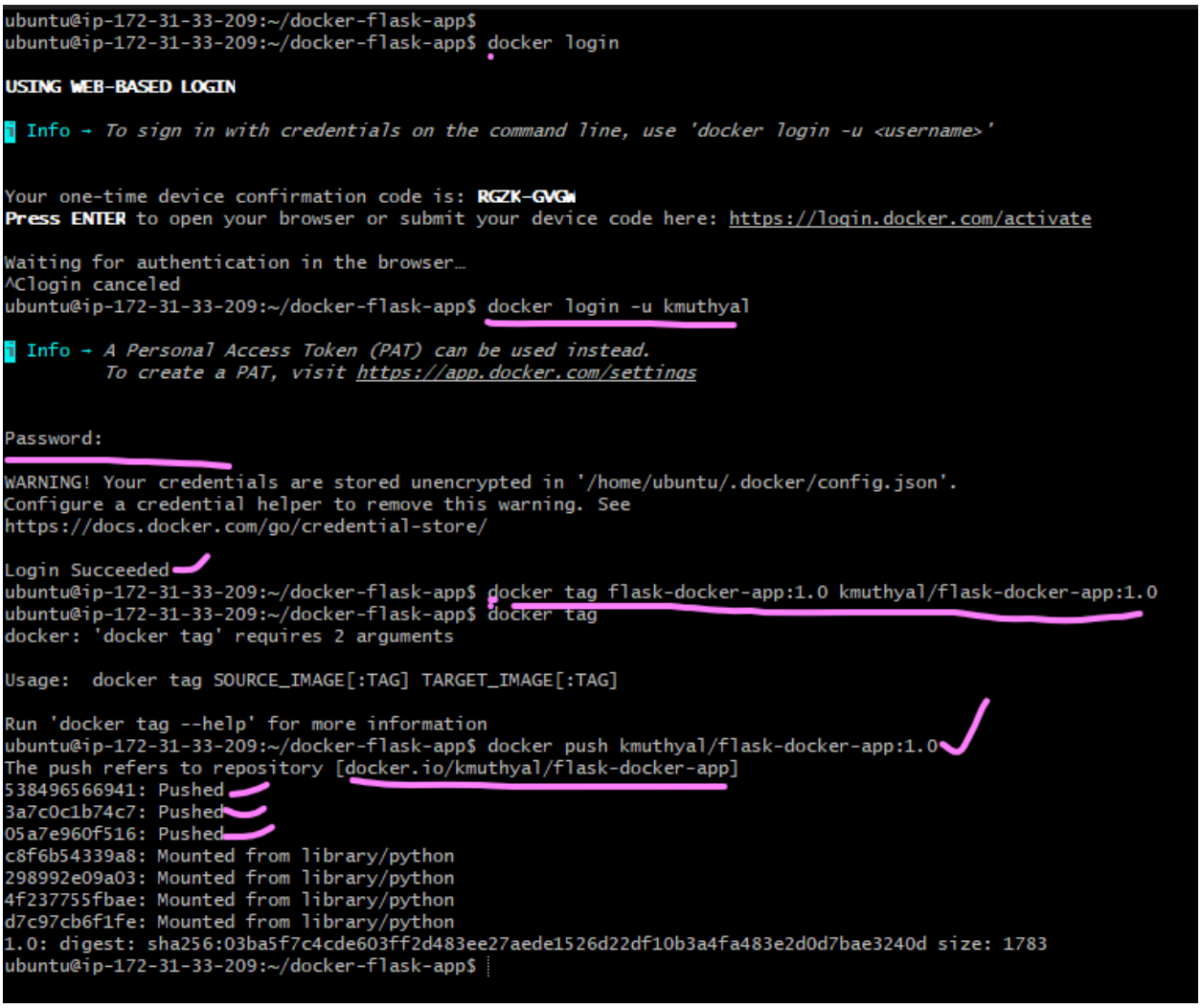
```
docker build -t flask-docker-app:1.0 .
```

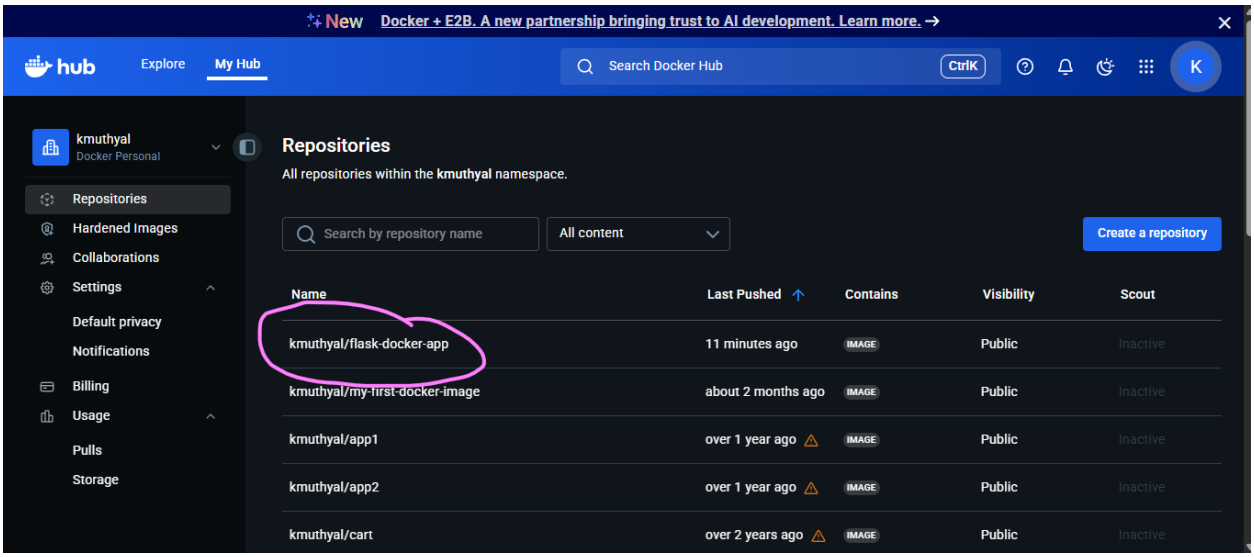
3. Run the container again:

docker run –d -p 8080:5000 flask-docker-app:1.0 -- before run this command pls stop existed containers or change port numbers on above command.
Host server port no is 8080, container port no is 5000
8080:500

Finally, Flask app is now **successfully deployed and running** inside the Docker container, and you can access it through the browser (e.g., http://<your-EC2-public-IP>:8080).

Step 8: push the image to docker hub (registry)





Topic: how to pull and use the existed image form docker-hub

Perfect 👏 — since you've already pushed your Docker image to **Docker Hub**, pulling and running it on a **new Ubuntu server** is very straightforward. Let's go step-by-step.

🧭 Step-by-step: Pull and run your image on a new server

☑️ Step 1 — Install Docker (if not installed yet)

If your new Ubuntu server doesn't have Docker installed, run:

```
sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
```

You can verify Docker is running:

```
docker –version
```

Step : Add your user to the Docker group

This lets you run `docker` commands without using `sudo` every time.

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

Then **logout and login again**, or run:

```
newgrp docker
```

☑ Step 2 — Log in to Docker Hub

Use your Docker Hub credentials to log in so the new server can pull private or public images:

```
docker login –u kmuthyal (username of dockerhub)
Passwd: password of above user
```

It will prompt for your **Docker Hub username** and **password (or access token)**.

☑ Step 3 — Pull your image

Now, pull your image from Docker Hub:

```
docker pull <your-dockerhub-username>/<image-name>:<tag>
```

Example:

```
docker pull kmuthyal/flask-docker-app:1.0
```

This downloads the image layers to your new server.

You can verify it's there:

```
docker images
```

☑ Step 4 — Run the container

Now run the Flask app container and expose it on your desired port:

```
docker run -d -p 8080:5000 --name flask-app kmuthyal/flask-docker-app:1.0
```

Explanation:

- `-d` → run in detached mode
- `-p 8080:5000` → map host port 8080 to container's internal port 5000 (Flask default)
- `--name flask-app` → container name
- `kmuthyal/flask-docker-app:1.0` → image from Docker Hub

☑ Step 5 — Verify it's running

```
docker ps
```

You should see something like:

```
CONTAINER ID   IMAGE                          COMMAND           STATUS
PORTS                    NAMES
```

```
abcd1234efgh   kmuthyal/flask-docker-app:1.0   "python app.py"   Up 5
seconds    0.0.0.0:8080->5000/tcp   flask-app

Step: please enable inbound rules (tcp and 8080 port) in aws server.
Before access in browser
```

☑ Step 6 — Access in browser

Now open your Flask app in the browser using your new server's **public IP**:

http://<your-server-public-ip>:8080

You should see your Flask app page (e.g., "Hello World!").