

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

1 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!
```

2 Create requirements.txt

flask

3 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

```
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker build -t flask-docker-app:1.0 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/
Sending build context to Docker daemon 4.096kB
Step 1/6 : FROM python:3.9-slim
--> 085da638e1b8
Step 2/6 : WORKDIR /app
--> Using cache
--> e9e1df6de85d
Step 3/6 : COPY . /app
--> Using cache
--> 9324c3892b32
Step 4/6 : RUN pip install -r requirements.txt
--> Using cache
--> 8b775014caea
Step 5/6 : EXPOSE 8080
--> Using cache
--> 97d85bed08f1
Step 6/6 : CMD ["python", "app.py"]
--> Using cache
--> 0e12cb0f7dc9
Successfully built 0e12cb0f7dc9
Successfully tagged flask-docker-app:1.0
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
flask-docker-app  1.0          0e12cb0f7dc9  16 minutes ago  133MB
<none>          <none>       03f8bf9ebddc  38 minutes ago  133MB
python           3.9-slim    085da638e1b8   2 days ago    122MB
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker run -d -p 8080:5000 flask-docker-app:1.0
docker: invalid reference format
Run 'docker run --help' for more information
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker run -d -p 8080:5000 flask-docker-app:1.0
928e1d9b30307ab6d28305dd3b0bbe91817206d24f125e88186748242f7795f0
ubuntu@ip-172-31-33-209:~/docker-flask-app$ |
```

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 hostport:containerport <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>



Hi All, Thanks for watching kasevaddi videos

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.



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)

```
ubuntu@ip-172-31-33-209:~/docker-flask-app$  
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker login  
  
USING WEB-BASED LOGIN  
  
Info - To sign in with credentials on the command line, use 'docker login -u <username>'  
  
Your one-time device confirmation code is: RGZK-GVGM  
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate  
  
Waiting for authentication in the browser...  
^Clogin canceled  
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker login -u kmuthyal  
  
Info - A Personal Access Token (PAT) can be used instead.  
To create a PAT, visit https://app.docker.com/settings  
  
Password:  
  
WARNING! Your credentials are stored unencrypted in '/home/ubuntu/.docker/config.json'.  
Configure a credential helper to remove this warning. See  
https://docs.docker.com/go/credential-store/  
  
Login Succeeded  
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker tag flask-docker-app:1.0 kmuthyal/flask-docker-app:1.0  
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker tag  
docker: 'docker tag' requires 2 arguments  
  
Usage: docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]  
  
Run 'docker tag --help' for more information  
ubuntu@ip-172-31-33-209:~/docker-flask-app$ docker push kmuthyal/flask-docker-app:1.0  
The push refers to repository [docker.io/kmuthyal/flask-docker-app]  
538496566941: Pushed  
3a7c0c1b74c7: Pushed  
05a7e960f516: Pushed  
c8f6b54339a8: Mounted from library/python  
298992e09a03: Mounted from library/python  
4f237755fbae: Mounted from library/python  
d7c97cb6f1fe: Mounted from library/python  
1.0: digest: sha256:03ba5f7c4cde603ff2d483ee27aede1526d22df10b3a4fa483e2d0d7bae3240d size: 1783  
ubuntu@ip-172-31-33-209:~/docker-flask-app$
```

The screenshot shows the Docker Hub 'My Hub' interface. On the left, there's a sidebar with options like 'Repositories', 'Hardened Images', 'Collaborations', 'Settings', 'Default privacy', 'Notifications', 'Billing', 'Usage', 'Pulls', and 'Storage'. The main area is titled 'Repositories' and shows a list of repositories under the 'kmuthyal' namespace. The list includes:

Name	Last Pushed	Contains	Visibility	Scout
kmuthyal/flask-docker-app	11 minutes ago	IMAGE	Public	Inactive
kmuthyal/my-first-docker-image	about 2 months ago	IMAGE	Public	Inactive
kmuthyal/app1	over 1 year ago	IMAGE	Public	Inactive
kmuthyal/app2	over 1 year ago	IMAGE	Public	Inactive
kmuthyal/cart	over 2 years ago	IMAGE	Public	Inactive

A pink oval highlights the first repository, 'kmuthyal/flask-docker-app'.

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!").