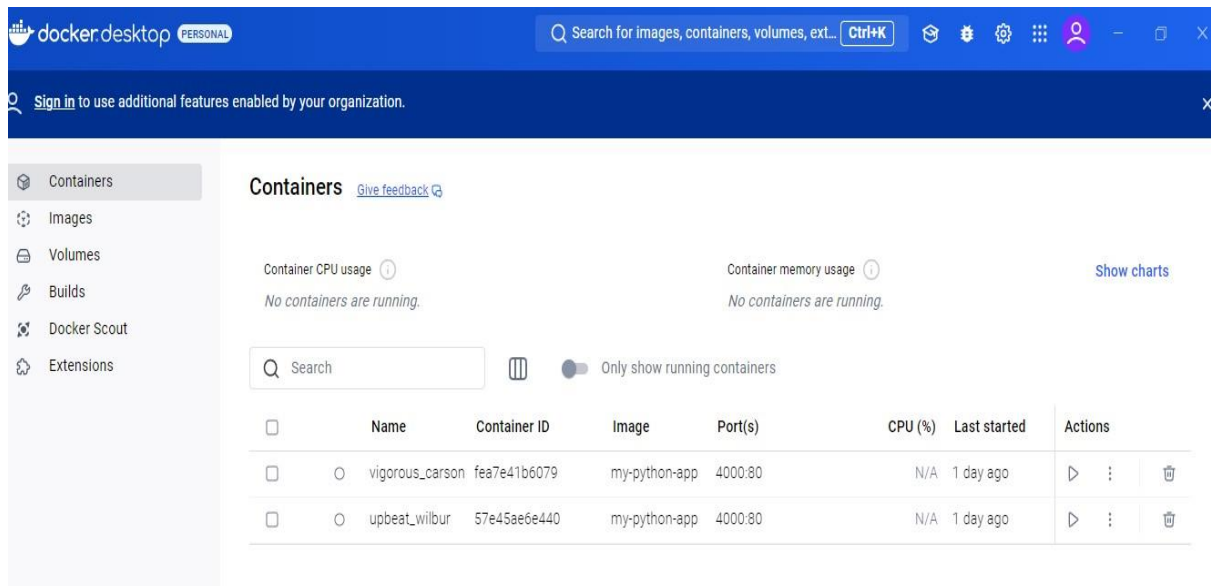


CONTAINERIZING A PYTHON APPLICATION WITH DOCKER: DEPLOYING PYTHON IN DOCKER WITH FLASK

Step 1: Write Your Python Application Code

1. Create a Python script called app.py with the following content:

```
# app.py print("Python Application using  
Docker!")
```



Step 2: Create a Dockerfile

1. Create a file named Dockerfile in the same directory as app.py. This file contains instructions for Docker to set up the environment for the Python application:

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim
# Set the working directory in the container
WORKDIR /app
# Copy the current directory contents into the container at /app
COPY . /app
# Install any needed packages specified in requirements.txt
# (Skip this step if no external packages are needed)
RUN pip install --trusted-host pypi.python.org -r requirements.txt || true
# Make port 80 available to the outside world
EXPOSE 80
# Define environment variable
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

```
C:\Users\suchithra>cd "C:\Users\suchithra\Desktop\Docke"
C:\Users\suchithra\Desktop\Docke>echo. >app.py
C:\Users\suchithra\Desktop\Docke>echo. >Dockerfile
```

Step 3: Build the Docker Image

1. Build the image with the following command, replacing my-python-app with your preferred image name: `docker build -t my-python-app .`

This command tells Docker to build an image using the Dockerfile in the current directory (.) and tag it as my-python-app.

```
C:\Users\suchithra>docker --version
Docker version 27.3.1, build ce12230

C:\Users\suchithra>cd "C:\Users\suchithra\Desktop\Docke"

C:\Users\suchithra\Desktop\Docke>echo. >app.py

C:\Users\suchithra\Desktop\Docke>echo. >Dockerfile

C:\Users\suchithra\Desktop\Docke>docker build -t my-python-app .
[+] Building 46.1s (5/9)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 620B
=> [internal] load metadata for docker.io/library/python:3.8-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613f
=> => resolve docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613f
=> => sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613f 10.41kB / 10.41kB
=> => sha256:314bc2fb0714b7807bf5699c98f0c73817e579799f2d91567ab7e9510f5601a5 1.75kB / 1.75kB
=> => sha256:b5f62925bd0f63f48cc8acd5e87d0c3a07e2f229cd2fb0a9586e68ed17f45ee3 5.25kB / 5.25kB
=> => sha256:302e3ee498053a7b5332ac79e8efebec16e900289fc1ecd1c754ce8fa047fcab 5.24MB / 29.13MB
=> => sha256:030d7bdc20a63e3d22192b292d006a69fa3333949f536d62865d1bd0506685cc 3.51MB / 3.51MB
=> => sha256:a3f1dfe736c5f959143f23d75ab522a60be2da902efac236f4fb2a153cc14a5d 1.05MB / 14.53MB
=> => sha256:3971691a363796c39467aae4cdce6ef773273fe6bfc67154d01e1b589befb912 248B / 248B
=> [internal] load build context
=> => transferring context: 1.31kB
```

ZZZ

Step 4: Run the Docker Container

1. Run the container with the following command:

```
bash Copy code docker run -p
4000:80 my-python-app
```

This command:

- Maps port 4000 on your machine to port 80 in the container.
- Starts the container and runs app.py, which will print "Python Application using Docker!".

```
C:\Users\suchithra\Desktop\Docke>docker run -p 4000:80 my-python-app
hello Docker
```

Step 5: Access Your Python Application

Since app.py simply prints text to the console and doesn't start a web server, there won't be anything to interact with at <http://localhost:4000>. To create a web-based application, you'll need to use a Python web framework like Flask.

To display the message in a web browser, follow these steps:

1. **Install Flask** by creating a requirements.txt file with this line:

Copy code Flask

2. **Update app.py** to create a simple web server:

```
python Copy code #
app.py from flask
import Flask
app = Flask(__name__)

@app.route("/") def
hello():
    return "Python Application using Docker!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=80)
```

3. **Rebuild the Docker Image** (after modifying app.py and requirements.txt):

docker build -t my-python-app .

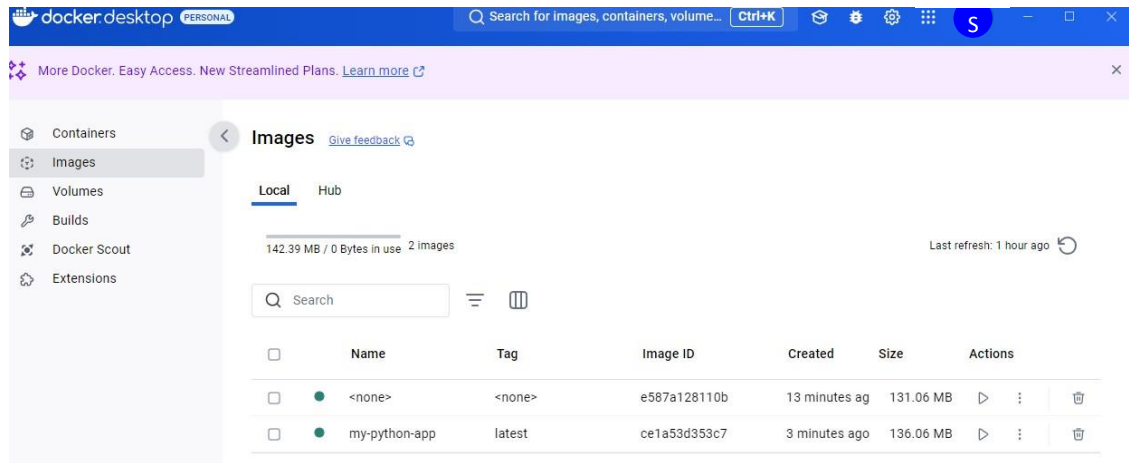
```
C:\Users\suchithra\Desktop\Docker>echo. > requirements.txt
C:\Users\suchithra\Desktop\Docker>docker build -t my-python-app .
] Building 37.6s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 620B
=> [internal] load metadata for docker.io/library/python:3.8-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613f
=> [internal] load build context
=> => transferring context: 348B
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY . /app
=> [4/4] RUN pip install --trusted-host pypi.python.org -r requirements.txt || true
=> exporting to image
=> => exporting layers
=> => writing image sha256:ce1a53d353c75bb15653ef770b01aaa79516017551fee4248cc72457bb87502e
=> => naming to docker.io/library/my-python-app

1 warning found (use docker --debug to expand):
- LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 13)
```

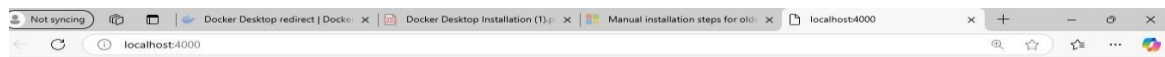
4 Run the Container Again: docker

run -p 4000:80 my-python-app

```
C:\Users\sriya\Desktop\Docker>docker run -p 4000:80 my-python-app
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:80
* Running on http://172.17.0.2:80
Press CTRL+C to quit
```



5. Access the Application by visiting <http://localhost:4000> in a browser, where you should now see "Python Application using Docker!".



Python Application Docker

This setup will give you a simple Flask web application running in Docker, accessible via your browser.