# 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!")
```

Search

| Name | Tag | Status | Created | Size | Actions |
|---|---|---|---|---|---|
| docker_demo_app<br>0bb8eb07ea3e | latest | In use | 5 days ago | 911.75 MB | |
| hello-docker<br>4f4699911e15 | latest | In use | 6 days ago | 125.5 MB | |
| mongo<br>77c59b638412 | latest | In use | 10 days ago | 855.24 MB | |
| nginx<br>3b25b682ea82 | latest | In use | 1 month ago | 191.67 MB | |
| hello-world<br>d2c94e258dcb | latest | In use | 2 years ago | 13.25 KB | |
| <none><br>b8959063377c | <none> | In use (dangling) | 9 minutes ago | 131.06 MB | |
| my-python-app<br>0509836f4512 | latest | In use | 5 minutes ago | 136.06 MB | |

## 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"]

**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 .***

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ibmtr\Desktop\python app>docker build -t my-python-app .
[+] Building 27.8s (5/9)                                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                                0.2s
 => => transferring dockerfile: 631B                                                0.1s
 => [internal] load metadata for docker.io/library/python:3.8-slim                 17.6s
 => [auth] library/python:pull token for registry-1.docker.io                       0.0s
 => [internal] load .dockerignore                                                   0.1s
 => => transferring context: 2B                                                     0.0s
 => [1/4] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13a0e4d073280665  9.9s
 => => resolve docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13a0e4d073280665  0.0s
 => => sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613 10.41kB / 10.41kB  0.0s
 => => sha256:314bc2fb0714b7807bf5699c98f0c73817e579799f2d91567ab7e9510f5601a5 1.75kB / 1.75kB   0.0s
 => => sha256:b5f62925bd0f63f48cc8acd5e87d0c3a07e2f229cd2fb0a9586e68ed17f45ee3 5.25kB / 5.25kB   0.0s
 => => sha256:302e3ee498053a7b5332ac79e8efebec16e900289fc1ecd1c754ce8fa047fcab 1.05MB / 29.13MB  9.8s
 => => sha256:030d7bdc20a63e3d22192b292d006a69fa3333949f536d62865d1bd0506685cc 0B / 3.51MB       9.8s
 => => sha256:a3f1dfe736c5f959143f23d75ab522a60be2da902efac236f4fb2a153cc14a5 14.53MB / 14.53MB  9.1s
 => => sha256:3971691a363796c39467aae4cdce6ef773273fe6bfc67154d01e1b589befb912 0B / 248B         9.8s
 => [internal] load build context                                                   0.1s
 => => transferring context: 722B                                                   0.0s
```

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

**Step 4: Run the Docker Container**

1. **Run the container** with the following command:

***docker run -p 4000:80 my-python-app***

```
C:\Users\ibmtr\Desktop\python app>docker run -p 4000:80 my-python-app
Python Application using Docker!
```

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!".

**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. If you want a web-based application, you'll need to use a Python web framework like Flask.

For example, if you want to display the message in a web browser, modify app.py to use Flask:

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

```
Flask
```

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

```
# 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\ibmtr\Desktop\python app>docker build -t my-python-app .
[+] Building 19.2s (9/9) FINISHED                                   docker:desktop-linux
 => [internal] load build definition from Dockerfile                                  0.1s
 => => transferring dockerfile: 631B                                                  0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                    2.0s
 => [internal] load .dockerignore                                                     0.0s
 => => transferring context: 2B                                                       0.0s
 => [internal] load build context                                                     0.1s
 => => transferring context: 328B                                                     0.0s
 => [1/4] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13a0e4d073280665  0.0s
 => CACHED [2/4] WORKDIR /app                                                         0.0s
 => [3/4] COPY . /app                                                                 0.1s
 => [4/4] RUN pip install --trusted-host pypi.python.org -r requirements.txt || true 16.3s
 => exporting to image                                                                0.4s
 => => exporting layers                                                               0.4s
 => => writing image sha256:0509836f451254762b3cf6beba2883cb560679ebecc677933fc1e0666f1c4a91  0.0s
 => => naming to docker.io/library/my-python-app                                      0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/vx69qxnx9mhg7bt8lbm6j
6pfv

 1 warning found (use docker --debug to expand):
 - LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line
 18)
```

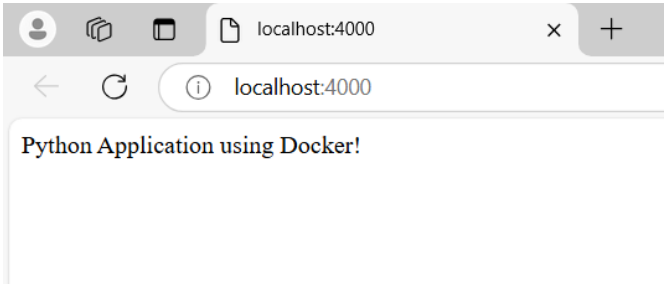
4. **Run the Container Again**:

docker run -p 4000:80 my-python-app

```
C:\Users\ibmtr\Desktop\python app>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
```

| Name | Image | Status | Port(s) | CPU (%) | Last started | Actions |
|---|---|---|---|---|---|---|
| busy_faraday<br>fd4936bf68be | docker_demo_app | Exited (255) | 3000:3000 | 0% | 5 days ago | |
| thirsty_brown<br>397060cdf494 | hello-docker | Exited | | 0% | 6 days ago | |
| youthful_pascal<br>67e2c67c3cd6 | my-python-app | Exited | 4000:80 | 0% | 6 minutes ago | |
| romantic_jennings<br>7c884ce9c13d | my-python-app | Running | 4000:80 | 0.03% | 2 minutes ago | |

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




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