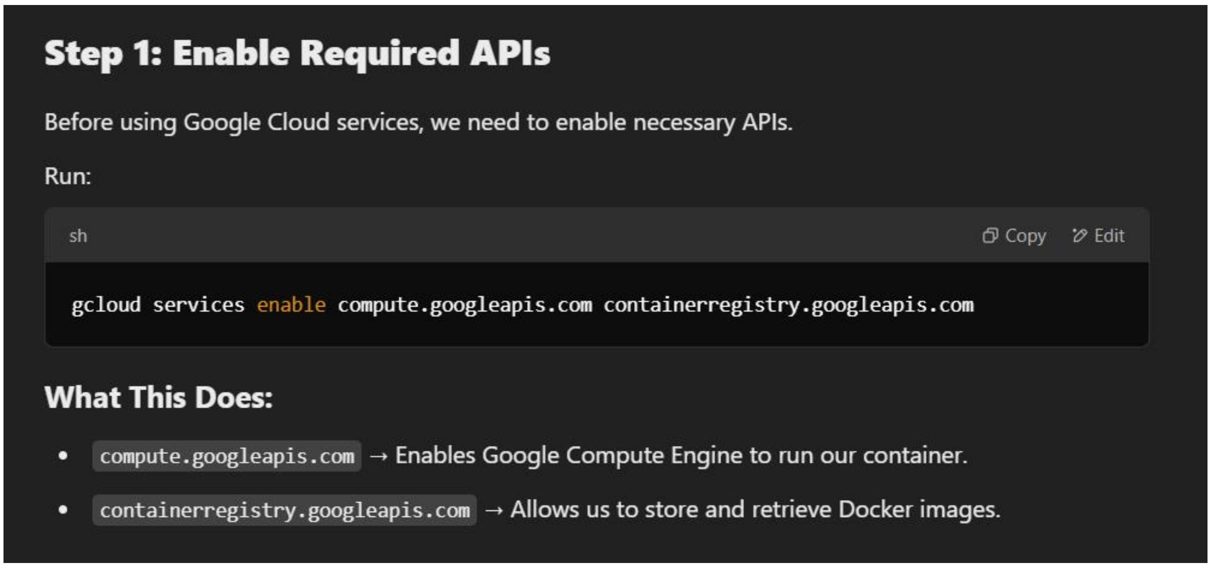
PRACTICAL 7



Step 1: Enable Required APIs

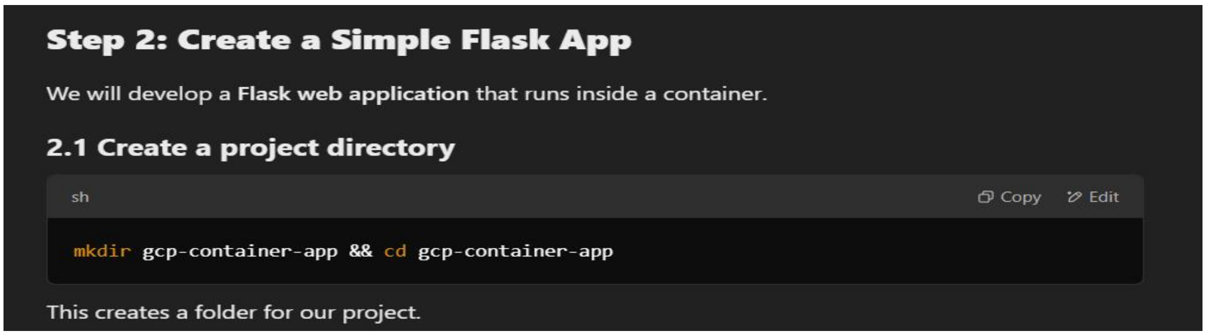
Before using Google Cloud services, we need to enable necessary APIs.

Run:

```sh
gcloud services enable compute.googleapis.com containerregistry.googleapis.com
```

What This Does:

- `compute.googleapis.com` → Enables Google Compute Engine to run our container.
- `containerregistry.googleapis.com` → Allows us to store and retrieve Docker images.



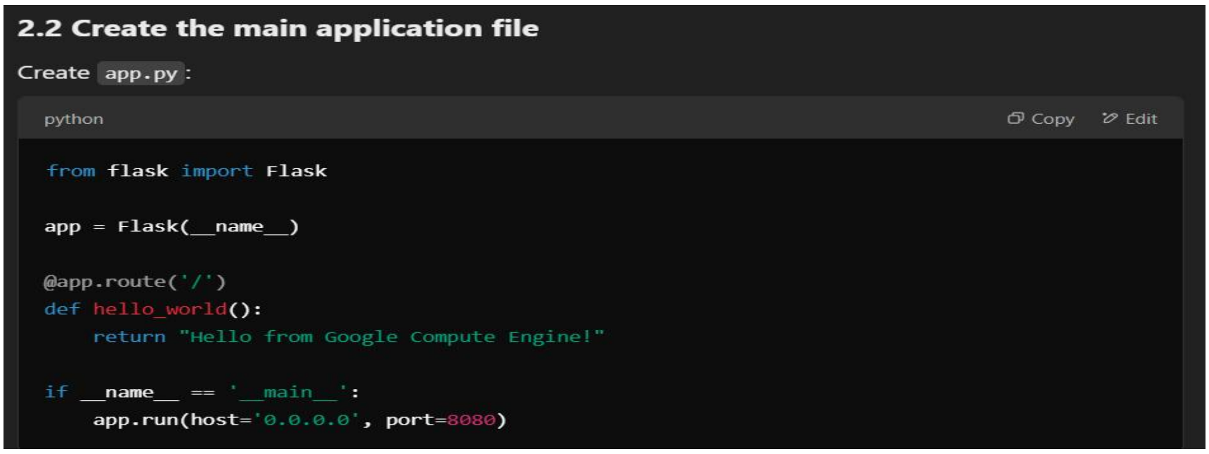
Step 2: Create a Simple Flask App

We will develop a **Flask web application** that runs inside a container.

2.1 Create a project directory

```sh
mkdir gcp-container-app && cd gcp-container-app
```

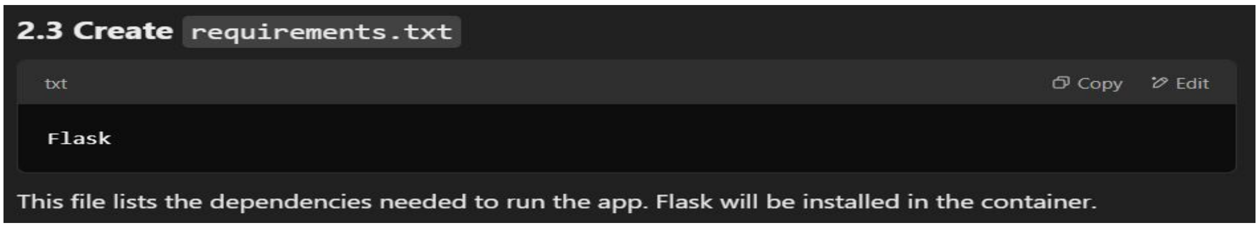
This creates a folder for our project.



2.2 Create the main application file

Create `app.py`:

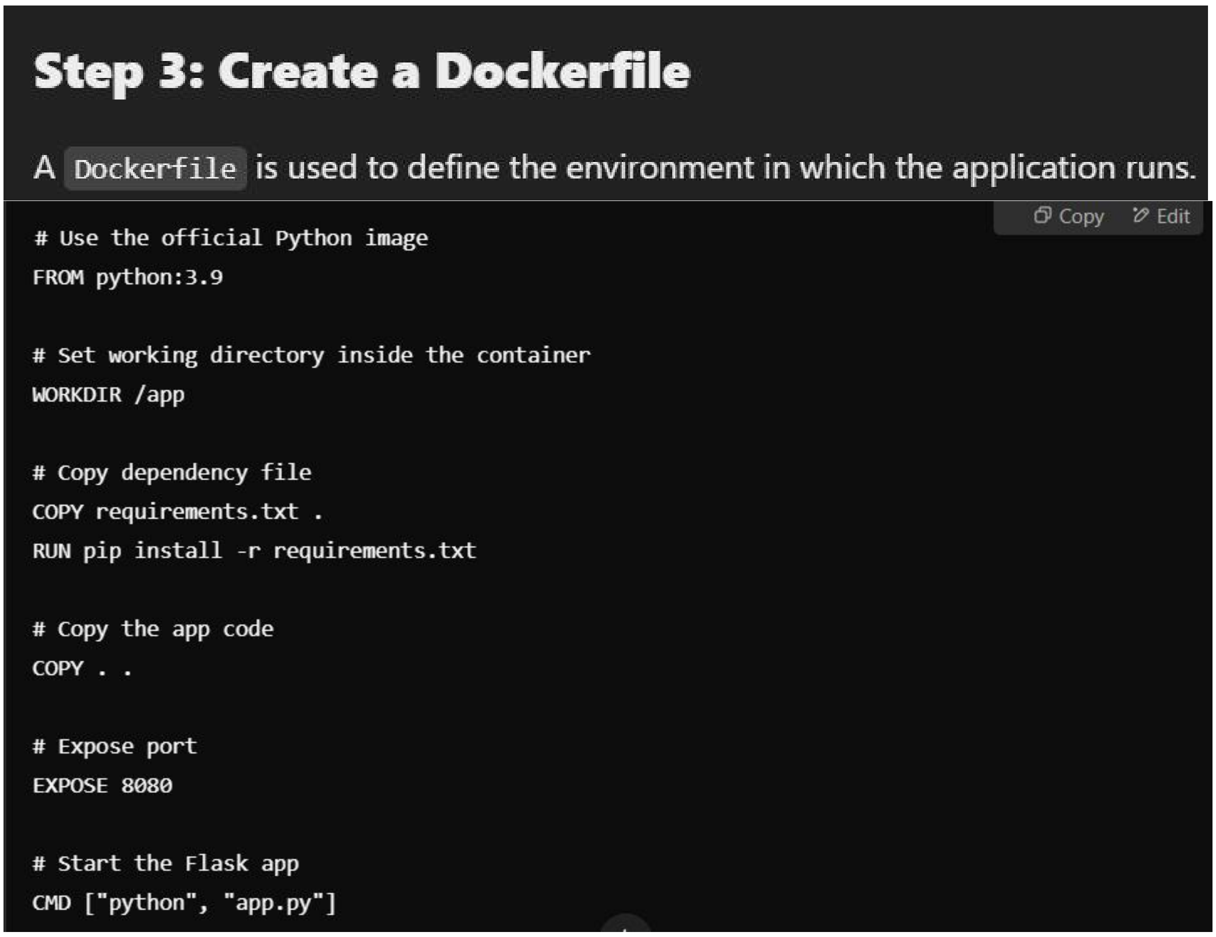
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello from Google Compute Engine!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```



2.3 Create `requirements.txt`

```txt
Flask
```

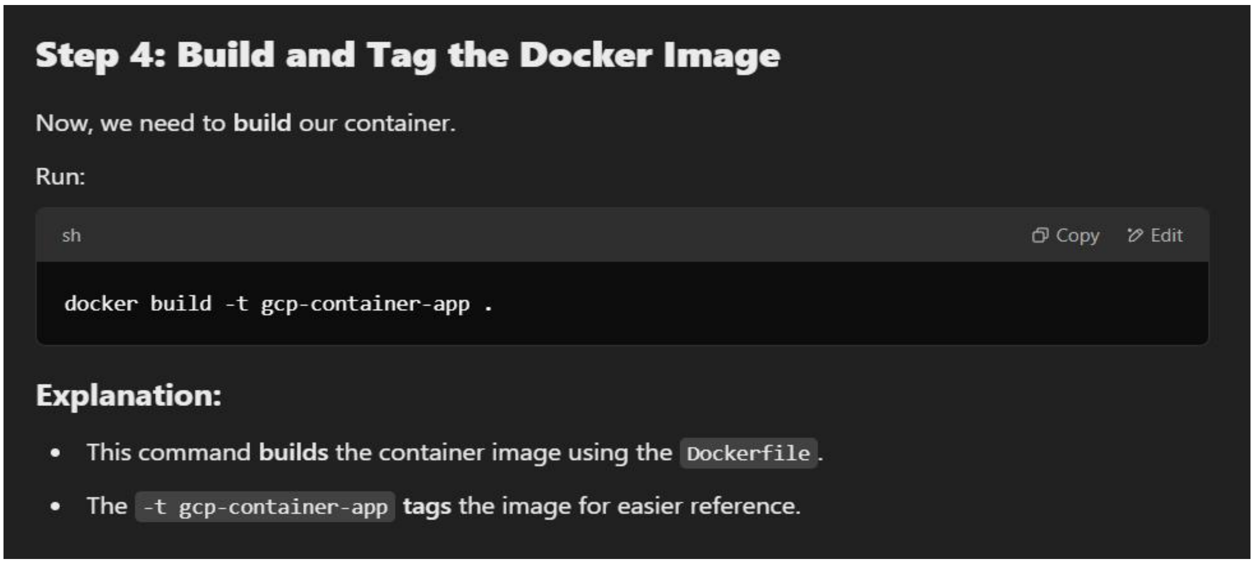
This file lists the dependencies needed to run the app. Flask will be installed in the container.



Step 3: Create a Dockerfile

A `Dockerfile` is used to define the environment in which the application runs.

```
# Use the official Python image
FROM python:3.9

# Set working directory inside the container
WORKDIR /app

# Copy dependency file
COPY requirements.txt .
RUN pip install -r requirements.txt

# Copy the app code
COPY . .

# Expose port
EXPOSE 8080

# Start the Flask app
CMD ["python", "app.py"]
```



Step 4: Build and Tag the Docker Image

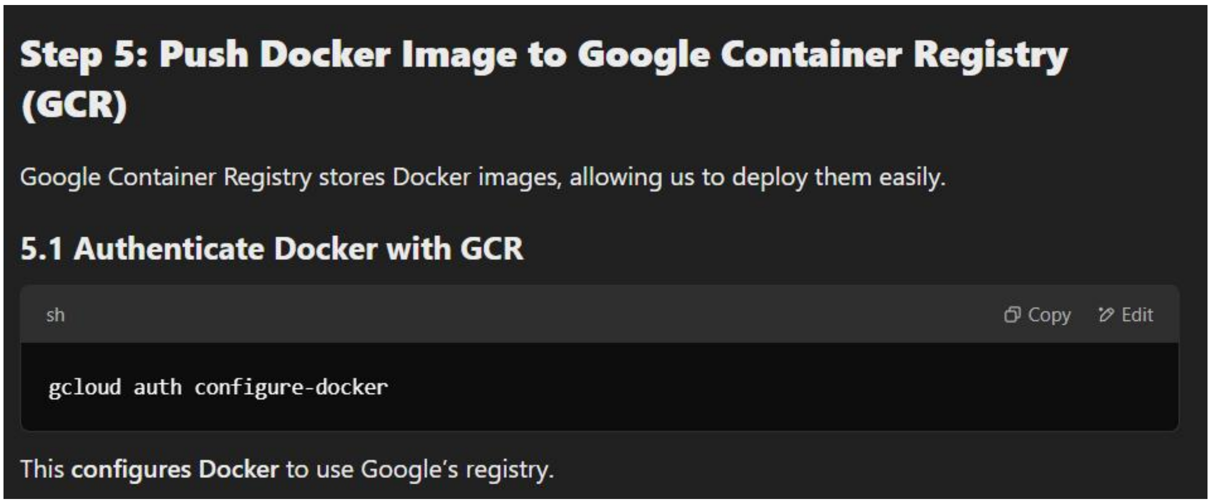
Now, we need to **build** our container.

Run:

```sh
docker build -t gcp-container-app .
```

Explanation:

- This command **builds** the container image using the `Dockerfile`.
- The `-t gcp-container-app` **tags** the image for easier reference.



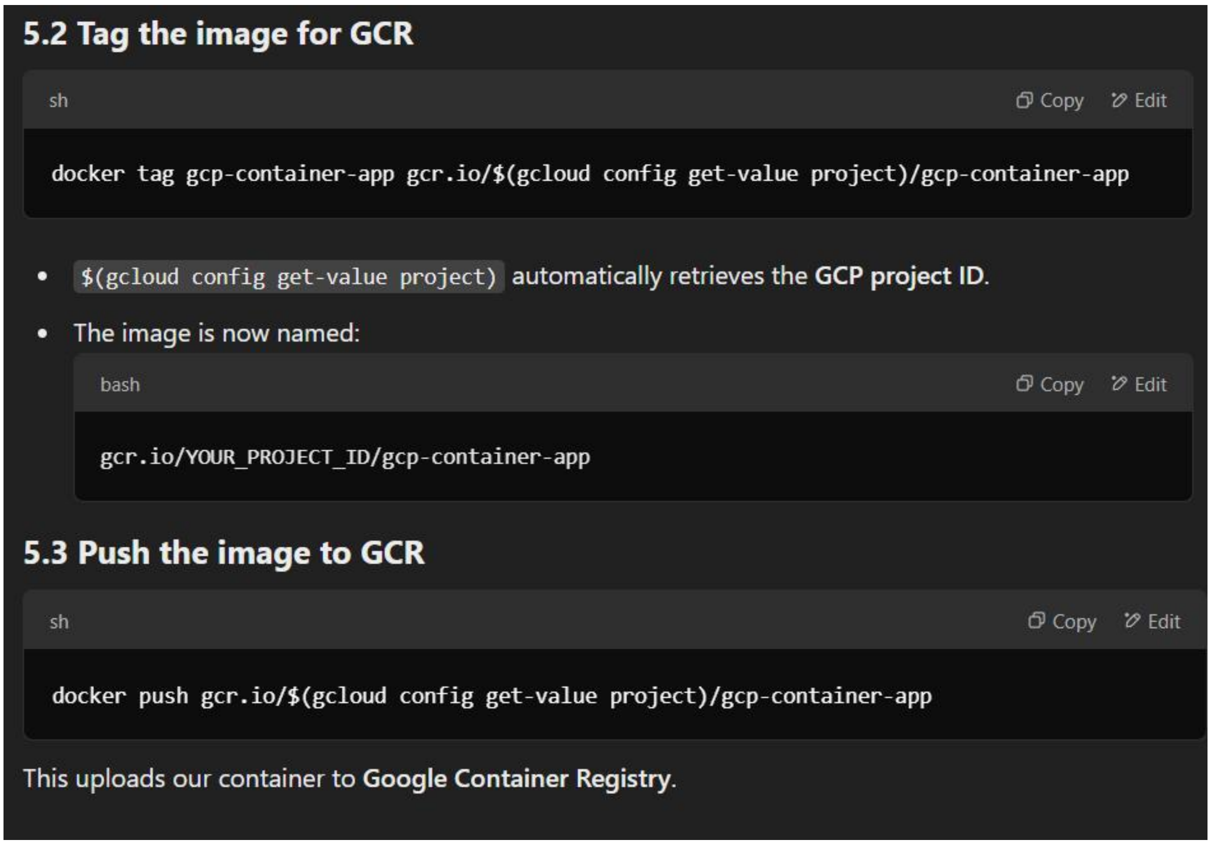
Step 5: Push Docker Image to Google Container Registry (GCR)

Google Container Registry stores Docker images, allowing us to deploy them easily.

5.1 Authenticate Docker with GCR

```sh
gcloud auth configure-docker
```

This **configures Docker** to use Google's registry.



5.2 Tag the image for GCR

```sh
docker tag gcp-container-app gcr.io/$(gcloud config get-value project)/gcp-container-app
```

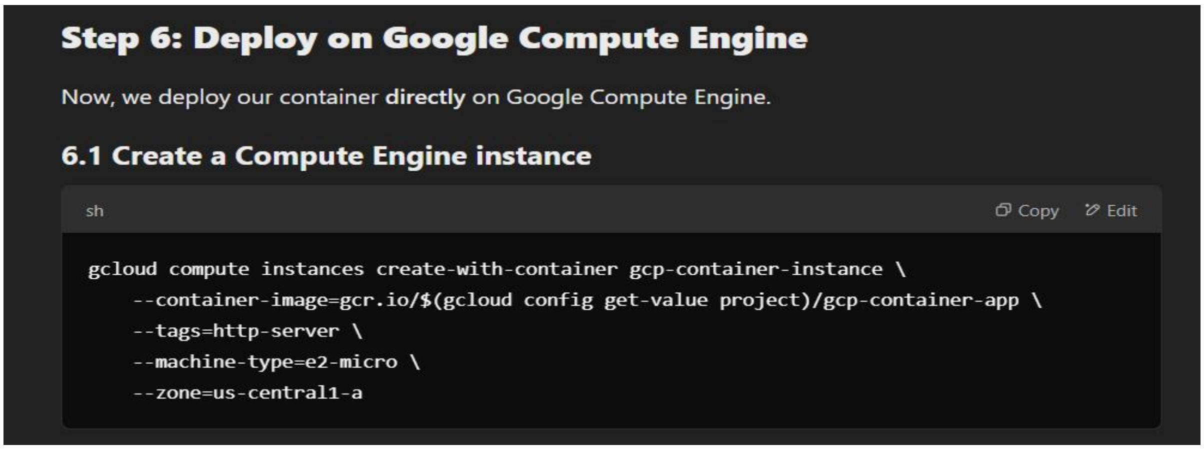
- `$(gcloud config get-value project)` automatically retrieves the **GCP project ID**.
- The image is now named:

```bash
gcr.io/YOUR_PROJECT_ID/gcp-container-app
```

5.3 Push the image to GCR

```sh
docker push gcr.io/$(gcloud config get-value project)/gcp-container-app
```

This uploads our container to **Google Container Registry**.

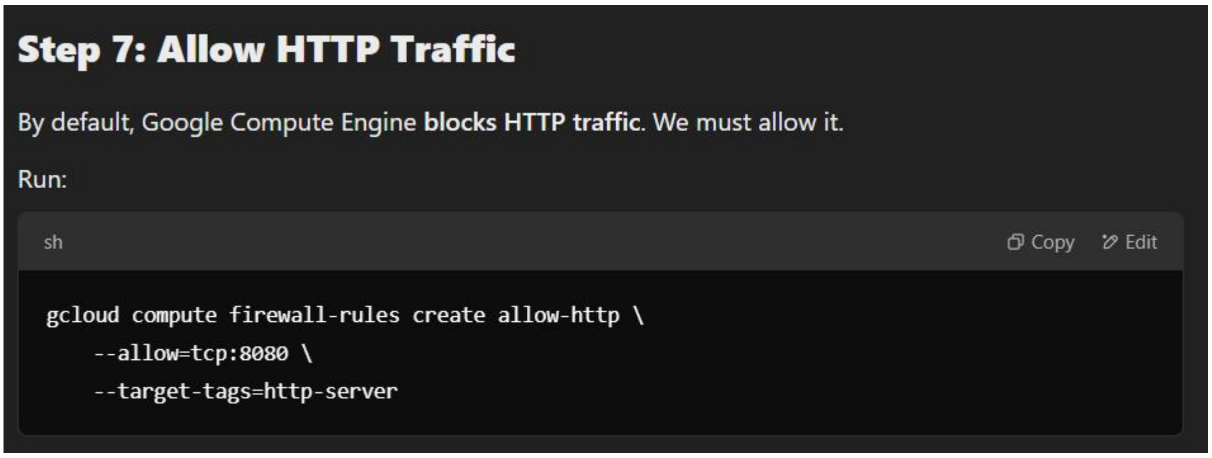


Step 6: Deploy on Google Compute Engine

Now, we deploy our container **directly** on Google Compute Engine.

6.1 Create a Compute Engine instance

```sh
gcloud compute instances create-with-container gcp-container-instance \
    --container-image=gcr.io/$(gcloud config get-value project)/gcp-container-app \
    --tags=http-server \
    --machine-type=e2-micro \
    --zone=us-central1-a
```

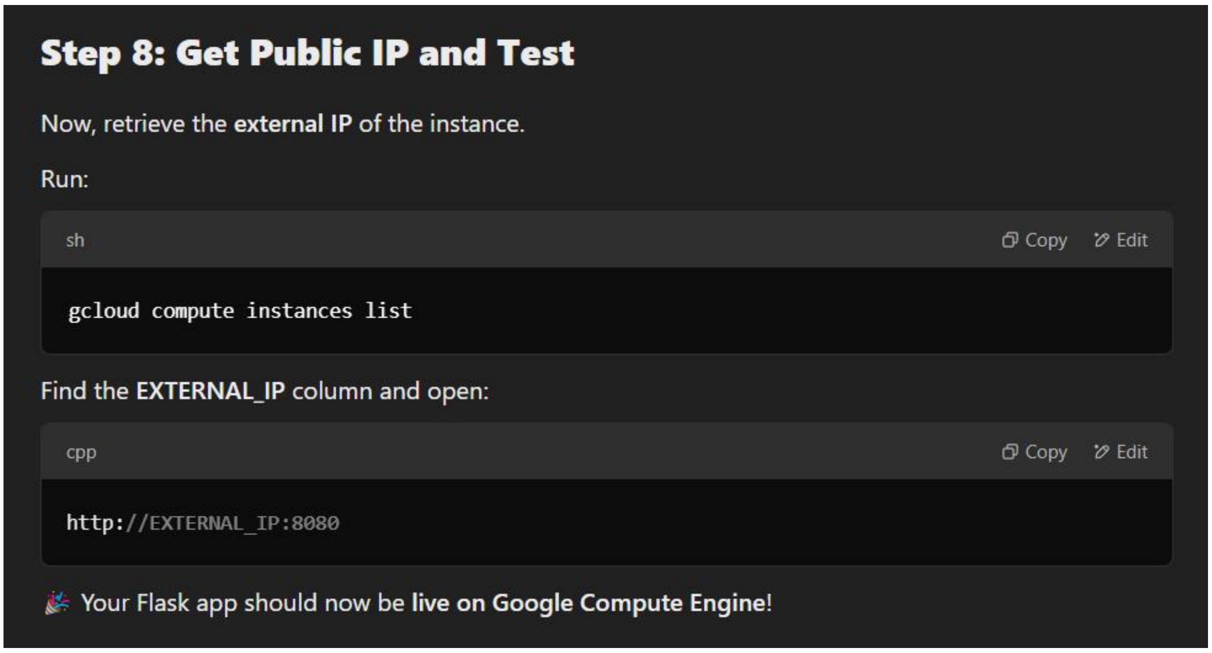


Step 7: Allow HTTP Traffic

By default, Google Compute Engine **blocks HTTP traffic**. We must allow it.

Run:

```sh
gcloud compute firewall-rules create allow-http \
    --allow=tcp:8080 \
    --target-tags=http-server
```



Step 8: Get Public IP and Test

Now, retrieve the **external IP** of the instance.

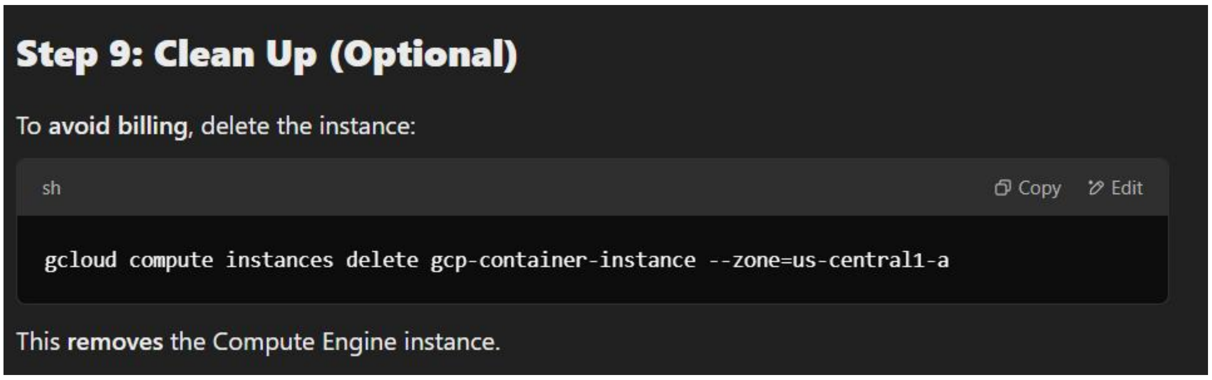
Run:

```sh
gcloud compute instances list
```

Find the **EXTERNAL_IP** column and open:

```cpp
http://EXTERNAL_IP:8080
```

🎉 Your Flask app should now be **live on Google Compute Engine**!



Step 9: Clean Up (Optional)

To **avoid billing**, delete the instance:

```sh
gcloud compute instances delete gcp-container-instance --zone=us-central1-a
```

This **removes** the Compute Engine instance.