**EXPERIMENT -6: Docker Compose(Python)**

**Prerequisites**

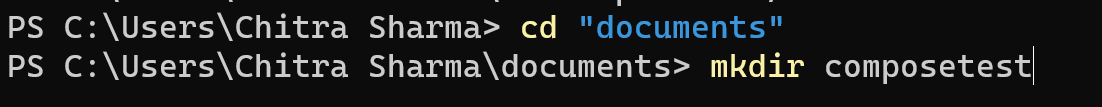
You need to have Docker Engine and Docker Compose on your machine. You can either:

* + Install Docker Engine and Docker Compose as standalone binaries
  + Install Docker Desktop which includes both Docker Engine and Docker Compose

**Steps**

**Step-1:** Firstly, create a directory for the project. Open a terminal and navigate to the folder where you want to make the project directory. After navigating to the project directory, make the directory using the command:

mkdir directory-name



**Step-2:** Now navigate to the folder/directory that you just created.

cd directory-name



**Step-3:** In the same project directory, create an “app.py” file and paste the following code inside it:

import time

import redis

from flask import Flask

app = Flask(\_\_name\_\_)

cache = redis.Redis(host='redis', port=6379)

def get\_hit\_count():

retries = 5

while True:

try:

return cache.incr('hits')

except redis.exceptions.ConnectionError as exc:

if retries == 0:

raise exc

retries -= 1

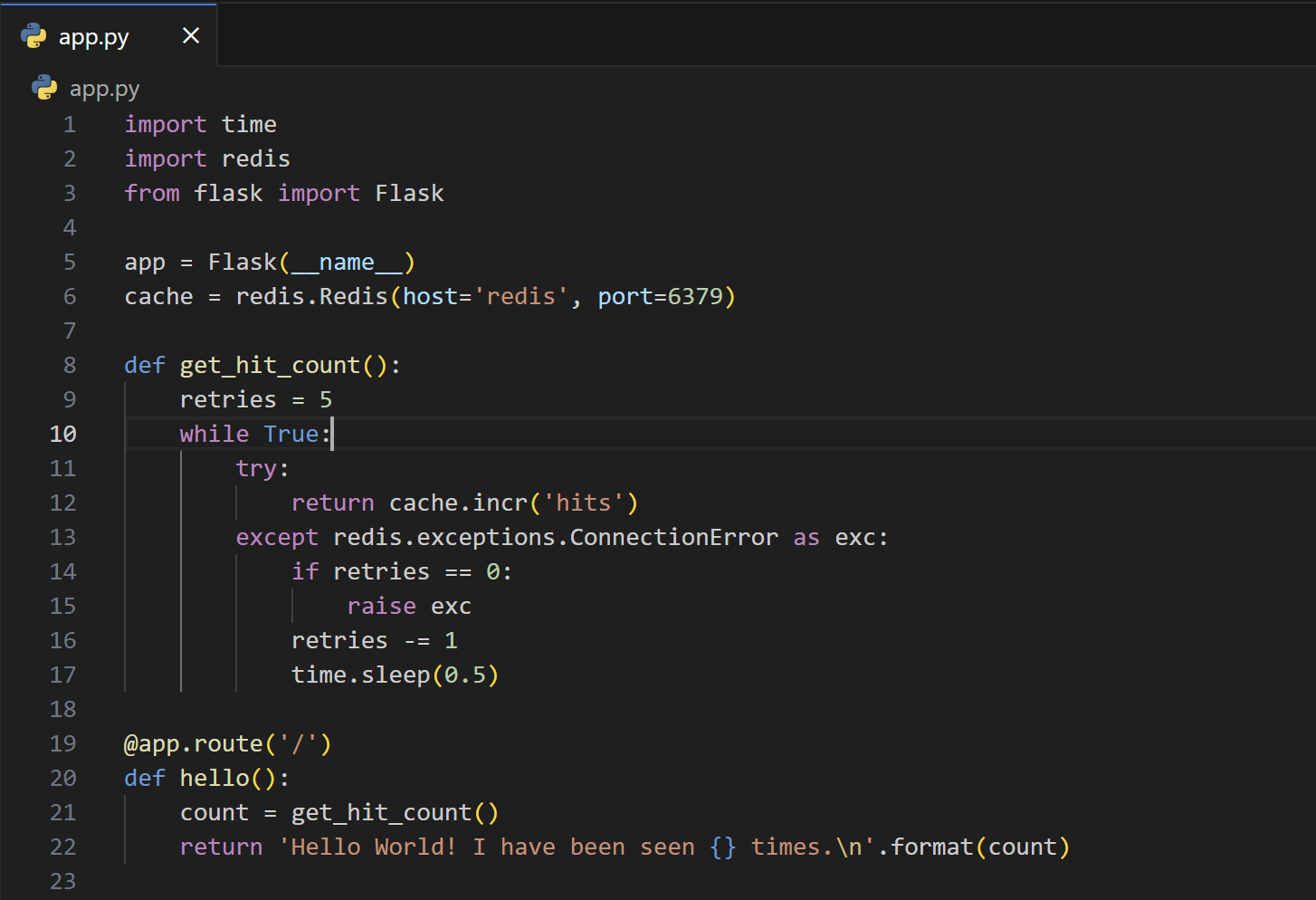
time.sleep(0.5)

@app.route('/')

def hello():

count = get\_hit\_count()

return 'Hello World! I have been seen {} times.\n'.format(count)

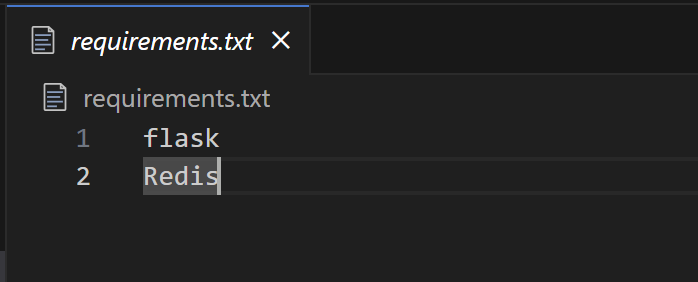


* This Python code sets up a Flask web application that, when accessed via a web browser, displays a simple message: "Hello World! I have been seen {} times," where the curly braces {} are replaced with the number of times the web page has been accessed.
* It uses a Redis database as a caching mechanism to store and increment a hit counter. The get\_hit\_count function increments this counter while handling potential connection errors to the Redis server with retries and sleeps. The web application's main route (/) invokes this function to retrieve and display the hit count in the response.

**Step-4:** Now, create the “requirements.txt” file in the project directory and paste the following code in:

flask

Redis



* This requirements.txt file is used in Python to specify and manage project dependencies.

**Step-5:** Now create a dockerfile. In the project directory, create a file named dockerfile; to be saved in all files format within the double quotes i.e. “dockerfile”. Then paste the following code in the file:

# syntax=docker/dockerfile:1

FROM python:3.7-alpine

WORKDIR /code

ENV FLASK\_APP=app.py

ENV FLASK\_RUN\_HOST=0.0.0.0

RUN apk add --no-cache gcc musl-dev linux-headers

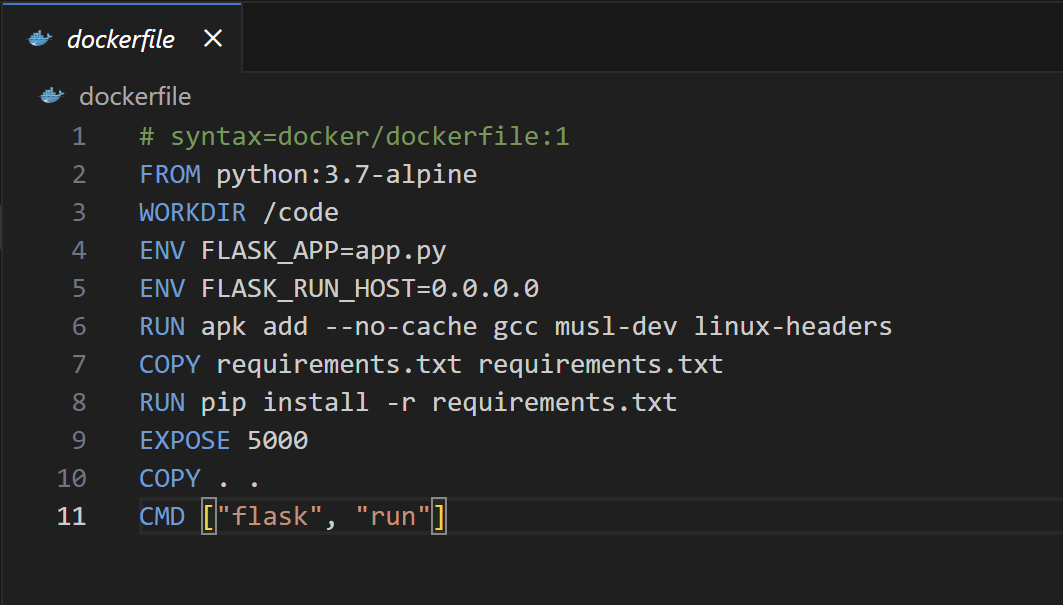
COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

EXPOSE 5000

COPY . .

CMD ["flask", "run"]



**This dockerfile tells Docker to:**

* Build an image starting with the Python 3.7 image.
* Set the working directory to /code.
* Set environment variables used by the flask command.
* Install gcc and other dependencies
* Copy requirements.txt and install the Python dependencies.
* Add metadata to the image to describe that the container is listening on port 5000
* Copy the current directory . in the project to the workdir . in the image.
* Set the default command for the container to flask run.

**Step-6:** At last, create a file named “compose.yaml” in the project directory and paste the following :

services:

web:

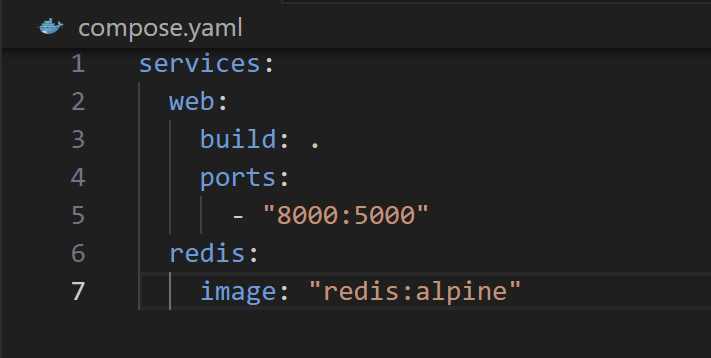
build: .

ports:

- "8000:5000"

redis:

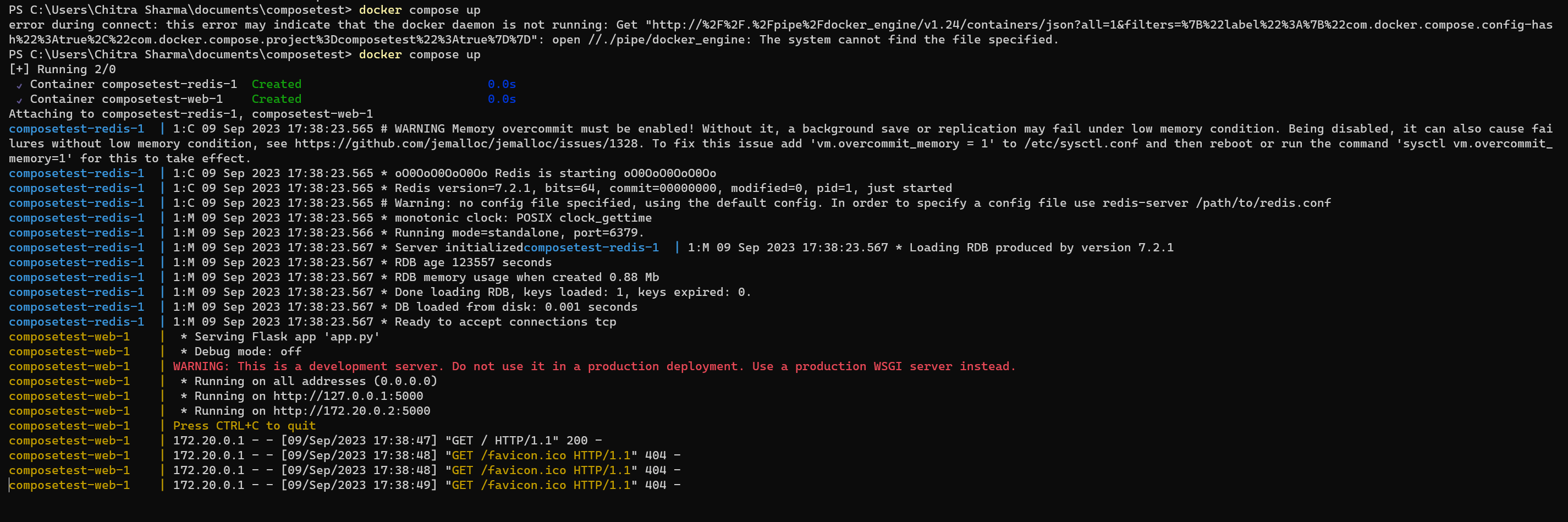
image: "redis:alpine"



* This docker-compose.yaml file is used to define and manage multi-container Docker applications.
* This Compose file defines two services: web and redis.

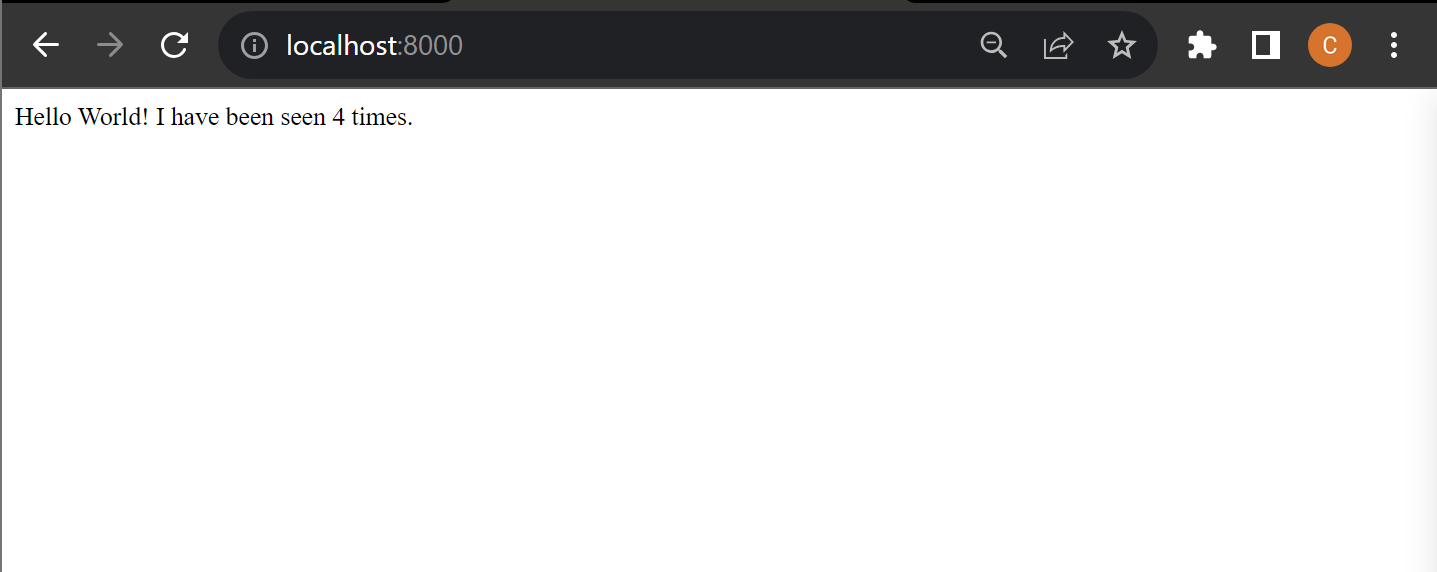
**Step-7: Now from the project directory, start up the application by running**

docker compose up

**  
  
Step-8:** Enter http://localhost:8000/ in a browser to see the application running. If this doesn't resolve, you can also try <http://127.0.0.1:8000>.

You should see a message in your browser saying:

**Hello World! I have been seen 1 times.**

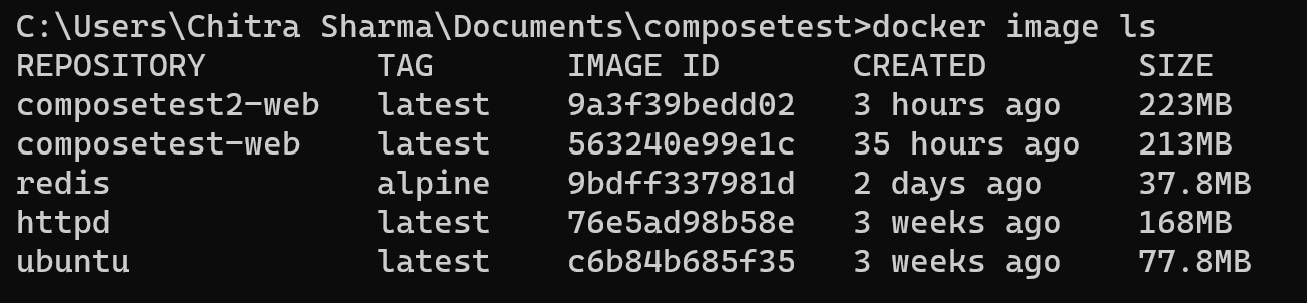


**Step-9:** Now, refresh the page and the number should increment.

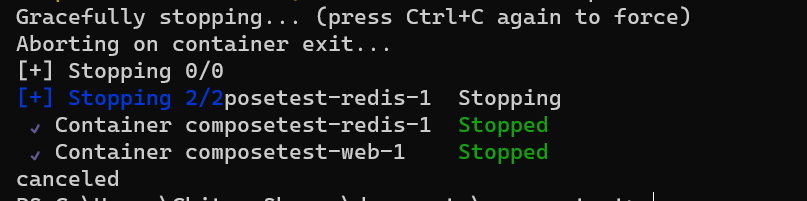


**Step-10:** Now switch to a different terminal window. Navigate to the project directory and list local images. Listing images at this point should return **redis** and **web**.

docker image ls

****

**Step-11**: Stop the application, either by running ***docker compose down***from within your project directory in the second terminal, or by hitting ***CTRL+C***in the original terminal where you started the app.



**Step-12:** Now, edit the Compose file to add a bind mount.

services:

web:

build: .

ports:

- "8000:5000"

volumes:

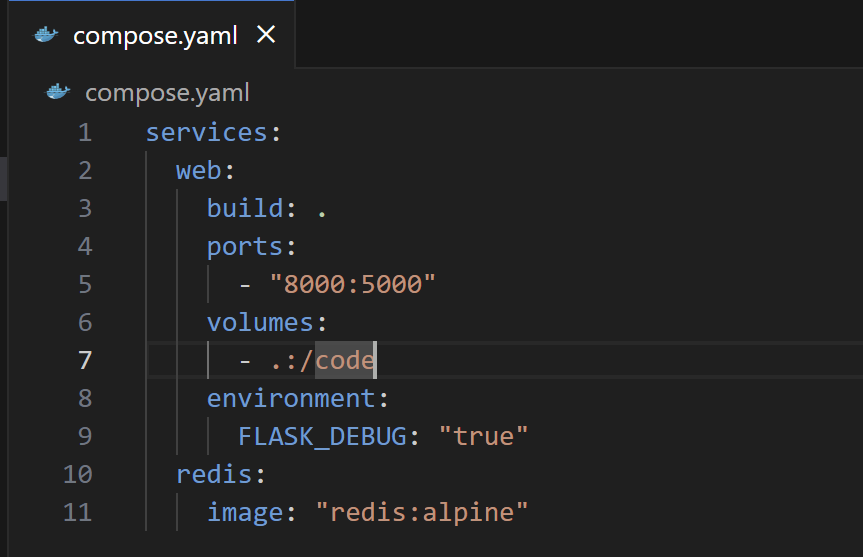
- .:/code

environment:

FLASK\_DEBUG: "true"

redis:

image: "redis:alpine"



**Step-13:** Now from the project directory, build the app with the updated Compose file ,and run it.

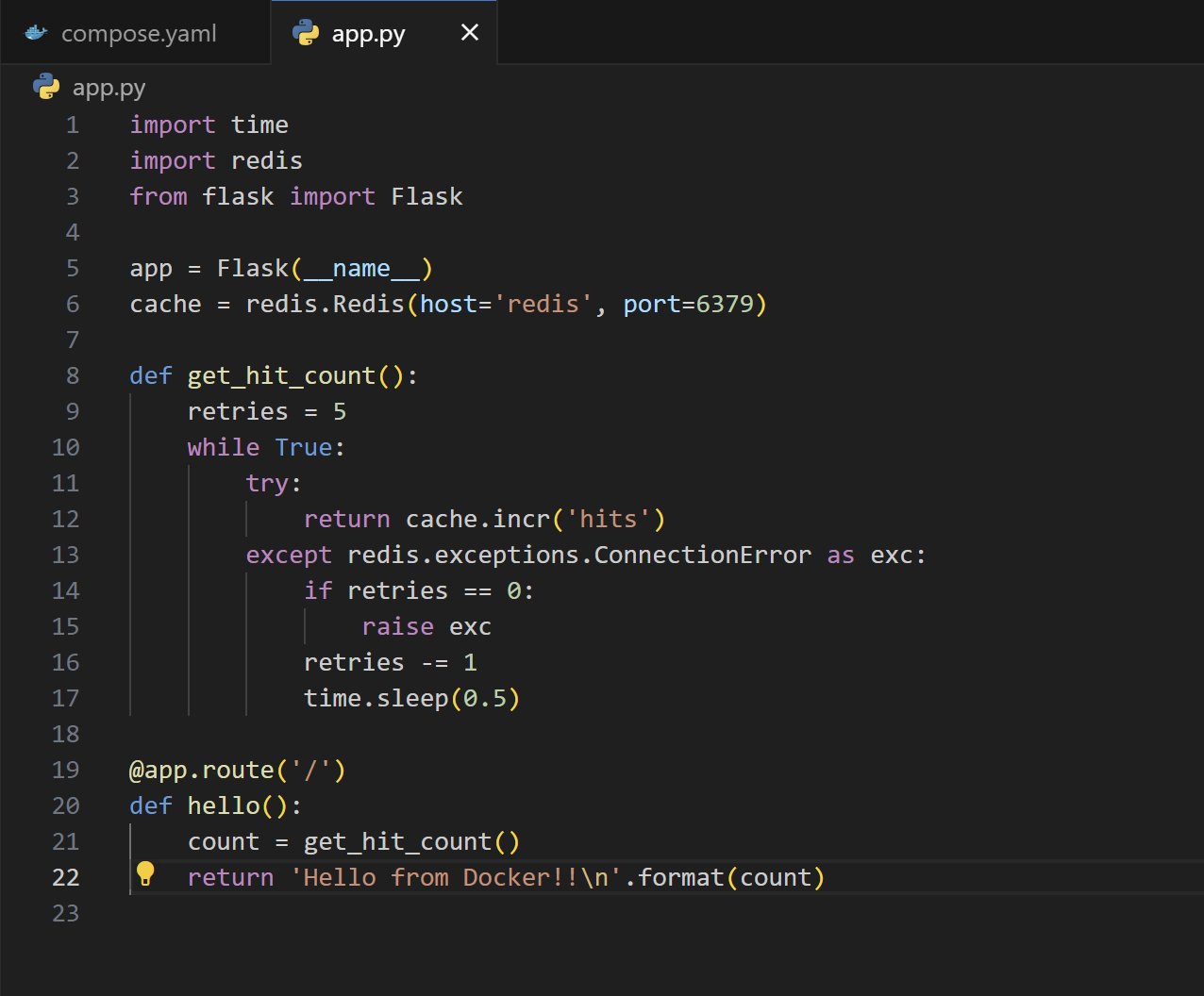
docker compose up –build



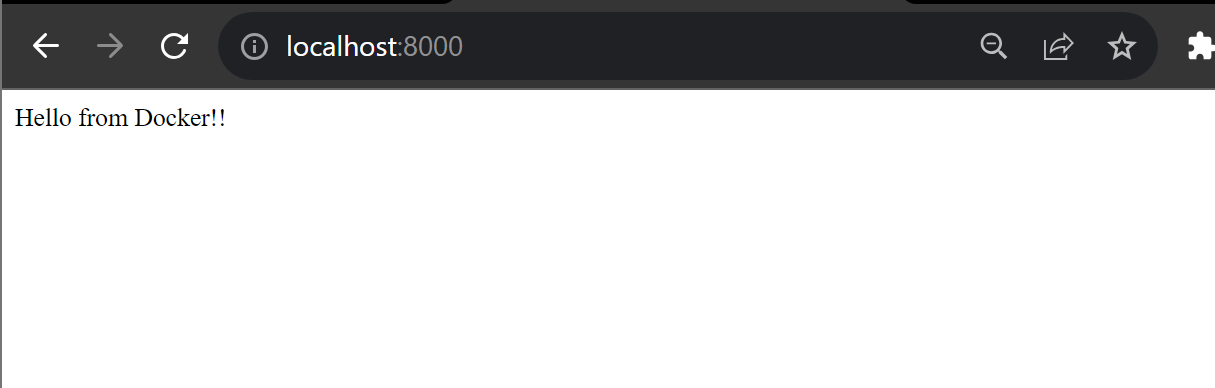


**Step-14:** Now, change the greeting in **app.py** and save it. Refresh the app in your browser. The greeting should be updated.

For example, change the Hello World! message **to Hello from Docker!!**



* As the application code is now mounted into the container using a volume, we can make changes to its code and see the changes instantly, without having to rebuild the image.



* We didn’t need to rebuild the application and run it since the application code is now mounted into the container using a volume