Program 7. Containerize applications using Docker

1. Create and open a new working folder with application code

- Create the **app** folder and add the **main.py** file to the folder.
- Create a simple **Flask application** in the **main.py** file that displays some text that is displayed in your browser when you access the application.

Main.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    out = (
        f'Example container application.<br>'
    )
    return out

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

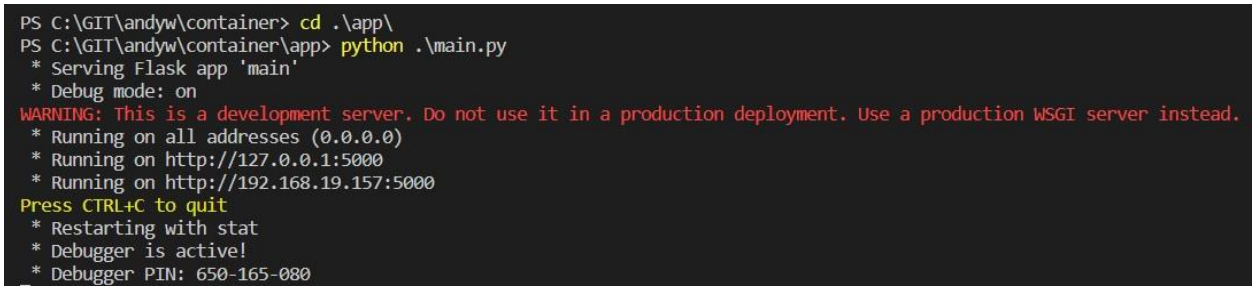
- This code is an example Flask application to listen for requests. When you send a request to the application, you will see the required text in your browser. The default listening port for the application is 5000.

2. Start the application

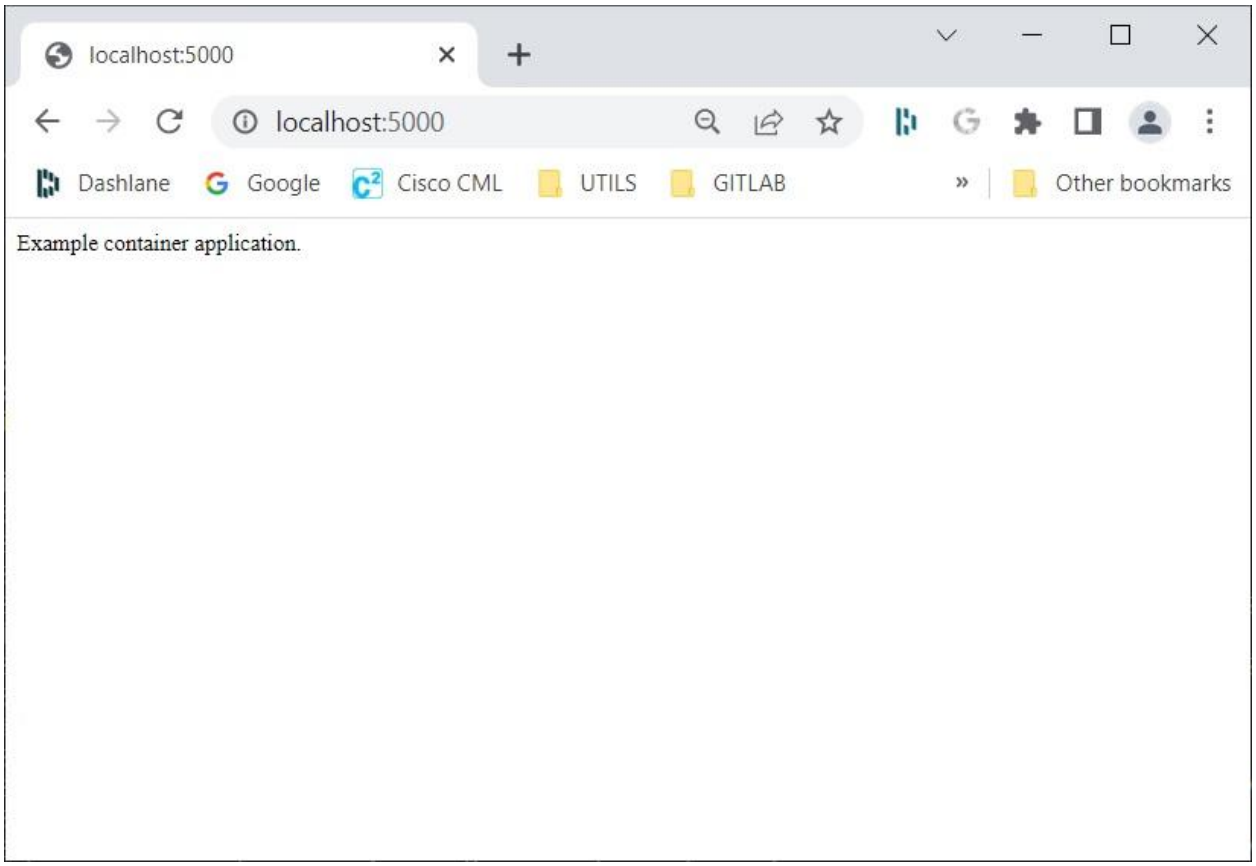
- In the terminal window, **cd** to the **app** folder and run the application.

  cd .\app
  python3 main.py

- The application is started and is listening on TCP port 5000



- In a web browser open the URL **http://localhost:5000** URL to access the application and check the required text is visible:



3. Update the requirements.txt file

- Create the **requirements.txt** file in the app folder, used to install required packages in the container and add the entry **Flask==2.1.1** to the file.

4. Define the Docker container

- Create a new file called **Dockerfile** in the **app** folder. Add the following content to the file:

Dockerfile

```
FROM python:3.7
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
EXPOSE 5000
```

CMD ["python3", "main.py"]

Explanation:

FROM python:3.7 — This instruction specifies that container will use the Python Docker container as a base. The text after ":" specifies the tag, which defines the specific version of the base container that you want to use.

COPY. /app — When you build the container, the content of the current folder (".") will be copied to the /app folder in the container.

WORKDIR/app — This instruction specifies that the command that follows it will be executed from this folder.

RUN pip install -r requirements.txt — The RUN instruction specifies the commands that will be executed when building the container. This particular instruction will install required Python packages to the container.

EXPOSE 5000 — This instruction specifies that the container will listen on the TCP port 5000 at run time.

CMD["python3", "main.py"]—This instruction specifies the commands that will be executed when you start the container.

- Build the container using the command:

  docker build -t app .

```
PS C:\GIT\andyw\container> docker build -t app .
[+] Building 38.3s (11/11) FINISHED
 => [internal] load build definition from Dockerfile                                0.1s
 => => transferring dockerfile: 941B                                                0.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 400B                                                   0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                 21.7s
 => [internal] load build context                                                   0.0s
 => => transferring context: 623B                                                   0.0s
 => [1/6] FROM docker.io/library/python:3.8-slim@sha256:26bd1964c63c88054562da7c   10.7s
 => => resolve docker.io/library/python:3.8-slim@sha256:26bd1964c63c88054562da7cd   0.0s
 => => sha256:31b3f1ad4ce1f369084d0f959813c51df0ca17d9877d5ee88 31.40MB / 31.40MB   8.3s
 => => sha256:f335cc1597f2f2d13ceea1c9b386aa1ac28efc46906a0a9cf1b 1.08MB / 1.08MB   1.6s
 => => sha256:2d22d27d211397686b4bf449ca6ba0922cb5c985a2b54645c 11.33MB / 11.33MB   7.7s
 => => sha256:26bd1964c63c88054562da7cd6eb6c612beec8e4629874bbd66 1.86kB / 1.86kB   0.0s
 => => sha256:36eb12ba900acfdcc6f1351c89b695c806d54439d584b290001 1.37kB / 1.37kB   0.0s
 => => sha256:bbf676da2044cefba42d51b5d7203e5a38bd9e2b60c0071713f 7.53kB / 7.53kB   0.0s
 => => sha256:9913e6455486eeeea6da9a3cfe24e3aac6afa38dbb68d02b57a3885 233B / 233B   1.9s
 => => sha256:eede1fc00b8a111ba9d6da2eaf130cd5f11ce57f244d0447bb5 3.18MB / 3.18MB   4.5s
 => => extracting sha256:31b3f1ad4ce1f369084d0f959813c51df0ca17d9877d5ee88c2db6ff   1.3s
 => => extracting sha256:f335cc1597f2f2d13ceea1c9b386aa1ac28efc46906a0a9cf1b4e368   0.1s
 => => extracting sha256:2d22d27d211397686b4bf449ca6ba0922cb5c985a2b54645c87a405e   0.4s
 => => extracting sha256:9913e6455486eeeea6da9a3cfe24e3aac6afa38dbb68d02b57a38854   0.0s
 => => extracting sha256:eede1fc00b8a111ba9d6da2eaf130cd5f11ce57f244d0447bb52d147   0.2s
 => [2/6] COPY requirements.txt .                                                   0.2s
 => [3/6] RUN python -m pip install -r requirements.txt                             4.8s
 => [4/6] WORKDIR /app                                                              0.0s
 => [5/6] COPY . /app                                                               0.0s
 => [6/6] RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R    0.6s
```

- Then test that your application in the container is working.

- Start the container with the command:

  docker run -it -p 5000:5000 app

```
PS C:\GIT\andyw\container\app> docker run -it -p 5000:5000 app
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
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://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 790-405-271
```

- Test that you can still access the application via the web browser at location **http://localhost:5000#**

5. Define the load balancer container

- Use nginx as a load balancer software to add two endpoints to the configuration.

- Update your **app/main.py** application to display the IP address of the current container that is used for the request.

- Create a new folder with the name **lb**. Add the **nginx.conf** file to the folder to be used as the configuration file for the load balancer.

nginx.conf

```
events {}
http {

  upstream myapp {
    server 172.20.0.100:5000;
    server 172.20.0.101:5000;
  }

  server {
    listen 8080;
    server_name localhost;

    location / {
```

```
        proxy_pass http://myapp;
        proxy_set_header Host $host;
    }
  }
}
```

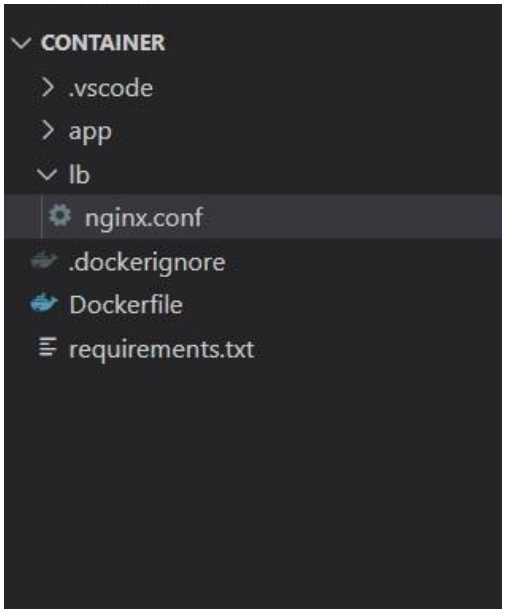
- Create the file **Dockerfile** in the **lb** folder to specify the load balancer container.

- Add the following content to the **Dockerfile** file.

Dockerfile

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 8080
CMD ["nginx", "-g", "daemon off;"]
```

Files



- Build the container in the **lb** folder, with the command:

  docker build -t lb .

- Add code to locate the IP address of the current application container and output the IP address in the HTTP request display. Update the code in the **app/main.py** file.

main.py

```
from flask import Flask
import socket

ip = socket.gethostbyname(socket.gethostname())

app = Flask(__name__)

@app.route('/')
def home():
    out = (
        f'Welcome to Cisco DevNet.<br>'
        f'IP address of the server is {ip}.<br><br>'
    )
    return out

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

- Then rebuild the application container from the app folder via the command:

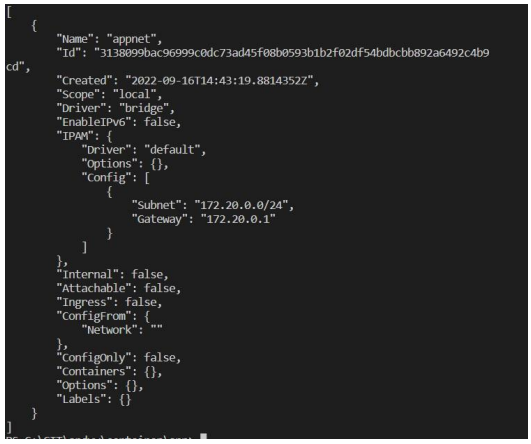
  docker build -t app .

- Create a new Docker network with the name appnet, the subnet 172.20.0.0/24, and the gateway 172.20.0.1. Use the command:

  docker network create --subnet=172.20.0.0/24 --gateway=172.20.0.1 appnet

- Check the network that you have created via the command:

  docker network inspect appnet command.



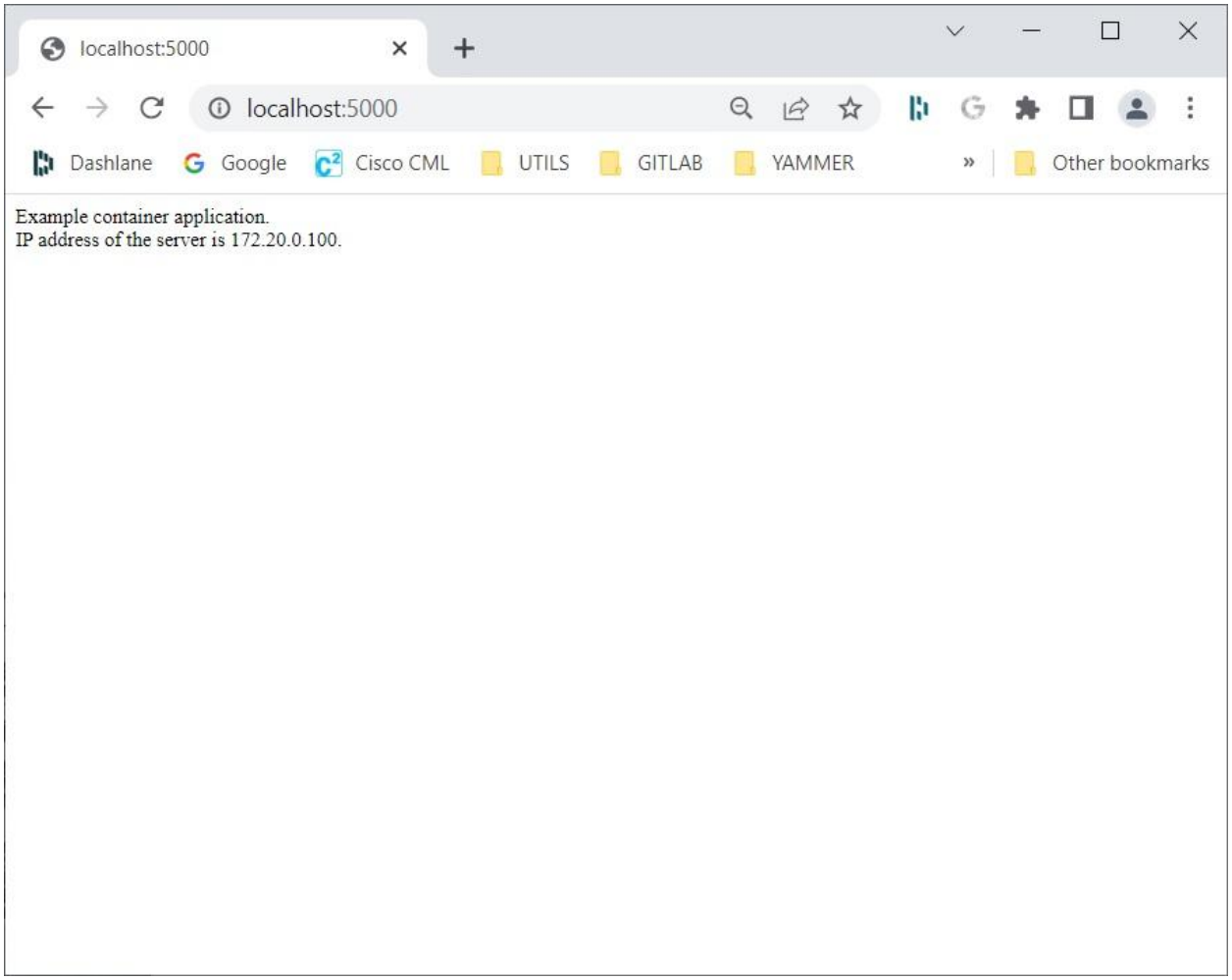
6. Run the app container using a specific network and IP address

- Use the command:

  docker run --net appnet --ip 172.20.0.100 -it -d -p 5000:5000 app

  to specify that you want to run the container on a particular network with a particular IP address.

- Open the web browser at address **http://localhost:5000** and verify the IP address of the Docker container in the response.

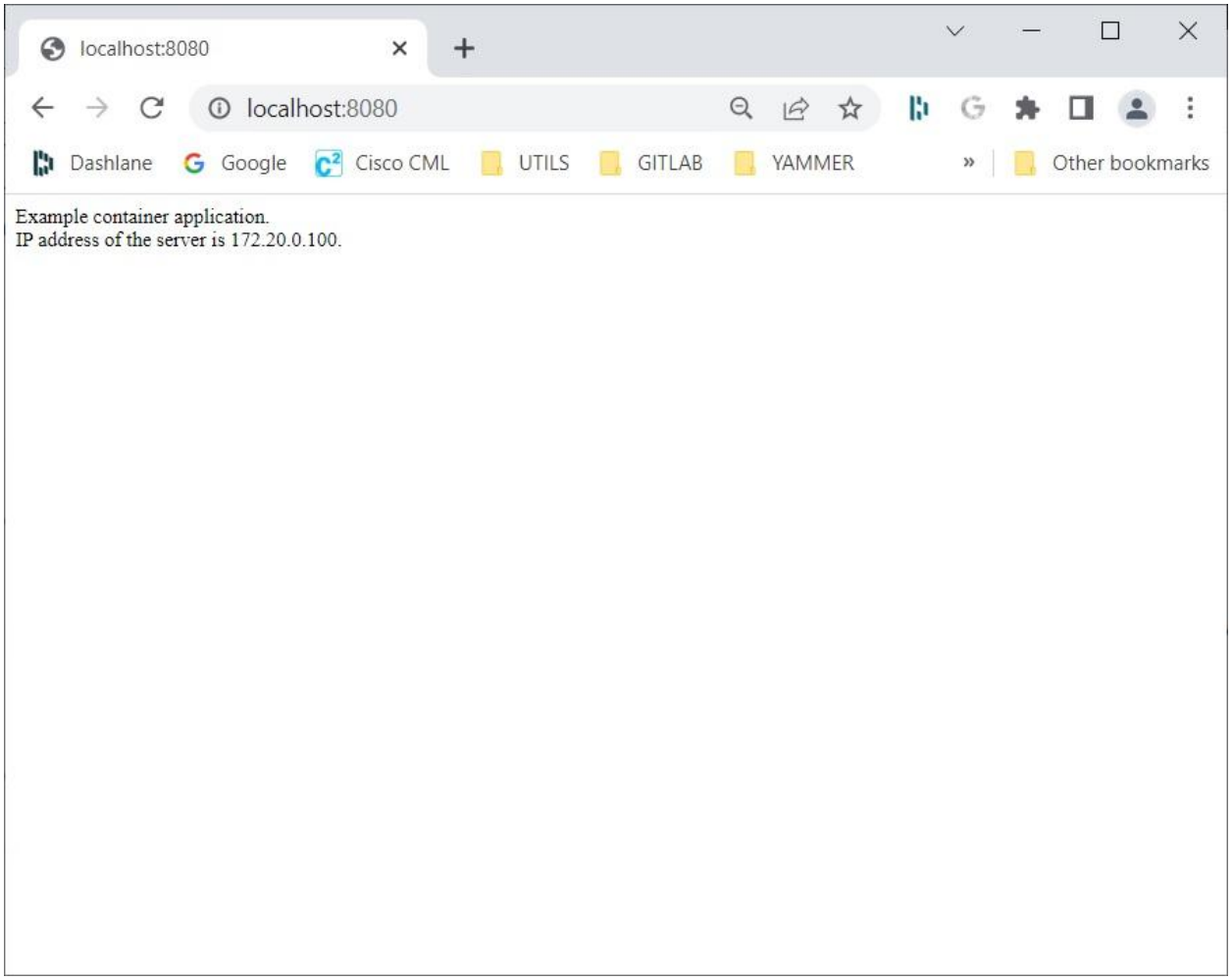


- Start the **lb** container. Use the command:

  docker run --net appnet --ip 172.20.0.10 -it -d -p 8080:8080 lb

- Open the web browser and access the **http://localhost:8080** URL. Verify that you see the same output as in the previous test, but this time

the request was done through the load balancer, confirming that the two containers were created correctly.



- Shut down both containers using the '**docker kill'** command. Use the hash IDs that you have received when the two containers were created.

- Use the command '**docker ps**' to obtain the ids of the containers if necessary.