Name of Student: Pushkar Sane		
Roll Number: 45	**Lab Assignment Number: 7**	
Title of Lab Assignment: To learn Docker file instructions, build an image for a sample web application using Docker file.		
DOP: 18-03-2024	**DOS: 03-04-2024**	
CO Mapped: **CO4**	**PO Mapped:** **PO2, PO3, PO5, PSO1, PSO2**	**Signature:**

Practical No. 7

Aim: To learn Docker file instructions, build an image for a sample web application using Docker file.

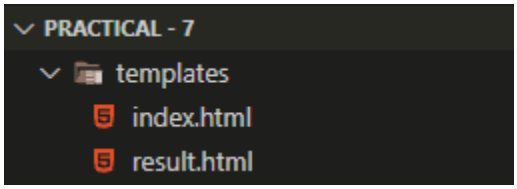
Introduction:

The following table contains the important Dockerfile instructions and their explanation.

Dockerfile Instruction	Explanation
FROM	To specify the base image that can be pulled from a container registry(Docker hub, GCR, Quay, ECR, etc.)
RUN	Executes commands during the image build process.
ENV	Sets environment variables inside the image. It will be available during build time as well as in a running container. If you want to set only build-time variables, use ARG instruction.
COPY	Copies local files and directories to the image
EXPOSE	Specifies the port to be exposed for the Docker container.
ADD	It is a more feature-rich version of the COPY instruction. It also allows copying from the URL that is the source and tar file auto-extraction into the image. However, usage of the COPY command is recommended over ADD. If you want to download remote files, use curl or get using RUN.
WORKDIR	Sets the current working directory. You can reuse this instruction in a Dockerfile to set a different working directory. If you set WORKDIR, instructions like RUN, CMD, ADD, COPY, or ENTRYPOINT get executed in that directory.
VOLUME	It is used to create or mount the volume to the Docker container.

USER	Sets the user name and UID when running the container. You can use this instruction to set a non-root user of the container.
LABEL	It is used to specify metadata information of Docker images.
ARG	Is used to set build-time variables with key and value. the ARG variables will not be available when the container is running. If you want to persist a variable on a running container, use ENV.
SHELL	This instruction is used to set shell options and default shell for the RUN, CMD, and ENTRYPOINT instructions that follow it.
CMD	It is used to execute a command in a running container. There can be only one CMD, if multiple CMDs then it only applies to the last one. It can be overridden from the Docker CLI.
ENTRYPOINT	Specifies the commands that will execute when the Docker container starts. If you don't specify any ENTRYPOINT, it defaults to /bin/sh -c. You can also override ENTRYPOINT using the --entrypoint flag using CLI. Please refer to CMD vs ENTRYPOINT for more information.

1. **Create a directory named "Webapp".Create New folder named as templates in the Webapp directory. And also create two files named as "index.html" and "result.html"**



Index.html

```
<!DOCTYPE html>
<html>
<head> <title>Calculator</title>
<style>
body {
```

```
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
}
h2 {
    color: #333;
}
form {
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    padding: 20px;
    max-width: 300px;
    margin: 20px auto;
}
label {
    display: block;
    margin-bottom: 10px;
    color: #666;
}
input[type="number"], input[type="submit"] {
    width: 100%;
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}
input[type="submit"] {
    background-color: #4CAF50;
    color: white;
    border: none;
```

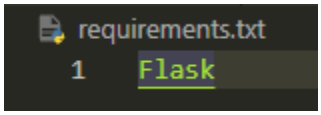
```
    cursor: pointer;
}
input[type="submit"]:hover {
    background-color: #45a049;
}
</style> </head>
<body>
<h2>Enter Three Numbers</h2>
<form action="/calculate" method="post">
    <label for="num1">Number 1:</label>
    <input type="number" name="num1" required><br>
    <label for="num2">Number 2:</label>
    <input type="number" name="num2" required><br>
    <label for="num3">Number 3:</label>
    <input type="number" name="num3" required><br>
    <input type="submit" value="Calculate">
</form>
</body> </html>
```

Result.html

```
<!DOCTYPE html>
<html>
<head> <title>Result</title>
<style>
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
}
h2 {
color: #333;}
p {
```

```
    margin-bottom: 10px;
}
a {
    color: #007bff;
    text-decoration: none;
}
a:hover {
    text-decoration: underline;
}
.result-container {
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    padding: 20px;
    max-width: 300px;
    margin: 20px auto;
}
</style>
</head>
<body>
<div class="result-container">
    <h2>Calculation Result</h2>
    <p>Total: {{ total }}</p>
    <p>Average: {{ average }}</p>
    <p>Product: {{ product }}</p>
    <a href="/">Back to Calculator</a>
</div>
</body> </html>
```

2. **Create a requirements.txt file in the "Webapp" directory not in templates folder. Mention all the required dependencies in this file. In my case I will only mention Flask.**



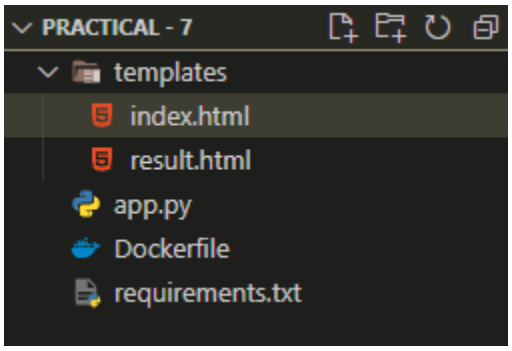
3. **Create the app.py file and write the logic behind the calculator webapp in it and also mention the webapp port.**

```
from flask import Flask, request, render_template
app = Flask(__name__)
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/calculate', methods=['POST'])
def calculate():
    # Get the numbers from the form
    num1 = float(request.form['num1'])
    num2 = float(request.form['num2'])
    num3 = float(request.form['num3'])
    # Perform calculations
    total = num1 + num2 + num3
    average = total / 3
    product = num1 * num2 * num3
    # Render the template with the result
        return render_template('result.html', total = total, average = average, product = product)
# Run the Flask app using the built-in development server
if __name__ == '__main__':
app.run(host='0.0.0.0', port=5000, debug=True)
```

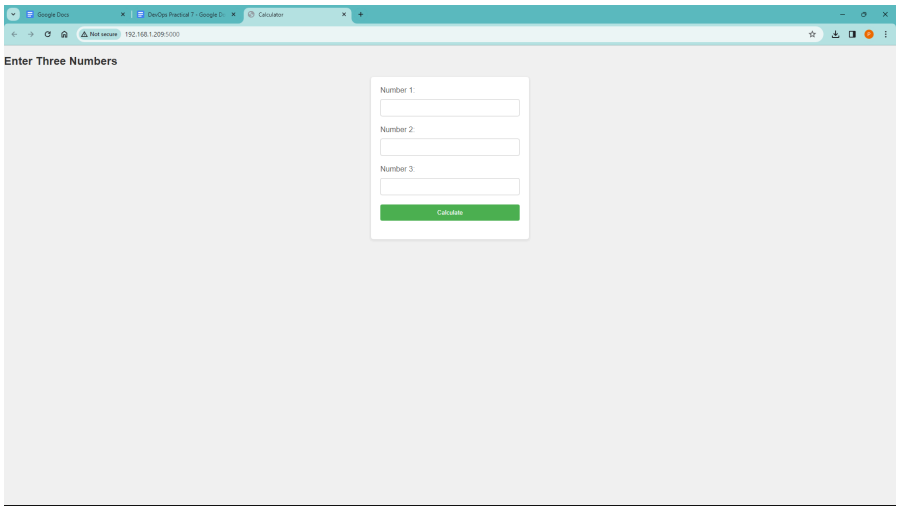
4. **Create a "Dockerfile" and mention which image to use and all the other commands to create an image and run the application on the web.**

```
Dockerfile
1   FROM python:3.8-slim
2   ENV PYTHONDONTWRITEBYTECODE 1
3   ENV PYTHONUNBUFFERED 1
4   WORKDIR /app
5   COPY requirements.txt /app/
6   RUN pip install --no-cache-dir -r requirements.txt
7   COPY . /app/
8   EXPOSE 5000
9   CMD ["python", "app.py"]
```

Make sure the file structure looks like below:



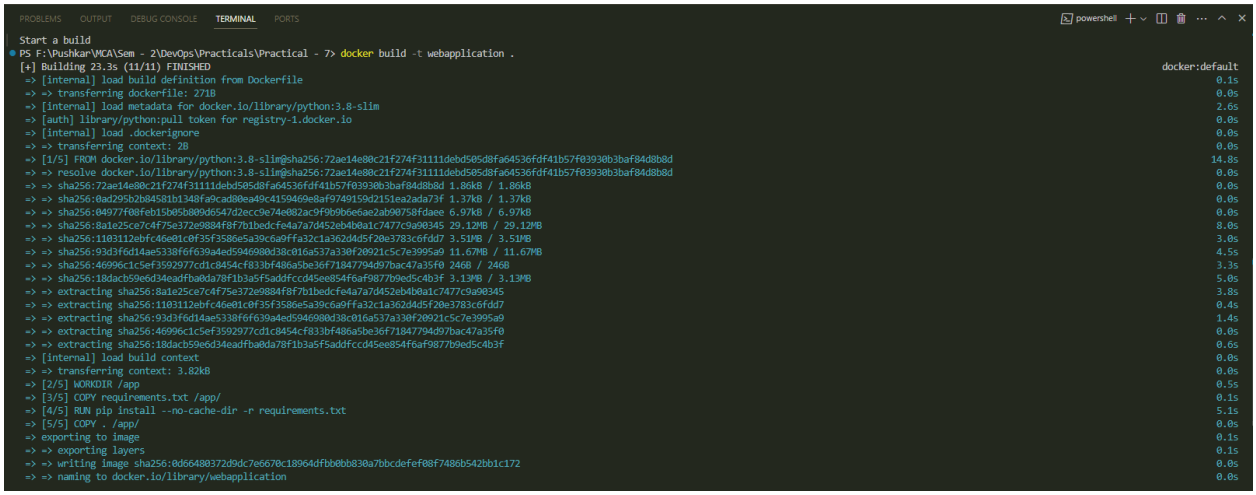
5. **Open the terminal in the VScode and try running the app.py file to check if the program is working as expected.**



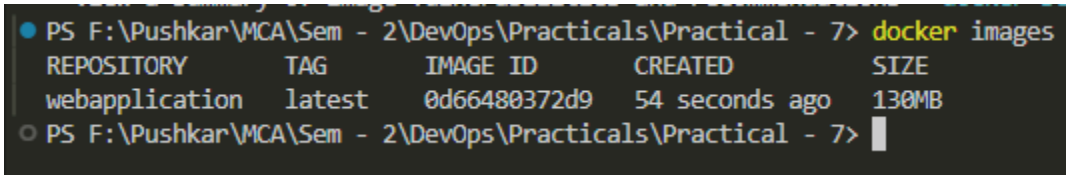
If the program is working as expected then proceed to build the docker image.

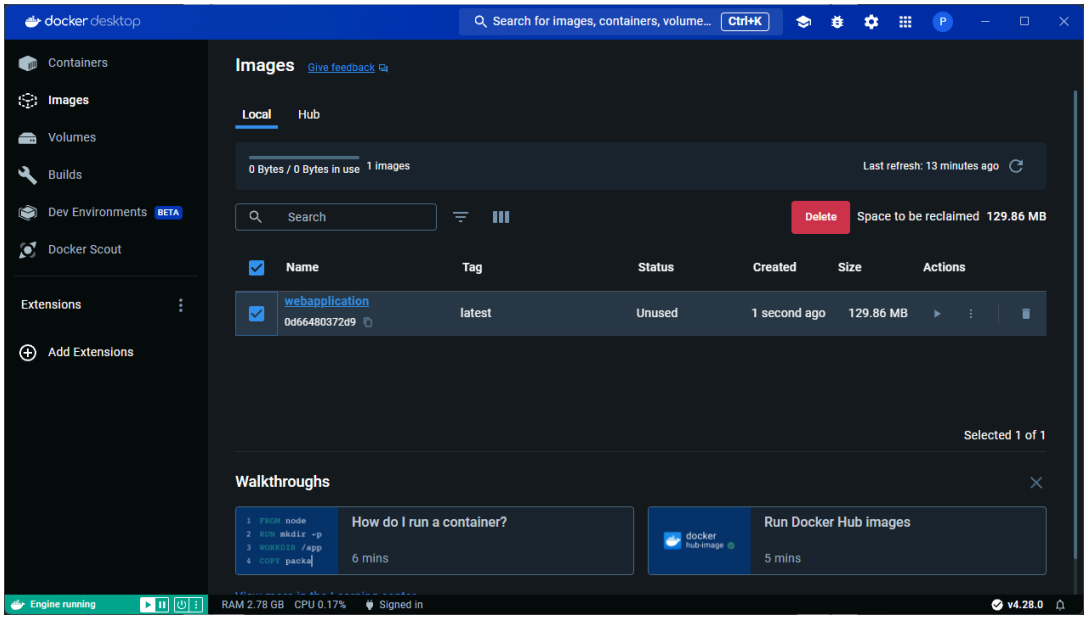
6. **Build the docker image using the “docker build -t <image_name_in_lower_case_> . “ command.**

Output should look like below after executing the command



Then use command “docker images” to check if the image has been created.

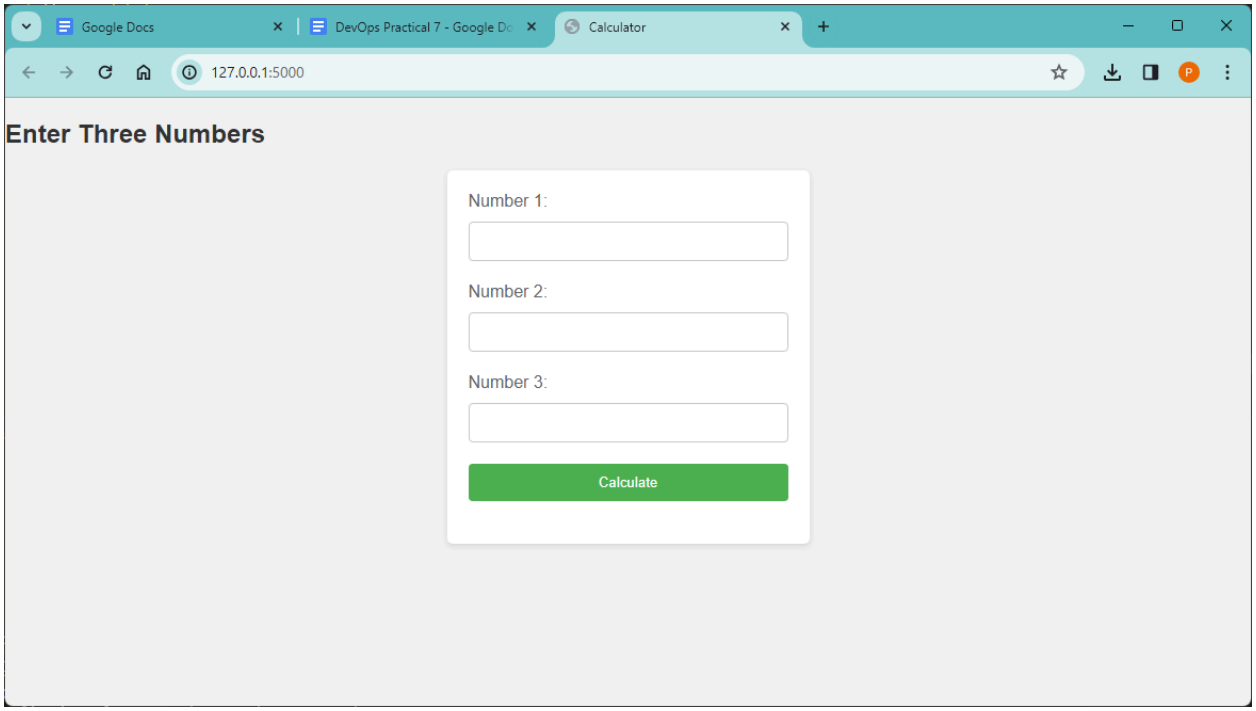


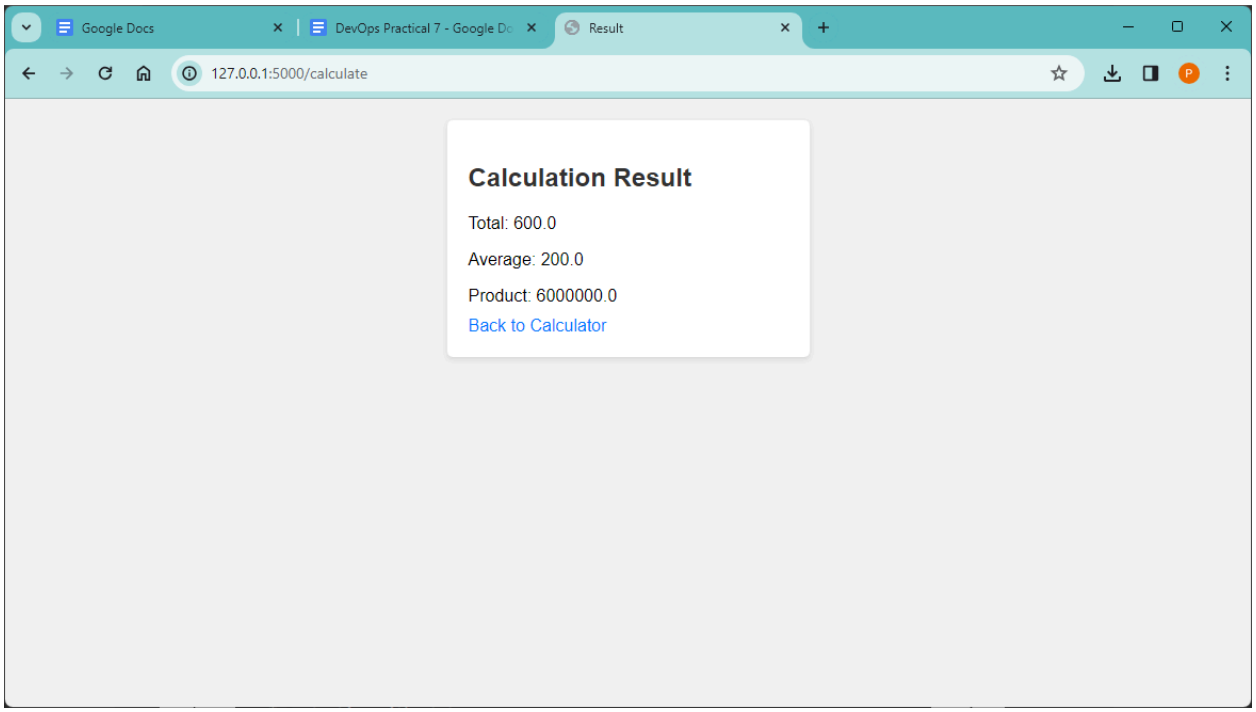


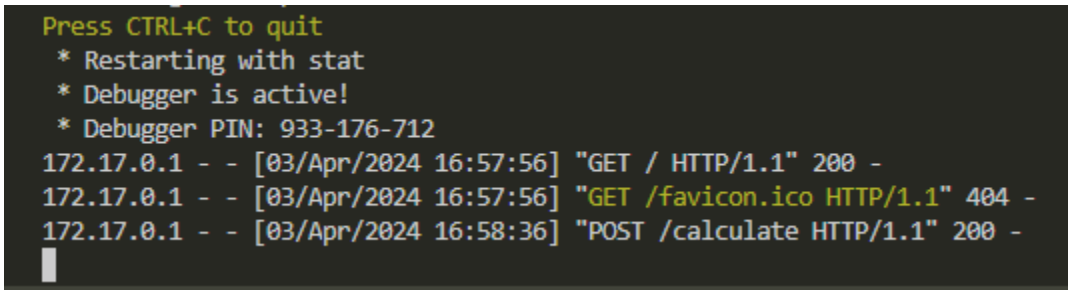
7. **Once the image has been successfully created then run the docker image on the web using the below command.**
   docker run -p 5000:5000 web application

Output:

```
PS F:\Pushkar\MCA\Sem - 2\DevOps\Practicals\Practical - 7> docker images
REPOSITORY       TAG       IMAGE ID       CREATED          SIZE
webapplication   latest    0d66480372d9   54 seconds ago   130MB
PS F:\Pushkar\MCA\Sem - 2\DevOps\Practicals\Practical - 7> docker run -p 5000:5000 webapplication
 * Serving Flask app 'app'
 * 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: 933-176-712
```







Running container with image “webapplication”.

Conclusion: In this lab, we've created a practical Dockerfile for a Python Flask web application, covering various Dockerfile instructions such as **FROM, RUN, COPY, EXPOSE, WORKDIR, ENV,** and **CMD**. This lab demonstrates the process of building a Docker image and running a container based on that image.