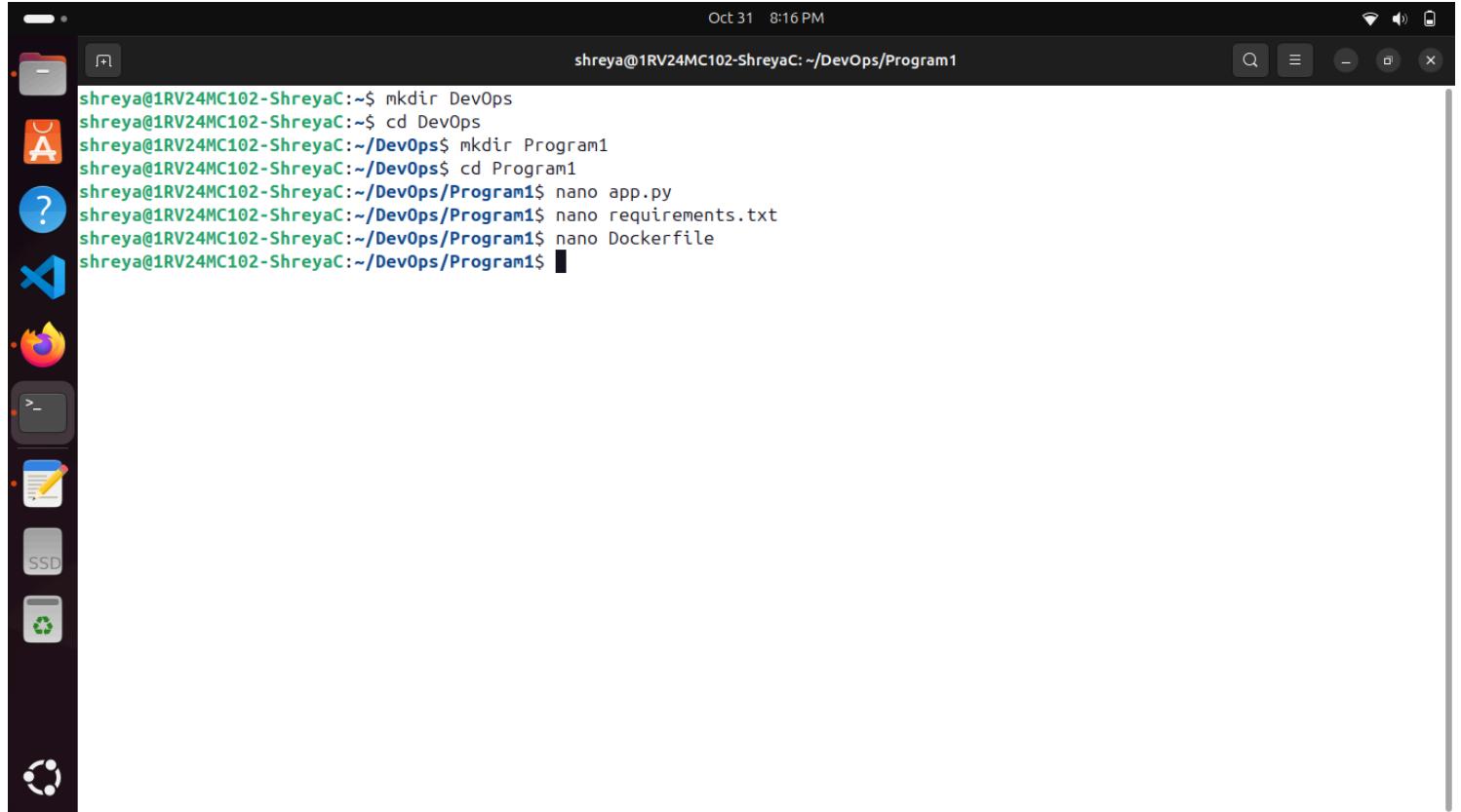


Program 1 : Build a Docker Container from a Custom Dockerfile

Folder Structure

```
DevOps
|-- Program1
    |-- app.py
    |-- requirements.txt
    |-- Dockerfile
```

Step 1: To build a docker container from custom Dockerfile, I've created a directory called DevOps navigate to DevOps directory and within that create a sub-directory with name Program1, navigate to Program1 directory and create the files such as [app.py](#), requirements.txt and Dockerfile.



The screenshot shows a Linux desktop environment with a terminal window open in the center. The terminal window has a dark background and displays the following command history:

```
shreya@1RV24MC102-ShreyaC:~$ mkdir DevOps
shreya@1RV24MC102-ShreyaC:~$ cd DevOps
shreya@1RV24MC102-ShreyaC:~/DevOps$ mkdir Program1
shreya@1RV24MC102-ShreyaC:~/DevOps$ cd Program1
shreya@1RV24MC102-ShreyaC:~/DevOps/Program1$ nano app.py
shreya@1RV24MC102-ShreyaC:~/DevOps/Program1$ nano requirements.txt
shreya@1RV24MC102-ShreyaC:~/DevOps/Program1$ nano Dockerfile
shreya@1RV24MC102-ShreyaC:~/DevOps/Program1$
```

To the left of the terminal window is a docked application bar containing icons for various desktop applications like a file manager, terminal, browser, and file browser.

Step 2: Add the following code in requirements.txt file, which is used to install the dependencies.

The screenshot shows a terminal window titled "shreya@1RV24MC102-ShreyaC: ~/DevOps/Program1". The file "requirements.txt" is open in the nano editor. The content of the file is "FLask". The terminal interface includes a menu bar at the top with "Oct 31 8:18 PM" and various icons. Below the menu bar is a toolbar with icons for Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Execute, Justify, Location, Go To Line, Undo, Redo, Set Mark, and Copy. The bottom of the window shows a status bar with the same set of keyboard shortcuts.

```
GNU nano 7.2 requirements.txt *
```

FLask

[New File]

[Read 10 lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-U Undo
M-A Set Mark ^M-E Redo M-6 Copy

Step 3: Code the following in [app.py](#) file, this code runs a simple web server that shows “Hello, Docker!” when we visit.

The screenshot shows a terminal window titled "shreya@1RV24MC102-ShreyaC: ~/DevOps/Program1". The file "app.py" is open in the nano editor. The content of the file is a Python script using the Flask framework to return the string "Hello, Docker!". The terminal interface is identical to the previous screenshot, with a menu bar, toolbar, and status bar.

```
GNU nano 7.2 app.py
```

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello, Docker!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

[Read 10 lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-U Undo
M-A Set Mark ^M-E Redo M-6 Copy

Step 4: Code the following in Dockerfile, it creates a lightweight container that installs Python and Flask dependencies, copies the code to /app and runs our Flask server on port 5000.

The screenshot shows a terminal window titled "shreya@1RV24MC102-ShreyaC: ~/DevOps/Program1". The file being edited is "Dockerfile". The content of the Dockerfile is as follows:

```
GNU nano 7.2
# This is a Dockerfile
# It builds a container image for a simple Flask app

# Use a slim Python base image
FROM python:3.9-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the requirements file and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application code
COPY . .

# Set Flask environment variables
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_RUN_PORT=5000

# Expose the port the Flask app will run on
EXPOSE 5000

# Define the command to run the Flask application
CMD ["flask", "run"]
```

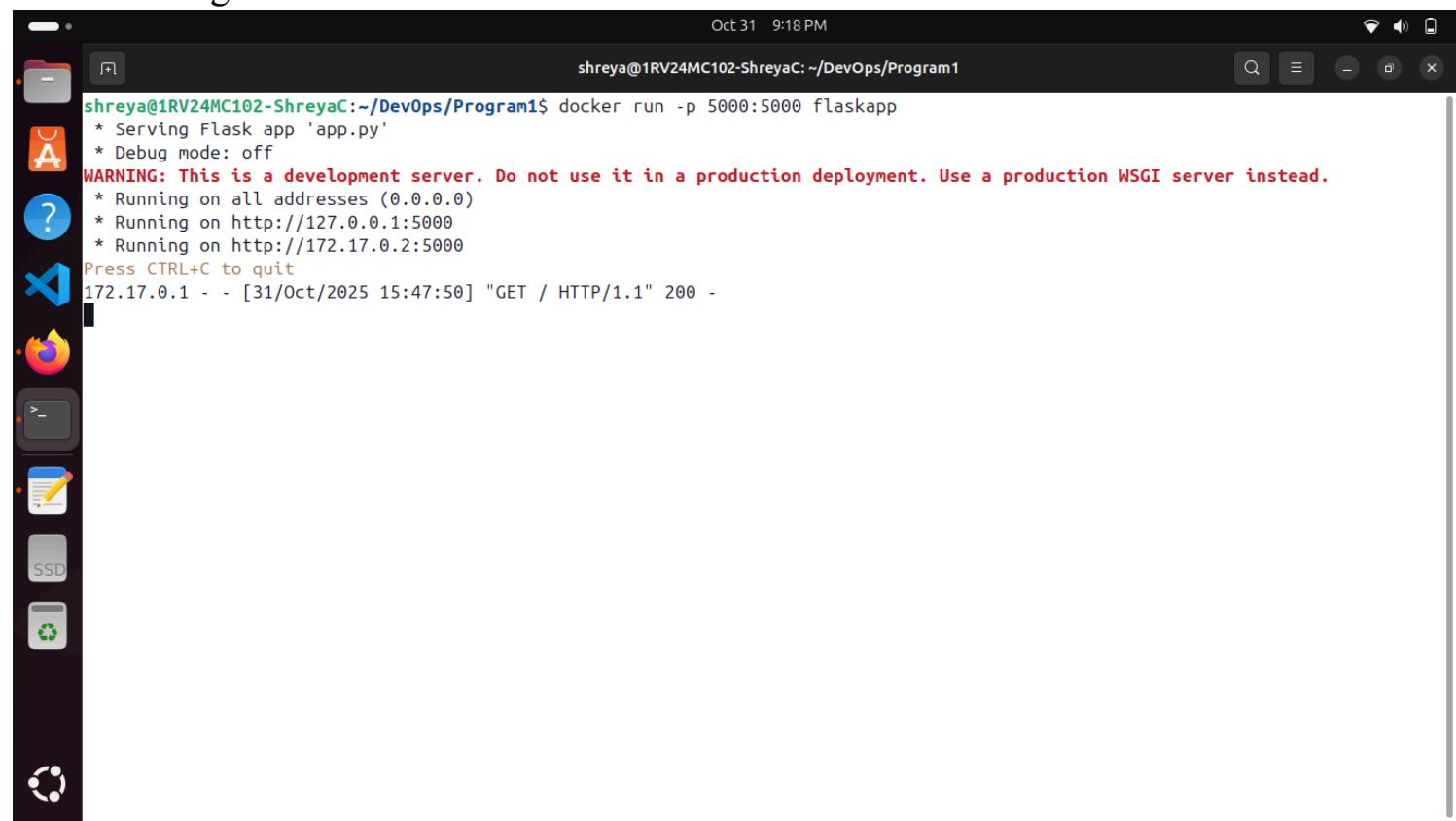
At the bottom of the terminal window, there are various keyboard shortcuts and status indicators. The status bar at the top right shows "Oct 31 9:12 PM".

Step 5: Now build the docker container using docker build command which tells the Docker to create new image using Dockerfile.

The screenshot shows a terminal window titled "shreya@1RV24MC102-ShreyaC: ~/DevOps/Program1\$". The command "docker build -t flaskapp ." is being run. The output shows the build process, including the creation of layers and the final image.

```
shreya@1RV24MC102-ShreyaC:~/DevOps/Program1$ docker build -t flaskapp .
[+] Building 15.5s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 635B
=> [internal] load metadata for docker.io/library/python:3.9-slim
=> [internal] load .dockernignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:545badebace9a958b98d3e272f0f0d46c0a1a389ac77e24c33f2e7b548ce1b6b
=> [internal] load build context
=> => transferring context: 895B
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY requirements.txt .
=> [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:d9ac790d65ab365bfbbe3fd096a3bdab65bd260fddc6c9cb795c6c8acc024b59
=> => naming to docker.io/library/flaskapp
shreya@1RV24MC102-ShreyaC:~/DevOps/Program1$
```

Step 6: Run the Docker container using docker run command, which creates the container from the image we built.



```
Oct 31 9:18 PM
shreya@1RV24MC102-ShreyaC:~/DevOps/Program1$ docker run -p 5000:5000 flaskapp
 * Serving Flask app 'app.py'
 * 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://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [31/Oct/2025 15:47:50] "GET / HTTP/1.1" 200 -
```

