

DevOps Lab

Program - 1

Build a Docker Container from a Custom Dockerfile

Project Structure

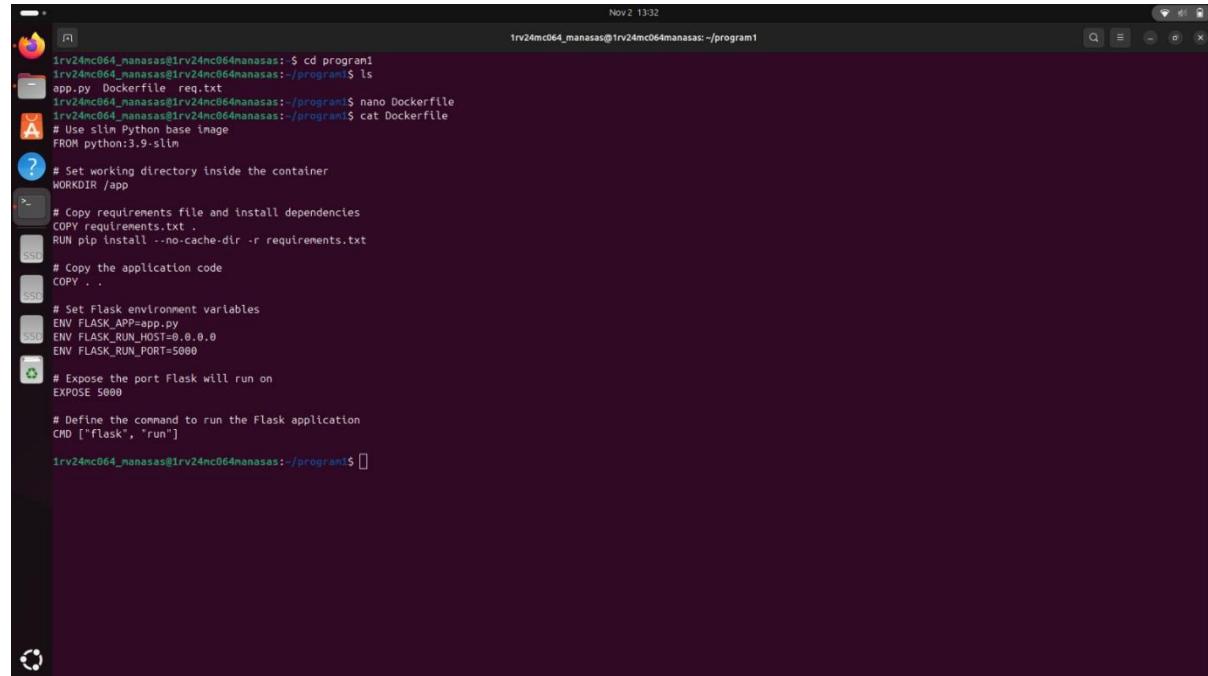
program1/

```
|  
|   └── Dockerfile  
|  
|   └── app.py  
└── requirements.txt
```

Step 1: Create Project Folder

```
mkdir program1
```

```
cd program1
```

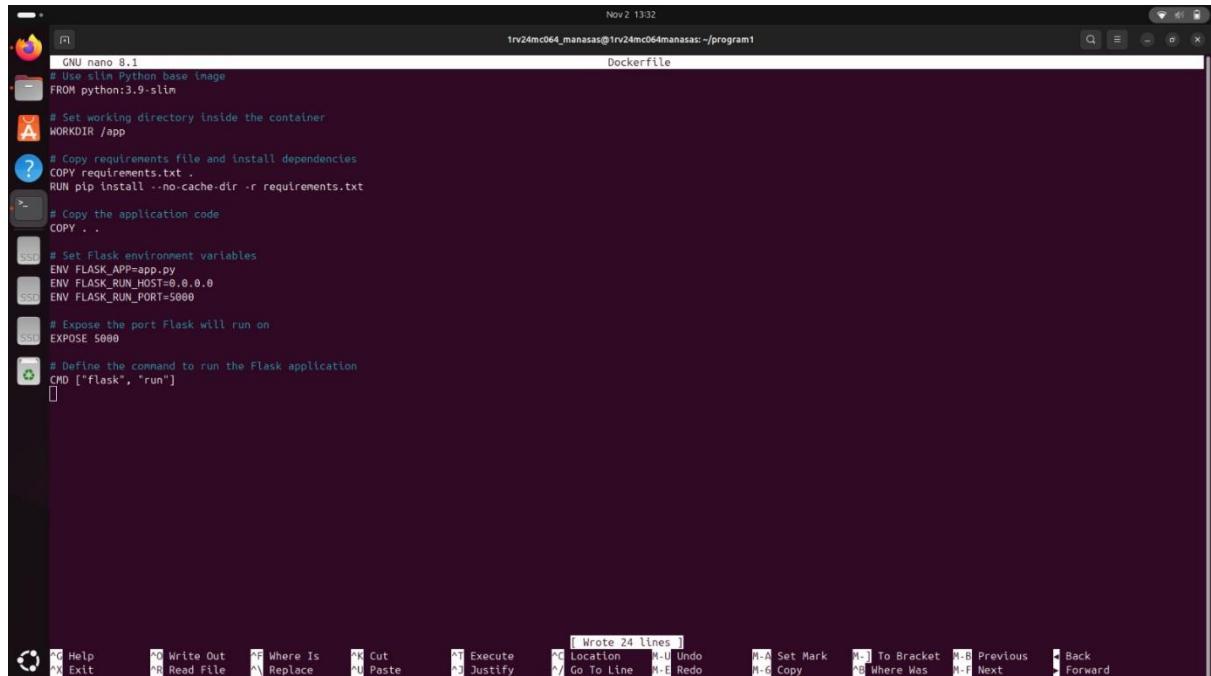


The screenshot shows a terminal window on a Linux system (Nov 2 13:32) with the command history:

```
irv24mc064_manasas@irv24mc064manasas: ~$ cd program1  
irv24mc064_manasas@irv24mc064manasas:~/program1$ ls  
app.py Dockerfile req.txt  
irv24mc064_manasas@irv24mc064manasas:~/program1$ nano Dockerfile  
irv24mc064_manasas@irv24mc064manasas:~/program1$ cat Dockerfile  
# Use slim Python base image  
FROM python:3.9-slim  
  
# Set working directory inside the container  
WORKDIR /app  
  
# Copy 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 Flask will run on  
EXPOSE 5000  
  
# Define the command to run the Flask application  
CMD ["flask", "run"]  
irv24mc064_manasas@irv24mc064manasas:~/program1$
```

Step 2: Create Dockerfile

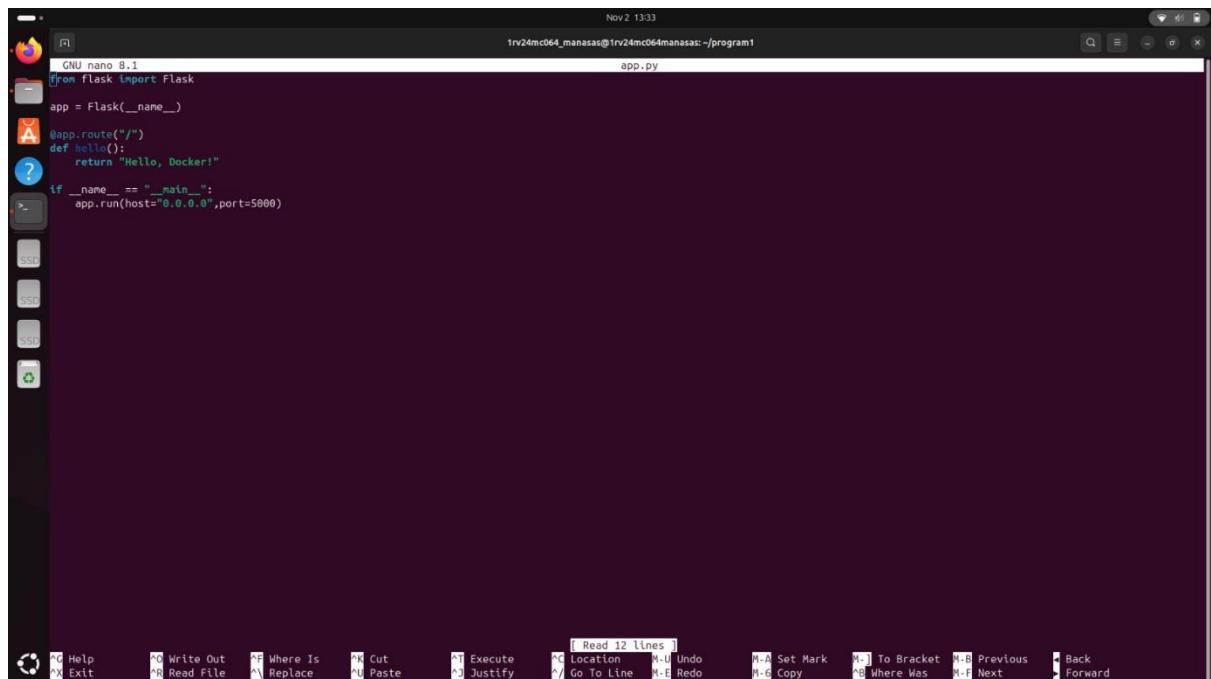
sudo nano Dockerfile



```
GNU nano 8.1
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY .
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_RUN_PORT=5000
EXPOSE 5000
CMD ["flask", "run"]
```

Step 3: Create Python Application File

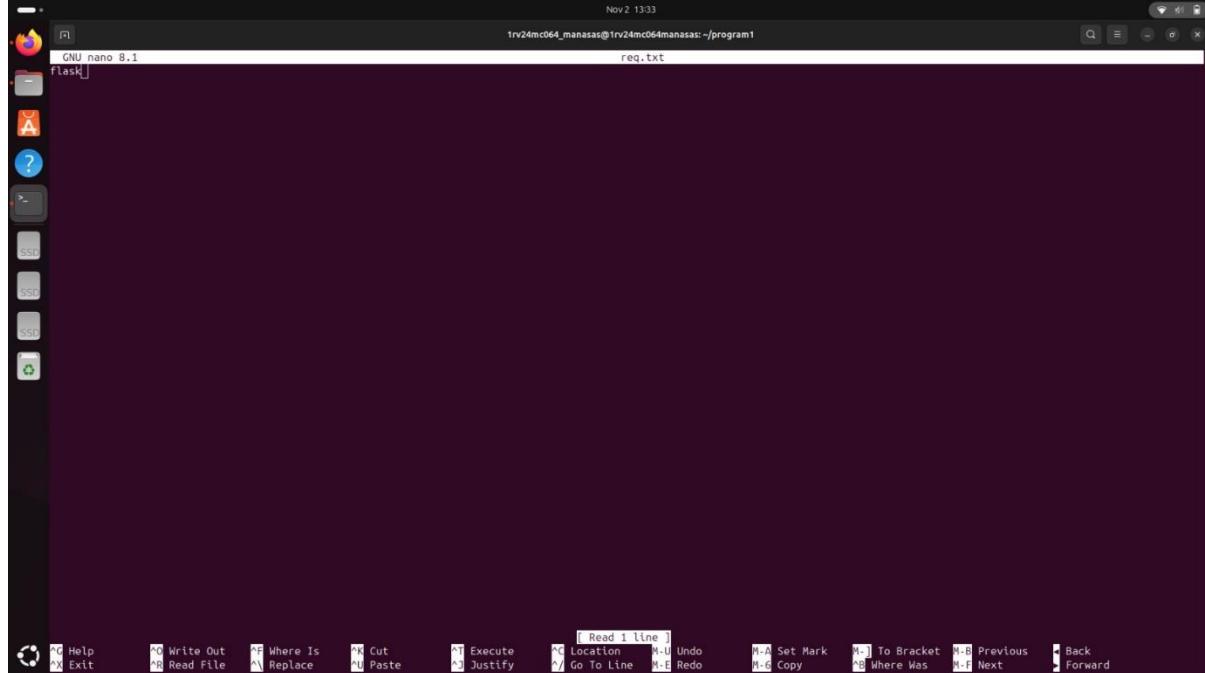
nano app.py



```
GNU nano 8.1
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)
```

Step 4: Create requirements.txt File

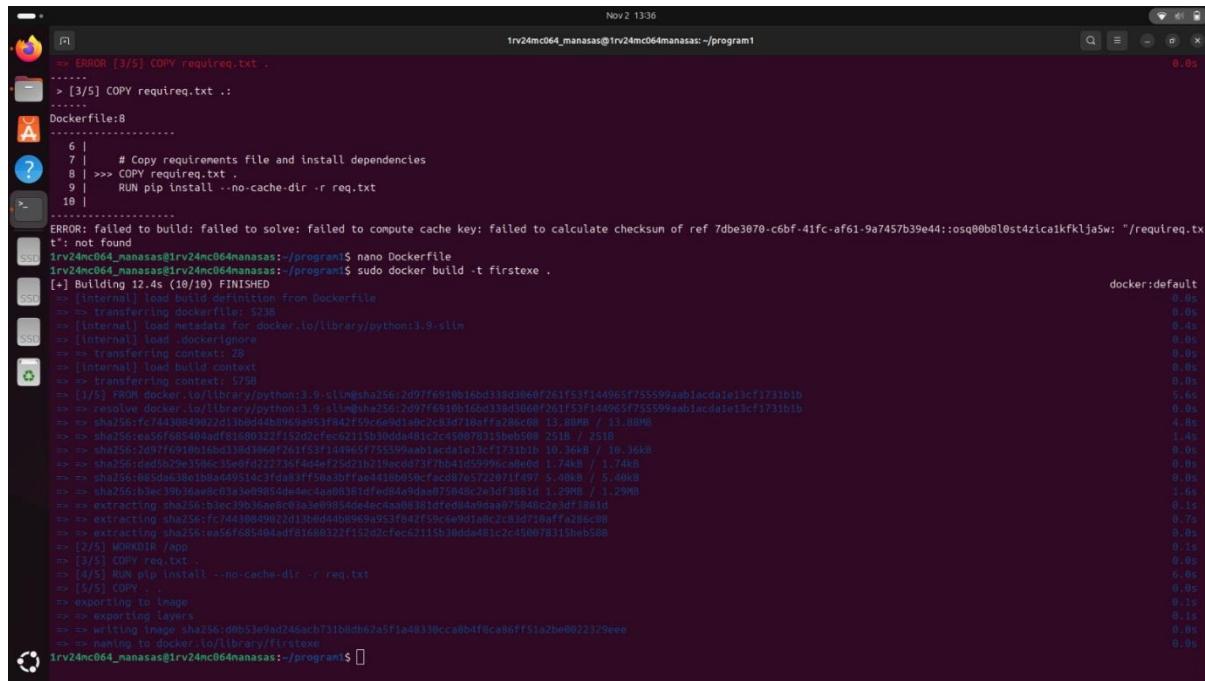
nano requirements.txt



The screenshot shows a terminal window titled "GNU nano 8.1" with the command "req.txt" in the title bar. The main area of the terminal is empty, indicating an empty file. The bottom of the window displays the standard nano editor key bindings.

Step 5: Build the Docker Image

Sudo docker build -t firstexe .



The screenshot shows a terminal window with the command "sudo docker build -t firstexe ." being run. The output shows the Dockerfile being read, the Docker image being built, and the resulting Docker container being run. The Docker container's logs show the contents of the requirements.txt file and the execution of the application.

Step 6: Run the Docker Container

docker run -d -p 5000:5000 program_1

Step 7: Verify the Application

Open your browser and go to: <http://localhost:5000>

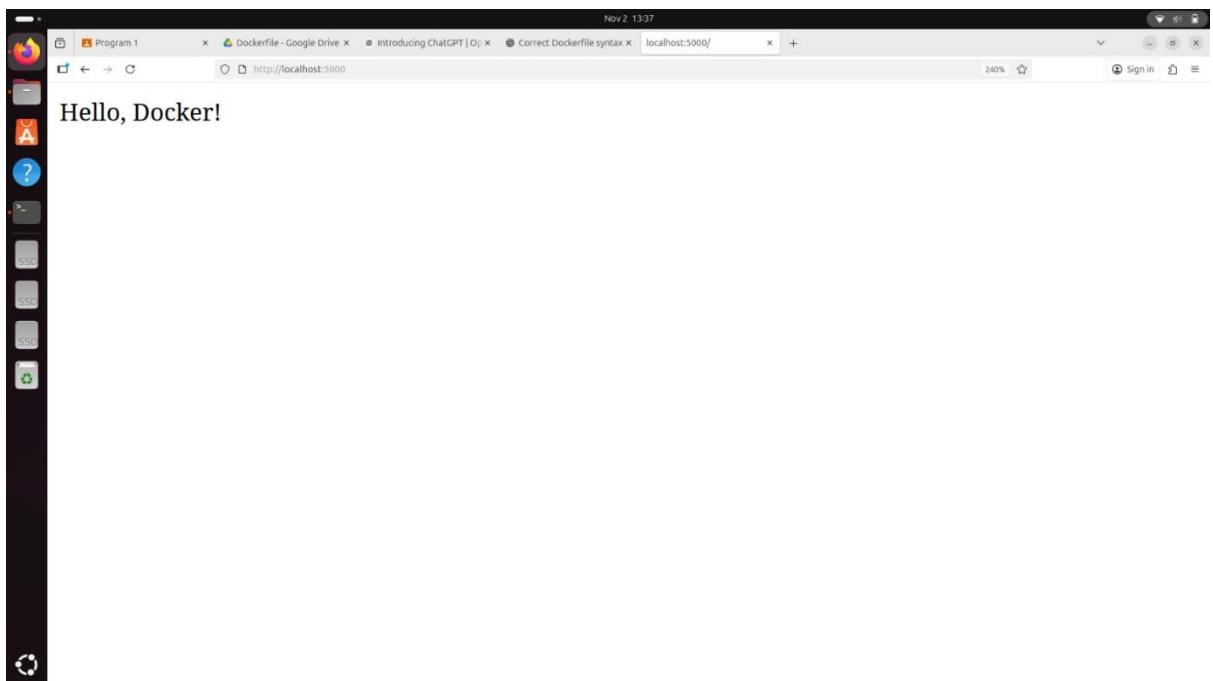
```

Nov 2 13:37
1rv24mc064_manasas@1rv24mc064manasas:~/program1
1rv24mc064_manasas@1rv24mc064manasas:~/program1
1rv24mc064_manasas@1rv24mc064manasas:~/program1

.....
Dockerfile:8
6 | 
7 |     # Copy requirements file and install dependencies
8 | >>> COPY req.txt .
9 |     RUN pip install --no-cache-dir -r req.txt
10 |

> ERROR: failed to build: failed to solve: failed to compute cache key: failed to calculate checksum of ref 7dbe3870-c6bf-41fc-af61-9a7457b39e44:osq00b8l0st4zicaikfk1ja5w: "/req.txt": not found
1rv24mc064_manasas@1rv24mc064manasas:~/program1$ nano Dockerfile
1rv24mc064_manasas@1rv24mc064manasas:~/program1$ sudo docker build -t firstexe .
[+] Building 12.4s (10/10) FINISHED
   == [internal] load build definition from Dockerfile
   == > transferring Dockerfile: 52B
   == [internal] load metadata for docker.libary/python:3.9-slim
   == [internal] load .dockerignore
   == > transferring context: 2B
   == [internal] load build context
   == > transferring context: 57B
   == [internal] FDRM docker .lo/19f94ry/python:3.9-slim@sha256:2997f6910b166d338d3866f7261f53f144965f755599aab1acda1e13cf1731b1b
   == > resolve docker .lo/19f94ry/python:3.9-slim@sha256:2997f6910b166d338d3866f7261f53f144965f755599aab1acda1e13cf1731b1b
   == > sha256:fc781f084922110b0d4b89996953f842f59c6e9d1a9c2cb3d7198ff280c98 13,808B / 13,808B
   == > sha256:e056f685494ad781689322f152dc7ec62115b30d0481cc49087815b60598 251B
   == > sha256:209776708106564318d396e291f53f144965f755599aab1acda1e13cf1731b1b 18,364B / 18,364B
   == > sha256:68051282595956507d22f3f74cd8f7921b193d74d9999c80090 1,748B / 1,748B
   == > sha256:885d9636e10a9e5143f5a3f758a30ff7ae4108959c7c2071f497 5,408B / 5,408B
   == > sha256:03ec19m10aepc33a709954d4-eaa018-df6ed49d4a075040c2a3f8891d 1,290B / 1,290B
   == > extracting sha256:03ec19m10aepc33a709954d4-eaa018-df6ed49d4a075040c2a3f8891d
   == > extracting sha256:fc7443984992213b6d4d08969a953f842f59c6e9d1aefcfc83d0718affa286c88
   == > extracting sha256:e056f685494ad781689322f152dc7ec62115b30d0481cc450078315be0580
[2/3] WORKDIR /app
[3/3] COPY req.txt .
[2/3] RUN pip install --no-cache-dir -r req.txt
[3/3] COPY .
[4/4] exporting to image
[4/4] writing manifest to ./firstexe
[4/4] saving to docker.lo/11b8ry/firstexe
1rv24mc064_manasas@1rv24mc064manasas:~/program1$ sudo docker run -d -p 5000:5000 firstexe
89f239a207f036c1a0fd6e1fb65429689c95684be4ab065e8674f6384b4f71e
1rv24mc064_manasas@1rv24mc064manasas:~/program1

```



Step 8: Verify Running Container

docker ps