

Program 2: Develop a Multi-Stage Dockerfile for Container Orchestration.

Folder Structure:

```
DevOps
|-- Program2
    |-- build
    |-- dist
        |-- index.js
    |-- src
    |-- Dockerfile
    |-- package.json
    |-- package-lock.json
    |-- node_modules
```

Step1 : I've created a new folder called Program2 to hold Node.js project, navigated inside it, and made three subfolders build, dist, and src to organize the code. Then I've opened the dist folder and created a file named index.js where the main server code will go. Also, created a file name Dockerfile which installs the dependencies. After that, navigated back to the main project folder and ran npm init -y to quickly create a default package.json file that keeps project info and dependencies. Finally, installed Express, a popular Node.js framework, so that we can easily build and run web server.

```
Oct 31 9:36 PM
shreya@1RV24MC102-ShreyaC: ~/DevOps/Program2

shreya@1RV24MC102-ShreyaC:~/DevOps$ mkdir Program2
shreya@1RV24MC102-ShreyaC:~/DevOps$ cd Program2
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$ mkdir build
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$ mkdir dist
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$ cd dist
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2/dist$ nano index.js
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2/dist$ cd ..
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$ mkdir src
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$ npm init -y
Wrote to /home/shreya/DevOps/Program2/package.json:

{
  "name": "program2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$ npm install express

added 68 packages, and audited 69 packages in 2s

16 packages are looking for funding
  run `npm fund` for details
```

Step 2: This code creates a simple web server using Express that runs on port 8000. When we open <http://localhost:8000>

```
Oct 31 9:40 PM
shreya@1RV24MC102-ShreyaC: ~/DevOps/Program2/dist

GNU nano 7.2 index.js
const express = require('express');
const app = express();
const port = 8000;

app.get('/', (req, res) => {
  res.send('Hello from Multi-Stage Dockerfile');
});

app.listen(port, () => {
  console.log(`Server running on port http://localhost:${port}`);
});

[ Read 13 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/_ Go To Line M-E Redo      M-G Copy
```

Step 3: This Dockerfile creates a lightweight Node.js container for our app. It starts with the Node 20 Alpine image, sets /app as the working folder, installs dependencies from package.json, copies the rest of our files, opens port 5000 for access, and finally runs the app using node dist/index.js.



The screenshot shows a terminal window with the nano text editor open, editing a file named 'Dockerfile'. The editor's title bar indicates the user is 'shreya@1RV24MC102-ShreyaC' in the directory '~/DevOps/Program2/dist'. The Dockerfile content is as follows:

```
GNU nano 7.2 Dockerfile
# Use official Node.js 20 Alpine image
FROM node:20-alpine

# Set working directory
WORKDIR /app

# Copy package files and install dependencies
COPY package.json package-lock.json ./
RUN npm install

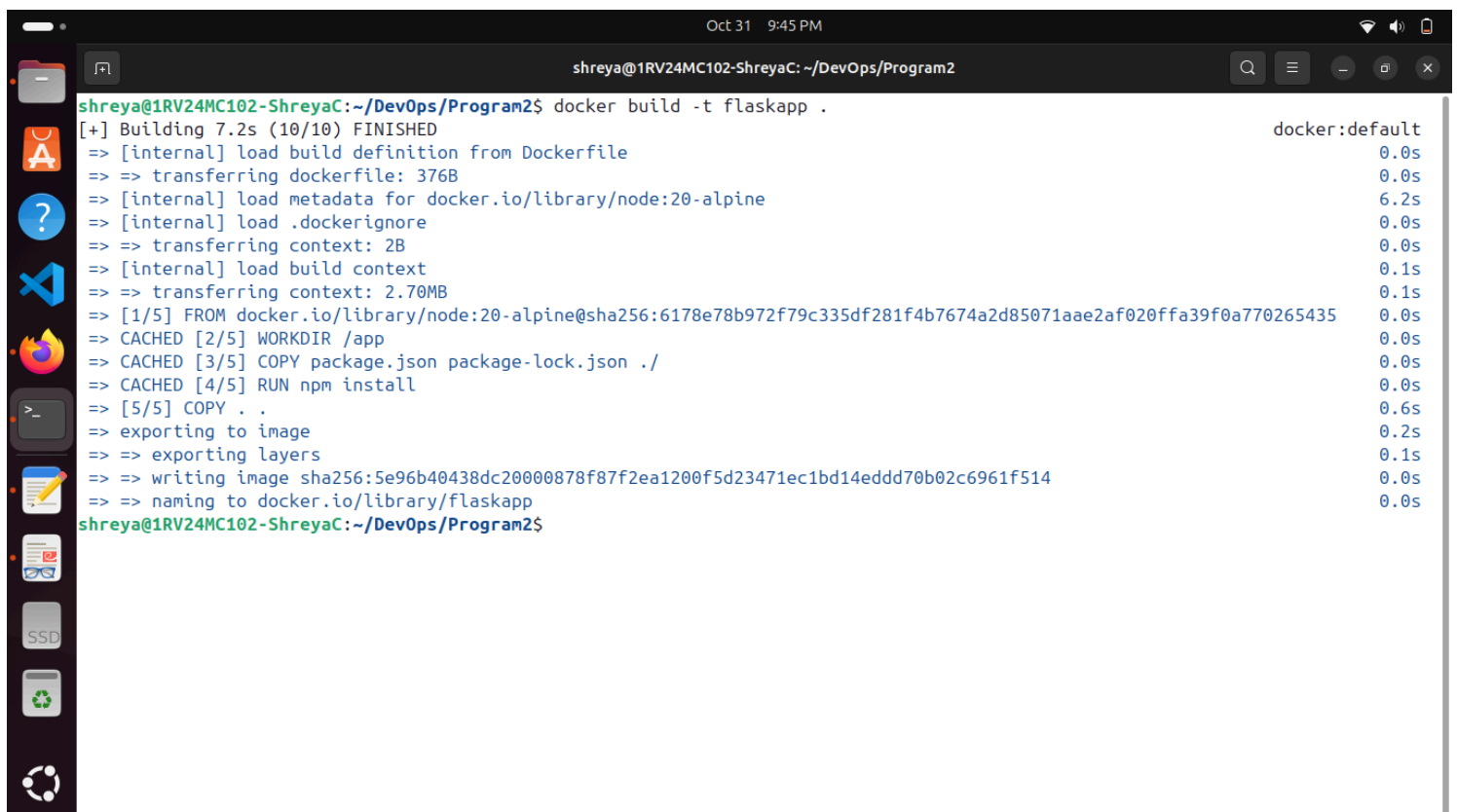
# Copy everything else
COPY . .

# Expose the application port
EXPOSE 5000

# Start the built application
CMD ["node", "dist/index.js"]
```

At the bottom of the terminal, a status bar displays various keyboard shortcuts: ^G Help, ^X Exit, ^O Write Out, ^R Read File, ^W Where Is, ^\ Replace, ^K Cut, ^U Paste, ^T Execute, ^J Justify, ^C Location, ^_ Go To Line, M-U Undo, M-E Redo, M-A Set Mark, and M-6 Copy. A message '[Read 19 lines]' is also visible.

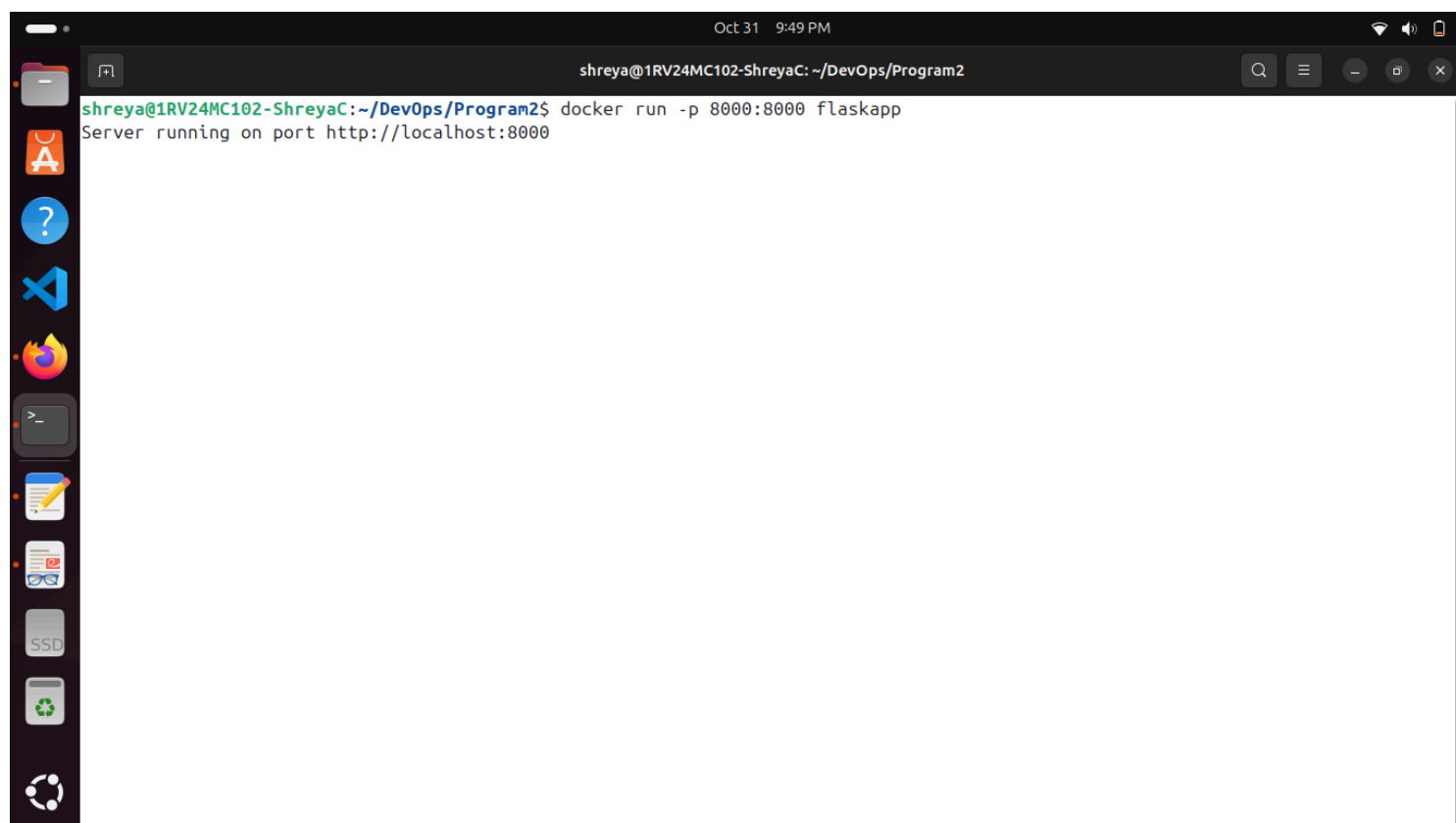
Step 4: Now build the docker container using docker build command which tells the Docker to create new image using Dockerfile.



The screenshot shows a terminal window where the command 'docker build -t flaskapp .' has been executed. The output shows the progress of building the Docker image, including downloading the base image and installing dependencies. The final output is:

```
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$ docker build -t flaskapp .
[+] Building 7.2s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 376B
=> [internal] load metadata for docker.io/library/node:20-alpine
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 2.70MB
=> [1/5] FROM docker.io/library/node:20-alpine@sha256:6178e78b972f79c335df281f4b7674a2d85071aae2af020ffa39f0a770265435
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY package.json package-lock.json ./
=> CACHED [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:5e96b40438dc20000878f87f2ea1200f5d23471ec1bd14eddd70b02c6961f514
=> => naming to docker.io/library/flaskapp
shreya@1RV24MC102-ShreyaC:~/DevOps/Program2$
```

Step 5: Run the Docker container using docker run command, which creates the container from the image we built.



A terminal window titled "shreya@1RV24MC102-ShreyaC: ~/DevOps/Program2" showing the command `docker run -p 8000:8000 flaskapp` and its output: `Server running on port http://localhost:8000`. The terminal is part of a desktop environment with a sidebar containing icons for various applications like a file manager, terminal, and web browser.

```
shreya@1RV24MC102-ShreyaC: ~/DevOps/Program2$ docker run -p 8000:8000 flaskapp
Server running on port http://localhost:8000
```

