

DevOps Lab

Program 2- Creating a Multi-Stage Dockerfile

Project Structure:

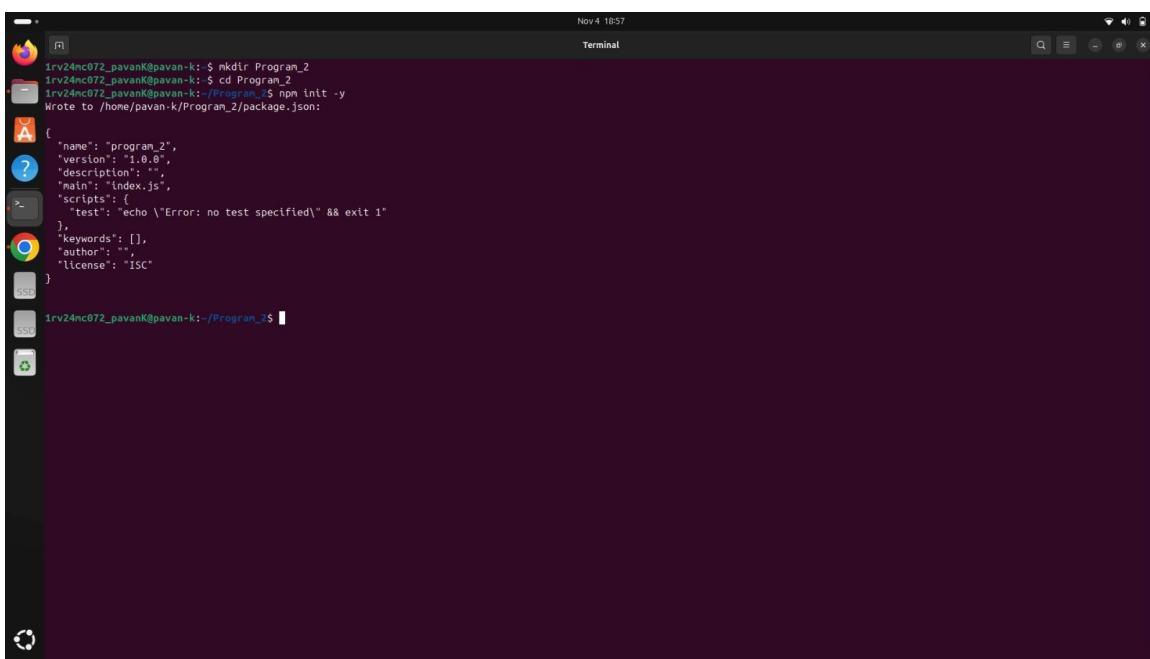
```
program_2
  Dockerfile
  package.json
  package-lock.json
  node_modules/
  src/
    index.js
```

Step 1: Create Project Folder

```
mkdir program_2
cd program_2
```

Step 2: Initialize Node.js Project

```
npm init -y
```



The screenshot shows a terminal window on a Linux desktop environment. The terminal title is "Terminal". The session log is as follows:

```
Nov 4 16:57
irv24nc072_pavanK@pavan-k: $ mkdir Program_2
irv24nc072_pavanK@pavan-k: $ cd Program_2
irv24nc072_pavanK@pavan-k:~/Program_2$ npm init -y
Wrote to /home/pavan-k/Program_2/package.json:

  {
    "name": "program_2",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
      "test": "echo \\\"Error: no test specified\\\" && exit 1"
    },
    "keywords": [],
    "author": "",
    "license": "ISC"
  }

irv24nc072_pavanK@pavan-k:~/Program_2$
```

Step 3: Install Express Framework

```
npm install express
```

```
Nov 4 16:57
Terminal
irv24mc072_pavanK@pavan-k: ~$ mkdir Program_2
irv24mc072_pavanK@pavan-k: ~$ cd Program_2
irv24mc072_pavanK@pavan-k: ~/Program_2$ npm init -y
Wrote to /home/pavan-k/Program_2/package.json:

{
  "name": "program_2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

SSD
irv24mc072_pavanK@pavan-k:~/Program_2$ npm install express
added 68 packages, and audited 69 packages in 14s
16 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
irv24mc072_pavanK@pavan-k:~/Program_2$
```

Step 4: Create Source Folder and Application File

```
mkdir src
```

```
cd src
```

```
nano index.js
```

```
Nov 4 16:58
Terminal
GNU nano 7.2
index.js
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
  res.send('Hello from multi-stage Docker!');
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Step 5: Create the Multi-Stage Dockerfile

```
cd ..
```

```
nano Dockerfile
```

```
Nov 4 19:01
GNU nano 7.2
Stage 1: Build Stage
FROM node:20-alpine AS builder
# Set working directory
WORKDIR /app

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

# Copy application source code
COPY .

# Build the application (for our simple app, this just copies src to dist)
RUN npm run build

# Stage 2: Production Stage
FROM node:20-alpine
# Set working directory
WORKDIR /app

# Copy only necessary files from build stage
COPY --from=builder /app/package.json .
COPY --from=builder /app/package-lock.json .
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules

# Expose port
EXPOSE 3000

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

The screenshot shows a terminal window with a dark theme. The title bar says "Terminal" and "DockerFile". The main area contains a Dockerfile with various commands like FROM, COPY, RUN, and CMD. Icons for file operations (copy, paste, cut, etc.) are placed next to the corresponding Dockerfile commands. The bottom of the window has a toolbar with icons for Help, Exit, Write Out, Read File, Replace, Cut, Paste, Execute, Justify, Location, Undo, Redo, Set Mark, To Bracket, Copy, Go To Line, Where Was, Previous, Next, Back, and Forward. A status bar at the bottom shows keyboard shortcuts for these functions.

Step 6: Build the Docker Image

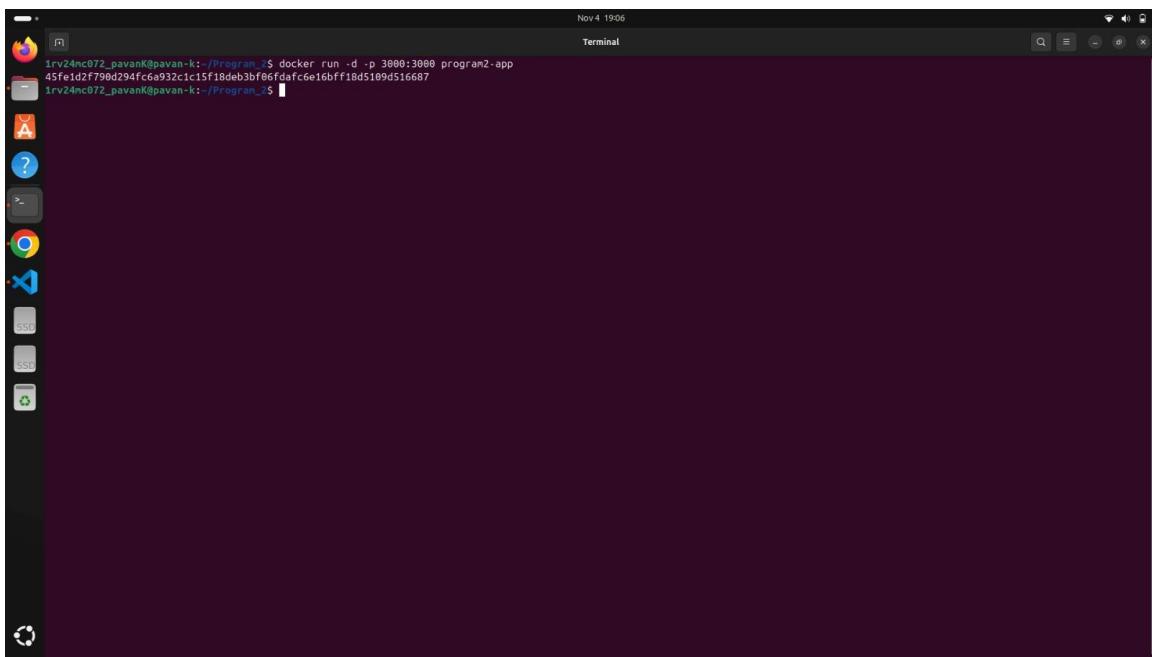
```
docker build -t program2-app .
```

```
Nov 4 19:05
irv24mc072_pavanK@pavan-k:~/Program . $ docker build -t program2-app .
[+] Building 8.4s (15/15) FINISHED
   ==> [internal] load build definition from Dockerfile
   ==> [internal] transfer Dockerfile: 752B
   ==> [internal] load metadata for docker.io/library/node:20-alpine
   ==> [internal] load .dockerignore
   ==> [internal] transfer context: 28B
   ==> [internal] load build context
   ==> [internal] transfer context: 43.86kB
   ==> CACHED [builder 2/6] WORKDIR /app
   ==> [builder 3/6] COPY package.json package-lock.json .
   ==> [builder 4/6] RUN npm install
   ==> [builder 5/6] RUN npm run build
   ==> [builder 6/6] RUN npm run build
   ==> [stage-1 3/6] COPY --from=builder /app/package.json .
   ==> [stage-1 4/6] COPY --from=builder /app/package-lock.json .
   ==> [stage-1 5/6] COPY --from=builder /app/dist ./dist
   ==> [stage-1 6/6] COPY --from=builder /app/node_modules ./node_modules
   ==> exporting image
   ==> writing manifest to docker.io/library/program2-app
   ==> naming to docker.io/library/program2-app
1rv24mc072_pavanK@pavan-k:~/Program . $
```

This screenshot shows the terminal output of the "docker build" command. It displays the progress of the build process, including the creation of a builder image, copying files, installing dependencies, and finally building the final image. The output is timestamped and includes the Dockerfile path and the resulting image name.

Step 7: Run the Docker Container

```
docker run -d -p 3000:3000 program2-app
```



A screenshot of a Linux desktop environment. On the left is a dark vertical dock containing icons for various applications like a file manager, terminal, browser, and file system. The main window is a terminal titled "Terminal" with the status bar showing "Nov 4 19:06". The terminal window contains the following text:

```
1rv24mc072_pavanK@pavan-k:~/Program_1$ docker run -d -p 3080:3080 program2-app
45fe1d2f790d294fc6a932c1c15f18deb3bf0efdfc6e16bff18d5109d516687
1rv24mc072_pavanK@pavan-k:~/Program_1$
```

Step 8: Verify the Application

Open <http://localhost:3000>

