DevOps – Lab Program-2 Documentation

Title:

Develop a Multi-Stage Dockerfile for Node.js Application

Objective:

To build and deploy a Node.js web application using a multi-stage Dockerfile, optimizing image size and enabling efficient container orchestration.

Software Requirements:

- Docker
- Node.js
- Express.js
- VS Code / Terminal

Files and Folder Structure:

```
SecondProgram/
│
├── node_modules/
├── src/
│   └── index.js
├── package.json
├── package-lock.json
└── Dockerfile
```

Source Code:

index.js

```
const express = require('express');
const app = express();
const port = 2000;

app.get('/', (req, res) => {
   res.send(`
       <script>alert("Hello from the docker file")</script>
   `);
});
```

```
app.listen(port, () => {
    console.log("running");
});
```

package.json

```
{
  "name": "secondprogram",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "build": "echo \"Building the project...\" && mkdir -p dist && cp -r src/*
dist/",
    "start": "node dist/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "dependencies": {
    "express": "^5.1.0"
  }
}
```

Dockerfile

```
# Stage 1: Build stage
FROM node:20-alpine AS builder
WORKDIR /app
COPY package.json ./
RUN npm install
COPY ./src ./src
RUN npm run build

# Stage 2: Production stage
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package.json ./
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
EXPOSE 2000
CMD ["npm", "start"]
```

Procedure:

1. Initialize Project:
 mkdir SecondProgram
 cd SecondProgram

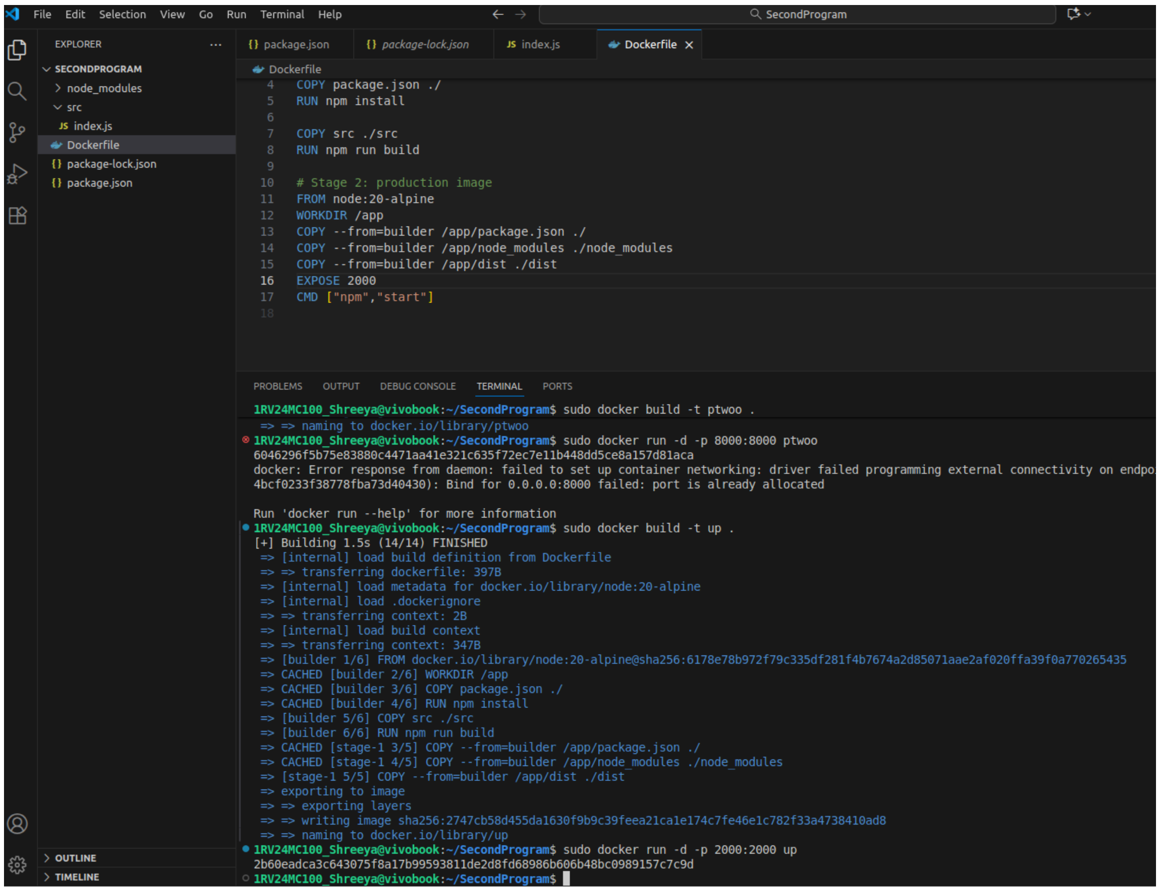
2. Build Docker Image:
 sudo docker build -t up .

3. Run Docker Container:
 sudo docker run -d -p 2000:2000 up

4. Open Browser and Visit:
 http://localhost:2000

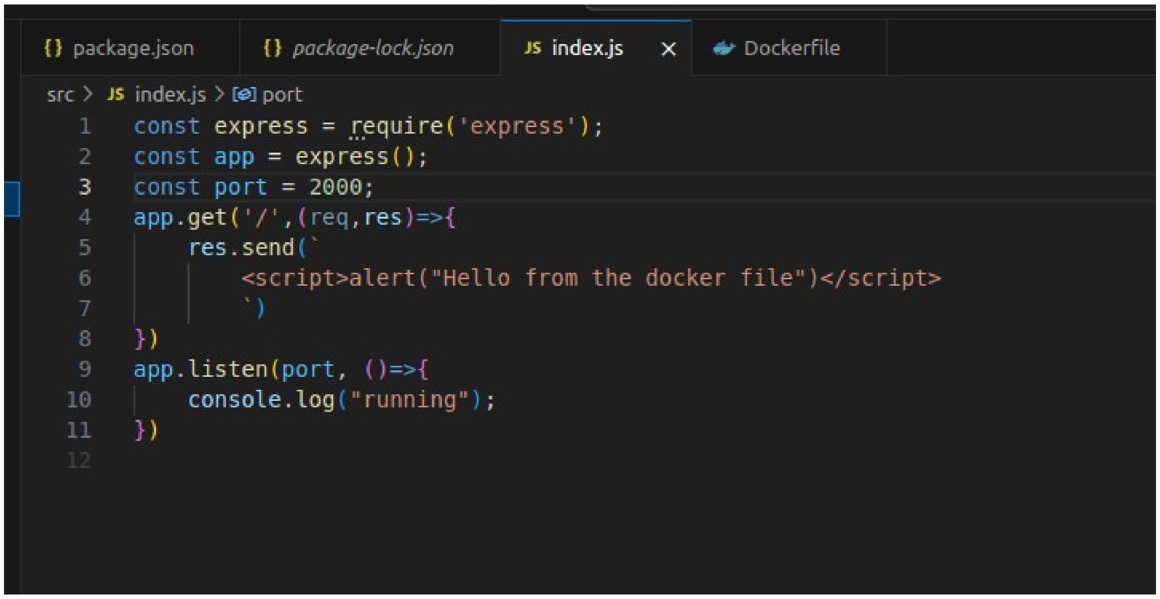
5. You will see the alert message:
 "Hello from the docker file"

Output:

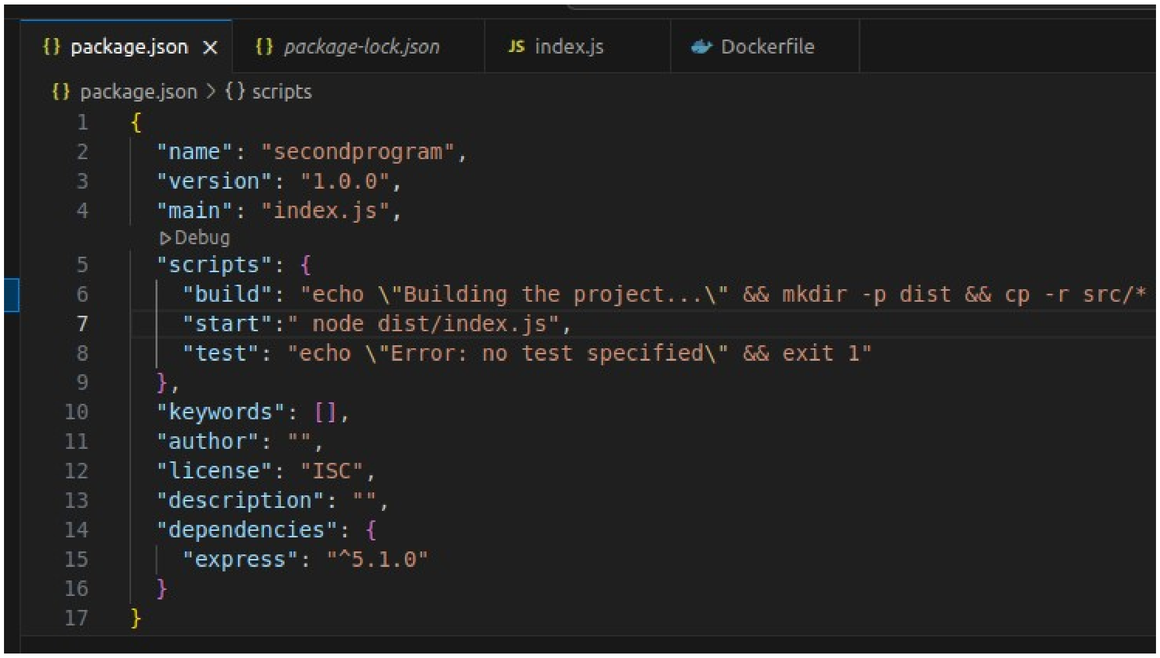


```
COPY package.json ./
RUN npm install

COPY src ./src
RUN npm run build

# Stage 2: production image
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/package.json ./
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
EXPOSE 2000
CMD ["npm","start"]
```

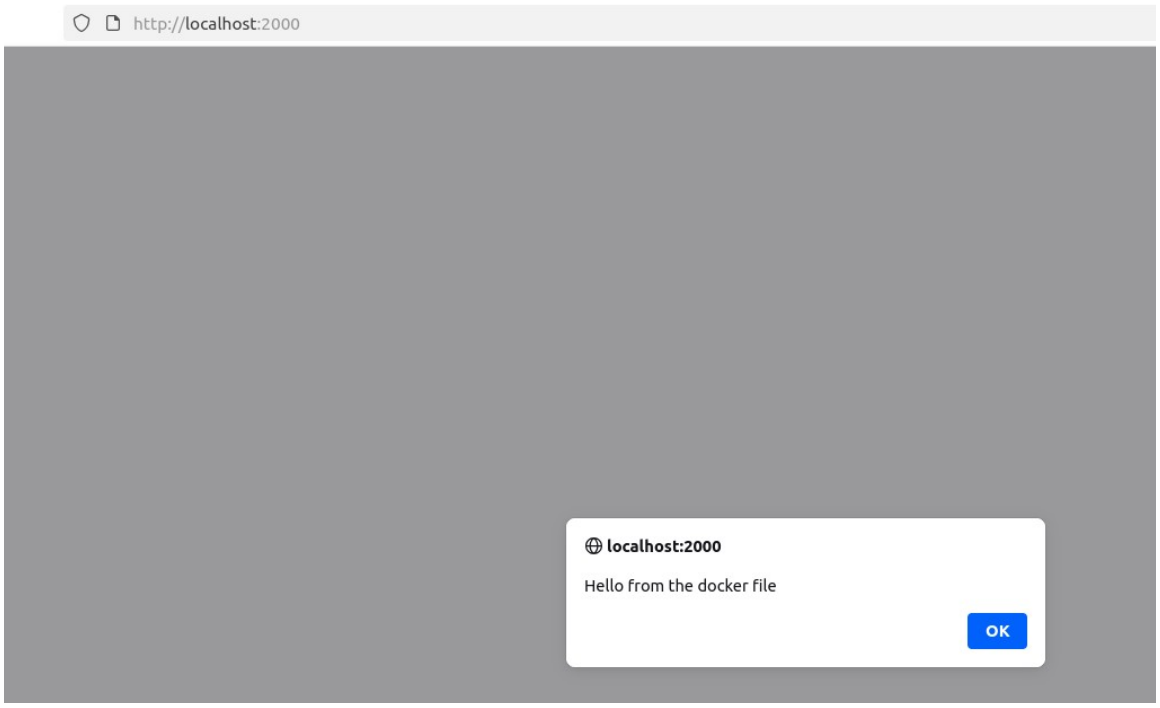
```
1RV24MC100_Shreeya@vivobook:~/SecondProgram$ sudo docker build -t ptwoo .
 => => naming to docker.io/library/ptwoo
1RV24MC100_Shreeya@vivobook:~/SecondProgram$ sudo docker run -d -p 8000:8000 ptwoo
6046296f5b75e83880c4471aa41e321c635f72ec7e11b448dd5ce8a157d81aca
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpo
4bcf0233f38778fba73d40430): Bind for 0.0.0.0:8000 failed: port is already allocated

Run 'docker run --help' for more information
1RV24MC100_Shreeya@vivobook:~/SecondProgram$ sudo docker build -t up .
[+] Building 1.5s (14/14) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 397B
 => [internal] load metadata for docker.io/library/node:20-alpine
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 347B
 => [builder 1/6] FROM docker.io/library/node:20-alpine@sha256:6178e78b972f79c335df281f4b7674a2d85071aae2af020ffa39f0a770265435
 => CACHED [builder 2/6] WORKDIR /app
 => CACHED [builder 3/6] COPY package.json ./
 => CACHED [builder 4/6] RUN npm install
 => [builder 5/6] COPY src ./src
 => [builder 6/6] RUN npm run build
 => CACHED [stage-1 3/5] COPY --from=builder /app/package.json ./
 => CACHED [stage-1 4/5] COPY --from=builder /app/node_modules ./node_modules
 => [stage-1 5/5] COPY --from=builder /app/dist ./dist
 => exporting to image
 => => exporting layers
 => => writing image sha256:2747cb58d455da1630f9b9c39feea21ca1e174c7fe46e1c782f33a4738410ad8
 => => naming to docker.io/library/up
1RV24MC100_Shreeya@vivobook:~/SecondProgram$ sudo docker run -d -p 2000:2000 up
2b60eadca3c643075f8a17b99593811de2d8fd68986b606b48bc0989157c7c9d
1RV24MC100_Shreeya@vivobook:~/SecondProgram$
```



```js
const express = require('express');
const app = express();
const port = 2000;
app.get('/',(req,res)=>{
    res.send(`
        <script>alert("Hello from the docker file")</script>
        `)
})
app.listen(port, ()=>{
    console.log("running");
})
```



```json
{
  "name": "secondprogram",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "build": "echo \"Building the project...\" && mkdir -p dist && cp -r src/*
    "start":" node dist/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^5.1.0"
  }
}
```



Hello from the docker file

Result:

Successfully developed a multi-stage Dockerfile that builds and runs a Node.js application on port 2000 using Express.js.

Conclusion:

This experiment demonstrates the use of multi-stage Docker builds for creating optimized, lightweight, and production-ready container images.