DevOps – Lab Program-2 Documentation

Title:

Develop a Multi-Stage Dockerfile for Container Orchestration

Objective:

To build a Node.js web application using a **multi-stage Dockerfile** for efficient image optimization and container orchestration.

Software Requirements:

- Docker
- Node.js
- Express.js

Files and Folder Structure:

```
program2/
├── build/
├── dist/
│   └── index.js
├── node_modules/
├── package.json
├── package-lock.json
└── Dockerfile
```

Important Files:

dist/index.js

```
const express = require('express');
const app = express();
const PORT = 8080;
app.get('/', (req, res) => {
res.send('Hello from multi-stage Docker!');
});

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

Dockerfile

```
FROM node:20-alpine AS builder

WORKDIR /app

COPY package.json package-lock.json ./
RUN npm install

COPY . .

RUN npm run build

FROM node:20-alpine

WORKDIR /app

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 8080

CMD ["node", "dist/index.js"]
```

Dockerfile specifying the base image, directory path, and commands to install and run for both Build environment and Production environment

Procedure:

1. Create the Project Directory:

mkdir program2
cd program2

Add the necessary files (package.json, Dockerfile, source code, etc.).

2. Build the Docker image:

docker build -t multi-stage-app .

3. Verify image creation:

docker images

4. Run Docker Container:

```
docker run -d -p 8080:8080 --name multi-stage-container multi-stage-app
```

5. Check Running Containers:

```
docker ps
```

6. Access the Application:

Open a browser and visit:

http://localhost:8080

You will see the message:

"Hello from multi-stage Docker!"

Result:

Successfully developed a multi-stage Dockerfile that builds and runs a Node.js application in separate build and runtime stages, reducing image size and improving efficiency.

Conclusion:

This experiment demonstrates the use of multi-stage builds in Docker for efficient image creation and container orchestration. By separating build and runtime environments, the resulting image is lightweight, secure, and production-ready.