Experiment 5.1

Aim

To create a production-ready Docker image for a React application using a multi-stage Docker build, optimizing image size and separating build dependencies from runtime.

Introduction

Docker allows packaging applications with all their dependencies into a single container, ensuring consistency across environments. Multi-stage Docker builds enable the separation of build and runtime environments, reducing the final image size and including only the necessary files for production. In this experiment, a React application is dockerized using Node.js for the build stage and Nginx for serving the production build.

Procedure

1. Create a simple React application using `create-react-app`.
2. Create a `.dockerignore` file to exclude unnecessary files such as `node_modules` and `build` from the Docker build context.
3. Write a multi-stage `Dockerfile`:
4. **Stage 1 (build stage)**: Use a Node.js image, copy project files, install dependencies, and run `npm run build`.
5. **Stage 2 (production stage)**: Use an Nginx image, copy the build files from the first stage, and serve them.
6. Build the Docker image locally using `docker build -t react-app .`.
7. Run the Docker container using `docker run -p 80:80 react-app`.
8. Access the React app at `http://localhost` and verify it works correctly.
9. Check the final image size and confirm it is optimized compared to including all development dependencies.

Code

.dockerignore

```
node_modules
build
.dockerignore
Dockerfile
```

Dockerfile

```
# Stage 1: Build
FROM node:18-alpine AS build
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: Production
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Output

- A working Docker container serving the React app at `http://localhost`.
- The final Docker image is smaller due to exclusion of development dependencies.
- Separation of build and production stages ensures efficient, production-ready image.

Example Commands

```
# Build Docker image
docker build -t react-app .

# Run Docker container
docker run -p 80:80 react-app
```

Learning Outcomes

- Understand how to dockerize a React application.
- Learn to implement multi-stage builds to reduce image size.
- Gain knowledge of separating build and production dependencies.
- Learn how to serve React apps using Nginx in a container.
- Verify and test containerized applications locally.