Practice 1 - Cloud Deployment

Title: Dockerize a React Application with Multi-Stage Build

Objective:

Learn how to create a production-ready Docker image for a React application using a multi-stage Docker build. This approach helps reduce image size, separate build dependencies from runtime, and prepare your app for deployment.

Required Materials:

```
- Installed Docker (latest version)
- Node.js (for local testing if needed)
- A simple React application (created using Create React App)
```

Steps to Follow:

1. **Create a React Application**
   ```bash
   npx create-react-app my-react-app
   cd my-react-app
   ```

2. **Create a Multi-Stage Dockerfile**
   Create a file named `Dockerfile` in the project root and add the following content:
   ```Dockerfile
   # Stage 1: Build the React app
   FROM node:18-alpine AS build
   WORKDIR /app
   COPY package*.json ./
   RUN npm install
   COPY . .
   RUN npm run build

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

3. **Create a .dockerignore File**
   Add the following lines to `.dockerignore` to avoid copying unnecessary files:
   ```
   node_modules
   build
   Dockerfile
   .dockerignore
   .git
   .gitignore
   ```

4. **Build the Docker Image**
   ```bash
   docker build -t react-app:latest .
   ```

5. **Run the Docker Container**
   ```bash
   docker run -d -p 80:80 react-app:latest
   ```

6. **Verify the Application**
   Open a browser and go to:
   ```
   ```

   http://localhost
   ```

7. **Check the Image Size**
   ```bash
   docker images
   ```
   You should see that the final image size is smaller compared to a single-stage build since dev dep

Expected Output:

- A working Docker container that serves your React app at http://localhost
- Optimized Docker image with significantly smaller size
- Clear separation between build and production stages in the Dockerfile
   ```