1. **Basic Dockerfile Structure** set env and port number.

* Create a simple Dockerfile for a Node.js application.
* Use an official Node.js base image.
* Set the working directory to /app.
* Copy the package.json and package-lock.json files into the container.
* Run the npm install command.
* Add an environment variable NODE\_ENV with a default value of production.
* Expose port 3000 for the application.

FROM node:21-alpine3.18

RUN apk update && apk add /bin/sh && \

     mkdir /app

ENV PROJECT\_HOME /app

WORKDIR $PROJECT\_HOME

COPY package.json package-lock.json  $PROJECT\_HOME

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node","app.js"]

1. **Multiple Stages:** Create a multi-stage Dockerfile for a Python application.

* Use an official Python base image.
* Copy only the necessary files for runtime from the builder stage.
* Ensure the final image is as small as possible.

1. **Custom Commands and Health Checks**:

* Build a Dockerfile for a Java Spring Boot application.
* Use an official OpenJDK base image.
* Set up a custom command to run the application.
* Add a health check to ensure the application is running correctly.

1. **Volume Mounts:**

* Develop a Dockerfile for a MySQL database.
* Use an official MySQL base image.
* Configure a volume for the database data.
* Set environment variables for database name, user, and password.

1. **Networking and Container Communication:**

* Write a Dockerfile for a simple web server using Nginx.
* Expose port 80.
* Demonstrate how to link this container with another container running a basic Node.js app.

1. **Security Best Practices:**

* Create a Dockerfile for a Python application.
* Implement best practices for security, such as running as a non-root user, using only necessary permissions, and minimizing layers.

1. **Docker Compose:**

* Write a docker-compose.yml file for a stack containing a web application (Node.js), a database (MySQL), and a caching service (Redis).
* Configure networking between the services.
* Use environment variables for configuration.

1. **Dockerfile for a Frontend App:**

* Develop a Dockerfile for a React.js application.
* Utilize a lightweight Node.js base image.
* Set up the container to serve the production build of the React app.

1. **Debugging in Docker:**

* Build a Dockerfile for a Python application.
* Configure it to run in development mode.
* Enable remote debugging capabilities.