Developing a Multi-stage Dockerfile for Container Orchestration

Project Structure:

```
progg/
├── src/
├── package.json
├── package-lock.json
├── node_modules/
├── Dockerfile
```

1. Create project directory:

```
user@1rv24mc057-kumarsameer:~$ mkdir progg && cd progg
user@1rv24mc057-kumarsameer:~/progg$ ls
```

2. Initialize `package.json`:

```
user@1rv24mc057-kumarsameer:~/progg$ npm init -y
Wrote to /home/user/progg/package.json:

{
  "name": "progg",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

3. Install **Express**:

```
user@1rv24mc057-kumarsameer:~/progg$ npm install express

added 68 packages, and audited 69 packages in 2s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

4. In the package.json file, add the scripts

```
"scripts": {
  "start":"node dist/index.js",
  "build":"mkdir -p dist && cp -r src/* dist/"
},
```

Note- Without the `build` script, `npm run build` in your Dockerfile will fail.

Without the `start` script, you cannot use `npm start` in production, which is a standard convention.

Create a Dockerfile manually

```
user@1rv24mc057-kumarsameer:~/progg$ nano Dockerfile
```

Inside the Dockerfile, write these commands -

```
# Use Node.js 20 on Alpine Linux for a lightweight build environment
FROM node:20-alpine AS builder

# Set working directory inside the container
WORKDIR /app

# Copy package.json to container to install dependencies
COPY package.json ./

# Copy package-lock.json to ensure exact dependency versions
COPY package-lock.json ./

# Install all dependencies defined in package.json
RUN npm install


# Copy the rest of the application source code into the container
COPY . .

# Run the build script (for transpiling, bundling, etc.)
# This will generate a 'dist' folder or build artifacts
RUN npm run build

# Use a fresh Node.js 20 Alpine image for the final production image
FROM node:20-alpine

# Set working directory for the production container
WORKDIR /app

# Copy only the package files from builder stage (needed for npm info)
COPY --from=builder /app/package.json ./
COPY --from=builder /app/package-lock.json ./

# Copy the compiled build artifacts (dist folder) from builder stage
COPY --from=builder /app/dist ./dist

# Copy the installed node_modules from builder stage to use in production
COPY --from=builder /app/node_modules ./node_modules

# Expose the port that the application listens on
EXPOSE 3000

# Define the command to run the app when the container starts
CMD ["node","dist/index.js"]
```

Now build the Docker Image

```
user@1rv24mc057-kumarsameer:~/progg$ docker build -t progg ./
[+] Building 45.7s (16/16) FINISHED                              docker:default
 => [internal] load build definition from Dockerfile                      0.0s
 => => transferring dockerfile: 424B                                      0.0s
 => [internal] load metadata for docker.io/library/node:20-alpine         0.7s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [internal] load build context                                         0.1s
 => => transferring context: 43.88kB                                      0.0s
 => [builder 1/7] FROM docker.io/library/node:20-alpine@sha256:6178e78b   7.9s
 => => resolve docker.io/library/node:20-alpine@sha256:6178e78b972f79c3   0.1s
 => => sha256:6178e78b972f79c335df281f4b7674a2d85071aae 7.67kB / 7.67kB   0.0s
 => => sha256:be8d32d651b3e0c9c2b28fdc1d3888408125d7032 1.72kB / 1.72kB   0.0s
 => => sha256:2b56f2779663b9e1a77bdb5235dc31f1a81e534cc 6.42kB / 6.42kB   0.0s
 => => sha256:60e45a9660cfaebbbac9bba98180aa28b3966b7 42.75MB / 42.75MB   4.5s
 => => sha256:e74e4ed823e9560b3fe51c0cab47dbfdfc4b12453 1.26MB / 1.26MB   2.2s
 => => sha256:da04d522c98fe12816b2bcddf8413fca73645f8fa60f2 444B / 444B   1.4s
 => => extracting sha256:60e45a9660cfaebbbac9bba98180aa28b3966b7f2462d1   0.7s
 => => extracting sha256:e74e4ed823e9560b3fe51c0cab47dbfdfc4b1245360431   0.0s
 => => extracting sha256:da04d522c98fe12816b2bcddf8413fca73645f8fa60f28   0.0s
 => [builder 2/7] WORKDIR /app                                            3.1s
 => [builder 3/7] COPY package.json ./                                    0.5s
 => [builder 4/7] COPY package-lock.json ./                               0.6s
 => [builder 5/7] RUN npm install                                         4.0s
 => [builder 6/7] COPY . .                                               10.5s
 => [builder 7/7] RUN npm run build                                       4.2s
 => [stage-1 3/6] COPY --from=builder /app/package.json ./                1.4s
 => [stage-1 4/6] COPY --from=builder /app/package-lock.json ./           2.6s
 => [stage-1 5/6] COPY --from=builder /app/dist ./dist                    2.6s
 => [stage-1 6/6] COPY --from=builder /app/node_modules ./node_modules    1.6s
 => exporting to image                                                    0.5s
 => => exporting layers                                                   0.5s
 => => writing image sha256:f9b187557168c68fb58eaf124799368ab88deb0c399   0.0s
 => => naming to docker.io/library/progg                                  0.0s
```

Check the image that was built - :

```
user@1rv24mc057-kumarsameer:~/progg$ docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
progg           latest    f9b187557168   23 seconds ago   137MB
prog            latest    cf09a4775f39   17 minutes ago   1.14GB
dockerrr3       latest    d94d2120c94c   14 hours ago     1.14GB
<none>          <none>    26a28ac6d1b7   14 hours ago     1.14GB
deepika-image   2.0       17d9c52f6880   3 days ago       147MB
wordpress       latest    7332768c717f   4 weeks ago      734MB
hello-world     latest    1b44b5a3e06a   2 months ago     10.1kB
```

Run the container-

```
user@1rv24mc057-kumarsameer:~/progg$ docker run -p 4005:3000 progg
Server is listening at port:3000
```

This maps the container port 3000 to host port 4005.

The Node.js app is accessible at: http://localhost:4005