FROM node:20-alpine as build-stage

WORKDIR /react

# If the package doesn't change, no need to copy it every time we run the build command   
COPY package\*.json .

RUN npm install

COPY . .

RUN npm run build

FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /react/dist /usr/share/nginx/html

# the container is running at port 3000. It helps DevOps just seeing the docker file   
# EXPOSE 3000

### #start nginx and serve the application CMD ["nginx","-g","daemon off;"] **Build Stage**

****Specify the Base Image for the Build Stage****

dockerfile

Copy code

FROM node:20-alpine as build-stage

* + This line specifies the base image to be used for the build stage. **node:20-alpine** is a lightweight Node.js image based on Alpine Linux.
  + **as build-stage** names this stage as **build-stage** for reference later.

****Set the Working Directory****

dockerfile

Copy code

WORKDIR /react

* + Sets the working directory inside the container to **/react**.
  + All subsequent commands will be run in this directory.

****Copy the Package Files****

dockerfile

Copy code

COPY package\*.json .

* + Copies **package.json** and **package-lock.json** from the host machine to the working directory in the container.
  + This is done before copying the entire source code to leverage Docker's caching mechanism. If **package.json** hasn't changed, the npm install layer can be reused.

****Install Dependencies****

dockerfile

Copy code

RUN npm install

* + Runs **npm install** to install all dependencies defined in **package.json**.
  + This step is separate from copying the rest of the source code to optimize build caching.

****Copy the Source Code****

dockerfile

Copy code

COPY . .

* + Copies the entire source code from the host machine to the working directory in the container.

****Build the Application****

dockerfile

Copy code

RUN npm run build

* + Runs the build command specified in the **package.json** (typically **npm run build**).
  + This compiles the React application into static files that can be served by a web server.

### **Production Stage**

****Specify the Base Image for the Production Stage****

dockerfile

Copy code

FROM nginx:stable-alpine as production-stage

* + This line specifies the base image to be used for the production stage. **nginx:stable-alpine** is a lightweight Nginx image based on Alpine Linux.
  + **as production-stage** names this stage as **production-stage** for reference.

****Copy Built Files from the Build Stage****

dockerfile

Copy code

COPY --from=build-stage /react/dist /usr/share/nginx/html

* + Copies the built static files from the **build-stage** to the Nginx directory that serves static content (**/usr/share/nginx/html**).
  + **--from=build-stage** references the previous build stage named **build-stage**.

****Expose the Port (Optional and Commented Out)****

dockerfile

Copy code

# EXPOSE 3000

* + This line is commented out. If it were active, it would indicate that the container listens on port 3000. However, in this case, Nginx typically listens on port 80.

****Start Nginx and Serve the Application****

dockerfile

Copy code

CMD ["nginx","-g","daemon off;"]

* + Sets the default command to start Nginx and keep it running in the foreground.
  + **nginx -g 'daemon off;'** starts Nginx with the **daemon off** directive to ensure it runs in the foreground, which is required for Docker containers.

### **Summary**

* The ****build stage**** uses a Node.js environment to install dependencies and build the React application into static files.
* The ****production stage**** uses an Nginx environment to serve the static files built in the first stage.
* By using multi-stage builds, the final Docker image is smaller because it only contains the Nginx server and the static files, excluding all Node.js dependencies and build tools.