Sample Dockerfile for a Node.js application along with a **detailed explanation of each step** so you understand exactly what's happening.

Dockerfile

dockerfile Copy Edit

```
# Step 1: Use an official Node.js base image
FROM node:18-alpine

# Step 2: Set the working directory inside the container
WORKDIR /app

# Step 3: Copy package.json and package-lock.json for dependency installation
COPY package*.json ./

# Step 4: Install dependencies
RUN npm install --production

# Step 5: Copy the rest of the application source code
COPY . .

# Step 6: Expose the application port
EXPOSE 3000

# Step 7: Define the command to run the app
CMD ["npm", "start"]
```

Detailed Step-by-Step Explanation

Step 1: Use an official Node.js base image

FROM node:18-alpine

- **Purpose:** Defines the starting point for our image.
- **Why node:18-alpine?**
  - **Node:18** → A specific LTS (Long-Term Support) version of Node.js for stability.
  - **Alpine** → A minimal Linux distribution, smaller in size (~5 MB) for faster builds and smaller images.
- This base image already includes Node.js and npm, so we don't have to install them manually.

Step 2: Set the working directory

dockerfile

CopyEdit

WORKDIR /app

- **Purpose:** Sets /app as the working directory inside the container.
- Any following commands (COPY, RUN, CMD) will be executed from here.
- Helps keep the container filesystem organized.

Step 3: Copy only dependency files first

dockerfile

CopyEdit

COPY package*.json ./

- **Purpose:** Copies package.json and package-lock.json into the container.
- **Why first?**
    - Docker uses **layer caching** — if dependencies haven't changed, this step is cached, so the npm install step won't need to run again on future builds.
- package*.json is a wildcard so it matches both package.json and package-lock.json.

Step 4: Install dependencies

RUN npm install --production

- **Purpose:** Installs only production dependencies (skipping dev dependencies) to reduce image size.
- Uses npm's cache to speed up repeated builds.

Step 5: Copy application source code

COPY . .

- **Purpose:** Copies the rest of your application code into the container's /app directory.

- This is done **after** installing dependencies to take advantage of Docker layer caching.

Step 6: Expose application port

EXPOSE 3000

- **Purpose:** Informs Docker (and anyone running the container) that the app listens on port 3000.
- This doesn't publish the port automatically — you must still map it with docker run -p 3000:3000.

Step 7: Start the application

CMD ["npm", "start"]

- **Purpose:** Defines the default command to run when the container starts.
- **Why array format?** It's the **exec form**, which avoids running through a shell and ensures signals (like CTRL+C) are properly passed to the Node.js process.

✅ **Build & Run Commands**

```
# Build the image
docker build -t my-node-app .


# Run the container, mapping port 3000
docker run -p 3000:3000 my-node-app
```