Study Notes: Dockerizing a Node.js Application

1. Dockerfile Basics

Original Dockerfile

Dockerfile

```
FROM node:8.16.1-alpine
WORKDIR /app
COPY . /app
RUN npm install
EXPOSE 5000
CMD node index.js
```

Issues

- **Node version too old** → Node 8 is deprecated, lacks modern npm features.
- **COPY . /app** → copies everything, including unnecessary files (node_modules, logs).
- **RUN npm install** → installs dev dependencies too, bloating the image.
- **CMD node index.js** → works, but not conventional compared to npm start.

2. Improved Dockerfile

Dockerfile

```
FROM node:18-alpine
# modern LTS
WORKDIR /app
# Copy only dependency files first for better caching
COPY package*.json ./
# Install only production dependencies
RUN npm install --omit=dev
# Copy rest of the source codeCOPY . .
EXPOSE 5000
# Use npm start (calls the "start" script in package.json)
CMD ["npm", "start"]
```

Justification

- **node:18-alpine** → lightweight + secure + supported LTS.
- **COPY package*.json ./** → ensures Docker caches dependency installs; faster rebuilds.
- **npm install --omit=dev** → skips devDependencies (like webpack) in production.

- **CMD ["npm", "start"]** → cleaner, uses package.json scripts convention.

3. Why package*.json

- The *** wildcard** copies both:
- **package.json** → dependency definitions.
- **package-lock.json** → exact versions for reproducibility.
- **Benefit:** Docker rebuilds dependencies only when these files change, saving time.

4. npm install vs npm ci

npm install

- Works with just **package.json.**
- Generates/updates **package-lock.json.**
- Flexible but **less reproducible**.
- **npm ci**
- Requires **package-lock.json.**
- **Installs exact versions** → reproducible builds.
- **Faster, ideal for CI/CD**.

Error you saw: npm ci failed because **no package-lock.json existed**. → **Solution:** either generate lockfile locally (npm install) or switch to npm install --omit=dev in Dockerfile.

5. package.json Explained
json

```
{
  "name": "nodejs-hello-world",
  "version": "1.0.0",
  "description": "nodejs-hello-world",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Sourav",
  "dependencies": {
    "express": "^4.18.0"
  },
  "devDependencies": {
    "webpack": "^4.41.5"
  },
  "license": "MIT",
  "repository": {
    "type": "git",
    "url": "https://github.com/Sourav356/nodejs-hello-world.git"
  }}
```

❖ **Key Fields**

- **scripts.start** → defines how to run app (npm start → node index.js).
- **dependencies** → runtime packages (Express).
- **devDependencies** → build tools (Webpack), excluded in production.
- **repository.url** → should point to repo root (.git form is canonical for Git).

6. CMD vs ENTRYPOINT

- **CMD ["npm", "start"]** → flexible, can be overridden at runtime.
- **ENTRYPOINT ["npm", "start"]** → locks the command, harder to override.
- **Best practice:** use CMD for app start.

7. Git vs Docker

- Docker build uses local files in your project folder.
- git push is not required for local builds.
- Push only if:
- You want CI/CD pipelines to build from GitHub.
- Or you want to build directly from GitHub with:
- **bash**
  - ➢ **docker build https://github.com/Sourav356/nodejs-hello-world.git#main**

8. Common Errors & Fixes

- ➢ **npm ci error** → **no lockfile** → **switch to npm install --omit=dev.**
- ➢ **bash**: **npm**: **command not foun**d → Node.js not installed locally → fix by using Docker (no need to install locally if you rely on Docker).

➢ **CMD ["node", "start"] error** → wrong usage → must be **CMD ["npm", "start"].**

9. Final Folder Structure

Code

```
nodejs-hello-world/
├── Dockerfile
├── package.json
├── index.js
├── node_modules/   (ignored in Docker build)
└── package-lock.json (optional, if generated)
```

10. Docker Commands

Build the image

Bash

- docker build -t souravdevopsdev/first-prod-nodejs:0.0.1 .

-t → tags the image with your repo name + version.

. → build context is the current directory (where Dockerfile is).

11. Run the container

Bash

```
docker run -d -p 5001:5000 souravdevopsdev/first-prod-nodejs:0.0.1
```

-d → run in detached mode (background).

-p 5001:5000 → maps host port 5001 → container port 5000.

souravdevopsdev/first-prod-nodejs:0.0.1 → the image you built.

- After this, open http://localhost:5001 in your browser to access the app.

12. Push the image to Docker Hub

Bash

- docker login
- docker push souravdevopsdev/first-prod-nodejs:0.0.1

docker login → authenticate with Docker Hub (enter your Docker Hub username/password).

docker push → uploads the image to your Docker Hub repo.

13. Justification for Commands

- Build → creates the image from your Dockerfile and tags it for versioning.
- Run → starts the container, maps ports so you can access it from your host.
- Push → makes the image available remotely (for CI/CD, Kubernetes, or sharing).

14. Summary

- Use modern Node base image (node:18-alpine).
- Copy package.json first* → caching.
- Use npm install --omit=dev if no lockfile.
- Define start script in package.json → run with npm start.
- Repository URL → point to repo root, not subfolder.
- Git push not required for local Docker builds.
- Errors like npm ci or npm not found are environment-related, solved by either generating lockfile or relying on Docker's Node runtime.