🚀 Advanced Dockerfile & Image Building for an Express App

📝 Quick Definitions (simple)

🔑 Term	💡 Meaning
🏗️ **Base Image**	The starting image you build on (like `node:18-alpine`)
🪄 **Multi-Stage Build**	Use multiple `FROM` stages (builder → runtime) to keep final image small
🧱 **Layer**	Each Dockerfile instruction creates a layer; fewer layers = smaller image
🚫 **.dockerignore**	Tells Docker which files **not** to send to build context (e.g., `.git`, `node_modules`)
🦴 **Alpine / Distroless**	Tiny runtime images: Alpine = small+shell, Distroless = even smaller+no shell

🏗️ Project Structure

```
express-demo/
├─ package.json
├─ package-lock.json
├─ index.js
├─ Dockerfile.base
├─ Dockerfile
└─ .dockerignore
```

⚙️ 1. Create a Tiny Express App

`package.json`

```
{
  "name": "express-demo",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

index.js

```
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;
app.get('/', (req, res) => res.send('👋 Hello from Dockerized Express!'));
app.listen(port, () => console.log('Server running on port', port));
```

🛠️ 2. Build a Custom Base Image

Dockerfile.base

```
FROM node:18-alpine AS base
LABEL maintainer="you@example.com"

# 👤 Add non-root user
RUN addgroup -S appgroup && adduser -S appuser -G appgroup

WORKDIR /home/app
ENV NODE_ENV=production
USER appuser
```

Build it:

```
docker build -f Dockerfile.base -t my-node-base:18-alpine .
```

✅ This gives you a reusable, safe base with a non-root user.

🧙 3. Multi-Stage Dockerfile

🅰️ Distroless (smallest)

```
# Stage 1 — dependencies
FROM my-node-base:18-alpine AS deps
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci --production --silent && npm cache clean --force

# Stage 2 — builder (if build step needed)
FROM my-node-base:18-alpine AS builder
WORKDIR /app
COPY --chown=appuser:appgroup . .

# Stage 3 — final runtime
FROM gcr.io/distroless/nodejs:18 AS runner
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY --from=builder /app .
ENV NODE_ENV=production
EXPOSE 3000
USER nonroot
CMD ["index.js"]
```

Build:

```
docker build -t express-demo:distroless .
```

🅱️ Alpine (debug-friendly)

```
FROM my-node-base:18-alpine AS deps
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci --production --silent && npm cache clean --force

FROM my-node-base:18-alpine AS builder
WORKDIR /app
COPY --chown=appuser:appgroup . .

FROM node:18-alpine AS runner
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY --from=builder /app .
ENV NODE_ENV=production
EXPOSE 3000
USER appuser
CMD ["node","index.js"]
```

Build:

```
docker build -t express-demo:alpine .
```

🚫 4. `.dockerignore`

```
node_modules
npm-debug.log
Dockerfile*
.dockerignore
.git
.gitignore
.vscode
.env
*.md
```

This keeps the build context **light** 🪶.

🏃 5. Build & Run

Build base:

```
docker build -f Dockerfile.base -t my-node-base:18-alpine .
```

Build app:

```
docker build -t express-demo:distroless .
```

Run container:

```
docker run --rm -p 3000:3000 express-demo:distroless
```

Visit 👉 http://localhost:3000

Check size:

```
docker images express-demo:distroless
```

🎯 6. Why This is Smaller

📝 Technique	💡 Effect
Multi-Stage	Builder tools don't end up in final image
`npm ci --production`	Only prod deps included
`.dockerignore`	Avoids sending big folders
Distroless / Alpine	Tiny OS footprint
Combine RUN & clear caches	Fewer, smaller layers
Non-root user	Security best practice

💪 7. Production Touches

- 🩺 **Healthcheck**
- 🏷️ **LABEL metadata**
- 🔒 **Secrets & env vars**
- 🛡️ **Image scanning**

Example `HEALTHCHECK`:

```
HEALTHCHECK --interval=30s --timeout=3s CMD wget -qO- http://localhost:3000/ || exit 1
```

🎓 8. Practice Challenge

1. 📝 Make the app.
2. 🏗️ Build `my-node-base:18-alpine`.
3. 📦 Build both images (Alpine + Distroless).
4. 📊 Compare sizes with `docker images`.
5. 🧹 Minimize size further (tighten `.dockerignore`, remove devDeps).

⚠️ 9. Common Mistakes to Avoid

- Copying whole app **before** `npm install` (breaks caching).
- Shipping your local `node_modules` into image.
- Forgetting `.dockerignore`.
- Leaving devDependencies in final image.
- Running as `root` in production.
