1. Install Docker

Download and install Docker Desktop

Docker provides the container runtime environment.

It lets you build, run, and manage isolated environments (containers).

2. Create Angular App (if not already)

ng new my-angular-app

cd my-angular-app

ng serve

This runs on http://localhost:4200.

3. Add Dockerfile

In your Angular project root, create a file named Dockerfile

# Step 1: Build the Angular app

FROM node:18 AS build

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build --prod

# Step 2: Serve it with nginx

FROM nginx:alpine

COPY --from=build /app/dist/my-angular-app /usr/share/nginx/html

EXPOSE 80

You’ve written a **multi-stage Dockerfile**:

1. **Stage 1: Build Stage (Node.js container)**
   * Builds your Angular app using Node.js (npm install, npm run build --prod)
   * Output: compiled Angular app (in dist/my-angular-app)
2. **Stage 2: Production Stage (Nginx)**
   * Takes the build output and serves it using Nginx (a lightweight web server)

So the final image contains **only the production-ready frontend**.

First stage compiles Angular app.

Second stage copies build to an Nginx server to serve it like a website.

WORKDIR /app - • Idi container lo create cheyyaboye folder name.

• /app ane path lo mana Angular project files copy cheyyadam jarugutundi.

Important: Idi me local project lo unde src/app/ folder kaadu. This is container inside lo create cheyyaboye custom folder.

• So, idi any name unte parledhu (e.g., /myproject), but /app is just a naming convention.

COPY package\*.json ./ - Idi shortcut syntax for:

• package.json

• package-lock.json

Explanation:

• package.json: Dependencies list untundi.

• package-lock.json: Exact version lock cheyyadam kosam, ensures same versions every install.

package\*.json means:

“Copy both package.json and package-lock.json if they exist.”

COPY package\*.json ./

RUN npm install

First copy package\*.json → then run npm install

This ensures npm install works even before copying entire code.

Docker caching improves: only re-runs npm install if package files change.

Then copy remaining project files: COPY . .

Idi Dockerfile lo chala common ga use chese command. Let’s break it down:

• First . → Refers to the source path from your local machine (build context).

• Second . → Refers to the destination path inside the Docker container (which is already set by WORKDIR /app earlier).

So practically:

WORKDIR /app

COPY . .

This means:

“Copy everything in the current local folder (project folder) into the /app folder inside the container.”

Example:

Let’s say your Angular project folder looks like this:

my-angular-app/

├── src/

├── package.json

├── angular.json

├── Dockerfile

If you’re running docker build from inside my-angular-app, then:

• First . → my-angular-app/ (everything inside this folder)

• Second . → /app/ (in the container)

Result: All your code will be inside /app in the container.

⸻

Important Note:

Use .dockerignore to exclude files/folders (like node\_modules, dist, etc.) — so they won’t be copied into the container.

4. Add .dockerignore (optional but recommended)

In dockerfile:

node\_modules

dist

.git

Exclude unnecessary files from Docker image to keep it clean.

5. Build Docker Image

In bash terminal

docker build -t my-angular-app .

This command reads the Dockerfile and creates an image.

• -t tags the image with a name.

6. Run Docker Container

In bash:

docker run -d -p 8080:80 my-angular-app

• -d: detached mode

• -p 8080:80: maps Docker container port 80 (Nginx) to localhost:8080

Now open: http://localhost:8080 — Angular app will load!

Now… What’s Different in Dev Containers?

If You Use Dev Containers:

• You won’t use Nginx to serve the app.

• You’ll run ng serve inside the container.

• Dev containers are for coding, debugging — not deployment.

• Docker containers are for running the final product.

COPY package\*.json ./

• First part (package\*.json): Tells Docker to copy files like:

• package.json

• package-lock.json

• Second part (./): Refers to destination folder inside the container, which is already set as: WORKDIR /app

So it means:

“Copy these JSON files into the /app folder in the container.”

RUN npm run build --prod

This command compiles your Angular code into production-ready static files.

It generates the output in:

dist/my-angular-app/ folder (HTML, JS, CSS files)

COPY --from=build /app/dist/my-angular-app /usr/share/nginx/html

This line is in the second stage of the Dockerfile (Nginx stage).

--from=build - Refers to the previous stage we named as build here: FROM node:18 AS build

“From the first stage (build), take the compiled Angular output folder and copy it into the Nginx HTML folder, which serves the website.”

• /usr/share/nginx/html: This is where Nginx serves static files from.

• So Angular app will be hosted by Nginx!

* COPY --from=build tells Docker to copy **from the build stage** (not from your local host)
* It copies the Angular build output **from inside the first container** (/app/dist/my-angular-app)
* To the path where Nginx serves static files:  
  ➤ /usr/share/nginx/html (inside the nginx container)

EXPOSE 80

Tells Docker: “This container listens on port 80.”

• Port 80 is the default HTTP port for browsers.

• This helps when you map ports using -p during docker run.

Note: This command itself doesn’t open the port, but acts as metadata for Docker.

WHAT DOES docker build -t my-angular-app . DO?

• Creates a Docker image using your Dockerfile.

• -t my-angular-app: Tags the image with a name.

• . (dot): Tells Docker to look in the current directory for the Dockerfile and code.

Why image creation?

• You are freezing your app into a single package.

• This can now run anywhere without setup — same version, same code, same environment.

docker run -d -p 8080:80 my-angular-app — what this does?

• -d: Detached mode (runs in background)

• -p 8080:80: Maps your host machine’s port 8080 to container’s port 80

• my-angular-app: Name of the image to run

my-angular-app/

├── src/

├── angular.json

├── package.json

├── Dockerfile <-- this is important

Let’s say your Angular project structure looks like this:

Then in bash terminal:

cd my-angular-app # Step into your project folder

docker build -t my-angular-app . # Build image from here

Why inside the project folder?

• Docker uses the current folder (.) as the build context.

• It looks for the Dockerfile and other files (like package.json) in that directory.

If you run it outside, Docker can’t find the Dockerfile and code, and it will fail.

Dev Containers (from Microsoft – VS Code):

Definition:

Dev Container ante development environment ni container lo set cheyyadam using a .devcontainer folder. Idi mainly VS Code lo use chestaru.

Use:

• Developer ki pre-configured coding environment ivvadam.

• Docker base mida untundi, but for writing & debugging code, not production.

• Project open chesinappude VS Code asks: “Reopen in Container?”

Main Purpose:

“Set up developer environment easily and consistently” across teams using Docker in VS Code.

So, Dev Containers are for development, and Docker Containers are for deployment & runtime.

Dev Containers use Docker internally but are developer-friendly.