**Docker Desktop**

**Dev-Containers Extension for VS Code**

**Docker Desktop Installation and Configuring Dockerfile in project folder:**

* **Container** is like a mini computer (virtual) running inside your real computer.
  + It uses your system’s CPU, RAM, Disk
  + But it’s isolated (doesn’t affect your local system)
  + You can have multiple containers running side-by-side

**Step-1:**

Download and install Docker Desktop from

Docker provides the **container runtime environment.**

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

**Step-2:**

Clone Angular project from GitHub Remote Repo or,  
Create Angular App (if not already)

ng new my-angular-app

cd my-angular-app

ng serve

Angular App runs on http://localhost:4200.

**Step-3:**

Add Dockerfile

In your Angular project root, create a file named **Dockerfile** without any extension.

Write the following commands inside **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

**Explaining each command in Dockerfile:**

* First stage(i.e., # Step 1) ---- compiles Angular app.
* Second stage (i.e., # Step 2) ------ copies build to a Nginx server to serve it like a website.

1. FROM node:18 AS build

* Docker installs Node.js version 18 inside the container.
* Then we use it to run following commands in **Dockerfile**:
  + **RUN npm install**
  + **RUN npm run build --prod**
* Even Angular CLI (@angular/cli) is installed as part of **package.json**, so Docker handles that too.

1. **WORKDIR** /app

* Idi container lo create cheyyaboye folder name.
* This sets the working directory inside the container to **/app.**
* **app** ane folder lo Angular project files copy avvadam jarugutundi.

Important: Idi me Angular project lo unde **src/app/** folder kaadu. Idi container lo create cheyyaboye custom folder.

* So, it can be any name (e.g., /myproject), but **/app** is just a naming convention.

Important: **WORKDIR /app** --- tells Docker to navigate into /app inside container. Hence Dockerfile lo **WORKDIR /app** command tarwata rase any command which specify relative path (like ./) will be on **/app**.

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

* **package.json** ---- has list of Dependencies
* **package-lock.json** ---- has metadata related to exact version lock cheyyadam kosam, ensures same versions every install.
* **package\*.json** ---- **means**: “Copy both **package.json** and **package-lock.json** if they exist.”

1. **COPY** package\*.json ./

**RUN** npm install

* First copy files that match **package\*.json** into **current working directory (./)** inside the container i.e., **/app** because of the **WORKDIR** set above → then run npm install → then copy **COPY . .** remaining project files.
* This ensures **npm install** works even before copying entire code.
* Docker caching improves: only re-runs npm install if package files change.

1. **COPY** . .

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

**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** command at WSL Ubuntu terminal 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.

1. **FROM** nginx:alpine
   * tells Docker to “Download the Nginx lightweight version (Alpine) from **Docker Hub** and use it as the base image.”
   * Then, Docker pulls Nginx from the cloud.
   * Then following command (**COPY** --from=build /app/dist/my-angular-app /usr/share/nginx/html) in above **Dockerfile**, copies Angular build output into: **/usr/share/nginx/html** → This is the default folder from where Nginx serves websites.
   * So, Docker itself will download and use Nginx when building the image — no need to install anything on your system!
   * **Nginx (pronounced “Engine-X”):**
     + Web server – serves static files like HTML, JS, CSS (perfect for Angular)
     + In our **Dockerfile** case, it is used to host your built Angular app (the files in container’s **dist/** folder).

**Step-4:**

Important Note: Add **.dockerignore** in project root folder (optional but recommended)

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

Write inside .**dockerignore**:

/node\_modules

dist

.git

Exclude unnecessary files from Docker image to keep it clean.

**Step-5:**

Build Docker Image

In WSL Ubuntu terminal:

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

docker build -t my-angular-app .

* This command reads the Dockerfile and creates an image.
* -t tags the image with a name.
* 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

**Step-6:**

Run Docker Container

In WSL Ubuntu terminal:

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!**

In Simple Terms:

* + **Dockerfile:** Recipe
  + **Image:** Final frozen version of app (like .exe file)
  + **Port mapping:** Lets you open app in browser
  + **Nginx:** Web server to serve your Angular app efficiently

**After ‘docker build’ command – what happens?**

When you run command in WSL Ubuntu terminal:

docker build -t my-angular-app .

* Docker reads the Dockerfile
  + Executes instructions step-by-step
  + Creates an image named **my-angular-app**
  + The image is stored inside Docker’s internal system (not visible as a file on your local folder)
* “ **docker build** “command won’t modify or create any files in your local disk or project folder.
* Everything (including Angular build output **dist/**) is created inside the Docker container.
  + Your local folder remains exactly the same

Where can you see the result?

* Use this command in WSL Ubuntu terminal:

docker images

* It will show something like:

**REPOSITORY TAG IMAGE ID CREATED SIZE**

**my-angular-app latest abcd1234xyz 2 minutes ago 135MB**

* That confirms your Docker image was created.

The command to run or starts container and see your angular project app:

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

* Then open browser:

http://localhost:8080

* You’ll see your Angular app running — served by Nginx inside the container.

Now… What’s Different in Dev Containers?

If You install Dev-Containers extension for VS code:

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

• You’ll run **ng serve** for Angular App, **npm start** for React app inside the container.

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

• Docker containers are for running the final product.

**Dev-Containers (from Microsoft – VS Code):**

**Definition:**

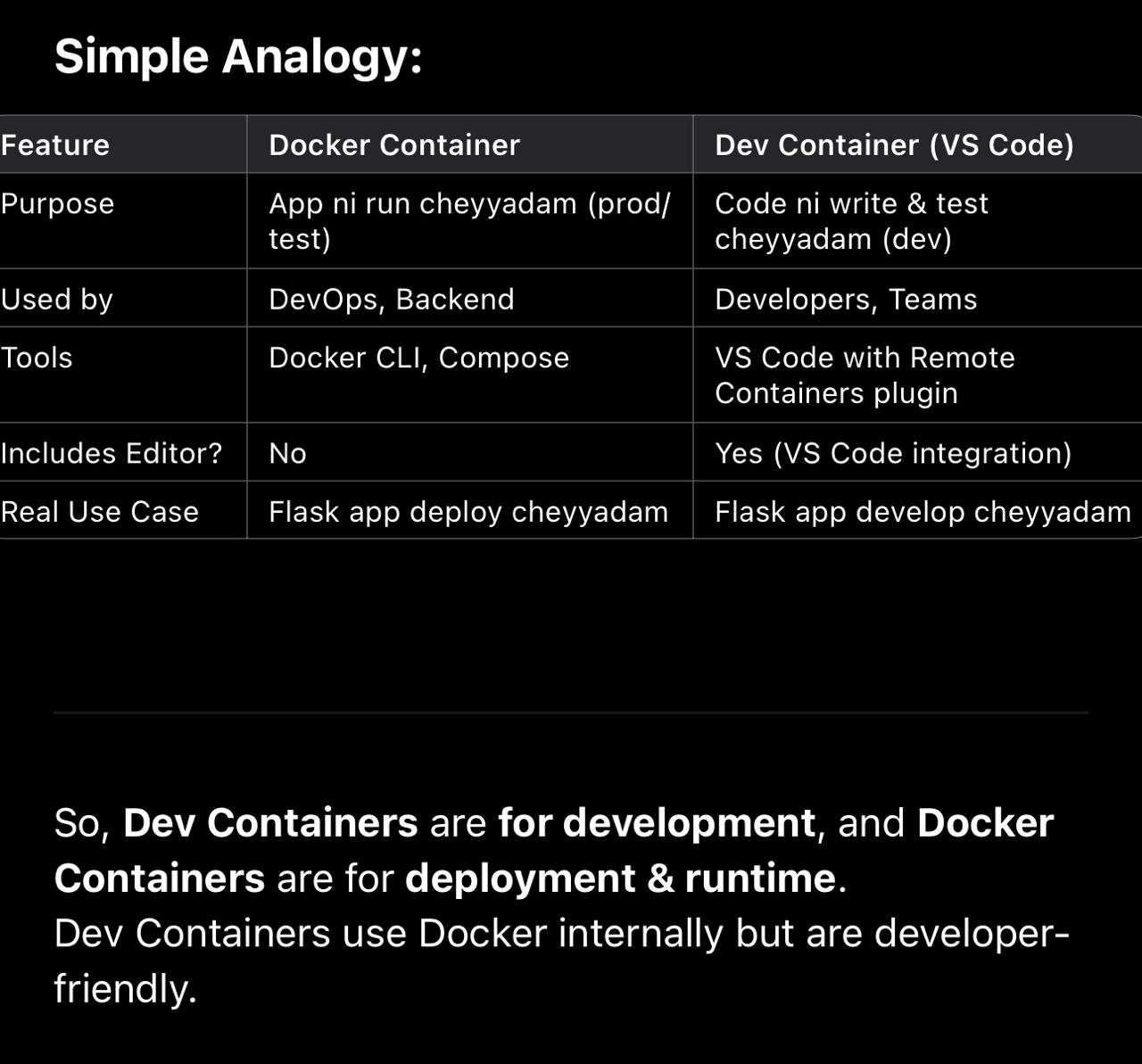
Dev Container ante development environment ni container lo set cheyyadam using a **.devcontainer** folder. Idi oka extension mainly VS Code lo use chestaru.

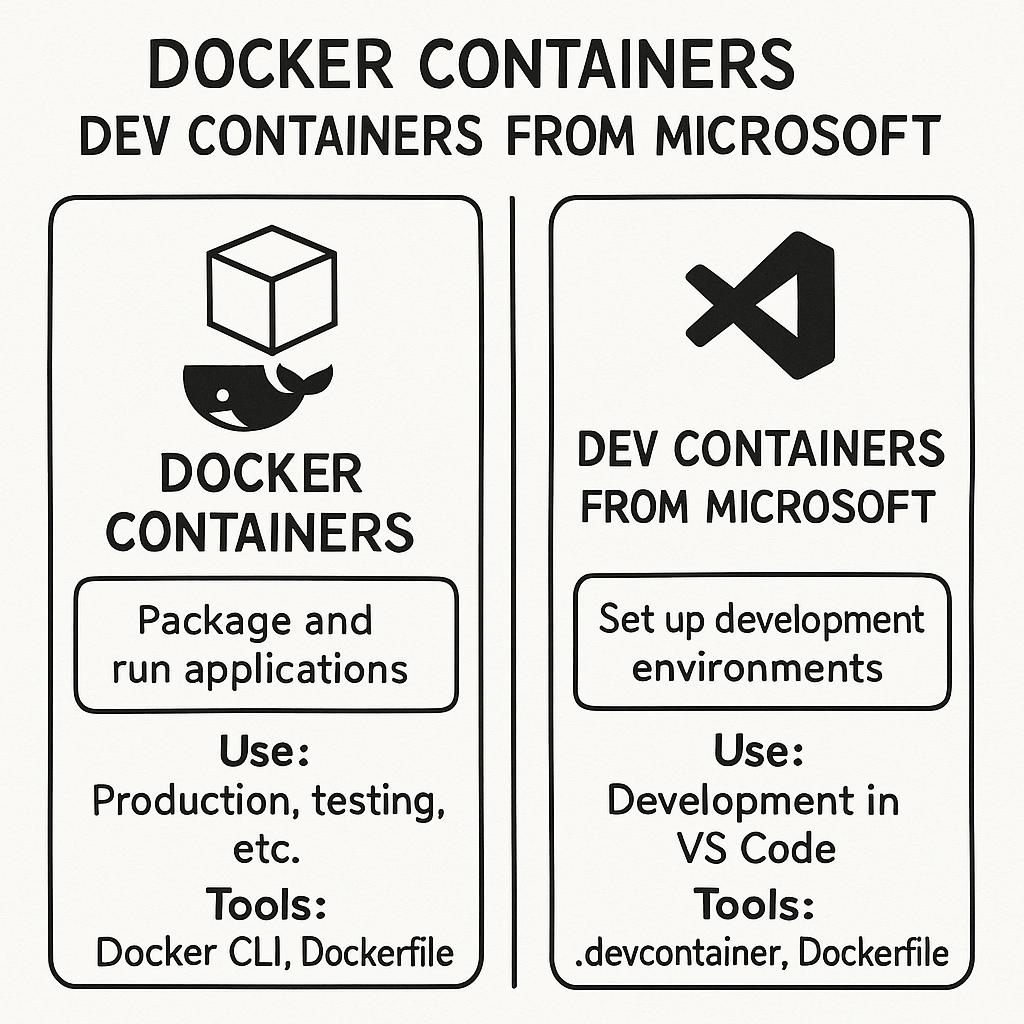
**Use:**

* **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.



****