📄 Dockerfile Command Reference

Each command in a Dockerfile defines a specific instruction for how to **build a Docker image**. Here's a detailed breakdown:

1️⃣ `FROM` — Set the Base Image 🏗️

```
FROM node:20-alpine
```

☑️ **What it does**: Defines the **base image** your custom image will build on top of.

> 📝 Must be the **first** instruction in the Dockerfile (except `ARG` sometimes).

2️⃣ `WORKDIR` — Set Working Directory 📁

```
WORKDIR /app
```

☑️ **What it does**: Sets the **working directory** inside the container where all subsequent commands (`COPY`, `RUN`, etc.) will execute.

> 📝 Automatically creates the directory if it doesn't exist.

3️⃣ `COPY` — Copy Files into the Image 📦

```
COPY package*.json ./
COPY . .
```

☑️ **What it does**: Copies files/directories from your host into the container filesystem.

> 💡 Use `.dockerignore` to exclude unnecessary files.

4️⃣ `RUN` — Execute a Shell Command 🛠️

```
RUN npm install
RUN apk add --no-cache curl
```

☑ **What it does**: Runs a command (like installing packages, cleaning up files) at **build time**, and creates a new layer.

> 💡 Combine multiple commands to reduce image size:

```
RUN apk add --no-cache curl && npm install
```

---

5 CMD — Default Command to Run on Container Start 🏁

```
CMD ["npm", "start"]
```

☑ **What it does**: Specifies the default command that gets run **when the container starts**.

> 💡 Only one `CMD` allowed — if multiple, only the last is used.

---

6 EXPOSE — Document the App's Port 🌐

```
EXPOSE 8000
```

☑ **What it does**: Informs Docker that the app runs on a specific port — used for documentation and port binding.

> 💡 Does **not** actually publish the port (use `-p` in `docker run`).

---

7 ENV — Set Environment Variables 🧪

```
ENV PORT=8000
ENV NODE_ENV=production
```

☑ **What it does**: Defines environment variables inside the container, usable by the app or shell.

> 💡 Use `process.env.PORT` in Node.js.

---

8 ARG — Build-Time Variables ⚙️

```
ARG NODE_VERSION=20-alpine
FROM node:${NODE_VERSION}
```

☑ **What it does**: Defines variables that you can pass during `docker build` with `--build-arg`.

> 📝 Unlike `ENV`, `ARG` is **not available** during runtime.

---

9 `ENTRYPOINT` — Hard-Coded Command 🚀

```
ENTRYPOINT ["node", "index.js"]
```

☑ **What it does**: Defines the main executable — like `CMD`, but **less override-able**.

> 💡 Combine with `CMD` for arguments:

```
ENTRYPOINT ["npm"]
CMD ["start"]
```

---

10 `LABEL` — Metadata About the Image 🏷️

```
LABEL maintainer="Darshan Vasani"
LABEL version="1.0"
```

☑ **What it does**: Adds metadata like author, version, description to your image.

> 💡 Useful for scanning tools and image documentation.

🔁 `VOLUME` — Define Persistent Storage Volumes 💾

```
VOLUME ["/app/data"]
```

☑ **What it does**: Creates a mount point with a persistent volume at build/run time.

> 💡 Avoid putting app logic here; use it for data.

🌍 `USER` — Switch to a Non-root User 🔐

```
RUN adduser -D myuser
USER myuser
```

☑ **What it does**: Sets the user under which the container runs (for security best practices).

> 💡 Don't run production apps as `root`.

🧪 BONUS: Multi-Stage Build Example 🧠

```
# Stage 1: Build
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .

# Stage 2: Runtime
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app .
CMD ["npm", "start"]
```

☑ Why?

- Keep final image clean
- Remove dev dependencies
- Build-only stuff stays out of final image

☑ Summary Table

Command	Purpose
`FROM`	Set base image
`WORKDIR`	Set working directory
`COPY`	Copy files into image
`RUN`	Execute shell commands at build time
`CMD`	Run command on container start
`EXPOSE`	Document listening port
`ENV`	Set environment variables
`ARG`	Pass build-time variables
`ENTRYPOINT`	Set main executable
`LABEL`	Add metadata to image
`USER`	Switch to non-root user
`VOLUME`	Declare persistent storage
