# Dockerfile Image build recipe

**START HERE**

**`FROM <image>:<tag>`**
Pick a base image

**`ENV <key>=<value>`**
Set an environment variable

**`WORKDIR /app`**
Set the working directory

**`EXPOSE <port>`**
Document a listening port

**`COPY <src> <dest>`**
Add files from the context

**`USER <user>`**
Drop root

**`RUN <command>`**
Run a command at build time

**`CMD ["exe", "arg"]`**
Default start command

**Base image** FROM → **Layers** one per step (RUN / COPY) → **Final stage** what ships (cache) → **Container** running process (CMD)

## CONCEPTS

Each instruction adds a layer, and a changed layer invalidates every layer after it. Put rarely changing steps first.

RUN, COPY and ADD happen at build time. CMD and ENTRYPOINT only describe what runs when a container starts.

## BASE AND STAGES

**`FROM <image>:<tag>`**
Start from a base image
*Without a tag you get latest. Pin a tag or an @sha256 digest for repeatable builds.*

**`FROM <image> AS <name>`**
Name a build stage

**`COPY --from=<stage> <src> <dest>`**
Copy files out of another stage

**`FROM scratch`**
Start from an empty image

**`FROM --platform=$BUILDPLATFORM <image>`**
Run the stage on the build host

**`FROM <image>@sha256:<digest>`**
Pin a base by digest

**`docker build --target <stage> .`**
Build only up to one stage

**`COPY --from=<image> <src> <dest>`**
Copy from an external image

**`ARG <name>=<default>`**
Before FROM: parametrize the base
*A global ARG is only usable in FROM lines. Redeclare it without a value inside a stage to use it there.*

## BUILD STEPS

**`RUN <command>`**
Run a command in a new layer

**`COPY <src> <dest>`**
Copy files from the build context

**`COPY --chown=<user>:<group> <src> <dest>`**
Copy and set owner

**`COPY --chmod=<mode> <src> <dest>`**
Copy and set permissions

**`COPY --link <src> <dest>`**
Copy into an independent layer

**`ADD <url> <dest>`**
Download a remote file

## BUILD STEPS (cont.)

**`ADD <archive>.tar.gz <dest>`**
Copy and auto-extract a local tar
*For plain files prefer COPY. ADD is for tar auto-extract and URLs.*

**`RUN --mount=type=cache, target=<dir> <command>`**
Reuse a package cache between builds

**`RUN --mount=type=secret,id=<id> <command>`**
Use a secret without baking it in
*Read it from /run/secrets/<id>. Pass it with docker build --secret id=<id>,src=<file>.*

**`RUN --mount=type=bind, source=<src>,target=<dir> <command>`**
Read context files without copying

**`RUN --mount=type=ssh <command>`**
Use the host SSH agent

**`RUN a && b && c`**
Chain steps in one layer

## CONFIGURATION

**`ENV <key>=<value>`**
Set a variable for build and runtime

**`ARG <name>[=<default>]`**
Declare a build-time variable
*Set with docker build --build-arg <name>=<value>. Not for secrets: values show up in image history.*

**`WORKDIR <path>`**
Set the directory for later steps
*Creates the directory if needed. Prefer it over RUN cd.*

**`USER <user>[:<group>]`**
Run later steps as this user

**`LABEL <key>=<value>`**
Attach image metadata

**`SHELL ["/bin/bash", "-c"]`**
Change the shell for shell-form steps

**`RUN useradd -r -u 10001 <user>`**
Create an unprivileged user

## RUNTIME

**`EXPOSE <port>[/tcp|/udp]`**
Document a port
*This does not publish anything. Use -p or -P at docker run.*

**`CMD ["exe", "arg"]`**
Default command (exec form)

**`ENTRYPOINT ["exe"]`**
Fixed executable for the container

## RUNTIME (cont.)

**`CMD ["arg1", "arg2"]`**
Default arguments for ENTRYPOINT

**`CMD <command> <arg>`**
Default command via /bin/sh -c
*Shell form runs under sh -c, so signals do not reach your process. Prefix with exec or use exec form.*

**`VOLUME ["<path>"]`**
Declare a mount point

**`HEALTHCHECK CMD <command>`**
Probe container health

**`HEALTHCHECK --interval=<d> --timeout=<d> --retries=<n> CMD <command>`**
Tune the probe
*Defaults: interval 30s, timeout 30s, retries 3. Exit 0 is healthy, 1 unhealthy.*

### ENTRYPOINT and CMD

| Dockerfile | Runs |
|---|---|
| `CMD ["a"]` | a |
| `ENTRYPOINT ["e"]` | e |
| `ENTRYPOINT ["e"] + CMD ["a"]` | e a |
| `docker run <image> x` | replaces CMD |
| `docker run --entrypoint <p> <image>` | replaces ENTRYPOINT |

## SYNTAX AND CONTEXT

**`# syntax=docker/dockerfile:1`**
Parser directive: current stable frontend
*Must be the very first line of the file.*

**`RUN <<EOF`**
Multi-line script with a heredoc

**`<command> \`**
Continue a line

**`# comment`**
Comment line

**`docker build -f <file> <context>`**
Use another Dockerfile

**`docker build -t <name> .`**
Build with a tag

**`node_modules`**
In .dockerignore: skip a path

**`**/*.log`**
In .dockerignore: match at any depth

**`!<pattern>`**
In .dockerignore: re-include a match
*The last matching line wins, so put exceptions after the pattern they carve out.*

Keep .git, node_modules and build output out of the context: it makes builds faster and keeps COPY . from invalidating the cache.

## CACHE AND SIZE

Copy dependency manifests first, install, then copy the rest of the source. Code edits then do not re-run the install.

Use a multi-stage build: compile in a fat stage, copy only the artifact into a slim final stage.

Clean package manager caches in the same RUN that created them. A later RUN cannot shrink an earlier layer.

Never put secrets in ARG, ENV or COPY: they stay in the layers. Use RUN --mount=type=secret.

Prefer exec form for CMD and ENTRYPOINT so the process gets SIGTERM from docker stop.

Group changing steps last and pin versions of base images and packages for reproducible builds.

## RECIPES

### Cache-friendly Node image

```
FROM node:22-slim
WORKDIR /app
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm npm ci
COPY . .
CMD ["node", "server.js"]
```

### Multi-stage Go binary

```
FROM golang:1.23 AS build
WORKDIR /src
COPY . .
RUN CGO_ENABLED=0 go build -o /out/app .

FROM gcr.io/distroless/static
COPY --from=build /out/app /app
USER 65532
ENTRYPOINT ["/app"]
```

### Entrypoint script that forwards signals

```
COPY entrypoint.sh /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
CMD ["serve"]
# last line of entrypoint.sh:
# exec "$@"
```

### Heredoc script with a build secret

```
RUN --mount=type=secret,id=token <<EOF
  set -e
  curl -fsS -H "Authorization: Bearer $(cat /run/secrets/token)" <url> -o /tmp/pkg
EOF
```