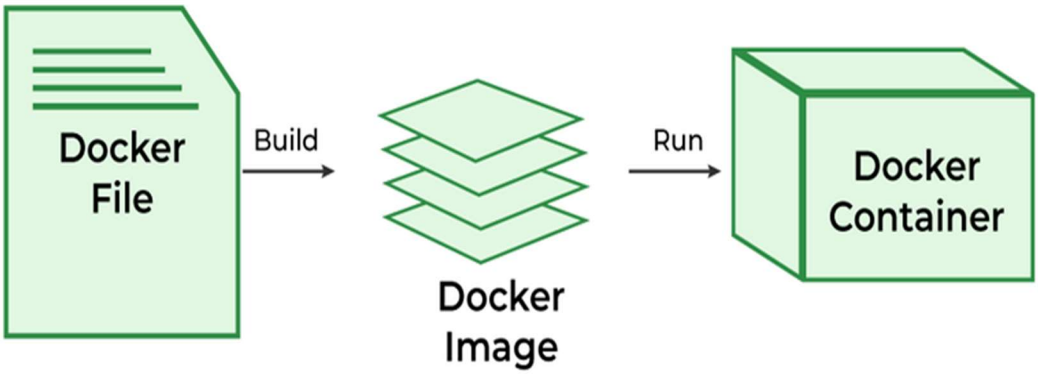
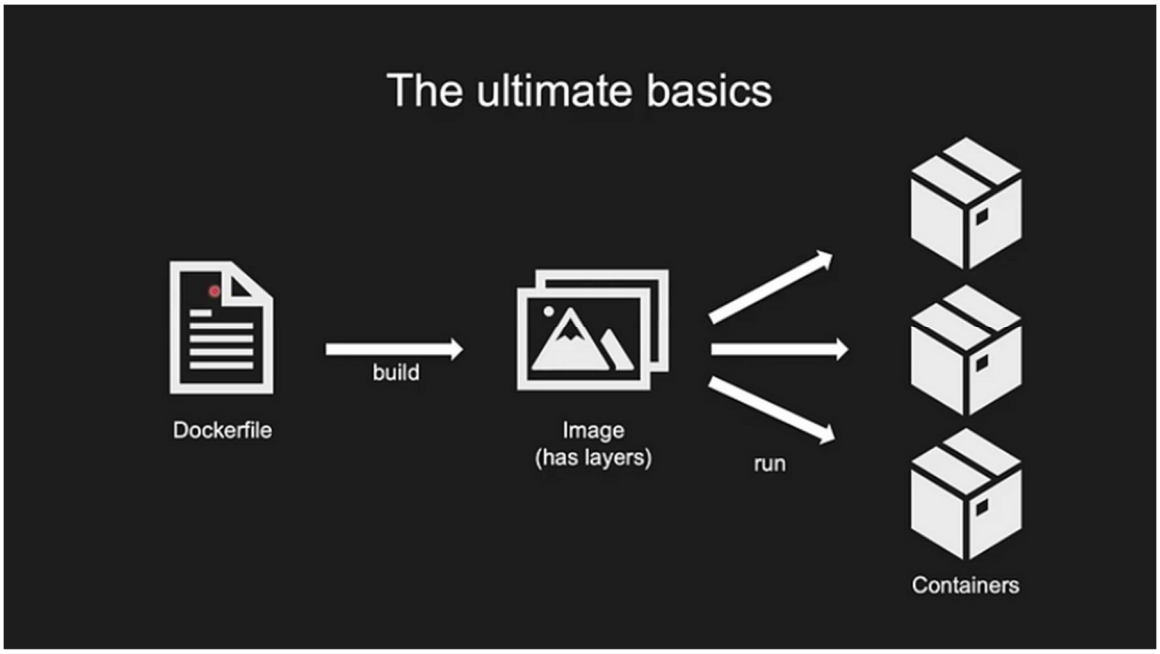



# Dockerfile Complete Guide

Davoud Azari

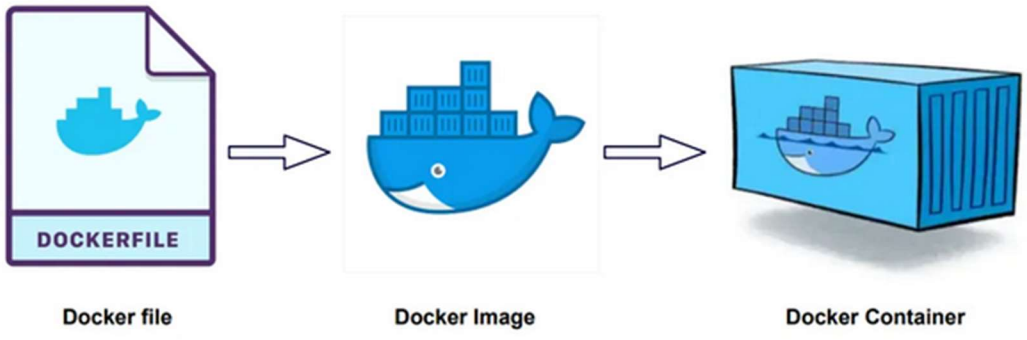
# Dockerfile Complete Guide

## Contents




A Dockerfile is a text document that contains instructions for Docker to build an image. These instructions define what goes into the image, how it should be built, and how it will run. Dockerfiles are used to automate the process of image creation, which can be reused and shared across environments.

| Instruction | Description |
|---|---|
| ADD | Adds local or remote files and directories. |
| ARG | Defines build-time variables. |
| CMD | Specifies default commands to run when the container starts. |
| COPY | Copies files and directories from the build context to the container's filesystem. |
| ENTRYPOINT | Specifies the default executable to run when the container starts. |
| ENV | Sets environment variables that will be available during the container's runtime. |
| EXPOSE | Documents which ports the application inside the container will listen on. |
| FROM | Creates a new build stage from a base image. |
| HEALTHCHECK | Defines a command to test whether the container is still working correctly. |
| LABEL | Adds metadata to the image, such as a description, version, or other relevant information. |
| MAINTAINER | Specifies the author of the image (deprecated in favor of LABEL for maintainership). |
| ONBUILD | Specifies instructions to be executed when the image is used as the base for another build. |
| RUN | Executes commands inside the container during the image build process. |
| SHELL | Sets the default shell to use for shell form commands like RUN, CMD, and ENTRYPOINT. |
| STOPSIGNAL | Specifies the system call signal that will be sent to the container to exit. |
| USER | Sets the user and group ID to use when running the container. |
| VOLUME | Creates mount points with specified paths for external volume mounts. |
| WORKDIR | Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions. |




Dockerfile

## Dockerfile Instructions

### FROM

The FROM instruction defines the base image to use for the Docker image. It must be the first non-comment instruction in a Dockerfile. If you do not specify a version or tag, Docker will use the latest tag by default.

```
FROM ubuntu:20.04
```

### ADD

The ADD instruction allows you to copy files or directories from your local file system or from a URL into your Docker image. Additionally, it can automatically extract tar archives.

```
ADD ./source.tar.gz /app
```

### ARG

The ARG instruction defines build-time variables. These variables can be passed to the build process using the --build-arg flag in the docker build command.

```
ARG VERSION=1.0
```

### CMD

The CMD instruction defines the default command that will run when a container is started from the image. There can only be one CMD instruction, and if multiple CMD instructions are specified, only the last one is used.

```
CMD ["python", "app.py"]
```

### COPY

The COPY instruction is used to copy files or directories from the host system into the image. Unlike ADD, it does not have any special functionality like extracting tar files or fetching remote files.

```
COPY ./src /app
```

### ENTRYPOINT

The ENTRYPOINT instruction allows you to specify the command that will always be executed when the container starts. It is often combined with CMD to define the default executable and its arguments.

```
ENTRYPOINT ["python"]

CMD ["app.py"]
```

### ENV

The ENV instruction sets environment variables that will be available to the running container. These variables can be accessed by programs running inside the container.

```
ENV APP_NAME="MyApp"
```

### EXPOSE

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. It does not open these ports on the host machine, but it provides a documentation for which ports are used.

```
EXPOSE 80
```

### HEALTHCHECK

The HEALTHCHECK instruction defines a command to check the health of the container. If the command returns a non-zero exit code, the container will be considered unhealthy.

```
HEALTHCHECK CMD curl --fail http://localhost/ || exit 1
```

### LABEL

The LABEL instruction adds metadata to an image, such as the author, version, or description. Labels are key-value pairs.

```
LABEL maintainer="davoudazari@example.com"

LABEL version="1.0"
```

### MAINTAINER

The MAINTAINER instruction specifies the author of the image. This instruction has been deprecated in favor of LABEL for maintainership.

```
maintainer="davoudazari@example.com"
```

## ONBUILD

The ONBUILD instruction adds a trigger instruction to be executed when the image is used as a base for another build. These instructions are inherited by child builds but not by grandchild builds.

```
FROM node:14
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

ONBUILD COPY package.json /usr/src/app/
ONBUILD RUN npm install
ONBUILD COPY . /usr/src/app

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

* **FROM node:14:**
This starts with the official Node.js 14 image as the base for the Docker image. It ensures that the container will have a Node.js environment.

- **RUN mkdir -p /usr/src/app**:
This creates a directory `/usr/src/app` inside the container. The `-p` flag ensures that the parent directories are also created if they don't exist.

- **WORKDIR /usr/src/app**:
This sets the working directory inside the container to `/usr/src/app`. All subsequent commands will be executed relative to this directory.

- **ONBUILD COPY package.json /usr/src/app/**:
This is an **ONBUILD** instruction, which means it will only be triggered when this image is used as a base image for another build. It copies the `package.json` file into the `/usr/src/app` directory of the container.

- **ONBUILD RUN npm install**:
This **ONBUILD** instruction will run `npm install` to install the dependencies defined in the `package.json` file. Again, this will only run when this image is used as the base image for a child build.

- **ONBUILD COPY . /usr/src/app**:
This copies all the application files from the current directory into the `/usr/src/app` directory inside the container. This will also only happen when this image is used as a base image.

- **CMD [ "npm", "start" ]**:
This sets the default command to run when a container is started from this image. In this case, it runs `npm start`, which will start the application (assuming the `start` script is defined in the `package.json` file).

## RUN

The RUN instruction is used to execute commands during the image build process. It can be used to install software or configure the container.

```
RUN apt-get update && apt-get install -y python3
```

### SHELL

The SHELL instruction allows you to specify the default shell to use for shell form commands like RUN, CMD, and ENTRYPOINT.

```
SHELL ["/bin/bash", "-c"]
```

### STOPSIGNAL

The STOPSIGNAL instruction specifies which signal Docker should use to stop the container. This can be a signal name (e.g., SIGTERM) or a signal number.

```
STOPSIGNAL SIGTERM
```

### USER

The USER instruction sets the user and group to use when running the container. This is useful for controlling the permissions and access inside the container.

```
USER appuser
```

### VOLUME

The VOLUME instruction creates a mount point with the specified path and marks it as holding volumes that can be mounted from outside the container.

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

### WORKDIR

The WORKDIR instruction sets the working directory for subsequent instructions like RUN, CMD, ENTRYPOINT, COPY, and ADD.

```
WORKDIR /app
```

## Best Practices for Writing Dockerfiles

- **Use official images**: Always start with an official base image (e.g., ubuntu, python) to ensure compatibility and security.
- **Leverage layer caching**: Docker caches layers to speed up builds. Reorder your instructions to ensure that more static instructions (like installing dependencies) come first.
- **Use COPY over ADD**: Unless you specifically need to extract tar files or fetch files from remote URLs, prefer COPY for copying files into the image.
- **Minimize image size**: Remove unnecessary files and install only the essential packages to reduce the size of the final image.
- **Avoid hardcoding values**: Use ARG and ENV for any values that might change based on the environment or build context.
- **Use `&&` to reduce layers**: When combining multiple commands in a `RUN` instruction, use `&&` to chain them together. This reduces the number of layers created and results in a smaller image size.

### Explanation

A Dockerfile is a text-based document that's used to create a container image. It provides instructions to the image builder on the commands to run, files to copy, startup command, and more.

As an example, the following Dockerfile would produce a ready-to-run Python application:

```
FROM python:3.12
WORKDIR /usr/local/app

# Install the application dependencies
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy in the source code
COPY src ./src
EXPOSE 5000

# Setup an app user so the container doesn't run as the root user
RUN useradd app
USER app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

## Common instructions

Some of the most common instructions in a `Dockerfile` include:

- **`FROM <image>`** - this specifies the base image that the build will extend.
- **`WORKDIR <path>`** - this instruction specifies the "working directory" or the path in the image where files will be copied and commands will be executed.
- **`COPY <host-path> <image-path>`** - this instruction tells the builder to copy files from the host and put them into the container image.
- **`RUN <command>`** - this instruction tells the builder to run the specified command.
- **`ENV <name> <value>`** - this instruction sets an environment variable that a running container will use.
- **`EXPOSE <port-number>`** - this instruction sets configuration on the image that indicates a port the image would like to expose.
- **`USER <user-or-uid>`** - this instruction sets the default user for all subsequent instructions.
- **`CMD ["<command>", "<arg1>"]`** - this instruction sets the default command a container using this image will run.

**The Dockerfile example and explanation were taken from the official Docker documentation: Writing a Dockerfile.**