Dockerfile

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfileis a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

This page describes the commands you can use in a Dockerfile. When you are done reading this page, refer to the [Dockerfile Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/) for a tip-oriented guide.

Usage

The [docker build](https://docs.docker.com/engine/reference/commandline/build/) command builds an image from a Dockerfile and a *context*. The build’s context is the set of files at a specified location PATH or URL. The PATH is a directory on your local filesystem. The URL is a Git repository location.

A context is processed recursively. So, a PATH includes any subdirectories and the URL includes the repository and its submodules. This example shows a build command that uses the current directory as context:

$ docker build .

Sending build context to Docker daemon 6.51 MB

...

The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it’s best to start with an empty directory as context and keep your Dockerfile in that directory. Add only the files needed for building the Dockerfile.

**Warning**: Do not use your root directory, /, as the PATH as it causes the build to transfer the entire contents of your hard drive to the Docker daemon.

To use a file in the build context, the Dockerfile refers to the file specified in an instruction, for example, a COPY instruction. To increase the build’s performance, exclude files and directories by adding a .dockerignore file to the context directory. For information about how to [create a .dockerignore file](https://docs.docker.com/engine/reference/builder/#dockerignore-file) see the documentation on this page.

Traditionally, the Dockerfile is called Dockerfile and located in the root of the context. You use the -f flag with docker build to point to a Dockerfile anywhere in your file system.

$ docker build -f /path/to/a/Dockerfile .

You can specify a repository and tag at which to save the new image if the build succeeds:

$ docker build -t shykes/myapp .

To tag the image into multiple repositories after the build, add multiple -t parameters when you run the build command:

$ docker build -t shykes/myapp:1.0.2 -t shykes/myapp:latest .

Before the Docker daemon runs the instructions in the Dockerfile, it performs a preliminary validation of the Dockerfile and returns an error if the syntax is incorrect:

$ docker build -t test/myapp .

Sending build context to Docker daemon 2.048 kB

Error response from daemon: Unknown instruction: RUNCMD

The Docker daemon runs the instructions in the Dockerfile one-by-one, committing the result of each instruction to a new image if necessary, before finally outputting the ID of your new image. The Docker daemon will automatically clean up the context you sent.

Note that each instruction is run independently, and causes a new image to be created - so RUN cd /tmp will not have any effect on the next instructions.

Whenever possible, Docker will re-use the intermediate images (cache), to accelerate the docker buildprocess significantly. This is indicated by the Using cache message in the console output. (For more information, see the [Build cache section](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#build-cache) in the Dockerfile best practices guide):

$ docker build -t svendowideit/ambassador .

Sending build context to Docker daemon 15.36 kB

Step 1/4 : FROM alpine:3.2

---> 31f630c65071

Step 2/4 : MAINTAINER SvenDowideit@home.org.au

---> Using cache

---> 2a1c91448f5f

Step 3/4 : RUN apk update && apk add socat && rm -r /var/cache/

---> Using cache

---> 21ed6e7fbb73

Step 4/4 : CMD env | grep \_TCP= | (sed 's/.\*\_PORT\_\([0-9]\*\)\_TCP=tcp:\/\/\(.\*\):\(.\*\)/socat -t 100000000 TCP4-LISTEN:\1,fork,reuseaddr TCP4:\2:\3 \&/' && echo wait) | sh

---> Using cache

---> 7ea8aef582cc

Successfully built 7ea8aef582cc

|  |
| --- |
| Dockerfile:  FROM  Usage:   * FROM <image> * FROM <image>:<tag> * FROM <image>@<digest>   Information:   * FROM must be the first non-comment instruction in the Dockerfile. * FROM can appear multiple times within a single Dockerfile in order to create multiple images. Simply make a note of the last image ID output by the commit before each new FROM command. * The tag or digest values are optional. If you omit either of them, the builder assumes a latest by default. The builder returns an error if it cannot match the tag value.   [Reference](https://docs.docker.com/engine/reference/builder/#from) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#from) |
| MAINTAINER  Usage:   * MAINTAINER <name>   The MAINTAINER instruction allows you to set the Author field of the generated images.  [Reference](https://docs.docker.com/engine/reference/builder/#maintainer) |
| RUN  Usage:   * RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on Linux or cmd /S /C on Windows * RUN ["<executable>", "<param1>", "<param2>"] (exec form)   Information:   * The exec form makes it possible to avoid shell string munging, and to RUN commands using a base image that does not contain the specified shell executable. * The default shell for the shell form can be changed using the SHELL command. * Normal shell processing does not occur when using the exec form. For example, RUN ["echo", "$HOME"] will not do variable substitution on $HOME.   [Reference](https://docs.docker.com/engine/reference/builder/#run) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#run) |
| CMD  Usage:   * CMD ["<executable>","<param1>","<param2>"] (exec form, this is the preferred form) * CMD ["<param1>","<param2>"] (as default parameters to ENTRYPOINT) * CMD <command> <param1> <param2> (shell form)   Information:   * The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well. * There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect. * If CMD is used to provide default arguments for the ENTRYPOINT instruction, both the CMD and ENTRYPOINT instructions should be specified with the JSON array format. * If the user specifies arguments to docker run then they will override the default specified in CMD. * Normal shell processing does not occur when using the exec form. For example, CMD ["echo", "$HOME"] will not do variable substitution on $HOME.   [Reference](https://docs.docker.com/engine/reference/builder/#cmd) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#cmd) |
| LABEL  Usage:   * LABEL <key>=<value> [<key>=<value> ...]   Information:   * The LABEL instruction adds metadata to an image. * To include spaces within a LABEL value, use quotes and backslashes as you would in command-line parsing. * Labels are additive including LABELs in FROM images. * If Docker encounters a label/key that already exists, the new value overrides any previous labels with identical keys. * To view an image’s labels, use the docker inspect command. They will be under the "Labels" JSON attribute.   [Reference](https://docs.docker.com/engine/reference/builder/#label) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#label) |
| EXPOSE  Usage:   * EXPOSE <port> [<port> ...]   Information:   * Informs Docker that the container listens on the specified network port(s) at runtime. * EXPOSE does not make the ports of the container accessible to the host.   [Reference](https://docs.docker.com/engine/reference/builder/#expose) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#expose) |
| ENV  Usage:   * ENV <key> <value> * ENV <key>=<value> [<key>=<value> ...]   Information:   * The ENV instruction sets the environment variable <key> to the value <value>. * The value will be in the environment of all “descendant” Dockerfile commands and can be replaced inline as well. * The environment variables set using ENV will persist when a container is run from the resulting image. * The first form will set a single variable to a value with the entire string after the first space being treated as the <value> - including characters such as spaces and quotes.   [Reference](https://docs.docker.com/engine/reference/builder/#env) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#env) |
| ADD  Usage:   * ADD <src> [<src> ...] <dest> * ADD ["<src>", ... "<dest>"] (this form is required for paths containing whitespace)   Information:   * Copies new files, directories, or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>. * <src> may contain wildcards and matching will be done using Go’s filepath.Match rules. * If <src> is a file or directory, then they must be relative to the source directory that is being built (the context of the build). * <dest> is an absolute path, or a path relative to WORKDIR. * If <dest> doesn’t exist, it is created along with all missing directories in its path.   [Reference](https://docs.docker.com/engine/reference/builder/#add) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#add-or-copy) |
| COPY  Usage:   * COPY <src> [<src> ...] <dest> * COPY ["<src>", ... "<dest>"] (this form is required for paths containing whitespace)   Information:   * Copies new files or directories from <src> and adds them to the filesystem of the image at the path <dest>. * <src> may contain wildcards and matching will be done using Go’s filepath.Match rules. * <src> must be relative to the source directory that is being built (the context of the build). * <dest> is an absolute path, or a path relative to WORKDIR. * If <dest> doesn’t exist, it is created along with all missing directories in its path.   [Reference](https://docs.docker.com/engine/reference/builder/#copy) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#add-or-copy) |
| ENTRYPOINT  Usage:   * ENTRYPOINT ["<executable>", "<param1>", "<param2>"] (exec form, preferred) * ENTRYPOINT <command> <param1> <param2> (shell form)   Information:   * Allows you to configure a container that will run as an executable. * Command line arguments to docker run <image> will be appended after all elements in an exec form ENTRYPOINT and will override all elements specified using CMD. * The shell form prevents any CMD or run command line arguments from being used, but the ENTRYPOINT will start via the shell. This means the executable will not be PID 1 nor will it receive UNIX signals. Prepend exec to get around this drawback. * Only the last ENTRYPOINT instruction in the Dockerfile will have an effect.   [Reference](https://docs.docker.com/engine/reference/builder/#entrypoint) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#entrypoint) |
| VOLUME  Usage:   * VOLUME ["<path>", ...] * VOLUME <path> [<path> ...]   Creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.  [Reference](https://docs.docker.com/engine/reference/builder/#volume) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#volume) |
| USER  Usage:   * USER <username | UID>   The USER instruction sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.  [Reference](https://docs.docker.com/engine/reference/builder/#user) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#user) |
| WORKDIR  Usage:   * WORKDIR </path/to/workdir>   Information:   * Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow it. * It can be used multiple times in the one Dockerfile. If a relative path is provided, it will be relative to the path of the previous WORKDIR instruction.   [Reference](https://docs.docker.com/engine/reference/builder/#workdir) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#workdir) |
| ARG  Usage:   * ARG <name>[=<default value>]   Information:   * Defines a variable that users can pass at build-time to the builder with the docker build command using the --build-arg <varname>=<value> flag. * Multiple variables may be defined by specifying ARG multiple times. * It is not recommended to use build-time variables for passing secrets like github keys, user credentials, etc. Build-time variable values are visible to any user of the image with the docker history command. * Environment variables defined using the ENV instruction always override an ARG instruction of the same name. * Docker has a set of predefined ARG variables that you can use without a corresponding ARG instruction in the Dockerfile.   + HTTP\_PROXY and http\_proxy   + HTTPS\_PROXY and https\_proxy   + FTP\_PROXY and ftp\_proxy   + NO\_PROXY and no\_proxy   [Reference](https://docs.docker.com/engine/reference/builder/#arg) |
| ONBUILD  Usage:   * ONBUILD <Dockerfile INSTRUCTION>   Information:   * Adds to the image a trigger instruction to be executed at a later time, when the image is used as the base for another build. The trigger will be executed in the context of the downstream build, as if it had been inserted immediately after the FROM instruction in the downstream Dockerfile. * Any build instruction can be registered as a trigger. * Triggers are inherited by the "child" build only. In other words, they are not inherited by "grand-children" builds. * The ONBUILD instruction may not trigger FROM, MAINTAINER, or ONBUILD instructions.   [Reference](https://docs.docker.com/engine/reference/builder/#onbuild) - [Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#onbuild) |
| STOPSIGNAL  Usage:   * STOPSIGNAL <signal>   The STOPSIGNAL instruction sets the system call signal that will be sent to the container to exit. This signal can be a valid unsigned number that matches a position in the kernel’s syscall table, for instance 9, or a signal name in the format SIGNAME, for instance SIGKILL.  [Reference](https://docs.docker.com/engine/reference/builder/#stopsignal) |
| HEALTHCHECK  Usage:   * HEALTHCHECK [<options>] CMD <command> (check container health by running a command inside the container) * HEALTHCHECK NONE (disable any healthcheck inherited from the base image)   Information:   * Tells Docker how to test a container to check that it is still working * Whenever a health check passes, it becomes healthy. After a certain number of consecutive failures, it becomes unhealthy. * The <options> that can appear are...   + --interval=<duration> (default: 30s)   + --timeout=<duration> (default: 30s)   + --retries=<number> (default: 3) * The health check will first run interval seconds after the container is started, and then again interval seconds after each previous check completes. If a single run of the check takes longer than timeout seconds then the check is considered to have failed. It takes retries consecutive failures of the health check for the container to be considered unhealthy. * There can only be one HEALTHCHECK instruction in a Dockerfile. If you list more than one then only the last HEALTHCHECK will take effect. * <command> can be either a shell command or an exec JSON array. * The command's exit status indicates the health status of the container.   + 0: success - the container is healthy and ready for use   + 1: unhealthy - the container is not working correctly   + 2: reserved - do not use this exit code * The first 4096 bytes of stdout and stderr from the <command> are stored and can be queried with docker inspect. * When the health status of a container changes, a health\_status event is generated with the new status.   [Reference](https://docs.docker.com/engine/reference/builder/#healthcheck) |
| SHELL  Usage:   * SHELL ["<executable>", "<param1>", "<param2>"]   Information:   * Allows the default shell used for the shell form of commands to be overridden. * Each SHELL instruction overrides all previous SHELL instructions, and affects all subsequent instructions. * Allows an alternate shell be used such as zsh, csh, tcsh, powershell, and others. |