Dockerfile

Enjoy this cheat sheet at its fullest within Dash, the macOS documentation browser.

Reference - Best Practices

Instructions

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 - Best Practices

MAINTAINER

Usage:

- `MAINTAINER <name>`

The `MAINTAINER` instruction allows you to set the Author field of the generated images.

Reference

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 - Best Practices

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 - Best Practices

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 `LABEL`s 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 - Best Practices

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 - Best Practices

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 - Best Practices

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 - Best Practices

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 - Best Practices

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 - Best Practices

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 - Best Practices

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 - Best Practices

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 - Best Practices

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

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 - Best Practices

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

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

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.

Reference

Notes

- Based on the information from Dockerfile reference and Docker file best practices.
- Converted by halprin.

You can modify and improve this cheat sheet here