DOCKERFILE QUICK REFERENCE

See `$ man dockerfile` for detailed reference

FROM *<image>*
FROM *<image>:<tag>*
: Sets the base image for subsequent instructions. Dockerfile must start with `FROM` instruction.

MAINTAINER *<name>*
: Sets the Author field for the generated images

LABEL ...
: Adds metadata to an image. A label is a key-value pair

  LABEL *<key>=<value> <key2>=<value2> ...*
  LABEL *<key> <value>*

RUN ...
: Executes any commands in a new layer on top of the current image and commits the results. There are two forms:

  RUN *<command>* Run the command in the shell `/bin/sh -c`
  RUN *["<executable>", "<param1>", "<param2>"]*
  : Executable form. The square brackets are a part of the syntax

CMD ...
: Provides defaults for executing container. There could be at most one `CMD` instruction in a Dockerfile

  CMD *["<executable>", "<param1>", "<param2>"]*
  : Executable form
  CMD *["<param1>", "<param2>"]*
  : Provide default arguments to `ENTRYPOINT`
  CMD *<command args ...>*
  : Run the command in the shell `/bin/sh -c`

ENTRYPOINT ...
: Helps you configure a container that can be run as an executable. The `ENTRYPOINT` instruction adds an entry command that is not overwritten when arguments are passed to docker run. This is different from the behavior of `CMD`. This allows arguments to be passed to the entrypoint

  ENTRYPOINT *["<executable>", "<param1>", "<param2>"]*
  : Executable form
  ENTRYPOINT *<command param1 param2 ...>*
  : Run the command in the shell `/bin/sh -c`

EXPOSE *<port1> <port2> ...*
: Informs Docker that the container listens on the specified network ports at runtime. Docker uses this information to interconnect containers using links and to set up port redirection on the host system

ENV *<key> <value>*
: Sets the environment variable `<key>` to the value `<value>`. This value is passed to all future `RUN`, `ENTRYPOINT`, and `CMD` instructions

COPY *<src> <dest>*
COPY *["<src>", ... "<dest>"]*
: Copies new files, directories or remote file URLs to the filesystem of the container at path `<dest>`. All new files and directories are created with mode 0755 and with the uid and gid of 0.

ADD *<src> <dest>*
ADD *["<src>", ... "<dest>"]*
: Like `COPY`, but additionally allows `<src>` to be an URL, and if `<src>` is an archive in a recognized format, it will be unpacked. The best practice is to prefer `COPY`

VOLUME *["/some/path"]*
: Creates a mount point with the specified name and marks it as holding externally-mounted volumes from the native host or from other containers

USER *<user>*
USER *<user>:<group>*
: Sets the username or UID used for running subsequent commands. `<user>` can be either username or UID; `<group>` can be either group name or GID

WORKDIR */path/to/workdir*
: Sets the working directory for the `RUN`, `CMD`, `ENTRYPOINT`, `COPY` and `ADD` Dockerfile commands that follow. Relative paths are defined relative to the path of the previous `WORKDIR` instruction.

ARG *<name>*
ARG *<name>=<default value>*
: 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

ONBUILD *<instruction>*
: Adds a trigger instruction to an image. The trigger is executed at a later time, when the image is used as the base for another build. Docker executes the trigger in the context of the downstream build, as if the trigger existed immediately after the FROM instruction in the downstream Dockerfile.

DOCKER CLI QUICK REFERENCE

See `$ man docker-<command>` for detailed reference; e.g. `$ man docker-build`

Building images

$ docker build *[<opts>] <path> | <URL>*
: Build a new image from the source code at `PATH`

  -f, --file *path/to/Dockerfile*
  : Path to the Dockerfile to use. Default: `Dockerfile`.
  --build-arg *<varname>=<value>*
  : Name and value of a build argument defined with `ARG` Dockerfile instruction
  -t *"<name>[:<tag>]"*
  : Repository names (and optionally with tags) to be applied to the resulting image
  --label *=<label>*
  : Set metadata for an image
  -q, --quiet Suppress the output generated by containers
  --rm Remove intermediate containers after a successful build

Creating, running and stopping containers

$ docker run *[<opts>] <image> [<command>] [<arg>...]*
: Run a command in a new container

  -i, --interactive
  : Keep STDIN open even if not attached
  -t, --tty Allocate a pseudo-TTY
  -v, --volume *[<host-dir>:]<container-dir>[:<opts>]*
  : Bind mount a volume. Options are comma-separated: `[ro,rw]`. By default, `rw` is used.
  --device *=<host-dev>:<container-dev>[:<opts>]*
  : Add a host device to the container; e.g. `--device="/dev/sda:/dev/xvdc:rwm"`.
  Possible `<opts>` flags: `r`: read, `w`: write, `m`: mknod
  -d, --detach Detached (daemon) mode
  -e, --env *NAME[="value"]*
  : Set environment variable. If the `value` is omitted, the value will be taken from the current environment.
  --env-file *file*
  : Read in a line delimited file of environment variables
  --entrypoint *"some/entry/point"*
  : Overwrite the default `ENTRYPOINT` of the image
  -h, --hostname *="<hostname>"*
  : Container host name
  --add-host *=<hostname>:<ip>*
  : Add a custom host-to-IP mapping
  --net *="<mode>"*
  : Set the network mode for the container (default: `bridge`):
    - `bridge`: create a network stack on the default Docker bridge
    - `none`: no networking
    - `container:<name|id>`: reuse another container's stack
    - `host`: use the Docker host network stack
    - `<network-name>|<network-id>`: connect to a user-defined network

  --group-add *=<groups>*
  : Add additional groups to run as
  --rm Automatically remove the container when it exits
  --name *"foo"* Assign a name to the container
  --detach-keys *="<keys>"*
  : Override the key sequence to detach a container. Default: `"ctrl-p ctrl-q"`

$ docker create *[<opts>] <image> [<command>] [<arg>...]*
: Create a new container, but don't run it (instead, print its id). See options for `docker run`

$ docker start *[<opts>] <container> [<container>...]*
: Start one or more containers

  -a, --attach Attach container's STDOUT and STDERR and forward all signals to the process
  -i, --interactive
  : Attach container's STDIN

$ docker stop *[<opts>] <container> [<container>...]*
: Stop one or more containers by sending SIGTERM and then SIGKILL after a grace period

  -t, --time *[=10]*
  : Number of seconds to wait before killing the container

$ docker kill *[<opts>] <container> [<container>...]*
: Kill a runing container using SIGKILL or a specified signal

  -s, --signal *[="KILL"]*
  : Signal to send to the container

$ docker pause *<container> [<container>...]*
: Pause all processes within a container

$ docker unpause *<container> [<container>...]*
: Unpause all processes within a container

DOCKER CLI QUICK REFERENCE (continued)

Interacting with running containers

$ docker attach *[<opts>] <container>*
Attach to a running container
`--no-stdin` Do not attach STDIN (i.e. attach in read-only mode)
`--detach-keys` *="<keys>"*
Override the key sequence to detach a container. Default: `"ctrl-p ctrl-q"`

$ docker exec *[<opts>] <container> <command> [<arg> ...]*
Run a process in a running container
`-i, --interactive`
Keep STDIN open even if not attached
`-t, --tty` Allocate a pseudo-TTY
`-d, --detach` Detached (daemon) mode

$ docker top *<container> [<ps options>]*
Display the running processes within a container. The `ps options` are any options you would give to the `ps` command

$ docker cp *[<opts>] <container>:<src_path> <host_dest_path>*
$ docker cp *[<opts>] <host_src_path> <container>:<dest_path>*
Copy files/folders between a container and the local filesystem. Behaves like Linux command `cp -a`. It's possible to specify `-` as either the `host_dest_path` or `host_src_path`, in which case you can also stream a tar archive.
`-L, --follow-link`
Follow symbol link in source path

$ docker logs *[<opts>] <container>*
Fetch the logs of a container
`-f, --follow` Follow log output: it combines `docker log` and `docker attach`
`--since` *="<timestamp>"*
Show logs since the given timestamp
`-t, --timestamps`
Show timestamps
`--tail` *="<n>"* Output the specified number of lines at the end of logs

$ docker wait *<container> [<container>...]*
Block until a container stops, then print its exit code

Saving and loading images and containers

$ docker save *[<opts>] <image> [<image>...]*
Save one or more images to a tar archive (streamed to STDOUT by default)
`-o, --output` *=""*
Write to a file instead of STDOUT

$ docker load *[<opts>]*
Load image(s) from a tar archive or STDIN. Restores both images and tags
`-i, --input` *="<tar-archive>"*
Read from a tar archive file, instead of STDIN. The tarball may be compressed with gzip, bzip, or xz.
`-q, --quiet` Suppress the load progress bar

$ docker export *[<opts>] <container>*
Export the contents of a container's filesystem as a tar archive
`-o, --output` *="<file>"*
Write to a file instead of STDOUT

$ docker import *[<opts>] <file>|<URL>|- [<repository>[:<tag>]]*
Create an empty filesystem image and import the contents of the tarball into it, then optionally tag it.
`-c, --change` *=[]*
Apply specified Dockerfile instructions while importing the image; one of these: `CMD, ENTRYPOINT, ENV, EXPOSE, ONBUILD, USER, VOLUME, WORKDIR`
`-m, --message` *="<msg>"*
Set commit message for imported image

Communicating with Docker Registry

$ docker login *[<opts>] [<server>]*
Log in to a Docker Registry on the specified `<server>`. If server is omitted, `https://registry-1.docker.io` is used. Credentials are stored in `~/.docker/config.json`
`-u, --username` *="<username>"*
`-p, --password` *="<password>"*

$ docker logout *[<server>]*
Log out from a Docker Registry on the specified `<server>`. If server is omitted, `https://registry-1.docker.io` is used.

$ docker push *[<registry_host>[:<registry_port>]/]<name>[:<tag>]*
Push an image or a repository to a Registry

$ docker pull *[<opts>] [<registry_host>[:<registry_port>]/]<name>[:<tag>]*
Pull an image or a repository from a Registry
`-a, --all-tags`
Download all tagged images in the repository

Listing images and containers

$ docker images *[<opts>]*
List images
`-a, --all` Show all images (by default, intermediate image layers aren't shown)
`--no-trunc` Don't truncate output
`-f, --filter` *="<filter>"*
Filter output based on these conditions:
- `dangling=true` - unused (untagged) images
- `label=<key>` or `label=<key>=<value>`

`--format` *="<template>"*
Pretty-print containers using a Go template, e.g. {{.ID}}. Valid placeholders:
- `.ID` - Image ID
- `.Repository` - Image repository
- `.Tag` - Image tag
- `.Digest` - Image digest
- `.CreatedSince` - Time since the image was created
- `.CreatedAt` - Time when the image was created
- `.Size` - Image disk size

$ docker ps *[<opts>]*
List containers
`-a, --all` Show all containers (including non-running ones)
`--no-trunc` Don't truncate output
`-q, --quiet` Only display numeric IDs
`-f, --filter` *="<filter>"*
Filter output based on these conditions:
- `exited=<int>` an exit code of `<int>`
- `label=<key>` or `label=<key>=<value>`
- `status=(created|restarting|running|paused|exited|dead)`
- `name=<string>` a container's name
- `id=<ID>` a container's ID
- `before=(<container-name>|<container-id>)`
- `since=(<container-name>|<container-id>)`
- `ancestor=(<image-name>[:tag]|<image-id>| image@digest)` - containers created from an image or a descendant
- `volume=(<volume-name>|<mount-point-destination>)`

`--format` *="<template>"*
Pretty-print containers using a Go template, e.g. {{.ID}}. Valid placeholders:
- `.ID` - Container ID
- `.Image` - Image ID
- `.Command` - Quoted command
- `.CreatedAt` - Time when the container was created.
- `.RunningFor` - Time since the container was started.
- `.Ports` - Exposed ports.
- `.Status` - Container status.
- `.Size` - Container disk size.
- `.Names` - Container names.
- `.Labels` - All labels assigned to the container.
- `.Label` - Value of a specific label for this container. For example {{.Label "com.docker.swarm.cpu"}}
- `.Mounts` - Names of the volumes mounted.

Inspecting images and containers

$ docker inspect *[<opts>] <container>|<image> [<container>|<image>...]*
Return low-level information on a container or image
`-f, --format` *="<format>"*
Format the output using the given Go template. You can see the available placeholders by looking at the total output without `--format`
`-s, --size` Display total file sizes if the type is container
`-t, --type` *="<container>|<image>"*
Return JSON for specified type only

Removing images and containers

$ docker rm *[<opts>] <container> [<container>...]*
Remove one or more containers from the host
`-f, --force` Force the removal of a running container (uses SIGKILL)
`-f, --link` Remove the specified link and not the underlying container
`-v, --volume` Remove the volumes associated with the container

$ docker rmi *[<opts>] <image> [<image>...]*
Remove one or more images from the host
`-f, --force` Force the removal of images of a running container
`--no-prune` Do not delete untagged parents

by Dmitry Frank http://dmitryfrank.com

Your contributions to this card are welcome: https://github.com/dimonomid/docker-quick-ref