**Docker**

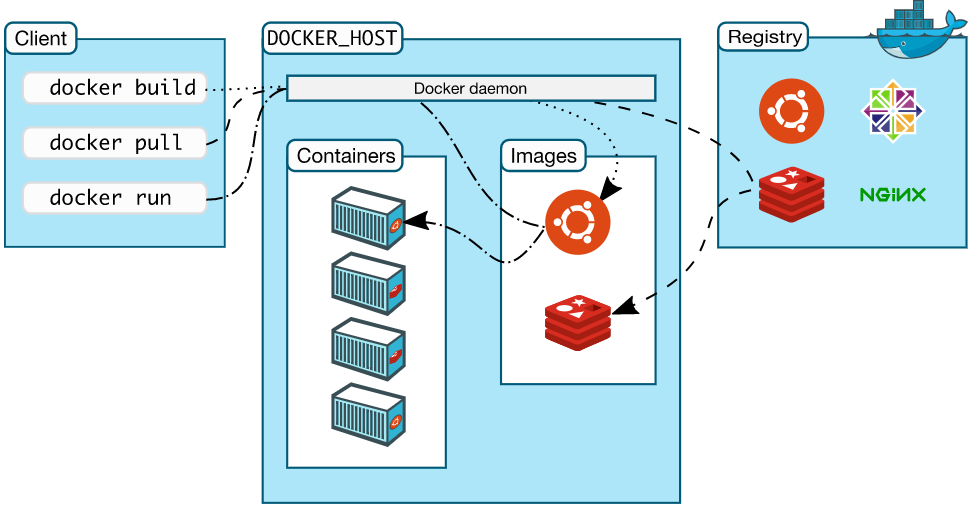
Last Update: July 2021.

Docker is a platform which simplifies building, deployment and shipping of any given program. It needs a single ‘dockerfile’ which instructs how it should build a ‘dockerimage’. The dockerfile has only links to files and how it should run them but a dockerimage has all the files + links to other docker images it needs to run the program in it. We simply share this dockerimage and it will run on any platform that supports docker.

Docker images can be stored on docker registry website which reduces an image’s size that uses the image as a dependency.

When a docker image is ran, it creates a container and runs in it. A container is an isolated process on the host, i.e., the programs in it cannot see any external services or anything, it is a sandbox that only has the image it was given running in it.

This is the architecture: There is docker client, daemon which can be on external host and a registry.



• Dockerfile: The building block of an app on docker. Each line in this file is run from top to bottom sequentially, this file is usually placed in the root of a project and has no extension, just this name. A dockerfile must have either CMD or EP or both.

Syntax:

Dockerfile:

<INSTRUCTION> <arguments>

Instruction can be lower case but uppercase is preferred for distinguishability.

First instruction must be FROM instruction to specify base dependency / Parent image. ARG can be before FROM only if the FROM’s argument needs an argument which has to be provided by ARG.

‘#’ is for comment line.

Parser directive: A key value pair that affect how subsequent lines in dockerfile are handled. They must be before any instruction and before any comment.

# <directive>=<value>

Only 2 directives are usable rn, syntax and escape.

Syntax has use in buildkit, escape is used to define an escape character.   
By default the escape character is ‘\’, same as directory path in windows. If an escape character is at the end of a line (except for RUN instruction), it causes the next line to be appended to the current line while building. So for windows if we have COPY x.txt c:\\ where c:\\ means c:\ (as single \ is removed by windows) docker assumes it as c:\<append next line>. To avoid this we can use # escape=’ directive.

INSTRUCTIONS:

FROM [--platform=<platform>] <image name>[:<tag> or @<digest>] [AS <name>] : The first instruction, we only need to specify an image name but others can be used.

ARG: <varname>[=<value>]: Argument, we can use this variable in FROM as it exists outside the build scope, it must precede it and loses its value after the first FROM (if declared before the first FROM). We can simply reinitialize it using ARG <varname> and now it can be used anywhere else.

ARG VERSION=latest

FROM someimage:$latest

ARG values are optional inside dockerfile but they must have either default values or be passed values through docker build.

ENV <varname>=<value>: Docker Environment Variable. Assigns the given value to the variable, this variable can be used by any instruction to assign this value+their value.

ENV FO=/bar

ADD . $FO #or ${FO}

Will replace FO with /bar. If we escape the $ then it won’t.

Bash modifiers are supported as well, so

${FO: -woo} means if FO variable exists then use that value else put woo instead.

${FO: +wo} means if FO exists then wo is put else empty string.

RUN <command> #or [“executable”,”param1”,”param2”…]: Runs the instruction on a new layer(image) and allows it to be used in the further instructions. RUN <command> (called shell form) runs a command with cmd /S /C on windows and /bin/sh -c in linux, shell form runs the command on the native shell. The [] form is called exec form.

Shel form does variable substitution but exec form normally doesn’t. To use variables in exec form we call the shell directly, like RUN [“cmd”,”/S”,”/C”,”$VARIABLE”] will work.

CMD <command> <params> (shell form) #or CMD [“param1”,”param2”] (passes these params to ENTRYPOINT) #or CMD [“command”,”param1”,”param2”] (exec form): Each dockerfile can run only 1 CMD, if it has more then last is ran. These are used to pass defaults to executing container. Variable substitution only in shell form or calling shell in exec form.   
The diff between RUN and CMD is that RUN executes a command where it is specified only at build time whereas CMD executes the command each time the image is executed, if we pass args to docker run then CMD args are overridden.

LABEL <key>=<value> <key2>=<value2>: Used to add metadata to an image, single dockerfile can have multiple LABELs and multiple labels in single line.   
To see an image’s labels, docker image inspect –format=’’ <imagename>

EXPOSE <port> [/<port type>]: Exposes a port which could be used by internal app, it also has to be mapped using docker run -p to be exposed.

ENV <key>=<Value>: Just like ARG but the variables created here persist even after build and can be used by an image that is ran.

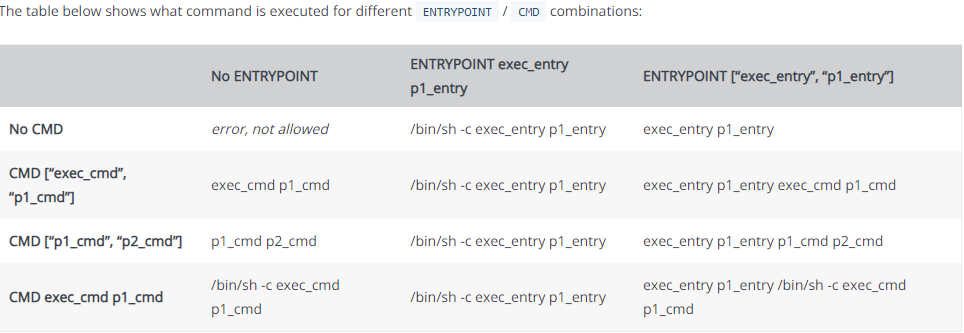
To see ENV variables of a dockerfile, use docker inspect <imagename>.

ADD [--chown=<user>:<group>] <src> <dest> #or ADD [--chown…] [“<src>”,…”<dest>”]: Copies files from src to dest to be used by the image. Chown is used in linux containers. Dest can be /<dest> which means absolute location or <dest> which means relative to context location for the files. The src must be inside the context, src can be a url and in that case the filename will be taken from url and if <dest> ends with / then placed inside <dest> named file at <dest>. Src can be file or folder or url. ADD also unpacks recognized (identity, gzip, bzip2 or xz) archives and sprays them in the dest.

COPY: Same as ADD but doesn’t accept URL src and doesn’t unpack archives.

ENTRYPOINT [“executable”,”param1”,”param2”] (exec form) #or ENTRYPOINT <command> <args> (shell form) : Entrypoint gives the executable that is ran when the image is ran. Any args passed to docker run are passed to entrypoint and overridden in cmd. Only 1 EP is allowed. To allow clean exit, shell form with ‘exec’ command is used. Like, ENTRYPOINT exec <stuff>.

CMD is used to pass default arguments to ENTRYPOINT.



VOLUME <”location”>: Sets the mount point as the given directory so all operations take place inside this directory (src is still going to call whatever external resource it calls).

USER <user>[:<group>] #or USER <UID>[:<GID>]: Set the user of the given group as the user for all subsequent RUN/CMD/EP instructions.

WORKDIR /path/: Sets the current directory as the given path, which will then be used for path by all other instructions such as src. If multiple workdirs are set and they follow relative path then they are relative to their previous WORKDIRs. They can use variables but only ones created by ENV and not through shell.

ONBUILD <INSTRUCTION>: This is a trigger instruction, i.e., it triggers an instruction on a specific trigger. The trigger here is triggered when the image is used as a base reference to another image. There can be multiple onbuilds in a dockerfile and are all sequentially appended in the image metadata. When a new image references this image they are executed right after FROM of the new image in the order of specificity. They are discarded from the image that uses them.   
Instruction can be anything except FROM and MAINTAINER.

STOPSIGNAL SIGNAL<NAME> #or STOPSIGNAL <uint signal>: Defines the stop signal for a container, by default it is SIGKILL or 9.

HEALTHCHECK [OPTION] CMD <args> #or HEALTHCHECK NONE: Checks if the command is running or not, if NONE is defined then it disables healthcheck of referenced images. There can only be 1 single HEALTHCHECK in an image and it runs after interval seconds.

[OPTIONS] are:

* --interval=DURATION (default: 30s)
* --timeout=DURATION (default: 30s)
* --start-period=DURATION (default: 0s)
* --retries=N (default: 3)

SHELL [“excecutable”,”params”]: Sets this executable as the default shell for all the subsequent RUN/CMD/EPs, overrides all previously declared SHELLs.

• Docker CLI: Run docker commands through the CLI.

docker [OPTIONS] COMMAND <args> <values>

OPTIONS are:

--config <str>: config location

-c, --context <str>: name of context to use to connect to daemon.

-D, --debug <bool>: enable debug mode.

--help

-H, --host []: Daemon sockets to connect to.

-l, --log-level <str>: Value can be debug, info, warn, error, fatal. Default info.

--tls: use tls.

--tlscacert <str>: Trust certs signed by this given CA only.

--tlscert <str>: Path to tls certs.

--tlskey <str>: Path to tls keyfile.

--tlsverify <bool>: Use --tls and verify the remote.

-v, --version

COMMANDS ARE:

docker attach: Attach local standard input, output, and error streams to a running container

docker build: Build an image from a Dockerfile

docker builder: Manage builds

docker checkpoint: Manage checkpoints

docker commit: Create a new image from a containerâs changes

docker config: Manage Docker configs

docker container: Manage containers

docker context: Manage contexts

docker cp: Copy files/folders between a container and the local filesystem

docker create: Create a new container

docker diff: Inspect changes to files or directories on a containerâs filesystem

docker events: Get real time events from the server

docker exec: Run a command in a running container

docker export: Export a containerâs filesystem as a tar archive

docker history: Show the history of an image

docker image: Manage images

docker images: List images

docker import: Import the contents from a tarball to create a filesystem image

docker info: Display system-wide information

docker inspect: Return low-level information on Docker objects

docker kill: Kill one or more running containers

docker load: Load an image from a tar archive or STDIN

docker login: Log in to a Docker registry

docker logout: Log out from a Docker registry

docker logs: Fetch the logs of a container

docker manifest: Manage Docker image manifests and manifest lists

docker network: Manage networks

docker node: Manage Swarm nodes

docker pause: Pause all processes within one or more containers

docker plugin: Manage plugins

docker port: List port mappings or a specific mapping for the container

docker ps: List containers

docker pull: Pull an image or a repository from a registry

docker push: Push an image or a repository to a registry

docker rename: Rename a container

docker restart: Restart one or more containers

docker rm: Remove one or more containers

docker rmi: Remove one or more images

docker run: Run a command in a new container

docker save: Save one or more images to a tar archive (streamed to STDOUT by default)

docker search: Search the Docker Hub for images

docker secret: Manage Docker secrets

docker service: Manage services

docker stack: Manage Docker stacks

docker start: Start one or more stopped containers

docker stats: Display a live stream of container(s) resource usage statistics

docker stop: Stop one or more running containers

docker swarm: Manage Swarm

docker system: Manage Docker

docker tag: Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

docker top: Display the running processes of a container

docker trust: Manage trust on Docker images

docker unpause: Unpause all processes within one or more containers

docker update: Update configuration of one or more containers

docker version: Show the Docker version information

docker volume: Manage volumes

docker wait: Block until one or more containers stop, then print their exit codes

• Build: Docker builds the image on the server / daemon and not on the client / CLI.

docker build <args> <values> <context> :Default docker builder

docker buildx <args> <values> <context> :BuildKit docker builder

The context is the location which has the dockerfile. All subdirectories in the given context / location are searched for dockerfiles and all dockerfiles are built. In building, docker follows a layered approach, each line in a dockerfile creates its own layer, a layer is an image. For layers that use external images, only links are copied and for layers that use internal files the files and their related data such as env vars are copied to the image. Usually we pass ‘.’ as context, it means current directory.

The .dockerignore is searched only in the root directory and acts just like .gitignore.

args:

-f: Define a location to a docker file (context is still needed).

-t: Give a tag to an image. Like x/y:v1 , here y is image name and v1 is a tag. Multiple -t tags can be used to give multiple tags to the same image name.

--nocache: Normally all RUN instructions create a cache (save their layer) which can be used when the same image is called next time, we can disable using it in current build with this.

- < <somefile> : Puts this file in ADD inside the build. Can only be URL since using this means we cannot pass a context to build.

--build-arg <varname>=<value>: Passes the value to the declared ARG inside dockerfile. Some ARGS are predeclared, they are:

* HTTP\_PROXY
* http\_proxy
* HTTPS\_PROXY
* https\_proxy
* FTP\_PROXY
* ftp\_proxy
* NO\_PROXY
* no\_proxy

BuildKit has more predeclared ARGS. Check ref.

• Execution: Executes a given image.

docker run [OPTIONS] <image-name>[:<image tag or @digest>] [COMMAND] <arg> <value>

Options are the args that are specified for running the image itself but COMMANDs are the args passed to the image.

When docker run is ran the console is attached to the process, all the stdin, stdout and stderr events can travel between console and process.

Option types have some properties,

docker run -d –p …

can be

docker run -dp …

i.e., multiple args can collapse. Just their values should be given in the same order.

Bool options such as -d=<bool> can be left empty, which means true.

For an arg -a=[] we can specify -a <value> multiple times in same line.

args:

-p <internal/exposed port>:[<host port>] : publish and map a port. Only Exposed ports can be mapped.

--env <key>=<value>: Used to change an ENV var for a specific image.

--entrypoint: Overrides the entrypoint of the image.

-d=<bool>: Detached mode, meaning the image will start as a background process. This means the process isn’t attached to the console that started it and hence further commands won’t affect it.

-a=[]: Value can be stdin,stdout and stderr. The values specified tell which type of std operation should the console allow to the attached process. If not used, docker attaches to stdout and stderr.

-i=<bool>: Keep the stdin open even if not attached to process.

--sig-proxy=<bool>: Proxies all received signals to the process.

-t=<bool>: Allocate a pseudo-tty. It makes the container act like a terminal driver

--name: Assign a name to container.

--cidfile=<path to file>: Write container id to file.

--pid=””: Value can be ‘host’ or ‘container:<name|id>’. Lets the container exist in the same namespace as the host or same as another container given its name or cid.

--ipc=””: IPC communication setting. Value can be “”, none, private, shareable, container:<name|id>, host. “” defaults to private or shareable. Allows containers to contact each other through memory if shareable or container is used. None or private means individual namespace. Host just spills the process namespace to the host.

--dns=[]: Set custom dns, default host dns.

--network=””: Connect container to given network. Value can be none, bridge, container:name|id, host, <network name|id>.

--network-alias=[]: Scoped network alias.

--add-host=: Add host.

--mac-address: Custom mac for container.

--ip=””: Custom ipv4.

--ip6=””

--link-local-ip=[]

--restart=””: Defines what the container does on exit. Values can be no for disable restart, on-failure[:max-retries], always, unless-stopped.

--rm=<bool>: This allows the container to remove filesystem after it exits. If we use -v with it then all directories without the name given to v are removed.

--security-opt=””: Sets a label for the container. Some values affect security of the container. Check doc for values.

--init: Specify the initial process started of the given image.

--cgroup-parent: Define a cgroup to run the container in. cgroups are like namespaces with their own limits for CPU and memory usages.

-m or –memory=””: Value <int>[<units>]. int value with b,k,m, or g literal, default is b. Max memory.

--memory-swap=””: Max Memory + swap.

--memory-reservation=””

--kernel-memory=””: Kernel max memory.

-c or –cpu-shares=<int>: Max cpu shares. Shares define priority on the scheduler, 0 being lowest.

--cpus=<double>: Max No. of cpus, 0.000 means no limit.

--cpu-period=<int>: Set CPU CFS period.

--cpuset-cpus=””: CPUs to allow execution in, like “0-2”.

--cpuset-mems=””: MEMs in which to allow execution, only for NUMA.

--cpu-quota=<int>: Set CPU CFS quota.

--cpu-rt-period=<int>: Set cpu real-time period in microseconds, requires cgroup to be set.

--cpu-rt-runtime=<int>: cpu rt runtime.

--blkio-weight=<int>: Block IO weight, value between 10 and 1000.

--blkio-weight-device=: Block IO weight, value “DEVICE\_NAME:WEIGHT”.

--device-read-bps=””: Max read rate from device. Value “<device-path>:<int>[<unit>]”. Units are kb,mb or gb.

--device-write-bps=””:

--device-read-iops=””: Read IO, Value “<dev path>:<int>”.

--device-write-iops=””

--oom-kill-disable=<bool>: Disable OOM killer. (Out of Memory Killer)

--oom-score-adj=<int>: Tune OOM prefs. Value -1000 to 1000.

--memory-swappiness=””. Tune memory swappiness, Value 0 to 100.

--shm-size=””: Size of /dev/shm/. Value <int>[<units>]. Default 64m.

--cap-add: Add linux capabilities. Refer doc for available capabilities.

--cap-drop: Drop linux capabilities.

--privileged: Give extended privileges to the container.

--device=[]: Allows to run devices inside the container without –privileged. These devices (/dev/device) can have a label of r, w or m. Default is all 3.

--log-driver=””: Container may have different logging driver. Values are none (don’t gen logs), json-file (write logs to json file), syslog, journald,gelf, fluentd, awslogs, splunk.

Almost all args can be overridden of a dockerfile, the only args that can’t be overridden are FROM, MAINTAINER, RUN and ADD.