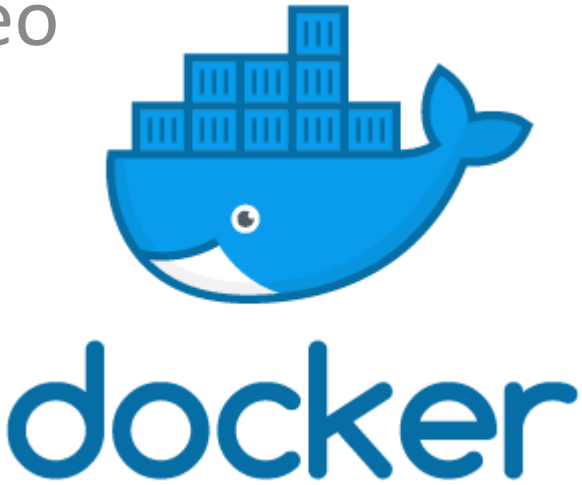
Docker

Riccardo Cattaneo



Da Applicazione a Container

I Container eseguono delle applicazioni o dei servizi. Quindi nasce spontanea l'esigenza di convertire un'applicazione e renderla un container.

Questo processo prende il nome di **Containerizing** oppure **Dockerizing**. Quindi dockerizing un'applicazione significa effettuare uno snapshot (istantanea) del filesystem e delle dipendenze dell'applicazione stessa. Questo snapshot non sarà altro che la nostra immagine docker.

DockerFile

Creare le proprie immagini grazie alla definizione di un **dockerfile** è un concetto fondamentale : ci permette di personalizzare le nostre immagini docker sulla base della nostra infrastruttura attuale. Questa è la sequenza :

dockerfile → immagine → container

Processo di dockerizing

Questo processo si può riassumere in 4 fasi :

- Creazione dell'applicazione e accesso al codice della stessa;
- Creazione del dockerfile che contiene tutte le informazioni dell'applicazione, le dipendenze e tutto ciò che sarà necessario per eseguirla;
- Creazione dell'immagine derivata dal dockerfile;
- Creazione ed esecuzione del container e quindi esecuzione della nostra applicazione.

Struttura di un dockerfile

Il dockerfile è un semplice file di testo che contiene tutte le istruzioni affinché si possa creare l'immagine voluta. Docker ci fornisce una serie di comandi da utilizzare all'interno di questo file, tra cui :

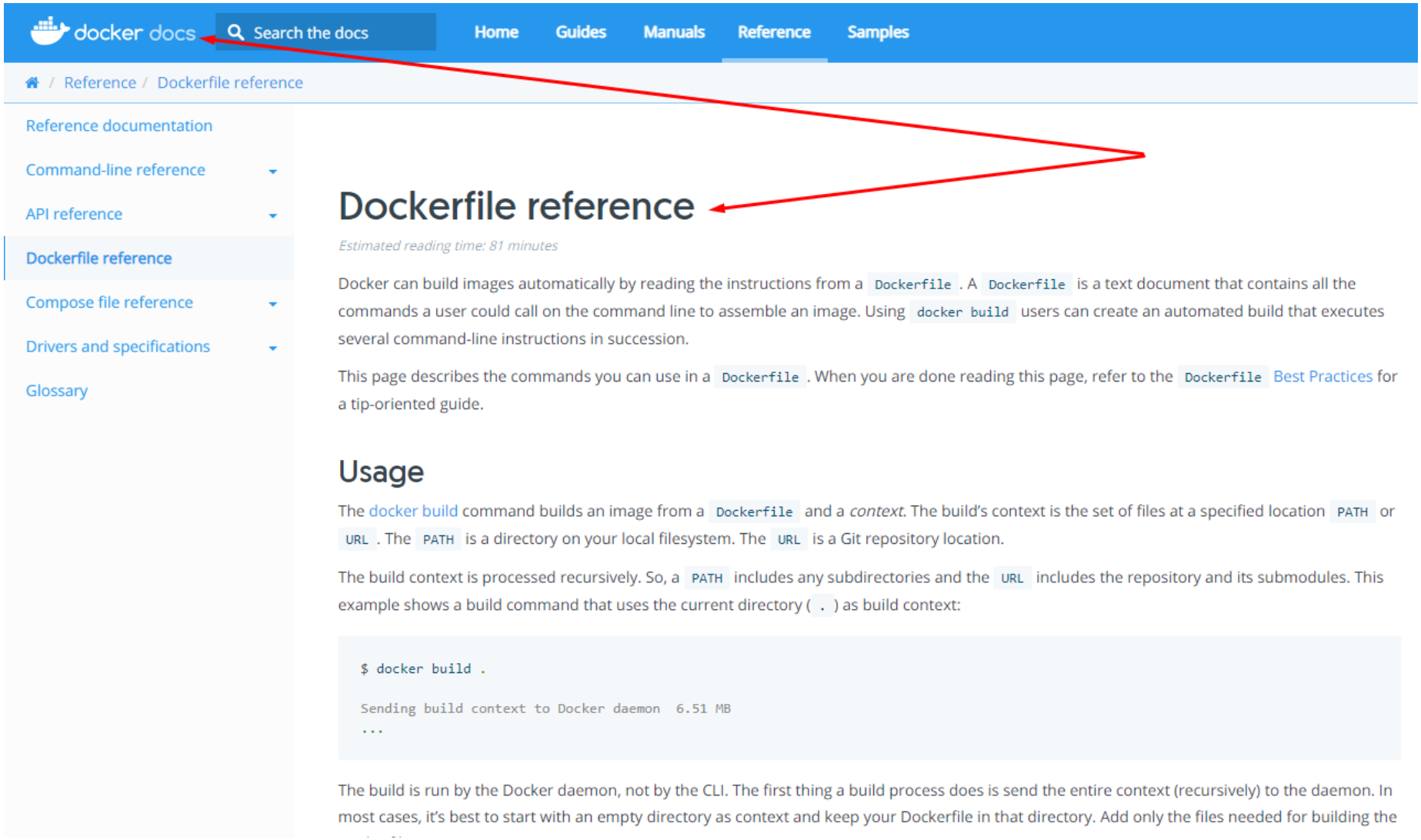
FROM
CMD
RUN
EXPOSE
COPY
ENTRYPOINT

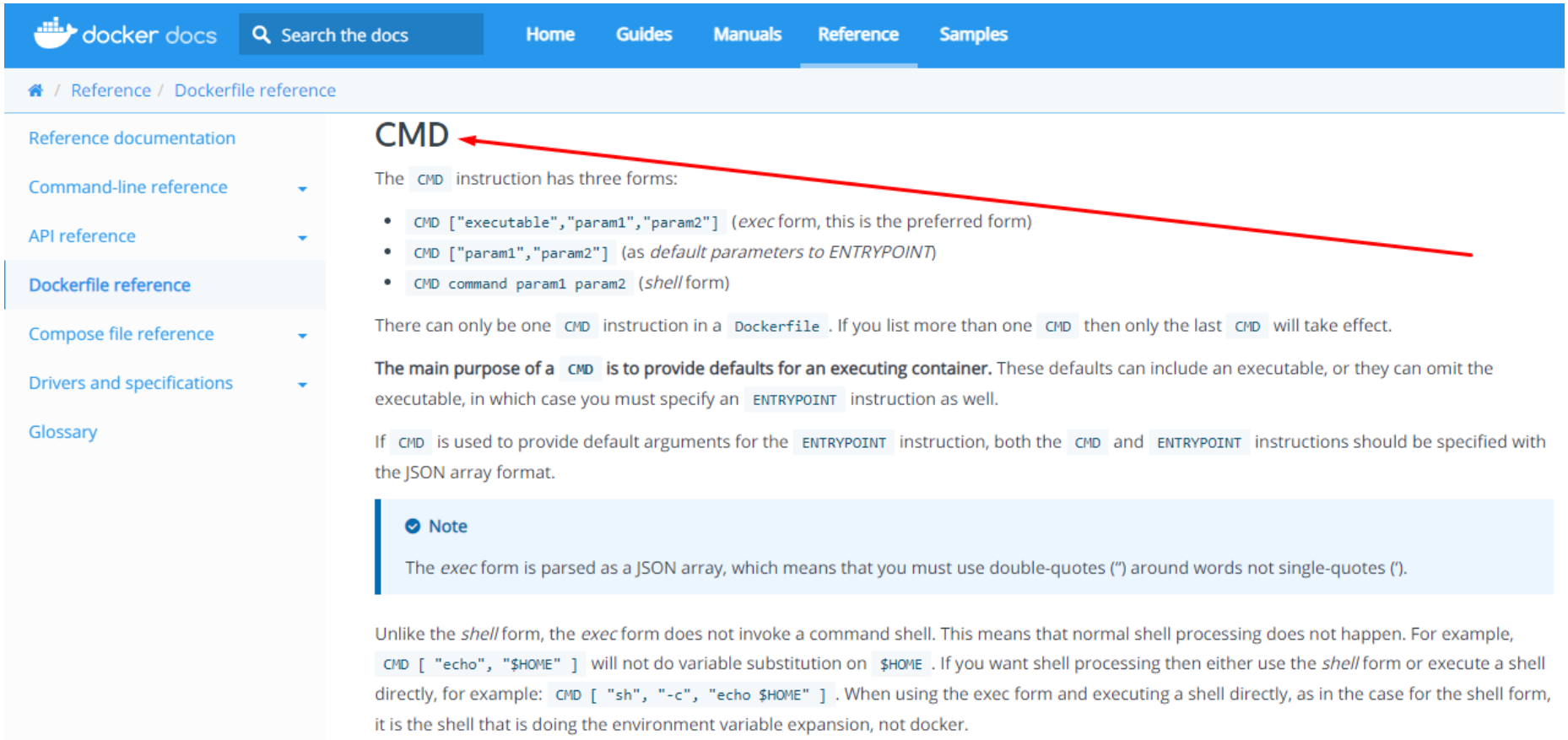
Quando si definisce una nuova immagine, è possibile partire completamente da zero oppure partire da un'immagine già presente (il caso più comune), ciò è possibile farlo tramite l'istruzione FROM all'interno del dockerfile.

Prendiamo adesso come esempio l'immagine alpine (distribuzione linux) e procediamo ad aggiungere dei tool non presenti nell'immagine base :

- FROM alpine:latest
- RUN apk update
- RUN apk add vim

Il comando RUN esegue i comandi Linux esplicati. APK è il package manager della distribuzione alpine.


docker docs
Search the docs
Home
Guides
Manuals
Reference
Samples
Reference / Dockerfile reference
Reference documentation
Command-line reference
API reference
Dockerfile reference
Compose file reference
Drivers and specifications
Glossary
Dockerfile reference
Estimated reading time: 81 minutes
Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is 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 for a tip-oriented guide.
Usage
The docker 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.
The build 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 build 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


docker docs
Search the docs
Home
Guides
Manuals
Reference
Samples
Reference / Dockerfile reference
Reference documentation
Command-line reference
API reference
Dockerfile reference
Compose file reference
Drivers and specifications
Glossary
CMD
The CMD instruction has three forms:
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)
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.
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.
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.
Note
The exec form is parsed as a JSON array, which means that you must use double-quotes (") around words not single-quotes (').
Unlike the shell form, the exec form does not invoke a command shell. This means that normal shell processing does not happen. For example, CMD ["echo", "$HOME"] will not do variable substitution on $HOME. If you want shell processing then either use the shell form or execute a shell directly, for example: CMD ["sh", "-c", "echo $HOME"]. When using the exec form and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

Creazione di un dockerfile

Per prima cosa creiamo una cartella dove andremo a salvare il nostro file, ad esempio possiamo chiamarla dockerfile. A questo punto possiamo creare il dockerfile (con windows possiamo farlo con notepad):

touch Dockerfile

vi Dockerfile (Vi è un editor di testo di linux)

Scrivere all'interno del file :

FROM alpine

CMD [''echo'','ciao mondo'']

Esco da Vi con il comdano :wq e poi invio. Una volta uscito dal vi controlliamo che abbiamo scritto tutto correttamente tramite il comando cat Dockerfile.

Una volta creato il dockerfile possiamo procedere con la build (costruzione) dell'immagine dal docker file :

docker build .

Con questo comando andrà automaticamente a cercare nella cartella corrente (grazie al punto) un dockerfile e se lo trova lo esegue. Posso controllare che è stata creata l'immagine con il comando

docker image ls

Dopodiché posso far partire un container a partire dall'immagine appena creata :

docker run - -name containerAlpine ID_IMMAGINE

Opzione COPY

Il comando COPY ci permette di copiare qualcosa dal nostro host verso il container. La sintassi è la seguente :

COPY SOURCE DEST

Proviamo a costruire un dockerfile con questo comando :

Per prima cosa creiamo una nuova cartella chiamata prova ed inseriamo un file script1.sh ed inseriamo all'interno il comando echo ''ciao mondo''. Creiamo ora il dockerfile :

- FROM alpine
- COPY script1.sh /script1.sh
- CMD ["sh","script1.sh"]

Proviamo ora a creare l'immagine con docker build –t nomeimg . (**opzione –t assegna un nome all'immagine**), ed eseguiamo poi il container dall'immagine appena creata con **docker run -- name c1 ID_IMMAGINE**

Comando RUN

RUN esegue un comando. Andiamo a creare un dockerfile partendo da Ubuntu ed installiamo sulla distribuzione il comando ping non presente di default. Creiamo una cartella prova1 ed all'interno andiamo a creare il nostro dockerfile così composto :

- FROM ubuntu
- RUN apt-get update
- RUN apt-get install -y iputils-ping

L'opzione –y sta a significare che **non** ci deve essere interazione con l'utente (come sappiamo, su linux, quando si esegue un comando, chiede conferme o comunque interazioni con l'utente).

Proviamo ora a creare l'immagine con docker build –t nomeimg .

Eseguiamo poi il container dall'immagine appena creata con **docker run -- name –it c2 ID_IMMAGINE /bin/bash**

Una volta dentro il nostro container proviamo ad eseguire un ping verso il sito di google e vediamo che esegue il comando…

ping www.google.it

RUN vs CMD

RUN è una fase di creazione dell'immagine, lo stato del contenitore dopo che un RUN verrà assegnato all'immagine del contenitore. Un file Docker può avere molti RUN.

CMD è il comando che il contenitore esegue per impostazione predefinita all'avvio dell'immagine creata. Un file Docker utilizzerà solo il finale CMD definito.

ENTRYPOINT

ENTRYPOINT permette di configurare un container nella sua esecuzione. Riprendendo l'esempio precedente del ping, abbiamo creato un'immagine che, quando viene creata in un container, dobbiamo poi eseguirlo in modalità interattiva per eseguire il ping sul sito di google. Mentre tramite ENTRYPOINT possiamo «dire» di eseguire un comando, ma che eventuali parametri verranno «passati» al momento della creazione del container.

Perché un Dockerfile sia valido, deve contenere almeno un comando tra CMD e ENTRYPOINT.

Quindi mentre prima il nostro dockerfile era composto dalle seguenti istruzioni :

- FROM ubuntu
- RUN apt-get update
- RUN apt-get install -y iputils-ping

Ora lo modifichiamo in questo modo :

- FROM ubuntu
- RUN apt-get update
- RUN apt-get install -y iputils-ping
- ENTRYPOINT [‘’ping’’,’’-c’’,’’5’’]

In questo modo stiamo dicendo al container che, quando verrà eseguito, si aspetterà un **parametro** che a questo punto diventa **obbligatorio** (l'opzione del comando ping –c sta ad indicare quante volte deve essere eseguito il comando ping, di seguito il valore) :

docker build .

docker run ID_CONTAINER **www.google.com**

Da Container a Immagine

E’ possibile creare un’immagine a partire da un container già configurato e convertito successivamente in un’altra immagine. Questa pratica non è molto usata in quanto è sempre preferibile usare il dockerfile. Può servire quando ad esempio in un container eseguo nuove installazioni e voglio creare un’immagine aggiornata:

docker commit ID_CONTAINER