Docker and Kubernetes: The Complete Guide

Jacopo De Angelis

21 settembre 2021

Indice

Capitolo 1

Perchè usare docker e cos'è?

Docker si prefigge di permettere di passare un software senza doversi preoccupare di dipendenze o altro.

Immagine: un singolo file con tutte le dipendenze e le configurazioni richieste per lanciare un programma.

Container: istanza di un'immagine.

Il docker client ci permette di interfacciarci con il docker server, un programma col quale non ci interfacciamo direttamente ma è un processo che lavora dietro le quinte.

Capitolo 2

Manipolazione dei container

2.1 docker run

docker run <***nome immagine***>: lancia il programma all'interno del programma.

docker run <***nome immagine***> <***comando***>: all'interno di un'immagine viene lanciato il comando specificato e non quello di default. Nel caso il comando non fosse presente all'interno delle cartelle verrà mostrato un messaggio d'errore.

Quando viene lanciato run viene creato un container.

2.2 docker ps

Tutti i container possono essere mostrati tramite ***docker ps,*** se vogliamo vedere anche quelli che sono stati terminati in passato allora lanciamo ***docker ps –all***. Se vogliamo lanciare un comando relativo ad uno specifico container allora dovremo utilizzare il suo ID o il suo nome.

```
jacopo@DESKTOP-1QEP600:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

jacopo@DESKTOP-1QEP600:~$ docker run hello-world echo hi
docker: Error response from daemon: OCI runtime create failed: container_linux.go:380: starting container process caused: exec: "echo":
 executable file not found in $PATH: unknown.
ERRO[0000] error waiting for container: context canceled
jacopo@DESKTOP-1QEP600:~$ docker run busybox echo hi
hi
```

Figura 2.1: Docker run

```
PS C:\Users\jacop> docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS          PORTS                                   NAMES
883bcb1f050d   docker101tutorial  "/docker-entrypoint.…"   27 minutes ago   Up 27 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp   docker-tutorial
PS C:\Users\jacop> docker ps --all
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS                      PORTS                                   NAMES
7f34a0e1bb2b   busybox            "sh"                     26 minutes ago   Exited (0) 26 minutes ago                                   strange_montalcini
12ad74861e01   hello-world        "/hello"                 27 minutes ago   Exited (0) 27 minutes ago                                   epic_robinson
883bcb1f050d   docker101tutorial  "/docker-entrypoint.…"   27 minutes ago   Up 27 minutes               0.0.0.0:80->80/tcp, :::80->80/tcp   docker-tutorial
f6f6992f96b0   alpine/git         "git clone https://g…"   29 minutes ago   Exited (0) 29 minutes ago                                   repo
```

Figura 2.2: Docker ps

2.3 docker run = docker create + docker start

docker run in realtà deriva dall'unione di due comandi: ***docker create*** e ***docker start***; col primo comando semplicemente prepariamo il container, preparandone le risorse.

docker create restituisce un ID che può essere usato tramite ***docker start***. ***docker start*** <***id***> semplicemente restituirà nuovamente l'id ma avviserà che il container sarà attivato, ***docker start -a*** <***id***> eseguirà anche il comando interno al container.

Quando un container è stato spento non vuole dire che non possa essere riavviato, semplicemente dovremo usare ***docker start -a*** <***id***> dove l'id è quello del container spento, così verrà rieseguito il comando eseguito precedentemente.

```
jacopo@DESKTOP-1QEP600:~$ docker create hello-world
c7b3824ac825358ff8640371c72b497d60e6d80159c303ca5ff4c925fb84296b
jacopo@DESKTOP-1QEP600:~$ docker start -a c7b3824ac825358ff8640371c72b497d60e6d80159c303ca5ff4c925fb84296b

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figura 2.3: docker create e docker start

2.4 Eliminazione e terminazione di un container

docker system prune rimuove tutti i container spenti, tutti i network inutilizzati, tutte le immagini non usate e la cache.

docker logs <container id> permette di recuperare tutti i log da un container. *docker logs* non fa ripartire un container, ne recupera solo i log.

```
jacopo@DESKTOP-1QEP600:~$ docker create busybox echo hi
e78bc77168de08b266a65d0a98cecb082938349d1834199c9d6bbbedff55224e
jacopo@DESKTOP-1QEP600:~$ docker start e78bc77168de08b266a65d0a98cecb082938349d1834199c9d6bbbedff55224e
e78bc77168de08b266a65d0a98cecb082938349d1834199c9d6bbbedff55224e
jacopo@DESKTOP-1QEP600:~$ docker logs e78bc77168de08b266a65d0a98cecb082938349d1834199c9d6bbbedff55224e
hi
```

Figura 2.4: docker logs

docker stop <container id> stoppa un container mandando un comando di terminazione del segnale per il container (SIGTERM), questo vuole dire che ci verrà restituito un segnale che potremo usare per fare partire altri comandi. Nel caso il container non termini entro 10 secondi verrà mandato un SIGKILL.

docker kill <***container id***> stoppa un container mandando un comando di terminazione definitiva per il container (SIGKILL), questo vuole dire che appena viene lanciato non c'è niente che possa essere fatto dopo in risposta.

2.5 Agire con un container

Se avviamo un container in docker e poi proviamo a lanciare dei comandi per interagire con esso dall'esterno come se nulla fosse otterremmo un errore, questo deriva dal fatto che le risorse del container sono accessibili solo da dentro il container. Infatti come si vede dall'immagine **??**, all'interno di docker è stato lanciato un server di redis ma se provassimo ad avviare una connessione a redis dall'esterno direttamente otterremmo un errore nel quale ci viene detto che redis non è attivo.

```
jacopo@DESKTOP-1QEP600:~$ docker run redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
a330b6cecb98: Pull complete
14bfbab96d75: Pull complete
8b3e2d14a955: Pull complete
5da5e1b21a2f: Pull complete
6af3a5ca4596: Pull complete
4f9efe5b47a5: Pull complete
Digest: sha256:e595e79c05c7690f50ef0136acc9d932d65d8b2ce7915d26a68ca3fb41a7db61
Status: Downloaded newer image for redis:latest
1:C 19 Sep 2021 12:43:43.815 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 19 Sep 2021 12:43:43.815 # Redis version=6.2.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 19 Sep 2021 12:43:43.815 # Warning: no config file specified, using the default config. In order to specify a config file
-server /path/to/redis.conf
1:M 19 Sep 2021 12:43:43.815 * monotonic clock: POSIX clock_gettime
1:M 19 Sep 2021 12:43:43.816 * Running mode=standalone, port=6379.
1:M 19 Sep 2021 12:43:43.816 # Server initialized
1:M 19 Sep 2021 12:43:43.816 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To
issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' f
 take effect.
1:M 19 Sep 2021 12:43:43.816 * Ready to accept connections
```

Figura 2.5: docker non accessibile dall'esterno

Per questo dobbiamo lanciare docker exec -it <container id> <comando>, in questo modo diciamo che vogliamo eseguire un comando (exec) attraverso un input (-it) all'interno del container specificato.

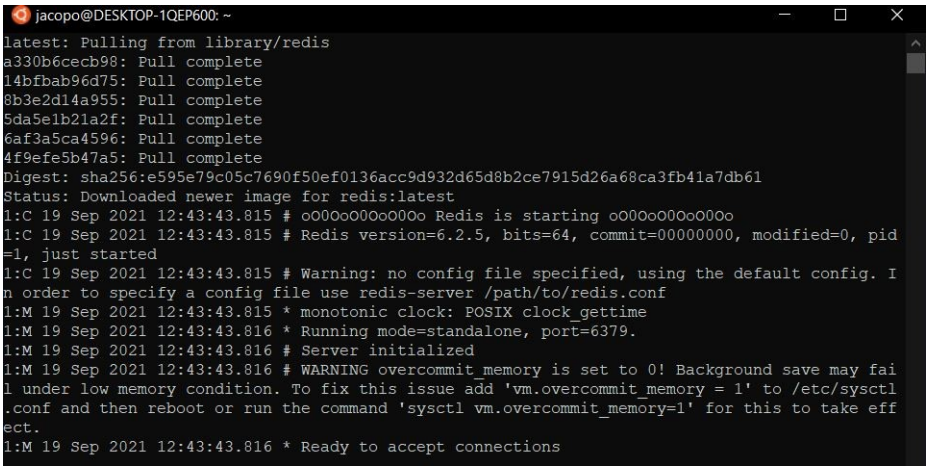

Figura 2.6: docker exec

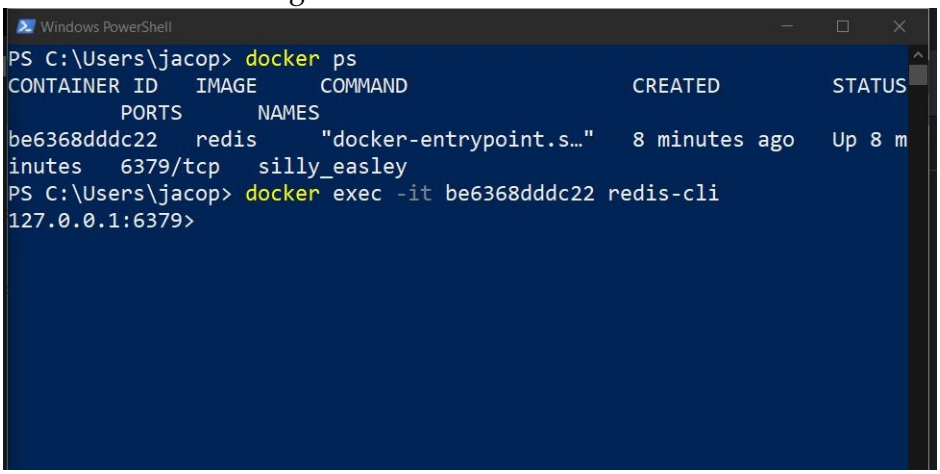

Figura 2.7: docker exec

Senza -it non potremmo inserire alcun input. Ogni processo in un container ha tre stream (stdin, stdout, stderr). Attenzione, it è in realtà -i -t, nel primo stiamo dicendo ricevi l'input in stdin, -t segnala che si vuole un output leggibile.

2.6 Aprire un prompt all'interno di un container

docker exec -it <container id> sh, quel sh ci permette di aprire un terminale all'interno del container. Per uscire dal container *ctrl + D*. ***docker run -it <container id> sh*** ottiene lo stesso risultato.

2.7 Isolamento dei container

Come è già stato detto un container ha il suo set di risorse e di memoria, questo vuole dire che i container non condividono nessuna informazione tra di loro.

Capitolo 3

Creazione di un'immagine

Dovremo creare un dockerfile. Immaginiamo di volere creare un'immagine che lancerà il redis server al suo avvio.

Prima di tutto creiamo una nuova cartella che conterrà i file, al suo interno creiamo un file chiamato **Dockerfile**, nessuna estensione. Al suo interno scriviamo un set di istruzioni:

```
# Usare un'immagine esistente come base
FROM alpine

# Scaricare e installare una dipendenza
RUN apk add --update redis

# Dire all'immagine cosa fare al suo avvio come
    ↪ container
CMD ["redis-server"]
```

A questo punto scriviamo in console da dentro la cartella ***docker build .***, così verrà creata l'ìmmagine scaricando l'immagine di ALPINE, aggiornando il pacchetto redis e poi creare di default il comando redis-server.

Il dockerfile contiene i vari step di creazione:

- **FROM**: da che base

```
docker build .
[+] Building 1.0s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 38B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [1/2] FROM docker.io/library/alpine@sha256:e1c082e3d3c45cccac829840a25941e679c25d438cd
 => CACHED [2/2] RUN apk add --update redis
 => exporting to image
 => => exporting layers
 => => writing image sha256:4f89c307d34ce8e791a8322af18b56d270761cc873c4c73ca0b49f68f47e8d
jacopo@DESKTOP-1QEP600:/mnt/c/Users/jacop/Documents/Github/Appunti-vari/Docker and Kuberne
docker run
alpine/git                busybox:latest            hello-world               redis:latest
alpine/git:latest         docker101tutorial         hello-world:latest
busybox                   docker101tutorial:latest  redis
jacopo@DESKTOP-1QEP600:/mnt/c/Users/jacop/Documents/Github/Appunti-vari/Docker and Kuberne
-server-image$ docker run 4f89c307d34ce8e791a8322af18b56d270761cc873c4c73ca0b49f68f47e8ca4
1:C 19 Sep 2021 13:31:26.156 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 19 Sep 2021 13:31:26.156 # Redis version=6.2.5, bits=64, commit=af329dfc, modified=0,
1:C 19 Sep 2021 13:31:26.156 # Warning: no config file specified, using the default confi
-server /path/to/redis.conf
1:M 19 Sep 2021 13:31:26.157 * monotonic clock: POSIX clock_gettime
1:M 19 Sep 2021 13:31:26.158 * Running mode=standalone, port=6379.
1:M 19 Sep 2021 13:31:26.158 # Server initialized
1:M 19 Sep 2021 13:31:26.158 # WARNING overcommit_memory is set to 0! Background save may
issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the comma
 take effect.
1:M 19 Sep 2021 13:31:26.158 * Ready to accept connections
```

Figura 3.1: docker build

- **RUN**: eseguire un comando durante la preparazione dell'immagine

- **CMD**: cosa verrà lanciato all'avvio

3.1 Cos'è un'immagine di base?

Immaginiamo di dovere installare un programma senza avere un sistema operativo installato, quale sarebbe il primo passo? Installare un sistema operativo.

3.2 E se modificassimo dockerfile?

Nel caso di una modifica e di una successiva build docker userebbe la cache per partire dall'ultimo punto valido, in questo modo risparmierebbe tempo e risorse.

3.3 Taggare un'immagine

docker build -t <docker ID/nome della repo/versione> permette di creare un tag per l'immagine. Solitamente questa è la sequenza usata, la versione tendenzialmente è "latest".

In questo modo potremmo usare un'immagine specifica tramite il tag usato, soprattutto se scaricato da docker hub.

3.4 docker commit

docker commit -c 'CMD <comando>' <container id> fondamentalmente permette di creare un'immagine da un container attivo, fa ciò che fa un dockerfile ma dopo svariate modifiche dentro ad un container.

Capitolo 4

docker compose

docker-compose permette di mettere in contatto più container senza dovere ripetere ogni volta gli stessi comandi. docker-compose sfrutta un file yml chiamato **docker-compose.yml**.

```
version: '3' # versione di docker-compose
services:
  redis-server:
    image: 'redis' # immagine su cui basarsi
  node-app:
    restart: always # riavvia automaticamente il
       ↪ servizio se si interrompe, on-failure si
       ↪ avvisa solo se l'exit code non  0
    build: . # cartella da costruire
    ports:
      - "4081:8081" # porta dell'host:porta del
         ↪ container
```

docker-compose si occupa di creare questi due container all'interno dello stesso network, in questo modo non serve collegarli manualmente.

Invece di usare ***docker run*** <***immagine***> ci basterà dire ***docker-compose up***.

Nel caso volessimo ricostruire le immagini ci basterebbe mettere ***docker-compose up –build***.

docker-compose down spegne tutti i container assieme.

docker-compose ps elenca i container attivi nella cartella.

Capitolo 5

docker volumes

I volumi sono semplicemente riferimenti a cartelle dell'host, in questo modo possiamo permettere un'interazione tra i due in maniera diretta. Il comando è ***docker run -p <porta host>:<porta container> -v <cartella relativa del volume> -v $(<nome all-interno del volume>:/<cartella host> <image ID>***.