moku
www.moku.io

Immagini - andiamo più a fondo

Fabio Ros

Quando scarichiamo un immagine digitando il suo nome abbiamo l'impressione di scaricare un pacchetto unico.

Docker rende trasparente l'esistenza dei layer.

Meccanismo di **cache dei layer**

Tuttavia, quando scarichiamo un immagine, capita di vedere che alcune parti di essa sono già presenti in cache.

UNION FILE SYSTEM

docker image history

Mostra l’elenco delle modifiche apportate all’immagine attraverso i layer che la compongono

Alcuni layer non modificano la dimensione dell'immagine perché aggiungono solo metadati e non modificano i file dell'immagine

Questi layer sono quelli che, nel comando docker image history, compaiono con la colonna "SIZE" valorizzata a "0B" (zero Byte)

Ogni immagine “parte” da un layer vuoto chiamato “scratch”.

In inglese “from scratch” significa “da zero”.

A dire il vero un immagine può partire anche da un’altra immagine, che può partire da un’altra immagine, e così via.

La prima immagine della catena, però, partirà da “scratch”.

“Ogni modifica” apportata all’immagine a partire da scratch, genera un nuovo layer.

Quando parleremo di Dockerfile, costruiremo i layer di un’immagine e capiremo nel dettaglio come lavorare al meglio con essi.

Ogni layer è identificato da un hash (SHA256) univoco.

Vantaggio: Sia in upload verso il registry, sia in download da esso, non dobbiamo mai caricare o scaricare un layer che è già presente nell'altro lato della comunicazione.

Condivisione dei layer

Come si riflette nei container? Conflitti sui dati?

Quando viene eseguito un container, Docker crea per esso un nuovo layer accessibile in lettura e scrittura

Il container potrà accedere solo in lettura ai file dell’immagine evitando ogni tipo di conflitto.

COW - Copy on Write

Avviene quando un container vuole modificare un file proveniente dall’immagine.

Siccome il file dell’immagine e quello del container divergono, questo file viene copiato dall’immagine al layer abilitato al read/write.

Fino a quel momento, quel file veniva letto direttamente dall’immagine

docker CLI - prima e dopo

Es. “docker image inspect” : mostra i metadati di un’immagine

prima: docker inspect alpine

dopo: docker **image** inspect alpine

Sono stati creati dei sotto-comandi. Per capirne di più basta guardare l’help della CLI di docker semplicemente eseguendo: “docker”

Nomi delle immagini

Nomi delle immagini

Quando scarichiamo un’immagine non scriviamo il suo hash
ma il suo nome

Nomi delle immagini

Come viene associato un nome ad un'immagine?

Per capirlo abbiamo bisogno del concetto di **Image Registry**

Image Registry

- Stoccaggio delle immagini
- Ogni immagine appartiene ad un repository che:
  - può contenere più immagini
  - può contenere più versioni della stessa immagine
  - utilizza un tag per contraddistinguere le immagini
    - anche se la chiave effettiva è l'hash ottenuto mediante SHA-256
    - i tag funzionano a la git, identificano una versione del repository

Image Registry > Tipologie

- Ufficiale: Docker Hub (hub.docker.com)
- Terze Parti
- On Premise (Installazione su macchine proprietarie)

Image Registry > Tipologie > Docker Hub

I repository di Docker Hub si dividono in ufficiali e non ufficiali:

ufficiali: Sviluppati e/o mantenuti dal team di Docker

es. https://hub.docker.com/_/alpine/

non ufficiali: A cura di figure terze, con un account Docker Hub

es. https://hub.docker.com/dockerdazero/alpine-sleep5

Ora che sappiamo cos'è un Registry, Repository e TAG.

```
docker pull <repository_ufficiale>:<tag>

docker pull <nome>/<repository_non_ufficiale>:<tag>
```

nome: nome o organizzazione del relativo account su docker hub

Quale versione dell'immagine viene scaricata all'esecuzione di `docker image pull <repository>` se non specifico il tag?

TAG **latest**

In caso di assenza di tag, Docker implicitamente userà il tag “latest”.

Attenzione! Non è scontato che latest sia il tag relativo all’ultima versione dell’immagine. Molti repository utilizzano tag alpha, beta, edge per indicare le ultime versioni disponibili.

Proprio come in git, ad un immagine possono essere associati molti tag.

È facile accorgersi di ciò guardando l’output di docker image ls: osserveremo un’occorrenza per ogni tag, ma l’hash (IMAGE ID) rimarrà lo stesso

Immagini con più tag

Eseguendo

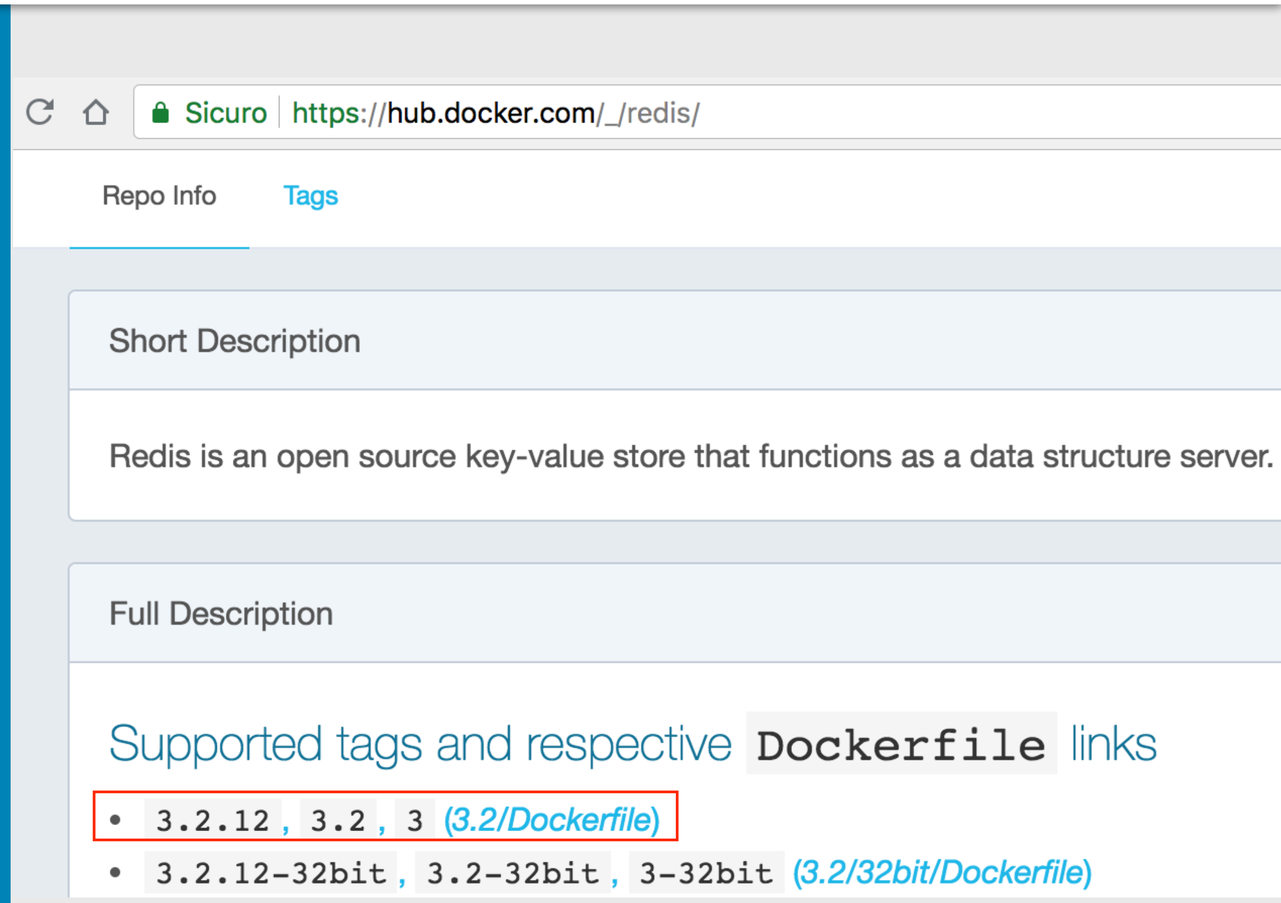
```
docker pull redis:3.2.12
```

```
docker pull redis:3.2
```

```
docker pull redis:3
```

Troverò tra le mie immagini 3 versioni, ma con lo stesso hash.

Sono tutte la stessa immagine, a cui sono stati assegnati tre tag diversi.



Immagini con più > Esempio con redis

```
[fabioros@FabioRos-moku-io [~] $docker pull redis:3.2.12
3.2.12: Pulling from library/redis
4d0d76e05f3c: Pull complete
aa39ed64d5b2: Pull complete
1593212ab2f2: Pull complete
19c40eafc076: Pull complete
1d372aabffb2: Pull complete
7ea1b1943ccb: Pull complete
Digest: sha256:b6da2922644fd139924f3814142758040a1fad5d3592cf9e5b8d9643a65fda90
Status: Downloaded newer image for redis:3.2.12

[fabioros@FabioRos-moku-io [~] $docker pull redis:3.2
3.2: Pulling from library/redis
Digest: sha256:b6da2922644fd139924f3814142758040a1fad5d3592cf9e5b8d9643a65fda90
Status: Downloaded newer image for redis:3.2
[fabioros@FabioRos-moku-io [~] $docker pull redis:3
3: Pulling from library/redis
Digest: sha256:b6da2922644fd139924f3814142758040a1fad5d3592cf9e5b8d9643a65fda90
Status: Downloaded newer image for redis:3
[fabioros@FabioRos-moku-io [~] $docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               3                   3df5dba00782        36 hours ago        99.7MB
redis               3.2                 3df5dba00782        36 hours ago        99.7MB
redis               3.2.12              3df5dba00782        36 hours ago        99.7MB
```