Docker: come "inscatolare" la propria app?

JUG TAA - giugno 2015

Cristian Consonni e Mario A. Santini

Parte 2

mettiamoci dentro le mani

Installare Docker

- Ubuntu:

```
$ wget -qO- https://get.docker.com/ | sh
```

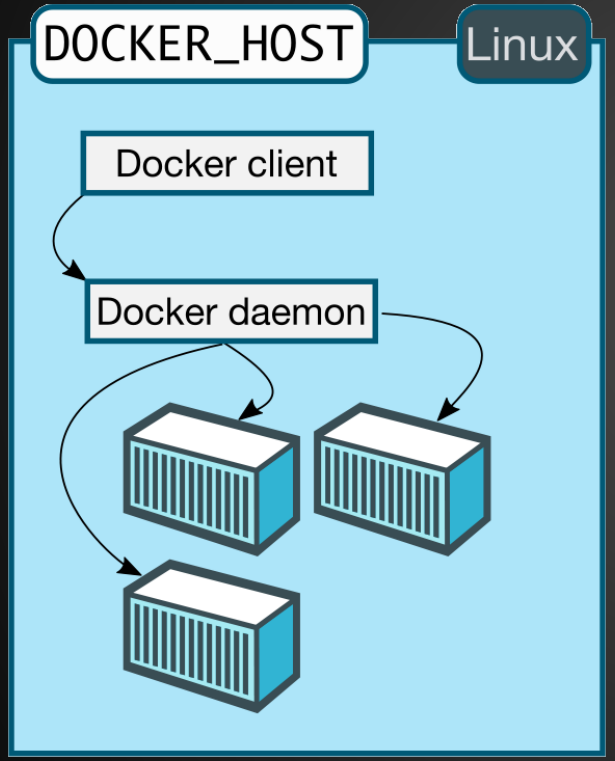
(versione >= 14.04 per versioni precedenti vedere i prerequisiti su: https://docs.docker.com/installation/ubuntulinux/)

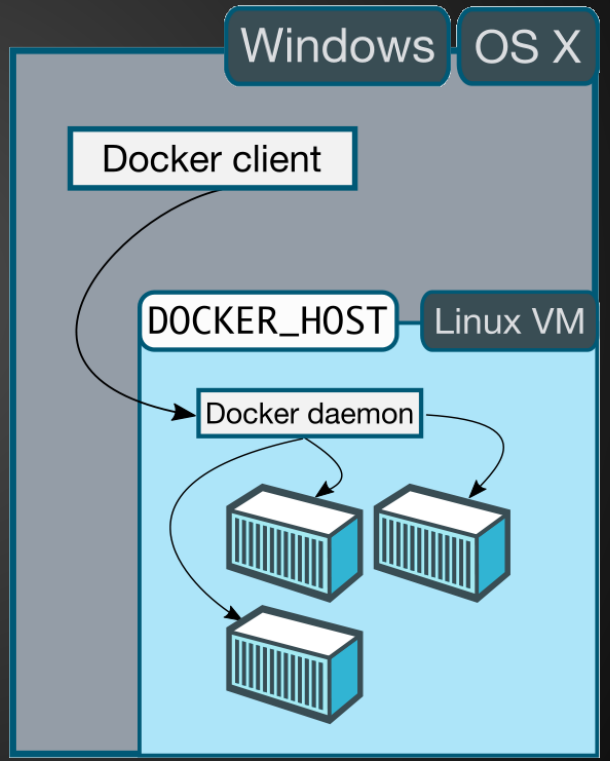
(`$ sudo usermod -aG docker ubuntu`)

(È anche pacchettizzato: `$ sudo apt-get install docker.io`)

- Mac OS X: Boot2Docker (CLI) o Kitematic (GUI)
- Windows: Boot2Docker (CLI)

Docker su Linux e su Mac/Win





Comandi del deamon di Docker (I)

`docker <subcommand>`

- `pull NAME[:TAG]`

scarica l'immagine `NAME[:TAG]` da DockerHub

- `run IMAGE [COMMAND]`

avvia un container con immagine `IMAGE` e esegue `COMMAND`

- `build PATH | URL | -`

costruisce una nuova immagine a partire dal Dockerfile in `PATH`

Comandi del deamon di Docker (II)

- `ps [-a]`

elenca i container che stanno girando, `-a` elenca tutti i container

- `images`

elenca le immagini disponibili localmente

- `rm`

rimuove un container

- `rmi`

rimuove un’immagine (non possono essere rimosse immagini in uso)

Proviamo: `hello world` **(II)**

```
docker ps -a
docker images
docker rm …
docker rmi …
```

(https://asciinema.org/a/b7j71yim7dewcka2w421s0398)

Qualcosa di più complicato: lanciare `bash` su Ubuntu

```
docker run -t -i ubuntu:latest
/bin/bash
```

(https://asciinema.org/a/5v36v9rxpvp4ywhqb4gk0bis5)

```
docker ps
```

(https://asciinema.org/a/333daetaiq7doh6cglbh8t3c8)

Comandi principali di un Dockerfile (I)

Dockerfile reference: https://docs.docker.com/reference/builder/

- `FROM <image>:<tag>`

Specifica la *Base Image* `<image>:<tag>` per le istruzioni successive

- `MAINTAINER <first_name> <last_name> "<email>"`

indica l'autore della nuova Docker image

Comandi principali di un Dockerfile (II)

- `RUN <command>`
- `RUN ["executable", "param1", "param2"]`

esegue un comando (crea un nuovo “layer”)

ATTENZIONE: i comandi eseguiti con RUN sono indipendenti

(quindi se voglio cambiare directory e eseguire un comando in quella directory devo specificarlo in un unico comando)

```
RUN cd $HOME/subdir && command
```

Comandi principali di un Dockerfile (III)

- `USER <user>`

Specifica il come utente che eseguirà i comandi dati con `CMD`

- `ENV <key> <value> (<key>=<value> …)`

Imposta il valore della variabile di ambiente `<key>` a `<value>`

- `EXPOSE <port> [<port>...]`

Informa Docker che il container ascolterà sulle porte `<port>` a runtime (si vedano le opzioni *-p* e *-P* del comando `docker run`)

Comandi principali di un Dockerfile (IV)

- CMD `["executable","param1","param2"]` (recommended)
- CMD `["param1","param2"]`
- CMD `command param1 param2`

Lo scopo di `CMD` è di fornire un default per l'esecuzione di un container.

ATTENZIONE: Ci può essere una sola istruzione `CMD` in un Dockerfile (se si usa più di una istruzione `CMD` solo l'ultima avrà effetto)

Una alternativa è usare l'istruzione `ENTRYPOINT`

Comandi principali di un Dockerfile (V)

- `ENTRYPOINT ["executable","param1","param2"]` (recommended)
- `ENTRYPOINT command param1 param2`

Un Dockerfile con il comando `ENTRYPOINT` permette di configurare un container che si comporta come un eseguibile.

Dal Dockerfile al container all'Hub

- `docker build \`
  `-t <my_hub_name>/<image_name>:<version_tag> PATH`

Costruisce l'immagine dandogli il nome specificato e la inserisce nel registry locale

- `docker push \ <my_hub_name>/<image_name>:<version_tag>`

Invia l'immagine a Docker Hub

Demo: web application con Spring

Tratto da: https://spring.io/guides/gs/spring-boot-docker

```
mvn package
```

(https://asciinema.org/a/665mytgmf3jwmpw6e1jgc0g3x)

```
mvn package docker:build
```

```
docker run -p 8080:8080 -t <image>
```

(https://asciinema.org/a/72n6hb7fwmzh5tge4smgudfcx)

(https://asciinema.org/a/6f6y8rgwm8sve7itq1mnpgp1u)

Opzioni di `docker run` (I)

- `-p <container_port>:<host_port>`

Linka la porta `<container_port>` alla porta `<host_port>`

- `-t`

Alloca uno pseudo-terminale

- `-i`

mantiene lo `STDIN` del container aperto anche se non è attaccato al terminale in uso

Opzioni di `docker run` (II)

- `--rm`

Rimuove automaticamente il container quando termina

- `-v <local_dir>:<container_dir>`

Monta una cartella locale nella directory specificata nel container (e. g. `$PWD:/mnt`)

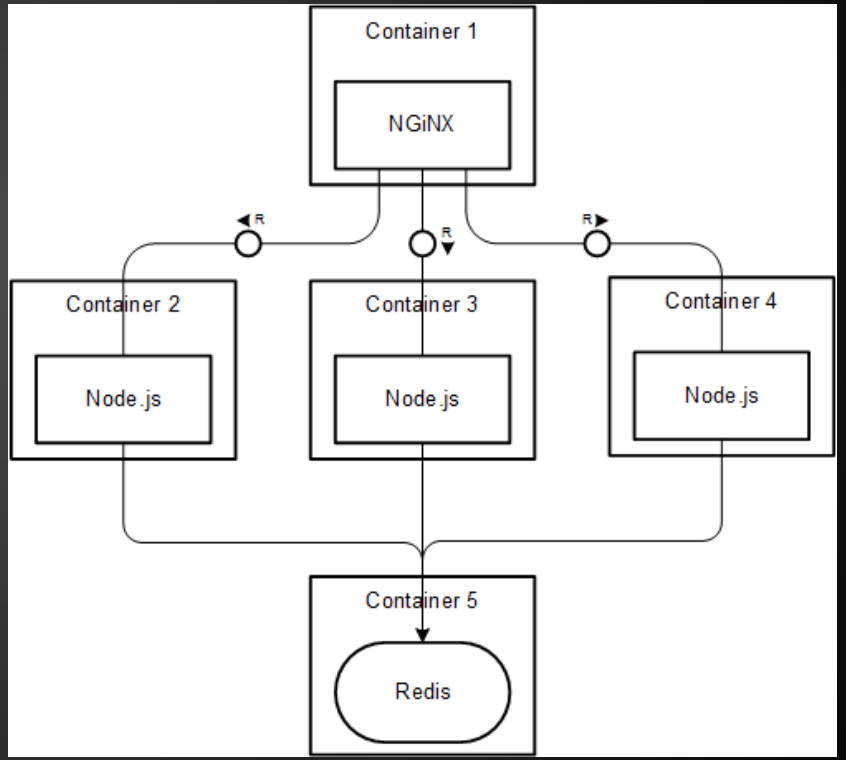
Extra

applicazioni multi-container

Redis + Node.js + nginx

Tratto da:

- http://anandmanisankar.com/posts/docker-container-nginx-node-redis-example/
- https://github.com/msanand/docker-workflow
- https://asciinema.org/a/aggiktwd4lqwe08ewywmpy10t



Docker Machine, Swarm, Compose

- Docker machine

permette di creare host Docker locali (VirtualB ox) o su provider remoti (DigitalOcean, Amazon EC2, Google Container Engine, Microsoft Azure, …)

- Docker Swarm

clustering nativo per Docker. Permette di creare ed accedere ad un pool di host Docker

- Docker Compose

è un tool per definire e lanciare applicazioni multi-container. I container sono descritti in un file con sintassi YAML e possono essere linkati tra loro.

Conclusioni

- Docker sfrutta la tecnologia dei *Linux Container* (`lxc`) per automatizzare il deploy di applicazioni
- I container forniscono degli ambienti completamente standardizzati (elimina il *“works on my machine”*)
- Si vuole arrivare a poter fare il deploy di applicazioni complesse con la stessa facilità sul computer dello sviluppatore e sul cluster in produzione
- Quello che abbiamo visto oggi è solo una *piccola parte* di un mondo in forte evoluzione

Grazie!

Cristian & Mario