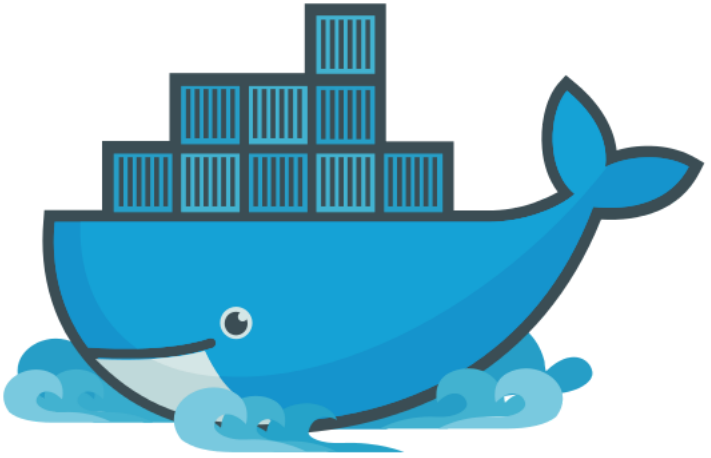
Introduzione a Docker



Davide Giunchi
davide@giunchi.net

Indice

- Storia
- Utilizzo nel mondo
- Funzionamento
- Uso command line base
- Debug
- Dockerfile
- Uso registry
- Docker-compose
- Consigli

Davide Giunchi

Sysadmin Linux da tempo ...

Free Software.
Php, Python, Perl.
servizi preferiti: http, smtp

La nascita

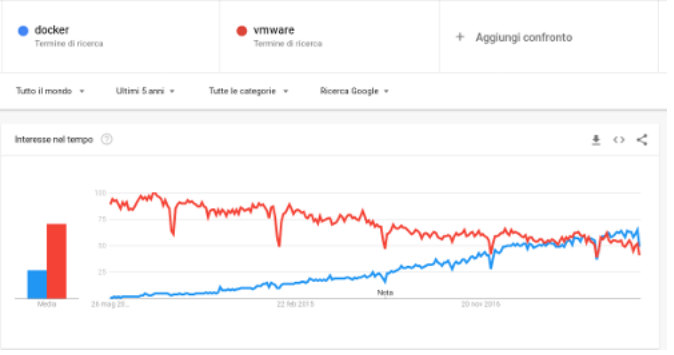
Solomon Hykes fondatore di dotCloud, in un talk alla Python Dev Conference in California nel 2013.
In poche settimane il programma ottiene grande risalto sulla stampa, viene reso open source (Apache 2.0), pubblicato su github.
Tempo 1 anno e tutti avevano sentito parlare di Docker.

Alcuni casi d'uso: pokemon go, spotify, google (altra runtime)

Alcuni concetti erano già presenti da tempo: chroot (1979), FreeBSD Jail (2000), Solaris Zones (2004), LXC (2008)
Docker è una rivisitazione di queste tecnologie, con funzionalità aggiunte nel kernel fino al 2013, il tutto con una CLI semplice ed un ecosistema di software in espansione.

Google Trend

Trend di ricerche negli ultimi 5 anni:



Container più Comuni

The most common technologies running in Docker are:

1. **NGINX:** Docker is being used to contain a lot of HTTP servers, it seems. NGINX has been a perennial contender on this list since we began tracking image use in 2015.
2. **Redis:** This popular key-value data store is often used as an in-memory database, message queue, or cache.
3. **Elasticsearch:** Full-text search continues to increase in popularity, cracking the top 3 for the first time.
4. **Registry:** 18% of companies running Docker are using Registry, an application for storing and distributing other Docker images. Registry has been near the top of the list in each edition of this report.
5. **Postgres:** The increasingly popular open source relational database edges out MySQL for the first time in this ranking.
6. **MySQL:** The most widely used open source database in the world continues to find use in Docker infrastructure. Adding the MySQL and Postgres numbers, it appears that using Docker to run relational databases is surprisingly common.
7. **etcd:** The distributed key-value store is used to provide consistent configuration across a Docker cluster.
8. **Fluentd:** This open source "unified logging layer" is designed to decouple data sources from backend data stores. This is the first time Fluentd has appeared on the list, displacing Logspout from the top 10.
9. **MongoDB:** The widely-used NoSQL datastore.
10. **RabbitMQ:** The open source message broker finds plenty of use in Docker environments.

fonte: datadog.com

Cosa non è docker

- Software di virtualizzazione (vmware,kvm ecc)
- Cloud Platorm (OpenStack, Aws ecc)
- Configuration Management (Puppet,Ansible)
- Deployment Framework (Capistrano)
- Workload Managemnt (Mesos, Fleet)
- Development Environment (Vagrant)

La promessa

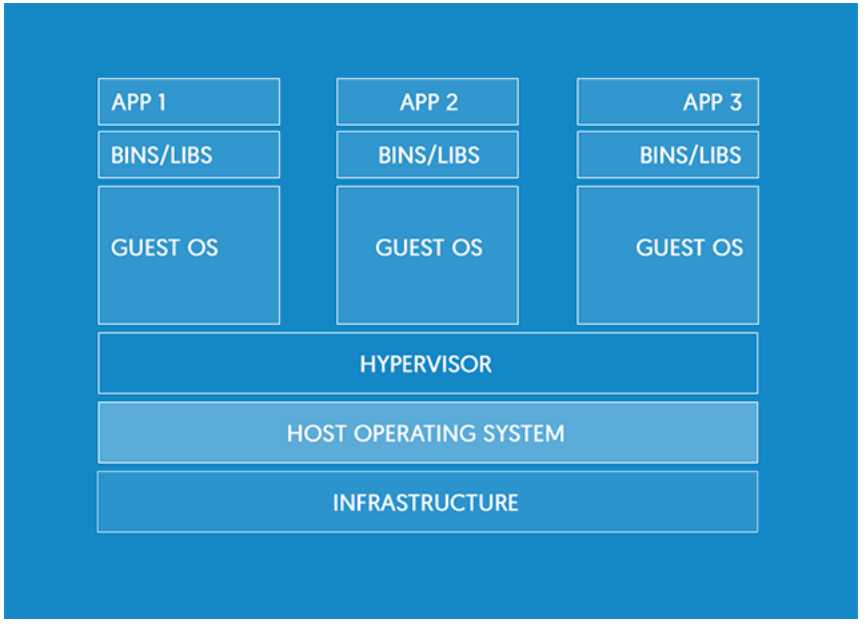
Alcuni pensano sia un tecnologia di virtualizzazione ma è molto più di questo.
I container hanno un differente approccio, tutti i container condividono un singolo kernel e l'isolamento è implementato interamente all'interno di questo singolo kernel, è chiamata operating system virtualization.

Docker nasce per rispondere alle difficoltà nella velocità ed efficacia dello sviluppo e delivery del software.

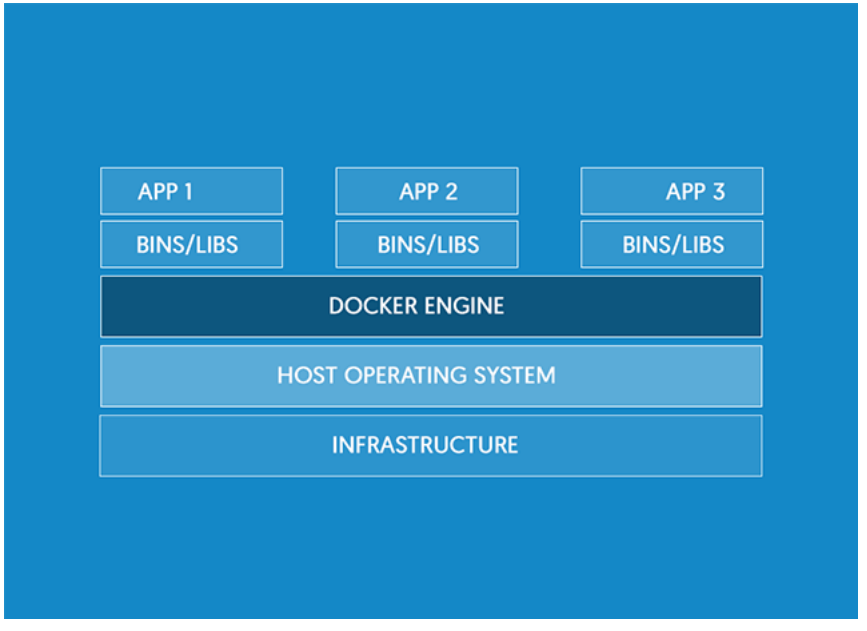
Tipico flusso: il dev sviluppa nel proprio pc, crea un pacchetto con l'applicazione e le librerie, invia all'operatore che scompatta il pacchetto all'interno di un ambiente gia' preparato di stg e poi prod.

Problemi: ambiente di dev e prod/stg sono diversi, lentezza nel delivery del software, il "pacchetto" è insufficente.
Gli ambienti si possono allineare, ma nel tempo cambieranno.

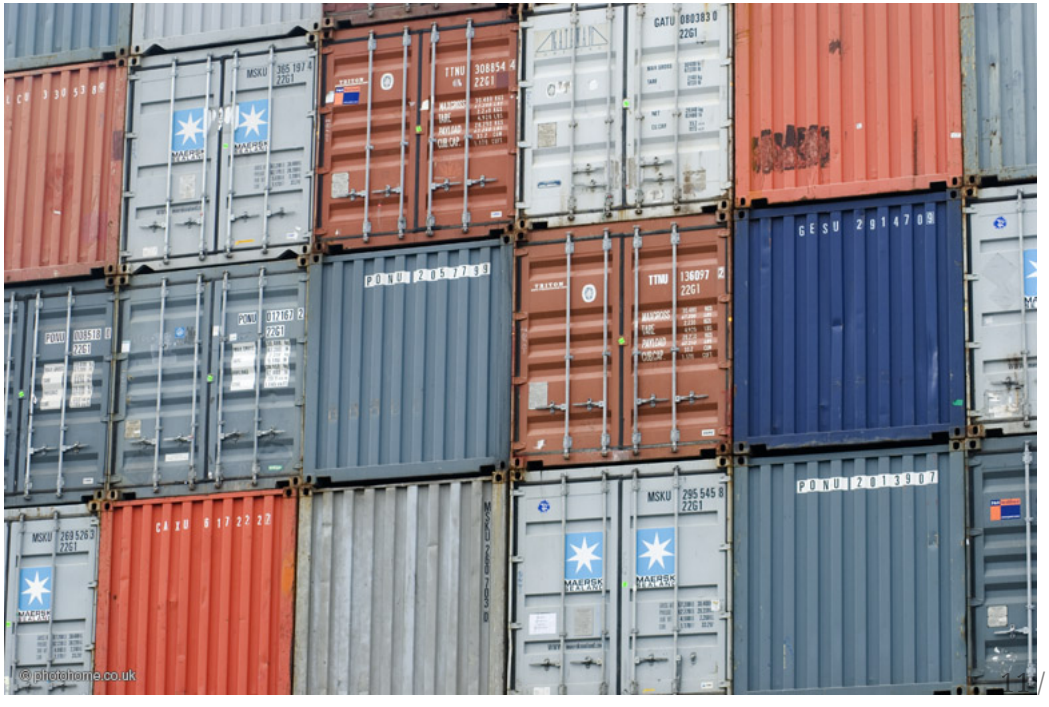
Funzionamento VM



Funzionamento Container



Caratteristiche Container



Un insieme ben riuscito

- **cgrous** funzione che limita, tiene traccia ed isola l'uso delle risorse.
  Inseriti nel kernel nel 2007, aggiornamento nel 2013.
  In docker, per ogni container viene creato un cgroup. Attualmente in sviluppo cgroups2
- **namespaces** partiziona le risorse in modo che un processo veda solo le risorse assegnategli: mount, process id, net, ipc, uts (host e domain name), User id
  Inseriti nel kernel nel 2002, aggiornamento nel 2006
- **LXC** virtualizzazione a livello di sistema operativo, inserita nel 2008.
  Inizialmente utilizzata docker, poi rimossa. Altre tecnologie: OpenVZ, chroot jails, Solaris Containers.

Caratteristiche Container

- Leggeri
- Immutabili
- Multistrato (filesystem Copy-On-Write)

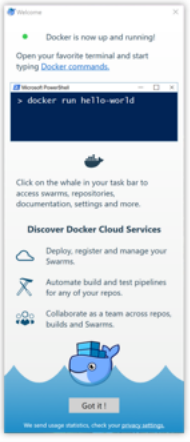
Tipico flusso di delivery:

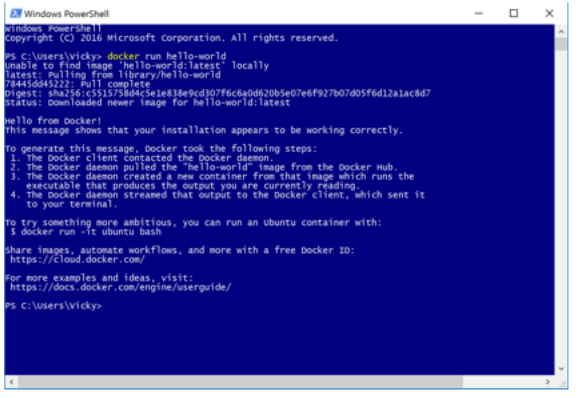
- il dev utilizza un container, ci sviluppa il codice, installa le librerie/dipendenze
- il dev effettua il build del container
- il dev effettua il push del container in un registry
- l'operatore scarica il container in stg/prod
- l'operatore imposta le variabili d'ambiente per le risorse esterne da utilizzare (db,files,smtp ecc)
- l'operatore esegue il container in stg/prod

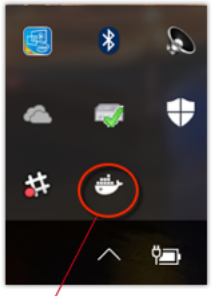
Tipico flusso di aggiornamento:

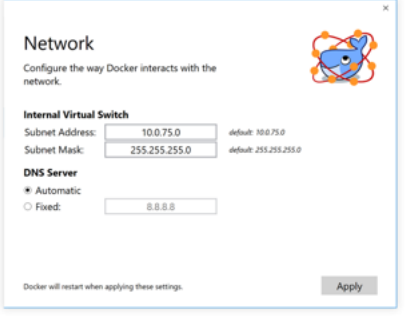
- il dev modifica il codice nel proprio pc, all'interno del container
- il dev effettua il build del container
- il dev effettua il push del container in un registry
- l'operatore scarica il container in stg/prod
- l'operatore esegue il container in stg/prod
- l'operatore ferma il container vecchio e lo distrugge

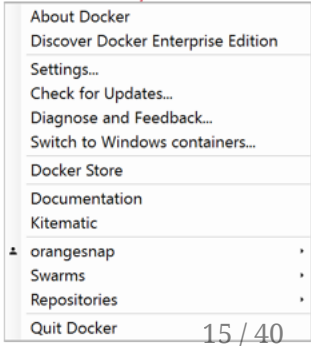
https://www.docker.com/docker-windows











Windows 10

- fino a Windows 7, è necessario VirtualBox (in bundle con l'installer docker), ricordati di impostare l'eventuale inoltro delle porte di rete nella vm di VirtualBox
- boot2docker è una vm linux che contiene docker preinstallato e configurato
- da Windows 10, è integrato con hyperv, non è più necessario nessun inoltro di porte

Installazione MacOSX

Installer su https://www.docker.com/docker-mac , oppure:

```
brew update
brew install caskroom/cask/brew-cask
brew install virtualbox
brew install docker docker-compose boot2docker
```

Installazione Linux

```
apt-get install docker.io docker-compose
```

Architettura

Scritto in GO, licenza Apache 2.0. Composto da due parti: il client ed il server, tutto in un'unico eseguibile.
Il server esegue e gestisce i container, il client impartisce comandi al server.
Server:

```
docker -d
```

Client:

```
docker info
docker run nginx
```

Porte di rete:

- tcp 2375 in chiaro
- tcp 2376 criptata (default in versioni recenti)

Networking

Il server docker fa da virtual bridge *docker0* tra il mondo ed i container che gestisce.
Ogni container ha il suo indirizzo ip assegnato alla sua interfaccia virtuale, quindi puoi effettuare il bind sulle stesse porte di altri container.
Per rendere accessibili all'esterno una o più porte, devi esporle con *-p*

Command line base

Esegui un container:

```
docker run nginx
docker run -p 8080:80 nginx
```

Visualizza i container attivi, terminali:

```
docker ps
docker stop nome_stato
```

Nomina i container:

```
docker run -p 8080:80 --name web nginx
docker ps
docker stop web
```

Debug

Avvia un container e collegati in console:

```
docker run --name centos centos:7
docker exec -ti centos /bin/bash
```

-t dice a docker di allocare una pseudo-tty, *-i* dice a docker di utilizzare una sessione interattiva.

Utilizzo spazio

Nota che, a meno che non si cancelli esplicitamente una immagine, questa resterà nel sistema per sempre, anche quando viene terminata. Per evitare che lo spazio utilizzato aumenti per sempre:

```
docker run --rm centos:7 echo OK
```

oppure su immagini non cancellate:

```
docker ps
docker rm nome_stato
```

Docker utilizza un filesystem COW: inizialmente era Aufs, ora il più utilizzato è overlay2 Esempio di attivazione container, creazione file, da host find in /var/lib/docker , stop+find, rm+find.

In prod o usi oltre al test, usa https://github.com/spotify/docker-gc di spotify .

rif: https://docs.docker.com/engine/reference/run/#clean-up---rm

Limitazione risorse

Limitazione a command line:

```
docker run --name centos --cpus 1 --memory 500M --rm centos:7 sleep 10000
docker exec -ti centos /bin/bash
```

All'interno del container:

```
yum install php-cli -y
truncate -s 1G /tmp/1G
php -d memory_limit=2G -r 'file("/tmp/1G");'
```

Ora proviamo con un file min 500Mb:

```
truncate -s 200M /tmp/200M
php -d memory_limit=2G -r 'file("/tmp/200M");'
```

Altre limitazioni:

- *cpu shares* da 0 a 1024, una specie di priorità
- i/o, network ecc

rif: https://docs.docker.com/config/containers/resource_constraints/#--kernel-memory-details

Persistenza dati

```
mkdir /tmp/html
echo "OK" > /tmp/html/index.html
docker run --name nginx --rm -p 8080:80-p 8080:80  -v /tmp/html:/usr/share/nginx/html nginx:
```

Tranne che in fase di DEV, il programma mettilo nel container, non in mount esterni.

Docker HUB

https://hub.docker.com

- privilegia le "Official"
- tante stars e tante pull = meglio
- clicca sul link Dockerfile per visualizzare il sorgente
- leggi le istruzioni

Creare immagini - Python

Crea una cartella vuota, con il sorgente ed i file che ti servono, + *Dockerfile*:

```
mkdir /tmp/webpy
vi /tmp/webpy/web.py
vi /tmp/webpy/Dockerfile
```

inserisci:

```
FROM python:2.7
MAINTAINER Davide Giunchi <davide@giunchi.net>
ADD ./* /
WORKDIR /
CMD ["python","web.py"]
```

build+esecuzione:

```
docker build -t example/webpy .
docker run -p 80:8000 example/webpy:latest
```

tip: usa ".dockerignore" con dentro *git*

Immagine con RUN e Layer

Dockerfile:

```
FROM centos:7
MAINTAINER Davide Giunchi <davide@giunchi.net>
ADD ./* /
WORKDIR /
RUN yum install python -y && \
rpm -ivh http://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm && \
yum install python-pip -y && pip install awscli
CMD ["python","web.py"]
```

build+esecuzione:

```
docker build -t example/webpy:2 .
docker run -p 80:8000 example/webpy:2
```

Immagine - PHP

index.php:

```
<?php phpinfo(); ?>
```

Dockerfile:

```
FROM php:7.2-apache
MAINTAINER Davide Giunchi <davide@giunchi.net>
COPY ./index.php /var/www/htm
```

build+run:

```
docker build -t example/myphp .
docker run -p 80:80 example/myphp:latest
```

Consigli per partire

- Parti con applicazioni stateless
- 1 container = 1 processo
- metti il programma all'interno del container, non fuori
- password e la configurazione delle risorse, mettila in variabili di ambiente
- Gli stati, falli gestire fuori
- usa immagini ufficiali, o cmq non esagerare

Registry privato

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

(sarebbe bene usare tipo -v /mnt/registry:/var/lib/registry x persistenza)
Scarica centos:7 dal docker hub:

```
docker pull centos:7
```

tagga l'immagine come *localhost:5000/my-centos*. Docker interpreta la prima parte come host, seconda come nome, utilizzato nel push:

```
docker tag centos:7 localhost:5000/my-centos
```

push dell'immagine nel registry locale privato:

```
docker push localhost:5000/my-centos
```

Rimuovi l'immagine locale in cache, scarica ed esegui l'immagine dal registry locale:

```
docker image remove centos:7
docker image remove localhost:5000/my-centos
docker run localhost:5000/my-centos sleep 1000
```

rif: https://docs.docker.com/registry/deploying/

Docker compose

(nota: dir *compose*)
Crea il file *docker-compose.yml*:

```
version: '2'

services:
  websrv:
    build:
      context: ./websrv/
    container_name: websrv
    ports:
      - "8080:80"
    links:
      - db
  db:
    image: mysql:5.7
    ports:
      - "3306:3306"
    volumes:
      - /tmp/db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=docker
      - MYSQL_DATABASE=docker
```

Ora crea il container websrv:

```
mkdir websrv
vi Dockerfile
```

Docker-compose - parte webserver

```
FROM php:7.0-apache
RUN docker-php-ext-install -j$(nproc) mysqli
COPY ./*.php /var/www/html/
```

i file php di esempio sono, index.php:

```
<?php
// host, user, pw, db
$mysqli = new mysqli('db', 'root', 'docker', 'docker');
echo 'Connected to MySQL';
?>
```

env.php:

```
<?php phpinfo(); ?>
```

Ora esegui il tutto:

```
cd ..
docker-compose up --build
```

Collegati alla porta 8080, anche url /env.php Altri comandi compose

```
docker-compose stop
```

Prova aggiungendo un file .php , rifacendo la build.

Errori tipici con JAVA

We need to understand that the docker switches (-m, –memory and –memory-swap) and the kubernetes switch (–limits) instruct the Linux kernel to kill the process if it tries to exceed the specified limit, but the JVM is completely unaware of the limits and when it exceeds the limits, bad things happen!

First, we need to recall what the JVM 9 ergonomic page says about “maximum heap size”. It says that it will be 1/4 of the physical memory. Since the JVM doesn’t know that it’s executing inside a container, it will allow the maximum heap size to be close to 260MB

rif: https://developers.redhat.com/blog/2017/03/14/java-inside-docker/

Immagini minimali

Alpine, molto usata per container minimali: https://www.alpinelinux.org/ .
Utilizza "musl libc" e busybox, occupa 8Mb. Ciclo di vita e supporto per 2 anni.
Distro utilizzata in ambiente embedded, anche container. Considera l'uso dei layer sul filesystem COW

Sicurezza

L'isolamento fornito dai namespaces non è lo stesso fornito da una vm. Il vantaggio dei container che li rende leggeri e veloci, è causa del suo "problema" di sicurezza: i container condividono lo stesso kernel dell'host, la ragione è che non tutto ciò che è nel kernel è protetto dai namespaces. I container non sono il sostituto di buone pratiche di sicurezza. La regola più importante è non eseguire i container come root:

```
FROM centos:7
RUN useradd -u 10001 utente
USER utente
RUN sleep 10000
```

Ora build, run ed esegui il ps da host

```
docker build -t /tmp/img/
ps auxw|grep sleep
```

Metodo a runtime:

```
docker run -ti --rm -u 1000 centos:7 sleep 100000
```

Altra regola: Solo gli utenti fidati devono poter eseguire il demone docker, altrimenti usa: "user_namespace" e "namespace.unpriv_enable".

rif: http://canihaznonprivilegedcontainers.info/

In produzione

Quando i container aumentano, ci vuole un orchestatore!

- Kubernetes - on premise, in cloud installato o gestito "saas"
- Docker Swarm
- Mesos
- DC/OS
- OpenShift (redhat, k8s)
- Rancher (k8s)
- Amazon ECS (saas)

Altre container runtime

- systemd-nspawn
- rocket
- cri-o
- ecc...

Letture consigliate

- "Docker: Up & Running" - O'reilly
- https://docs.docker.com