

Corso di Sistemi Distribuiti e Cloud Computing

Corso di Laurea Magistrale in Ingegneria Informatica A.A. 2019/2020
DIMES - Università degli Studi della Calabria



INTRODUCTION TO DOCKER CONTAINERS

Virtualizzazione

- E' una procedura informatica rivolta alla definizione di versioni virtuali di risorse hardware o software (es. dischi, blocchi di memoria, periferiche, processori, programmi e sistemi operativi), con la finalità di consentirne un pieno accesso e utilizzo come nel caso di quelle fisiche.
- Un sistema informatico virtuale viene chiamato "macchina virtuale" (VM) ed è un contenitore software totalmente isolato, dotato di sistema operativo e applicazioni.
- Ogni macchina virtuale è completamente indipendente.

Virtualizzazione

- La collocazione di più **macchine virtuali** su un singolo computer consente l'esecuzione di più sistemi operativi e applicazioni su un unico server fisico.
- Le risorse hardware fisiche e i software vengono resi virtuali e allocati in maniera indipendente fra gli utenti grazie all'uso di software chiamato "**hypervisor**" o "**virtual machine monitor**".
- L'hypervisor ha il compito di presentare all'utente i sistemi operativi delle macchine guest e di gestire la loro esecuzione.

Virtualizzazione - Vantaggi

- **Partizionamento:** esecuzione di più sistemi operativi su una macchina fisica
 - Suddivisione delle risorse di sistema tra le macchine virtuali
 - Possibilità di testare sistemi operativi o software quando un hardware dedicato non è disponibile
- **Isolamento:**
 - Isolamento di guasti e problemi di sicurezza a livello di hardware
 - Protezione delle prestazioni grazie a controlli avanzati delle risorse
- **Incapsulamento**
 - Salvataggio su file dell'intero stato di una macchina virtuale
 - Spostamento e copia delle macchine virtuali con estrema facilità, in modo analogo ai file
- **Indipendenza dall'hardware**
 - Provisioning o migrazione delle macchine virtuali a qualsiasi server fisico

Virtualizzazione - Esempi

- Esempi di piattaforme di virtualizzazione: VMWare, Virtualbox, QEMU, Microsoft Virtual PC, ProxMox, OpenVZ, ecc.
- Nel Cloud computing, tutti i server vengono virtualizzati e distribuiti sotto forma di una macchine virtuali



VirtualBox

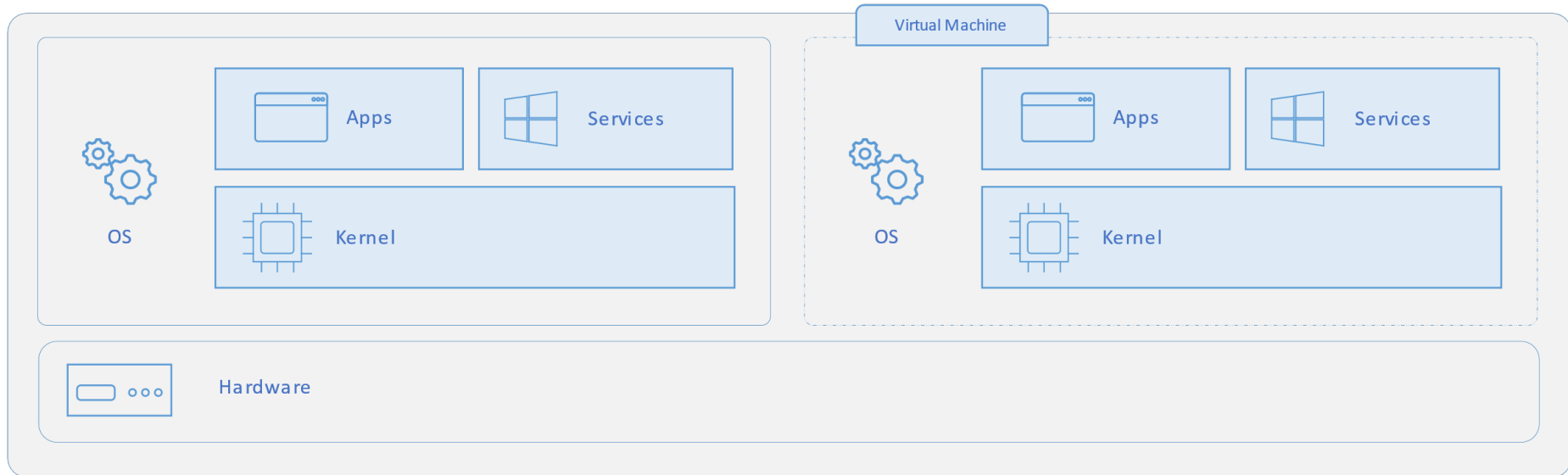


PROXMOX



Virtual Machine

- All'interno di ogni macchina virtuale viene eseguito un sistema operativo guest univoco (es. Linux, Windows, Solaris).
- Le macchine virtuali con sistemi operativi diversi possono essere eseguite sullo stesso server fisico.
- Ogni macchina virtuale ha i propri file binari, librerie e applicazioni, con dimensioni spesso elevate (diversi GB).



Container

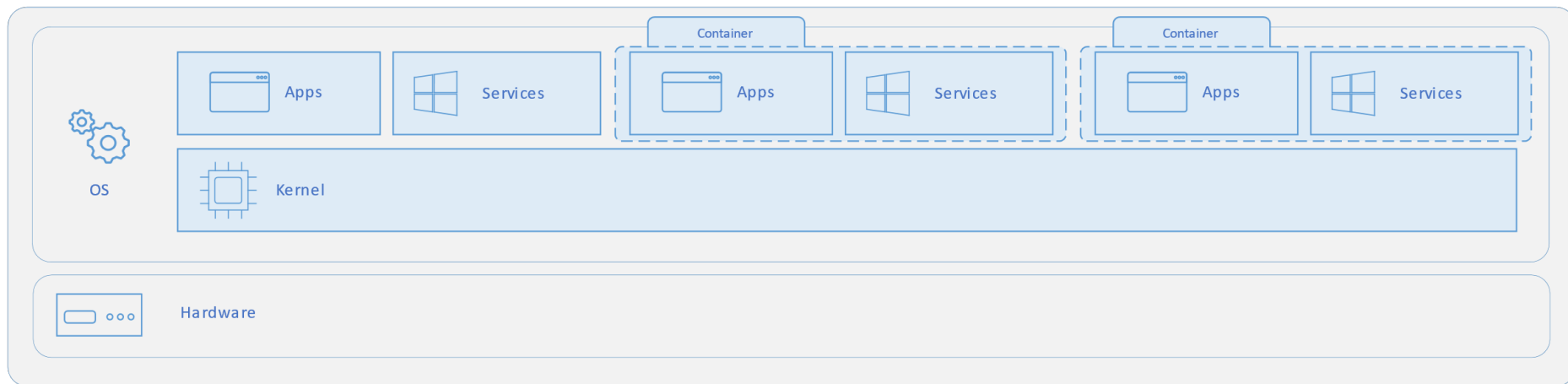
- La **containerizzazione** è un approccio allo sviluppo del software in cui un'applicazione o un servizio, le sue dipendenze e la sua configurazione sono raggruppati insieme come un'immagine del container.
- L'applicazione containerizzata può essere eseguita come istanza dell'immagine **container** nel sistema operativo host (OS).
- I **container** incapsulano ed isolano le applicazioni l'una dall'altra su un sistema operativo condiviso.

Perché usare i container?

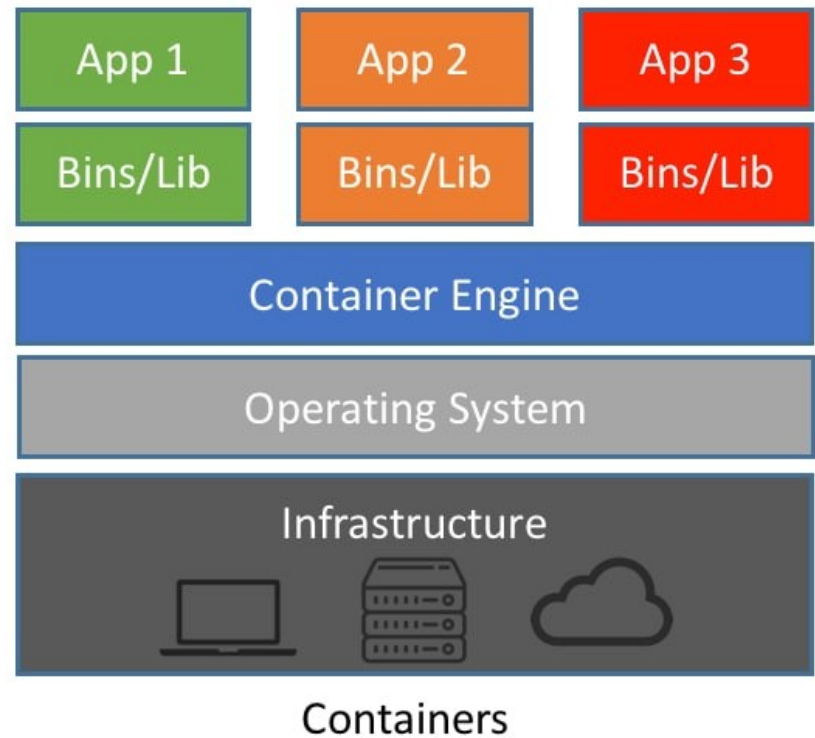
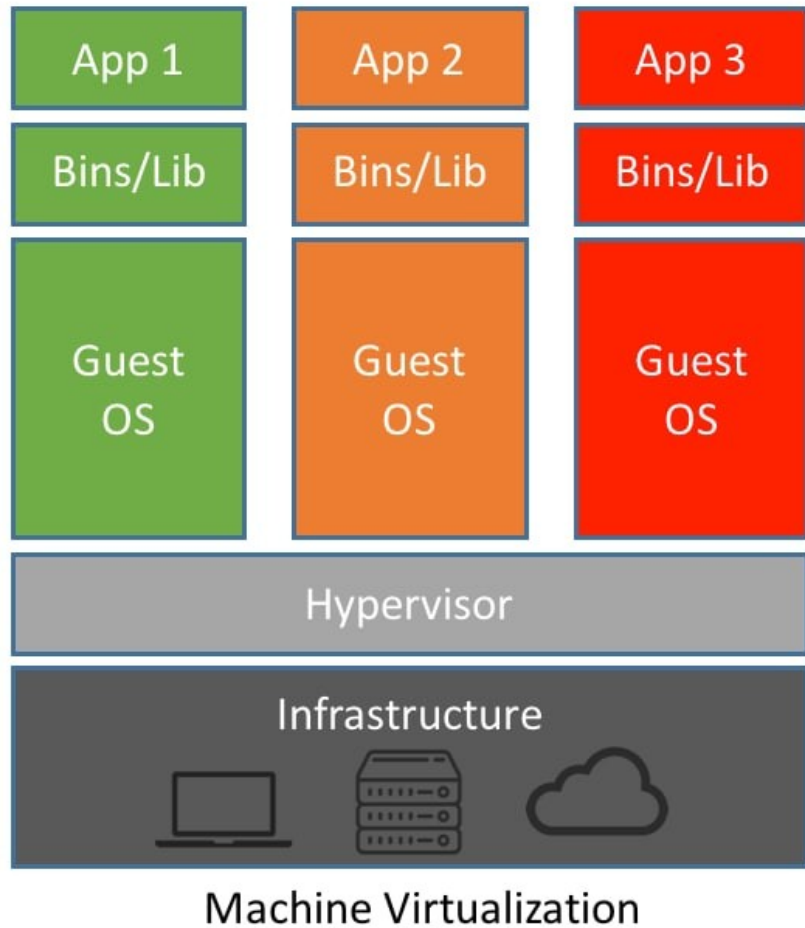
- ✓ Sono portabili e coerenti in tutti gli ambienti.
- ✓ Sono leggeri e veloci perché condividono il kernel del sistema operativo.
- ✓ Hanno un footprint notevolmente inferiore rispetto alle immagini di macchine virtuali (VM).
- ✓ I processi incapsulati in un container vengono eseguiti in isolamento sullo stesso sistema operativo.

Container vs Virtual Machine

- Un container è un "recipiente" isolato e leggero per l'esecuzione di un'applicazione sul sistema operativo host.
- I contenitori si basano sul kernel del sistema operativo host e contengono solo app, librerie, API e servizi leggeri del sistema operativo, che vengono eseguiti in modalità utente.

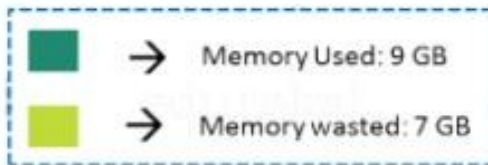
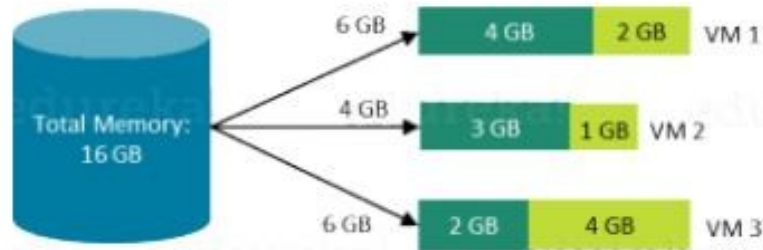


Container vs Virtual Machine



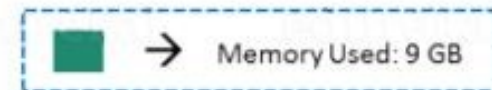
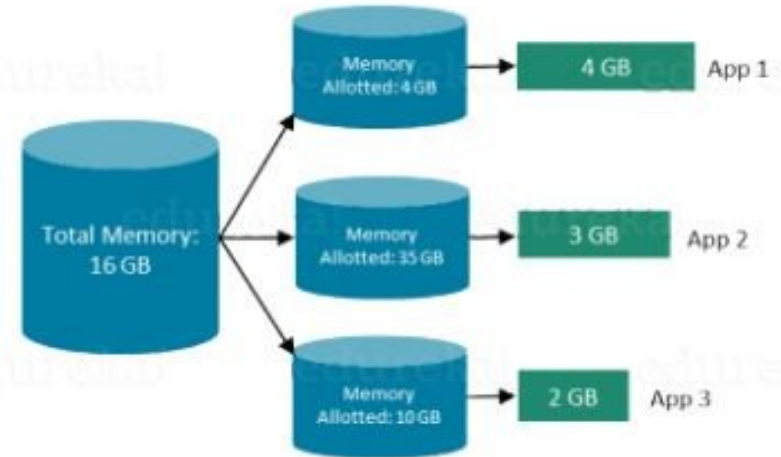
Container vs Virtual Machine

Virtual Machine



7 Gb of Memory is blocked and cannot be allotted to a new VM

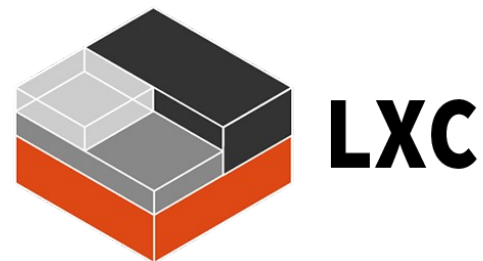
Container



Only 9 GB memory utilized;
7 GB can be allotted to a new Container

- **LXC** (*LinuX Containers*) è un ambiente di virtualizzazione che opera a livello del sistema operativo e permette di eseguire diversi ambienti Linux virtuali isolati tra loro (container) su una singola macchina reale avente il **kernel Linux**.

✓ Il **kernel** costituisce il nucleo o core di un sistema operativo, ovvero il software che fornisce un accesso sicuro e controllato dell'hardware ai processi in esecuzione sul computer.



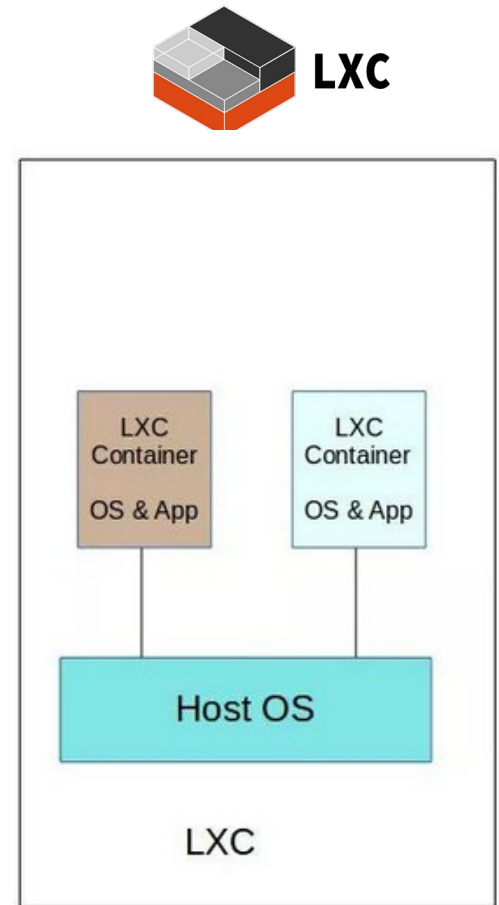
Architettura di LXC

L'architettura di LXC prevede due componenti principali: il sistema host e i container. Il sistema host è il sistema operativo Linux sottostante su cui è installato il software LXC, mentre i container sono le istanze di sistema operativo Linux **isolati** tra di loro e dal sistema host.

Il sistema host ospita i servizi di base e le risorse di sistema comuni a tutti i container (kernel, file system, memoria, CPU, rete). LXC si occupa di isolare queste risorse tra i vari container, garantendo un elevato livello di sicurezza e di isolamento tra di essi.

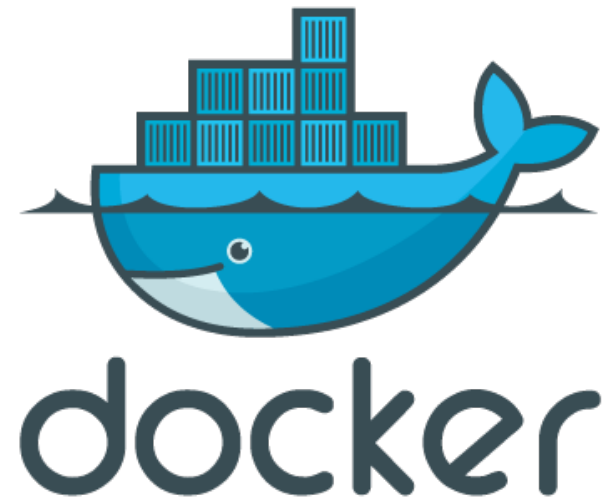
I container sono invece delle istanze di sistema operativo Linux che condividono il kernel del sistema host ma hanno il proprio spazio di sistema files, processi e rete isolati tra di loro e dal sistema host.

LXC garantisce l'isolamento dei container mediante **cgroups**, una funzionalità del kernel Linux che limita, tiene conto e isola l'utilizzo delle risorse (CPU, memoria, I/O su disco, rete, ecc.) di una raccolta di processi.



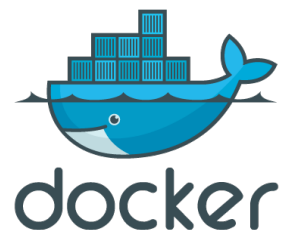
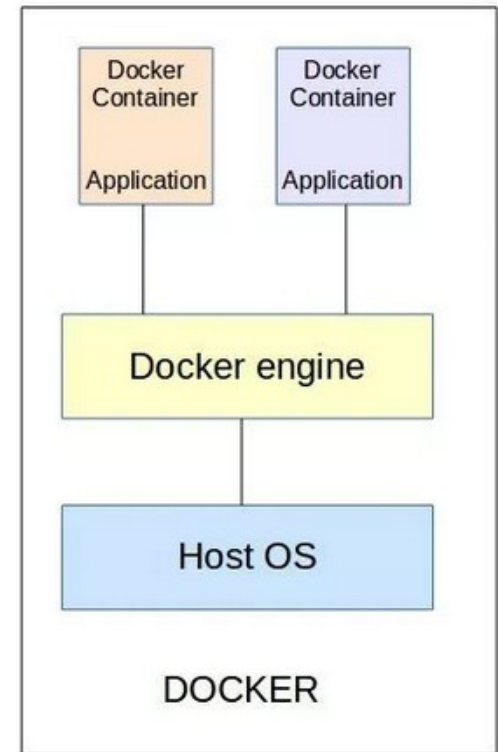
Docker

- **Docker** è il più popolare strumenti open-source che permette di pacchettizzare e distribuire un'applicazione e le sue dipendenze tramite container.
- Inizialmente utilizzava di default LXC per la virtualizzazione a livello di sistema operativo, per poi passare a **libcontainer** dalla v.0.9.
- **Libcontainer** è una libreria scritta in Go che consente di gestire il ciclo di vita del contenitore, dalla sua creazione, consentendo anche l'esecuzione di operazioni aggiuntive dopo la creazione del contenitore stesso.



Docker

- **Docker** fornisce un isolamento a livello di applicazione, quindi il container diventa un contenitore isolato per un'applicazione e tutte le sue librerie/dipendenze.
- Diversamente da LXC, Docker fornisce un unico ambiente di esecuzione fornito dal Docker Engine, indipendente dal sistema operativo host.
- **Maggiore portabilità:** Docker è progettato per essere altamente portabile, consentendo di creare un container in un ambiente e distribuirlo facilmente in un altro ambiente. LXC, invece, è più legato al sistema operativo host su cui è in esecuzione.

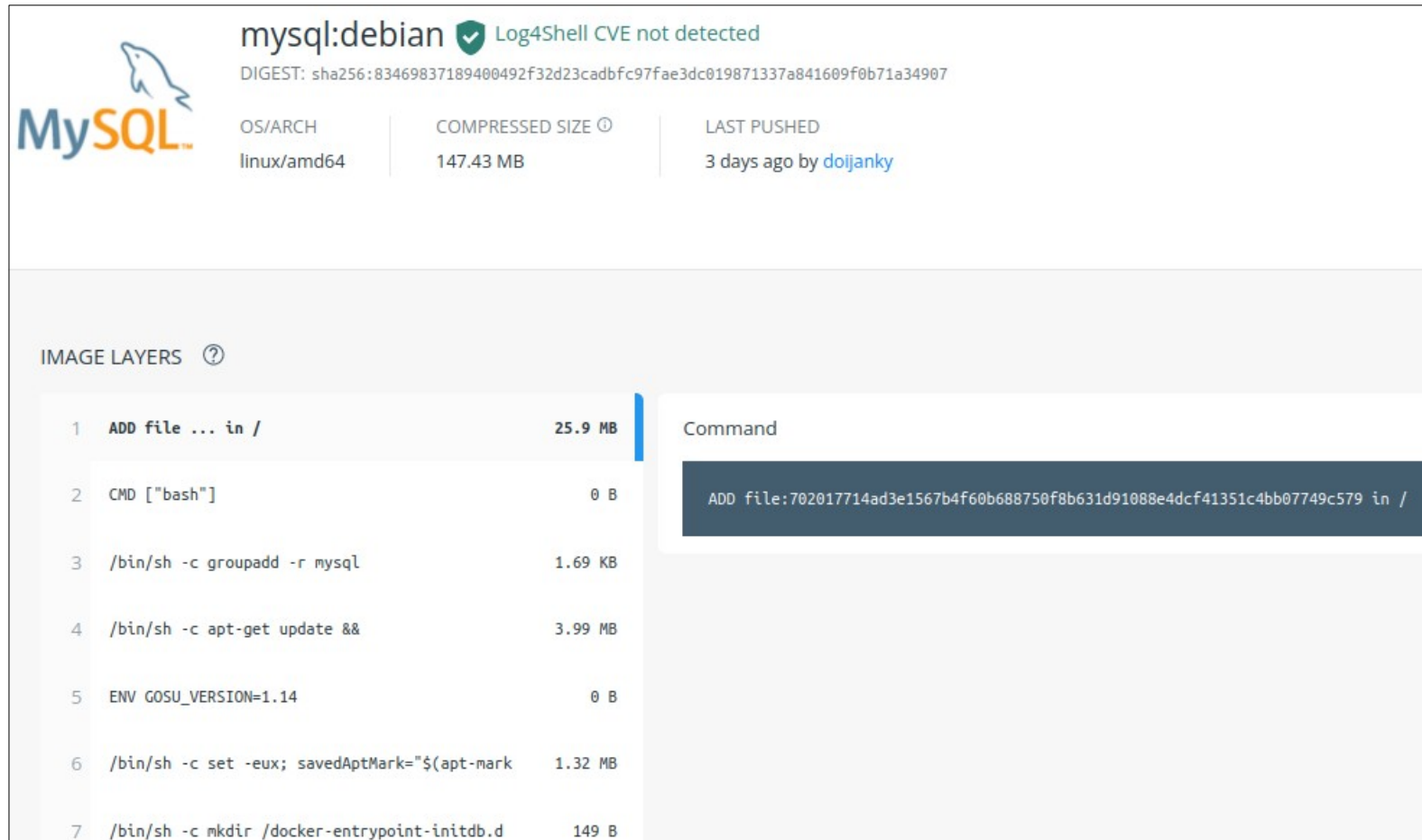


Container vs Image

- **Immagine:** template costituito da una serie di layer, che definisce come un container dovrà essere costituito una volta eseguito.
- E' definita mediante in file immutabile che contiene il source code, le librerie, le dipendenze ed altri strumenti necessari a far funzionare un'applicazione.
- Es: immagine di MySQL 5.6, su sistema operativo Debian
- Ogni **layer** rappresenta un cambiamento all'immagine di partenza, producendo un'**immagine intermedia**)
- Es. parto da un'immagine base di Ubuntu, poi aggiungo un layer eseguendo un comando su questa immagine, e così via.
- I layer consentono di velocizzare le fasi di build dell'immagine, migliorando la riutilizzabilità dei layer e risparmiando spazio sul disco (si memorizzano solo le variazioni per ogni layer).
- Un **container** non è altro che un'istanza di un'immagine.

Docker Image - Layer

- Ogni layer è distinto da un identificativo, mentre l'immagine finale, caratterizzata da uno specifico **tag**, è identificata da un **digest** (un hash code) univoco.
- Ogni operazione applicata ad un layer produce un altro layer.
- Se il tag non viene esplicitato si assume essere "latest".



The screenshot displays the Docker Hub interface for the `mysql:debian` image. At the top, the MySQL logo is visible alongside the image name and a security check status: "Log4Shell CVE not detected". Below this, the digest is shown: `DIGEST: sha256:83469837189400492f32d23cadbfc97fae3dc019871337a841609f0b71a34907`. The image's OS/ARCH is `linux/amd64`, its compressed size is `147.43 MB`, and it was last pushed `3 days ago` by `dojanky`.

The "IMAGE LAYERS" section is expanded, showing a list of seven layers. The first layer, "ADD file ... in /", is highlighted with a blue bar. To the right of this layer, a "Command" box displays the specific Docker command used to create this layer: `ADD file:702017714ad3e1567b4f60b688750f8b631d91088e4dcf41351c4bb07749c579 in /`.

Layer	Command	Size
1	ADD file ... in /	25.9 MB
2	CMD ["bash"]	0 B
3	/bin/sh -c groupadd -r mysql	1.69 KB
4	/bin/sh -c apt-get update &&	3.99 MB
5	ENV GOSU_VERSION=1.14	0 B
6	/bin/sh -c set -eux; savedAptMark="\$(apt-mark	1.32 MB
7	/bin/sh -c mkdir /docker-entrypoint-initdb.d	149 B

Download Image

```
(base) MACBE16107:~ sterbini$ docker pull jupyter/scipy-notebook
Using default tag: latest
latest: Pulling from jupyter/scipy-notebook
83ee3a23efb7: Already exists
db98fc6f11f0: Already exists
f611acd52c6c: Already exists
724e03a65ce0: Already exists
513e138b0e94: Already exists
48f0bd4aefb5: Already exists
4f4fb700ef54: Pull complete
b4de871b77c8: Already exists
0b03df09b2e6: Already exists
fa78867bfcd3: Already exists
47cf22ed95c5: Already exists
9ecce3ab59b4: Already exists
5ec66779f1e5: Already exists
cd7851ccfb49: Already exists
e80214e887dd: Already exists
b96af1630e18: Already exists
bec9edeb94f8: Already exists
dbb61cd8d6ce: Already exists
f880f4d488be: Already exists
6b4097a5c485: Pull complete
c5dd0250935f: Pull complete
f9c371873f19: Pull complete
Digest: sha256:6cb01b1d6653efada4841bcad3a1f8e168e21dccc38dbd6fac7095208872c96f
Status: Downloaded newer image for jupyter/scipy-notebook:latest
(base) MACBE16107:~ sterbini$
```

Tag dell'immagine

Hashcode identificativo del layer

Digest dell'immagine

Esempio Dockerfile

- An example of a Dockerfile for building an image based on official Ubuntu 18.04 with installing Nginx:

```
# Use the official Ubuntu 18.04 as base
FROM ubuntu:18.04
# Install nginx and curl
RUN apt-get update &&
    apt-get upgrade -y &&
    apt-get install -y nginx curl &&
    rm -rf /var/lib/apt/lists/*
```

- How to build an image from a Dockerfile:

```
# docker build -t <nome_image>:<tag_image> .

e.g.: ---> # docker build -t ubuntu18-04-nginx:v1.0 .
or
# docker build -t <nome_image>:<tag_image> -f Dockerfile
```

Installazione di Docker

- **Docker Engine** è disponibile su Linux, macOS e Windows 10 tramite Docker Desktop e come installazione binaria statica.

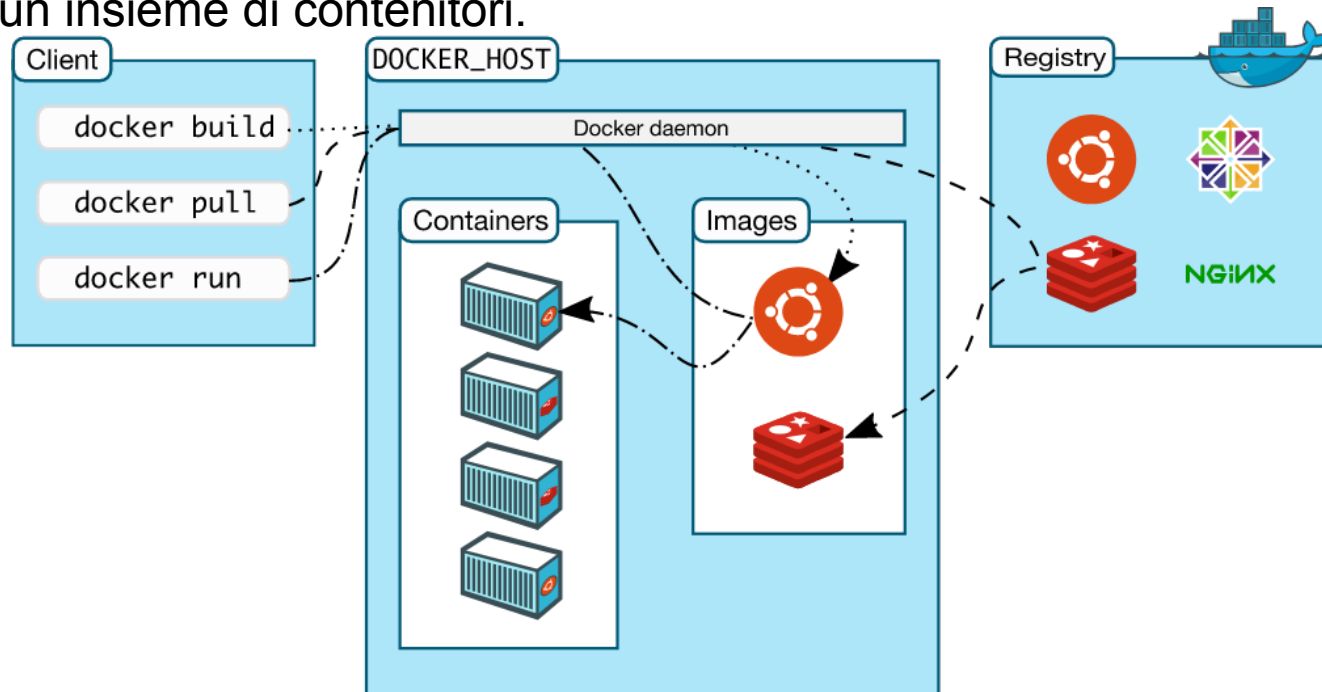
Platform	x86_64 / amd64	arm64 (Apple Silicon)
Docker Desktop for Mac (macOS)	✓	✓
Docker Desktop for Windows	✓	

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	s390x
CentOS	✓	✓		
Debian	✓	✓	✓	
Fedora	✓	✓		
Raspbian			✓	
RHEL				✓
SLES				✓
Ubuntu	✓	✓	✓	✓
Binaries	✓	✓	✓	

<https://docs.docker.com/engine/install/>

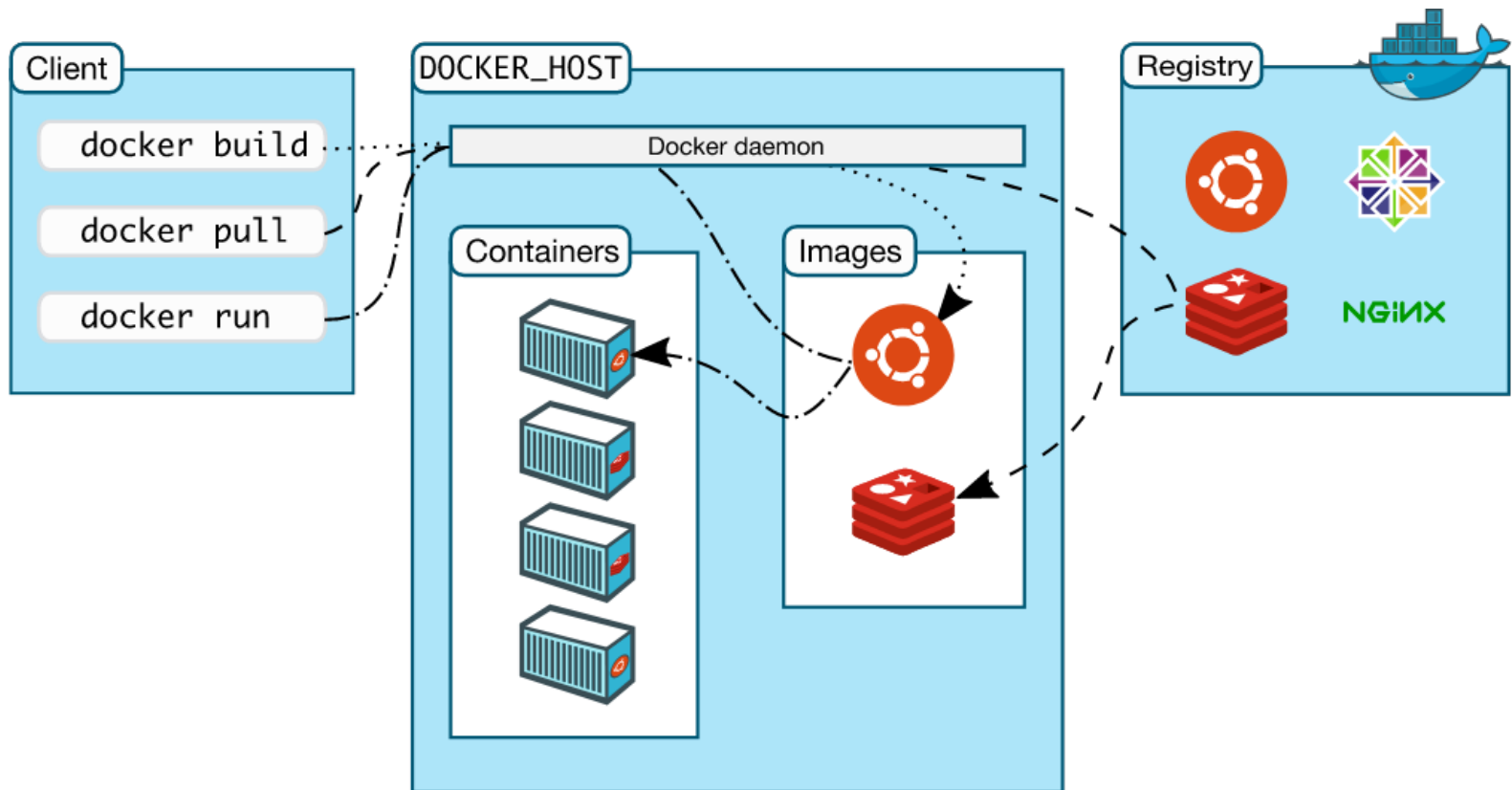
Architettura di Docker

- Docker utilizza un'architettura client-server. Il client Docker comunica con il daemon Docker (*dockerd*), che si occupa del lavoro di costruzione, esecuzione e distribuzione dei container Docker.
- Il client Docker e il daemon possono essere eseguiti sullo stesso sistema oppure puoi connettere un client Docker a un demone Docker remoto.
- Un altro client Docker è Docker Compose, che consente di lavorare con applicazioni costituite da un insieme di contenitori.



Architettura di Docker

- Il client può inviare una serie di comandi al demone Docker per richiedere:
 - la compilazione di un'immagine (build),
 - il download di un'immagine da un registry (run)
 - oppure l'esecuzione di un container come istanza di un'immagine (run).



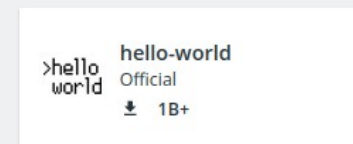
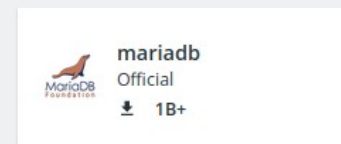
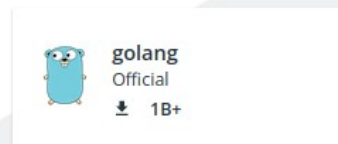
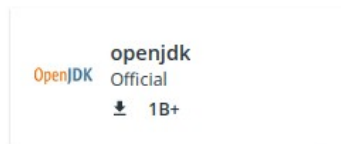
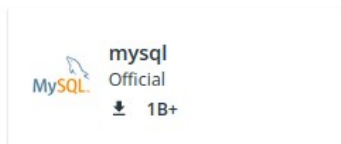
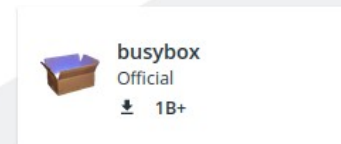
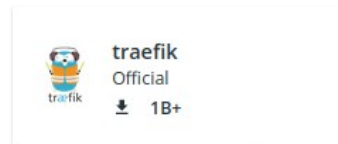
Docker Hub

- Docker Hub è un registro pubblico che chiunque può utilizzare e Docker è configurato per cercare immagini su Docker Hub per impostazione predefinita.
- E' possibile tuttavia usare dei registri privati.

<https://hub.docker.com/>

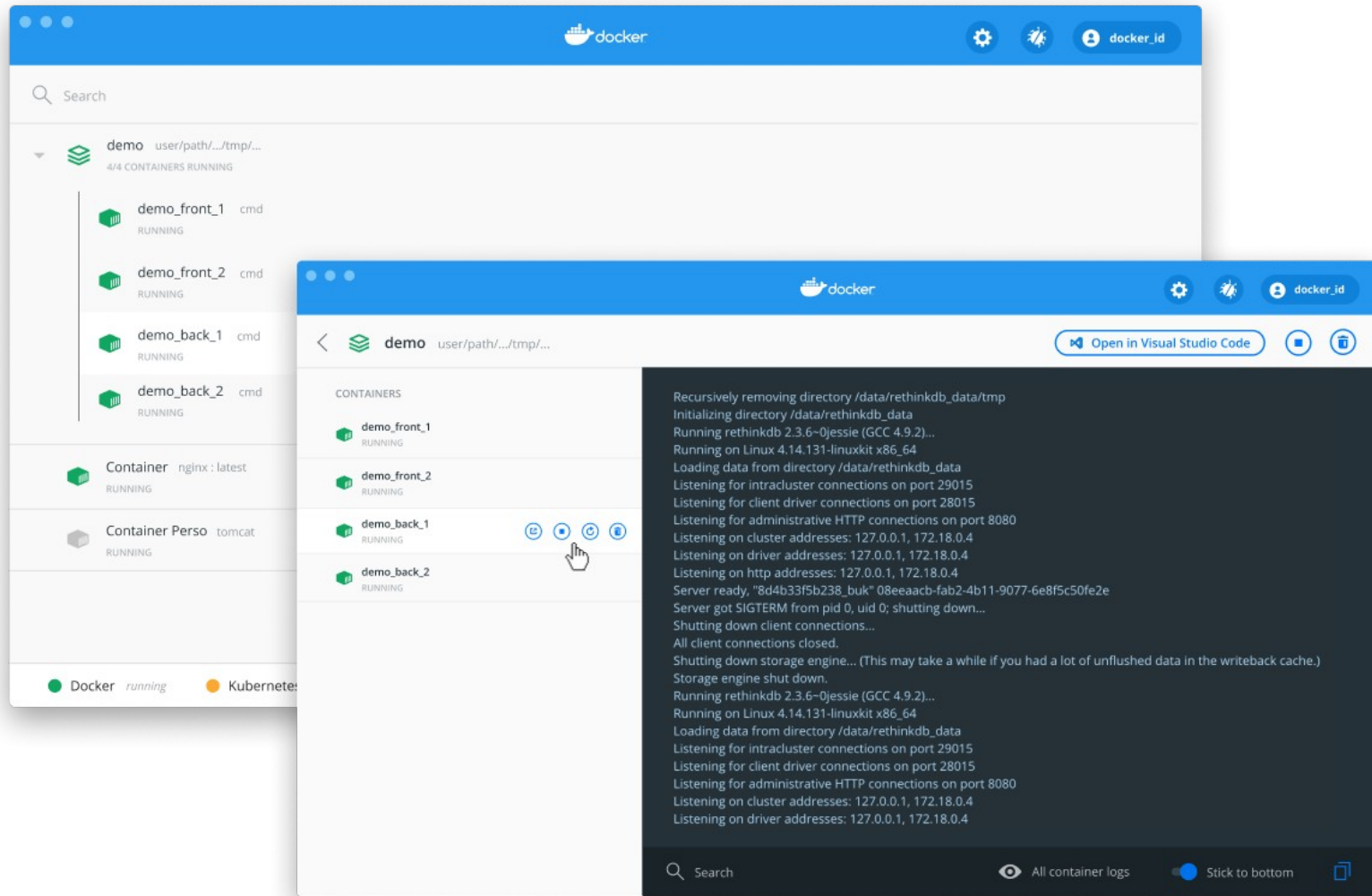
Docker Hub is the world's largest
library and community for container images

Browse over 100,000 container images from software vendors, open-source projects, and the community.



Docker Desktop

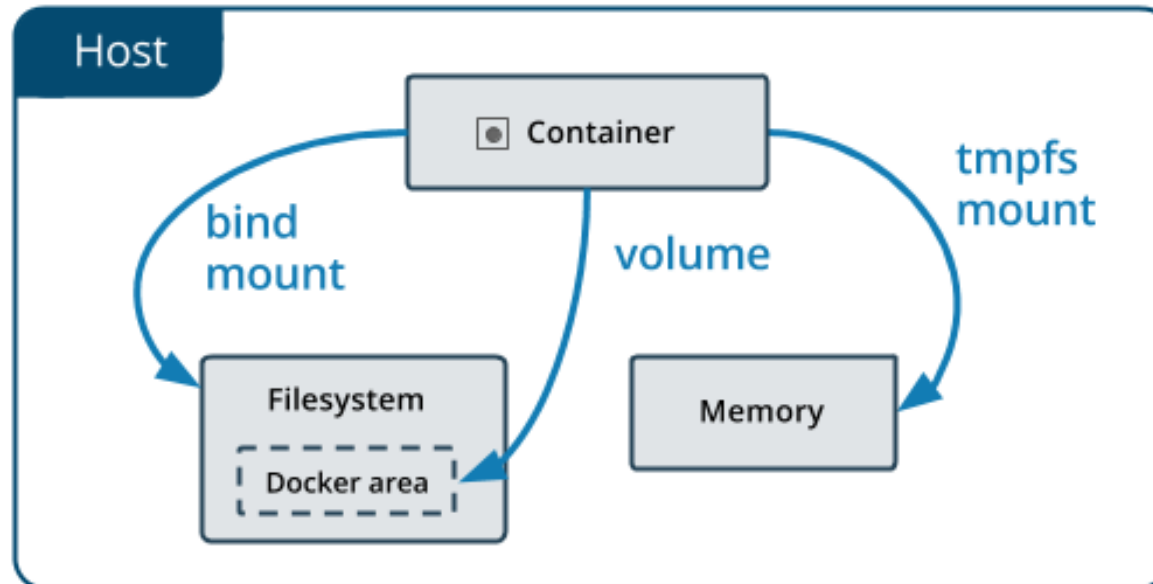
- **Docker Desktop** è un'applicazione per Mac e Windows che consente di gestire container Docker e immagini sul proprio pc.



Docker Volume

Esistono diverse soluzioni in Docker per la persistenza dei dati:

- I **volumi** sono archiviati in una parte del filesystem host che è gestito da Docker (es. /var/lib/docker/volumes/ su Linux).
- I **bind mount** possono essere archiviati in qualsiasi punto del sistema host. Possono essere modificati anche da processi esterni al container.
- I **mount di tmpfs** sono archiviati solo nella memoria del sistema host e non vengono mai scritti nel filesystem del sistema host.



Docker Volume

- I volumi hanno diversi vantaggi rispetto al bind mount:
 - È più facile eseguire il backup o la migrazione.
 - I volumi funzionano sia su contenitori Linux che Windows (non dipendono dal file system).
 - I volumi possono essere condivisi in modo più sicuro tra più contenitori.
 - I driver di volume consentono di archiviare volumi su host remoti o provider cloud, per crittografare il contenuto dei volumi o per aggiungere altre funzionalità.
 - I nuovi volumi possono avere il loro contenuto precompilato da un contenitore.
 - I volumi hanno prestazioni molto più elevate rispetto ai bind mount da host Mac e Windows.

Docker Network

- In container Docker possono essere interconnessi tramite delle reti.
- Di default, ciascun container è associato ad una rete **bridge**, costituita da una subnet (es. 172.17.0.0/16), che può gestire fino a un massimo di $2^{16} = 65536$ container.
- Docker gestisce in automatico l'assegnazione degli IP all'interno di una subnet.
- Con il driver di rete bridge, i container e l'host rimangono comunque isolati.
- Esistono anche altri driver per il networking in Docker, tra cui:
 - **none**: disabilita tutte le funzioni di reti per il container.
 - **host**: il container non è poiù isolato dall'host, ma ne condivide lo stack di rete; al container non viene assegnato un proprio indirizzo IP.
 - **overlay**: consente di connettere tra di loro container docker che si trovano su macchine host diverse; è usato spesso in ambiente distribuito con cluster di nodi che gestiscono container (es. Docker Swarm).

Docker Network

Command	Description
<code>docker network connect</code>	Connect a container to a network
<code>docker network create</code>	Create a network
<code>docker network disconnect</code>	Disconnect a container from a network
<code>docker network inspect</code>	Display detailed information on one or more networks
<code>docker network ls</code>	List networks
<code>docker network prune</code>	Remove all unused networks
<code>docker network rm</code>	Remove one or more networks

CLI di Docker

- **docker ps** visualizza la lista dei container attivi. Aggiungendo l'argomento -a visualizza tutti i container attivi e non
- **docker stop {CONTAINER}** ferma un container
- **docker rm {CONTAINER}** elimina container (deve essere precedentemente stoppato)
- **docker logs {CONTAINER}** log del container
- **docker images** lista immagini sul sistema
- **docker volume ls** lista dei volumi registrati sull'host
- **docker exec -it {CONTAINER} {COMMAND}** esegue comando (bash/sh) dentro un container
- **docker attach {CONTAINER}** entra nel terminal del container
- **docker rmi {IMAGE}** elimina la docker image (non devono essere connessi container attivi)
- **docker start {CONTAINER}** avvia un container creato in precedenza
- **docker inspect {CONTAINER}** restituisce tutti i parametri di sistema del container

CLI di Docker

- **docker volume prune** rimuove tutti i volumi non più usati dai container che sono stati rimossi
- **docker volume rm \$(docker volume ls -q)** rimuove tutti i volumi, anche quelli in uso (NB: nei volumi sono contenute tutti i dati usati dai container, rimuovendoli i dati andranno perduti)
- **docker rm -v \$(docker ps -q)** rimuove tutti i container attivi e i volumi a loro associati
- **docker rm -f \$(docker ps -a -q)** rimuove tutti i container, anche quelli in esecuzione
- **docker rm -f -v \$(docker ps -a -q)** rimuove tutti i container attivi e non e i volumi a loro associati
- **docker rmi \$(docker images -q)** rimuove tutte le immagini. In caso di conflitto o errori (es. container attivi) l'operazione viene interrotta
- **docker rmi -f \$(docker images -q)** rimuove tutte le immagini anche quelle associate a container attivi
- **docker exec -it {CONTAINER} bash** entra nel container con il terminale (bash) a riga di comando
- **docker run [OPTIONS] {{IMAGE}} [COMMAND] [ARGS...]** avvia un container docker a partire da un'immagine. L'esecuzione può essere personalizzata con opzioni, comandi da eseguire e relativi parametri.

Dockerfile

- Un **Dockerfile** è un domain-specific language (DSL), ovvero un insieme di istruzioni per la definizione di immagini Docker.
- Viene definito mediante un semplice file di testo che descrive le personalizzazioni che vogliamo apportare ad un template (immagine) Docker di partenza.

Comandi Dockerfile

- **FROM:** permette di specificare un'immagine di base (base image) da cui partire per derivare l'immagine personalizzata.

```
FROM <nome_immagine>  
FROM <nome_immagine>:<tag>  
FROM <nome_immagine>@<hash>
```

- **WORKDIR:** imposta la directory all'interno del container su cui avranno effetto tutte le successive istruzioni.

```
# path assoluto  
WORKDIR /path1/path2  
# path relativo  
WORKDIR path1/path2
```

- **RUN:** consente di eseguire dei comandi all'interno del container (es. installare pacchetti).

```
# shell form  
RUN <comando> <parametro1> ... <parametroN>  
# exec form  
RUN [<"comando">, "<parametro1>", ... , "<parametroN>"]
```

- **LABEL:** consente di aggiungere metadati all'immagine, come coppia chiave-valore.

```
LABEL "<chiave>"="<valore>" ...
```

Comandi Dockerfile

- **ADD e COPY:** comandi per copiare file e directory dal build context (path del Dockerfile) all'interno del filesystem dell'immagine. ADD supporta anche file remoti e archivi.

```
# shell form
ADD <src> <dest>
# exec form
ADD ["<src>", "<dest>"]

# shell form
COPY <src> <dest>
# exec form
COPY ["<src>", "<dest>"]
```

- **ENTRYPOINT:** permette di eseguire un comando all'interno del container non appena questo si è avviato.

```
# shell form
ENTRYPOINT <comando> <parametro_1> ... <parametro_n>
# exec form
ENTRYPOINT ["<comando>", "<parametro_1>", ..., "<parametro_n>"]
```

- **EXPOSE:** definisce le porte sul quale il container resterà in ascolto quando avviato.

```
EXPOSE <porta_1> [<porta_n>]
```


Comandi Dockerfile

- **CMD:** definisce gli argomenti passati al comando ENTRYPOINT. Se ENTRYPOINT non è usato (di default è /bin/sh -c), allora CMD esegue una sequenza di comandi.

```
# shell form
CMD <comando> <parametro_1> ... <parametro_n>
# exec form
CMD ["<comando>", "<parametro_1>", ..., "<parametro_n>"]
# if entrypoint does not exist
CMD ["ls"]
```

- **ENV:** offre la possibilità di impostare variabili di ambiente valide per tutto il contesto di esecuzione del Dockerfile. Per usare una variabile d'ambiente all'interno del Dockerfile, basta richiamarla antepoendo il carattere \$ alla chiave stessa (es. \$JAVA_HOME).

```
ENV <chiave>=<valore>
```

- **VOLUME:** consente di specificare un path all'interno del file system del container, che potrà essere collegato al file system dell'host in fase di esecuzione del container, tramite un'operazione di mapping.

```
VOLUME ["/www"]
```

Play With Docker

- Play with Docker (PWD) è un progetto creato da Marcos Liljedhal e Jonathan Leibiusky e sponsorizzato da Docker Inc.
- PWD è una piattaforma online che consente agli utenti di eseguire comandi Docker, testando dei container all'interno di una macchina virtuale Alpine Linux gratuita nel browser.
- Consente di creare ed eseguire container Docker e persino creare cluster mediante l'orchestratore Docker Swarm.
- PWD include anche un sito di formazione composto da un'ampia serie di laboratori Docker e quiz dal livello principiante al livello avanzato disponibili su <https://training.play-with-docker.com>.



Play With Docker

- <https://training.play-with-docker.com/beginner-linux/>

Play with Docker classroom

[About](#)

Docker for Beginners - Linux

Aug 1, 2019 • @mikegcoleman

In this lab, we will look at some basic Docker commands and a simple build-ship-run workflow. We'll start by running some simple containers, then we'll use a Dockerfile to build a custom app. Finally, we'll look at how to use bind mounts to modify a running container as you might if you were actively developing using Docker.

Difficulty: Beginner (assumes no familiarity with Docker)

Time: Approximately 30 minutes

Tasks:

- [Task 0: Prerequisites](#)
- [Task 1: Run some simple Docker containers](#)
- [Task 2: Package and run a custom app using Docker](#)
- [Task 3: Modify a Running Website](#)

Task 0: Prerequisites

You will need all of the following to complete this lab:

- A clone of the lab's GitHub repo.
- A DockerID.

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
boot etc lib lib64 media opt root sbin sys usr
root@0a8a59fa7da4:/# docker
bash: docker: command not found
root@0a8a59fa7da4:/# docker ps
bash: docker: command not found
root@0a8a59fa7da4:/# ls /
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr
root@0a8a59fa7da4:/# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   2.0  0.0   4108  3452 pts/0    Ss   15:41   0:00 bash
root        13   0.0  0.0   5892  2792 pts/0    R+   15:41   0:00 ps aux
root@0a8a59fa7da4:/# cat /etc/issue
Ubuntu 20.04.4 LTS \n \l

root@0a8a59fa7da4:/# exit
exit
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
[node1] (local) root@192.168.0.28 ~
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
8525e3ca049f   alpine    "hostname" 3 minutes ago Exited (0) 3 minutes ago
musing_wu
[node1] (local) root@192.168.0.28 ~
$
```

Docker Compose

- **Compose** è uno strumento per definire ed eseguire applicazioni Docker multi-container, definiti e configurati mediante un file YAML.
- L'uso di Compose è un processo composto da tre fasi:
 1. Definizione dell'ambiente dell'app mediante un Dockerfile, portabile e dunque eseguibile ovunque.
 2. Definizione dei servizi che compongono l'app in un file **docker-compose.yml** in modo che possano essere eseguiti insieme in un ambiente isolato.
 3. Esecuzione di Docker Compose tramite il comando "docker compose" o, in alternativa, usando il comando "docker - compose" per eseguire l'intera app multi-container.

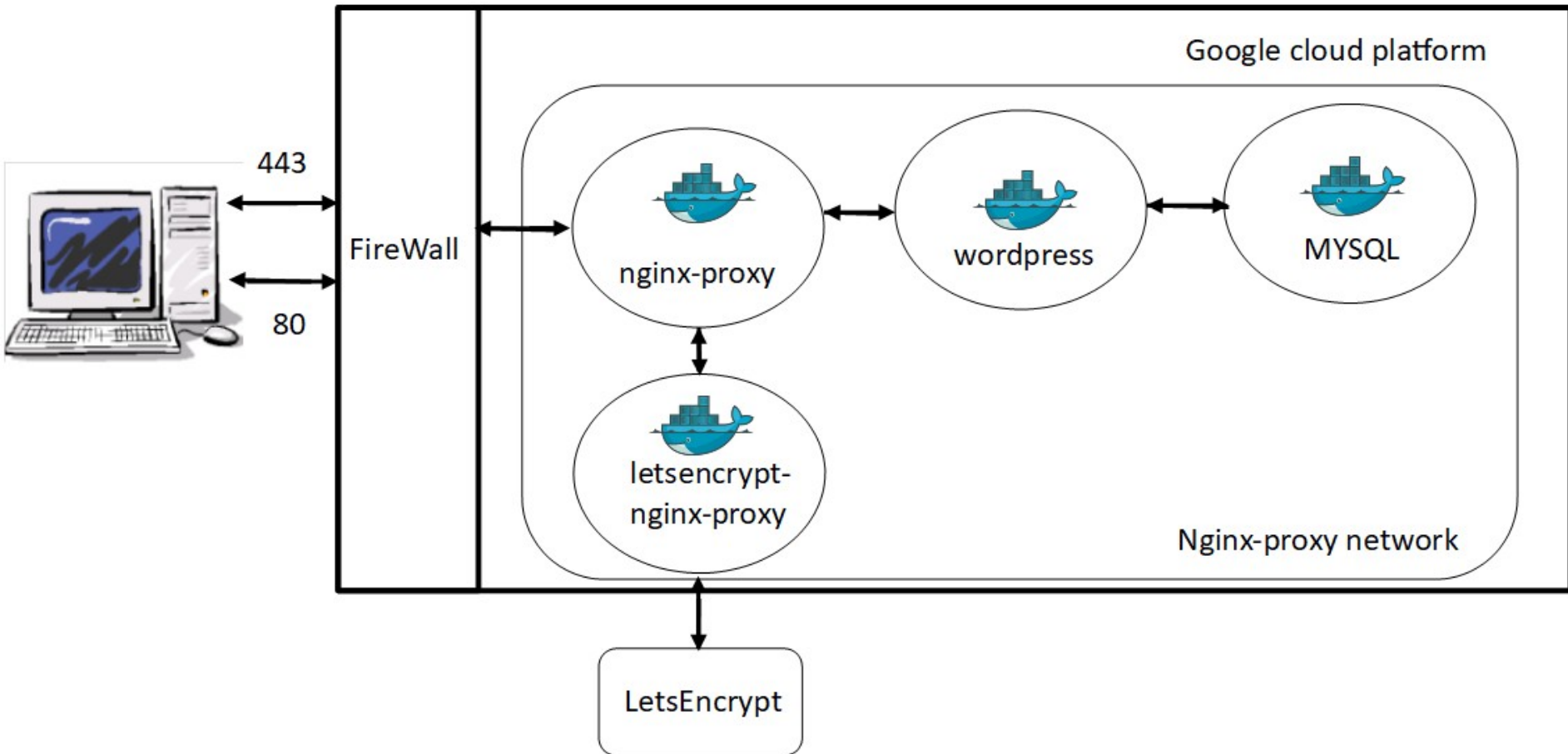
Docker Compose

- **Compose** è uno strumento per definire ed eseguire applicazioni Docker multi-container, definiti e configurati mediante un file **YAML**.
- L'uso di Compose è un processo composto da tre fasi:
 1. Definizione dell'ambiente dell'app mediante un Dockerfile, portabile e dunque eseguibile ovunque.
 2. Definizione dei servizi che compongono l'app in un file **docker-compose.yml** in modo che possano essere eseguiti insieme in un ambiente isolato.
 3. Esecuzione di Docker Compose tramite il comando "docker compose" o, in alternativa, usando il comando "docker - compose" per eseguire l'intera app multi-container.

<https://docs.docker.com/compose/compose-file/compose-file-v3/>

Docekr compose - Esempio

Using docker-compose setup Wordpress + nginx + letsencrypt + mysql



Docker compose - Esempio

```
version: "3.7"
services:
  db:
    image: mariadb:10.5.6
    ports:
      - "3306:3306"
    volumes:
      - ./mysql:/var/lib/mysql
    environment:
      - MARIADB_ROOT_PASSWORD=root
      - MYSQL_ROOT_PASSWORD=root
    restart: always
  wordpress:
    image: wordpress:php7.3-fpm-alpine
    depends_on:
      - db
    links:
      - db
    volumes:
      - /data/html:/var/www/html
    restart: always
    environment:
      - WORDPRESS_DB_HOST: db
      - MYSQL_ROOT_PASSWORD: mysql_root_pass
      - WORDPRESS_DB_NAME: db_name
      - WORDPRESS_DB_USER: user_name
      - WORDPRESS_DB_PASSWORD: user_pass
      - WORDPRESS_TABLE_PREFIX: wp_
```

Bind delle porte tra host e container
(utile per accedere al db mediante la
porta della macchina host)

Monta il percorso della macchina
host ./mysql nel percorso del contenitore
/var/lib/mysql.

Variabili di ambiente di MariaDB

Implica che i servizi vengono avviati e
arrestati nell'ordine di dipendenza.

Consente al servizio wordpress di
comunicare con il servizio mariadb.

Monta il percorso della macchina host
/data/html nel percorso del contenitore
/var/www/html.

Variabili di ambiente di Wordpress

Kubernetes

- **Kubernetes** è un orchestratore, ovvero una piattaforma di distribuzione e gestione dei container Linux.
 - La crescita di Kubernetes è stato supportato da Google per diversi anni prima di essere offerto alla comunità open.
 - Offre numerose funzionalità, tra cui scaling, distribuzione automatica e fault tolerance, ridondanza su nodi multipli.
 - Supporta diversi container runtime: Docker, OCI (es., rkt e CRI-O).

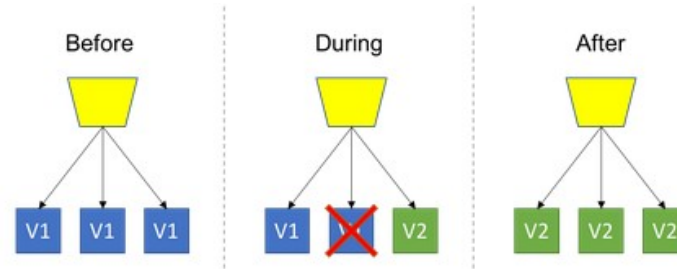
- **System Performance:** aumenta/diminuisce il numero di unità di elaborazione (pods) in base al carico della CPU o altri criteri.
- **System Monitoring:** lo stato di ogni pod viene controllato continuamente e, in caso di crash, una nuova istanza viene messa in servizio.
- **Deployment:** gestisce il deploy automatico di nuove versioni dei container, anche mediante rolling upgrade o blue/green deployment



Kubernetes - Strategie di deployment

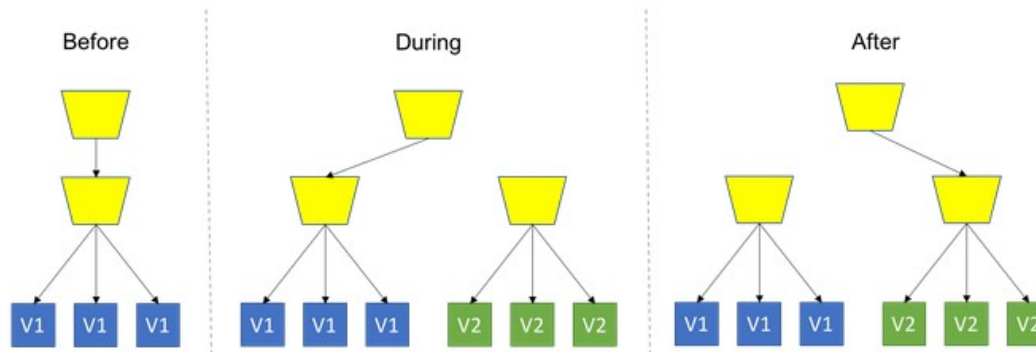
Rolling Upgrade: la nuova versione del codice è introdotta nella distribuzione esistente in maniera graduale, durante la disattivazione del vecchio codice.

- La distribuzione esistente diventa un pool eterogeneo di vecchia versione e nuova versione, con l'obiettivo finale di sostituire lentamente tutte le vecchie istanze con le nuove istanze.



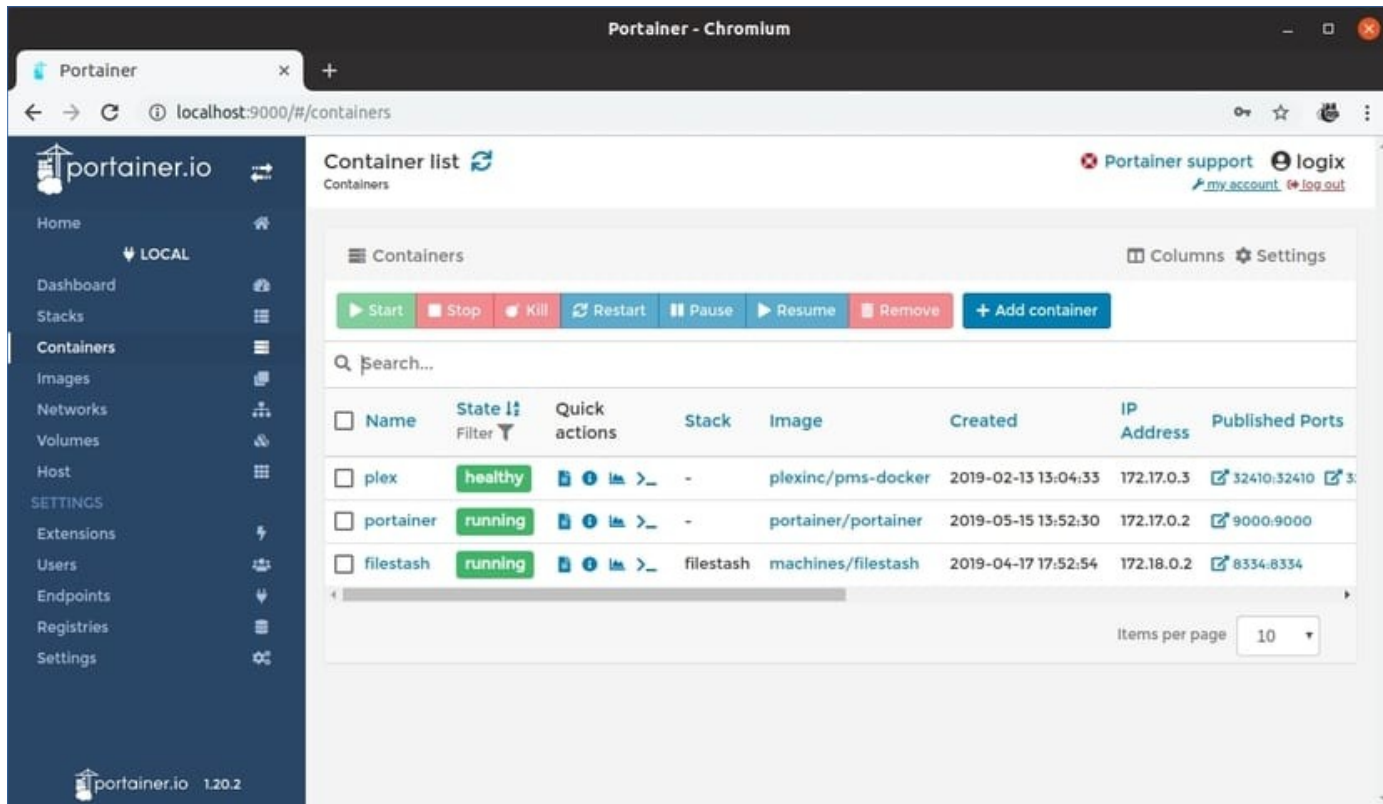
Blue/Green Upgrade: crea una nuova distribuzione separata per la nuova versione, senza influire su quella attuale. Prova la nuova versione e, una volta pronto, inizia a instradare gli utenti alla nuova versione.

- Questa è la strategia più sicura e viene utilizzata da molti per i carichi di lavoro di produzione.



Portainer

- **Portainer** è un servizio per la gestione centralizzata di container, che può essere installato come container Docker standalone, dentro Kubernetes o Docker Swarm.



Installare Portainer Server con Docker

```
# creare un volume docker per memorizzare il db e i dati di Portainer
docker volume create portainer_data

# creare un container di Portainer
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always
-v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data
portainer/portainer-ce:latest

# usare la web GUI di Portainer disponibile all'url:
https://localhost:9443
```



Esempio Custom Ubuntu Image

- Lanciamo container da un'immagine standard "ubuntu" accedendo alla sua console bash.
- Accediamo al container tramite shell e installiamo un pacchetto "figlet" per stampare delle scritte stilizzate sulla console.

```
docker container run -ti ubuntu /bin/bash
```

```
# => -t: Allocate a terminal
```

```
# => -i: interactive
```

```
apt-get update
```

```
apt-get install -y figlet
```

```
figlet "hello docker"
```

<https://training.play-with-docker.com/ops-s1-images/>

Esempio Custom Ubuntu Image

- Come previsto il container creato termina la sua esecuzione, ma possiamo vederlo tra i container terminati.

```
docker ps -a  
docker container ls -a
```

```
# Ci consente di vedere tutte le modifiche apportate rispetto all'immagine originale  
docker container diff <container ID>
```

```
# Generiamo una nuova immagine a partire dal container custom  
docker container commit <container ID>
```

```
# Vediamo ora l'immagine del container tra la lista delle immagini disponibili  
docker image ls
```

```
# Assegniamo un repository name all'immagine personalizzata  
docker image tag <IMAGE_ID> <MY_IMAGE_NAME>:<VERSION_TAG>
```

```
# Lanciamo un container con la nuova immagine  
docker container run <MY_IMAGE_NAME>:<VERSION_TAG> figlet "New message"
```

<https://training.play-with-docker.com/ops-s1-images/>

Esempio Custom Ubuntu Image

- La stessa immagine docker può anche essere creata mediante un Dockerfile che contiene:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y figlet
```

```
# Effettuiamo la build dell'immagine, con nome myubuntu2 e tag latest (implicito)
docker build -t myubuntu2 -f Dockerfile .
```

Lanciamo un comando

```
docker container run myubuntu2:latest figlet "New message"
```

```
loris@mymint ~/Dropbox/Corsi/Sistemi distribuiti/Sistemi Distribuiti and Cloud 2022 $
docker container run myubuntu2:latest figlet "New message"
```

New Mexico

Esempio Custom Ubuntu Image

- La stessa immagine docker può anche essere creata mediante un Dockerfile che contiene:

```
FROM ubuntu:latest  
RUN apt-get update && apt-get install -y figlet
```

Effettuiamo la build dell'immagine, con nome myubuntu2 e tag latest (implicito)
docker build -t myubuntu2 -f Dockerfile .

Lanciamo un comando

docker container run myubuntu2:latest figlet "New message"

```
loris@mymint ~/Dropbox/Corsi/Sistemi distribuiti/Sistemi Distribuiti and Cloud 2022 $  
docker container run myubuntu2:latest figlet "New message"
```

```
New message
```

Esempio Docker: Java RMI server

- Prepariamo un'immagine Docker in grado di eseguire un'applicazione Java.
- L'applicazione e le sue dipendenze verranno distribuite usando un file JAR.
- Usiamo un Dockerfile per definire i layer che costituiranno l'immagine del container:

```
FROM openjdk:8-jdk-alpine

MAINTAINER lbelcastro@dimes.unical.it

COPY policy.all /usr/lib/jvm/java-1.8-openjdk/jre/lib/security/java.policy

COPY sisdis-1.0-SNAPSHOT.jar /app.jar

ENV JAVA_OPTS="-Xms1G -Xmx1G"

EXPOSE 1099

ENTRYPOINT ["java", "-Djava.rmi.server.useCodebaseOnly=false", "-Djava.security.policy=/policy.all", "-jar", "/app.jar"]
```


Esempio Docker: Java RMI server

- L'applicazione Java lato server da eseguire è *ComputeEngineServer*, che richiede una policy di sicurezza che consente di operare con client esterni.
- Per semplicità usiamo il file `policy.all` con:
`grant { permission java.security.AllPermission; };`

```
FROM openjdk:8-jdk-alpine
```

```
MAINTAINER lbelcastro@dimes.unical.it
```

```
COPY policy.all /usr/lib/jvm/java-1.8-openjdk/jre/lib/security/java.policy
```

```
COPY sisdis-1.0-SNAPSHOT.jar /app.jar
```

```
ENV JAVA_OPTS="-Xms1G -Xmx1G"
```

```
EXPOSE 1099
```

```
ENTRYPOINT ["java", "-Djava.rmi.server.useCodebaseOnly=false", "-Djava.security.policy=/policy.all", "-jar", "/app.jar", "computeEngineCodebase.server.ComputeEngineServer"]
```

Esempio Docker: Java RMI server

- STEP 1 - Build dell'immagine: `docker build -t javarmi -f Dockerfile .`
- STEP 1 – Esecuzione del container:
 - `docker run -ti javarmi /bin/sh`
 - `docker run -d --name javarmi -t javarmi`

```
FROM openjdk:8-jdk-alpine
```

```
MAINTAINER lbelcastro@dimes.unical.it
```

```
COPY policy.all /usr/lib/jvm/java-1.8-openjdk/jre/lib/security/java.policy
```

```
COPY sisdis-1.0-SNAPSHOT.jar /app.jar
```

```
ENV JAVA_OPTS="-Xms1G -Xmx1G"
```

```
EXPOSE 1099
```

```
ENTRYPOINT ["java", "-Djava.rmi.server.useCodebaseOnly=false", "-Djava.security.policy=/policy.all", "-jar", "/app.jar", "computeEngineCodebase.server.ComputeEngineServer"]
```

Esempio Radius Server

- **RADIUS** (*Remote Authentication Dial-In User Service*) è un protocollo AAA (authentication, authorization, accounting) utilizzato in applicazioni di accesso alle reti o di mobilità IP.
- Esistono tanti software che implementano questo protocollo, es. Freeradius (<https://freeradius.org/>), che possono essere installati su un OS esistenti o eseguiti come container Docker.
- **freeradius/freeradius-server** è un repo su Docker Hub del FreeRADIUS Server Project.
- La struttura delle cartelle del radius server è la seguente:

```
clients.conf      <---- contiene le configurazioni del sistema (es. reti client autorizzati)
mods-config/
mods-config/files/
mods-config/files/authorize  <-- contiene credenziali utenti autorizzati al login
```

Esempio Radius Server

- Realizziamo un server Radius per autenticare del client Java usando dei container Docker.
- **STEP 1: Creazione immagine docker personalizzata**
 - Si vuole realizzare un'immagine custom, a partire da `freeradius/freeradius-server:latest`
 - Preparare i file di configurazione con le credenziali utente e le reti abilitate a connettersi al RADIUS server.
 - L'immagine dovrà caricare le impostazioni del Radius server da cartelle locali. Creare quindi un Dockerfile con le seguenti caratteristiche:

```
FROM freeradius/freeradius-server:latest
MAINTAINER lbelcastro@dimes.unical.it
COPY raddb /etc/raddb
```

Esempio Radius Server

File mods-config/files/authorize

```
bob Cleartext-Password := "test"
```

File clients.conf

autorizziamo gli altri container docker (rete 172.17.0.0/16) a connettersi al server

```
client dockernet {  
    ipaddr = 172.17.0.0  
    netmask = 16  
    secret = testing123  
}
```

autorizziamo gli indirizzi Unical a connettersi al server

```
client unical {  
    Ipaddr = 160.97.0.0  
    netmask = 16  
    secret = testing123  
}
```

Esempio Radius Server

- **STEP 2: Build immagine custom e creazione container**

build immagine custom

docker build -t myradius -f Dockerfile .

esecuzione container in modalità debug

docker run -ti --rm --name radius myradius -X

```
    type = "acct"
    ipv6addr = ::
    port = 0
  limit {
    max_connections = 16
    lifetime = 0
    idle_timeout = 30
  }
}
listen {
  type = "auth"
  ipaddr = 127.0.0.1
  port = 18120
}
Listening on auth address * port 1812 bound to server default
Listening on acct address * port 1813 bound to server default
Listening on auth address :: port 1812 bound to server default
Listening on acct address :: port 1813 bound to server default
Listening on auth address 127.0.0.1 port 18120 bound to server inner-tunnel
Listening on proxy address * port 38149
Listening on proxy address :: port 36315
Ready to process requests
```

Esempio Radius Server

- **STEP 3: Creazione client Java per connettersi al server RADIUS**
- Usiamo la libreria **tinyradius**

```
public class TestRadiusLogin {
    public static void main(String[] args) {
        String host = "172.17.0.3";
        String sharedSecret = "testing123";
        String username = "bob";
        String password = "test";

        RadiusClient rc = new RadiusClient(host, sharedSecret);
        try {
            if (rc.authenticate(username, password)) {
                System.out.println("Utente autenticato con successo!");
            } else {
                System.out.println("Credenziali non valide");
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (RadiusException e) {
            e.printStackTrace();
        }
    }
}
```