Introduzione a Docker

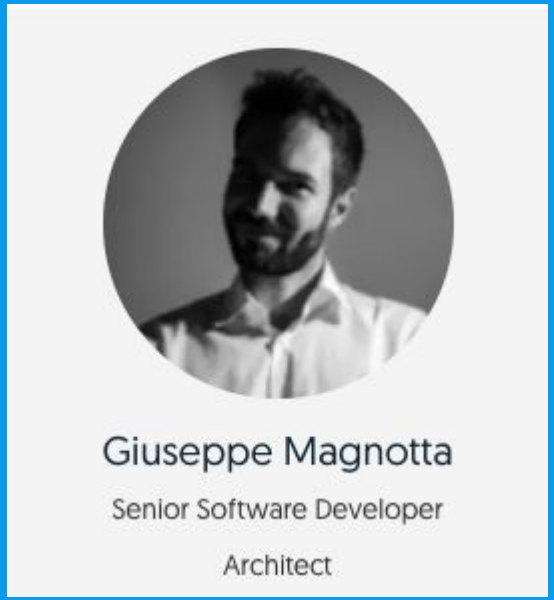
Giuseppe Magnotta <giuseppe.magnotta@gmail.com>

Roma, 06/02/2020

Giuseppe Magnotta

Senior Software Developer - Architect

Agile Senior Software Developer and Software Engineer, technology enthusiast with experience in system integration, Unix environments, mobile, embedded, web platforms, blockchain, Docker and Kubernetes. Always interested in discovery new technologies and new challenges. Open Source contributor.



Organizzazione del Talk

- Cosa sono i sistemi operativi (esempio del kernel Linux) e perché sono stati inventati
- Cosa sono le macchine virtuali e perché sono state inventate
- Quali sono i limiti delle macchine virtuali
- Cosa sono i containers e perché sono stati inventati
- Introduzione a Docker

Cosa sono i sistemi operativi e perché sono stati inventati

Il software applicativo

- Il software applicativo è quel software progettato per risolvere un problema specifico.
- Web browser, client di posta elettronica, videogames e app per smartphone sono un classico esempio di software applicativo,
- Ogni programmatore svolge il suo lavoro utilizzando librerie software in modo da semplificare l'implementazione di task complessi o ripetitivi. Si pensi ad una chiamata HTTP: in quanti aprono una socket TCP, scrivono il messaggio, leggono la risposta, la processano e chiudono la socket? Meglio usare un client HTTP!
- Sfortunatamente, però, il software viene eseguito dall'Hardware!
- Cosa succede se il programmatore dovesse interagire ogni volta con componenti hardware di basso livello?
- Si immagini se il nostro software fosse l'unico programma in esecuzione sull'hardware. Cosa si dovrebbe fare per leggere un byte dal disco?

Esempio lettura I/O

```
425 do_chs:
426         /*
427          * BIOS call "INT 0x13 Function 0x2" to read sectors from disk into
428          * memory
429          *       Call with       %ah = 0x2
430          *                       %al = number of sectors
431          *                       %ch = cylinder & 0xFF
432          *                       %cl = sector (0-63) | rest of cylinder bits
433          *                       %dh = head
434          *                       %dl = drive (0x80 for hard disk)
435          *                       %es:%bx = segment:offset of buffer
436          *       Return:
437          *                       carry set: failure
438          *                           %ah = err code
439          *                           %al = number of sectors transferred
440          *                       carry clear: success
441          *                           %al = 0x0 OR number of sectors transferred
442          *                                 (depends on BIOS!)
443          *                                 (according to Ralph Brown Int List)
444          */
445         movb    $CHAR_CHS_READ, %al
446         call    Lchr
447
448         /* Load values from active partition table entry */
449         movb    1(%si), %dh             /* head */
450         movw    2(%si), %cx             /* sect, cyl */
451         movw    $0x201, %ax             /* function and number of blocks */
452         xorw    %bx, %bx                /* put it at %es:0 */
453         int     $0x13
454         jnc     booting_os
455
```

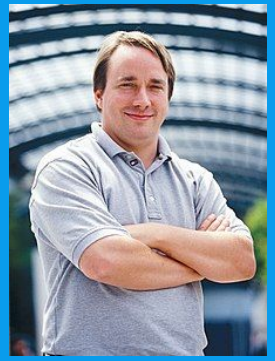
Il software applicativo

- Scrivere applicazioni diventerebbe un’attività ancora più complessa perché bisognerebbe gestire tutto l’hardware differente dei vari fornitori,
- Ed ancora più difficile se si volesse eseguire più software in maniera concorrente sullo stesso hardware!
- Gli sviluppatori dovrebbero essere protetti dalla complessità dell’hardware: dovrebbero utilizzare una libreria che mostri una visione semplificata della macchina in modo da farli concentrare soltanto sulla logica di business,
- Sarebbe anche meglio se questa libreria sia uguale per differenti architetture hardware in modo che i programmi possano funzionare su ambienti eterogenei,
- Questa “libreria” è il sistema operativo.

Il sistema operativo

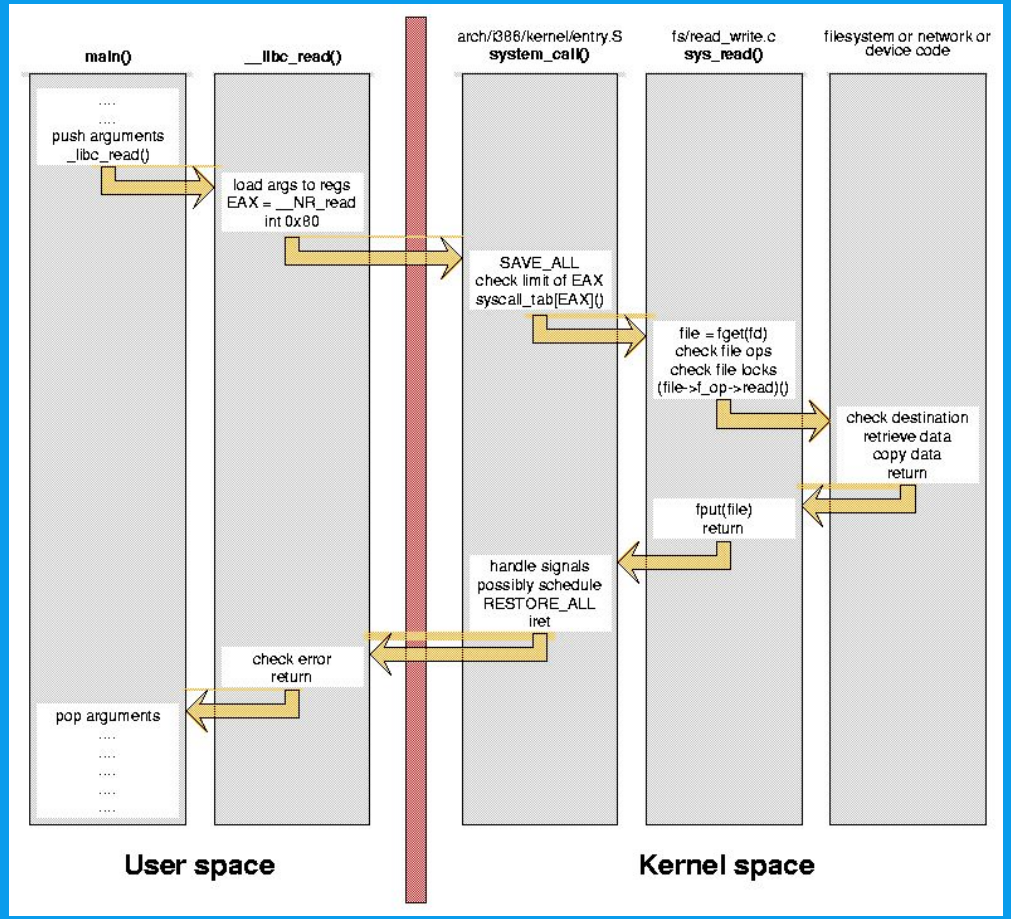
- Il sistema operativo è quell'insieme di software che gestisce le risorse hardware e fornisce una vista unificata della macchina al software applicativo,
- Ci sono tanti sistemi operativi disponibili, ognuno con le proprie caratteristiche e funzionalità. Ad esempio:
  - single e multi task
  - single e multi utente
  - real time
  - embedded
  - distribuiti
- Quelli maggiormente conosciuti sono Unix *BSD, Ubuntu (Linux), Red Hat Enterprise (Linux), Microsoft Windows, MacOS, iOS ed Android.

Il kernel Linux



- Linux è un kernel (non un sistema operativo completo!) creato da Linus Torvalds nel 1991.Il kernel è la parte più importante di un sistema operativo: è responsabile di gestire l’hardware e presentare al programmatore una vista semplificata del sistema.
- Di solito il kernel astrae l’hardware con i concetti di Processo, servizi di rete, gestore della memoria, filesystem e device driver.
- Sulle architetture che offrono la protezione hardware (es. modalità protetta x86) il kernel è l’unico software autorizzato ad accedere all’hardware.
- I programmatori invocano le primitive del kernel tramite syscall

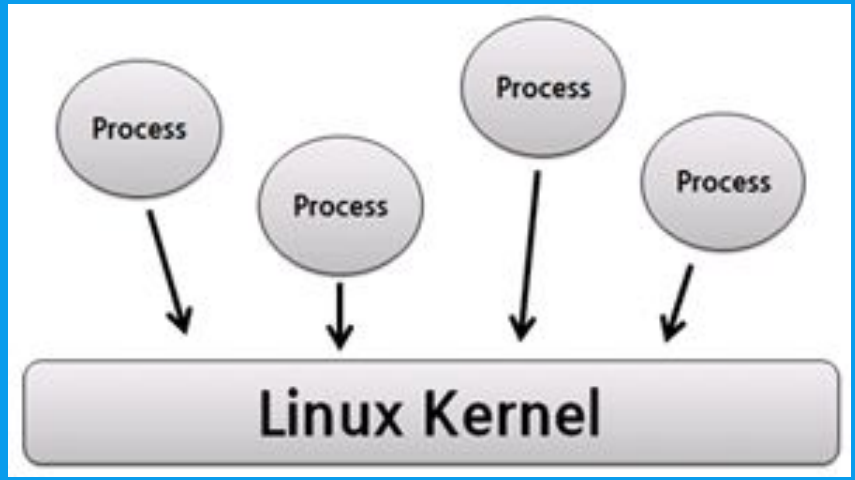
Il kernel Linux



Il kernel Linux

In Linux, quando un programma viene caricato in memoria, diventa un **Processo**:

- è un’astrazione fornita dal kernel per gestire più applicazioni in esecuzione in concorrenza,
- rappresenta lo stato di un programma in esecuzione in un particolare istante di tempo,
- non può modificare la memoria assegnata ad altri processi ma soltanto la sua.



I limiti del Kernel Linux

- E’ possibile eseguire un programma compilato per Linux su Windows senza adottare soluzioni particolari? Ed il contrario?

  No, perchè ci sono diverse incompatibilità tecniche (differenti syscall, loaders, etc)
- E se volessimo eseguire due sistemi operativi nativamente ed in contemporanea su di una macchina?

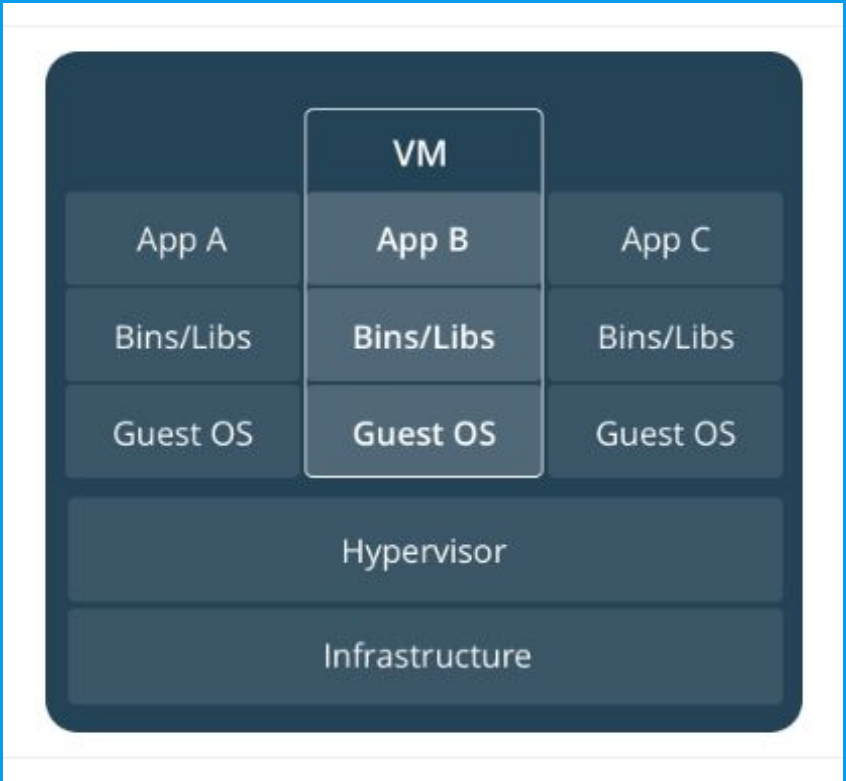
  Non sempre l’hardware offre questa funzionalità.

Cosa sono le macchine virtuali e perché sono state inventate

Le macchine virtuali

- Le macchine virtuali sono state inventate per risolvere il problema di avere Sistemi Operativi multipli in esecuzione su di uno stesso hardware: sono applicazioni software the replicano il comportamento di una macchina completa (CPU, memoria, scheda video, etc),
- Dato che le macchine virtuali sono software, esse sono eseguite come processi su di un kernel specifico. L'hardware vero che esegue una virtual machine è detto Host,
- Una macchina virtuale (conosciuta anche come Guest) rappresenta un'istanza di una macchina completa simulata via software. Essa esegue un kernel di un sistema operativo che non sa di essere in esecuzione su di un hardware simulato,
- La magia delle macchine virtuali avviene traducendo tutti gli interrupt hardware della macchina simulata come syscall sul kernel della macchina host (questa è una semplificazione, c'è dell'altro!).

Le macchine virtuali



La virtualizzazione è bella!

- **Vantaggi**:
  - permette di comprare una singola macchina molto potente e creare su di essa una moltitudine di macchine virtuali ciascuna separata dalle altre (differenti processi sullo stesso host),
  - Il business ama le macchine virtuali perché permette di risparmiare soldi ed aumentare l'utilizzo dell'hardware.
- **Svantaggi**:
  - Lentezza! La traduzione degli interrupt consuma tante risorse,
  - Tipicamente le virtual machine sono oggetti stateful: dato che è richiesto tempo per creare una virtual machine, installarci un sistema operativo ed i programmi applicativi necessari viene eseguita per più tempo possibile,
  - Avviare e fermare virtual machine dinamicamente non è una buona pratica (diminuisce il throughtput dell'hardware).

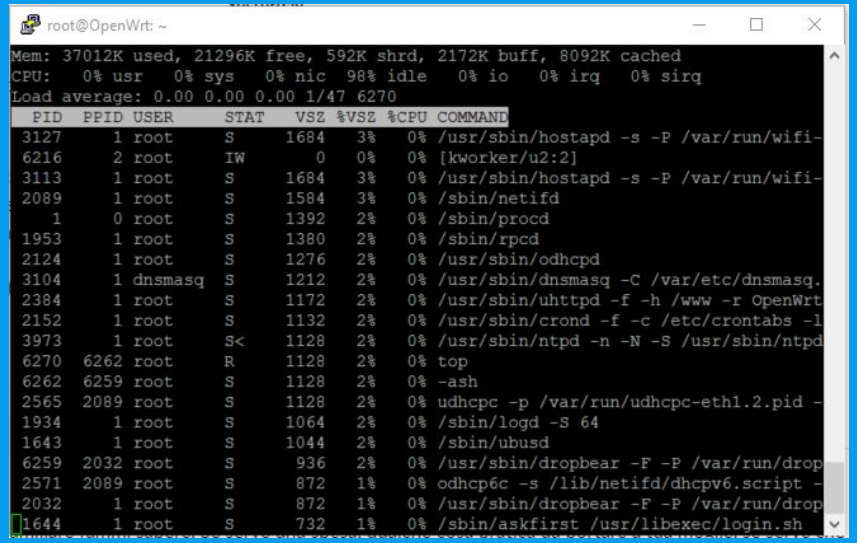
Cosa sono i containers e perché sono stati inventati

I container Linux

In Linux ogni processo può vedere tutti gli altri processi in esecuzione nel sistema e può venire a conoscenza delle risorse che essi stanno utilizzando.

Questo vuol dire che nel caso in cui un'applicazione contenga delle vulnerabilità essa potrebbe esporre ad un attaccante tutte le informazioni alle quali essa può accedere.

Si pensi alla lettura di /etc/passwd da parte di httpd.



I container Linux

Quali soluzioni sono offerte dal Kernel Linux per evitare questi problemi di sicurezza?

Nel corso degli anni sono state implementate alcune primitive:

- **Chroot** - permette di confinare un processo in una porzione specifica del filesystem
- **Namespace** - permette di raggruppare i Processi in gruppi in modo che le risorse di ogni gruppo come alberatura di processi, risorse di rete, ID utente e filesystem montati non siano visibili dagli altri gruppi
- **Cgroup** - permette di gestire limiti e priorità nell'uso delle risorse: CPU, accesso alla memoria, accesso al disco, utilizzo di rete, etc.

Un programmatore che volesse rendere più sicuri i suoi programmi dovrebbe utilizzare queste primitive offerte dal Kernel Linux eseguendo una specifica syscall durante la fase di inizializzazione (e non è semplicissimo da fare!).

Questo è il motivo per cui sono nati i container Linux!

I container Linux

Cos’è un Container?

- è un insieme di uno o più processi che sono isolati dal resto del sistema,
- possiede tutti i files necessari ad eseguire questi processi: questo permette al container di essere portabile in ogni ambiente,
- contiene un software con tutte le sue dipendenze: librerie statiche o dinamiche, file di configurazione, etc...
- è Isolato, ossia, possiede il suo dispositivo di rete, il suo indirizzo Ip, il suo filesystem root, etc...

I container Linux

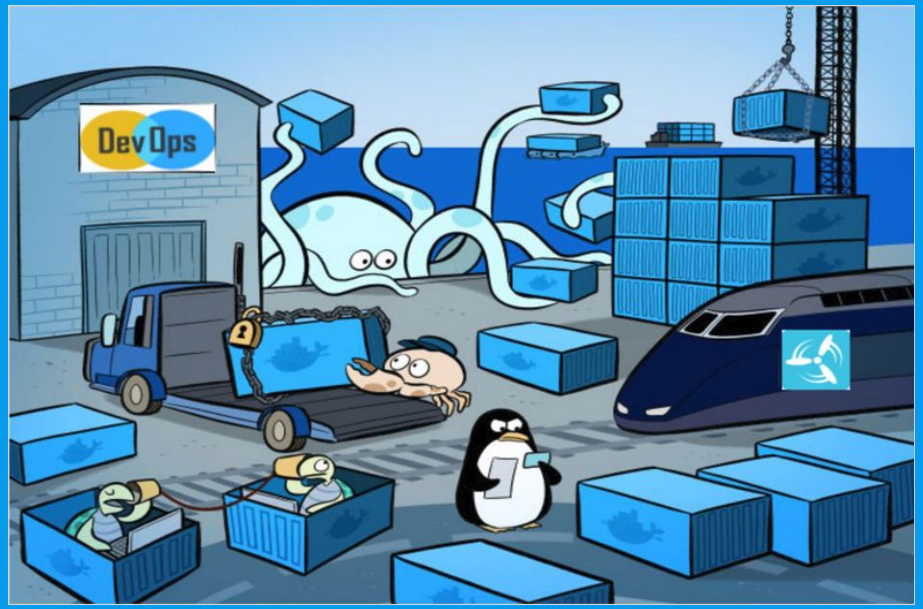
- **Vantaggi**:
  - forniscono tutti i benefici delle macchine virtuali ma sono più leggeri: essi condividono lo stesso Kernel della macchina host (usano meno memoria) e non è eseguita alcuna virtualizzazione (più throughput),
  - Le applicazioni non sanno di essere eseguite all'interno di un Container: i programmatori non devono eseguire delle chiamate speciali alle syscall,
  - Container platform sono ambienti di esecuzione che automaticamente gestiscono i container per conto dell'utente.
- **Svantaggi**:
  - I Container Linux richiedono che tutte le applicazioni siano scritte per il Kernel Linux (non è possibile mischiare applicazioni Windows e Linux).

I Container platform per Linux più conosciuti sono LXD, RKT e Docker.

Introduzione a Docker

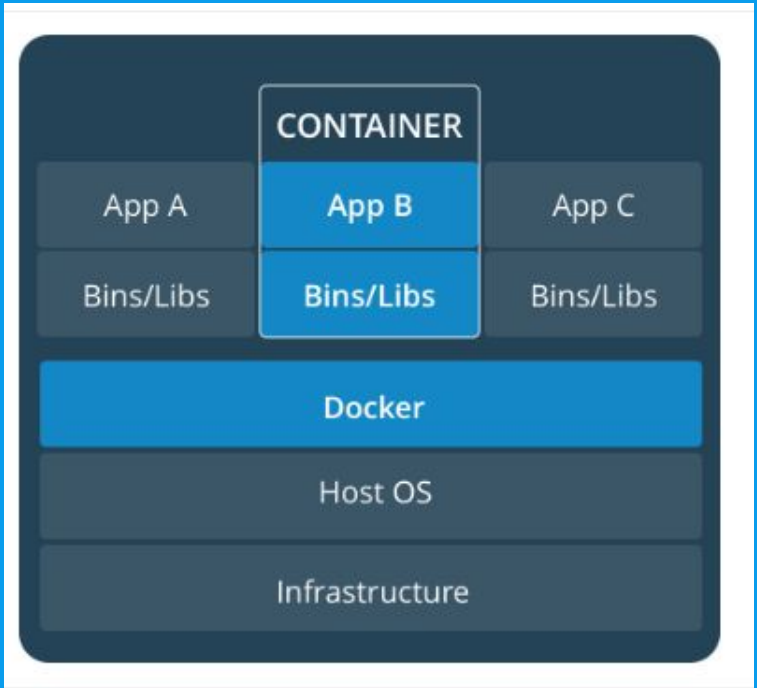
Cos’è Docker

Docker è una piattaforma per programmatori e sistemisti per sviluppare, schierare (deploy) ed eseguire applicazioni tramite container.



Cos’è Docker

Architettura di alto livello



Cos’è Docker

I componenti base dell’architettura

- **Immagini Docker**: sono la base dei container. Un’immagine è semplicemente un file di testo che contiene un insieme di filesystem impilati uno sull’altro. Un’immagine non ha stato ed è immutabile,
- **Docker container**: è una istanza in esecuzione di un’immagine Docker. Consiste di: un’immagine Docker, un ambiente di esecuzione ed un insieme standard di istruzioni,
- **Docker Repository**: è un insieme di immagini Docker. Può essere condiviso su di un server Registry. Tutte le immagini presenti nel repository possono essere identificate con delle label,
- **Docker Registry**: è un servizio hosted che contiene repository di immagini Docker. Il Registry di default è Docker Hub (è un servizio offerto da Docker per condividere immagini Docker),
- **Docker Trusted Registry**: è la soluzione di grado enterprise per condividere immagini Docker all’interno di una azienda.

Docker

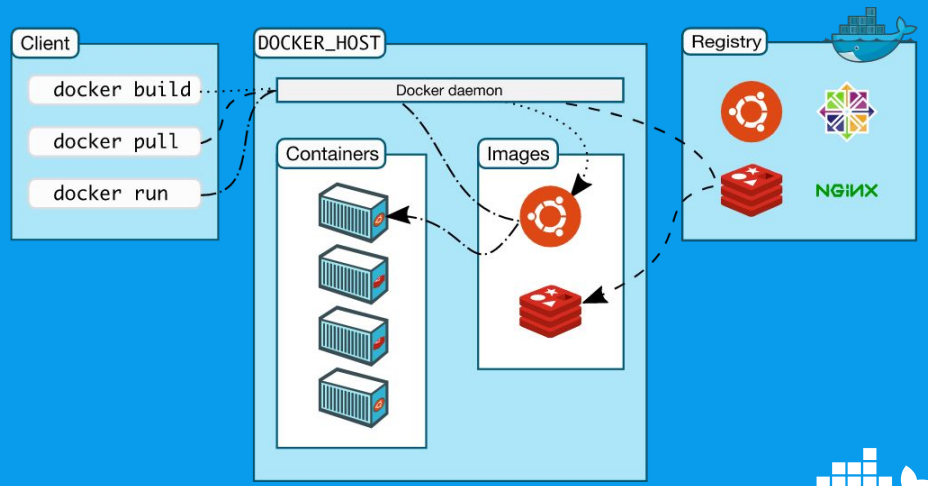
Docker usa una architettura client-server ed è composto da due componenti:

- Docker client
- Docker daemon (può essere installato su di un server remoto, ma anche sul client stesso)

Il Docker client parla con il Docker daemon, che esegue tutto il vero lavoro: costruisce, esegue e distribuisce i container Docker.

Questa immagine mostra uno scenario tipico in cui l'utente interagisce con il client Docker per:

- costruire una immagine Docker da un file,
- spingere l'immagine su di un Docker Registry,
- eseguire l'immagine come un container.



Docker

Il Docker daemon:

- è un servizio che è eseguito su di una macchina Linux ed è solitamente avviato durante la fase di avvio del sistema,
- è identificato dal binario dockerd.

Può essere configurato in due modi:

- usando un file di configurazione json (/etc/docker/daemon.json),
- passando parametri al binario dockerd.

Il Docker daemon persiste tutti i dati in una singola directory dell'host. Di default essa è /var/lib/docker

Docker

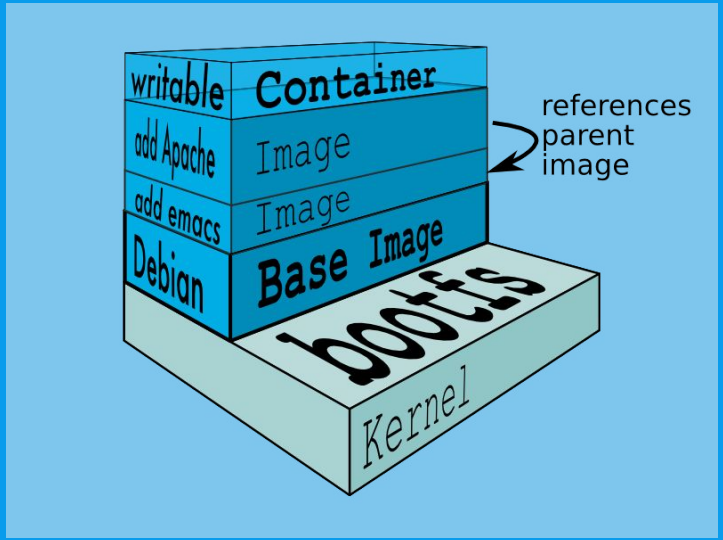
Una immagine Docker è un insieme di livelli a sola lettura. Ogni livello contiene codice (applicazioni) e istruzioni che permettono al Docker daemon di istanziare un Docker Container. Un livello può riutilizzare i file presenti nei livelli padre.

Esempio: un'immagine Docker può contenere il web server Apache.

Questa immagine potrebbe essere composta da diversi livelli:

- un livello che contiene i files base della distribuzione ubuntu (/etc, /bin, /sbin, …),
- un altro livello che estende il precedente aggiungendo tutti i file del web server Apache (/home/tomcat/catalina, …).

Una Immagine diventa un container nel momento in cui viene istanziata.



Docker Registry

Docker Registry è il componente dell'architettura Docker che ha lo scopo di memorizzare le immagini Docker.

Docker Hub è un registry pubblico che tutti possono accedere ed utilizzare.

Docker è configurato per utilizzare le immagini da Docker Hub di default.

E' possibile eseguire i propri Docker Registry privati: essi non sono altro che applicazioni REST che permettono di caricare e scaricare Immagini Docker tramite il protocollo HTTP.

Docker

Per design i container Docker sono **effimeri**!

Effimero vuol dire che quando un container è cancellato, tutti i dati scritti nel container che non sono stati persistiti in uno storage durevole scompaiono con il container.
Il container sparisce dal sistema.
Per questo motivo non bisogna considerare un Container Docker come una macchina virtuale: i Container Docker possono essere distrutti e ricreati.
Questo design permette di creare dalla stessa Immagine Docker molti container che saranno tutti equivalenti: si avrà la garanzia che la stessa immagine che è stata testata da Q.A. raggiungerà l'ambiente di produzione con esattamente lo stesso comportamento!

Kubernetes estremizza questo concetto!

Docker

Docker ci obbliga a cambiare mentalità nell'uso e nella gestione dei Container:

- i Container dovrebbero essere in grado di essere avviati nel minor tempo possibile,
- i Container dovrebbero essere pronti a morire in qualsiasi momento,
- i Container non dovrebbero essere riutilizzati: meglio creare un nuovo container!
- i file all'interno dei container non dovrebbero essere modificati.

Di solito un container Docker è identificato con un solo processo.

Quando esso termina il container è distrutto.

Una nuova istanza di una immagine Docker sarà eseguita in un nuovo Container Docker

I Container Docker sono **immutabili**.

Docker

I Container Docker sono **immutabili**

Ciò significa che:

- nel momento in cui un container è avviato, il suo filesystem è identico a quello della sua immagine
- qualsiasi scrittura nel filesystem del container è persa quando il container cessa di esistere
- le applicazioni non possono scrivere su filesystem il loro stato in modo da effettuare il recovery con un riavvio perchè quando l'applicazione muore, il container è distrutto
- tutte le informazioni importanti di un'applicazione devono essere salvate in uno storage persistente

Docker

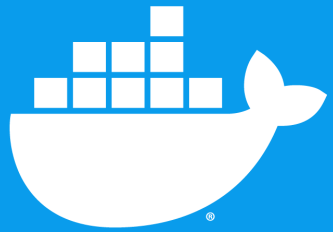
Come bisogna comportarsi per salvare i dati?

Un container docker può montare a runtime una o più directory dell'host al suo interno: ogni file scritto in quella directory sarà, in realtà, salvato nel filesystem dell'host!

E' possibile persistere i dati in una base di dati esterna.

Docker

Q&A



THANK YOU :)