Approfondimento hypervisor Luciano Alessio

**La virtualizzazione di container**

Dalle VM ai container

La virtualizzazione

La virtualizzazione è una tecnica attraverso la quale è possibile nascondere e astrarre le caratteristiche fisiche di una macchina mediante una sua rappresentazione logica detta Virtual Machine (VM) o *guest*, la quale usufruisce delle risorse dell’host fisico per mezzo di un software intermedio: l’hypervisor o Virtual Machine Monitor (VMM).

Questa tecnica ha rivoluzionato i modi di gestione delle risorse soprattutto in aziende e datacenter che ne traggono diversi benefici, tra cui la riduzione di macchine fisiche e di conseguenza di spazio, costi energetici e di costi di mantenimento, migliori sistemi di sicurezza e protezione dei dati, meccanismi per la *disaster recovery* e l’aumento della disponibilità e della potenza di calcolo dell’intero sistema di cluster tramite apposite procedure di recupero e *load balancing*.

Svantaggi delle VM

Le VM però non sono assolutamente impeccabili, in quanto richiedono per funzionare un intero sistema operativo, consumano elevate risorse hardware complicandone la gestione, il loro alto grado di isolamento rende difficile la condivisione di risorse e dispongono di limitata flessibilità e portabilità.

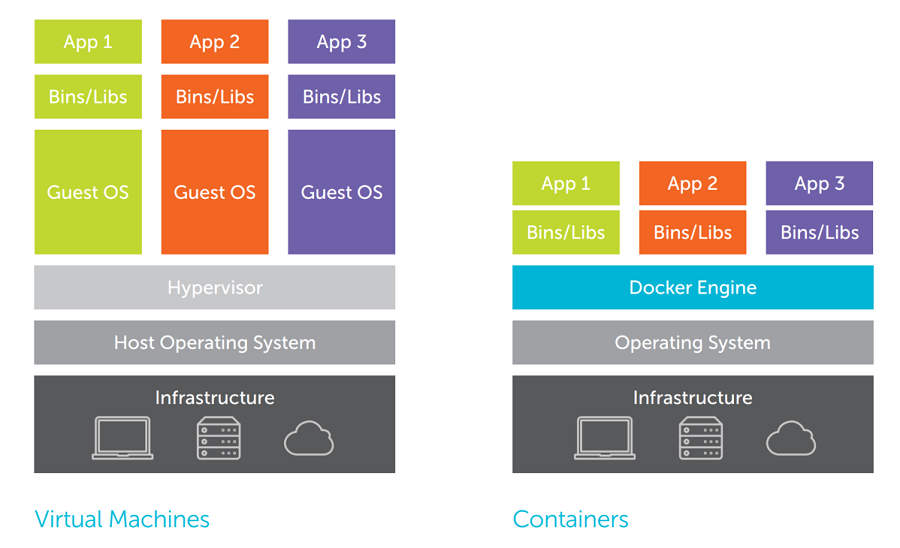
Una tecnologia le cui caratteristiche ponderano e risolvono le suddette è quella dei container.

Introduzione ai Container

Esistono diversi tipi di virtualizzazione: quella Hardware-level, che prevede l’impiego dell’hypervisor per virtualizzare un indispensabile hardware fisico sottostante creando delle macchine virtuali *guest* basate su di esso, e quella Software-level. Quest’ultima in particolare si basa sul concetto di “container”, istanze virtuali dell’host non dipendenti completamente dal suo sistema operativo.

Cos’è un container?

Un container quindi è un ambiente virtuale isolato che comprende tutte le risorse essenziali per l’esecuzione di un’applicazione. Questo è differente dalle VM perché non possiede un proprio sistema operativo, ma usufruisce del kernel del sistema host per funzionare, includendo file binari, runtimes, tool di sistema, librerie e tutto ciò che serve per eseguire ciò che contiene. Come per le Virtual Machine, ciò che avviene all’interno dei container non influisce in alcun modo sul sistema host e viceversa.



VM

Container

Perché preferirli alle VM?

**Leggerezza ed efficienza**

I container non hanno una propria immagine di sistema operativo ma usano quella dell’host. Questo li rende più leggeri delle VM, consentendo avvii più veloci e esecuzioni di container multipli e in misura maggiore (sempre rispetto alle VM) su un singolo host grazie alla maggiore efficienza nell’uso di risorse e al minor overhead.

**Isolamento leggero**

I container offrono un isolamento leggero, che si traduce semplicemente nel condividere lo stesso spazio di indirizzamento virtuale. Certo appare meno sicuro delle VM, ancora più fortemente isolate, ma nonostante ciò i container, oltre ad avere un proprio spazio dei processi privato e la propria memoria, permettono una condivisione delle risorse estremamente più facile rispetto alle VM.

**Portabilità**

I container sono progettati per essere eseguiti in qualsiasi ambiente. Le applicazioni e le loro dipendenze possono essere “*containerizzate*”, garantendo in primis un comportamento coerente su qualsiasi ambiente di sviluppo, test o produzione che supporti il runtime dei container; e inoltre risparmiando tempo e fatica agli sviluppatori, che non devono più prevedere di scrivere del codice per adattare l’applicazione ad un contesto diverso da quello per cui è stata progettata.

**Sviluppo agevolato**

Le applicazioni sviluppate coi container si staccano dal concetto di applicazione monolitica, ovvero un’applicazione distribuita come un unico blocco coeso e interconnesso dove le sue funzionalità sono gestite da un singolo processo, per implementare un’architettura a microservizi (generalmente). I microservizi sono funzionalità dell’applicazione sviluppate ognuna indipendentemente l’una dall’altra (ogni microservizio corrisponde a 1 container) al fine di distribuirle, aggiornarle o ritirarle sempre in maniera indipendente, senza che ciò influisca sul resto dell’applicazione.

Aspetti importanti e ambiti di applicazione

Vari aspetti relativi ai container

L'adozione della tecnologia basata su container ha trasformato radicalmente il panorama della gestione delle risorse informatiche, portando con sé una serie di aspetti rivoluzionari, che gli hanno permesso di superare sotto quasi ogni punto di vista la tecnologia delle VM. Ore verranno presentati nel dettaglio degli aspetti fondamentali della virtualizzazione di container, esponendone le caratteristiche.

Sicurezza nei container

Quando si parla di sicurezza, anche i container non sono esenti dall’argomento in quanto, per un’applicazione containerizzata, è fondamentale garantire la sicurezza su vari livelli (dati, accessi e network) adottando appositi meccanismi di protezione.

L’ambito della sicurezza comprende uno spettro che spazia dal semplice workload containerizzato, all’host che permette l’esecuzione di tale applicazione o microservizio; e in questo, si trovano diversi problemi di sicurezza:

***Infected images***

Le immagini dei container (o *build*) possono essere infettate con **malware** o presentare **vulnerabilità**. È quindi fondamentale mantenerle sempre aggiornate all’ultima versione disponibile; inoltre, è bene effettuare dei controlli periodici sulla sicurezza di queste attraverso opportune scansioni.

***Container breakout***

Le applicazioni containerizzate potrebbero presentare al loro interno dei **bug** e gli utenti, sfruttandolo, potrebbero scalare la piramide di privilegi sino al livello di ***root*** e ottenere accesso all'host. Malgrado la remota possibilità di questo evento, è necessario eseguire controlli periodici per validare la robustezza e la sicurezza dei container e assicurarsi che le applicazioni e le dipendenze non abbiano problemi di vulnerabilità.

***Compromised secrets***

Durante l’esecuzione le applicazioni nei container accedono periodicamente ad informazioni sensibili, chiavi API e credenziali, detti *secrets*. La mancata implementazione di un controllo degli accessi potrebbe portare a effetti gravi e indesiderati; e ciò comprometterebbe l’intera sicurezza dei servizi in esecuzione.

***Kernel exploit***

Nel caso in cui sia l’host dei container a subire un attacco che sfrutti un exploit del sistema operativo, tutti i container in esecuzione su di esso sono a rischio in quanto usufruiscono del kernel dell’host direttamente. Per questo motivo, per garantire la sicurezza degli ambienti containerizzati, è fondamentale il **patching**, l’applicazione di correzioni e aggiornamenti (patch), e l’**hardening**, l’miglioramento della sicurezza con controllo degli accessi, disabilitazione di servizi non necessari e utilizzo di firewall, del sistema operativo host.

Networking dei container

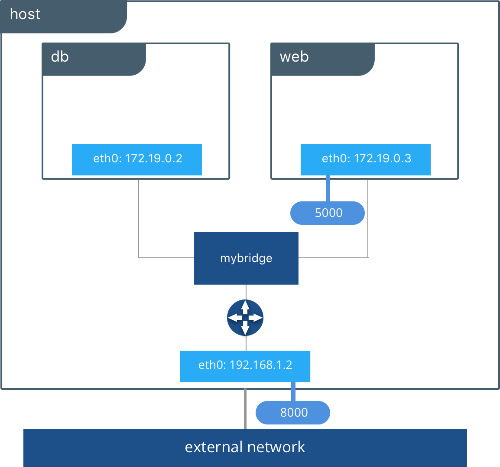
Similmente alle VM, i container sono anch’essi eseguiti in ambienti isolati, ma in alcuni casi potrebbe essere necessario consentirgli di comunicare tra loro o con risorse esterne. Vi sono varie modalità di networking di container:

***None network***

Il container riceve uno stack di rete privo di connessione esterna e con abilitata solo l’interfaccia di *loopback*. Questa modalità è utile per il *testing* e viene implementata nei container che non richiedono comunicazioni esterne.

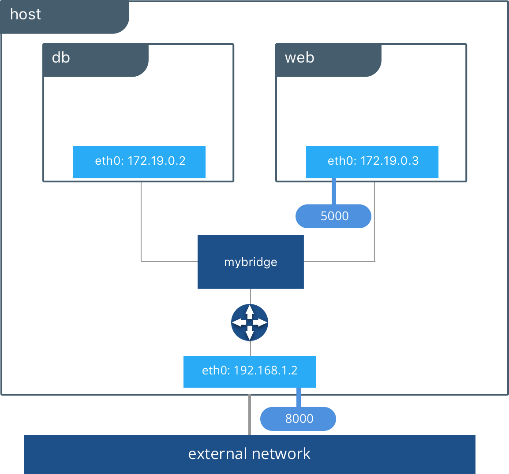
***Bridge network*** (default per Docker)

I container sono collegati a un bridge su una rete host interna e autorizzati a comunicare solo con altri container connessi al bridge sullo stesso host; ciò e possibile con l’implementazione del NAT (*Network Address Translation*), l’unico problema è causato dall’overhead e dalla bassa performance di rete del NAT. Essendo una rete privata dell’host, non è possibile accedere ai container dall'esterno.



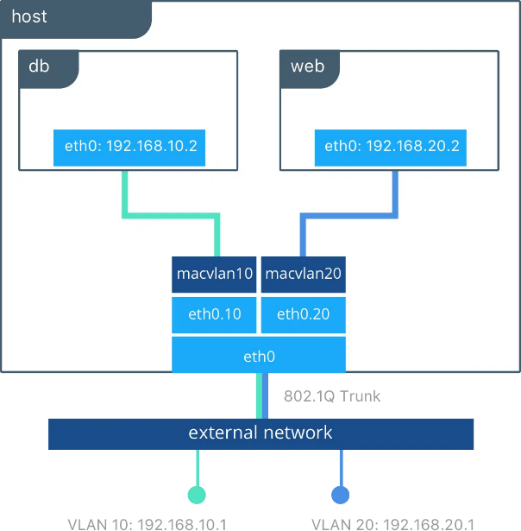
***Host network***

Questa configurazione consente ai container di condividere lo stesso indirizzo IP e il numero di porta dell’host, concedendo ad essi l'accesso a tutte le interfacce di rete dell'host. Sebbene sia la meno complessa tra le configurazioni di rete esterna, essa non offre alcun grado di isolamento di rete ed è spesso soggetta a conflitti tra le porte. In parole povere, è come se le applicazioni stiano venendo eseguite direttamente sull’host.



***MACVLAN network***

Adottando una questa tecnica di networking, è possibile assegnare a ogni container un indirizzo MAC univoco (ma anche un indirizzo IP), interfacciandolo sulla rete fisica e rendendolo visibile come se fosse una macchina a sé stante. Con questa, è possibile associare (senza usare bridge o NAT) un’interfaccia virtuale di un container ad una fisica dell’host; inoltre è possibile assegnarla ad una certa VLAN in modo da consentire un maggiore controllo sulla segmentazione dei pacchetti e sul traffico di rete.



Persistenza dei dati

Come tutte le applicazioni, anche quelle containerizzate necessitano di salvare dati per il loro funzionamento; però i container, al contrario delle VM, sono effimeri: vengono creati, modificati, aggiornati, cancellati… Un aspetto cruciale nell’ambito dei container è la persistenza dei dati che salvano, ovvero la capacità di conservare le informazioni al di là del ciclo di vita effimero dei container stessi, siccome quando un container viene eliminato e poi lo si ricrea, i dati e le modifiche fatte precedentemente in esso sono anch’esse cancellate.

Al fine di garantire la persistenza dei dati nei container, esistono 2 metodi principali:

***Bind mount***

Come per le VM, i bind mount permettono di montare una directory della macchina host all’interno del sistema virtualizzato, in questo caso il container. Sono molto comodi per quanto riguarda lo scambio di dati tra l’host e il container, però il fatto che questi si basino sul file system dell’host espone a un rischio: nel caso alcuni processi del container cancellino indesideratamente dei contenuti nel bind mount, allora quei contenuti saranno cancellati anche dal file system dell’host.

***Volumes***

I volumi invece sono dei componenti che consentono la persistenza dei dati anche dopo la rimozione o aggiornamento di un container in modo che i dati essenziali dell’applicazione non vadano persi durante una qualsiasi operazione. Al contrario dei bind mount, i volumi non sono associati al file system dell’host, quindi presentano diversi vantaggi: è possibile eseguirne facilmente il backup, sono condivisibili tra più container e, più importante, sono portabili. Grazie alla portabilità i volumi sono spostabili tra diversi host e container senza troppe difficoltà.

Nell’ambiente Docker esistono 2 tipi di volumi:

* Volumi anonimi, dei volumi senza un nome definito dall’utente ma che sono creati automaticamente quando si crea un container e gli viene subito assegnato un ID univoco. Sono generalmente difficili da gestire ed archiviare in quanto non presentano un identificatore facilmente leggibile dall’uomo. Usati per l’archiviazione temporanea.
* Volumi denominati, che hanno un nome definito dall’utente che li rende facili da identificare, gestire e condividere tra più container. Sono salvati in una posizione specifica del file system dell’host e sono amministrati dal software di containerizzazione.

Usi avanzati dei container

Deploy app distribuite + CI/CD + DevOps + testing + scalabilità orizzontale + automazione (deploy, rollback, scalabilità automatica e integrazione CI/CD…)

Monitoraggio e gestione di container ???

Monitoro prestazioni + metrica + logging

Architettura a microservizi (o meglio elencare tutti i tipi di architetture?)

?

Tecnologie principali

La storia dei container / Software di containerizzazione (partendo dai primi fino a Docker, brevi)

Docker (se ? o perché fatto prima o perchè da fare dopo con K8s)

- intro

- funzionamento di docker e container (come docker isola i container e separa dipendenze e roba da host e altri cont)

- immagini docker (snapshot con codice di applicazione, librerie e dipendenze + docker hub)

- volumi docker (x dati persistenti e condivisi)

- Docker Daemon e Docker Client (DD = serv background che gestisce container, DC = CLI UI x interagire con daemon)

- struttura Dockerfiles (desc dockerfile + config) ?

- layered file system (spiega come funge e come ottimizza dati e riduce duplicazione)

- link symbolic (desc symlink di docker x gestione dipendenze) ?

- deployment (e come funge)

- docker networking ?

- docker Sicurezza ?

- scalabilità ?

- docker compose ?

- best practices ?

Kubernetes

- intro

- funzionamento k8s (orchestrazione container)

- Master e nodi (M = contiene componenti controllo, N = contengono container in esecuzione)

- Unità di lavoro (pod, gruppo di container con stesso contesto di rete e archiviazione)

- Kubelet e Kube-Proxy (K = gestire container nei nodi, KP = gestire rete nei cluster)

- Namespace (x isolare risorse e oggetti in 1 cluster)

- Configurazione dichiarativa (come stato sistema definito con file YAML)

- Controller/ReplicaSet (x garantire n° desiderato di repliche di pod sempre in esecuzione + scalabilità orizzontale per aumentare o diminuire repliche di pod in base a carico di lavoro)

- Service discovery e Load balancing

- Deploy e aggiornamenti

- Persistenza dati (PV, PVC e Container Attached Storage) ?

- Sicurezza ?

- Monitoraggio e logging ?

- API ?

- Operators