ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SECONDA FACOLTA' DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

LA TECNOLOGIA SOFTWARE CONTAINER - RUOLO NELL'INGEGNERIA DEI SISTEMI SOFTWARE

Elaborata nel corso di: Sistemi Operativi LA

Tesi di Laurea di:
ANDREA FABBRI

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2016–2017
SESSIONE III

PAROLE CHIAVE

Software Container

Virtualizzazione

Macchina Virtuale

Microservizi

Docker

Indice

Introduzione

Le aziende di oggi sono costantemente sotto pressione per l'evoluzione digitale di questi anni. La lista di applicazioni e architetture software a loro disposizione è sempre più ampia e diversificata, ma le loro scelte sono spesso limitate dalle infrastrutture esistenti. E' necessaria una soluzione che permetta l'adozione e l'esecuzione di software indipendentemente dal tipo di struttura sottostante. Il modello dei container può essere un'ottima soluzione per questo genere di problema perché permette di creare una vera e propria indipendenza tra applicazioni e infrastruttura.

In questa tesi si studieranno i software container e le motivazioni che li hanno portati ad essere uno dei principali casi di studio e sviluppo nel mondo delle operazioni IT. Nel primo capitolo si osserveranno le ragioni che hanno portato alla nascita del modello dei container e le aziende che hanno partecipato al suo sviluppo. Nel secondo capitolo saranno presentati i risultati ottenuti da questa evoluzione, le principali piattaforme per la gestione dei container attualmente in uso e le caratteristiche che ciascuna di queste piattaforme ha sviluppato. Inoltre sarà presentato un confronto approfondito tra questo modello e le macchine virtuali. Infine, nel terzo capitolo, si studieranno gli innumerevoli vantaggi che i container hanno apportato al mondo dello sviluppo e distribuzione di software, senza tralasciare le problematiche ancora presenti attualmente e le soluzioni in fase di sviluppo.

Capitolo 1

Storia dei Software Container

In questo primo capitolo si introduce il modello dei container dandone una prima definizione e osservando l'analogia tra questi e i container fisici utilizzati nel mondo dei trasporti. In particolare, nel paragrafo 1.1 si vedranno le motivazioni che hanno portato alla nascita del modello e al suo successivo sviluppo mentre nel paragrafo 1.2 saranno presentati i principali eventi storici legati all'argomento e le relative date.

1.1 Cos'è un Container?

Un container è un ambiente software in grado di eseguire e isolare dall'esterno l'esecuzione di processi e applicazioni. Questo ambiente di lavoro è leggero, rapido nell'avvio e nello spegnimento, facilmente modificabile in base alle necessità e altamente portatile. Per comprendere al meglio i concetti e le motivazioni alla base del modello dei software container, si può osservare il funzionamento dei container fisici nel mondo dei trasporti. Quando delle merci vengono trasferite, devono passare attraverso una varietà enorme di mezzi di trasporto e depositi merci, come camion, treni, navi, magazzini e porti. Questi snodi di commercio e mezzi di trasporto devono essere in grado di gestire un'ampia varietà di merci di diverse dimensioni e con requisiti differenti (ad esempio, fusti di sostanze, scatole di elettrodomestici, sacchi di prodotti alimentari, materie prime, strumenti da lavoro, etc). Per la gestione di queste diverse tipologie di prodotti e per ridurre al minimo il costo dato dallo scaricamento e dalla movimentazione delle merci singole, è stato sviluppato il trasporto intermodale.

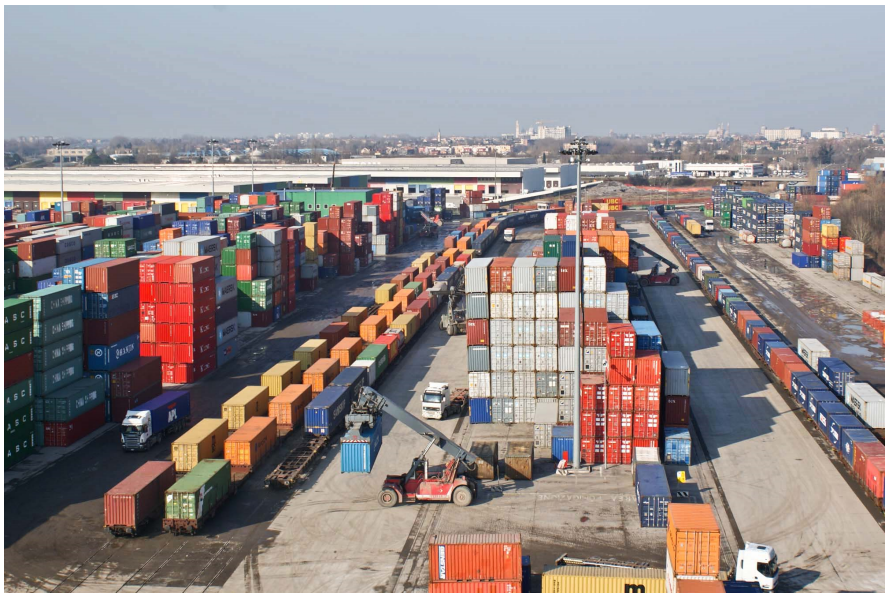

Figura 1.1: Interporto Padova SPA, un esempio di snodo commerciale per container.

Il trasporto intermodale impiega unità di carico, dette comunemente container, atte a essere utilizzate da diverse tipologie di mezzi di trasporto. Questi container presentano dimensioni standard, possono essere equipaggiati in base alle necessità per trasportare tipi di merci anche molto differenti tra loro (per esempio, un container isolato ermeticamente e refrigerato può essere utilizzato per merci deperibili) e sono progettati per essere spostati da un mezzo di trasporto ad un altro utilizzando la quantità minima di lavoro manuale (basta pensare alle gru utilizzate nei porti mercantili). La maggior parte delle macchine odierne utilizzate per la gestione delle merci sono progettate per lavorare con container intermodali. Si parla quindi di standardizzazione dei contenitori per il trasporto di merci. I vantaggi della standardizzazione non si fermano al solo involucro delle merci ma si estendono in tutto il sistema, dal semplice trasporto e trasferimento ordinato al metodo di catalogazione e indicizzazione dei prodotti.
Un software container condivide lo stesso obiettivo del trasporto intermodale, cioè lo sviluppo di un contenitore facilmente trasferibile che permetta l'esecuzione di una o più applicazioni al suo interno. La separazione data da questo involucro permette alle applicazioni eseguite al suo interno di svol-

gere i loro compiti in maniera sicura e senza che si debbano preoccupare dell'ambiente esterno.

La creazione di indipendenza tra il software e l'hardware o più in generale, l'indipendenza tra strati a diversi livelli, non è un concetto nuovo. Per esempio si pensi alle macchine virtuali, all'architettura software a strati e al modello OSI. E' un obiettivo che si è sempre perseguito perché i vantaggi di una architettura sviluppata a strati sono enormi: sviluppo indipendente dei diversi strati, maggiore sicurezza, interazioni semplici e dirette tra diversi livelli, facili interventi e manutenzione di piccole porzioni del software, etc. Il modello dei container è riuscito a portare tutti questi vantaggi a livello applicativo.

1.2 Nascita e Sviluppo dei Container

Il concetto di container, o più in generale, di ambiente di esecuzione isolato e controllato, è presente nel mondo dell'informatica già da molti anni.
Il modello odierno non è nato in un momento preciso né è stato sviluppato da una singola azienda. E' invece il risultato di una lenta ma continua evoluzione e grazie al codice diffuso in maniera continua e gratuita, sono state innumerevoli le aziende e le organizzazioni che hanno cooperato per il suo sviluppo: aziende come Google, Amazon e Red Hat hanno apportato il loro contributo facendo investimenti, inserendo modifiche al codice, creando le proprio piattaforme per la gestione di container, aggiungendo servizi e funzionalità al modello, etc. Fino a qualche anno fa, i software container non erano identificabili come una area di codice precisa di un kernel e la loro struttura era data da un insieme di astrazioni, componenti e meccanismi del sistema operativo sul quale operava. Al giorno d'oggi invece, dopo anni di sviluppi e interventi, sono disponibili piattaforme completamente autonome e perfettamente equipaggiate per l'utilizzo di container.

Le prime semplici forme di isolamento di file di sistema sono state introdotte per la prima volta nei vecchi sistemi Unix: attraverso l'uso dei comandi Chroot e Jail, si potevano creare alberi di cartelle nei quali eseguire una o più applicazioni. Un albero di cartelle di questo tipo segnava un confine invalicabile sia per le applicazioni al suo interno, sia per quelle

all'esterno. Queste primitive forme di isolamento sono state ampiamente utilizzate e anche se il modello ha impiegato molti anni prima di essere sviluppato in maniera adeguata, l'interesse nei suoi confronti è costantemente aumentato portandolo a una lenta ma continua crescita. Dopo la creazione delle prime forme di contenimento, il passo successivo è stato l'introduzione di strumenti, librerie e applicazioni per la loro gestione. Con il passare del tempo infatti, sempre più aziende si sono accorte delle potenzialità del modello dei container e hanno sviluppato e integrato nelle proprie applicazioni nuove versioni personalizzate del modello. Arrivando al giorno d'oggi, l'ultimo passo nell'evoluzione è stato la creazione di vere e proprie piattaforme completamente indipendenti e ricche di strumenti per la gestione di funzionalità avanzate legate ai software container.

Osserviamo i principali contributi all'argomento e le date relative:

- 1979: nascita della chiamata di sistema Chroot[1].
  La chiamata di sistema chroot (Change Root) venne introdotta durante lo sviluppo di Unix V7 nel 1979. Chroot permetteva di cambiare la cartella di root di un processo in esecuzione e un esempio di utilizzo consisteva tipicamente in due fasi: prima di tutto si creava un albero di cartelle in cui copiare o spostare tutti i file di sistema necessari all'esecuzione, quindi si utilizzava la chiamata di sistema chroot() sul processo bersaglio per modificare la sua directory di riferimento. In questo modo, il processo "trasferito" e tutti i suoi eventuali processi figli si ritrovano a lavorare in un nuovo ambiente appositamente preparato. Dato che i processi all'interno di quell'ambiente non possedevano i permessi necessari per accedere in lettura o in scrittura a file al di fuori del proprio albero di cartelle modificato, si poteva affermare con sicurezza che l'operazione di chroot era in grado di creare un certo livello di separazione tra l'esterno e l'interno di un ambiente di esecuzione. Inizialmente questa capacità fu utilizzata per test e debug di applicazioni e solo in seguito ci si rese conto delle grandi potenzialità di questo comportamento incapsulante.

- 1991: primo uso concreto di una Jail[2].
  Un primo uso del termine *jail* applicato a chroot si ebbe in una relazione di un caso di studio di Bill Cheswick nel 1991. In quegli anni Cheswick era uno scienziato dei laboratori AT&T Bell e ideò un espe-

rimento per studiare e monitorare i movimenti e i comportamenti di un cracker. Creò un honeypot (letteralmente "barattolo di miele", cioè un insieme di risorse allettanti e facilmente accessibili) all'interno di un ambiente isolato per attirare e imprigionare il cracker. Questo isolamento venne definito Jail (prigione) e ovviamente venne creato tramite il comando chroot. Nonostante l'esperimento ottenne ottimi risultati, le jail sviluppate tramite la chiamata di sistema chroot vennero definite troppo complesse da gestire e poco sicure.

- 2000: prima Jail ufficiale in FreeBSD[3].
  FreeBSD continuò l'espansione del concetto di isolamento introducendo il comando Jail con FreeBSD 4.X e migliorandolo nelle versione successive. Il meccanismo di FreeBSD Jail consentiva agli amministratori di sistema di suddividere un sistema di computer in diversi sistemi indipendenti più piccoli, denominati jail, con la possibilità di assegnare un indirizzo IP e configurazioni personalizzate a ognuna di quelle partizioni. Il comando Jail venne ideato e utilizzato principalmente per creare una separazione netta tra i propri servizi e quelli dei propri client. I principali benefici ottenuti da questo comportamento furono un incremento della sicurezza e una migliore e più facile amministrazione del sistema.

- 2000-2008: primi strumenti per la gestione di container.
  Dagli anni 2000 in avanti ci sono stati diversi sviluppi riguardo all'argomento e negli ambienti specializzati ha cominciato a diffondersi il termine container. In questo periodo, container e jail sono spesso utilizzati come sinonimi e riguardano soprattutto questioni legate alla sicurezza, alla indipendenza e alla separazione tra diversi ambienti di lavoro. Diverse aziende presentano la propria applicazione per la gestione di jail e container; per esempio, nel 2005 Virtuozzo[4] (una azienda capofila nello sviluppo e distribuzione di codice open source) sviluppa Open VZ, una virtualizzazione a livello di sistema operativo per la creazione di container, la quale è in funzione e utilizzata ancora al giorno d'oggi. Altre tecnologie sviluppate sono: Linux Vserver[5], un meccanismo di jail che può partizionare le risorse (file di sistema, indirizzi di rete, memoria) su un sistema informatico; Oracle Solaris Containers, il quale combina i controlli delle risorse di sistema con la separazione fornita dall'isolamento dei container; Process Containers,

lanciato da Google e focalizzato sulla gestione dell'uso delle risorse fisiche (CPU, memoria, I/O del disco, rete) senza l'utilizzo di una vera e propria macchine virtuale. Process Containers è stato rinominato "Control Groups (cgroups)" un anno dopo e infine unito al kernel Linux 2.6.24.

- 2008: rilascio di LXC (Linux Containers)[6].
  LXC è stata la prima implementazione completa di gestione container su Linux, ottenuta ragruppando Control Groups, Namespaces di Linux e tutte le tecnologie sviluppate precedentemente (ad esempio chroot e jail). LXC attraverso una semplice ma potente API, consente agli utenti Linux di creare e gestire facilmente container di sistema o applicazioni. Essendo il primo vero pacchetto completo per l'utilizzo di container e data la sua natura open source, LXC venne utilizzato ampiamente negli anni successivi dalle tecnologie più moderne e di successo; per esempio Warden nel 2011 e Docker nel 2013.

- 2011: Warden[7].
  CloudFoundry ha avviato Warden nel 2011, utilizzando LXC nelle fasi iniziali e in seguito sostituendolo con la propria implementazione. Caratteristiche distintive di Warden erano la capacità di creare ambienti isolati in ogni sistema operativo e l'utilizzo di demoni per la gestione di questi ambienti. Inoltre sviluppò modelli client-server per l'utilizzo di collezioni di container su più host. Warden è in funzione ancora oggi assieme alla piattaforma CloudFoundry.

- 2013: LMCTFY.
  Let Me Contain That For You (LMCTFY) era una implementazione open source lanciata nel 2013 da Google. Funzionava in maniera molto simile a LXC (utilizzava Control Groups) e lavorava allo stesso livello fornendo container per applicazioni Linux. Una novità decisiva di questa applicazione era la capacità di rendere le applicazioni *container aware*, cioè erano in grado di interagire con il container e di avviare i propri sotto container. La distribuzione attiva di LMCTFY si è interrotta nel 2015 dopo che Google ha iniziato a contribuire a libcontainer (la nuova libreria base di Docker) con codice e concetti appartenenti a LMCTFY.

- 2013: Docker[8].
  Nel 2013 viene rilasciato al pubblico la prima versione open source di Docker. Anche questo progetto, come Warden, è nato utilizzando LXC. Successivamente, sempre come Warden, ha sviluppato e adottato una nuova implementazione personale chiamata libcontainer. Grazie alla semplicità di utilizzo e alla politicà di diffusione del proprio codice, Docker è diventata l'applicazione capofila nel settore dei container. Come molte altre aziende, ha sempre messo a disposizione gratuitamente il proprio codice sviluppato, collaborando con la comunità mondiale di sviluppatori. In questo modo ha favorito lo sviluppo, la diffusione e il miglioramento nella risoluzione di bug e altri tipi di problematiche. A giugno 2015 Docker donò la specifica e il codice di runtime (ora noto come runC) all'Open Container Initiative (OCI) per aiutare a stabilire la standardizzazione man mano che il modello dei container cresceva e maturava.

- 2015: OCI[9].
  Docker, CoreOS e altre aziende di punta nel proprio settore, fondano nel 2015 l'Open Container Initiative (OCI). OCI è una comunità *open source*, nata con l'esplicito scopo di creare standard aperti per i container in ambienti Unix. Al giorno d'oggi continua con il proprio lavoro e i suoi standard sono ampiamente diffusi e accettati.

- 2016: Windows Containers[10].
  Nel 2016 la Microsoft aggiunge nei sistemi operativi Windows Server moderni la possibilità di inserire container per applicazioni Windows. Successivamente, tramite l'uso di particolari implementazioni di macchine virtuali, questa disponibilità è stata estesa anche ad applicazioni non appartenenti a Windows, come per esempio Docker. Nei più recenti sistemi operativi della Microsoft (ad esempio Windows 10) l'operazione è ancora più semplice perché non necessita più di macchine virtuali di supporto e Docker è in grado di eseguire i propri container su Windows in maniera nativa.

1.3 Osservazioni sullo Sviluppo del Modello nel Tempo

Osservando le motivazioni che hanno portato all'evoluzione del modello dei container, si nota subito il profondo legame con l'ambiente lavorativo. Ogni cambiamento del modello è sempre stato apportato per motivazioni legate alla sicurezza dello sviluppo di software, oppure per migliorare la distribuzione di applicazioni o per ridurre le spese negli ambienti di sviluppo. Sono state innumerevoli le aziende che hanno prodotto la propria versione del modello proprio perché ognuna di queste cercava di portare nel proprio ambiente di sviluppo i vantaggi dei container. Si parlerà di questi vantaggi nel prossimo capitolo e di come hanno cambiato le tecniche di sviluppo del software nel capitolo 3.

Capitolo 2

I Software Container

In questo capitolo sono presentate le caratteristiche principali del modello dei container e gli aspetti che ne hanno decretato il successo. Si parlerà in dettaglio di leggerezza, rapidità, portabilità e sicurezza dei container nei paragrafi 2.1, 2.2, 2.3 e 2.4. Successivamente si confronteranno i container con le macchine virtuali in quanto le similitudini tra questi due modelli sono innumerevoli. Questa analisi verrà svolta nel paragrafo 2.5 e si osserveranno gli aspetti in comune, le differenze tra i due modelli e quale dei due è il migliore. Nel paragrafo 2.4 verrano presentati alcuni esempi di problematiche legate alla sicurezza nelle principali piattaforme utilizzate al giorno d'oggi, le quali verranno approfondite nel paragrafo 2.6 tramite lo studio dei loro punti di forza e le loro caratteristiche distintive.

2.1 Caratteristiche dei Software Container

Lo scopo principale del modello dei software container è la creazione di un ambiente di esecuzione personalizzato, indipendente e isolato da tutto il resto. Quindi, in termini comuni, un container lo si può definire come un ambiente di esecuzione che permette di eseguire processi e applicazioni in maniera isolata dal resto del mondo. Da un punto di vista più tecnico e informatico invece, un container è una istanza runtime di un tipo particolare di immagine software. Questa immagine è un pacchetto eseguibile, leggero e autonomo che include tutto quello di cui c'è bisogno per eseguire del software. Diversamente dalle macchine virtuali, le quali virtualizzano l'intera infrastruttura fisica (processore, memorie, connessioni di rete, etc),

un container attiva una istanza virtuale solo dello spazio utente, cioè dell'ambiente di esecuzione a livello applicativo. Quindi i container si possono definire anche come spazi utente, isolati gli uni dagli altri e in esecuzione sul sistema operativo di un calcolatore.

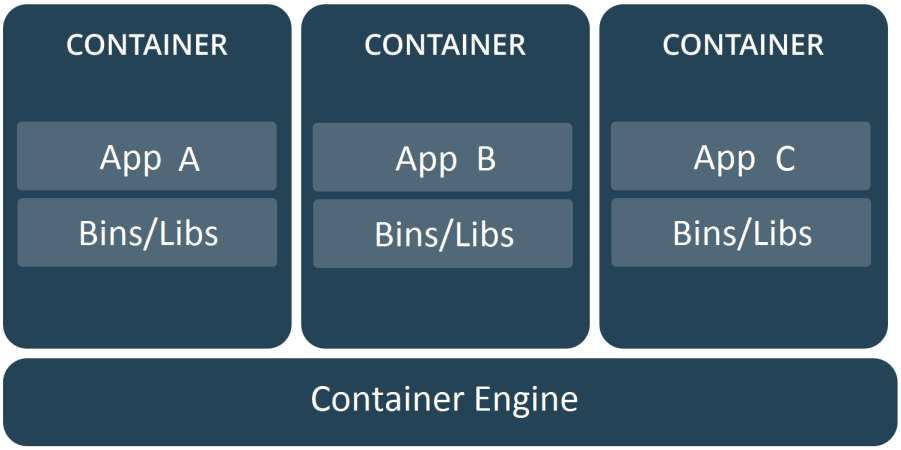

Figura 2.1: Esempio di utilizzo di software container

Quando si crea un nuovo container, bisogna assegnargli un certo contesto, cioè bisogna indicare un percorso ad una o più cartelle nelle quali saranno contenuti i file che si vogliono utilizzare e che potrebbero essere utili alla applicazione da eseguire al suo interno (si può indicare anche una cartella vuota o una cartella alla base di un albero già sviluppato e utilizzato da altri container). I file contenuti in quelle cartelle saranno utilizzati nel nuovo container e qualunque applicazione eseguita al suo interno sarà completamente autonoma nella loro gestione e potrà accedergli in completa libertà. Data questa capacità di personalizzare nel dettaglio l'ambiente di esecuzione, ogni container è tipicamente dedicato ad un contesto applicativo specifico e presenta quindi, caratteristiche specifiche per il tipo di applicazioni che verranno eseguite al suo interno. Questo significa che si possono ottenere tanti piccoli ambienti di esecuzione, con caratteristiche anche molto differenti tra loro, per l'esecuzione di tante piccole diverse applicazioni. E' un concetto alla base di diverse tecniche odierne di sviluppo software; per esempio l'architettura a microservizi punta allo sviluppo di un sistema composto da tanti piccoli moduli indipendenti e interagenti tra

di loro. Si parlerà più in dettaglio di questo aspetto nel capitolo 3. Anche con il modello delle macchine virtuali si possono avere tanti diversi ambienti di esecuzione ma ad un costo molto maggiore: infatti, mentre le macchine virtuali utilizzano quantità elevate di risorse e necessitano di tempo per avviarsi, i container presentano una struttura molto più leggera e in grado di attivarsi più rapidamente.

Riassumendo, quando si vuole creare una nuova applicazione basata su container, il lavoro degli sviluppatori consiste tipicamente in due fasi:

1. preparazione e personalizzazione dell'ambiente di esecuzione. In questa fase si scelgono o si creano le immagini desiderate in maniera da approntare un ambiente di lavoro adeguato. Si predispone inoltre, l'applicazione da eseguire una volta che il container sarà avviato;

2. avviamento del container. Dalle immagini create precedentemente siamo in grado di avviare il container (o i container) con l'applicazione al suo interno. Tutte le modificate effettuate in ambiente runtime sono applicate solo al container e non si ripercuotono sulle immagini utilizzate.

Leggerezza, rapidità nell'avvio, autonomia e la capacità di creare un ambiente isolato, hanno portato al successo e alla ampia diffusione nel mondo dell'informatica del modello dei container.

2.2 Leggerezza e Semplicità

Le applicazioni odierne per la gestione dei container consistono tipicamente in una piattaforma installata al di sopra del sistema operativo; questa piattaforma crea un certo livello di astrazione e ha il pieno controllo sui container permettendo la loro istanziazione, gestione ed eliminazione. Dato che l'astrazione non è di basso livello ma avviene a livello applicativo, più container attivi possono condividere facilmente il sistema operativo, le librerie e altri file comuni necessari all'esecuzione delle applicazioni. Questo tipo di condivisione riduce al minimo l'utilizzo del disco e delle altre risorse, permettendo e incentivando l'esecuzione di più container sulla stessa macchina host senza che questo influisca sulle prestazioni. La stessa piattaforma per la gestione dei container presenta tecniche avanzate per l'allocazione delle

risorse estremamente flessibili e dinamiche. Significa che è in grado di variare la quantità di risorse messe a disposizione ad un certa applicazione molto velocemente e senza interromperne l'esecuzione.

L'installazione di una nuova piattaforma per la gestione dei container richiede pochissimo tempo. Per esempio, l'installazione di Docker Toolbox sul sistema operativo Windows 7 è eseguita in pochi minuti. Una volta completata l'installazione della piattaforma, si può già cominciare a impostare la prima immagine, creando la cartella che farà da root del nostro ambiente di lavoro, copiando i file che abbiamo intenzione di utilizzare, etc. Una volta predisposto l'ambiente di esecuzione desiderato (il container), possiamo montare la nostra applicazione al suo interno, avviare il container ed eseguirla come se stessimo lavorando in maniera nativa sul sistema operativo. L'intero procedimento è molto semplice e i tempi sono ridotti al minimo (tempi di avvio dei container sono dell'ordine di millisecondi). Come se non bastasse, le piattaforme odierne per la gestione dei container semplificano ulteriormente la preparazione e l'avvio di nuove istanze, mettendo a disposizione per gli utenti, immagini già funzionanti e collaudate: gli utenti possono scaricare ed eseguire applicazioni complesse senza dover perdere tempo per risolvere problemi legati alla configurazione, all'installazione o preoccuparsi delle specifiche richieste dal proprio sistema. Date le ridotte dimensioni di queste immagini, anche questa operazione è molto veloce e poco impegnativa per quanto riguarda l'uso di risorse del sistema.

La struttura stessa del modello dei container implica un approccio leggero e semplice, infatti l'architettura software di un container è costituita da piccoli strati: prima si crea una immagine (l'immagine stessa è un agglomerato di diversi strati e si parlerà di questo nel caso specifico di Docker nel paragrafo 2.6) e poi si avvia il container come involucro più esterno. Quest'ultimo strato prende il nome di "container layer". Qualunque cambiamento o modifica fatta in esecuzione all'interno di un container (per esempio, la scrittura e l'eliminazione di file) viene scritta solo nello strato più esterno e lascia invariata l'immagine e i file usati. File utilizzati ma non modificati non vengono copiati. Questo tipo di funzionamento non solo porta alla condivisione di risorse tra i diversi container ma permette anche di avere il "container layer" il più piccolo possibile. Inoltre tutte le modifiche effettuate in quel ambiente sono locali ed eliminabili semplicemente con l'e-

liminazione del container. Le potenzialità di questo tipo di approccio sono innumerevoli: per esempio si può avere una sola immagine e innumerevoli container in esecuzione su quell'immagine, ciascuno con il proprio stato e il proprio ambiente di esecuzione. Tutte le modifiche sviluppate da quei container sono limitate al loro ambiente di esecuzione e l'immagine utilizzata come *pattern* rimane invariata, pronta per essere utilizzata da altri container e altre immagini.

Un altro aspetto dei container che contribuisce alla sua leggerezza è la fase di preparazione dell'ambiente di esecuzione: ogni container è tipicamente predisposto per l'esecuzione di una applicazione, cioè i file al suo interno sono scelti in base alle caratteristiche dei processi che verranno eseguiti. Significa che difficilmente, all'interno di un container, saranno presenti risorse non necessarie e file inutilizzati. Questo comportamento permette di preparare container di dimensioni anche molto piccole. Per esempio, una applicazione di piccole dimensioni che necessita di poche risorse deve essere eseguita all'interno di un container: lo stesso container occuperà sicuramente poco spazio, essendo fatto su misura delle dimensioni dell'applicazione.

Il modello dei software container è stato sviluppato per lavorare con applicazioni piccole e leggere e in quel caso rivela tutta la sua potenza. Se al suo interno si utilizza una applicazione pesante che necessita di modificare grandi quantità di dati, si perde la leggerezza del "container layer". In questi casi è consigliabile usare alternative alle classiche piattaforme per la gestione dei container o anche altri modelli per la virtualizzazione (per esempio una macchina virtuale). Dei problemi legati alla proliferazione di container, o più in generale, della utilizzo di container con applicazioni complesse e di grandi dimensioni, si parlerà nel paragrafo 3.2.

2.3 Isolamento e Portabilità

Dato il basso consumo di risorse hardware e software ottenuto con l'uso dei container, è molto facile che in una sola macchina siano presenti più di uno di questi ambienti di esecuzione. Più applicazioni possono essere in esecuzione in un calcolatore, ciascuna all'interno del proprio container. E' imperativo che tutte queste applicazioni si trovino in ambienti separati gli

uni dagli altri in maniera da avere una esecuzione sicura senza la possibilità di avere problemi di concorrenza, effetti a catena causati da altri processi, accessi a risorse modificate da altri processi, etc.

Il modello dei container è nato con lo scopo di separare un ambiente di esecuzione dal resto del mondo; al giorno d'oggi questo aspetto è stato ampiamente sviluppato e si può dire che il software in esecuzione all'interno di un container è perfettamente isolato dall'ambiente circostante. Significa che l'applicazione in esecuzione all'interno del container è completamente indifferente ai cambiamenti esterni al suo ambiente di esecuzione e che funzionerà sempre allo stesso modo, indipendentemente dall'ambiente al quale appartiene il container, dal sistema operativo dell'host e dalle caratteristiche fisiche della macchina. Questo comportamento permette di spostare un container dalla macchina nella quale sta lavorando, ad un qualsiasi altro ambiente da lavoro mantenendo le stesse funzionalità e garantendo lo stesso livello di astrazione al livello applicativo.

Le principali piattaforme odierne per la gestione di container si basano su standard aperti e ampiamente condivisi e questo permette di lavorare con i container su tutte le principali distribuzioni Linux, Microsoft Windows e su molte altre infrastrutture, incluse le macchine virtuali e il cloud. Questa disponibilità gratuita di standard è dovuta principalmente al fatto che il modello dei container è nato e si è sviluppato in sistemi Unix in pieno stile open source. Molte delle aziende che hanno partecipato a questa evoluzione hanno messo a disposizione gratuitamente il proprio codice sviluppato, al punto da fondare iniziative e consorzi per lo sviluppo e la diffusione della tecnologia (per esempio, OCI - Open Container Initiative, CNCF - Cloud Native Computing Foundation). La distribuzione gratuita di codice e standard ha portato ad un aumento considerevole del numero di sviluppatori e ne ha ulteriormente rafforzato il mercato. Questo è un altro degli aspetti principali che ha portato alla diffusione e al successo di questo modello.

2.4 Sicurezza e Robustezza

Sono necessari molti accorgimenti per assicurare completamente la sicurezza di un ambiente di esecuzione. Daniel J Walsh[13] ha detto che le

regole principali da applicare ai container sono le stesse di qualunque altra applicazione server: "limita i privilegi dell'utente il più possibile e non lavorare nella cartella di root". Questa pratica è estremamente importante per quanto riguarda i container e la sicurezza da loro esercitata.

Il modello dei container è spesso definito "sicuro" perché applicazioni, processi e utenti all'interno di un container non possono uscire al di fuori dei confini del loro ambiente di lavoro, né possono valicare i confini di un altro container. Questo isolamento è dato dal fatto che al momento della creazione del container, si sceglie un albero di cartelle che farà da base all'ambiente di esecuzione. Se questo albero è diverso dal root e l'utente al suo interno non ha particolari permessi, allora questo utente è obbligato a lavorare solo in quell'albero. Data questa struttura, si riscontra il primo vero problema del modello dei container (e probabilmente anche il più importante al giorno d'oggi). Si osservi un classico esempio di utilizzo dei container: una macchina server è in funzione con diversi container avviati, ciascuno con una singola applicazione in esecuzione al suo interno. Su ogni applicazione sta lavorando un utente. Questi utenti sono vincolati a lavorare all'interno del proprio container perché non hanno i permessi sufficienti per cambiare root. Se per qualche motivo particolare l'utente all'interno di un container avesse permessi da amministratore, potrebbe facilmente evitare l'intero isolamento del container, uscire dal proprio ambiente di esecuzione e danneggiare l'host. Un altro esempio riguarda la modalità di avviamento di container: alcune piattaforme preparano ed eseguono i propri container nella cartella di root; successivamente provvedono a cambiare l'albero di cartelle e i permessi dell'utente legato a quel container. Il problema è che per un breve lasso di tempo, l'utente ha pieni permessi di accesso in una area molto importante del sistema. Un utente malitenzionato potrebbe approfittare di questo punto debole per arrecare danno all'intero sistema. Lo stesso Docker (per dettagli più approfonditi si legga il paragrafo 2.6), dato il suo demone Dockerd che gestisce tutto il sistema in maniera centralizzata, utilizzava questa modalità di lavoro fino a poco tempo fa. Un ulteriore aspetto insicuro dei container è il fatto che molte piattaforme si sono sviluppate su sistemi di tipo Unix, utilizzando tecnologie sviluppate per compiti diversi dall'assicurare la sicurezza. Per esempio, Docker utilizza Namespace di Linux attraverso i quali altera la visione dell'intero sistema ai processi interni ai container. Il problema principale è che il sistema operativo ha parti che non fanno parte dei Namespace e se un utente riesce a comunicare

o attaccare in qualche maniera una di queste aree, può estendere il proprio controllo su tutto il sistema. Per diversi motivi quindi, non si può dire che il modello dei container sia completamente sicuro.

Molte piattaforme hanno cercato di migliorare questo aspetto e si sono sviluppate per assicurare una maggiore sicurezza alle applicazioni in esecuzione sui propri container. RKT (si veda il paragrafo 2.6 per approfondire) è una di queste e non fa uso di Namespace ma di meccanismi standard Unix, ampiamente utilizzati e verificati, per la gestione diretta dei privilegi tra diverse operazioni. Inoltre utilizza strumenti per controllare che non avvengano modifiche dei privilegi impreviste.

I servizi web e le applicazioni distribuite odierne necessitano di un alto livello di robustezza. Questo aspetto può essere facilmente garantito attraverso l'uso di copie ridondanti di file. In questa maniera, se una parte della rete si guasta o diventa irraggiungibile, subentra al suo posto la copia o le copie del software danneggiato. L'utilizzo di copie ridondanti ha anche altri vantaggi: in ambienti distribuiti, avere copie locali diminuisce il consumo di banda e migliora le prestazioni percepite a livello utente. Il modello dei container è molto efficiente per quanto riguarda la creazione di copie di file: non solo i container sono facilmente replicabili con la minima spesa, ma la loro mobilità è perfetta per questo scopo. Si può immaginare un semplice esempio di utilizzo: un server è in funzione e mette a disposizione un certo servizio utilizzando container; un secondo server di riserva è pronto ad attivarsi in caso di guasto della macchina primaria e in una zona di memoria extra vengono salvate e costantemente aggiornate le immagini e i container utilizzati dai client. Se si guasta il server principale, (senza contare i tempi di avvio delle infrastrutture fisiche) è questione di pochi secondi avviare una serie di nuovi container in una nuova macchina. E la perdità di dati è minima in quanto è molto semplice e veloce l'operazione di caricamento di sessioni precedentemente usate. L'alta mobilità dei container inoltre, permette di bilanciare la rete del sistema nel caso si presentino carichi di lavoro o cali di performance in certe aree.

Osservando gli esempi presentati in questo paragrafo, si capisce che il modello dei container non è un sistema perfettamente sicuro a prova di cracker. Sono diversi i punti deboli facilmente raggiungibili e attaccabili.

Al giorno d'oggi, nel mondo del web, è fondamentale avere strumenti che garantiscono la sicurezza delle applicazioni e per questo le aziende maggiormente coinvolte nei vari progetti legati ai software container continuano a investire in questo campo.

2.5 Confronto tra Macchine Virtuali e Container

Un confronto tra il modello delle macchine virtuali (vm) e il modello dei container è d'obbligo perché alcuni concetti alla base del funzionamento sono molto simili tra loro se non identici: container e vm hanno vantaggi di isolamento molto simili ed entrambi virtualizzano le risorse della macchina host. Per questo motivo si sente spesso parlare di container come "versione più leggera di macchine virtuali". Ovviamente non sono la stessa cosa e si nota una prima e profonda differenza osservando gli obiettivi fondamentali dei due modelli: le vm nascono per adempiere alla necessità di emulare un ambiente di esecuzione estraneo a quello attuale mentre lo scopo di un container è rendere le applicazioni portatili e autonome.

I container e le vm presentano un comportamento simile perché sono unità di esecuzione. Entrambi, attraverso la virtualizzazione, costruiscono una immagine che può essere eseguita su una piattaforma host adeguatamente equipaggiata. La stessa immagine può poi essere spostata su un altro host (sempre adeguatamente predisposto), senza che si presentino problemi nell'esecuzione delle applicazioni al suo interno. Una grande differenza tra i due modelli in questo caso, sta nel fatto che i container virtualizzano il sistema operativo mentre le vm virtualizzano tutto l'hardware. Si parla di virtualizzazione a due livelli completamente diversi, con grandi differenze sia nel funzionamento, sia nell'utilizzo delle risorse.

Un altro aspetto simile tra i due modelli è l'isolamento da loro creato, infatti, entrambi permettono l'esecuzione di applicazioni al loro interno senza che queste possano accedere a file e risorse esterne al container o alla vm. E' differente invece, il modo in cui sono ottenuti questi isolamenti: le vm sono isolate dai supervisori che fanno uso di funzionalità hardware della infrastruttura mentre i container utilizzano controlli software forniti dal

kernel. Questo significa che i processi in esecuzione all'interno dei container sono equivalenti ai processi nativi sull'host e non comportano i costi associati all'esecuzione del hypervisor delle vm. Un'altra importante differenza per quanto riguarda l'isolamento è il "cosa" viene isolato: i container separano l'ambiente interno da quello esterno a livello applicativo mentre le vm operano ad un livello più basso. Quindi più container possono condividere tra di loro file appartenenti al sistema operativo mentre le vm necessitano di risorse personali e private (una copia completa del SO, l'applicazione in esecuzione, le librerie necessarie, l'hardware virtualizzato inclusi adattatori di rete virtualizzati, memorie e CPU, etc). Questa differenza è il motivo principale per il quale si dice che i container sono più leggeri, portatili ed efficienti (nell'uso delle risorse) rispetto alle vm. Le seguenti figure mostrano la differenza appena citata:

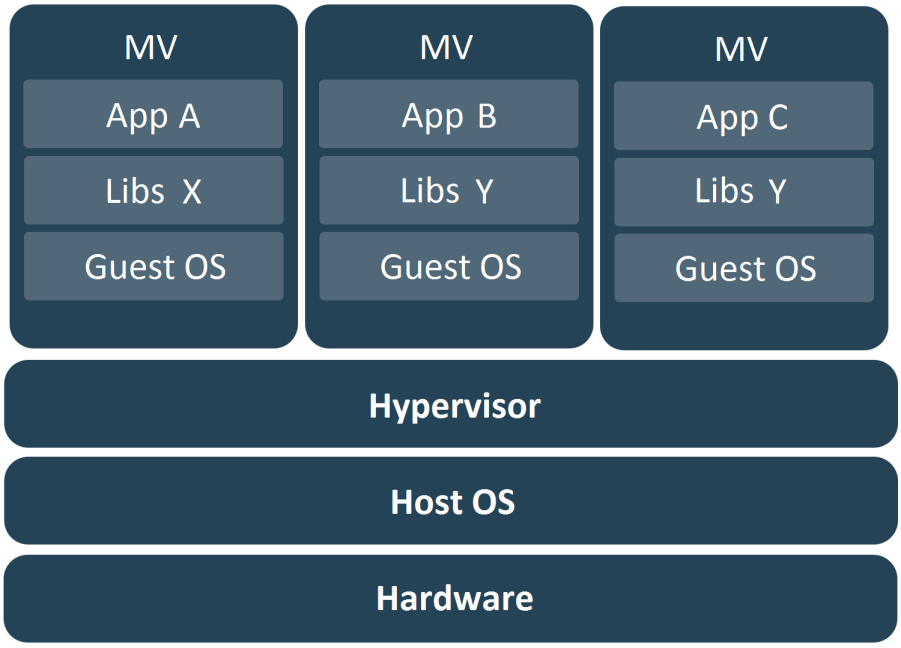

Figura 2.2: Esempio di sistema utilizzante macchine virtuali

La Figura 2.2 mostra tre applicazioni eseguite su 3 distinte vm avviate sullo stesso host. Il hypervisor crea ed esegue le macchine virtuali, controllando l'accesso al sistema operativo, all'hardware e alle chiamate di sistema

quando necessario. Le applicazioni B e C, eseguite sulle vm due e tre, necessitano della stessa libreria Libs Y ma non possono condividerla tra loro. Il fatto che ogni vm debba possedere una copia personale e privata delle risorse utilizzate, può portare ad avere ridondanza e una diminuzione delle prestazioni.

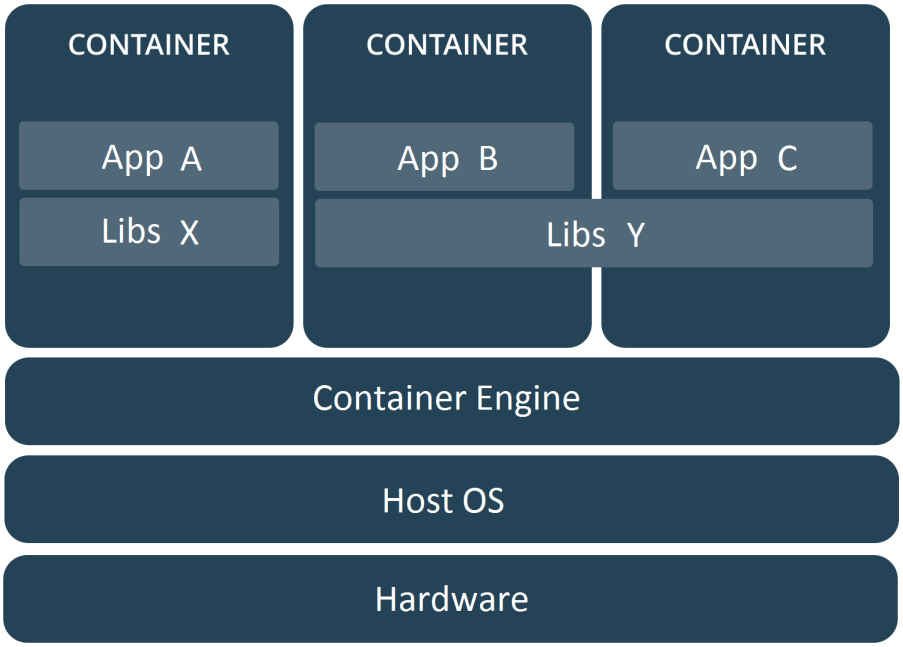

Figura 2.3: Esempio di sistema utilizzante i software container

Al contrario, la Figura 2.3 mostra come le stesse tre applicazioni possono essere eseguite in un sistema che sfrutta il modello dei container. A differenza delle macchine virtuali, il kernel è condiviso tra i container in esecuzione. Le applicazioni B e C usano le stesse librerie e possono condividerle piuttosto che avere copie ridondanti. In questo caso è il container engine a essere il responsabile dell'avvio e dell'arresto dei container in modo simile al hypervisor su una vm.

Nell'avviamento di nuove istanze sono presenti ulteriori differenze tra i due modelli. Per quanto riguarda i container, vengono prima di tutto scelti i file ai quali si è interessati e successivamente si avvia il container layer.

Per una macchina virtuale invece si parte da un sistema operativo completo e si eliminano i componenti non necessari. Si tratta di due metodi di avviamento completamente diversi in quanto il modello dei container utilizza un approccio simile al modello di sviluppo bottom-up (strategia nella quale si scelgono prima di tutto i singoli elementi necessari e successivamente si aggregano tra loro) mentre le vm usano un approccio simile al modello top-down (si parte dal sistema completo e si scende solo successivamente nel dettaglio).

Si è visto nel paragrafo 2.2 che le tempistiche di avvio di applicazioni per la gestione di container sono dell'ordine di qualche minuto e che l'operazione nel suo complesso non impiega molte risorse. Per quanto riguarda le macchine virtuali le spese sono nettamente superiori: l'installazione di una applicazione per la creazione di vm (per esempio VMware e Sun VirtualBox) non impiega molto tempo in se stessa; l'installazione del sistema operativo invece, necessita di tempo, risorse fisiche e attenzione per il settaggio delle impostazioni. Avere un container in funzione per pochi minuti o anche secondi è un'idea del tutto ragionevole e fattibile mentre non lo è per le vm. Parlando della velocità dei due modelli si può introdurre anche l'aspetto legato allo spazio in memoria occupato. Ogni vm include una copia completa del sistema operativo, una o più applicazioni in esecuzione, file e librerie necessarie al funzionamento, etc. Tutto questo si riassume in decine di GB occupati. Dato il caricamento del sistema operativo inoltre, le macchine virtuali possono essere molto lente da avviare. Si è visto nel paragrafo 2.2 quanto può essere facile creare le immagini relative ad un container e quelle immagini sono in genere molto, molto più piccole rispetto alle vm. Per questo motivo, occupano pochissima memoria e possono essere attivate e disattivate molto rapidamente.

La sicurezza è un aspetto di fondamentale importanza quando si utilizzano tecnologie per la virtualizzazione in quanto i due concetti sono profondamente legati. Come già visto nel paragrafo 2.4, per quanto riguarda i container, bisogna fare molta attenzione che gli utenti non ottengano permessi da amministratore e che non riescano a uscire dal proprio ambiente controllato perché non sono presenti altre barriere di protezione tra il gestore dei container e il sistema operativo. In una vm il discorso è molto diverso. Anche se un utente riuscisse a prendere il controllo del sistema operativo

della vm, difficilmente riuscirebbe a superare la separazione creata dal hypervisor. Inoltre, le vm sono un software con una elevata esperienza mentre i container sono ancora relativamente giovani. Significa che le vm sono state testate in innumerevoli scenari, hanno superato attacchi di cracker e resistito a malware e virus. Tutte queste prove ne hanno rafforzato la struttura, migliorando la sicurezza e la stabilità. Questi aspetti, non solo possono essere di fondamentale importanza per lo sviluppo di determinate applicazioni, ma sono anche tenuti in alta considerazione dalle aziende sviluppatrici di software.

Qual'è il modello migliore tra i due?

Non esiste un vero vincitore tra questi due modelli perché differiscono tra loro sotto tanti aspetti (incluso il loro scopo originario) e quindi non ci può essere un vero confronto. Ci sono però casi d'uso nei quali un modello presenta pregi migliori e quindi funziona meglio rispetto all'altro. La leggerezza dei container si paga ad esempio con l'impossibilità di avere un container con ambienti di esecuzione diversi al suo interno. Una applicazione come VMWare invece è in grado di avere innumerevoli sistemi operativi installati e ambienti di esecuzione anche di diverso tipo al suo interno al costo però, di una quantità di risorse utilizzate nettamente superiore. Data una piattaforma per container installata su una singola macchina, il sistema operativo in comune potrebbe diventare un altro punto debole dei container: in linea teorica un container potrebbe compromettere la stabilità del kernel stesso influenzando negativamente tutti gli altri container in funzione. Le vm sono più sicure sotto questo punto di vista e garantiscono una maggiore stabilità sempre al costo di un consumo di risorse nettamente superiore. Bisogna osservare il contesto nel quale si vuole lavorare prima di fare una scelta tra i due modelli. Se c'è bisogno di un ambiente di sviluppo o di esecuzione agile, veloce, flessibile allora i container sono la scelta migliore. Al contrario, le vm garantiscono una maggiore sicurezza, una migliore stabilità e una più grande varietà di ambienti di esecuzione e di sviluppo.

2.6 Piattaforme per la Gestione dei Container

Al giorno d'oggi sono disponibili diverse piattaforme per la gestione dei container tramite le quali possiamo costruire e scaricare nuove immagini, avviare container ed eliminarli quando non sono più utili. Lo stesso tipo di container può essere eseguito su macchine con caratteristiche fisiche (memoria, cpu, porte, etc) e software (sistema operativo, drivers, etc) anche molto diverse tra loro. Le applicazioni eseguite all'interno di un container di quel tipo, funzioneranno anche su tutte le macchine in grado di avviare quel container. Di seguito è presentato un elenco delle principali piattaforme utilizzate:

- Docker.
  Docker è nato come progetto interno di dotCloud (dotCloud era una compagnia di PaaS) e dopo anni di continui sviluppi è diventato l'azienda che guida il movimento dei container e l'unico fornitore di piattaforme container ad affrontare ogni applicazione attraverso il cloud ibrido. Si tratta di una piattaforma gratuita per lo sviluppo, la spedizione e l'esecuzione di software attraverso l'uso di container. Offre anche servizi a pagamento per la gestione di aspetti complessi spesso legati alla gestione aziendale e alla distribuzione del software sviluppato.
  Tutto il sistema Docker ovviamente ruota attorno all'uso di container e immagini. Le immagini sono composte da una serie di strati, ciascuno dei quali racchiude il precedente, e possono essere scaricate attraverso il cloud oppure create da zero utilizzando i Dockerfile. Gli strati delle immagini dipendono dalle istruzioni contenute nel Dockerfile, cioè ad ogni istruzione corrisponde uno strato della immagine risultante. Per esempio:

  Questo esempio di Dockerfile contiene quattro comandi, ognuno dei quali crea un livello sopra al livello precedente. L'istruzione FROM inizia creando uno strato dall'immagine ubuntu:15.04. Il comando COPY aggiunge alcuni file dalla directory corrente del client Docker. Il comando RUN crea l'applicazione usando il comando make. Infine, l'ultimo livello specifica quale comando eseguire all'interno del container.

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Figura 2.4: Esempio di Dockerfile[11]

Eseguendo il codice contenuto nell'esempio precedente, si è in grado di creare solo una immagine, cioè un template di sola lettura. Ancora non è stato avviato nessun tipo di processo o applicazione. Quando si avvia un nuovo container viene aggiunto un nuovo strato modificabile, detto appunto "container layer", in cima alla pila di strati precedentemente sviluppati dall'immagine. Quest'ultimo strato è quello nel quale lavorerà l'applicazione da eseguire e qualunque modifica effettuata durante l'esecuzione sarà scritta solo a questo livello.
Docker presenta anche funzionalità e strumenti per elaborare svariate situazioni. Per esempio, se si utilizzano applicazioni di grandi dimensioni e con un elevato numero di container in funzione, Docker Volumes (un volume è un insieme di cartelle persistente utilizzato per la condivisione di file tra container) e Docker Swarm (gestisce facilmente gruppi numerosi di macchine e di container) sono due strumenti adatti a gestire questa complessità.
Docker è scritto in linguaggio Go e sfrutta diverse tecnologie di Linux per poter svolgere le proprie funzioni tra le quali sono presenti:

1. Namespace
   Docker utilizza i namespace per sviluppare l'isolamento base dell'ambiente di esecuzione di un container e garantiscono che i container non si possano vedere o influenzare tra di loro. Quando viene creato un nuovo container, ogni suo aspetto viene eseguito in un separato namespace. Per esempio il namespace "pid" viene utilizzato per isolare i processi (più processi con lo stesso pid in diversi container), il "net" per la gestione di interfacce di rete, etc.

2. Control Groups
   I gruppi di controllo allocano e controllano le risorse (memoria,

cpu, I/O, etc) necessarie al funzionamento di una singola applicazione . Con questa tecnologia è molto facile gestire l'accesso di diversi container alle risorse hardware. Inoltre è flessibile e la quantità di risorse a disposizione di una applicazione può essere modificata anche durante la sua esecuzione.

3. Union File System
   Sono file system che operano creando strati e sono la tecnologia utilizzata per creare la struttura a livelli delle immagini e del container.

La piattaforma Docker ha due componenti principali: il Docker Engine (che è responsabile per la creazione e l'esecuzione di container) e Docker Hub (un servizio cloud per la distribuzione). Docker Hub fornisce un enorme quantitativo di immagini di container pubbliche da scaricare, consentendo agli utenti di iniziare a lavorare rapidamente ed evitare di duplicare il lavoro già svolto da altri. Il Docker Engine invece è un modello client-server gestore dell'interazione con gli utenti e i client e responsabile della creazione ed esecuzione dei container. Una architettura centralizzata in questa maniera è ottima per la distribuzione di software. Il Docker Engine presenta una interfaccia per la comunicazione tra client e server facilmente utilizzabile dagli utenti comuni. Prima di questa tecnologia (in particolare quando si usava LXC), per lavorare con una piattaforma container erano richieste conoscenze abbastanza specialistiche e una discreta quantità di lavoro manuale.
Docker Engine è composto da tre componenti principali:

1. un processo demone (Docker + Demon = Dockerd) che funge da server e che svolge la maggior parte del lavoro creando, gestendo ed eliminando tutti gli oggetti Docker (immagini, container, reti e volumi);
2. una REST API che specifica l'interfaccia attraverso la quale si può comunicare con Dockerd;
3. un client con interfaccia a linea di comando (CLI) che utilizza Docker REST API per controllare e interagire con Dockerd.

Nela figura sottostante possiamo osservare queste tre parti e l'incapsulamento creato dai tre livelli.

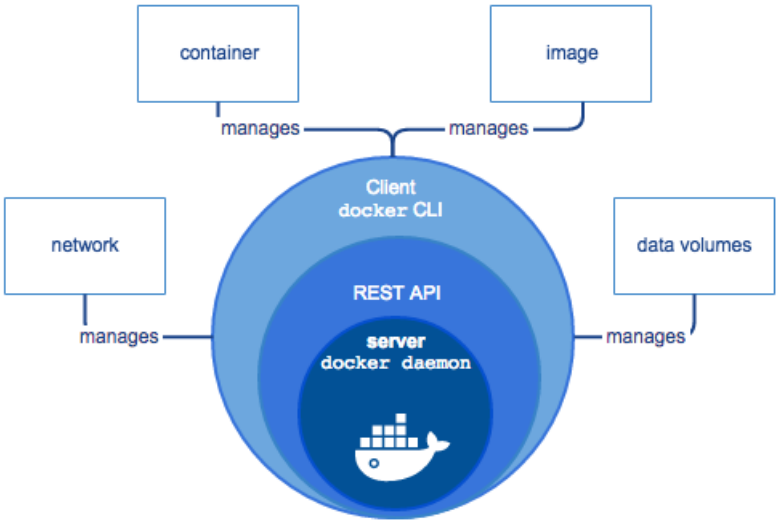

Figura 2.5: Struttura del Docker Engine[12]

Per studiare un classico esempio di funzionamento, l'intera struttra di Docker si può suddividere in tre aree di lavoro distinte: lato client, lato server e registro. Questo non significa che devono essere presenti tre macchine distinte per far funzionare Docker, infatti le tre aree possono trovarsi anche sullo stesso host. Il demone Dockerd è sempre in ascolto per eventuali richieste da parte di client o di demoni Dockerd di altre macchine. Un tipico utente interagisce solitamente con il Dockerd tramite il client Docker. Quando l'utente utilizza particolari comandi come *docker run*, il client comunica questi comandi a Dockerd, il quale porta a compimento l'operazione richiesta. La terza area di lavoro chiamata registro è di un database contenente tutti i pattern di immagini precedentemente sviluppati dalla comunità Docker. Si può utilizzare sia un registro privato e locale, sia i principali archivi pubblici come Docker Hub e Docker Cloud. E' sempre Dockerd che funge da intermediario quando un client necessita di immagini presenti in un certo registro, sia che questo sia locale o remoto.
Dalla versione 1.11 in avanti, Dockerd non gestisce più direttamente l'esecuzione dei container. Questa operazione è ora gestita da con-

tainerd (è uno standard runtime per container ampiamente diffuso e utilizzato principalmente da Docker e Windows). Più precisamente, il demone Docker prepara l'immagine come un pacchetto OCI e invoca containerd affinché avvii il pacchetto. Containerd quindi avvia il container usando runC.

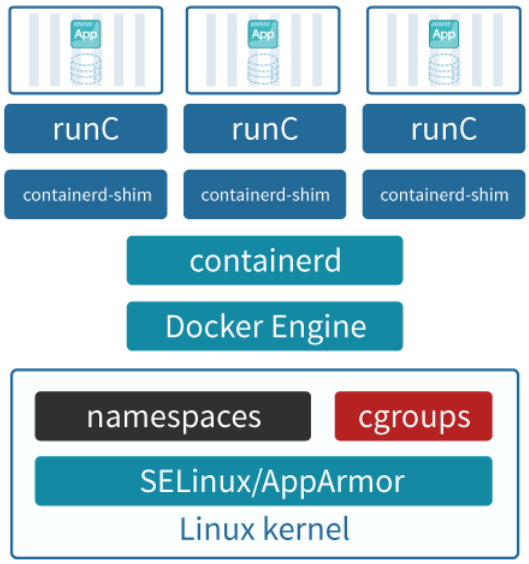

Figura 2.6: Struttura di Docker nelle versioni più moderne.

Vantaggi:

- Docker supporta diversi sistemi operativi e piattaforme cloud;
- Grazie al Docker Hub gli utenti hanno accesso a innumerevoli risorse Docker;
- Essendo la piattaforma più diffusa al momento, la comunità mondiale continua a sviluppare nuovi strumenti e funzionalità;

Svantaggi:

- Supporta esclusivamente il proprio formato di container;
- I container sono focalizzati sulle singole applicazioni e non offrono valide alternative per una virtualizzazione full system.

- RKT ("Rocket" o "Rock-It")
  RKT è il diretto avversario di Docker per quanto riguarda il mercato. Uno degli scopi principali di questa piattaforma è l'incremento della sicurezza dei propri container. Segue questo obiettivo utilizzando ulteriori funzionalità rispetto alle altre piattaforme come Docker. Alcune delle tecnologie usate sono Kvm (Kernel-based Virtual Machine, una virtualizzazione del kernel Linux), l'estensione SELinux (Security Enhanced Linux, provvede ulteriori funzionalità per la sicurezza) e ulteriori specifiche per il controllo per quanto riguarda le immagini e gli ambienti sviluppati.
  Un altro aspetto abbondantemente studiato e sviluppato da RKT è la portabilità; infatti sono tanti gli standard accettati e adottati tra i quali possiamo trovare il vecchio Application Container, le immagini Docker e diversi formati di OCI. Il fatto di supportare diversi standard permette di aumentare il numero di modi di preparare le immagini e quindi aumenta il numero di possibili sviluppatori e piattaforme partecipanti.
  Il nucleo centrale di RKT non funziona con il demone di Docker, fa uso invece di sistemi init affermati e abbondantemente utilizzati come systemd e upstart. Quindi RKT non presenta una architettura centralizzata attorno a un demone e avvia container direttamente dal client, in uno stile più conforme ai sistemi Linux rispetto a Docker.

  Vantaggi:

  - RKT supporta sia il proprio formato di container, sia quello di Docker ed è in grado inoltre di convertire ogni altra immagine nel proprio formato;
  - Livello di sicurezza elevato;
  - Alta robustezza dato il nucleo centrale abbondantemente verificato;

  Svantaggi:

  - Non è diffuso quanto Docker e quindi ha meno integrazioni e tecnologie sviluppate da terzi;

- I container sono focalizzati sulle singole applicazioni e non offrono alternative per una virtualizzazione full system.

- LXC
  Si tratta di una raccolta di template, tools e librerie che vanno a comporre una interfaccia per l'utente per le funzioni container native dei sistemi Linux. LXC è stato il primo vero pacchetto di strumenti per la gestione di container completo ed è tuttora utilizzato.

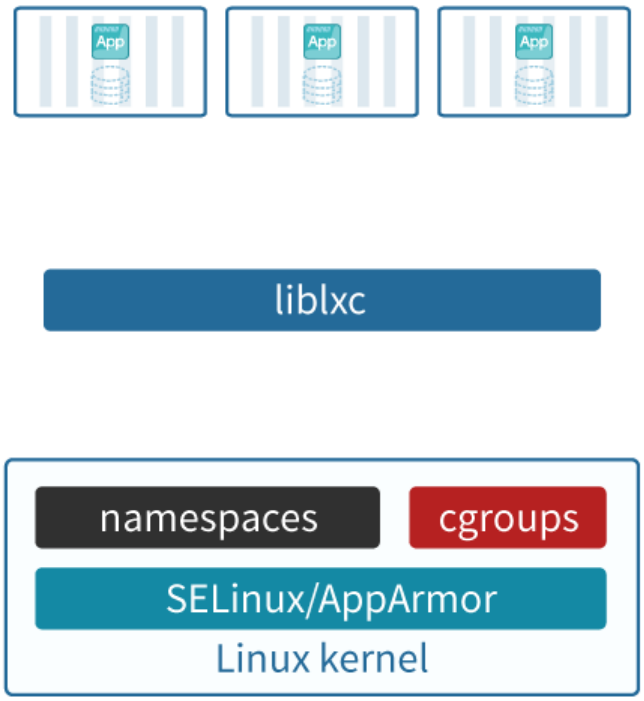

Figura 2.7: Struttura di LXC.

Si può osservare nella figura 2.8 che la libreria LXC presenta una struttura molto semplice se confrontata con piattaforme più complesse come Docker o RKT.
Come molte altre piattaforme, utilizza funzionalità legate ai gruppi di controllo e ai namespace. Lo scopo principale di LXC è quello di creare un ambiente per software container che si differenzi il meno possibile da una installazione standard di Linux. Si parla non di

ambiente container per singole applicazioni ma di container dedicati all'esecuzione di più processi.

Vantaggi:

- LXC è ottimizzato per l'utilizzo di container full system;

Svantaggi:

- LXC perde molte funzionalità se i container avviati si focalizzano sull'esecuzione di singole applicazioni;
- Non supporta altri sistemi operativi al di fuori di Linux.

- RunC
  Docker, Google, Microsoft, RedHat e altre aziende coinvolte nel settore svilupparono RunC con l'intenzione di creare una specifica runtime standard e comune basata su libcontainer e le altre tecnologie donate a OCI. Al giorno d'oggi persegue il suo obiettivo ed è ampiamente utilizzato dalle principali piattaforme per lo sviluppo di software container (per esempio, Docker).
  RunC è uno strumento a linea di comando per il controllo e la gestione di container in accordo con le specifiche di OCI. Permette un accesso semplificato e di basso livello a container in esecuzione senza creare carichi di lavoro per il sistema. Ha quindi un approccio poco invasivo ed è uno strumento ottimo per il debug di applicazioni in esecuzione. RunC è anche ampiamente utilizzato per i test e per le prime distribuzioni di nuove funzionalità legate al modello dei container. Per esempio è stato utilizzato per lo sviluppo e la diffusione di Linux Checkpoint/Restore In Userspace, una funzionalità attualmente utilizzata da altri linguaggi di più alto livello come Docker.

Vantaggi:

- RunC ha un approccio molto più leggero e di basso livello rispetto ad altre piattaforme come Docker e Rocket;
- Strumento potente per test e debug relativi alla sicurezza;
- Si basa su standard ampiamente accettati e utilizzati(OCI);

Svantaggi:

- Necessita di una minima conoscenza tecnica per l'utilizzo;
- Per potere creare immagini e container deve utilizzare strumenti esterni.

2.7 Riassumendo.....

Cosa comportano queste caratteristiche? Osservando il capitolo appena concluso si nota subito che i container presentano caratteristiche ottime per gli sviluppatori di software; data la disponibilità gratuita di piattaforme per la loro gestione, sono anche facilmente accessibili da semplici utenti e non solo dalle grandi aziende. Come si vedrà nel prossimo capitolo, lo sviluppo di software è il campo nel quale il modello dei container ha avuto più successo. Inoltre, saranno presentate diverse nuove tecnologie legate ai container, tecnologie attuali e con grandi potenzialità per il futuro.

Capitolo 3

Ruolo dei Software Container nell'Ingegneria

In questo capitolo si parlerà dei cambiamenti che il modello dei container ha portato nel mondo dell'ingegneria. Nel paragrafo 3.1 si studieranno le nuove metodologie di lavoro introdotte dai container per lo sviluppo di software. Successivamente, nel paragrafo 3.2, si osserveranno i principali casi d'uso odierni del modello.

3.1 Impatto dei Container nel Settore di Sviluppo del Software

Le conclusioni del capitolo 1 mostrano come l'evoluzione del modello dei container sia stata guidata dalla necessità di diverse aziende di rendere i propri ambienti di sviluppo di software compatibili con i container. Infatti, sono innumerevoli i vantaggi apportati dal modello al mondo della produzione di software e la sua adozione è stata in gran parte guidata dagli sviluppatori di applicazioni. Questi, negli ultimi anni, hanno ricevuto tutti gli strumenti necessari per utilizzare i container in modo efficace e produttivo.

Le caratteristiche studiate nel secondo capitolo, cioè rapidità nell'avvio, leggerezza, portabilità, isolamento, etc, sono tutti aspetti estremamente importanti per sviluppatori e aziende perché permettono di ridurre i costi

e i tempi di produzione. Osservando più in particolare le singole proprietà notiamo che:

1. La rapidità dei container nell'avvio e nello spegnimento è essenziale per gli sviluppatori che desiderano cicli di sviluppo rapidi ed iterativi in cui possano immediatamente vedere i risultati dei loro test. Il riavvio dell'ambiente di esecuzione e il recupero della sessione di lavoro precedente è quasi istantaneo. Inoltre, in caso di necessità, l'intero container può essere facilmente e velocemente eliminato per rimuovere tutte le modifiche apportate in quel test particolare. Questa rapidità è ovviamente presente anche nella gestione di gruppi di container, e permette istanziamenti e rimozioni multiple in tempi molto brevi. Avere questa velocità a disposizione consente di ridurre al minimo i tempi legati alla produzione, alle fasi di test e alla manutenzione delle applicazioni software (e quindi ridurre notevolmente le spese);

2. I software container, non dovendo attivare un loro sistema operativo dedicato e dato il minimo utilizzo di risorse, sono molto più "leggeri" rispetto a molti altri ambienti per lo sviluppo di applicazioni software (per esempio, come si è osservato nel paragrafo 2.5, le macchine virtuali). Riescono facilmente a condividere tra loro i file utilizzati e questo comportamento riduce sensibilmente il numero di copie ridondanti. A questo, si aggiunge la capacità di lavorare sulle immagini senza modificarle (si ricorda che tutte le modifiche vengono scritte solo nel container layer, lo strato più esterno) permettendo un loro uso multiplo e concorrente. Inoltre, dato che si può preparare il container nella maniera più consona alla applicazione o al servizio che si intende eseguire al suo interno, lo sviluppatore è libero di renderlo il più piccolo possibile e questo significa che scriverne di nuovi e buttare via quelli vecchi diventa molto praticabile, riducendo al minimo gli sprechi. Questi aspetti sono fondamentali per le aziende in quanto un minore e migliore uso di risorse significa avere una spesa minore e quindi un maggiore guadagno;

3. Le garanzie di isolamento dei container facilitano la collaborazione tra diversi gruppi di sviluppatori, riducendo al minimo i conflitti e evitando che le problematiche legate ai test di una applicazione si presentino al di fuori dell'ambiente di un container. Questo significa lavorare

sempre con il proprio ambiente pulito e in ordine, costruito secondo le proprie esigenze, senza che fattori esterni interferiscano. L'isolamento dato da un container permette anche una rimozione più pulita della applicazione usata: infatti, l'ambiente utilizzato per l'esecuzione presenta dei confini invalicabili dal punto di vista della applicazione al suo interno. Quindi si capisce facilmente fino a quale punto possono esserci state delle modifiche nel codice e questo concede il via libera ad ogni tipo di test sulla propria applicazione. L'isolamento dato dai container permette ad una azienda di avere innumerevoli ambienti di sviluppo perfettamente separati, consentendo ai diversi gruppi di sviluppatori di lavorare tralasciando completamente le problematiche legate ai conflitti tra questi diversi ambienti;

4. L'idea di sviluppare codice che funzioni ovunque è sempre stata interessante e i container permettono di sfruttare appieno questo vantaggio. L'alta portabilità di questo modello permette di risolvere innumerevoli problematiche legate alla esecuzione di applicazioni in diversi ambienti di lavoro o su host differenti. Gli sviluppatori possono essere certi che il loro codice funzionerà in tutti gli ambienti che utilizzeranno la stessa tipologia di container, indifferentemente dalle caratteristiche hardware e software della macchina sottostante. Molte piattaforme odierne inoltre, sono in grado di utilizzare e convertire diversi formati di container, ampliando ulteriormente questa possibilità. In questa maniera, gli sviluppatori possono concentrarsi sullo sviluppo della applicazione invece che sui problemi di compatibilità tra ambienti diversi. Le interazioni tra diversi gruppi di lavoro sono semplificate in quanto un container può essere replicato, condiviso e utilizzato su altre macchine senza la necessità di avere le stesse tecnologie sullo sfondo. Le aziende fornitrici delle piattaforme per la gestione dei container invece possono focalizzarsi su problematiche legate all'esecuzione dei container, alla gestione di risorse, all'avvio e all'arresto degli ambienti, alla migrazione tra i server, tralasciando completamente i problemi legati al trasferimento di container tra diversi host.

L'utilizzo del modello dei container nel mondo dello sviluppo di software non è limitato solo al caso "avvia un container, avvia l'applicazione al suo interno, lavora sull'applicazione" ma si può applicare molto facilmente ad altre possibilità di lavoro. Per esempio, si possono avviare una serie di

container, comunicanti e interagenti tra loro, per simulare un ambiente distribuito. In questo caso si possono studiare aspetti come l'interazione fra le diverse applicazioni in esecuzione all'interno dei container, i meccanismi utilizzati per la comunicazione tra gli ambienti separati, la sicurezza presente in un certo ambiente di esecuzione, etc. Un altro esempio di utilizzo può essere la simulazione di una applicazione di grandi dimensioni composta da componenti più piccoli, ciascuno dei quali avviato in un container. In questa maniera è molto facile osservare e modificare il comportamento di ciascuna delle sotto parti della applicazione principale.

I container, al giorno d'oggi, non sono utilizzati solo in ambienti per lo sviluppo di software: le loro caratteristiche distintive infatti, ne hanno permesso la diffusione anche in altri campi. Per esempio, sono sempre più numerosi i casi di utilizzo di container per la distribuzione di servizi software. In questo campo, è molto importante gestire agilmente i picchi di connessioni in entrata e i software container eccellono sotto questo aspetto. Per comprenderne meglio il funzionamento, si osservi il il seguente esempio: uno o più server sono costantemente in attesa di richieste da parte dei client; una volta ricevuta una di queste richieste, viene tipicamente creata e assegnata una connessione privata tra il client che l'ha effettuata e il server; per ogni client servito con questa modalità, viene probabilmente avviata una istanza del web server o del servizio fornito e queste istanze possono essere facilmente avviate all'interno di un container. Data la rapidità dei software container, è questione di frazioni di secondo l'avviamento (e il successivo spegnimento) di decine o centinaia di container per la gestione delle connessioni in entrata. Sono molto importanti anche gli aspetti come la rimozione pulita delle istanze create precedentemente, l'assenza di problemi legati alla concorrenza, l'approccio poco invasivo dei container, etc.
La flessibilità nella gestione delle risorse rende i software container ben indicati per gli ambienti in cui il carico di elaborazione varia sensibilmente e in maniera non prevedibile a priori. La maggior parte delle applicazioni in Internet ricade in questa categoria e questo mostra un altro dei motivi che ha portato alla diffusione del modello dei container: in caso si presentino picchi di richieste presso un server, è immediato l'avvio di container supplementari per coprire queste richieste. Inoltre, l'alta mobilità di questo modello permette di spostare facilmente le immagini e i container stessi da una macchina ad un'altra con la minima spesa. In questa maniera si è

in grado di bilanciare al meglio la rete, trasferendo o eliminando eventuali istanze già avviate.

Nel paragrafo 2.6 si sono introdotte alcune delle principali piattaforme per la gestione di container, ciascuna delle quali presenta vantaggi e svantaggi in base al contesto utilizzato. Gli sviluppatori di software, o qualunque altro utente che necessiti di utilizzare i container, ha una vasta gamma di possibilità tra le quali scegliere. Dato che ogni piattaforma presenta determinate caratteristiche, diventa necessario analizzare le necessità dell'utente in maniera da scegliere l'opzione migliore. Per esempio, Docker e RKT utilizzano una astrazione di alto livello con un gran numero di funzionalità. Con queste due piattaforme si possono avviare in maniera molto facile container e applicazioni di alto livello e gestirli con innumerevoli strumenti. RunC e Containerd (un altro progetto open source Docker utilizzato nei motori Docker 1.11 e superiori) al contrario, mantengono un profilo molto più basso e sono orientati alla semplicità e alle performance. Con piattaforme di questo tipo si ha un grande controllo di basso livello al costo però, di un numero inferiore di funzionalità di alto livello e strumenti più complessi per la gestione dei container. Lo sviluppatore dovrà scegliere quale o quali piattaforme utilizzare in base ovviamente, all'obiettivo da lui perseguito.

3.2 Aspetti Avanzati

3.2.1 Architettura a Microservizi e Container

L'architettura a microservizi è una delle soluzioni dominanti moderne per lo sviluppo di applicazioni che necessitano di scalare e evolversi velocemente. La struttura delle applicazioni sviluppate con questa architettura è costituita da tante piccole parti dette microservizi. I microservizi (come si sottointende dal nome), sono dei servizi "piccoli" ed autonomi che interagiscono tra di loro e che hanno come scopo quello di svolgere un singolo compito e di farlo bene. L'insieme di questi microservizi va a comporre l'applicazione finale desiderata.

Nella figura 3.1 si osserva un confronto tra l'architettura a microservizi e una classica architettura di tipo monolitico. Una applicazione monolitica è tipicamente un'unica unità di codice composta da componenti disegnati per lavorare insieme, condividere lo stesso spazio in memoria e le stesse risorse

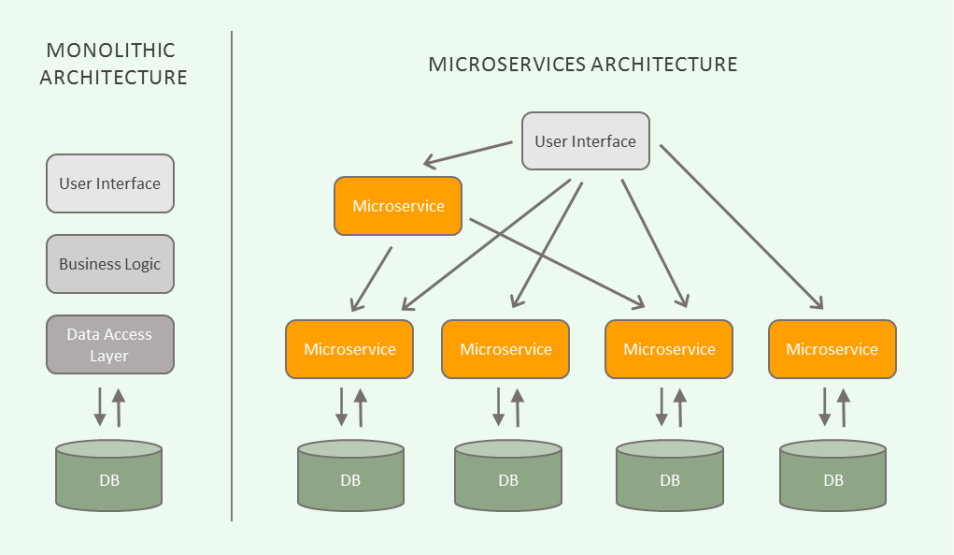

Figura 3.1: Confronto tra architetture.

in generale. Al contrario, come si può osservare anche nell'immagine, una applicazione sviluppata con microservizi è composta da diverse applicazioni più piccole e indipendenti, ciascuna con una propria interfaccia utente e un accesso personale ai database. Questo tipo di architettura porta innumerevoli vantaggi nello sviluppo di applicazioni e si può osservare come questi vantaggi possano essere facilmente legati al modello dei container:

1. La separazione dei microservizi riduce la probabilità che un singolo guasto si rifletta sull'intero sistema. La manutenzione, la modifica o la sostituzione di un singolo servizio è semplificata e non compromette l'intero sistema. Significa avere un alto grado di separazione tra i diversi ambienti di esecuzione e questo aspetto è alla base del modello dei container;

2. Sviluppo, produzione e avvio di microservizi sono operazioni molto semplici e possono essere facilmente automatizzate. Lo stesso discorso è facilmente applicabile ad una piattaforma che prepara, avvia e elimina i container;

3. I microservizi sono facilmente scalabili e altamente mobili e tutte le modifiche necessarie possono essere fatte su richiesta. Anche i contai-

ner sono facilmente intercambiabili e trasferibili con la minima spesa: in caso di necessità, più container possono essere avviati per rispondere ad un picco di lavoro e spostati o eliminati una volta terminato il carico di lavoro;

4. I microservizi devono avere dimensioni ridotte e essere il più leggeri possibile. I container eccellono sotto questi due aspetti grazie alla loro capacità di utilizzare solo le informazioni necessarie e tralasciando tutto il resto.

5. Dato che ogni microservizio viene sviluppato nel modo più opportuno, è normale che vengano utilizzate tecnologie differenti e ambienti di esecuzione diversi. Entrambi questi aspetti non sono difficili da gestire per un ambiente di sviluppo e di esecuzione funzionante con container.

6. Le dimensioni ridotte suggeriscono l'utilizzo di pratiche agili e veloci per lo sviluppo del software. Per esempio, piccoli team di lavoro autonomi che si occupano ognuno di un singolo microservizio. Questa modalità di lavoro è la stessa utilizzata in ambienti di sviluppo facenti uso di container;

Vediamo quindi che sono innumerevoli le similitudini tra il modello dei container e l'architettura a microservizi. Qualunque applicazione sviluppata a microservizi non può che trarre vantaggio dall'uso di container in fase di sviluppo. Una soluzione del genere, cioè una architettura a microservizi sviluppata con container, è ottima in campi sempre alla ricerca di leggerezza e rapidità. Per esempio, l'erogazione di servizi attraverso il cloud computing e lo stesso web, sono campi che possono sfruttare appieno i vantaggi ottenuti combinando microservizi e container. Per quanto riguarda il cloud sono già presenti diverse Platform-as-a-Service (ad esempio OpenShift, CloudFoundry) e altrettante Infrastructure-as-a-Service (OpenStack e CloudStack) facenti uso del modello dei container e in grado di sviluppare architetture a microservizi. Le stesse piattaforme per la gestione dei container possono ovviamente essere distribuite attraverso il cloud come PaaS, o meglio, come *Container as a Service*.
Una architettura a microservizi è anche un'ottima soluzione per compensare i problemi legati alla tendenza odierna a produrre complesse applicazioni software di grosse dimensioni: questo tipo di applicazioni sono troppo

spesso sviluppate come un singolo blocco di codice in pieno stile monolitico. Utilizzando una architettura a microservizi sviluppata con container, la produzione di questi programmi software potrebbe sfruttare tutti i vantaggi legati a questi due modelli. In questa maniera, nonostante le dimensioni dell'applicazione nel suo complesso, si avrebbe una struttura costituita da tanti piccoli moduli indipendenti, facilmente gestibili e sostituibili.

Una architettura sviluppata a microservizi presenta anche degli svantaggi. Il problema principale presente sia in ambiente di sviluppo, sia nella fase di distribuzione, è legato al numero di microservizi in funzione in un sistema. Osservando la struttura di una applicazione sviluppata in questo modo, bisogna prestare molta attenzione alla eccessiva complessità: *divide et impera* è il concetto alla base del modello e abbiamo visto che consiste nello scomporre il più possibile una applicazione nelle sue sottoparti. Questo comporta un aumento considerevole del numero di componenti e della complessità dell'intero sistema. Inoltre tutte queste singole parti devono essere indipendenti e in grado di interagire tra di loro. La complessità di un sistema di questo tipo quindi, aumenta in maniera esponenziale e potrebbe portare ad un sovraccarico delle linee di comunicazione. Utilizzare il modello dei container peggiora ulteriormente la situazione perché presenta lo stesso tipo di problema: in particolare, si è visto che un elevato numero di container in esecuzione su una macchina può portare a perdita di performance dell'intero sistema e a problemi di gestione dei singoli ambienti di esecuzione. D'altronde, l'obiettivo iniziale del modello era la creazione di un ambiente isolato e solo successivamente si è sfruttata la capacità di proliferazione dei container. Questi problemi legati ai cluster di container sono uno dei campi più studiati e sviluppati attualmente. Diverse aziende ci stanno lavorando proprio perché la tendenza odierna porta allo sviluppo di gruppi numerosi di container e diventa necessario introdurre politiche per la gestione di questi cluster. Uno dei prodotti di maggior successo ottenuto finora da questi studi è Kubernetes.

3.2.2 Kubernetes

Kubernetes è un altro sistema open source per la gestione e lo sviluppo di container che permette di risolvere facilmente i problemi presentati precedentemente: cluster e ridondanza di container con conseguente calo di

prestazioni. Kubernetes è nato da una collaborazione tra Google e RedHat con lo scopo di eliminare la maggior parte dei processi manuali coinvolti nel deployment e nella manutenzione di sistemi funzionanti con container. Gli ottimi risultati ottenuti hanno portato alla diffusione del progetto e all'incremento di aziende partecipanti.

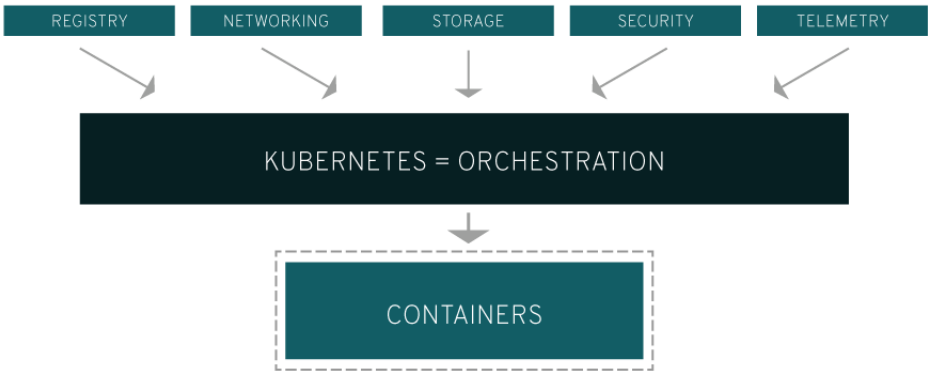

Figura 3.2: Struttura di Kubernetes

La differenza fondamentale di Kubernetes rispetto alle altre piattaforme, come si vede anche nella figura 3.2, è l'introduzione di un ulteriore livello di virtualizzazione al di sopra dei container. Questo livello extra permette agli sviluppatori di anticipare e programmare i carichi di lavoro, e fornisce i servizi necessari, tra cui rete e memoria, ai container stessi. Questa forma di virtualizzazione è realizzata tramite l'uso delle unità di base di Kubernets, i pod. Un pod consiste in un gruppo di uno o più container in esecuzione su un singolo nodo (una macchina fisica o virtuale). Tutti i container all'interno di un pod condividono le stesse risorse di rete, (indirizzo IP, nome dell'host, etc), risorse fisiche come memoria e capacità di elaborazione e presentano file di informazione per l'utilizzo dei container al suo interno. Come i container, i pod astraggono dalle risorse sottostanti e sono quindi facilmente gestibili e trasferibili. L'intero sistema è gestito con una struttura master-slave, dove il master ovviamente, è il punto di origine di tutti i processi e unico gestore dei nodi e dei pod.

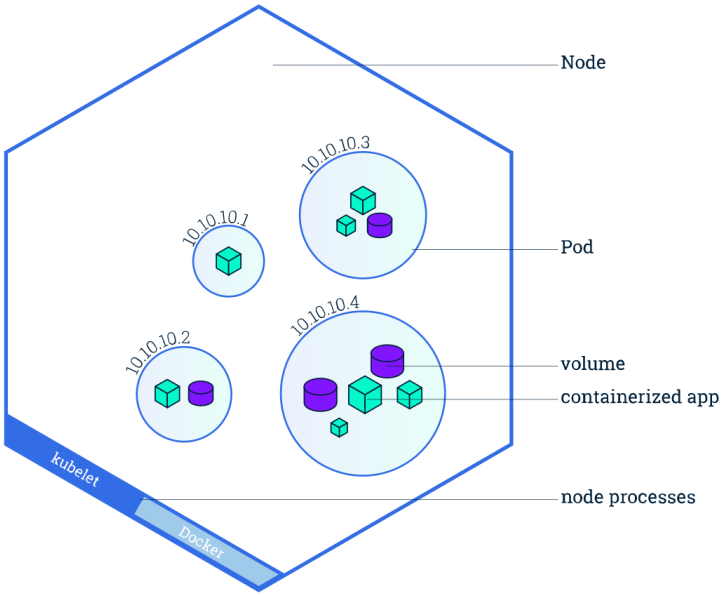

Figura 3.3: Esempi di pod all'interno di un nodo[14]

Come si può osservare nella figura 3.3, ogni nodo deve avere in esecuzione sempre almeno due processi:

1. Kubelet, il processo responsabile della comunicazione tra il master e il nodo;

2. un processo per il runtime dei container (come Docker, rkt), il quale è responsabile della esecuzione classica dei container.

Kubernetes non è un programma a sé stante ma collabora con numerosi altri progetti open source per il suo funzionamento. Il suo scopo principale è lo sviluppo di una gestione completamente automatizzata dei sistemi di container. Lo sviluppatore di software si può interfacciare attraverso una API al master e programmare le innumerevoli funzionalità di Kubernetes. Per esempio si possono settare i parametri legati al deployment delle applicazioni, oppure cambiare le modalità di aggiornamento dei container o lavorare con i servizi per la gestione dei load-balancing (per esempio Ingress o LoadBalancer). Quest'ultimo aspetto è uno dei più importanti in verità: se un programmatore utilizza Kubernetes, sicuramente ha intenzione di lavorare con cluster di container; in questo caso è molto probabile che

si presentino problematiche legate alla distribuzione dei carichi di lavoro. Kubernetes ha due modalità per la gestione di questi carichi:

1. si possono prevedere alcuni di questi carichi e gestirli al momento del deployment dei container e dei pod, suddividendo le risorse a disposizione in maniera da evitare eventuali colli di bottiglia. Questa modalità è molto facile da implementare grazie agli strumenti di Kubernetes ma non si possono prevedere perfettamente eventuali picchi di lavoro futuri;

2. controllo del traffico in real-time. Il servizio principale è Ingress, il quale opera attraverso un pod specializzato. Include un insieme di regole per governare il traffico e un demone per l'applicazione di queste regole. L'intero set di regole è completamente accessbile e personalizzabile da un programmatore.

Kubernetes è sicuramente l'applicazione di questo tipo più utilizzata ma altre soluzioni si sono diffuse precedentemente o sono state sviluppate recentemente: per esempio Swarm di Docker e Amazon Container Service di Amazon. Questo dimostra la tendenza a sviluppare sistemi composti da un elevato numero di container e la costante necessità di avere a disposizione funzionalità per la loro gestione. Kubernetes rimane la piattaforma con più potenzialità, in costante miglioramento. Lo stesso Docker, come tante altre importanti piattaforme, ha integrato Kubernetes nel proprio codice.

Capitolo 4

Conclusioni

I software container stanno lentamente cambiando il modo di sviluppare, distribuire e eseguire software e il loro utilizzo continua ad aumentare in innumerevoli settori, dalle più piccole aziende alle grandi imprese. Questa evoluzione è in atto da diversi anni e per molti aspetti non è ancora terminata. Visti i continui investimenti e sviluppi di aziende come Google, Microsoft e Redhat, si può facilmente immaginare le grandi potenzialità di questo modello e le alte aspettative per il futuro.

Osservando il capitolo precedente, si capisce come i container siano profondamente legati a diverse tematiche odierne come i microservizi, i cluster di applicazioni e il cloud. Per questo motivo gli sviluppatori e gli ingegneri operativi dovrebbero aspettarsi di utilizzare regolarmente container entro i prossimi anni. In futuro, potrebbero diventare la principale tecnologia utilizzata per la virtualizzazione, cooperando con macchine virtuali o addirittura sostituendole. Si deve osservare però che macchine virtuali e container non sono due opzioni antitetiche. Gli svantaggi di un modello possono essere compensati dai vantaggi dell'altro. Le macchine virtuali hanno un grado di isolamento superiore (dato dal hypervisor) e sono una tecnologia affidabile e abbondantemente verificata. Al contrario, i container sono relativamente nuovi e molte organizzazioni sono riluttanti a fidarsi completamente delle caratteristiche di isolamento di un nuovo modello prima che abbiano una comprovata esperienza. Per questo motivo, è comune trovare sistemi ibridi con container eseguiti all'interno di macchine virtuali per sfruttare entrambe le tecnologie. Utilizzati insieme offrono una grande

flessibilità nella distribuzione e nella gestione delle applicazioni portando i vantaggi di entrambi i modelli ed eliminando quasi tutti gli svantaggi.

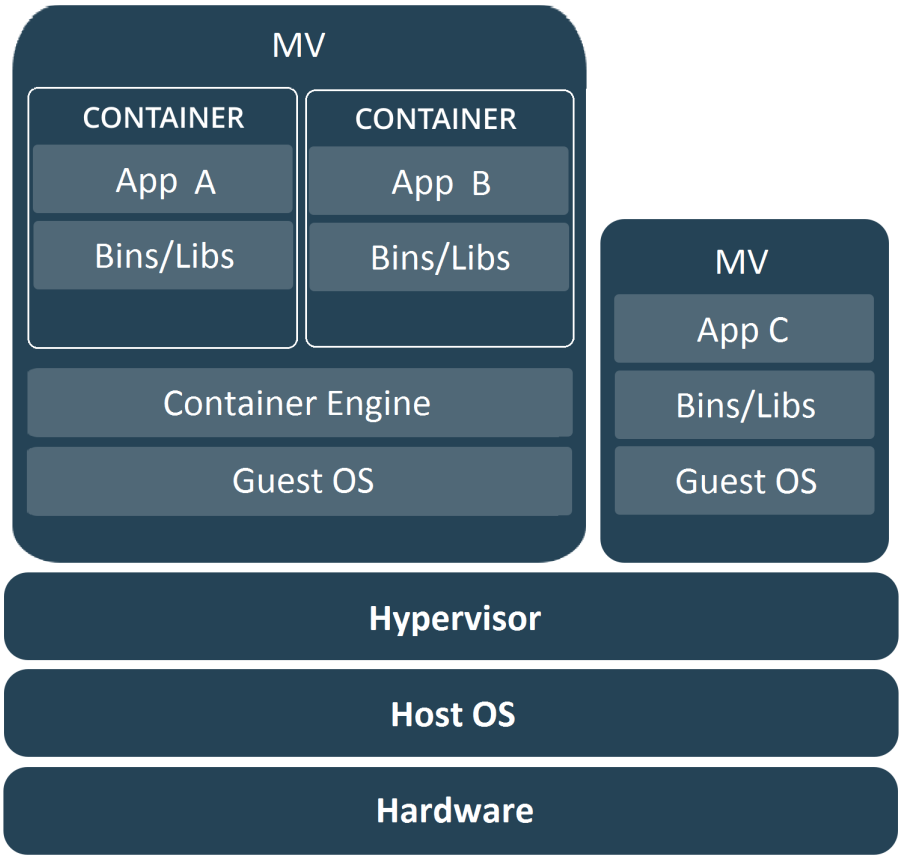

Figura 4.1: Esempio di sistema con container funzionanti all'interno di macchine virtuali.

Nella figura soprastante si può osservare un esempio di sistema utilizzante entrambe le tecnologie: sulla macchina host è installato un hypervisor gestore di macchine virtuali; l'applicazione C viene eseguita nel modo classico all'interno di una macchina virtuale; le applicazioni A e B sono invece eseguite all'interno di due container, controllati da un gestore di container installato in un'altra macchina virtuale; si potrebbe avere anche un terzo caso dato dalla presenza all'interno della stessa macchina virtuale sia di ge-

stori di container, sia di applicazioni libere. Con questo tipo di architettura si ottengono innumerevoli vantaggi:

- i problemi di sicurezza legati ai container sono completamente risolti grazie alla presenza del hypervisor che protegge la macchina sottostante da qualunque tipo di attacco;
- il sistema ha un elevato grado di stabilità perché riesce a utilizzare la robustezza di entrambi modelli;
- data la presenza delle macchine virtuali, si possono avere innumerevoli ambienti di lavoro differenti e questo significa che possono essere presenti anche più formati di container in funzione e diverse piattaforme per la loro gestione, ciascuna installata sulla propria macchina virtuale;
- tutte le caratteristiche dei container come rapidità, flessibilità e semplicità, sono ancora presenti e applicabili all'interno di ogni macchina virtuale;

Gli svantaggi al contrario, sono ridotti al minimo. Tra questi è ancora presente la pesantezza delle macchine virtuali. Al giorno d'oggi però, sta diventando un problema secondario grazie alla presenza di nuove tecnologie e tecniche che migliorano costantemente la gestione di risorse. Un sistema facente uso di entrambi i modelli è un ottimo strumento per introdurre nuove aziende all'uso dei container. Infatti, le aziende di oggi sono costantemente sotto pressione per la trasformazione digitale e per la "necessità" di rinnovare continuamente il software da loro utilizzato; sono però limitate dalle applicazioni e dalle infrastrutture utilizzate. Inoltre, il passaggio da una tecnologia ad un'altra potrebbe portare via diverso tempo e avere un certo costo. Piattaforme come Docker consentono una vera indipendenza tra applicazioni e infrastruttura permettendo passaggi molto semplici e veloci tra diversi tipi di software, senza il bisogno di modificare il sistema attualmente in uso. Il principale ostacolo all'adozione dei container è legato al fatto che si tratta di una tecnologia relativamente giovane e quindi non proprio verificata sotto ogni tipo di aspetto (come si è visto nei precedenti capitoli, sono ancora presenti problematiche legate alla sicurezza). L'utilizzo combinato di container e macchine virtuali elimina completamente questo difetto

predisponendo le aziende alla facile introduzione dei software container.

Docker, Rkt, Kubernetes sono tutti progetti open source di notevole successo nella storia recente. La presenza di innumerevoli piattaforme per lo sviluppo di software container e la loro continua crescita ci dimostra che in campo ingegneristico sono ampiamente utilizzate. Il modello dei container sta fondamentalmente spostando il modo in cui le persone pensano a costruire, spedire e gestire le applicazioni. Questo cambiamento rende più facile la creazione di applicazioni a microservizi, la collaborazione per lo sviluppo di codice open source, interazioni tra diversi sviluppatori, etc. I container stanno modificando sia il ciclo di vita dello sviluppo dell'applicazione, sia le pratiche di ingegneria legate ad esso.

Bibliografia

[1] Marshall Kirk Mckusick, *Sito di documentazione di FreeBSD* ,https://docs.freebsd.org/44doc/papers/jail/jail-9.html

[2] Bill Cheswick, *An Evening with Berferd In Which a Cracker is Lured, Endured, and Studied* , http://www.cheswick.com/ches/papers/berferd.pdf, 1991

[3] Poul-Henning Kamp, *Sito di documentazione di FreeBSD* https://www.freebsd.org/doc/it_IT.ISO8859-15/books/handbook/jails-intro.html

[4] *Sito ufficiale di Virtuozzo* https://virtuozzo.com/open-source/

[5] Jacques Gélinas, *Sito ufficiale di Linux VServer* http://linux-vserver.org/Overview#History

[6] *Sito di documentazione di container Linux* https://linuxcontainers.org/it/lxc/manpages/man5/lxc.container.conf.5.html

[7] *Containers in OpenStack*, capitolo 1, pagina 9, Editore:Packt Publishing, Autore: Pradeep Kumar Singh e Madhuri Kumari, Madhuri Kumari, 2017

[8] *Sito ufficiale di Docker* , https://www.docker.com/company

[9] *Sito ufficiale di OCI* , https://www.opencontainers.org/

[10] *Sito ufficiale di Windows Server*, https://www.windowserver.it/2015/11/windows-server-2016-introduzione-ai-container/

[11] *Sito di documentazione di Docker*, `https://docs.docker.com/storage/storagedriver/`, 2018

[12] *Sito di documentazione di Docker*, `https://docs.docker.com/engine/docker-overview/#docker-engine`, 2018

[13] Daniel J Walsh, *Sito ufficiale di OpenSource.com* , `https://opensource.com/business/14/7/docker-security-selinux`, 2014

[14] *Sito ufficiale di Kubernetes*, `https://kubernetes.io/docs/tutorials/kubernetes-basics/explore-intro/`, 2018