

Luca Cabibbo
Architettura
dei Sistemi
Software

Container e virtualizzazione basata su container

dispensa asw660
ottobre 2024

Welcome to the container revolution.
Bret Fisher

1

Container e virtualizzazione basata su container

Luca Cabibbo ASW



- Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
 - Capitolo 39, **Container e virtualizzazione basata su container**
- ❑ LXC (Linux Containers) <https://linuxcontainers.org/>
- ❑ Docker <https://www.docker.com/>
- ❑ Siti web di diversi fornitori di servizi per container nel cloud
 - <https://aws.amazon.com/containers/>
 - <https://cloud.google.com/containers/>
- ❑ Velichko, I. **Learning Containers From The Bottom Up: Efficient Learning Path to Grasp Containers Fundamentals**. 2021.
 - <https://iximiuz.com/en/posts/container-learning-path/>
- ❑ Richardson, C. **Microservices Patterns: With examples in Java**. Manning, 2019.

2

Container e virtualizzazione basata su container

Luca Cabibbo ASW



- Obiettivi e argomenti

□ Obiettivi

- introdurre i container e la virtualizzazione basata su container
- confrontare container e macchine virtuali
- discutere i container come opzione per il rilascio del software

□ Argomenti

- introduzione
- richiami di nozioni preliminari
- virtualizzazione basata su container
- container
- tecniche per la virtualizzazione basata su container
- container e macchine virtuali a confronto
- container e rilascio del software
- discussione



* Introduzione

□ Ci sono due modi per presentare i container e la virtualizzazione basata su container

- da un punto di vista tecnico, come una variante delle VM e della virtualizzazione di sistema
- in riferimento all'uso che se ne può fare

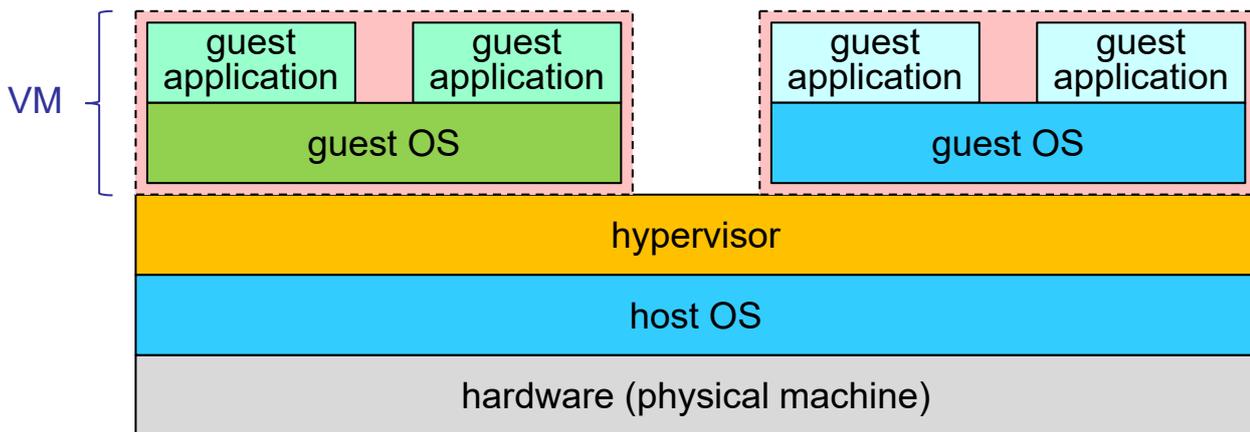
□ Consideriamo entrambi i punti di vista

- gli aspetti tecnici, per fare un confronto tra VM e container
- l'utilizzo dei container, che sono comunque l'aspetto più rilevante per l'architettura del software
- alla fine, discutiamo l'uso dei container nel contesto del rilascio del software



* Richiami di nozioni preliminari

- La virtualizzazione di sistema, basata su un hypervisor, offre l'astrazione delle macchine virtuali
 - una VM è un computer virtuale – ciò che viene virtualizzato è l'hardware di un computer
 - in ciascuna VM è poi possibile installare un OS completo ed eseguire applicazioni e servizi



5

Container e virtualizzazione basata su container

Luca Cabibbo ASW



Nozioni preliminari

- La virtualizzazione di sistema
 - offre diversi benefici
 - fornisce flessibilità operativa
 - sostiene l'isolamento tra le VM, ciascuna con i propri servizi e applicazioni
 - per questo, ha numerose applicazioni
 - può però introdurre un overhead elevato
 - in particolare, nella gestione dell'I/O
 - inoltre, un host che ospita N macchine virtuali deve occuparsi della gestione e dell'esecuzione di N (istanze di) sistemi operativi completi

6

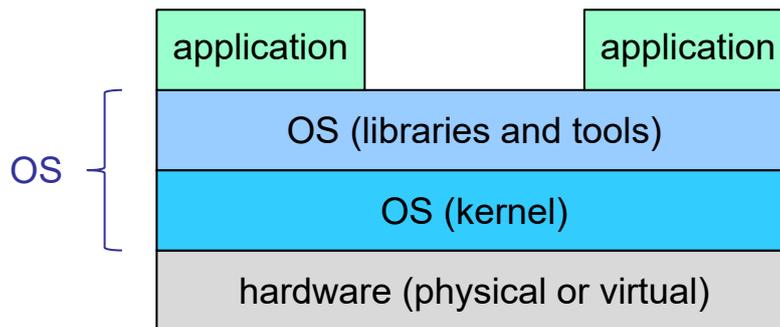
Container e virtualizzazione basata su container

Luca Cabibbo ASW



Nozioni preliminari

- Un **sistema operativo (OS)** è composto da diversi elementi software
 - il **kernel** – che gestisce alcune responsabilità critiche dell'OS
 - un insieme di **librerie** e **strumenti** e degli ulteriori programmi di utilità – che operano sopra al kernel

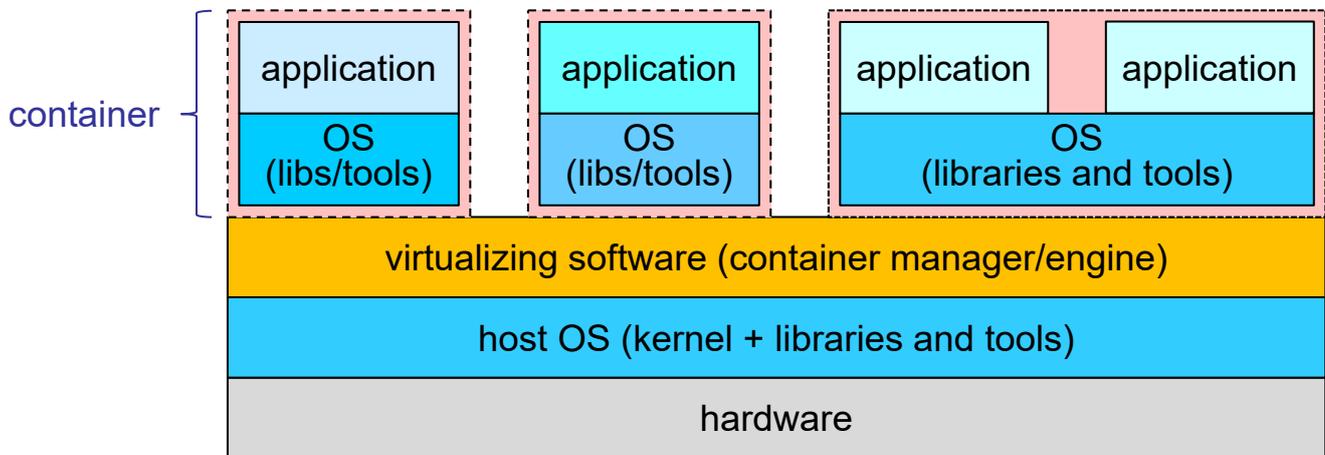


* Virtualizzazione basata su container

- La **virtualizzazione basata su container** (*container-based virtualization*) – chiamata anche *OS-level virtualization*
 - fornisce l'astrazione dei **container** (“contenitori”) – chiamati anche *lightweight container*
 - un container è un’“entità virtuale” che comprende l’hardware di un computer insieme al kernel di un OS
 - un container è simile a una VM, ma virtualizza il sistema operativo anziché l’hardware di un computer
 - in pratica, il kernel di ogni container corrisponde al kernel dell’OS host
 - in ciascun container è poi possibile installare un OS (librerie e strumenti) ed eseguire applicazioni e servizi



Virtualizzazione basata su container



9

Container e virtualizzazione basata su container

Luca Cabibbo ASW



Implementazione

- Alcune considerazioni sulla virtualizzazione basata su container
 - è diffusa soprattutto nel mondo Unix/Linux
 - è una forma di virtualizzazione leggera, che è supportata direttamente dal kernel dell'OS host
 - non richiede un hypervisor sull'host
 - non usa tecniche di emulazione dell'hardware e non richiede il supporto alla virtualizzazione dell'hardware
 - il software di virtualizzazione per container (*container manager* o *container engine*) consente di definire un container come un insieme di processi e di altre risorse
 - ad es., un container che esegue un servizio Java consiste essenzialmente di un processo JVM (eseguito nell'host) – più tutto ciò che serve per questo processo JVM

10

Container e virtualizzazione basata su container

Luca Cabibbo ASW



Implementazione

- Alcune considerazioni sulla virtualizzazione basata su container
 - nel sistema host viene eseguito un solo kernel condiviso
 - questo kernel gestisce sia le risorse del sistema host che quelle dei container in esecuzione
 - il kernel è condiviso dai container
 - ma ogni container può eseguire un proprio OS ed ha delle risorse “virtuali” proprie – ad es., ha un proprio file system completo e un proprio network stack, con un proprio indirizzo IP



Implementazione

- Alcune considerazioni sulla virtualizzazione basata su container
 - il kernel condiviso gestisce tutti i processi e tutte le altre risorse (ad es., i file), sia dell’host che dei container
 - i processi di un container sono gestiti come processi dell’host
 - il file system di un container viene gestito come sotto-albero del file system dell’host
 - ad es., a partire da
`/var/lib/docker/containers/containerid/filesystem/`
 - l’accesso a un file in un container viene realizzato come l’accesso a un file nel file system dell’host



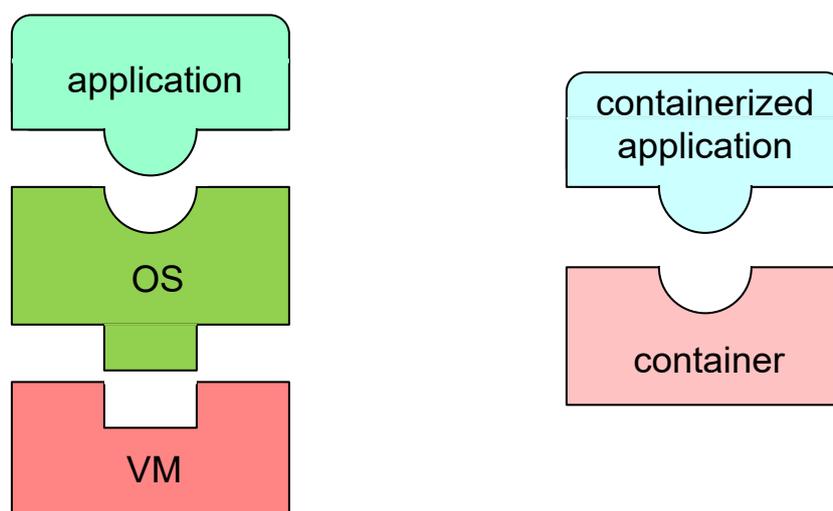
Implementazione

- Alcune considerazioni sulla virtualizzazione basata su container
 - il software di virtualizzazione per container si occupa anche di garantire l'isolamento tra i diversi container – e tra i container e l'host
 - ad es., un container non può interferire con i processi di un altro container o accedervi al file system – e non può nemmeno accedere direttamente alle risorse dell'host
 - tuttavia, i container potrebbero non essere completamente isolati dall'host



Container e interfaccia

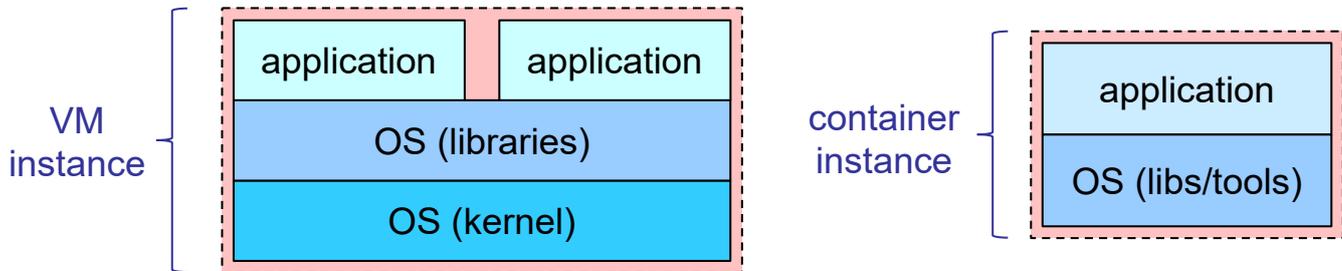
- In termini di interfaccia, confrontando VM e container
 - l'interfaccia esposta da una VM è quella dell'hardware di un computer
 - l'interfaccia esposta da un container è quella del kernel di un OS – l'interfaccia delle chiamate di sistema del kernel





Container e istanze

- Un'*istanza di VM* comprende anche l'OS e le applicazioni che vi sono installate
 - in modo analogo, un'*istanza di container* comprende anche le librerie e gli strumenti e le applicazioni che vi sono installate



- il termine container viene spesso utilizzato anche per indicare un'istanza di container



Container e immagini

- Un'*immagine di VM* comprende il contenuto di una VM – per facilitare la creazione di una o più istanze di VM a partire da quell'immagine
 - in modo analogo, un'*immagine di container* consiste nell'immagine del file system di un container
 - comprende una o più applicazioni da eseguire nel container – insieme alle librerie e gli strumenti e a tutto il software necessario per eseguire quelle applicazioni
 - in modo che sia possibile creare facilmente una o più istanze di container a partire da quell'immagine
 - un'immagine di container diventa un'istanza di container a runtime, quando viene eseguita in un container manager/engine



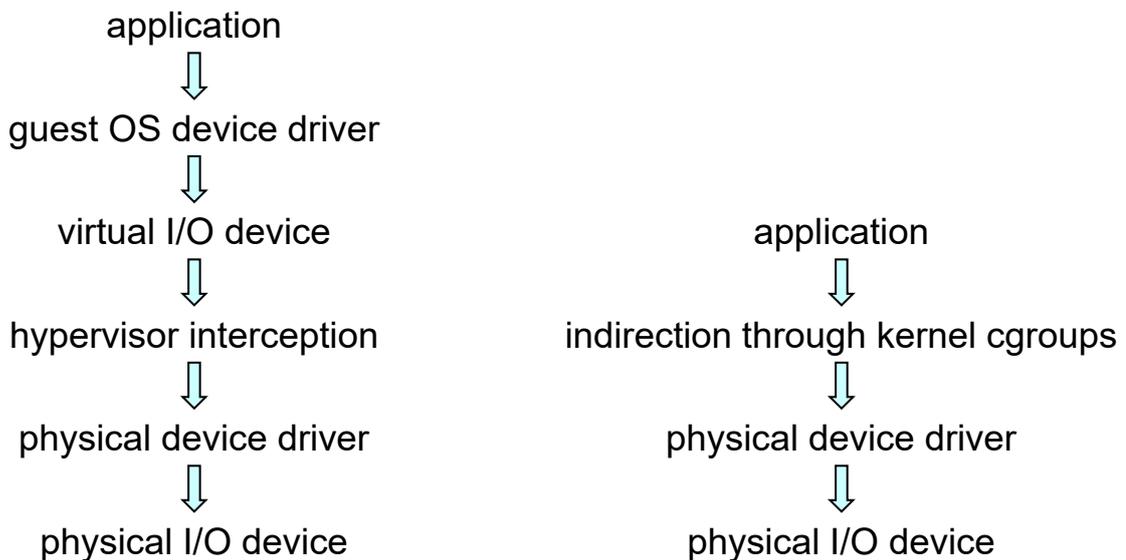
Discussione

- Un confronto preliminare tra container e VM
 - i container introducono un overhead minore rispetto alle VM (sono “più leggeri”)
 - le prestazioni sono quasi native – non viene utilizzata né la virtualizzazione del processore né la virtualizzazione dell’I/O né un hypervisor



Discussione

- Un confronto preliminare tra container e VM
 - i container introducono un overhead minore rispetto alle VM (sono “più leggeri”)
 - esempio: gestione di un’operazione di I/O tramite hypervisor e container





Discussione

- Un confronto preliminare tra container e VM
 - i container introducono un overhead minore rispetto alle VM (sono “più leggeri”)
 - le prestazioni sono quasi native – non viene utilizzata né la virtualizzazione del processore né la virtualizzazione dell’I/O né un hypervisor
 - i container offrono però una flessibilità operativa minore rispetto alle VM
 - l’OS di un container deve essere compatibile con il kernel eseguito nell’host
 - i container offrono anche un isolamento minore rispetto alle VM



* Container

- Consideriamo ora i container dal punto di vista del loro utilizzo
- Ogni container (istanza di container) viene in genere utilizzato per eseguire un servizio software o un’applicazione specifica (anche se talvolta più di uno/una)
 - il container viene dunque usato per realizzare l’ambiente di esecuzione virtuale richiesto da quello specifico servizio software
 - il container “contiene” tutto ciò che serve per eseguire questo servizio software – il servizio è “contenitorizzato”



Container

- Un **container** è un'unità di software standardizzata, che impacchetta una o più applicazioni software, insieme alle loro configurazioni e dipendenze – in modo tale che queste applicazioni possano essere eseguite in modo veloce e affidabile in un opportuno ambiente di esecuzione per container



Container

- Un container è un'unità di software standardizzata, che impacchetta una o più applicazioni software, insieme alle loro configurazioni e dipendenze
 - il container ha lo scopo di fornire al software applicativo di interesse un ambiente di esecuzione completo e autonomo, con tutte le dipendenze necessarie – senza nessuna dipendenza verso l'host
 - queste dipendenze comprendono in genere le librerie e gli strumenti dell'OS, le librerie runtime richieste dai linguaggi usati e il middleware
 - le dipendenze specifiche del software applicativo di interesse vengono installate e configurate nel container (anziché nell'host) – insieme al codice del software applicativo



Container

- Un container è un'unità di software standardizzata, che impacchetta una o più applicazioni software, insieme alle loro configurazioni e dipendenze
 - il container è anche un ambiente standardizzato di esecuzione per la sua applicazione (o applicazioni)
 - ciascuna applicazione (nel proprio container) può così essere rilasciata ed eseguita in modo coerente in una varietà di piattaforme, sia nel proprio computer che on premises che nel cloud
 - esistono diversi formati standard per container – quello più popolare oggi è Docker



Tipi di container

- Una classificazione dei container, in relazione al loro utilizzo
 - un *OS container* è un container pensato per essere usato come una VM leggera – con un proprio OS, in cui eseguire più applicazioni o servizi
 - un *application container* è un container pensato per contenere ed eseguire una singola applicazione o servizio
 - ci concentriamo soprattutto sugli application container



Application container

- Gli application container consentono di focalizzare ciascun container su una singola applicazione o servizio
 - il container deve contenere solo le dipendenze per quello specifico servizio software
 - questo riduce il rischio di inconsistenze nello stack software
 - inoltre, i singoli container sono più leggeri – possono essere creati e avviati più velocemente, e a runtime usano solo le risorse richieste per il loro specifico servizio
 - questo sostiene disponibilità, scalabilità e modificabilità
 - i container sono isolati tra di loro
 - questo sostiene affidabilità e sicurezza
 - dal punto di vista di un'applicazione o servizio in esecuzione in un container, è come se l'applicazione o il servizio venisse eseguito in un proprio nodo – con un proprio indirizzo IP e un proprio file system



* Tecniche per la virtualizzazione basata su container

- Esistono diverse tecnologie per container, nel contesto dei sistemi operativi UNIX e Linux
 - ad es., LXC, OpenVZ per Linux, Solaris Containers per Solaris, FreeBSD jail per FreeBSD e Docker
 - descriviamo alcune tecniche di virtualizzazione per container



- Supporto per i container in Linux

- Alcuni intuizioni sui container *LXC* (*Linux Containers*, 2008) – la prima implementazione completa di container per Linux
 - nel kernel Linux, ogni processo può generare altri processi, in modo gerarchico
 - un container è un sottoalbero dell'albero dei processi del sistema – a cui sono associate delle risorse (come CPU, memoria e disco) e che viene mantenuto isolato dagli altri container (con le loro risorse)
 - il kernel di un container è quello dell'host – ma l'OS di un container può essere diverso dall'OS dell'host

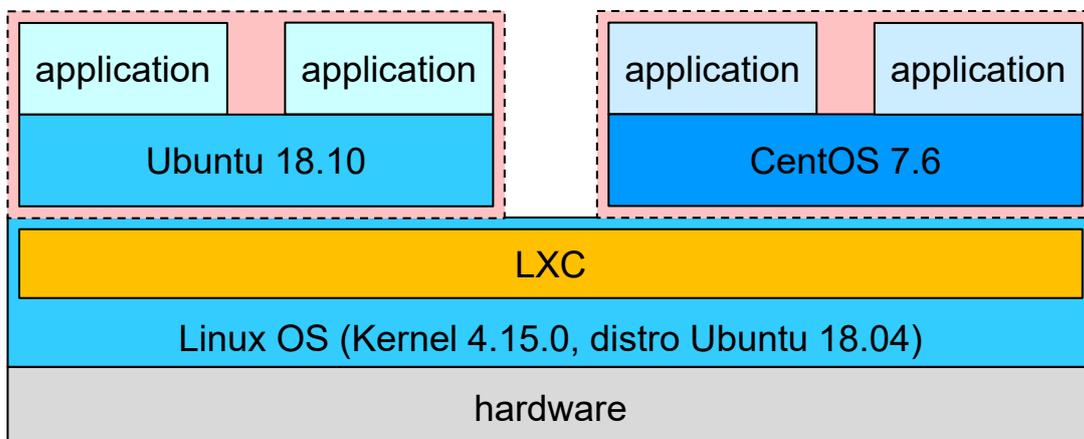
27

Container e virtualizzazione basata su container

Luca Cabibbo ASW



Container LXC



28

Container e virtualizzazione basata su container

Luca Cabibbo ASW



Container LXC

- In pratica, *LXC* (*Linux Containers*) è un insieme di strumenti semplici – ma basati su un'API potente – per creare e gestire container in un host Linux (reale o virtuale)
 - LXC consente di creare ed eseguire uno o più container – con le loro applicazioni
 - i container sono isolati tra di loro e dall'OS host, e si comportano come macchine indipendenti
 - i container LXC sono simili per funzionalità alle VM
 - ma sono controllati direttamente dal kernel dell'OS host, senza la necessità di un hypervisor
 - LXC fornisce l'astrazione dei container utilizzando e combinando alcune funzionalità del kernel Linux – in particolare
 - i control group – per controllare l'uso delle risorse
 - i namespace – per controllare la visibilità delle risorse



Control group

- Un *control group* (*cgroup*) è
 - un gruppo di processi – che ha radice in un certo processo e comprende tutti i suoi figli (e discendenti) correnti e futuri
 - a cui sono associati dei parametri e/o dei limiti nell'uso di risorse (come CPU, memoria, rete, file system, ...)
 - i cgroup consentono di isolare, limitare e misurare l'uso di risorse assegnate a un gruppo di processi



Namespace

- Un *namespace* rappresenta una collezione autocontenuta di risorse a cui sono dati dei nomi virtuali, che vengono poi mappati su delle risorse reali
 - esempi di risorse sono gli id dei processi e degli utenti, le risorse di rete, il nome dell'host e le sue porte, i file nel file system
 - i namespace consentono di disaccoppiare un gruppo di processi dalle risorse reali che gli verranno assegnate – per controllare la visibilità delle risorse e per evitare conflitti nei nomi e inconsistenze nei riferimenti
 - in pratica, i namespace consentono di separare le risorse di gruppi di processi differenti



Discussione

- Un container LXC offre un ambiente di esecuzione simile a una distribuzione Linux standard, con un certo livello di controllo e di isolamento delle risorse
 - sono in genere utilizzati come “OS container” – per eseguire anche più applicazioni o servizi
- L'utilizzo di LXC avviene mediante degli strumenti per la gestione dei container
 - che sono basati su un'API potente – ma che non è semplice da utilizzare



- Sul software di virtualizzazione per container



- Dopo aver visto l'esempio dei container LXC, è utile astrarre un po' per comprendere meglio il funzionamento tecnologico dei container e del software di virtualizzazione per container



Container e container runtime



- Da un punto di vista tecnico, è possibile dire che
 - un **container** è un insieme di processi limitato (cgroup) e isolato (namespace)
 - i container non sono macchine virtuali!
 - per avviare un container non è sufficiente avviare i suoi processi – bisogna prima creare e configurare le sue risorse (cgroup e namespace) – ovvero, bisogna anche preparare un “box” in cui eseguire questi processi
 - la creazione di questi “box” viene svolta da un **container runtime** – un software di virtualizzazione per container, di livello piuttosto basso
 - un esempio di container runtime è la libreria **runc**



Container e container runtime



- Un esempio di container runtime è la libreria *runc*
 - in pratica, *runc* può essere usato come un normale strumento dalla linea di comando – senza usare nessuno altro strumento di virtualizzazione per container di alto livello (come LXC o Docker)
 - per avviare un container con *runc* è richiesto un “bundle” (“pacchetto”) contenente gli eseguibili e i parametri del container
 - questo bundle può essere minimale – non deve necessariamente includere le librerie complete di un OS
 - dunque, per eseguire un container non è necessaria un’immagine completa di container
 - tuttavia, è comune organizzare un tale bundle come un file system che ricorda la struttura di una distribuzione Linux tipica – ovvero, come un’immagine di container



Container e container manager



- Un container runtime ha l’obiettivo di gestire il ciclo di vita di un singolo container – tuttavia, è comune eseguire in un singolo host decine o centinaia di container
 - un *container manager* è un software di virtualizzazione per container, di livello più alto, per gestire l’esecuzione di molti container e la loro coesistenza in un singolo host
 - un esempio di container manager è la libreria *containerd*
 - le responsabilità di un container manager includono, ad es., la gestione dello storage e delle reti utilizzate dai container, la gestione dei log, nonché un supporto alla gestione delle loro immagini
 - in pratica, anche *containerd* può essere usato come un normale strumento dalla linea di comando



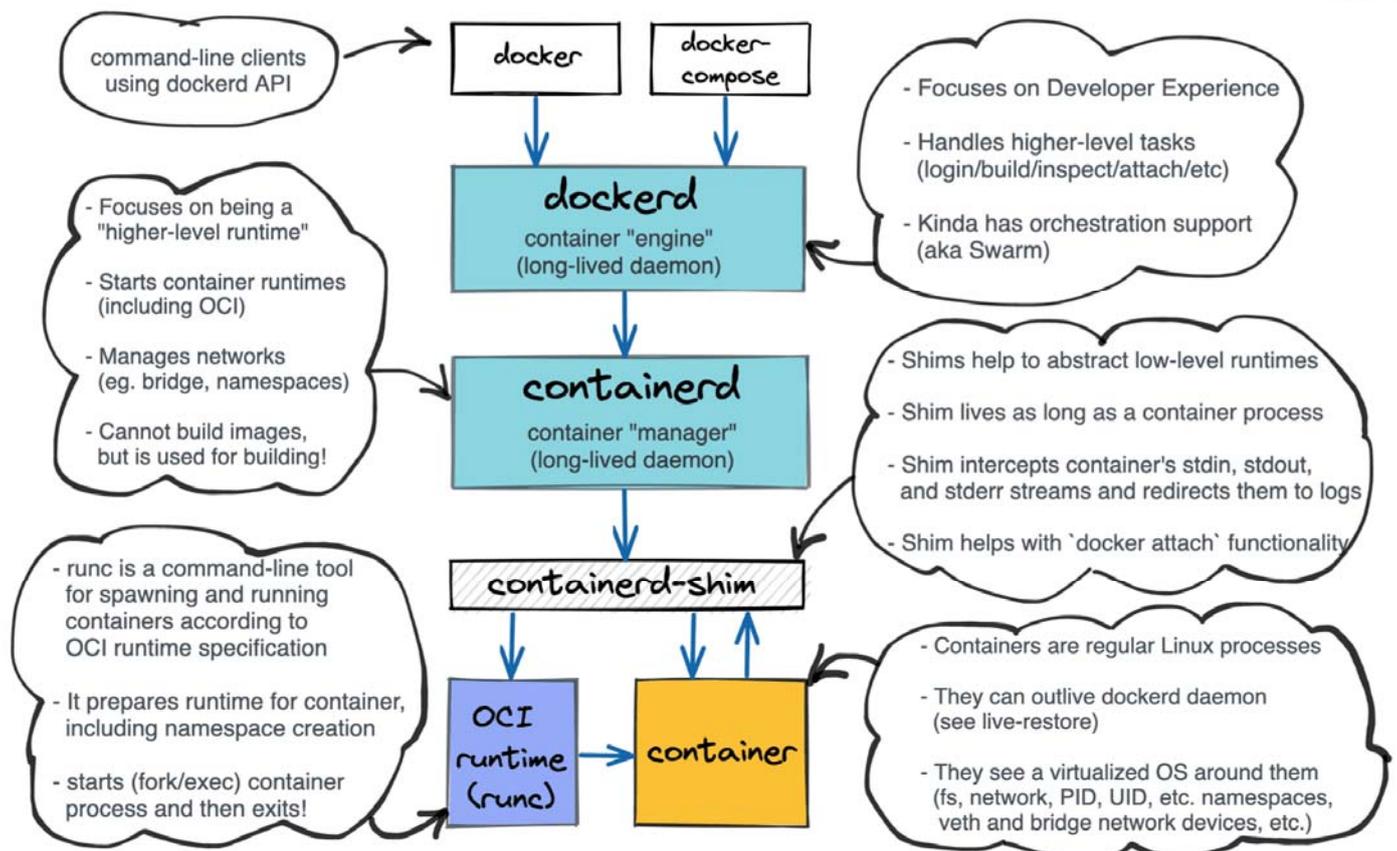
Container e container engine



- Inoltre, un **container engine** è un software di virtualizzazione per container, di livello ancora più alto, che ha lo scopo di semplificare per lo sviluppatore l'esperienza di uso dei container
 - esempi di container engine sono **Docker** (<https://www.docker.com/>) e **Podman** (<https://podman.io/>)



Container runtime, manager e engine



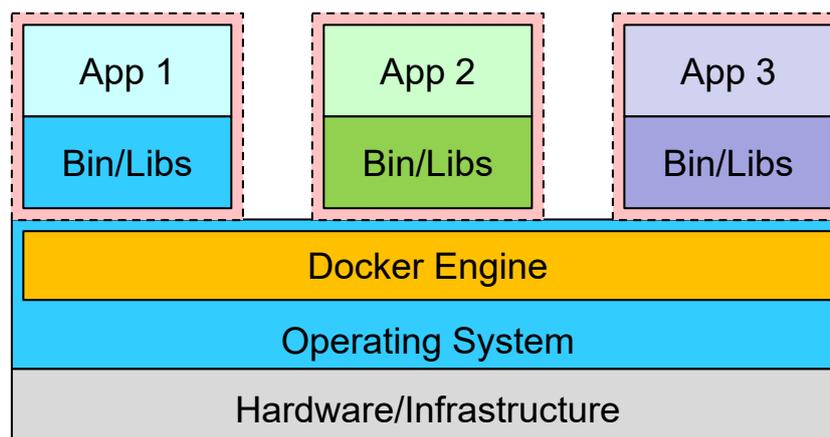


- Un container manager (o un container engine) ha l'obiettivo di gestire i container in un singolo host – tuttavia, può essere desiderabile poter eseguire i container di un'applicazione in un cluster di computer anziché in un singolo host
 - un *container orchestrator* è un software speciale e di livello ancora più alto per container, per gestire e coordinare l'esecuzione di molteplici container in un cluster di host
 - un esempio di container orchestrator è *Kubernetes*
 - parleremo di orchestrazione di container in un successivo capitolo



- Introduzione ai container Docker

- *Docker* è una piattaforma per container (un container engine)
 - per costruire, rilasciare ed eseguire applicazioni distribuite – in modo semplice, veloce, scalabile e portabile





Container Docker

- Un **container Docker** è un'unità di software standardizzata, che impacchetta un servizio software, insieme alle sue configurazioni e dipendenze
 - un container contiene ogni cosa necessaria per eseguire quel servizio software – codice eseguibile, configurazioni, librerie e strumenti di sistema
 - i container Docker sono leggeri, standardizzati e aperti e sicuri
 - un'immagine di container Docker diventa un'istanza di container a runtime quando viene eseguita nel Docker Engine



Funzionalità e utilizzo

- Ecco le principali funzionalità offerte dalla piattaforma Docker
 - creare un container (un'istanza di container) a partire da un'immagine di container
 - avviare, monitorare, ispezionare, arrestare e distruggere container
 - creare e gestire immagini di container
 - gestire gruppi correlati di container – in cui eseguire applicazioni distribuite multi-container



Discussione

- Docker (dal 2013) è stato un successo immediato ed è utilizzato in produzione da molte aziende – poche tecnologie hanno visto un tasso di adozione simile
 - i benefici principali di Docker sono leggerezza, efficienza, semplicità, velocità di provisioning, apertura, possibilità di rilascio su una varietà di piattaforme
 - una delle caratteristiche principali di Docker è proprio la portabilità
 - i container Docker sono ottimizzati per il rilascio di applicazioni o servizi individuali – sono “application container”
 - l’ecosistema di strumenti per Docker è molto interessante
 - in particolare, supporta la composizione e l’orchestrazione di container (discusse in successivi capitoli e dispense)
 - ulteriori informazioni su Docker (presenti nel Capitolo 39 del libro) sono presentate nella dispensa su Docker



* Container e macchine virtuali a confronto

- I container e le macchine virtuali hanno caratteristiche che sono tra loro complementari
 - le VM sono molto flessibili
 - ogni VM ha un proprio OS completo e le proprie applicazioni
 - i container offrono invece una flessibilità minore
 - l’OS di un container deve essere compatibile con l’OS dell’host (che è di solito Unix o Linux)
 - l’isolamento tra VM è completo
 - l’isolamento offerto dai container non è invece completo
 - tuttavia, la flessibilità e l’isolamento forniti dalla virtualizzazione di sistema non sono sempre richiesti
 - i container offrono degli ambienti di esecuzione che sono adeguati per molte applicazioni



Container e macchine virtuali a confronto

- ❑ I container e le macchine virtuali hanno caratteristiche che sono tra loro complementari
 - inoltre, la flessibilità offerta dalle VM ha un costo – una VM
 - richiede una quantità di risorse maggiori nel sistema host
 - può introdurre un overhead di esecuzione maggiore
 - richiede un tempo maggiore per l'avvio (minuti)
 - i container sono invece più “leggeri” delle VM
 - le prestazioni sono quasi native
 - richiedono una quantità minore di risorse
 - ad es., un'installazione Linux minimale richiede 1.1MB ca – le librerie di Ubuntu Server richiedono 180MB ca
 - è possibile una maggior densità di container per host
 - i container possono essere creati e avviati più velocemente
 - frazioni di secondo o al più secondi (escluso il pull dell'immagine)

45

Container e virtualizzazione basata su container

Luca Cabibbo ASW



* Container e rilascio del software

- ❑ I container sono un'altra opzione di rilascio per i sistemi software distribuiti
 - ogni container (“application container”) incapsula un servizio software, insieme allo stack software necessario per quel servizio

46

Container e virtualizzazione basata su container

Luca Cabibbo ASW



Benefici

- Benefici nell'usare i container per il rilascio del software
 - ogni container incapsula un singolo servizio software – il rilascio di un'istanza di quel servizio può essere gestito, in modo semplice e affidabile, come la creazione di un container
 - isolamento dei guasti e sicurezza – ogni container (con il relativo servizio) viene eseguito in modo abbastanza isolato
 - i container sono leggeri – è possibile allocare risorse ai container (e ai relativi servizi) e scalarli a grana fine
 - la creazione e l'avvio di un container richiedono in genere da una frazione di secondo a pochi secondi – meno che una VM
 - i container possono essere rilasciati sia nel cloud che on premises, in un proprio data center privato
 - in particolare, i container possono essere rilasciati in una piattaforma per l'orchestrazione di container (ad es., Kubernetes), sia on premises che nel cloud



Inconvenienti

- Inconvenienti nell'usare i container per il rilascio del software
 - l'isolamento tra container non è completo
 - overhead nell'amministrazione e nell'aggiornamento delle immagini dei container
 - overhead nell'amministrazione dell'infrastruttura di esecuzione dei container – a meno che i container non vengano eseguiti in una soluzione ospitata nel cloud (come AWS EKS o Google GKE)



* Discussione

- Sia i container che le macchine virtuali hanno i loro vantaggi e inconvenienti
 - le VM offrono un isolamento e una generalità maggiore – ma al prezzo di un overhead maggiore
 - i container offrono prestazioni migliori e un migliore utilizzo delle risorse – ma con un isolamento e una flessibilità minori
 - dunque, container e VM hanno caratteristiche complementari
 - ciascuna tecnologia offre dei vantaggi che potrebbero essere utili in situazioni specifiche



Discussione

- È limitativo pensare ai container solo come a una forma leggera di virtualizzazione
 - i container stanno cambiando in modo significativo il modo in cui i sistemi software distribuiti vengono rilasciati e mandati in esecuzione – e quello in cui vengono progettati e sviluppati
 - l'adozione dei container richiede un cambiamento nell'architettura del software
 - l'adozione dei container è così rapida, che nel giro di pochi anni ci si attende un uso regolare dei container in molti sistemi software
 - “da Gmail a YouTube passando dalla Ricerca, tutti i prodotti e servizi Google vengono eseguiti in container ... ogni settimana eseguiamo oltre diversi miliardi di container”
 - “l'80% di tutti i container nel cloud viene eseguito in AWS”