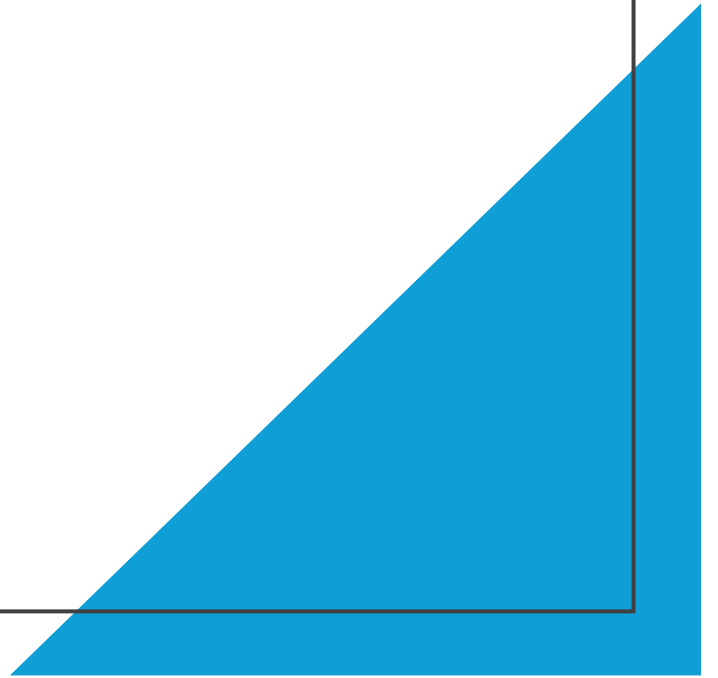



Introduzione a Docker e ai container

IIS OLIVETTI 2025




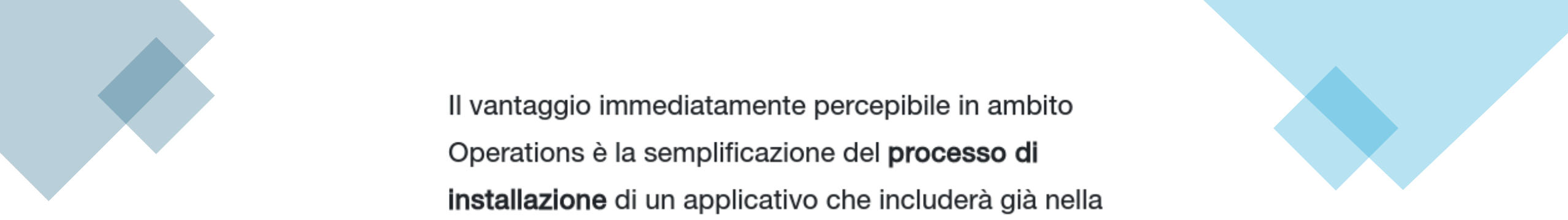
Ambiti d'uso di Docker

Un **contenitore** (o container) è una sorta di **bundle** configurabile che contiene **lo specifico applicativo e tutti i suoi requisiti e dipendenze**; il suo avvio richiede **pochi secondi**, come pure il suo arresto. I contenitori operano in **sandbox separate**, con i vantaggi derivanti da questo isolamento, ma è sempre possibile che insiemi di contenitori possano interagire tramite rete; infatti, come si vedrà in seguito, Docker permette anche la creazione di reti virtuali anch'esse implementate come contenitori. È inoltre possibile realizzare cluster di container dello stesso tipo per ottenere ridondanza e parallelismo operativo.



L'adozione di Docker può coinvolgere l'intero ciclo di vita del software. Sia la parte sviluppo –**Development** o **Dev**– che la fase di esercizio e conduzione operativa –**Operations** o **Ops**– possono ottenere vantaggi dall'adozione della tecnologia a container.

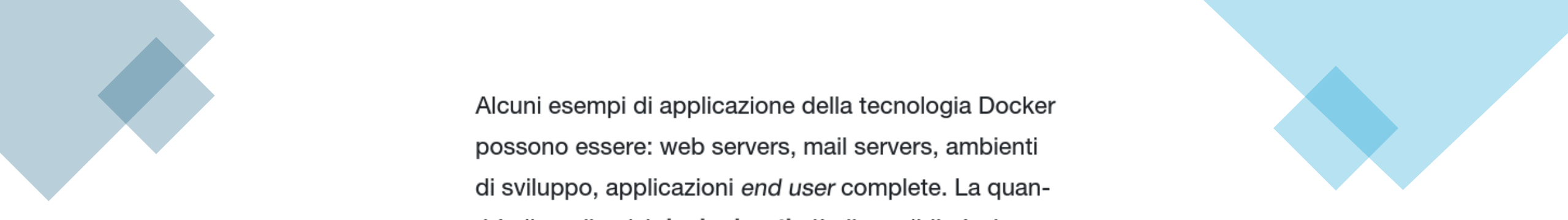


The top-left and top-right corners of the slide feature decorative geometric shapes. These consist of overlapping squares and rectangles in various shades of blue, creating a modern, abstract design.


Il vantaggio immediatamente percepibile in ambito Operations è la semplificazione del **processo di installazione** di un applicativo che includerà già nella sua **immagine** (l'archetipo del container) i pacchetti, le dipendenze e le configurazioni che altrimenti richiederebbero procedure di installazione dedicate; i contenitori consentono infatti di impacchettare in un **unico bundle** l'applicazione con tutto ciò di cui ha bisogno. Questa caratteristica garantisce che l'applicazione verrà eseguita su qualsiasi altra istanza Docker, indipendentemente dalle impostazioni specifiche della macchina Linux sottostante che potrebbero differire da quella utilizzata per scrivere e testare il codice. Da questa prospettiva potremmo vedere i container come una **tecnologia di delivery delle applicazioni**.

In ambito Development i contenitori favoriscono l'adozione di moderni paradigmi di programmazione quali:

- modularizzazione e interoperabilità
- continuous integration
- continuous deployment
- microservices



Alcuni esempi di applicazione della tecnologia Docker possono essere: web servers, mail servers, ambienti di sviluppo, applicazioni *end user* complete. La quantità di applicativi **dockerizzati** già disponibile è alta e aumenta giornalmente. Attualmente tra i più scaricati (decine di milioni di download) del Docker Hub sono:

- nginx
 - alpine
 - httpd
 - redis
 - busybox
 - ubuntu
 - mysql
 - mongo
 - postgres
 - node
- 

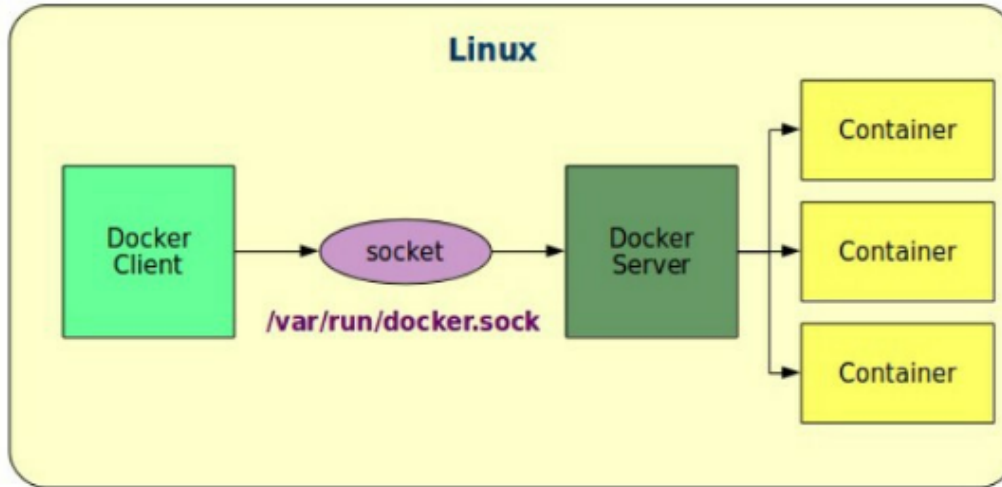
Architettura

Docker è basato su un'architettura Client-Server in cui:

- **docker** è il componente client che invia comandi e richieste al server
- **dockerd** è il server che gestisce i contenitori e le immagini

Solitamente client e server si trovano sulla stessa macchina. Il **server è responsabile dei propri contenitori locali** e l'utente vi interagisce attraverso il client, solitamente collegandosi al client avviene tramite `ssh`; è necessario che l'utente che dà i comandi client sia configurato nel gruppo `docker`. I messaggi tra client e server sono in chiaro e non esiste (ancora) un vero strato software di identificazione e autorizzazione.

Il collegamento tra client e server su un'unica macchina avviene tramite un **socket Unix**: `/var/run/docker.sock`



È possibile anche configurare Docker per usare una connessione **tcp**, in modo da avere il client e il server su macchine diverse, ma è una soluzione sconsigliata in quanto introduce criticità legate alla sicurezza.

La soluzione raccomandata in fase di primo apprendimento è:

- client e server sullo stesso host
- accesso alla macchina tramite ssh

Origini ed Evoluzione

Origini ed Evoluzione

La prima introduzione di meccanismi di isolamento di un processo risale al 1979 con l'introduzione nella versione 7 di unix di **chroot**, una feature del kernel utilizzata per isolare i limiti operativi di un'applicazione¹.

La prima versione commerciale di una tecnologia basata su container risale al **2004**, quando Sun rilascia Solaris 10, che include i Solaris Containers, più tardi evolutosi nelle **Solaris Zones**.

Nel **2006 Google** inizia lo sviluppo di una nuova feature di Linux nota con il nome di **process containers**. Più tardi fu denominata **Control groups** è entrata a far parte della **Linux kernel 2.6.24**, release dal gennaio 2008. I control groups (o cgroup) costituiscono la base fondamentale per le attuali implementazioni dei container.

Nel **2008** nasce **LXC** (Linux Containers): il primo framework di supporto all'utilizzo dei container, basato su cgroup.

Nel **2013** viene rilasciato **Docker**, basato su LXC e cgroup ma con notevoli migliorie, e nello stesso anno Google rilascia **Imctfy** per competere con Docker.

Nel **2014** Google annuncia che da tempo utilizza come tecnologia i container e rilascia con licenza open-source il suo prodotto per l'orchestrazione di container, **Kubernetes**.

Nel **2015** nasce l'**OCI2**, con l'obiettivo di definire degli standard aperti di riferimento per le soluzioni basate su container.

Docker nasce quindi dal 2013, dalle idee inizialmente proposte da Solomon Hykes.

Richiede un'architettura Linux a 64 bit con **Kernel 2.6** o superiore. È basato sulla feature architettónica detta **cgroup**, che altri sistemi operativi non hanno. Tuttavia è disponibile anche per Windows e Mac, ma questi sistemi eseguono Docker all'interno di una macchina virtuale Linux. È distribuito con licenza Apache3 (open) ed è scritto nel linguaggio di programmazione Go4. I Control Groups sono una feature del kernel la cui creazione è iniziata nel 2006 ad opera di ingegneri Google (principalmente Paul Menage e Rohit Seth). Il nome iniziale era “**process containers**”, successi-

vamente rinominato nel 2007 in **Control Groups** per evitare confusione con i Linux Containers. Il merge nel kernel Linux è avvenuto con la versione 2.6.24 e in seguito diverse nuove features sono state aggiunte.

Ma a cosa servono i **cgroup**? Semplicemente servono a **gestire le risorse della macchina** (CPU, memoria, network I/O, disk I/O, ecc.) ed a **profilare la loro allocazione ai processi in esecuzione**. In modo molto semplificato possiamo vederli come **collezioni di processi accomunati da criteri e limiti comuni sulle risorse**. Anche i Linux Container utilizzano i cgroup per profilare l'uso di risorse. Con Docker possiamo creare un container e passare opzioni come la seguente:

```
$ docker run -d --cpu-shares=20 \
```

```
--memory=1024 --name myapache docker.io/httpd
```

Un comando del genere, grazie ai cgroup, avvia un container di nome myapache con un **limite di uso cpu**

del 20% del totale disponibile sul sistema e **1G di memoria**. In pratica i **cgroup ci permettono di decidere**, per i vari processi o servizi, **quante risorse** (CPU, memoria, disk I/O) **allocare**.

Ad ogni modo la direzione futura della tecnologia container è quella indipendente dal sistema operativo.

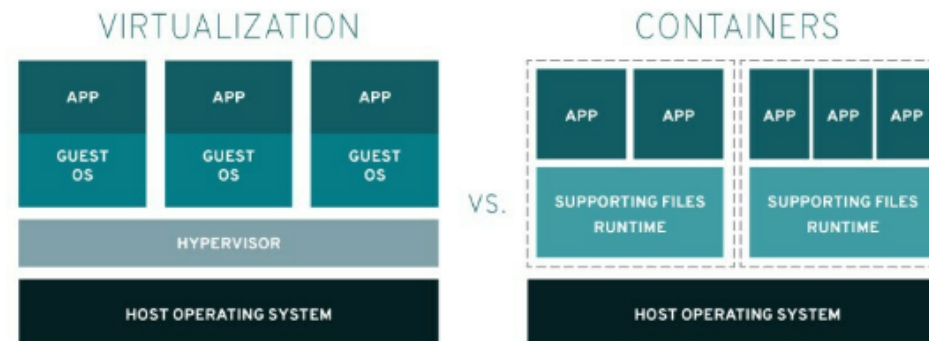
Virtual Machine e Container

Virtual Machine e Container

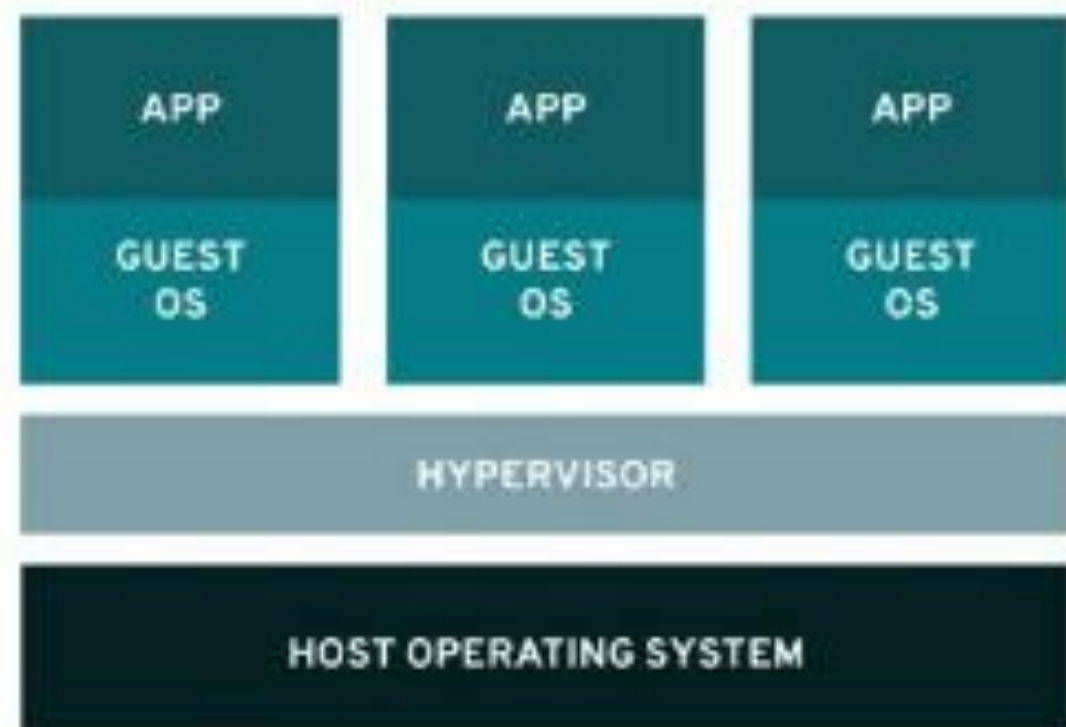
È fondamentale comprendere la differenza tra macchine virtuali e contenitori, e come questi ultimi vengono implementati su diverse piattaforme.

Una sostanziale differenza riguarda il sistema operativo:

- la virtualizzazione consente di eseguire più sistemi operativi contemporaneamente in un singolo sistema host
- i container condividono lo stesso kernel del sistema operativo e isolano i processi applicativi dal resto del sistema

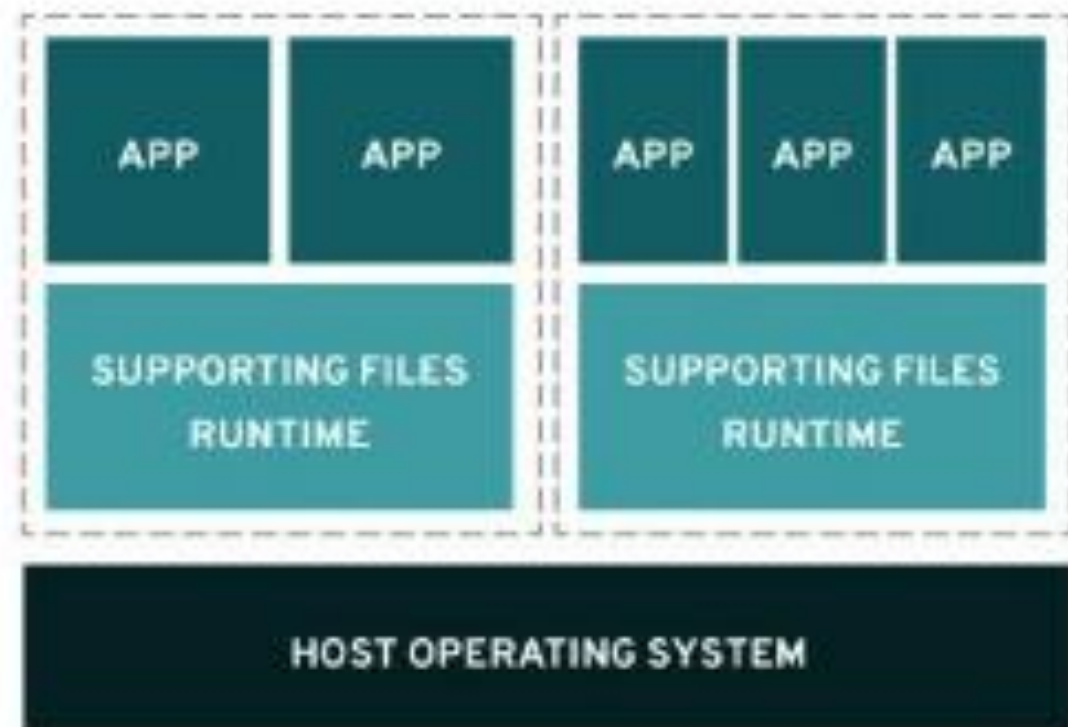


VIRTUALIZATION



VS.

CONTAINERS



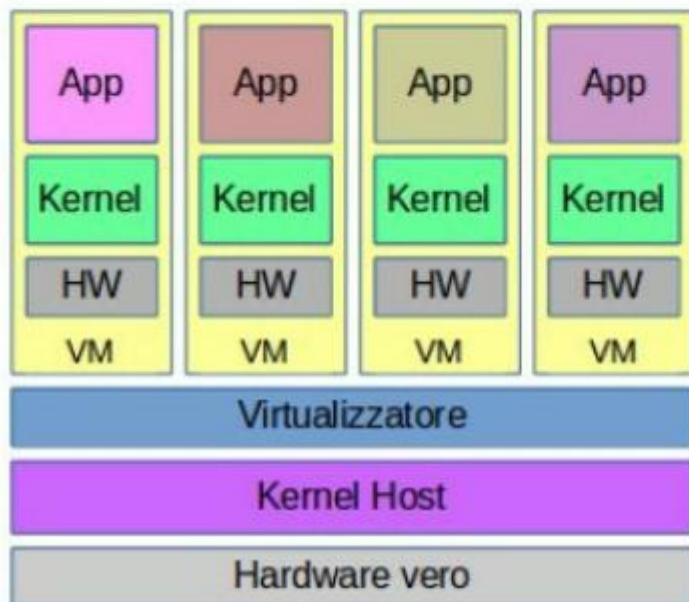
Come avremo modo di apprendere nel seguito, avere svariati sistemi operativi in funzione su un hypervisor, il software che consente la virtualizzazione, non è una soluzione leggera quanto avere l'uso di container.

Virtual machine

In ambito sistemistico con il termine **virtualizzazione** si indica la possibilità di **astrarre le componenti hardware**, cioè fisiche, **degli elaboratori al fine di renderle disponibili al software in forma di risorsa virtuale**.

Tramite questo processo è possibile installare sistemi operativi su hardware virtuale. L'insieme delle componenti hardware virtuali (disco fisso, RAM, CPU, scheda di rete) prende il nome di **Virtual Machine** e su di essa può essere installato il software come, appunto, il sistema operativo e le applicazioni. A livello enterprise le tecnologie di virtualizzazione vengono utilizzate per disporre di un **certo numero di sistemi operativi su un numero minore di piattaforme hardware**, spesso anche in modalità cluster di ridondanza (fail-over) o in

configurazione di load balancing. Uno dei principali vantaggi della virtualizzazione è la **razionalizzazione e l'ottimizzazione delle risorse hardware** grazie ai meccanismi di distribuzione delle risorse effettivamente disponibili su una piattaforma fisica. Si ottiene che più macchine virtuali possono girare contemporaneamente su un sistema fisico condividendo le risorse della piattaforma.



Tra gli altri vantaggi della virtualizzazione possiamo citare:

- la disponibilità di immagini di più sistemi diversi
- un notevole grado di flessibilità di configurazione
- una tecnologia matura.

Tra gli svantaggi ricordiamo:

- la considerevole richiesta di risorse
- installazione, configurazione e manutenzione onerosa in termini di tempo.

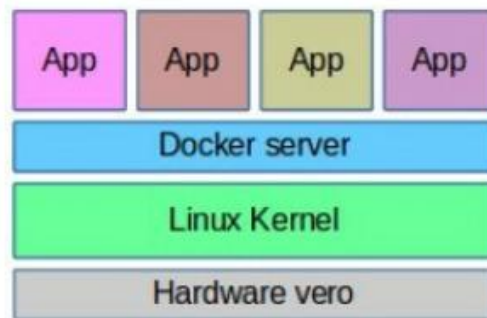
Container

I contenitori rappresentano una “virtualizzazione” dell’**ambiente operativo software** necessario alle applicazioni, **non dell’hardware**; possono essere definiti come un **meccanismo di isolamento di un processo**. Con Docker si realizza **un’istanza virtuale solo dello spazio utente**, quindi essenzialmente dell’ambiente di

L’isolamento dei container è realizzato attraverso l’utilizzo dei **kernel namespaces**, cioè una feature del kernel che realizza un’astrazione di alcune risorse di sistema globali come PID¹, Mount, Network, Users. Diversi container potranno quindi utilizzare contemporaneamente la stessa risorsa senza generare conflitti.

esecuzione delle applicazioni; dal sistema operativo “in giù” verso l’hardware **nulla è virtuale**, bensì reale e condiviso fra tutti i container in esecuzione. **Non dovendo inglobare tutte le risorse di un server**, in particolare il kernel, i **container sono molto più “leggeri”** delle macchine virtuali, richiedono poche risorse e possono essere attivati in pochi istanti. Questo li rende particolarmente adatti alle situazioni in cui il carico di elaborazione da sostenere è fortemente variabile nel tempo ed ha picchi poco prevedibili.

L’implementazione dei Contenitori su Linux è illustrato di seguito.



- **deploy** – semplifica il deployment poiché l'unità di deployment è un'immagine **autoconsistente** e **versionata** pronta per essere eseguita
- **portabilità** – un container eseguito in un'istanza di Docker può essere facilmente replicato in **un'altra istanza Docker**, con la stessa consistenza e funzionalità, garantendo un alto livello di astrazione dal livello dell'infrastruttura
- **isolamento** – assicura che le applicazioni siano separate poiché ogni container ha risorse isolate da quelle degli altri
- **sicurezza** – le applicazioni eseguite sui container sono completamente **isolate** le une dalle altre. Nessun container **vede i processi in esecuzione all'interno di un altro** container e può interferire con il suo funzionamento. Ovviamente bisogna conoscere le tematiche di sicurezza coinvolte nell'adozione di un'infrastruttura basata su container e curarne ogni aspetto.

- **provisioning** – su sistemi tradizionali la configurazione ed il provisioning di un nuovo hardware possono richiedere molto tempo. **Con Docker i file descrittori delle immagini sono in configurazione sul sistema di versionamento del codice (es: git) e le immagini sono sempre disponibili sul registry**; l'avvio avviene con una riga di comando ed in pochi secondi
- **efficienza** – non dovendo inglobare tutte le risorse di un server, in particolare il **kernel del sistema operativo**, i container sono molto più **"leggeri"** delle macchine virtuali, richiedono poche risorse di CPU e possono essere avviati in **con delay minimi**.

Docker è una implementazione nativa di Linux e richiede il **Kernel Linux**, ma in teoria la tecnologia dei contenitori può essere implementata su diversi sistemi operativi ospitanti - come detto ne esisteva una ver-