

Cloud Computing

Internet of Things

L'Internet degli oggetti è un sistema di oggetti fisici che possono essere scoperti, monitorati, controllati o con cui si può interagire tramite dispositivi elettronici che comunicano attraverso varie interfacce di rete e che, alla fine, possono essere collegati a Internet.

Che cosa può essere un "Thing"? Oggetti fisici dotati di uno o più dei seguenti elementi:

- Sensori (temperatura, luce, movimento...)
- Attuatori (display, suono, motori...)
- Calcolo (può eseguire programmi e logica)
- Interfacce di comunicazione (cablate o wireless)

Edge Computing

Il "Edge Computing" è un paradigma di elaborazione dei dati distribuito che sposta la capacità di calcolo e l'elaborazione dei dati più vicino alle sorgenti di generazione dei dati, ovvero ai "bordi" della rete. In sostanza, l'Edge Computing si contrappone al modello tradizionale in cui tutti i dati sono elaborati in data center centralizzati o nel cloud.

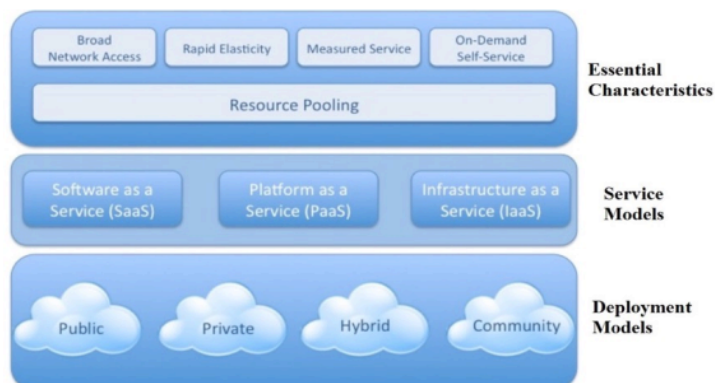
Sfide

1. Aumento delle capacità delle risorse Edge
2. Il trasferimento dei dati da/verso centri dati centrali ben forniti è costoso.
3. Problemi di sicurezza/privacy nello spostamento dei dati
4. Non tutti i dati edge sono essenziali

Cloud Computing

Il cloud computing è un modello che consente di accedere in rete in modo ubiquo, conveniente e on-demand a un pool condiviso di risorse informatiche configurabili (ad esempio, reti, server, storage, applicazioni e servizi) che possono essere rapidamente messe a disposizione e rilasciate con un minimo sforzo di gestione o di interazione con il fornitore di servizi.

Questo modello di cloud promuove la disponibilità ed è composto da *cinque caratteristiche essenziali*, *tre modelli di servizio* e *quattro modelli di distribuzione*.



11.1 Caratteristiche del Cloud Computing

- **On-demand self-service:** un consumatore può fornire unilateralmente capacità di elaborazione, come il tempo del server e l'archiviazione di rete, secondo le necessità, automaticamente, senza (o con poca) interazione umana con un fornitore di cloud.
- **Broad network access:** le funzionalità sono disponibili in rete e accessibili attraverso meccanismi standard che ne favoriscono l'utilizzo da parte di piattaforme client eterogenee, sottili o spesse (ad esempio, telefoni cellulari, computer portatili e PDA), nonché da altri servizi software tradizionali o basati su cloud.
- **Resource pooling:** le risorse informatiche del provider sono messe in comune per servire più consumatori utilizzando un modello multi-tenant, con diverse risorse fisiche e virtuali assegnate e riassegnate dinamicamente in base alla domanda dei consumatori.
- **Rapid elasticity:** le capacità possono essere fornite rapidamente ed elasticamente per scalare rapidamente, e rilasciate rapidamente per scalare rapidamente, automaticamente. Per i consumatori, le capacità disponibili per il provisioning sembrano spesso illimitate e possono essere acquistate in qualsiasi quantità e in qualsiasi momento.
- **Measured service:** I sistemi cloud controllano e ottimizzano automaticamente l'utilizzo delle risorse sfruttando una capacità di misurazione a un livello di astrazione adeguato al tipo di servizio. L'utilizzo delle risorse può essere monitorato, controllato e segnalato, fornendo trasparenza sia al fornitore che al consumatore del servizio.

11.2 Modelli di servizio

1. **Software as a Service (SaaS)** si riferisce a un modello di distribuzione del software in cui le applicazioni sono ospitate da un provider di servizi cloud e messe a disposizione degli utenti tramite Internet.
2. **Platform as a Service (PaaS)** è un ambiente di sviluppo e distribuzione completo basato su cloud, fornito da un provider di servizi cloud.
3. **Infrastructure as a Service (IaaS)** è il livello più fondamentale tra i tre, fornendo risorse di calcolo virtuali e infrastruttura su richiesta tramite Internet.

11.3 Modelli di distribuzione

	Public Cloud	Private Cloud
Infrastruttura	Omogenea	Eterogenea
Modello di policy	Definizione comune	Personalizzato e creato appositamente
Modello di risorse	Condiviso e multi-tenant	Dedicato
Modello di costo	Spendi per ciò che usi quando ti serve	Grande investimento iniziale
Modello di economia	Grande risparmio di massa ma meno controllo sulle cose	Meno risparmio ma controllo su tutto

- Il **cloud pubblico** è un modello di distribuzione in cui i servizi cloud sono offerti da un provider di servizi cloud attraverso Internet pubblica. Le risorse, come server e storage, sono condivise tra più clienti e gestite dal provider. Questo modello è economico, flessibile e può scalare facilmente le risorse in base alle esigenze.
- Il **cloud privato** è un modello di distribuzione in cui le risorse cloud sono utilizzate esclusivamente da un singolo utente o organizzazione. Può essere ospitato internamente o esternamente, ma l'accesso e il controllo sono limitati al singolo utente o all'organizzazione. Offre maggiore controllo e sicurezza rispetto al cloud pubblico.
- Il **cloud ibrido** è un modello che combina elementi di cloud pubblico e cloud privato. Consente alle organizzazioni di distribuire le loro applicazioni e carichi di lavoro tra più ambienti cloud (pubblico e privato) in base a requisiti specifici di sicurezza, compliance, e performance.
- Il **cloud community** è un modello di distribuzione in cui le risorse cloud sono condivise tra più organizzazioni che hanno interessi comuni, come requisiti di sicurezza, conformità o settore industriale specifico. È simile al cloud pubblico, ma è più mirato e personalizzato per le esigenze di una comunità specifica di utenti.

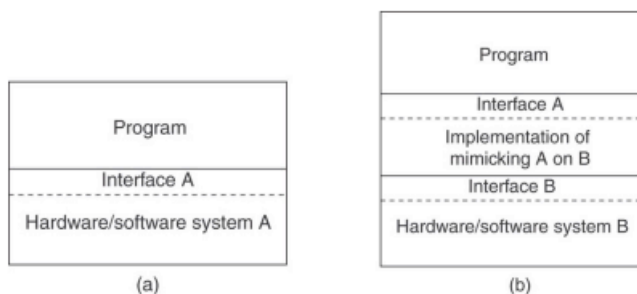
11.4 Virtualizzazione

La funzione più importante della virtualizzazione è la **capacità di eseguire più sistemi operativi e applicazioni contemporaneamente**, indipendentemente dalla piattaforma o dall'hardware sottostante.

La virtualizzazione

- aiuta a isolare i guasti causati da errori o problemi di sicurezza;
- consente di introdurre nuove funzionalità di sistema senza aggiungere complessità a hardware e software già complessi;
- può solitamente migliorare le prestazioni complessive delle applicazioni grazie a una tecnologia in grado di bilanciare le risorse e di fornire solo ciò di cui l'utente ha bisogno.

I recenti progressi nelle tecnologie di virtualizzazione possono essere applicati a diverse aree di ricerca, come la gestione delle risorse virtualizzate, il monitoraggio e il recupero dagli attacchi.



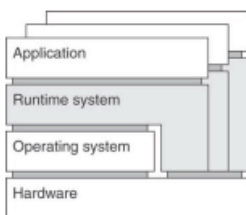
(a) Organizzazione generale tra programma, interfaccia e sistema.

(b) Organizzazione generale della virtualizzazione del sistema A sul sistema B.

11.4.1 Tipi di virtualizzazione

Process Virtual Machine: virtualizzazione tramite interpretazione o emulazione

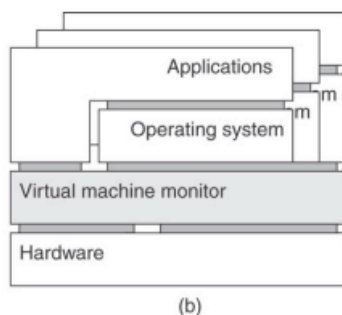
- Essenzialmente viene eseguita per un solo processo
- Fornisce un set di istruzioni astratto per la macchina; i programmi vengono compilati in codice "macchina" che viene poi interpretato (ad esempio, Java) o emulato (ad esempio, Windows).



Un process virtual machine, con istanze multiple di combinazioni di (applicazione, runtime).

Virtual Machine Monitor: o hypervisor, in grado di fornire una macchina virtuale a molti programmi diversi contemporaneamente; come se più CPU fossero in esecuzione su un'unica piattaforma.

- Particolarmente importante per i sistemi distribuiti, perché le applicazioni sono isolate e gli errori sono limitati a una singola macchina virtuale.
- Esempi: VMware e Xen



Un virtual machine monitor, con istanze multiple di combinazioni di (applicazioni/sistema operativo).

Microservizi

12.1 Architettura a microservizi

Architettura a microservizi

Un'architettura a microservizi inserisce ogni elemento di funzionalità in un servizio separato e rende ogni servizio un'unità indipendente di implementazione.

I microservizi promuovono i principi e le pratiche dell'ingegneria del software

- Basso accoppiamento e alta coesione
- Asincrono rispetto a sincrono
- Coreografia piuttosto che orchestrazione

I microservizi sono progettati secondo il **principio della singola responsabilità**:

- Ogni modulo o classe deve avere la responsabilità su una singola parte della funzionalità fornita dal software e tale responsabilità deve essere interamente incapsulata dalla classe.

I microservizi riguardano le **persone**:

- I microservizi accelerano la consegna minimizzando la comunicazione e il coordinamento tra le persone e riducendo la portata e il rischio di cambiamento.

12.2 Applicazione monolitica

Applicazione monolitica

Il nucleo dell'applicazione è la logica di business, implementata da moduli che definiscono servizi, oggetti di dominio ed eventi. Intorno al nucleo ci sono gli adattatori che si interfacciano con il mondo esterno. Esempi di adattatori sono l'accesso al database, i componenti di messaggistica e i componenti web che espongono API o implementano un'interfaccia utente. Nonostante l'architettura logicamente modulare, l'applicazione viene confezionata e distribuita come un monolite. Il formato effettivo dipende dal linguaggio e dal framework dell'applicazione.

Ad esempio, molte applicazioni Java sono confezionate come file WAR e distribuite su server di applicazioni come Tomcat o Jetty.

Problema: Monoliti di grandi dimensioni

- Sono difficili da mantenere ed evolvere

- Soffrono dell'"inferno delle dipendenze", in cui l'aggiunta o l'aggiornamento delle librerie dà luogo a sistemi incoerenti
- Richiedono il riavvio per qualsiasi cambiamento in un modulo
- Rappresentano un lock-in tecnologico per gli sviluppatori
- Può soffrire di problemi di affidabilità: un bug in un qualsiasi modulo, come una perdita di memoria, può potenzialmente far fallire l'intero processo
- Mostrano un'implementazione costosa o non ottimale rispetto ai requisiti dei singoli moduli.

L'idea dell'architettura a microservizi è proprio quello di distruggere l'idea del creare un unico monolite e suddividere l'applicazioni in un insieme di piccoli servizi interconnessi.

12.3 Servizi

Un **servizio** implementa tipicamente un insieme di caratteristiche o funzionalità distinte, come la gestione degli ordini, la gestione dei clienti, ecc.

Ogni microservizio è una mini-applicazione.

Alcuni microservizi espongono

- un'API che viene utilizzata da altri microservizi o client;
- un'interfaccia web.

Vantaggi

1. Affronta il **problema della complessità**.

- L'applicazione è stata suddivisa in parti gestibili o servizi.
- Ogni servizio ha un confine ben definito sotto forma di API RPC o messaggi.
- I singoli servizi sono molto più veloci da sviluppare e molto più facili da comprendere e mantenere.

2. Permette a ogni servizio di essere **sviluppato in modo indipendente** da un team che si concentra su quel servizio.

- Gli sviluppatori sono liberi di scegliere le tecnologie che ritengono più opportune, purché il servizio rispetti il contratto API.
- Hanno la possibilità di utilizzare la tecnologia attuale. Inoltre, poiché i servizi sono relativamente piccoli, diventa possibile riscrivere un vecchio servizio utilizzando la tecnologia attuale.

3. Permette a ogni microservizio di essere **distribuito in modo indipendente**.

- Il pattern dell'architettura a microservizi rende possibile il deployment continuo.
- Gli sviluppatori non hanno mai bisogno di coordinare il deployment delle modifiche che sono locali al loro servizio. Questo tipo di modifiche possono essere distribuite non appena sono state testate. Il team dell'interfaccia utente può, ad esempio, eseguire test A/B e iterare rapidamente le modifiche dell'interfaccia.

4. Permette di **scalare ogni servizio in modo indipendente**.

- È possibile distribuire solo il numero di istanze di ciascun servizio che soddisfa i vincoli di capacità e disponibilità. Inoltre, è possibile utilizzare l'hardware più adatto ai requisiti di risorse di un servizio.

Scalabilità

- Scalabilità sull'asse X: esecuzione di più copie di un'applicazione dietro un bilanciatore di carico.
- Scalabilità sull'asse Y: suddivisione dell'applicazione in più servizi diversi.
- Scalatura sull'asse Z: ogni server è responsabile solo di un sottoinsieme dei dati.

L'architettura a microservizi è un'applicazione dell'asse Y, ma analizziamo anche gli assi X e Z.

Consigli

- Non prendete nemmeno in considerazione i microservizi a meno che non abbiate un sistema troppo complesso da gestire come monolite
- La maggior parte dei sistemi software dovrebbe essere costruita come un'unica applicazione monolitica.
- Prestate attenzione a una buona modularità all'interno del monolite, ma non cercate di separarlo in servizi distinti.

Containers

Problemi:

1. I componenti funzionali (applicazioni) [possono] necessitare di componenti aggiuntivi (strumenti e librerie) per fornire le proprietà e le qualità richieste
2. Applicazioni + librerie + strumenti dovrebbero essere distribuiti insieme per semplificare la gestione e fornire un comportamento coerente (ad esempio, dalle infrastrutture di sviluppo a quelle di produzione).

Container

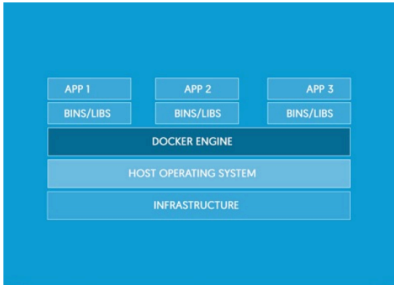
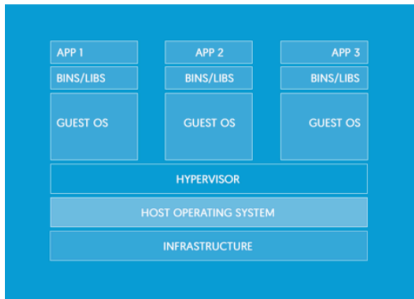
Un modo standard per impacchettare un'applicazione e tutte le sue dipendenze in modo che possa essere spostata da un ambiente all'altro ed eseguita senza modifiche. I contenitori funzionano isolando le differenze tra le applicazioni all'interno del contenitore, in modo che tutto ciò che è al di fuori del contenitore possa essere standardizzato.

I container sono unità più leggere, non legate ad alcuna infrastruttura specifica.

Le applicazioni nei container sono (potenzialmente) più robuste, sicure e performanti.

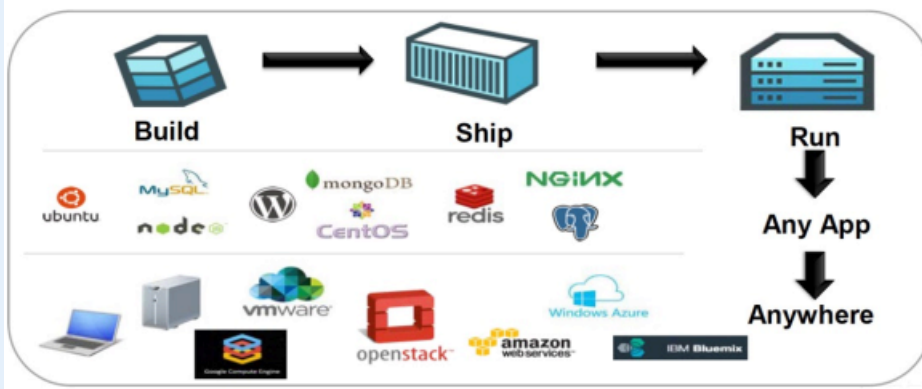
13.1 Docker

VMM Model vs Container Model


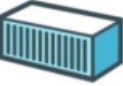


Container	VMM
Le applicazioni vengono impacchettate con tutte le loro dipendenze in un'unità standardizzata per lo sviluppo e la distribuzione del software.	Le applicazioni dipendono dal sistema operativo ospitante.
 The diagram shows a stack of layers. At the bottom is 'INFRASTRUCTURE'. Above it is 'HOST OPERATING SYSTEM'. Then 'DOCKER ENGINE'. On top of the Docker Engine are three separate boxes, each containing 'APP 1', 'APP 2', and 'APP 3' stacked vertically, with 'BINS/LIBS' below each app name.	 The diagram shows a stack of layers. At the bottom is 'INFRASTRUCTURE'. Above it is 'HOST OPERATING SYSTEM'. Then 'HYPERVISOR'. On top of the Hypervisor are three separate boxes, each containing 'GUEST OS'. Above each Guest OS are three separate boxes, each containing 'APP 1', 'APP 2', and 'APP 3' stacked vertically, with 'BINS/LIBS' below each app name.

Docker

Docker è una piattaforma aperta per la creazione di applicazioni distribuite per sviluppatori e amministratori di sistema.



13.2 Basi di Docker

Elemento	Definizione
Immagine Docker 	Un'immagine Docker è un pacchetto leggero, autonomo ed eseguibile che include tutto ciò che è necessario per eseguire un pezzo di software, compresi il codice, i runtime, le librerie e le dipendenze, nonché le impostazioni necessarie.
Container Docker 	Un container Docker è un'istanza runtime di un'immagine Docker. I container sono ambienti di runtime isolati che includono tutto il necessario per eseguire un'applicazione.
Docker Hub/Registry 	Docker Hub è un servizio di registry pubblico dove gli utenti possono creare, memorizzare e distribuire le loro immagini Docker. Esistono anche registri privati (registry) che possono essere configurati per uso interno alle organizzazioni.
Docker Engine 	Docker Engine è il motore runtime che permette di costruire e gestire i container. Consiste in un daemon server-side e una CLI (command line interface) client-side.

13.3 Architettura Docker

Docker utilizza un'**architettura client-server**.

- Il client e il demone Docker possono essere eseguiti sullo stesso sistema, oppure è possibile collegare un client Docker a un demone Docker remoto. Comunicano utilizzando un'API REST, tramite socket UNIX o un'interfaccia di rete.

- Build:** crea un'immagine (un'applicazione)
- Pull:** recupero di un'immagine (un'applicazione) da un registro
Un registro Docker memorizza le immagini Docker. Docker Hub e Docker Cloud sono registri pubblici e Docker è configurato per cercare le immagini su Docker Hub per impostazione predefinita. È anche possibile gestire un registro privato.
- Run:** esegui un'istanza dell'immagine (un'applicazione)

13.3.1 Docker Engine

Docker Engine è un'applicazione client-server con questi componenti principali:

- Un server che è un tipo di programma a lunga durata chiamato processo daemon (il comando dockerd).
- Un'API REST che specifica le interfacce che i programmi possono usare per parlare con il daemon e istruirlo su cosa fare.
- Un client per l'interfaccia a riga di comando (CLI) (il comando docker).

La CLI utilizza l'API REST di Docker per controllare o interagire con il daemon Docker tramite script o comandi diretti della CLI. Molte altre applicazioni Docker utilizzano l'API e la CLI sottostanti.

Il daemon crea e gestisce gli oggetti Docker, come immagini, contenitori, reti e volumi.

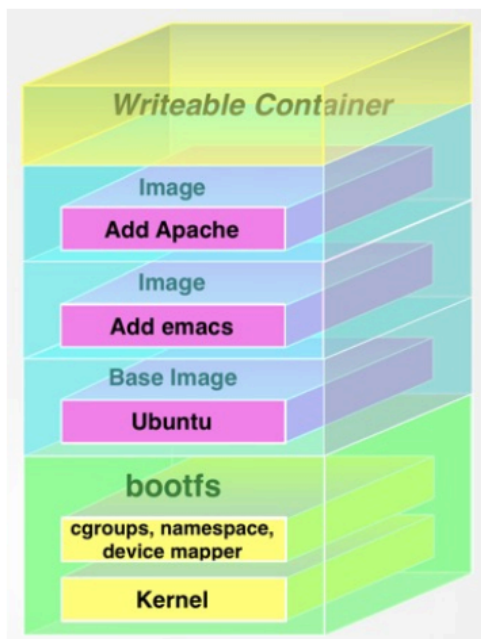
13.3.2 Docker Image

Un'immagine Docker è composta da filesystem sovrapposti.

Alla base c'è un filesystem di avvio, bootfs, che assomiglia al tipico filesystem di avvio Linux/Unix.

Utilizza la tecnica dell'**union mount**, che consente di montare diversi filesystem contemporaneamente, ma che appaiono come un unico filesystem.

- Lo schema "**copy on write**" è una delle caratteristiche che rendono Docker così potente.
 - Quando Docker avvia per la prima volta un contenitore, il livello iniziale di lettura e scrittura è vuoto.
 - Man mano che si verificano modifiche, queste vengono applicate a questo livello; per esempio, se si desidera modificare un file, questo verrà copiato dal livello di sola lettura sottostante al livello di scrittura.
 - La versione di sola lettura del file continuerà a esistere, ma ora è nascosta sotto la copia.



13.4 Per cosa puoi usare Docker

- Contribuire a rendere il flusso di lavoro di sviluppo e creazione locale più veloce, efficiente e leggero.
- Esecuzione di servizi e applicazioni stand-alone in modo coerente su più ambienti.
- Usare Docker per creare istanze isolate per eseguire i test.
- Creare e testare applicazioni e architetture complesse su un host locale prima di distribuirle in un ambiente di produzione.
- Costruire un'infrastruttura Platform-as-a-Service (PAAS) multiutente.
- Fornire ambienti sandbox leggeri e autonomi per lo sviluppo, il test e l'insegnamento di tecnologie, come la shell Unix o un linguaggio di programmazione.
- Applicazioni Software as a Service.

- Distribuzione di host altamente performanti e su larga scala.