Riassunto di Cloud computing

# Cos’è il cloud computing?

Il cloud computing è un modello che permette la richiesta di servizi in modo on demand e semplice. Tutti i sistemi cloud hanno le stesse caratteristiche, la loro differenza sta nell’implementazione.

Più precisamente, il cloud computing è un gruppo di macchine appositamente configurate da permettere all’utente la richiesta di un dato numero di macchine virtuali. L’ubicazione di queste macchine virtuali non ha importanza, così come le specifiche tecniche delle macchine fisiche che li ospitano.

Il cloud può essere paragonato alla corrente elettrica, infatti per sopperire al carico ha bisogno di adattarsi alla situazione, aumentando anche vertiginosamente.

I servizi cloud vengono forniti attraverso delle macchine virtuali, queste vengono opportunamente accese o spente in base alla quota pagata.

Il cloud è ovunque! Infatti è molto utilizzato dai social e dai motori di ricerca, inoltre esistono servizi che permettono il salvataggio e la modifica dei dati.

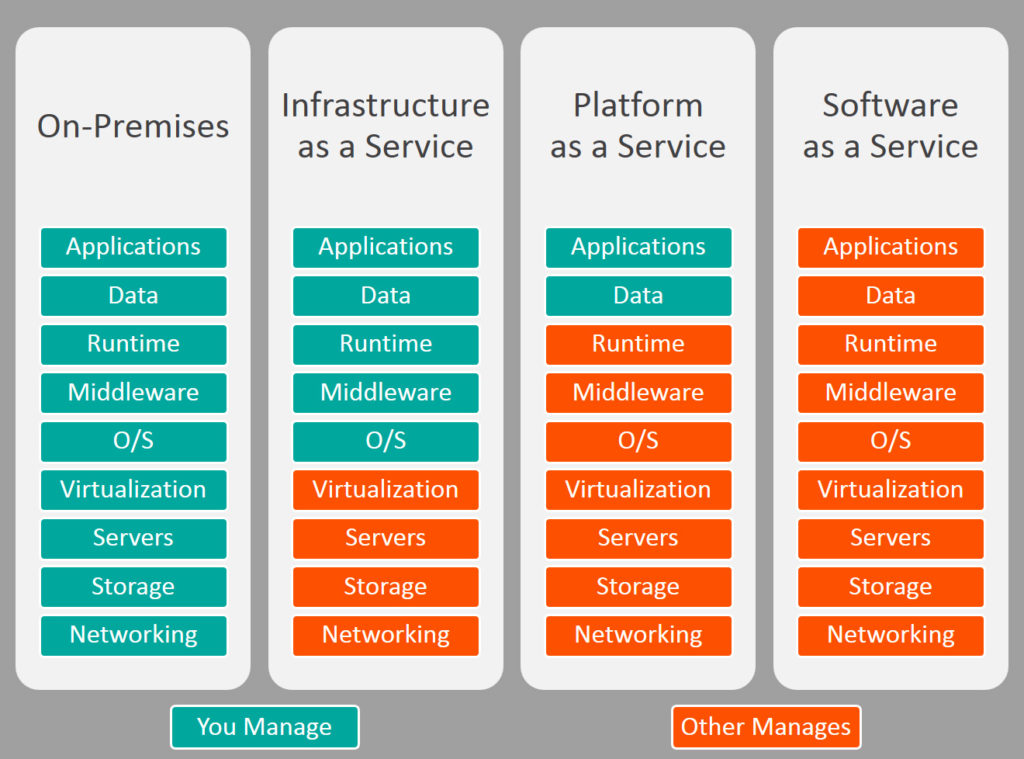
In particolare, i social network si adattano benissimo al cloud, permettendo più elasticità, in questo ambito inoltre il numero di richieste degli utenti è variabile nel tempo.

Essendo un sistema di tipo distribuito, il cloud ha bisogno di un lavoro di tolleranza ai guasti mica da ridere!

## Caratteristiche dei sistemi cloud

* On demand: gli utenti possono chiedere risorse in ogni momento senza l’intervento umano;
* Accessibilità da internet senza limiti;
* Le risorse non hanno una locazione specifica, se questa è presente, è definita ad alto livello. In Questo punto è utile gestire la privacy;
* Elasticità rapida, cioè la capacità di adattarsi alla situazione in modo rapido;
* Tutti i servizi vengono misurati, ciò permette di controllare e ottimizzare le risorse.

## Modelli di servizio

* Software as a Service (SaaS): questo modello fornisce dei software nascondendone l’implementazione, esso può essere replicato per garantire una certa tolleranza ai guasti. La gestione degli aggiornamenti viene fatta dal fornitore stesso. Un esempio di SaaS è la G Suite;
* Platform as a Service (PaaS): In questo caso il consumatore ha una piattaforma su cui effettuare il deploy delle applicazioni, l’accesso è garantito attraverso delle API. I fornitori di PaaS definiscono combinazioni di sistema operativo e applicazioni server, come ad esempio LAMP o App Engine;
* Infrastructure as a Service (IaaS): il consumatore ha un’intera infrastruttura a sua disposizione con accesso a processing, storage e rete, un vero e proprio computer in remoto.

## Modelli di deployment

I modelli di deployment indicano come il consumatore si approccia al cloud, ce ne sono di diversi tipi.

### Cloud pubblico

Il cloud pubblico è quello utilizzabile da tutti. I possessori sono terze parti i quali gestiscono e risolvono i vari problemi che occorrono. L’approccio utilizzato è il “Pay as you go”, cioè si paga quello che effettivamente si utilizza. L’infrastruttura è condivisa da tutti i consumatori, questi ultimi vengono gestiti con:

* configurazioni limitate;
* protezioni di sicurezza;
* disponibilità variabili.

## Cloud privato

Il cloud privato è un servizio costruito e utilizzato da una sola enterprise, esso risulta utile nella sicurezza dei dati e permette più controllo.

Per questo tipo di cloud, esistono due opzioni:

* cloud on premise;
* cloud ospitato esternamente.

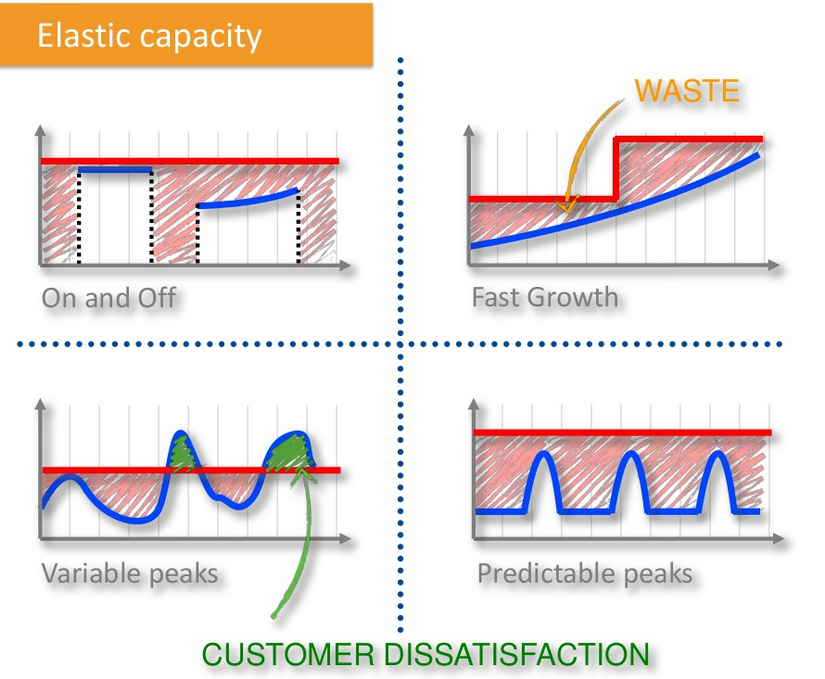
Per quanto riguarda quest’ultima tipologia di cloud, questi vengono ospitati insieme a un cloud provider, il quale facilita l’ambiente esclusivo e un privacy piena garantita.

Questa soluzione si adatta a società che non vogliono il cloud pubblico a causa della condivisione delle risorse.

## Cloud ibrido

Il cloud ibrido è una combinazione di quello pubblico e privato. I fornitori dei servizi possono utilizzare servizi di terze parti pienamente o parzialmente in modo da incrementare la flessibilità. Inoltre permette di gestire aumenti anomali di carico di lavoro usando risorse di cloud pubblico, come una sorta di estensione del cloud privato.

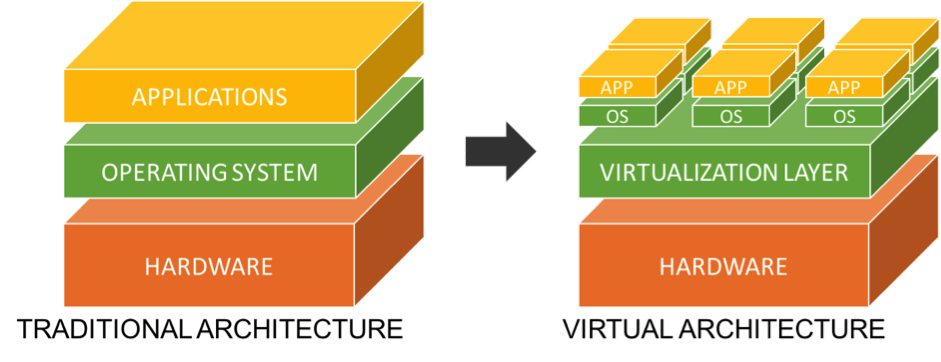
Questa soluzione permette di:

* ridurre i costi grazie al pay as you go;
* incrementare lo storage;
* rendere il tutto più flessibile, questa proprietà è importante dato che da essa dipende la soddisfacibilità dell’utente.

L’unica nota dolente del cloud ibrido riguarda la privacy e la sicurezza dei dati, le società hanno paura di perderli e quindi i provider sono responsabili della loro sicurezza, onde evitare di passare a un altro provider.

Il cloud computing deve inoltre garantire recovery e disponibilità dei dati, tutte le applicazioni di business hanno un accordo di livello di servizio (SLA) che deve essere seguito in maniera ferrea! Proprio per questo motivo, in cloud sono presenti orchestratori che permettono il load balancing e altri servizi come l’autoscaling.

# Grid computing

Il passato del cloud computing era il grid computing, un sistema in cui diversi device indipendenti vengono connessi insieme attraverso una rete ad alta velocità. Tutto ciò viene visto come un sistema unico grazie a un middleware. Tutti i nodi di un grid non devono per forza avere la stessa configurazione, nonostante l’omogeneità permetta di aumentarne le performance.

La virtualizzazione ha permesso ai sistemi grid di diventare cloud, permettendo la gestione dei carichi di lavoro in modo più scalabile.

# Fog Computing

Il fog computing è un’infrastruttura generalizzata in cui i dati, lo storage, la computazione e le applicazione si trovano da qualche parte tra le sorgenti e il cloud. Questa nuova infrastruttura permette gli stessi vantaggi del cloud in un luogo più vicino alla creazione dei dati.

## FemToCloud

Il FemToCloud è un ramo del fog computing in cui si mette a disposizione la potenza di calcolo non utilizzata dagli utenti. In cambio, gli utenti possono ricevere degli sconti o una valuta in denaro.

# Caratteristiche del cloud computing

Le varie piattaforme cloud non seguono uno standard, infatti una stessa operazione viene effettuata in modo differente in diverse piattaforme.

Le parti che si possono considerare comuni sono le seguenti:

* API, cioè il metodo di accesso al cloud;
* la zona di disponibilità di un insieme di VM in cloud;
* la tolleranza ai guasti;
* La migrazione di una VM da una macchina a un’altra (anche a caldo);
* Il monitoraggio tramite tool interni o esterni;
* Il supporto a OVF (Open Virtualization Format);
* Lo scaling, cioè la possibilità di aggiungere o togliere risorse in automatico quando è necessario;
* La gestione da parte dell’utente, ad esempio se è presente un’interfaccia grafica, c’è bisogno di privilegi, eccetera.

## Architettura

La foto a destra indica l’architettura di una piattaforma cloud:

* in basso sono presenti i nodi, i device veri e propri, ognuno di essi è provvisto di un sistema operativo, un hypervisor e un insieme di macchine virtuali;
* Il sistema operativo di ogni nodo non deve per forza essere uguale per tutti;
* DHCP e DNS permettono di gestire MAC e IP delle VM;
* Una repository delle immagini dei dischi;
* un collegamento col frontend per gli utenti;
* La piattaforma cloud vera e propria.

## LAMP

LAMP è uno stack che permette l’esecuzione di un’applicazione web, precisamente utilizza:

* Linux come sistema operativo;
* Apache come web server;
* MySQL come DBMS;
* PHP o Python come linguaggio di programmazione.

Oltre a ciò, LAMP permette l’installazione di un Content Management System (CMS), un programma che permette la gestione delle pagine web in modo semplice.

## Gestione dei volumi e degli snapshot

Un volume è un blocco di storage simile a una penna USB ma in cloud, ciò vuol dire che si può “inserire” in una VM, modificarne il contenuto, “staccarlo” e usarlo da altre parti. Un volume può essere cancellato se e solo se non è attaccato a nessun volume.

Una cosa molto importante sui volumi riguarda il loro accesso: questo infatti è esclusivo e al momento non c’è modo di ottenere degli accessi concorrenti.

Uno snapshot indica una versione di un volume in un dato momento del tempo, questo risulta utile per i backup e/o mantenere la continuità del lavoro in caso di guasti. Da uno snapshot infatti è possibile ricavare un volume da attaccare in seguito a una macchina virtuale.

## Scaling

Lo scaling indica la possibilità di un sistema distribuito di aumentare o diminuire le risorse per sopperire al carico di lavoro, esso può essere:

* verticale quando si aumenta la potenza di una stessa macchina per sopperire al traffico;
* orizzontale quando vengono create più istanze di una macchina virtuale, tutte con la stessa potenza.

## Load balancing

Il load balancing è un metodo utile per distribuire il carico tra più server, quindi evitare che solo uno di questi faccia tutto e rischi sovraccarichi.

## Baremetal

A volte le VM classiche non bastano a causa delle specifiche limitate e delle risorse condivise. Cosa si può fare? Si può sfruttare il baremetal, un accesso a un server fisico dedicato in maniera esclusiva. L’accesso a questi server dedicati è a tempo: ogni utente che sottoscrive ha un tempo di utilizzo in cui può utilizzare il baremetal in maniera esclusiva, al termine la macchina verrà passata all’utente successivo.

## Map Reduce

Map Reduce è un design pattern adatto al processing di grandi quantità di dati pubblicato da Google nel 2004. Tutto ciò avviene in parallelo dividendo il lavoro in task indipendenti.

Questo design pattern funziona come segue:

* divide l’input in parti più piccole;
* si esegue il codice della Map su ogni parte;
* si esegue uno o più volte il codice della Reduce per unire tutti i risultati.

Questo modello si presta bene all’high performance computing tradizionale dal momento che ogni CPU utilizzata processa in parallelo una parte di un problema più grande.

L’HPC tradizionale però presenta i seguenti problemi:

* i dischi hanno un throughput limitato;
* il numero di core per ogni server/OS è limitato e ciò limita anche il throughput;
* Possono esserci problemi di scalabilità;
* L’affidabilità viene meno dato che non verrebbero completati i task in caso di guasti.

Cosa si fa in questi casi?

Si distribuisce il map reduce! Un esempio di map reduce distribuito è Hadoop, il quale è:

* scalabile dato che vengono aggiunti dei nodi per incrementare sia lo storage e il recupero, sia le performance;
* affidabile, infatti più nodi incrementano questa proprietà;
* tollerante ai guasti dal momento che i dati vengono replicati e processati in modo distribuito.

## Hadoop

L’ecosistema di Hadoop è composto da tre livelli:

* il livello di storage;
* il livello di scheduling;
* il livello di esecuzione, cioè Map Reduce.

## Livello di storage

Il livello di storage é HDFS, un file system distribuito sui vari nodi (le VM) del sistema, precisamente i file vengono divisi in blocchi da 64 MB e distribuiti/replicati in ogni nodo.

Ogni blocco presenta un checksum che serve per rilevare la corruzione durante la trasmissione dei dati.

Quando un nodo si guasta, le copie addizionali vengono ricreate negli altri nodi.

Sempre in questo livello è presente un nodo speciale che tiene traccia della distribuzione dei blocchi, ognuno di essi è identificato che dal nodo, dal cluster e dal datacenter. Solitamente ce ne sono due copie per ogni cluster più una terza in un altro datacenter.

Il livello di reduce invia delle funzioni ai nodi in cui sono presenti i dati:

* la funzione di Map parsifica il file di input e raccoglie ogni istanza di un valore V per ogni chiave K, essa viene fatta in automatica dal pattern;
* la funzione di Reduce riassume un set di valori V per ogni chiave K, il come queste coppie vengono ridotte in un set viene deciso dai programmatori.

## Livello di scheduling e di esecuzione

Il livello di scheduling di Hadoop è YARN e presenta un job tracker il quale decide come distribuire i vari job e tracciare i loro progressi. Ogni job viene diviso in task indipendenti i quali vengono inviati ai vari nodi, qui vengono gestiti dai task tracker (uno per ogni nodo) che li eseguono .

Quando un job fallisce, i task tracker ricominciano il task in un nuovo nodo, lo stesso task può inoltre essere sottomesso da più nodi e aspettare quello che finisce prima.

Il livello di esecuzione è quello che implementa MapReduce, esso ha lo scopo di inviare le funzioni di processing ai nodi contenenti i dati.

Precisamente, il livello di esecuzione è formato da:

* una funzione di Map che parsifica l’input in un insieme di oggetti key-value;
* una funzione di shuffle col compito di raggruppare tutti i key-value aventi la chiave uguale;
* una funzione di reduce che riduce tutti gli oggetti key-value con la stessa chiave in uno unico.

## Applicazioni del map reduce

Il map reduce è utilizzato in diversi ambiti, alcuni di questi sono:

* il machine learning e il data mining;
* l’indirizzamento di documenti o pagine web;
* contare la frequenza di parole in insiemi di documenti;
* processare file di log o archivi di immagini.

Tutte queste applicazioni hanno in comune il fatto che i file/blocchi vengono processati in modo indipendenti e vengono facilmente uniti, inoltre sono scalabili per il numero di nodi e CPU e la dimensione dei dati.

## Limiti

Il pattern MapReduce è stato sviluppato per jobs in stile batch, di conseguenza non sono presenti interazioni. Inoltre ogni jobs ha esattamente tre funzioni associate: una di map, una di shuffle e una di reduce. Dopo ogni step, i risultati devono essere scritti in HDFS.

Non tutti i processi possono essere facilmente scomposti in tre step.

Questo paradigma potrebbe essere causa di overhead o latenza.

## Apache Spark

Spark è un framework che giro su HDFS, esso rimpiazza il livello di esecuzione di Hadoop per provare a risolvere i limiti citati prima, infatti gli obiettivi sono:

* il processing parallelo dei job;
* l’interattività;
* il supporto a MapReduce ma anche di SQL, l’analisi dei grafi e del machine learning.

## Architettura di Spark

L’architettura di Spark è formata da 4 livelli:

* il data Storage, di solito HDFS o AWS S3;
* le API nei vari linguaggi di programmazione;
* Il gestore delle risorse, YARN o Mesos;
* La shell di Spark;
* I tool SQL per la linea di comando.

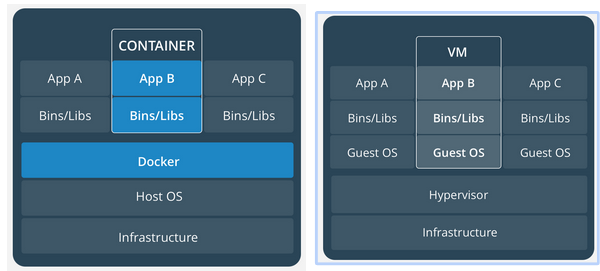
Una particolarità di SPark è il Resilient Distribuited Datasets (RDDs), l’unità di storage fondamentale, essi sono distribuiti, partizionati, in cache e tolleranti ai guasti. Inoltre sono replicati su più nodi.

## Container

I container sono un modo per pacchettizzare le applicazioni in modo da eseguirle e spostarle facilmente tra macchine differenti, essi possono contenere tutto ciò che l’applicativo ha bisogno.

I container sono comparabili alle classiche macchine virtuali, rispetto a questo però sono più light.

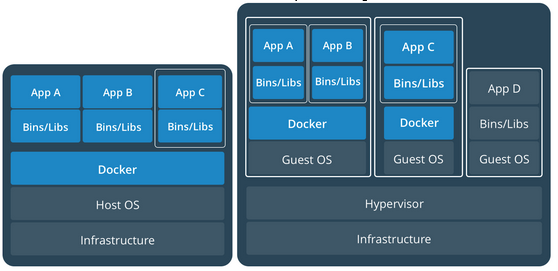
Confrontandole, VM e container hanno entrambe il compito di isolare l’applicazione e le sue dipendenze in un’unità che può girare ovunque, rimuovendo quindi il bisogno di hardware fisico e utilizzando le risorse in modo più efficiente.

Dove stanno le differenze? Ebbene, le differenze stanno nell’architettura.

Nella foto a destra, si può vedere come container e VM utilizzino lo stesso ragionamento ma in maniera differente. Infatti, i container isolano la sola applicazione con le sue dipendenze, tutti questi vengono gestiti da uno strato Docker che funge da hypervisor.

La leggerezza dei container è quindi data dal fatto che questi girano su una singola macchina e che condivida il kernel del sistema operativo della macchina, ciò significa che presenta una velocità maggiore e un risparmio in termini di CPU e RAM.

Inoltre le immagini dei container condividono il file system, ciò si traduce in un utilizzo del disco minore.

In termini di sicurezza, i container permettono di isolare le singole applicazioni l’une dalle altre e anche rispetto all'infrastruttura. Questo isolamento non è relativo all’intera macchina bensì al singolo container.

In ogni caso, Container e VM non si autoescludono l’un l’altro, infatti è possibile avere delle macchine virtuali che fanno girare dei container all’interno.

## Docker

DOcker è un progetto opensource basato sui container di Linux, infatti utilizza le feature del kernel come namespace e gruppi di controllo per creare i container.

Esso è più famoso dal momento che:

* E’ il più facile da utilizzare, infatti una volta buildato gira ovunque;
* è leggero e veloce, il container infatti è una sandbox;
* Esiste una repository in cloud con molte immagini docker disponibili (il suo nome è DOcker Hub);
* E’ modulare e scalabile, infatti è facile dividere un’applicazione in differenti container.

L’architettura di DOcker è composta da un client il cui compito è ricevere i comandi dalla CLI, esso li inoltre al Docker daemon che si trova all’interno del Docker Host, esso si occupa della generazione dei container sulla base dell’immagine selezionata.

## Kubernetes

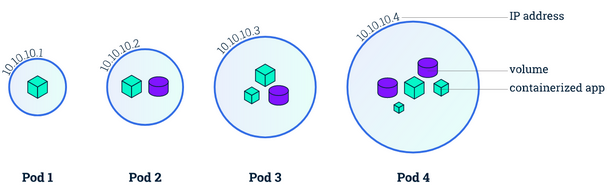
Kubernetes è una piattaforma che permette di orchestrare container, gestendo lo scheduling, il load balancing, la distribuzione, eccetera.

Esso semplifica:

* l’esecuzione dei container su macchine differenti;
* il mantenimento di uno storage consistente tra più istanze di una stessa applicazione;
* lo scaling per l’aggiunta/rimozione delle risorse quando il carico di lavoro cambia;
* il load balancing del carico di lavoro;
* la gestione del fail over, facendo girare un nuovo container se uno di questi smette di funzionare.

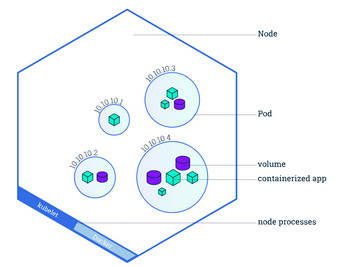
## Concetti chiave di Kubernetes

I pods sono un’astrazione ad alto livello dei container, precisamente un insieme di questi i quali lavorano insieme per un'applicazione specifica. I container in un pod possono comunicare facilmente tra loro e scalare insieme in base al carico, oltre che condividere memoria (in termini di volumi), risorse di rete e indirizzo IP. Un’esempio è un’applicazione LAMP in cui ogni componente è un container.

I nodi sono un’astrazione dell’hardware e possono essere server fisici o macchine virtuali. 

Ogni nodo contiene un dato numero di pods, quindi tutti i container girano sulla stessa macchina.

La comunicazione ta utenti e nodi non avviene in modo diretto.

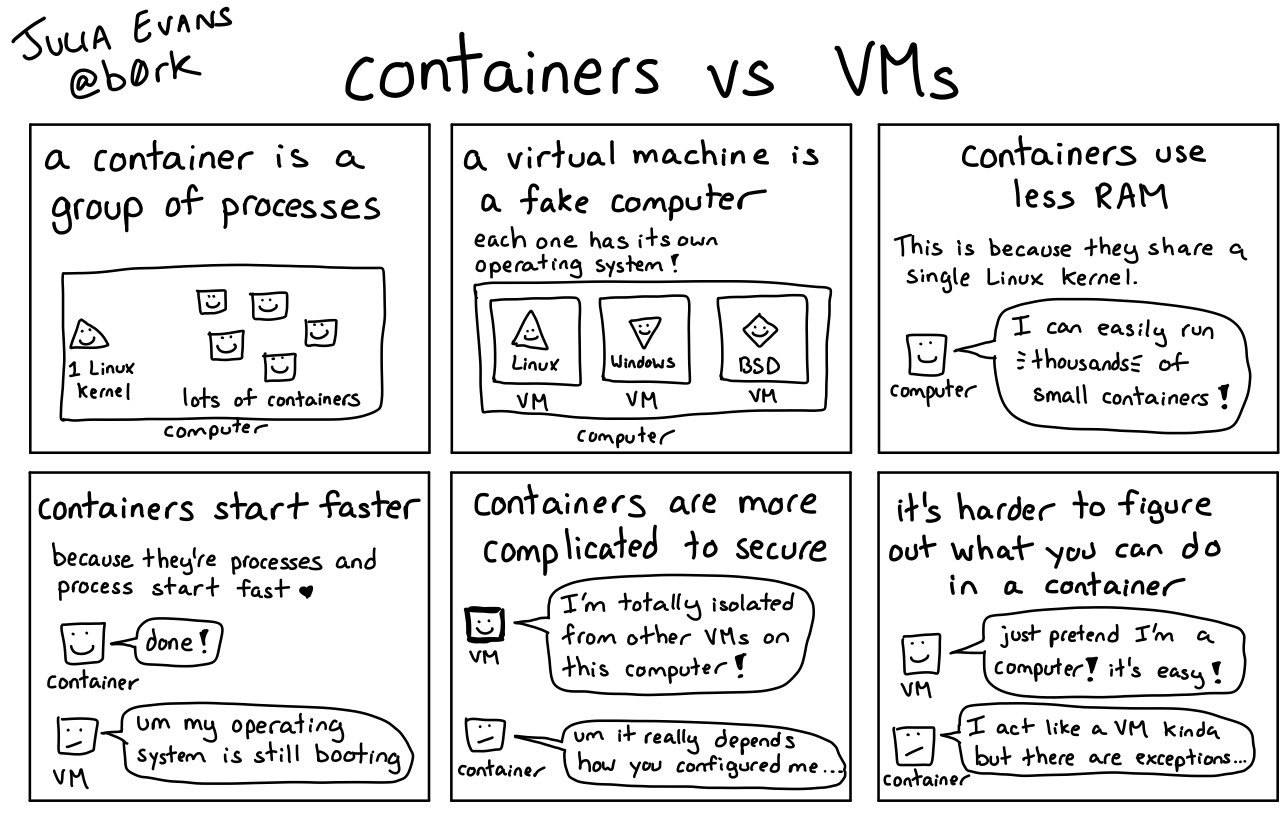
I Cluster sono insiemi di nodi, Kubernetes può aggiungerne o rimuoverne in base al carico di lavoro.

## Deployment su Kubernetes

I deployment sono astrazioni ad alto livello utilizzate dagli utenti per lavorare, esse descrivono lo stato desiderato.

Un deployment specifica il numero di repliche che un pod dovrebbe avere in un cluster, ciò si fa per load balancing e per fail over.

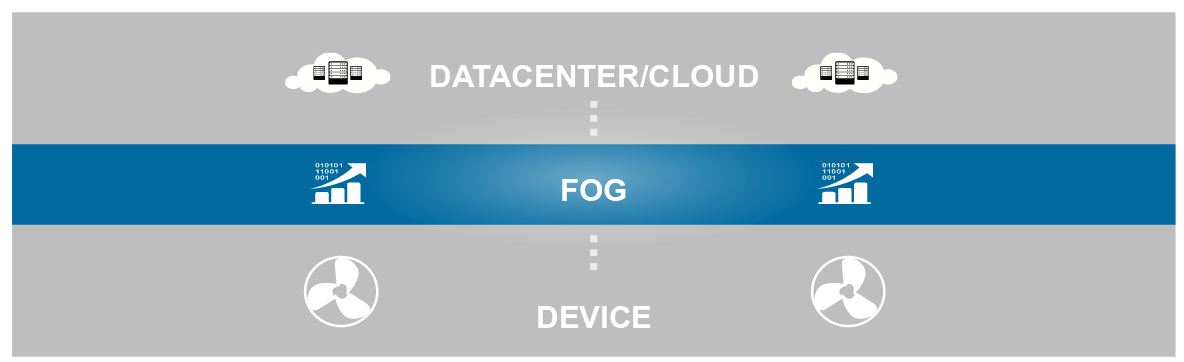
Per fare un esempio, se un nodo ha un pod di un’applicazione che smette di funzionare, Kubernetes creerà un’altra copia del pod su un nodo differente dello stesso cluster.



## Fog computing

Dal momento che L’IoT sta crescendo molto velocemente, non si possono processare alcuni dati sul posto? Sì e in alcuni casi è pure necessario a causa di possibili latenze e bandwidth bassi forniti dai cloud provider, come ad esempio nei sistemi di emergenza.

Il Fog computing ha dato il via all’industra 4.0 ed è una tecnica che permette di avvicinare il cloud alle sorgenti di dati. Un esempio è un sensore di temperatura su un macchina di una fabbrica per segnalare dei guasti imminenti.

Passare i dati dal sensore al cloud in maniera diretta implica un gran consumo di banda e alta latenza, cose che non si vogliono ottenere. Inoltre i dati dell’IoT hanno bisogno di essere protetti adeguatamente.

Cosa si può fare in questi casi?

Si possono processare i dati in un livello intermedio tra la sorgente e il cloud, in questo modo è possibile ottenere bassa latenza e di conservare della banda, oltre che protezione dei dati stessi.

I nodi fog possono essere visti come un’estensione del cloud nella rete, permettendo azioni di durata quasi istantanea che in cloud impiegherebbero anche giorni.

## Interazione tra fog e cloud

Come interagiscono i nodi fog e il cloud? prima di tutto, i nodi fog ricevono i dati dai device IoT in tempo reale, questi vengono analizzati (con un response di pochi millisecondi) e immagazzinati per ½ ore.

Periodicamente, i dati analizzati vengono inviati alla piattaforma cloud, qui i dati di più nodi fog vengono analizzati insieme ad altre risorse, ottenendo una panoramica della situazione.

In base a questa insight, la piattaforma cloud può inviare nuove regole che i nodi fog devono rispettare.

Questo nuovo approccio porta ad avere:

* più agilità in termini di business, dal momento che lo sviluppo di applicazioni fog si fa dove ce n’è il bisogno;
* Una sicurezza migliore con dati più vicini alla sorgente e grazie alle analisi locali;
* Una diminuzione dei costi dato che si conserva banda.