

Cloud computing

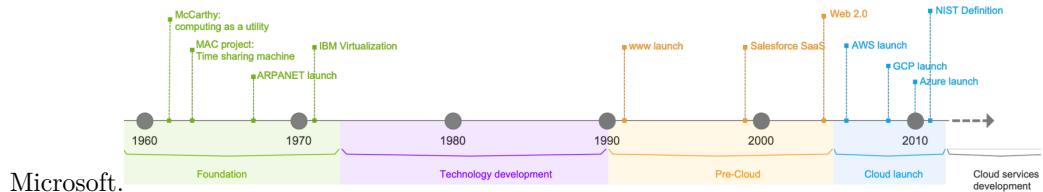
Storia

Idea nata nel '61 da John McCarthy, che consisteva nel fornire capacità di calcolo come un utility, ma mancavano le tecnologie. Negli anni 70 c'è la prima apparizione della virtualizzazione, dove viene diviso un server in chunk con proprie risorse e S.O. verrà chiamato macchina virtuale.

Negli anni 90 iniziano le distribuzioni di software via internet ma per farlo servono grossi datacenter, all'inizio in housing con molte problematiche relative alla scalabilità, infatti se gli utenti crescono deve crescere la potenza ma se scendono la potenza superflua è una spesa inutile (costi di server, manutenzione e montaggio).

Per questo Salesforce inizia a fornire un Software as a Service (SaaS), ovvero un'applicazione utilizzabile da chiunque, via internet, che risiede nei loro server e gestibile da remoto dall'azienda che non si accorge di lavorare in un server condiviso con tanti altri utenti.

Dal 2000 le principali aziende iniziano a creare datacenter per il globo per fornire servizi con latenze basse, nel 2006 Amazon lancia AWS dove noleggia lo spazio dei propri datacenter non utilizzato, seguito successivamente da Google e



Definizione (formale):

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models

Caratteristiche

Secondo il NIST il cloud prevede 5 (almeno) caratteristiche:

1. On-demand, self-service: risorse attive solo quando servono in maniera automatica;
2. Broad network access: si può accedere via internet con client eterogenei;

3. Resource pooling: risorse in comune fra tutti i clienti (multi-tenant) e (ri)assegnate in base alle richieste dei singoli. I clienti non sanno dove sono collocate le risorse, possono scegliere al massimo la posizione (approssimata) del datacenter utilizzato;
4. Rapid elasticity: diminuzione e aumento delle risorse in maniera rapida, elastica e/o automatico
5. Measured service: chi fornisce il servizio monitora e ottimizza l'uso delle risorse.

Successo

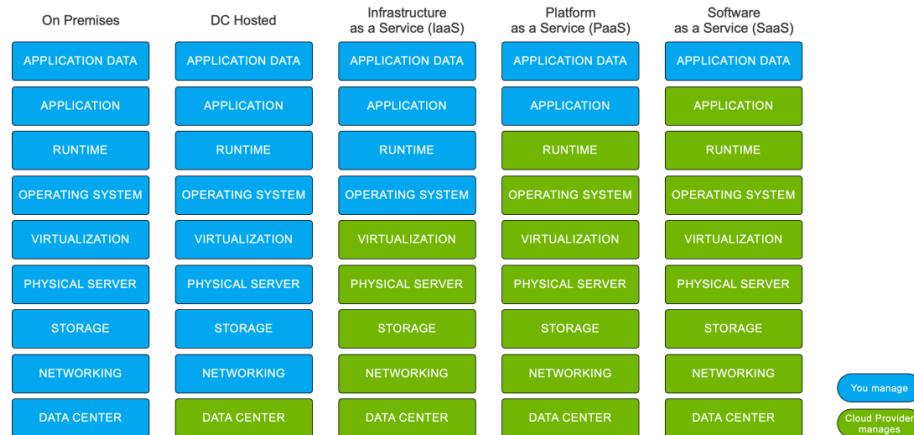
I fattori sono:

- modello di costo: paghi solo quello che richiedi in base al tempo di utilizzo (PAYG) con costi bassi grazie ad economie di scala (+ clienti - costi) e senza investimenti iniziali (CapEx - Capital Expenditure).
- scalabilità: puoi aumentare o diminuire risorse quando vuoi senza problemi visto che i data center e tutto ciò al suo interno è gestito dal provider;
- disponibilità: accedi e usi quando vuoi e dove vuoi.
- affidabilità: il cloud funzionerà sempre e senza errori;
- durabilità: i tuoi dati saranno sempre al sicuro da perdite o corruzione da errori da fuori e da dentro;
- sicurezza: protezione da accessi non autorizzati.

Cloud Service Models

Il provider può fornire le proprie risorse su livelli diversi per questo nascono i modelli di servizio.

Le risorse messe a disposizione sono sempre le stesse ma da modello a modello cambia cosa gestisci tu e cosa gestisce il provider:



Nella parte verde (quella gestita dal provider) qualunque problema o servizio è gestito senza che il cliente se ne debba preoccupare, al contrario la parte blu è di sua responsabilità e quindi ogni problema riscontrato non viene gestito dal provider.

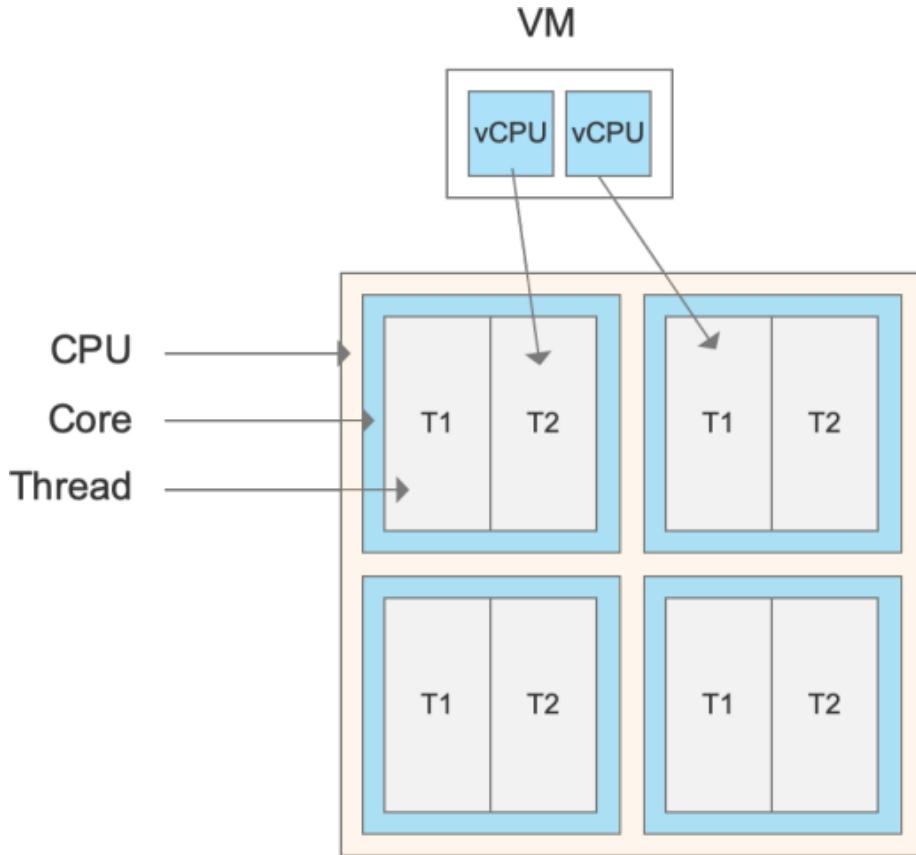
Questo modello dove ognuno è responsabile della propria parte di stack è detta Shared responsibility model.

I vari servizi sono:

IaaS - Infrastructure as a Service

Il provider gestisce tutto ciò a basso livello come la VM, lo storage (virtuale) e il networking. Il cliente può decidere il S.O. e scaricare qualsiasi software nella VM.

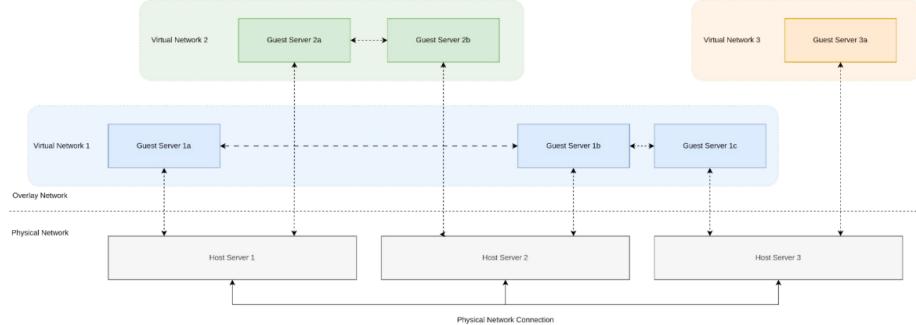
Vengono usate vCPU cioè singole unità di elaborazione:



Il disco invece è astratto in storage pools, combinazione di tecnologie diverse utilizzate come un'unica entità, permettendo l'aggiunta o la rimozione spazio in

maniera arbitraria.

Per isolare logicamente a livello di rete le VM si usano reti virtuali create su quella fisica; questo grazie a protocolli particolari che permettono di creare connessioni virtuali tra nodi virtuali, mappando il tutto sulla rete fisica (Overlay Network).



Google Compute Engine

Soluzione IaaS di GCP, creazione/gestione/configurazione e connessione di VM (istanze) con fatturazione al minuto.

L'istanza è configurabile nei seguenti aspetti:

Machine Type	Preset	Custom		
Operating System	Public Image	Custom Image	Snapshot	
Storage	Standard (HDD)	Balanced SSD	SSD	Extreme SSD
Networking	Default		Custom	

- Machine type: è possibile scegliere una tipologia divise in Series (determina la qtà min/max di vCPU e RAM).

Le serie possono essere:

- Preset: numero di vCPU e RAM già deciso con il nome in questo formato:

e2-standard-32

--	--	--

Serie	Type	vCPU
-------	------	------

- Custom: numero di vCPU e RAM da decidere
- Operating system: l'immagine per caricare il SO è scelto fra i seguenti modi:
 - Public Images: Selezionato da un elenco predefinito di SO.
 - Custom Image: Selezionando un'immagine disco creata precedentemente, anche non in GCP.
 - Snapshot: Selezionando lo snapshot di un PD creato in precedenza

L'images è una copia completa del disco dell'istanza utile per riusare le istanze ma non supporta il disaster recovery; lo snapshot è una copia del contenuto di un singolo Persistent Disk quindi perfetto per il backup (supporto alla copia differenziale, cioè save dei soli dati modificati rispetto allo snapshot prima).

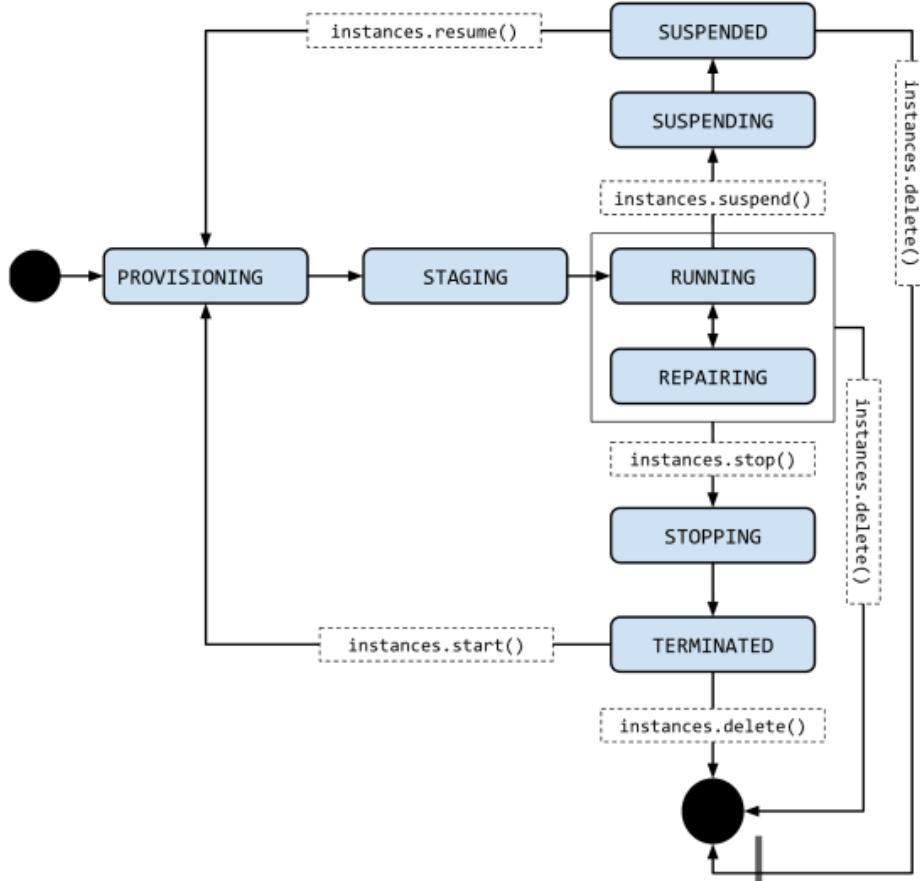
- Storage: l'istanza chiede almeno un disco con possibilità di inserirne altri successivamente.

I dischi che vengono collegati all'istanza sono detti Persistent Disk (PD), in quanto possono sopravvivere alla cancellazione della VM e sono Standard, Balanced SSD, SSD o Extreme SSD.

- Networking: ogni istanza è collegata ad una VPC (Virtual Private Cloud) con di default un indirizzo pubblico per essere accessibile da internet.

Esistono altri tipi di istanze dette spot o preemptible, che possono essere interrotte quando vuoi e fortemente scontate, ma non sempre disponibili e non coperte da SLA (Service Level Agreements).

Ogni istanza ha un proprio lifecycle dove può rientrare in alcuni di questi stati:



- PROVISIONING: Le risorse vengono allocate alla VM che non è ancora in esecuzione. Nessun costo in questa fase
- STAGING: Le risorse vengono acquisite dalla VM che si sta preparando all'avvio
- RUNNING: La VM è in fase di avvio o esecuzione. Viene eseguito lo startup script e avviati tutti i servizi (compreso ssh e rdp)
- REPAIRING: La VM è in fase di riparazione a seguito di un errore dell'infrastruttura Google. In questa fase l'istanza non è accessibile e non genera costi
- STOPPING: La VM è in fase di arresto, a seguito di errore, azione utente o per lo stop di istanze spot
- TERMINATED: La VM è stata arrestata. E' possibile riavviarla o eliminarla
- SUSPENDING: La VM è in fase di sospensione

- SUSPENDED: La VM è in stato di sospensione. E' possibile ripristinarla o eliminarla.

PaaS - Platform as a Service

Viene fornita una piattaforma già pronta all'utilizzo per lo sviluppo software senza la gestione di librerie, S.O., ecc.

Scalabile ,sicuro e con un basso Time to market (tempo che va dallo sviluppo alla commercializzazione di un prodotto).

SaaS - Software as a Service

Servizio più completo e meno personalizzabile, il provider offre un applicativo fatto e finito utilizzabile con i propri dati e in maniera isolata; la manutenzione è a carico del provider.

Es: Google Suit, Dropbox, Teams...

Cloud Deployment Models

Come il cloud viene implementato cambia da azienda ad azienda oppure la stessa azienda può avere policy diverse e quindi diverse implementazioni, per questo esistono 4 modelli di deploy:

Deployment Model	Pro	Contro
Public cloud	<ul style="list-style-type: none"> • Modello pay as you go • Risorse disponibili su richiesta 	<ul style="list-style-type: none"> • Le risorse sono di proprietà del provider
Private cloud	<ul style="list-style-type: none"> • Sicurezza e controllo • Indipendente dal cloud provider 	<ul style="list-style-type: none"> • Costoso • Mancanza di scalabilità in tempi rapidi
Community cloud	<ul style="list-style-type: none"> • Maggiore scalabilità • Condivisione dei costi 	<ul style="list-style-type: none"> • Possibili "buchi" di sicurezza • Possibile mancanza di risorse adeguate
Hybrid cloud	<ul style="list-style-type: none"> • Tutti quelli dei modelli public e private • Ottimizzazione dei costi 	<ul style="list-style-type: none"> • Possibili problemi di integrazione tra gli ambienti

Public Cloud

Quello visto fino ad ora, ne fanno parte i Cloud Service Models, dove chiunque può usare i vari servizi forniti con modalità PAYG.

Private Cloud

Le risorse fornite sono usate solo da una singola organizzazione per uso interno,

possono essere salvate all'interno della stessa azienda o ospitate da un provider esterno che ne assicura la protezione e l'accesso controllato.

Community Cloud

Più organizzazioni con lo stesso scopo (es: agenzie governative) condividono un cloud privato andandolo a rendere più flessibile ed economico come il public mantenendo la sua sicurezza.

Hybrid Cloud

Unione del community al public con la creazione di due situazioni:

- Classic Hybrid Cloud: Viene usato il private cloud per i dati e le applicazioni proprietarie, ed è usato il public cloud per applicazioni non critiche;
- Cloud Bursting Hybrid Cloud: Viene principalmente usato il private cloud ricorrendo al public cloud solo per gestire i carichi di lavoro che eccedono la capacità dell'infrastruttura privata.

Datacenter

Strutture fisiche dove risiedono tutti componenti che rendono il cloud funzionante ed efficace come generazione di corrente (primaria/secondaria), sistemi di raffreddamento e antincendio e sorveglianza; la ridondanza è obbligatoria per aumentare la tolleranza ai guasti.

Struttura fisica

I server e le varie risorse che usano sono aggregate in:

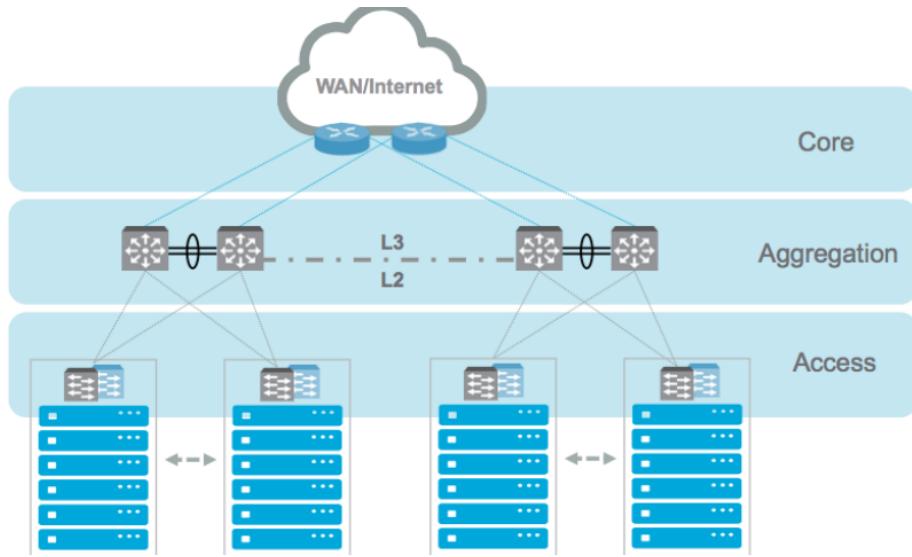
- Rack: contenente CPU, RAM HDD/SSD ed interfacce di rete, i vari rack possono essere inseriti/rimossi in qualsiasi momento in un armadio rack.
- Blade: uguali ai rack per caratteristiche ma più piccoli montati sui chassis che alimentano e forniscono connessione ad ogni blade, per via delle dimensioni generano più calore.

Entrambi possono essere aggregati nei cluster rendendoli identificabili come un singolo sistema garantendo ridondanza (spostando il lavoro da un server ad un altro senza interruzioni) e modularità.

Connettività

Si utilizza la topologia Core/Aggregation/Access layers dove ogni livello ha un compito:

- Access: rete con collegamento diretto ai server.
- Aggregation: rete per lo smistamento del traffico nelle VLAN.
- Core: backbone del datacenter, collega l'aggregation layer con l'esterno.



Per la connettività esterna il data center ha uno o più PoP (Point of Presence), punto importantissimo perché influisce sulla latenza e sulle prestazioni.

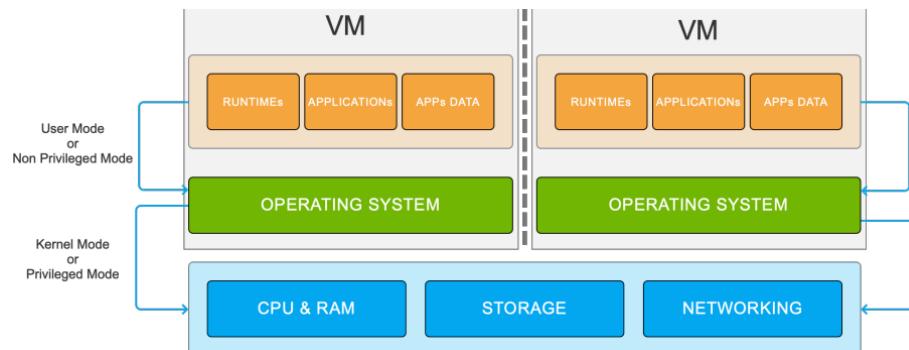
Virtualizzazione

Tecnologie

Nato nel '74 con la pubblicazione del paper: "Formal Requirements for Virtualizable Third Generation Architectures".

L'obiettivo è eseguire sulla stessa macchina fisica più macchine virtuali (VM) ciascuna:

- composta da S.O.
- Isolata.
- Self contained.
- Con risorse condivise.



Questo comporta una ottimizzazione dell'utilizzo hardware (meno server e meno spreco di risorse), riduzione dei costi CapEx e OpEx.

Hypervisor

Per rendere possibile l'utilizzo contemporaneo di più S.O. è necessario creare un nuovo strato tra hardware e software detto Hypervisor che supera il problema di instabilità dato dal Kernel Mode, gestisce e crea le VM e le loro risorse.

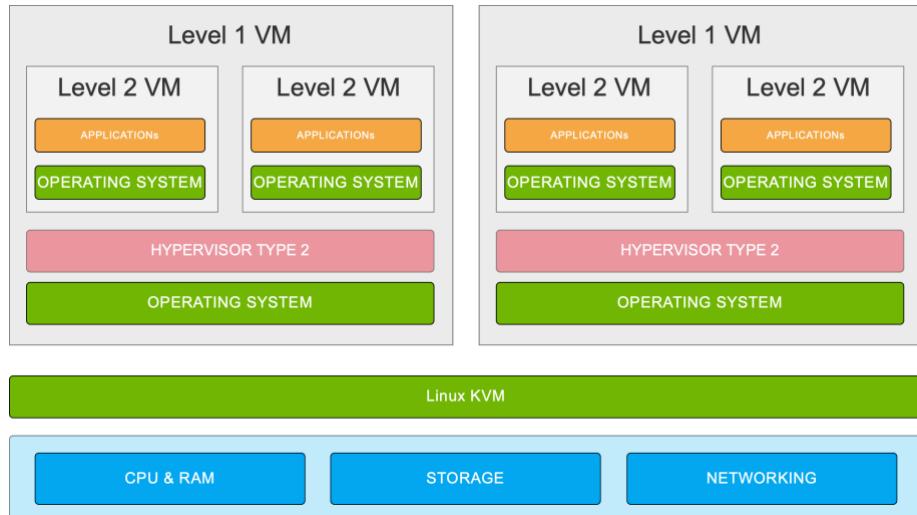
In base a dove posiziono l'Hypervisor cambia il funzionamento e il tipo:

- Type 1 o Bare Metal: installato nell'hardware andando a sostituire il S.O. rendendolo molto efficiente ed ideale per ambienti server.
Si può classificare in:
 - Full virtualization o Emulation: la VM usa una CPU emulata e il suo SO pensando si quella vera effettua chiamate in kernel mode che vengono intercettate e replicate sull'hardware fisico; l'utilizzo di SO standard porta svantaggi ma è comunque possibile.
 - Paravirtualization: le VM usano SO modificati che effettuano chiamate direttamente all'hypervisor (hypercall) che vengono ripetute sull'hardware fisico; migliori performance.
 - Hardware virtualization: grazie all'utilizzo di feature i nuovi processori possono nativamente dividere e isolare le risorse per ogni VM senza quindi l'overhead dell'emulazione.
 - Type 2: è un applicativo classico installato su un SO (host os), meno efficiente (più mediatori) e adatto ad ambienti desktop.

Kernel level virtualization

Non è necessario usare un hypervisor grazie al kernel Linux speciale che gestisce ogni VM come un singolo processo agevolando il ciclo di vita; ora è possibile la nested virtualization.

Usata dal GCP.



BORG

Progetto di Google per l'allocazione centralizzata delle risorse dei data center in maniera scalabile ed efficiente.

Tramite un pannello di controllo si gestiscono un intero cluster assegnando ad ogni worker un task, rispondendo ai cambiamenti nello stato del cluster e monitorando lo stato di task e worker per riavviare le operazioni se qualcosa va storto.

La scalabilità sta nel fatto che il pannello NON esegue i task direttamente.

OMEGA

Progetto più recente nato per essere più flessibile e adatto alle nuove applicazioni come Batch jobs (machine learning), Servizi web con alto traffico o bassa latenza.

Approccio maggiormente decentralizzato nella gestione dell'allocazione di risorse dove ogni macchina è responsabile di gestire i propri task e comunicare con le altre per coordinare l'allocazione di risorse.

VPC

Virtual Private Cloud (VPC) è il servizio di networking virtuale che fornisce la connettività alle risorse cloud, tra cui le istanze compute engine.

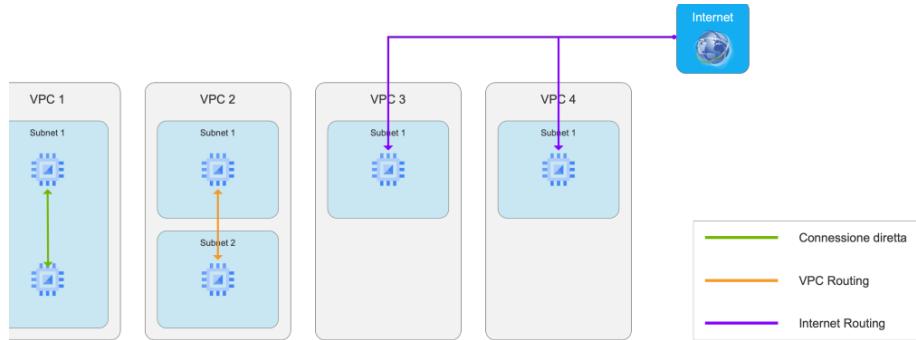
Essendo create in software non hanno i limiti delle reti fisiche.

Esistono solo all'interno dei progetti e nessun progetto non ne può avere, possono essere condivise.

Alla VPC sono associate:

- subnet: sottoreti con range di indirizzi IPv4 privati associati;
- regole di routing: per gestire le comunicazioni fra VM di subnet diverse;
- regole di firewall: per gestire traffico in entrata e uscita.

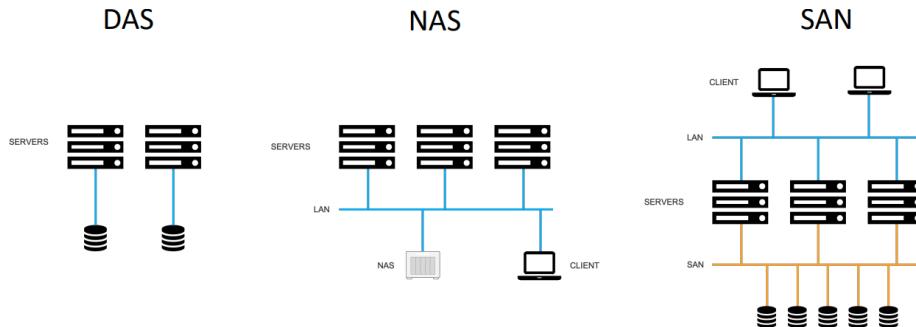
Le comunicazioni possibili sono mostrate nel grafico:



Storage di dati

Storage Pools

i provider offrono diversi tipi di storage che a livello fisico non sono altro che numerosi HDD o SSD gestiti per comparire come un'unica entità logica, la gestione può essere fatta in diversi modi:



DAS - Direct Attached Storage

Dischi fisicamente collegati ad un server tramite connessioni ad alta velocità e utilizzabili solo al server a cui sono collegati.

Questo porta alte prestazioni e bassi costi a scapito della flessibilità e con un Single Point Of Failure.

NAS - Network Attached Storage

Dischi collegati tramite rete e accessibili tramite IP, vengono ospitati in dispositivi con capacità di calcolo e un proprio sistema operativo per gestire:

- policy di accesso
- ridondanza RAID
- condivisione dati tramite rete

SAN - Storage Area Network

Dischi collegati al server tramite una rete dedicata che utilizza la fibra, per questo appaiono come dischi locali.

Rispetto al DAS sono flessibili e anche resilienti grazie al backup o snapshot.

Data replication

Si tratta di replicare gli stessi dati su più nodi portando ad un aumento di prestazioni (eliminazione di colli di bottiglia), disponibilità e sicurezza avendo copie di backup ma anche diminuzione di latenza portando i dati vicino agli utenti che ne hanno bisogno.

Le strategie per replicare sono:

Full replication

Ogni nodo ha una copia di tutto il dataset (ogni nodo è uguale), ottimo per aumentare le performance di lettura e la sicurezza ma pessimo per la scrittura visto che le modifiche dovranno essere propagate a tutti i nodi (portando ad un overhead sulla rete e maggiori tempi di esecuzione).

Partial replication

Solo alcuni nodi hanno le repliche portando ad una propagazione delle modifiche più snella.

Diminuiscono le performance visto che dovrò cercare il nodo specifico e la sicurezza (meno backup).

Consistenza

In entrambe le strategie si genera il problema dell'inconsistenza dei dati perché bisogna garantire che tutte le repliche siano uguali.

Per risolvere abbiamo due approcci:

Strong consistency

In ogni istante tutti i nodi sono uguali, per questo si bloccano tutte le operazioni fino a propagazione conclusa, con degrado rapido delle performance.

Weak consistency

In questo caso le operazioni sui nodi sono possibili nonostante la propagazione delle modifiche, questo può portare ad inconsistenza e corruzione ma le performance non ne risentono.

Read replicas

Si utilizza una copia primaria su cui effettuare sia le operazioni di lettura che di scrittura e una o più copie secondarie su cui effettuare solo le operazioni di lettura.

Tipologie di storage

Un public cloud provider offre alcuni se non tutti questi storage:

Block Storage

Viene memorizzato tutti in strutture di dimensioni fissa (block) in questo tipo di storage vengono installati software o SO che richiedono accesso diretto a questi block.

Utilizzato da VM per il boot disk e secondary disk.

File Storage

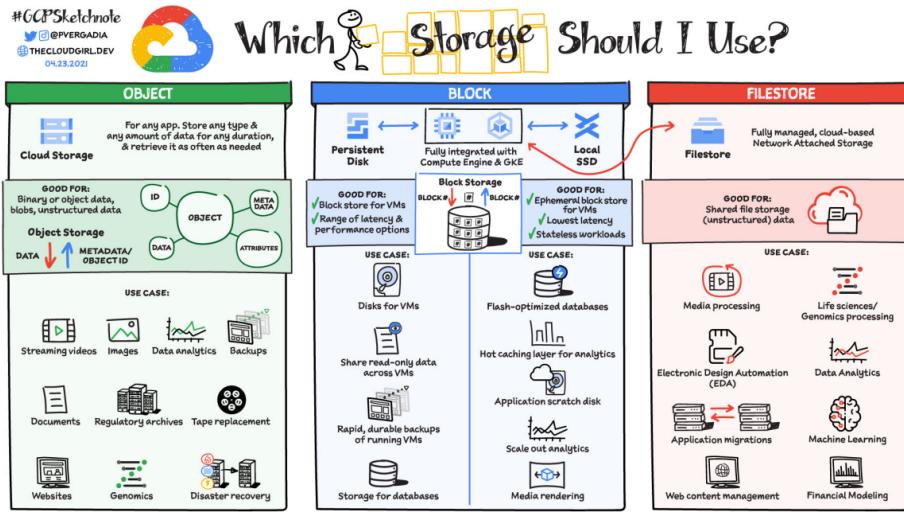
Forniscono file system accessibili via rete e servono per realizzare soluzioni NAS (vedi su), sono indipendenti rispetto al file system delle VM garantendo condivisione di file tra più applicazioni.

Object Storage

Dati memorizzati in termini di Oggetti o Blob, che di solito sono file, ma in maniera non analoga ad un file system (cartelle).

Ogni oggetto ad un indirizzo con cui si può accedere anche senza VM; l'accesso è pubblico o gestito da policy.

Gli oggetti sono solo sovrascrivibili ed è possibile avere una storicizzazione delle versioni.



Cache

I dati vengono mantenuti qui per un accesso rapido e con latenze $<$ del millisecondo; ma essendo volatili ad un riavvio del server i dati, a meno che non siano salvati nel system of truth, vengono persi.

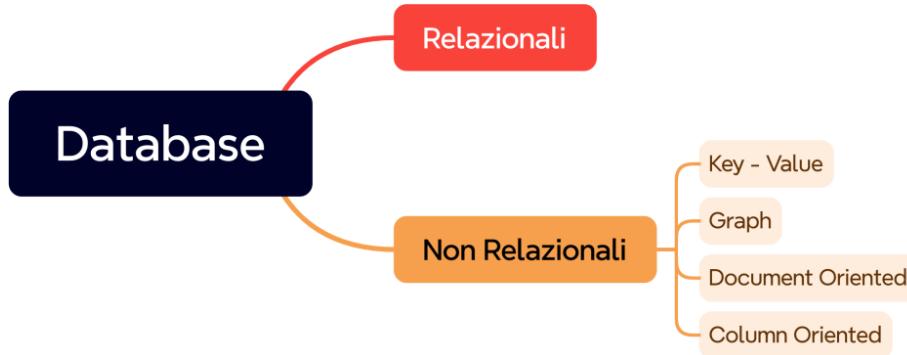
Capita che vadano fuori sync rispetto al system of truth, se le modifiche non vengono replicate correttamente.

Database

Un Database Management System (DBMS) è una applicazione che si occupa della memorizzazione e gestione dei dati, con le seguenti caratteristiche:

- Gestisce grandi quantità di dati con particolare attenzione all'efficienza.
- Gestisce la persistenza dei dati garantendo la fault tolerance.
- Gestisce la condivisione dei dati garantendo il controllo degli accessi e della concorrenza
- Definisce uno o più linguaggi di programmazione per interagire con i dati
- Divide lo schema fisico dallo schema logico
- Prevede uno o più sistemi per effettuare il backup e il ripristino dei dati
- Può prevedere la possibilità di replicare i dati su più server, per aumentare la disponibilità e le prestazioni

I DBMS sono classificati in due famiglie:



Relazionali

I dati sono modellati tramite il concetto di Relazione rappresentato in forma tabellare, dove ad ogni occorrenza di dominio si associa un nome univoco, detto attributo. Questo permette di aumentare la leggibilità, di migliorare l'espressività e di rendere irrilevante l'ordine con cui si considerano i domini. Ad ogni relazione è inoltre associato un nome univoco.

Schema					
Nome della relazione	Anagrafica				
Attributi	Codice	Cognome	Nome	DataNascita	Email
	1	Rossi	Luigi	01/07/1980	luigi.rossi@gmail.com
Tupla o record	2	Bianchi	Marco	23/04/1977	marco.bianchi@gmail.com
	3	Verdi	Giuseppe	01/09/2000	giuseppe.verdi@gmail.com

Istanza

Per essere considerati validi i dati devono seguire dei vincoli di integrità, cioè, proprietà che deve essere soddisfatte dalle istanze.

Tutta la parte dei vincoli QUI (slide 38 - 43)

Per lavorare sulle tabelle si usa un linguaggio dichiarativo detto SQL che tramite query è possibile fare le seguenti operazioni:

- DDL - Data Definition Language: Per creare/modificare/cancellare database e tabelle
- DML - Data Manipulation Language: Per interrogare e modificare i dati presenti nelle tabelle
- DCL - Data Control Language: Per gestire i diritti di accesso degli utenti alle tabelle

Il vantaggio di questo modello è il concetto di transazione, cioè l'insieme di operazioni correlate secondo la proprietà ACID:

- Atomicity: Una transazione è atomica, ovvero è trattata come un'unità indivisibile. In caso di errore l'intera transazione viene annullata (rollback)
- Consistency: La transazione lascerà il database sempre in uno stato consistente
- Isolation: Le transazioni in esecuzione concorrente non interferiscono tra di loro
- Durability: L'effetto di una transazione terminata con successo è permanente.

Non relazionali

Non usano le relazioni per modellare i dati e ne esistono di più tipi:

- Key-Value: dati archiviati in vettori ciascuno acceduto da una specifica chiave
- Graph: dati rappresentati in termini di entità e relazioni tra loro, molto usati per social network o strumenti di analisi
- Document Oriented: modello in cui ogni dato è trattato come un documento JSON che può essere annidato ad altri in maniera gerarchica
- Column Oriented: dati archiviati in tabelle con colonne che variano dinamicamente in base alle tipologie di dati

Non imponendo uno schema rigido e vincolato, risultano più performanti ma non garantiscono allo stesso modo l'integrità dei dati. Hanno alcune caratteristiche comuni:

- Basically Available: garantisce che il database risponderà alle richieste, ma non per forza con la versione dei dati più recente
- Soft-state: lo stato del database può cambiare nel tempo anche senza input (esempio per sincronizzare dati)
- Eventually consistent: il database non sarà subito consistente. Potrebbero esserci ritardi tra le operazioni di scrittura e la loro sincronizzazione su tutte le repliche

Storage GCP

Le categorie viste in precedenza, anche se con nomi diversi, vengono offerti come servizi da GCP, eccoli:

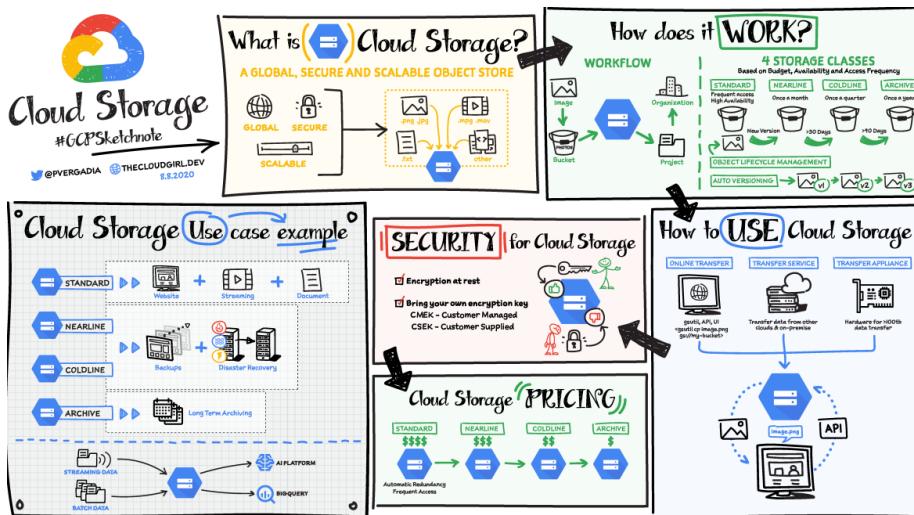
Categoria Storage	Servizi GCP
Block Storage	Persistent Disks
File Storage	Cloud Filestore
Object Storage	Cloud Storage
Cache	Cloud Memorystore
Database Relazionali	VM Instances Marketplace Cloud SQL Cloud Spanner
Database NON Relazionali	Cloud Bigtable (column based) Cloud Firestore (document based)

Vediamone alcuni:

Cloud Filestore

Servizio di NAS utilizzabile con VM o cluster Kubernetes, utilizza NFSv3 per usare i filesystem di rete con performance alte adatte a carichi di lavoro intensivi. Scalabile fino a 100TB.

Cloud Storage



Servizio Object Storage con oggetti di qualsiasi tipo, immutabili e organizzati in bucket univoci a livello globale; è possibile avere uno storico delle versioni precedenti (identificati da un numero di generazione).

Oggetti accessibili tramite policy e url univoco.

Cloud storage è adatto a memorizzare oggetti gestiti come una "unica unità", ovvero oggetti che vengono sempre letti e scritti nella loro interezza. Non è adatto ad oggetti che richiedono un accesso parziale.

Può essere regionale o multiregionale, spazio scalabile all'infinito.

I dati possono essere archiviati in 4 classi di archiviazione, che differiscono per tempi di accesso e per costi:

- Standard: È la classe da utilizzare per accessi frequenti
- Nearline: È la classe per accessi che avvengono una volta al mese
- Coldline: È la classe per accessi che avvengono una volta ogni 90 giorni
- Archive: È la classe per accessi che avvengono una volta all'anno

Le classi Nearline, Coldline e Archivie, oltre al costo per lo spazio utilizzato, prevedono un costo per ogni singolo accesso. La classe può essere cambiata per ogni oggetto anche in maniera automatica tramite policy dette lifecycle management.

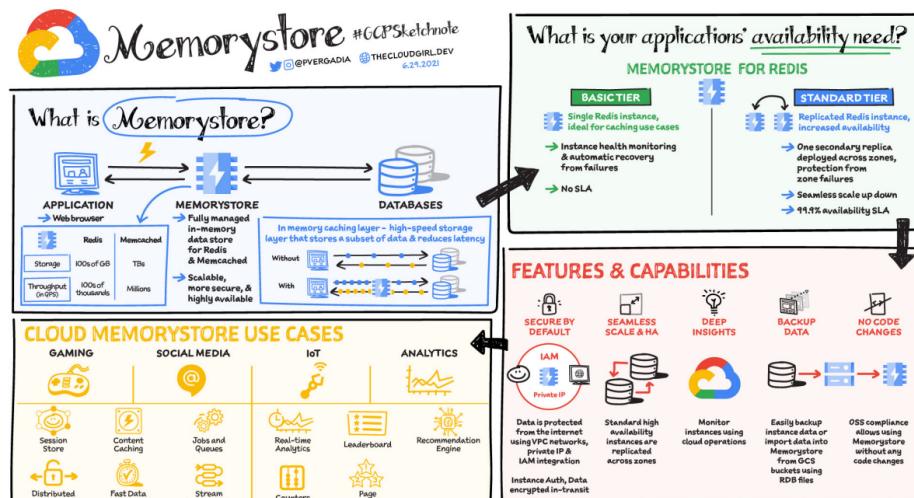
Cloud Memorystore

È il servizio di cache in-memory, per aumentare le performance di accesso ai dati più frequentemente usati

Supporta due dei principali sistemi open source di caching:

- Redis
- Memcached

È sufficiente specificare la dimensione della cache, lasciando alla piattaforma la gestione di tutti gli aspetti amministrativi



Cloud SQL

È il servizio che mette a disposizione i principali DBMS relazionali in modo completamente gestito da Google.

Le attività di amministrazione, quali backup, aggiornamenti e gestione del failover sono completamente gestite da Google. L'unica attività amministrativa necessaria riguarda la definizione degli utenti e dei relativi diritti di accesso.

Progettato per gestire le repliche ed essere a bassa latenza, dati crittografati at-rest e in-transit.

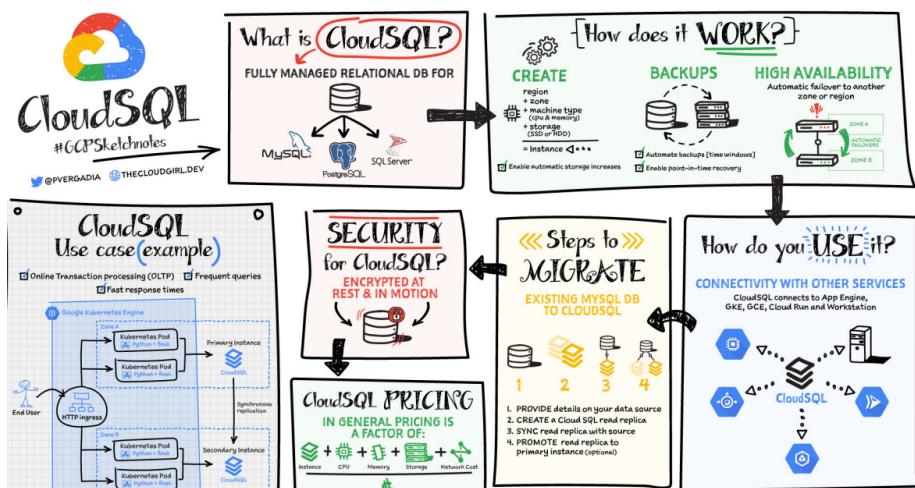
La replica è gestita tramite istanza primaria e fino a 10 istanze secondarie in sola lettura e ospitate in:

- Stessa region ma zona differente (Read Replica)
- Region differente (Cross Region Read Replica)
- On Premises (External Replica)

Le repliche possono essere "promosse" a istanza primaria, con eccezione delle External Replica.

La loro utilità sta nei seguenti scenari:

- Aumentare le performance, dividendo il carico di lettura tra tutte le repliche
- Migrare database da una region ad un'altra, creando una nuova replica nella nuova region e promuovendo a istanza principale
- Gestire la High Availability, promuovendo una replica ad istanza primaria in caso di fallimento dell'attuale

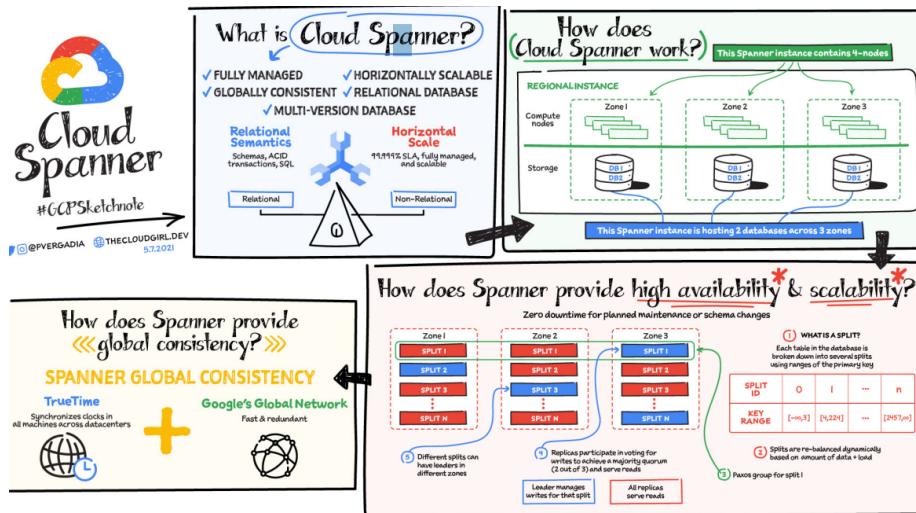


Cloud Spanner

È il database relazionale "cloud-native" e distribuito globalmente, combina i vantaggi dei database relazionali e di quelli noSQL:

- Forte consistenza dei dati
- Transazioni ACID
- Scalabilità orizzontale (repliche)relazionale "cloud-native" e distribuito globalmente.

Come Cloud SQL c'è la crittografia at-rest e in-transit, supporta comunque SQL standard.



Cloud Bigtable

Servizio completamente gestito di database NoSQL column based perfetto per la gestione di Terabyte e Petabyte di dati, garantendo bassa latenza e alto throughput; non prevede le transazioni per cui deve essere usato in contesti in cui non c'è tale necessità

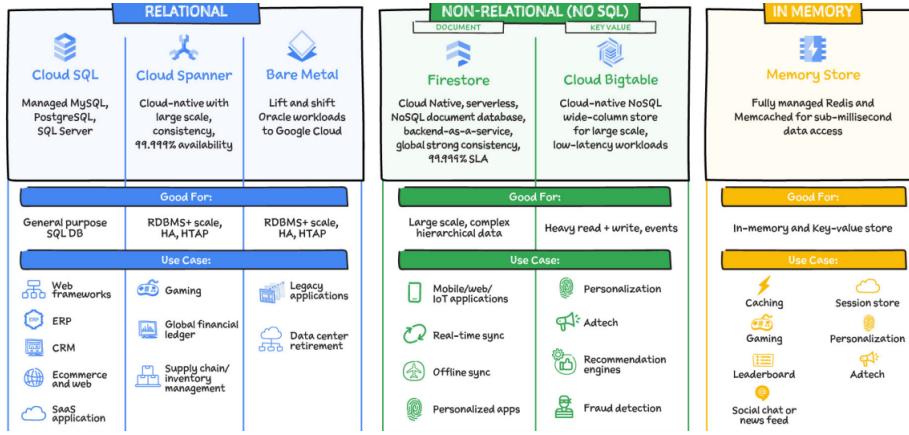
È un servizio regionale con la possibilità di replica automatica dei dati.

Scala orizzontalmente in maniera automatica, pensato per applicazioni real time e analisi di dati.

Cloud Firestore

È un servizio di database NoSQL basato su documenti. progettato per essere real time, avere alte performance e per scalare automaticamente.

Gestisce le transazioni e prevede un linguaggio di interrogazione SQL Like precedentemente noto come Cloud Datastore, e può essere configurato in modalità Datastore.



Scalable Computing

Scalabilità

Si intende l'aggiunta di risorse in un ambiente di lavoro per adeguarlo al carico di lavoro (rapid elasticity), la scalabilità può essere verticale/orizzontale.

Verticale

Scalabilità molto versatile e applicabile a qualsiasi applicazione, dove si potenzia semplicemente il numero di risorse già esistenti (aumento di vCPU e RAM).

La versatilità si paga con il dover riavviare una risorsa prima di poterla cambiare configurazione o lo spreco di risorse dovuto ad una diminuzione di risorse non tempestiva.

Orizzontale

In questo caso si duplicano le risorse (creazione di una seconda VM uguale alla prima) andando poi a dividere il lavoro.

Questo tipo di scalabilità non è possibile con le applicazioni che mantengono uno stato applicativo, applicazione stateful, visto che la duplicazione creerebbe copie degli stessi dati con successivo disallineamento.

Per questo esistono le applicazioni stateless, perfette per la scalabilità orizzontale, dove non c'è nessun tipo di dato gestito (come una webapp che eroga solo HTML / CSS senza un backend).

L'applicazione stateful può essere scalata orizzontalmente andando a centralizzare lo stato applicativo, condividendolo con tutte le istanze.

Horizontal Autoscaling

Quando il provider scala in automatico orizzontalmente una risorsa tramite policy definite dall'utente, normalmente su delle soglie.

La scalabilità avviene su gruppi di risorse, es: gruppo di VM dove se ne aggiunge una nuova.

In GCP si chiama Managed Instance Group, dove si crea un set di VM identiche detto instance group che può essere:

- Managed (MIG): configurati per scalare automaticamente orizzontalmente e per bilanciare il traffico tra le istanze; si possono creare zonal MIG (tutte le istanze nella stessa zona) o regional MIG (istanze sparse su una regione e più resilienti).
- Unmanaged: le VM appartenenti al gruppo possono essere differenti tra di loro e per questo l'istanza va creata manualmente e anche le varie scalature.

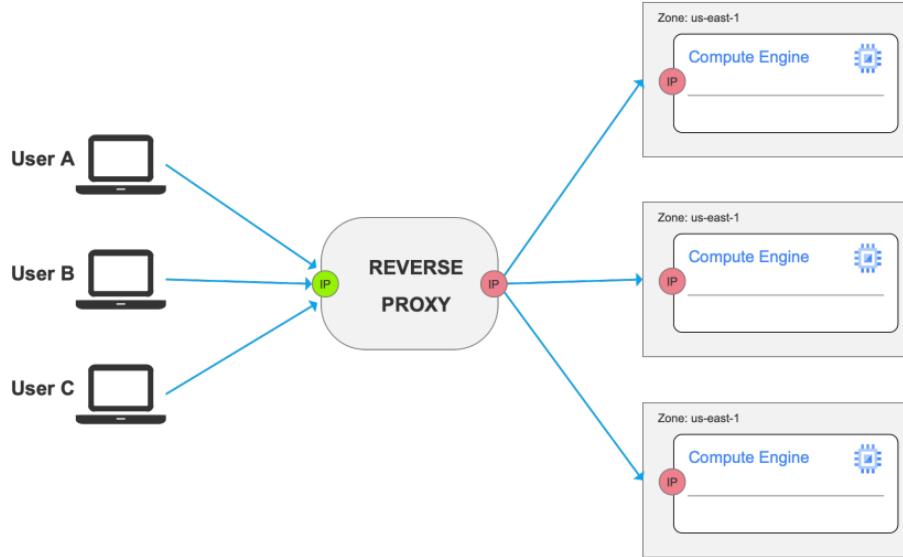
Quando si configura l'autoscaling delle MIG bisogna tenere conto i tempi di boot/shutdown delle VM, infatti se il tempo tra due controlli di autoscaling è troppo piccolo ci si può trovare nella situazione in cui una VM appena aggiunta non è ancora completamente operativa (aggiunta non voluta di più istanze).

Le MIG effettuano degli health checks, dove si mantengono in running le istanze e si riavviano VM bloccate, con anche aggiornamenti a rotazione per evitare che tutte le VM si blocchino nello stesso momento per aggiornare.

Reverse Proxy

Andando a scalare orizzontalmente si crea il problema dell'accesso diverso ad ogni istanza, infatti sarebbe opportuno avere un indirizzo IP per ogni VM e informare i client di tutti i nuovi IP ad ogni scaling.

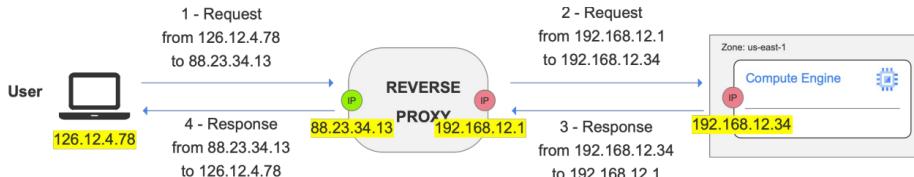
Per questo è nato il reverse proxy, un intermediario che accetta tutte le richieste in entrata dai client e restituisce le varie risorse andandole a prendere dalle varie istanze senza che il cliente sappia da chi arriva la risposta; il proxy dovrà sempre sapere quante sono le VM e quindi informato ad ogni scalo orizzontale.



Es: Nginx e Apache

Il proxy sarà l'unico ad avere IP pubblico andando a migliorare la sicurezza, gestendo i vari accessi ed eventualmente effettuare elaborazioni sul traffico.

La connessione tra client e VM essendo intermediata dal proxy viene divisa in due e sarà proprio lui ad effettuare la terminazione della connessione.



Il reverse proxy presenta anche dei limiti come:

- Single point of Failure: In caso di malfunzionamento l'intero sistema diventa irraggiungibile
- Scalabilità limitata: può scalare solo verticalmente
- Complessità
- Latenza aggiuntiva

Load Balancing

È il processo effettuato da un proxy per distribuire il traffico di rete verso molteplici server, garantendo che nessuno sia sovraccarico, ottimizzando le risorse, garantendo il minor tempo di risposta e il massimo throughput.

Il load balancing è applicabile tramite software (algoritmo aggiunto al proxy tramite app) oppure hardware (componenti fisici installati in un data center).

Per farlo sono stati creati alcuni algoritmi che differiscono in base alla scelta a chi indirizzare le richieste.

Round Robin

Algoritmo semplice, basilare e senza garanzie che manda le richieste in maniera sequenziale senza controlli sul carico di lavoro o connessioni attive.

Least Connection

La richiesta viene inoltrata al server con meno richieste all'attivo, quindi si prova a bilanciare le richieste su tutti i server.

Ottimo in scenari in cui le connessioni hanno tutte circa lo stesso numero di richieste, ma meno adatto a scenari più dinamici.

Altri

- Least Response Time: Le richieste sono inoltrate al server con minor tempo di risposta
- Weighted Round Robin: Analogo al RR, ma ad ogni server viene assegnato un peso, ovvero una % di richieste che deve ricevere
- Random: Inoltra le richieste in modo completamente randomico
- IP Hash: Inoltra le richieste provenienti dallo stesso client sempre allo stesso server, in modo da poter gestire localmente la sessione
- Adaptive: Valutano molteplici aspetti del server (carico, connessioni, response time...)

Cloud Load Balancer

Soluzione offerta dai provider dove l'utente non deve gestire nulla, basta configurarlo in termini di protocolli da usare, algoritmo, porte, e su quali server bilanciare.

Google Load Balancing

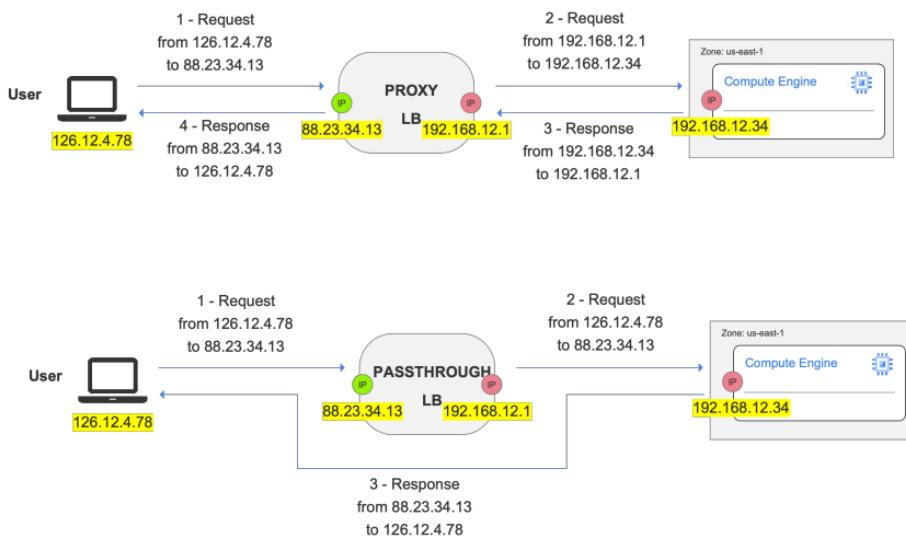
Sono servizi distribuiti globalmente ed interamente gestiti in software da Google. Non si richiedono configurazioni hardware né la creazione di VM dedicate; con autoscaling e l'health check utilizzando managed instance group come backend.

GCP ne prevede 6 tipologie combinate in quattro dimensioni:

- Global vs Regional: i globali usano EDGE Network, erogati dai EDGE POP, usati se gli utenti sono sparsi.

I regionali distribuiscono il traffico verso istanze che si trovano tutte nella stessa regione.

- External vs Internal: i primi smistano il traffico proveniente dalla Internet pubblica e diretto alle istanze interne a GCP, i secondi smistano solamente traffico interno alla rete di Google.
- Proxy vs Passthrough: I load balancer proxy interrompono le connessioni in entrata e aprono nuove connessioni dal bilanciatore ai backend. I backend risponderanno quindi al LB che inoltrerà le risposte al client. I load balancer passthrough non interrompono le connessioni e si limitano ad inoltrare i pacchetti a destinazione.



I load balancer di GCP si differenziano per il tipo di traffico che possono gestire e inoltrare:

External / Internal	Global / Regional	Proxy / Passthrough	Traffic	Load Balancer
Internal	Regional	Passthrough	TCP, UDP	Internal TCP/UDP Load Balancer
		Proxy	HTTP, HTTPS	Internal HTTP(S) Load Balancer
External	Global	Proxy	HTTP, HTTPS	External HTTP(S) Load Balancer
		Proxy	SSL	SSL Proxy Load Balancer
External	Regional	Proxy	TCP	TCP Proxy Load Balancer
		Passthrough	TCP, UDP, ICMP	Network Load Balancer
		Proxy	HTTP, HTTPS	Regional External HTTP(S) Load Balancer

CDN

O Content Delivery Network, sistema di caching e indirizzamento di client verso

il nodo più vicino per evitare sovraccarichi di rete e attese troppo lunghe.

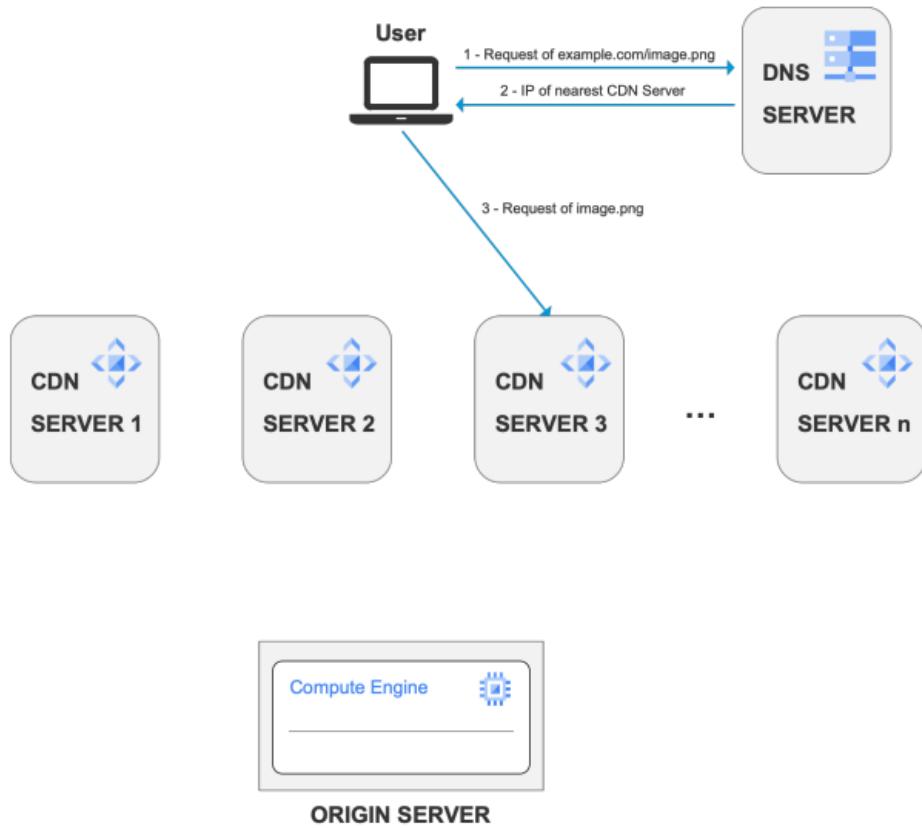
L'idea è nata perché un utente che accede a dei dati probabilmente accederà ad altri dati logicamente simili e può riaccedere allo stesso dati più volte in un breve lasso di tempo.

Ogni CDN è composto da:

- Origin Server: dove si trovano i dati da consegnare.
- CDN Server: dove si mantengono i dati forniti dall'origin server e distribuiti in posizioni tali da limitare la latenza. Questi dati non vengono copiati in modo sistematico ma è fatto quando il client lo richiede, possono avvenire due casistiche:
 1. Cache Miss: il contenuto non è nel CDN e si richiede.
 2. Cache Hit: il contenuto è già nel CDN e si invia.

I contenuti in cache possono essere cancellati dopo un tot di tempo (tramite TTL) e si dice Cache TTL o si svuota la cache a necessità (Cache Invalidation).

- DNS Redirection: per risolvere il nome di dominio con l'indirizzo IP del CDN Server più vicino al client.

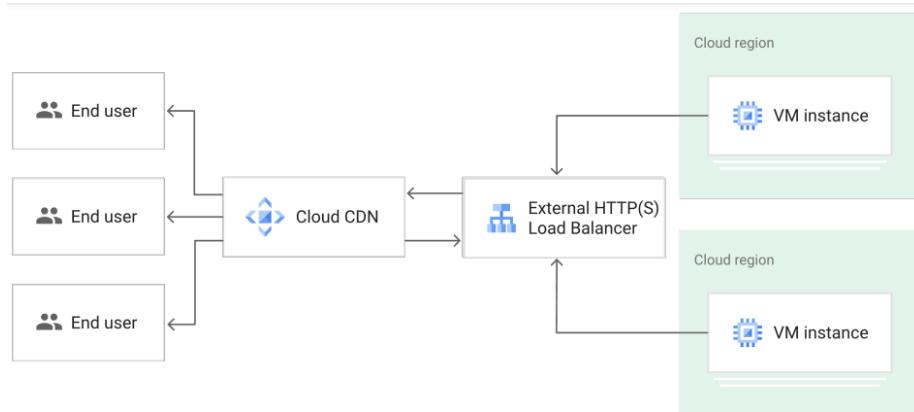


I vantaggi portati dalle CDN sono:

- Velocità
- Scalabilità
- Protezione DDoS
- Monitoraggio traffico
- Costi ridotti

Google Cloud CDN

Sistema usato globalmente con EDGE POP per avere dati sempre vicino agli utenti, abbinato ad un load balancer per avere un unico IP.



Cloud CDN prevede tre modalità di gestione della cache:

- USE_ORIGIN_HEADERS: Vengono usati appositi header, presenti nelle richieste dei dati, per determinare cosa inserire in cache e cosa no.
- CACHE_ALL_STATIC: In automatico inserisce in cache tutti i dati statici, ovvero tutti i dati in cui è assente la direttiva no-store o no-cache.
- FORCE_CACHE_ALL: Vengono inserite in cache tutte le risposte, indipendentemente dagli header presenti

Infrastructure as Code (IaC)

Invece di creare infrastrutture cloud manualmente si usa il codice, IaC permette:

- creazione, gestione, rimozione di infrastrutture
- integrazione con Continuos Integration
- centralizzare la definizione dell'infrastruttura con anche il versioning

Gli strumenti IaC sono Configuration o Provisioning.

Configuration tools

Gestiscono la configurazione di infrastrutture già esistenti, definendone uno stato di arrivo e gestendo le operazioni necessarie ad ottenerlo.

Implementati tramite linguaggi imperativi (definiscono le istruzioni da eseguire per ottenere uno stato finale) e per questo sono non idempotenti (due esecuzioni consecutive dello stesso codice possono portare a risultati diversi ed inconsistenti) e non tengono conto dello stato dell'infrastruttura tra un'esecuzione e l'altra.

Provisioning tools

Definiscono le operazioni necessarie alla creazione di nuove risorse infrastrutturali in maniera veloce e consistente

Implementati tramite linguaggi dichiarativi (descrivono il risultato finale desiderato senza specificare i passi necessari per raggiungerlo) quindi il codice è una rappresentazione diretta dello stato attuale dell'infrastruttura, sono idempotenti (molteplici esecuzioni dello stesso codice portano sempre allo stesso risultato).

Fra i vantaggi abbiamo:

- Version control
- Riuso
- Modularità: possibilità di divisione del codice in diversi file con anche microservizi
- Minori Errori

Terraform

Tool IoC, idempotente, open source, platform-agnostic (utilizzato su molteplici cloud provider), per il provisioning dichiarativo di infrastrutture cloud.

Sia Agentless che Masterless, quindi nessuna installazione nelle risorse cloud.

Si usa HCL (QUI da slide 28 a 50) come linguaggio (dichiarativo) per i file di configurazione che descrivono a Terraform le risorse necessarie per realizzare una singola applicazione o l'intera architettura; che poi genera un piano di esecuzione che descrive le azioni da eseguire per raggiungere lo stato desiderato partendo dallo stato attuale. Deploy nel dettaglio:



Le versioni sono:

Versione	Pro	Contro
Terraform Open Source	<ul style="list-style-type: none"> • Nessun costo di licenza • Da installare in locale 	<ul style="list-style-type: none"> • Non supporta il deploy in parallelo • Interfaccia solo da riga di comando • Non supporta il versioning (deve essere gestito a mano)
Terraform Cloud	<ul style="list-style-type: none"> • Versione SaaS • Supporta tutte le funzionalità • Tre piani di abbonamento • Disponibilità di interfaccia grafica e di riga di comando 	<ul style="list-style-type: none"> • Costi di licenza
Terraform Enterprise	<ul style="list-style-type: none"> • Installazione onPremises • Supporta tutte le funzionalità della versione Cloud 	

Terraform mantiene uno "stato" dell'infrastruttura e delle configurazioni previste, permette a terraform di mantenere le corrispondenze tra le risorse create nell'infrastruttura cloud e la loro definizione nei file tf.

La consistenza dello stato può diventare un problema quando più utenti lavorano contemporaneamente alla definizione dell'infrastruttura.

Architetture moderne

Normalmente l'approccio che si usa per creare applicazioni è di tipo monolitico, dove tutte le funzionalità sono nella stessa codebase. Questo rende l'applicazione architetturalmente semplice migliorando la comunicazione fra i moduli e la scalabilità verticale.

Questo approccio porta svantaggi durante l'aggiornamento o la modifica dell'applicazione con una complessità di gestione non indifferente.

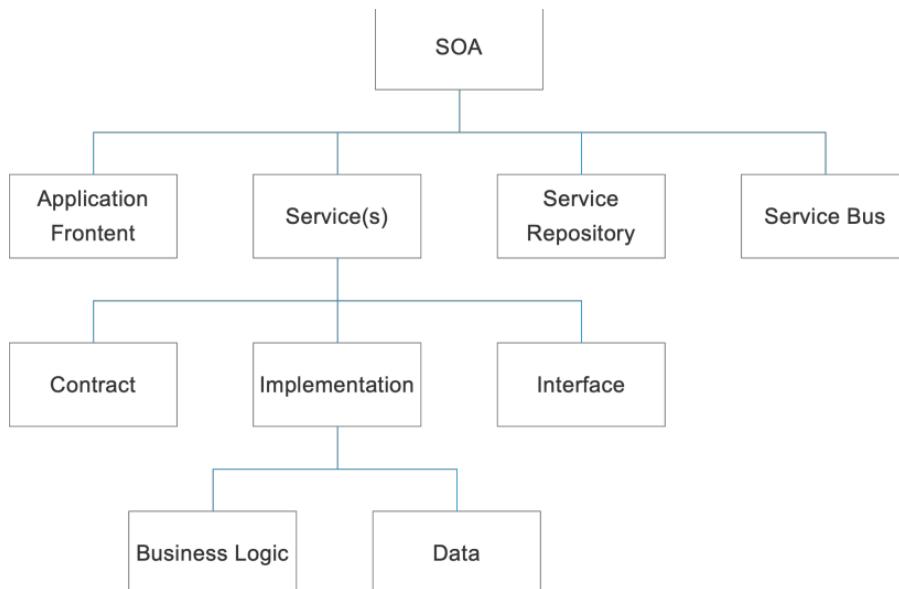
In sostanza è un approccio per progetti piccoli o legacy aumentando la semplicità di sviluppo e di deploy.

SOA

Service Oriented Architecture, al contrario dell'approccio monolitico lo scopo qui è dividere il più possibile in servizi sempre più piccoli con specifici compiti che comunicando insieme formano un'applicazione e andando a scalare senza problemi in entrambi i sensi.

I servizi sono fatti per essere il più autonomi possibili (sviluppati/aggiornati in maniera separata), garantendo modularità (unione di più servizi come si vuole per creare funzionalità), astrazione e riusabilità.

Architettura



I vari componenti della SOA sono:

- Service: fulcro dell'architettura, coloro che implementano le funzionalità, composti da:
 - Contract: definisce i dettagli tecnici e funzionali del servizio
 - Implementation: codice che implementa il contract
 - Interface: punto d'accesso al servizio
- Service Repository: archivio centralizzato con le informazioni di tutti i servizi disponibili nell'architettura
- Application Frontend: applicazione finale che utilizza i servizi
- Service Bus: middleware per la comunicazione tra servizi, funzionalità:
 - Mediazione: comunicazione senza la consapevolezza di quanti servizi ci sono.
 - Routing dei messaggi: instradamento di messaggi dai servizi mittenti ai servizi destinatari in base alle regole di routing.
 - Trasformazione dei messaggi: trasformazione dei messaggi da un formato a un altro.
 - Gestione delle transazioni: coordinazione delle operazioni di più servizi.
 - Gestione degli errori e del monitoraggio: fornisce funzionalità per il monitoraggio dei servizi e per la gestione degli errori (gestione delle cose, ripristino operazioni fallite, registrazione eventi, ecc).

Per comunicare, due servizi, devono seguire 4 fasi:

1. Il mittente (Service Consumer) contatta il Service Bus utilizzando il proprio formato di richiesta
2. Il Service Bus controlla il Service Registry per determinare il formato di richiesta accettato dal destinatario. Se necessario effettua la traduzione
3. Il Service Bus contatta il destinatario (Service Producer) usando il formato da esso richiesto
4. Il Service producer risponde utilizzando il proprio formato, che sarà tradotto dal service bus prima di inviarlo al service consumer

SOAP

O Simple Object Access Protocol, è un protocollo che indica il formato standard per la comunicazione tra servizi web. Usato per evitare al Service Bus di effettuare traduzioni.

Il messaggio è formato dalle Envelope che contiene l'header (info del messaggio come mittente e destinatario) e il body (messaggio vero e proprio) della richiesta.

WSDL

O Web Services Definition Language, un linguaggio di descrizione di servizi

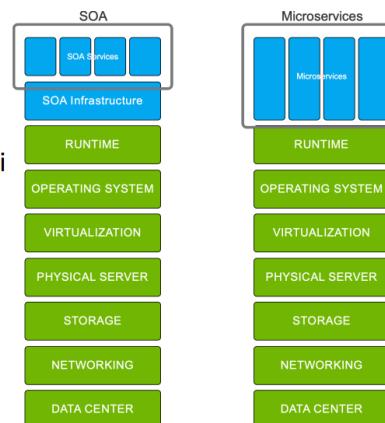
utilizzato per definire l'interfaccia in modo standardizzato e indipendente dal linguaggio di programmazione.

Microservizi

L'evoluzione di SOA per tutte quelle aziende che devono muoversi in maniera rapida per adattarsi ai cambiamenti del mercato e non possono permettersi di seguire un modello ancora troppo centralizzato ed articolato.

Per questo è nato un pattern architettonale che struttura un'applicazione come un insieme di microservizi indipendenti, completamente decentralizzati e pensati come una istanza applicativa. Favorevole alla virtualizzazione, al cloud e ai IaC e che tramite un approccio metodologico chiamato Agile Movement consente di sviluppare più rapidamente un software e le sue funzionalità.

- SOA può essere visto come un livello applicativo unico (data la necessità dei componenti centralizzati) su cui costruire i servizi
- Nell'approccio a microservizi invece ogni servizio è una applicazione a sé stante



Inoltre il DB, al contrario del SOA, è diviso per ogni micro servizio che ne ha uno proprio.

Ovviamente anche i microservizi hanno svantaggi, soprattutto se si viene dal mondo SOA, infatti si ha un cambiamento significativo nel progettare app. Per non parlare della complessità dell'architettura sottostante che porta i temi di load balancing, sicurezza e monitoraggio a tener conto di molteplici elementi.

REST

Soluzione alternativa a SOAP per i microservizi, basato sul protocollo HTTP quindi:

- Orientato alle risorse: i microservizi espongono risorse accessibili tramite REST (identificate da URI).
- Client server e request-response: il server aspetta le richieste (i vari verbi HTTP come GET , POST, ecc) e successivamente risponde solo se arrivano.

- Stateless: il server non ha stati applicativi.

I microservizi espongono la propria interfaccia API per richiedere le risorse e per questo i servizi sono Restful API. Le API sono nella forma classica; Esempio:

- GET /students → Recupera tutti gli studenti
- POST /students → Aggiunge un nuovo studente
- GET /students/{id} → Recupera uno studente dato il suo id
- PUT /students/{id} → Modifica uno studente identificato dal suo id
- DELETE /students/{id} → Cancella uno studente identificato dal suo id
- GET /students/{id}/esami → Recupera i voti di uno studente
- GET /students/{id}/media-voti → Recupera la media voti di uno studente

Questo modo rende la piattaforma e il linguaggio indipendente con una semplicità e flessibilità unica. Ed essendo stateless la scalabilità orizzontale è possibile sui microservizi.

Il formato dei messaggi è il JSON (QUI da slide 46 a 50).

Event Driven Architecture

O EDA, è un pattern che si basa sulla comunicazione asincrona basata su eventi tra sistemi tramite rapporto publisher/subscriber. Usata in applicazioni IoT o messaggistica.

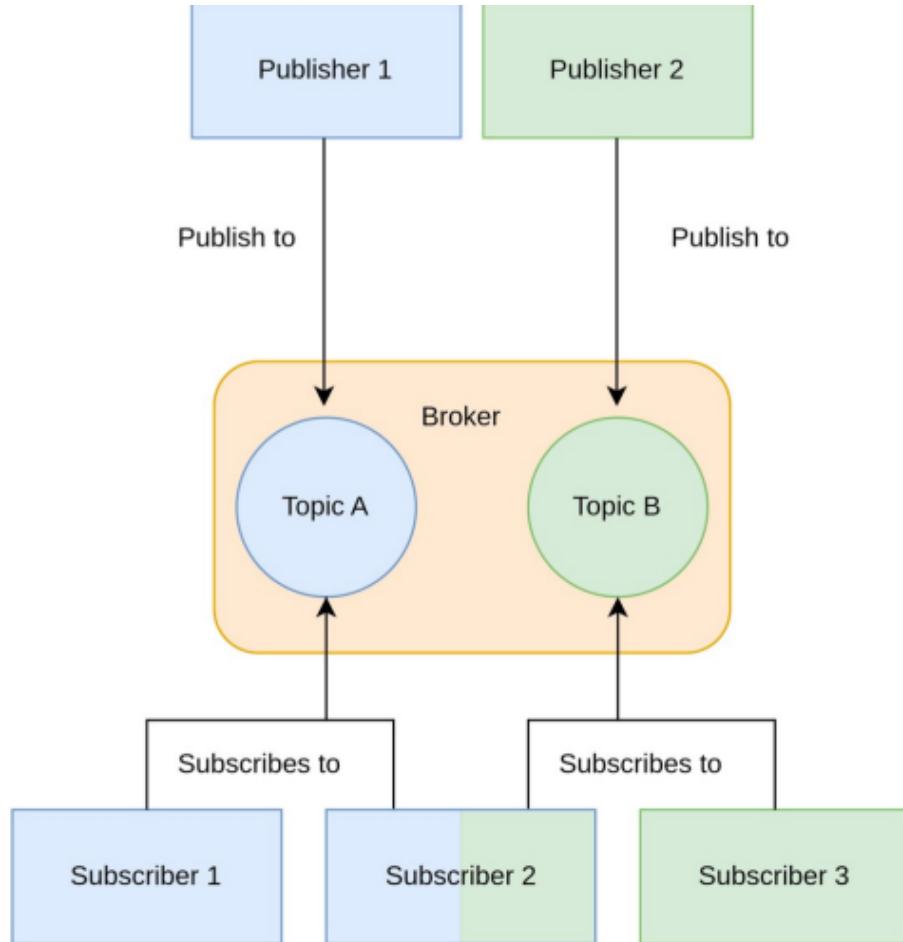
In sostanza il publisher è un sistema che pubblica un evento in determinate condizioni e il sistema subscriber si registra a quell'evento ricevendo una notifica/messaggio quando le condizioni sono verificate.

Per funzionare c'è bisogno di un Event Broker che gestisce le pubblicazioni, le sottoscrizioni e le notifiche.

i vantaggi sono:

- Indipendenza tra servizi.
- Scalabilità.
- Real-time grazie all'alto grado di parallelismo si possono gestire grandi volumi di messaggi e flussi dati.

Pub / Sub model



Nel dettaglio:

- **Publisher:** le entità che producono e inviano il messaggio (evento) verso un message broker
- **Subscriber:** Entità che ricevono ed elaborano il messaggio da un message broker a cui si sono registrati
- **Message broker:** strato intermedio che riceve i messaggi dai publisher e li inoltra agli appropriati subscriber. Si compone di una serie di **topic**, ciascuno adibito a contenere messaggi di una determinata categoria

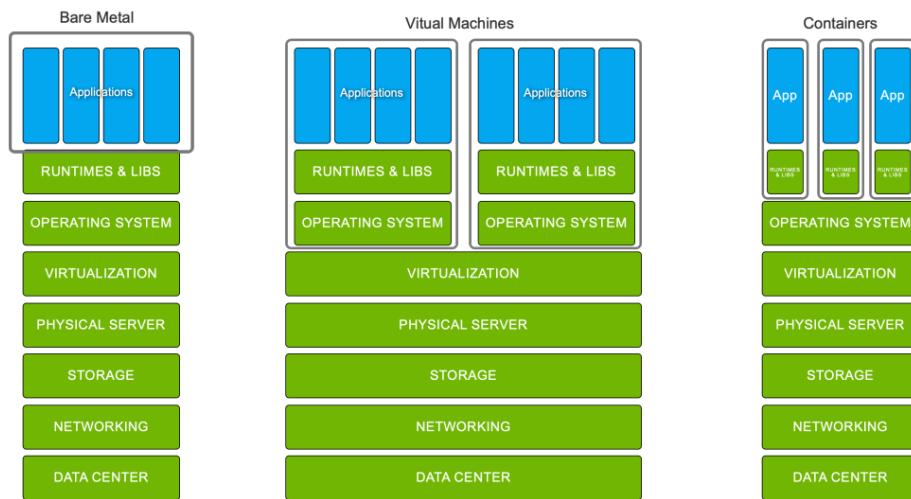
I sub si possono iscrivere a uno o più topic e quando il pub manda l'evento al

topic il broker lo riceve e lo archivia. Successivamente tramite broker si può fare Push (inoltro ai sub) o Pull (i sub verificano da soli i messaggi).

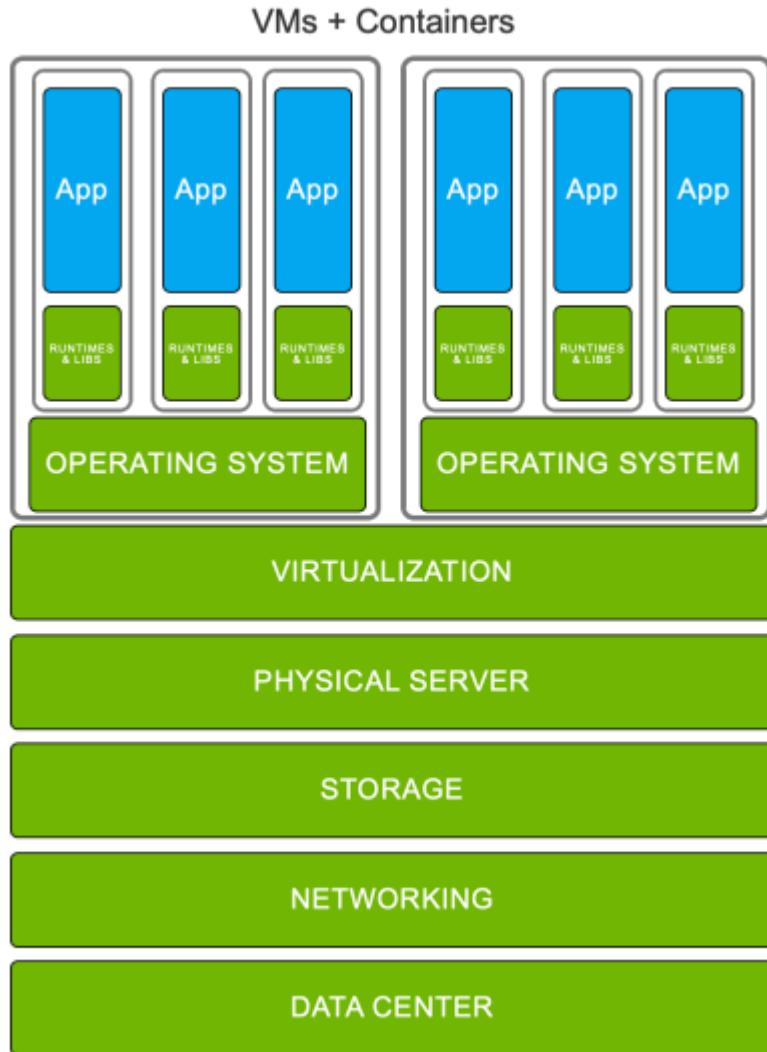
Containers

Per risolvere le criticità che i microservizi portano alle applicazioni, come l'impossibilità di isolare i microservizi iper aggiornamenti o non poterli trasferire da un SO ad un altro, sono nati i container.

Tecnologia di virtualizzazione senza hypervisor (a livello SO) dove ogni container è un pacchetto di applicazioni con relative dipendenze che condividono fra loro solo il kernel linux del SO host, completamente indipendenti in termini di file system, processi, memoria e networking.



Rispetto alle VM i container hanno migliori performance, meno dipendenze con il SO e virtualizzazione innestata cioè VM con dentro containers:



L'isolamento è reso possibile dal LinuX Container (LXC) composto da due tecnologie, Cgroups e Namespaces, che ora vediamo.

Control Groups

O cgroups, utilizzato dal kernel linux per la gestione dell'utilizzo delle risorse da un insieme di processi specifici, organizzati in maniera gerarchica con struttura ad albero con ogni nodo che è un gruppo con certe regole.

Namespaces

Il kernel Linux lo usa per creare risorse virtuali, come astrazione di risorse fisiche,

che sostanzialmente dice ad un determinato processo di usare solamente le risorse assegnate al suo stesso namespace.

S.O.

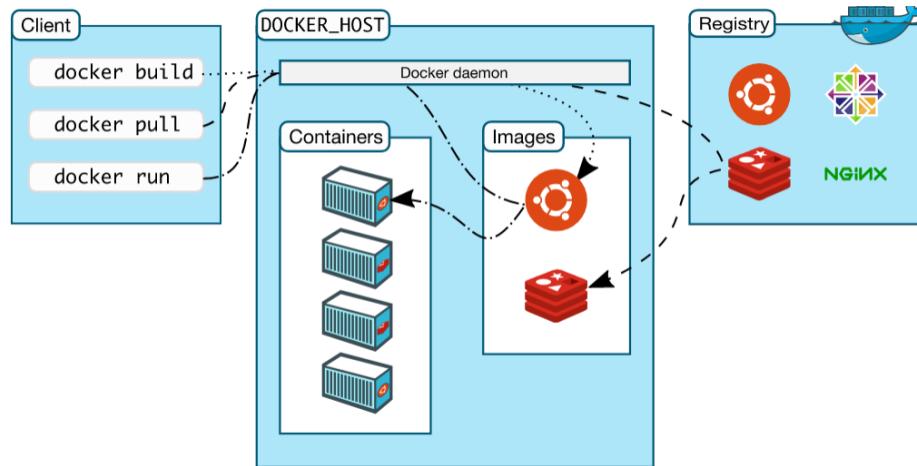
I container non possono eseguire SO differenti a Linux, ma possono decidere quale distribuzione usare, anche se differente a quella dell'host.

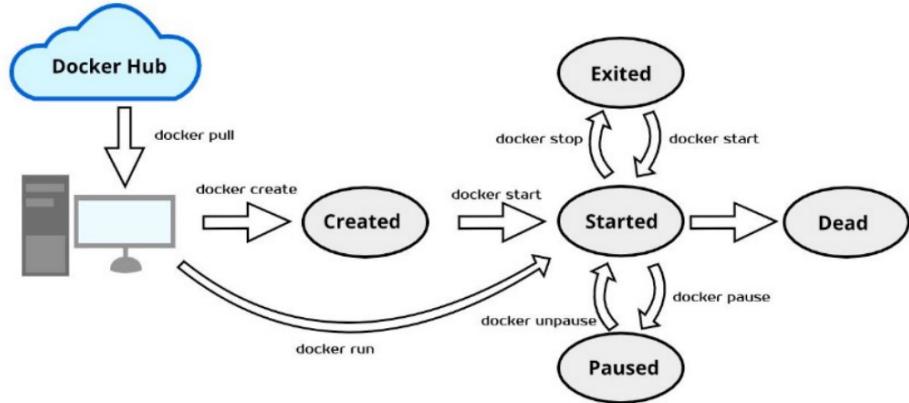
Come già si sarà intuito l'unico modo per usare i container su Windows è farlo con una VM Linux, quindi impossibile farlo in maniera nativa.

Docker

Sistema di virtualizzazione open source e client server basato su container, composto da:

- Docker Daemon: il processo responsabile della gestione di immagini e container
- Docker Client: l'applicazione con cui fornire comandi al daemon
- Docker Registry: Il repository delle immagini





ciclo di vita

Docker si basa sul concetto di immagini, template in sola lettura dai quali derivare i container che verranno messi in esecuzione.

Il rapporto tra immagine e container può essere paragonato al rapporto tra classe e oggetto esistente nei linguaggi di programmazione (container è una istanza di una immagine)

Immagini

Versione immutabile, in sola lettura e strutturata in layer ordinati di un software, salvati in questo modo:

host/registry/image:tag

Dove:

- Host: Il nome del server in cui è memorizzata l'immagine (p.e. docker.io)
- Registry: Il nome del repository in cui è contenuta l'immagine
- Image: Il nome dell'immagine
- Tag: L'identificativo della versione dell'immagine, scelto in modo arbitrario, quello di default è latest.

Ogni layer definisce un aspetto di quello che sarà il filesystem finale dell'immagine e in fase di creazione sono "eseguiti" in sequenza, dando origine al file system definitivo.

Anch'essi sono immutabili permettendo la condivisione tra immagini.

Docker File

File che definisce come i layer compongono un'immagine, dove ad ogni riga equivale ad un layer:

```
FROM node:18-alpine3.18
RUN mkdir -p /opt/app
WORKDIR /opt/app
COPY ./ .
RUN npm install
CMD [ "npm", "start"]
```

Container

La creazione di un container aggiunge un ulteriore layer a quelli presenti nell'immagine di partenza, collocato sopra a tutti gli altri (eseguito per ultimo) e non condivisibile.

Questo layer permette ai container di effettuare modifiche al file system in modo indipendente dagli altri container nati dalla stessa immagine.

Docker Hub

Repo gestito direttamente da docker per la condivisione di immagini

Artifact Registry

Repo gestito direttamente da Google per la condivisione di immagini (non solo docker)

Google Cloud Run

Servizio PaaS per l'esecuzione di docker container completamente gestito da Google, fatturato solo il tempo di utilizzo, calcolato al decimo di secondo.

Cloud Run genera un url pubblico (in https) con cui accedere direttamente al container, deploy configurato in termini di:

- Variabili d'ambiente
- Modalità di accesso (autenticata / non autenticata)
- Networking (gestione del traffico in entrata e in uscita)

Cloud run inoltre gestisce l'aggiornamento del container, in modo che questo possa avvenire senza interrompere il servizio.

Kubernetes

Container Orchestration

Su Docker la gestione può diventare estremamente complessa in scenari altamente strutturati, per questo sarebbe utile disporre di Orchestratori di Container, ovvero applicazioni in grado di gestire tutti gli aspetti legati all'utilizzo combinato di più container.

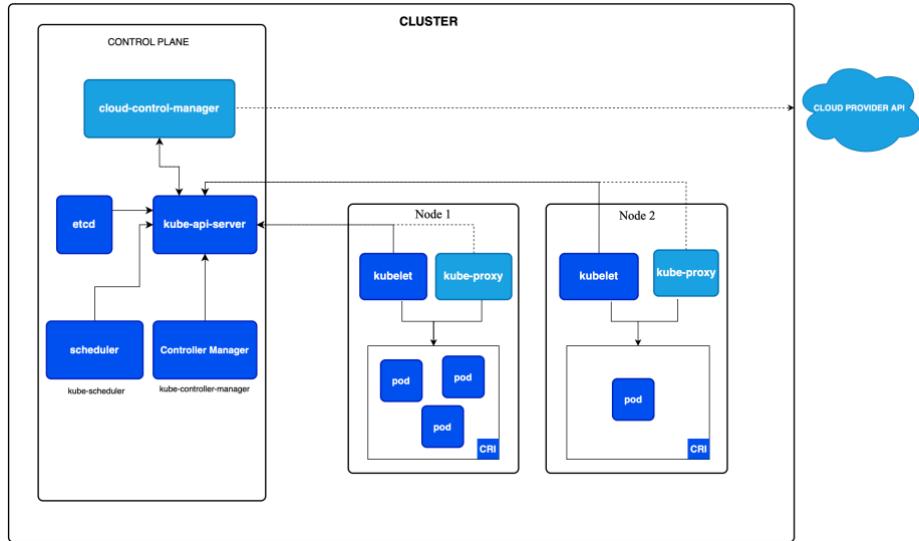
I sistemi di container orchestration hanno caratteristiche comuni:

- L'architettura è centralizzata, composta da un master e da un set di worker
- Il **master** è l'orchestratore, responsabile della gestione dei cluster di VM, dello scaling e dello scheduling dei container sui worker. Ha un suo pannello di controllo e una sua interfaccia amministrativa.
- I **worker** sono responsabili dell'esecuzione materiale dei container e dell'esecuzione dei compiti assegnati dal master. Contengono l'ambiente containerizzato (p.e. Docker) e gli strumenti necessari per la gestione.
- Master e Workers comunicano in rete e possono anche essere fisicamente lontani

Kubernetes

Nato per permettere ai container di sfruttare le potenzialità dell'infrastruttura cloud, superando i limiti di Docker nel coordinare progetti complessi con la possibilità di automatizzare il deploy, la scalabilità e le operazioni di gestione di applicazioni containerizzate su gruppi di host (istanze VM).

Architettura Master/Worker, con cluster composti da nodi dove uno specifico è il master con funzionalità diverse dai worker.



i componenti worker sono:

- Pod: è la più piccola unità che può essere creata e gestita in Kubernetes. I pod contengono i container da eseguire (approfondiremo tra poco)
- Kubelet: Servizio che gestisce i pod e le risorse da essi utilizzate (container, immagini, volumi...)
- Kube-proxy: Servizio di proxy che opera anche da load balancer per inoltrare il traffico verso il pod appropriato.

I componenti master sono:

- Scheduler: Determina su quale nodo deve essere eseguito un Pod
- Api Server: È l'interfaccia REST per interagire con il master
- Controller Manager: Responsabile dei controlli di replicazione e delle relative azioni
- Etcd: Database key/value usato dal Master per memorizzare tutte le informazioni persistenti di cui ha bisogno

Pods

Unità più piccola gestita, insieme di container raggruppati logicamente per poter lavorare insieme. I container dei pod possono essere visti come processi di una VM (dove la VM è il pod) che comunicano fra loro condividendo IP e porta e volumi condivisi (come dischi di una VM).

Hanno un ciclo di vita corto, si creano, si assegnano un ID e vengono schedulati sui nodi.

Deployment & Replicasets

Strumenti per la replicazione di pod, il primo lavora ad un livello più alto del secondo.

Il deployment è un insieme di Pod identici definiti tramite yaml, il suo scopo è specificare il numero di repliche che dovranno essere eseguite, se alcune falliscono le riattiva finché non si torna a quel determinato numero, e applicare ogni modifica fatta al template yaml ricreando i pod con strategie di aggiornamento.

Un ReplicaSets funziona come un deployment (garantisce che un certo numero di pods in ogni istante di tempo), solo che non gestisce strategie di aggiornamento.

Services

I servizi sono un insieme di pod indipendenti affiancati da modalità d'accesso, raggruppati tramite selettori, che assicura la possibilità di raggiungere il pod appropriato senza preoccuparsi della posizione.

Per farlo si usa un IP (o nome di dominio) che i consumatori useranno, come questo servizi vengono esposti si creano alcuni tipi:

- ClusterIP: Espone il servizio sulla rete interna del cluster, rendendolo inaccessibile dall'esterno. È il default.
- NodePort: Espone il servizio mappando un cluster ip su una porta di ogni nodo, rendendo il servizio raggiungibile tramite l'ip di ogni nodo.
- LoadBalancer: Espone il servizio su un indirizzo ip raggiungibile dall'esterno. Fa uso di NodePort (e quindi di ClusterIP)

GKE - Google Kubernetes Engine

Servizio Google per creare/utilizzare cluster kubernetes, con gestione di master e worker e funzionalità native per sfruttare al meglio kubernetes tra cui load balancing, scaling automatico dei nodi, monitoraggio delle risorse, patch di sicurezza ecc.

Devops

SDLC

Software Development Life Cycle, letteralmente il ciclo di vita del software che parte dall'analisi e finisce con la pubblicazione e il futuro mantenimento.



Analisi

Raccolta dei requisiti funzionali e non funzionali tramite interviste, sondaggi o osservazioni fatte alle persone rilevanti, i cosiddetti stakeholder.

Alla fine della raccolta si scrive tutto su un documento dove si ordinano i requisiti in ordine di priorità.

Progettazione

Si progettano, in maniera coerente con il documento di analisi, le varie parti architettoniche dell'applicazione, ogni parte ha una tecnica specifica:

- Struttura del codice à pattern architettonici, diagrammi UML (class, sequence, activity diagrams)
- Struttura dei database à progettazione E/R
- User Interface à User experience design
- Caratteristiche dei server o delle piattaforme cloud à cloud engineering

Sviluppo

Si inizia a scrivere il codice andando dietro a quello fatti nella fase di progettazione, utilizzando versioneing e scrivendo i primi test.

Test

Si verifica che il codice sia conforme con tutti i requisiti (funzionali e non) e a tutte le scelte progettuali, i test hanno diversi livelli:

- Unit e Integration testing: Per verificare l'assenza di bug nel codice
- System testing: Per verificare la presenza di tutte le feature richieste e la corrispondenza con le specifiche
- Performance testing
- Usability testing
- Security testing

Deploy

Fase cruciale dove si rilascia il software, per farlo si usano diverse tattiche. Bisogna garantire continuità e rollback.

Complete Deployment

O Big Bang Deployment, consiste nel fare deployment e rilascio di tutte le componenti in una volta.

Alto rischio, usata per architetture monolitiche, con poche opzioni di ripristino se qualcosa va storto.



Rolling Deployment

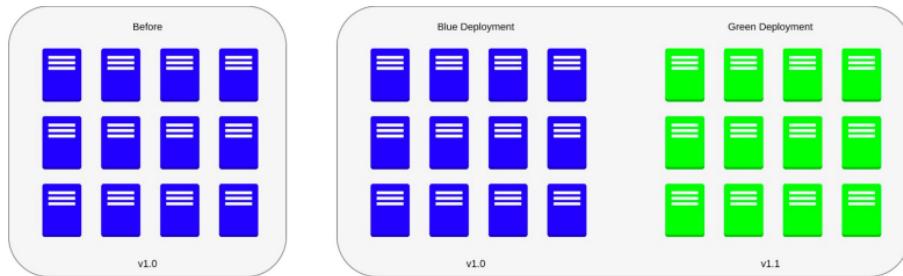
Fatto una componente alla volta, procedendo alla successiva se la precedente funziona correttamente, minimizzando di parecchio il rischio di fallimento completo.



Blue/Green Deployment

Vengono mantenute due copie identiche dell'infrastruttura, denominandole blu e verde. L'ambiente in uso dagli utenti è quello blu, mentre gli aggiornamenti sono deployati sul verde. Se quella verde ha problemi si fa il rollback sulla blu.

Costosa ma efficace.



Canary Deployment

Viene fatto il deployment e il rilascio della nuova versione solo ad una piccola percentuale di utenti (casualmente scelti), si monitora l'andamento fino al rilascio completo.



Manutenzione

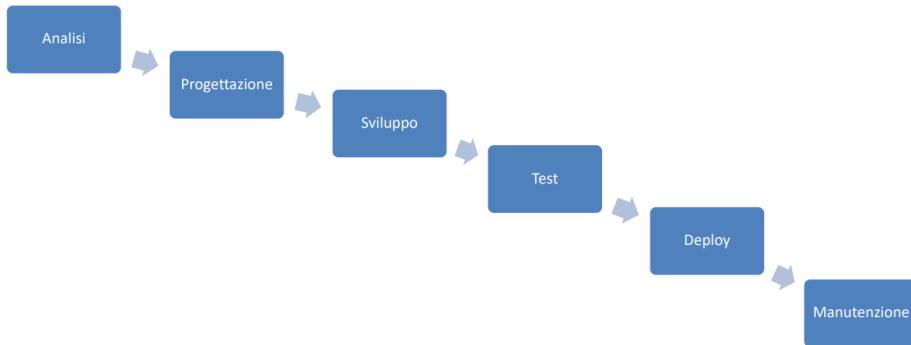
Si monitora il software per problemi e bug tramite sistemi di bug tracking ed eventualmente risolti in base alla priorità.

Metodologie

Waterfall

Fasi eseguite in stretto ordine sequenziale. Si può procedere ad una fase solo se la successiva è completamente terminata.

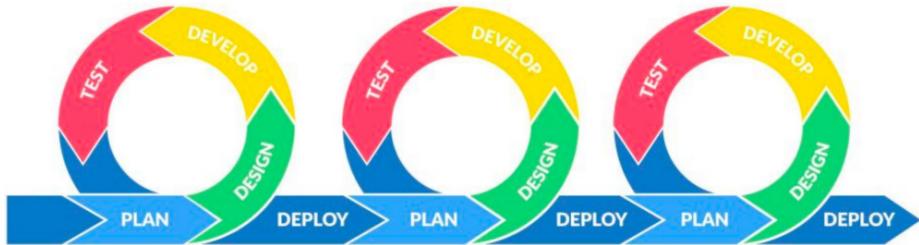
Adatto per grandi progetti, ben strutturati, con requisiti chiari e ben definiti.



Agile

Ne esistono tante, ma tutte sono volte alla flessibilità con rilasci intermedi del software, non complete ma funzionanti.

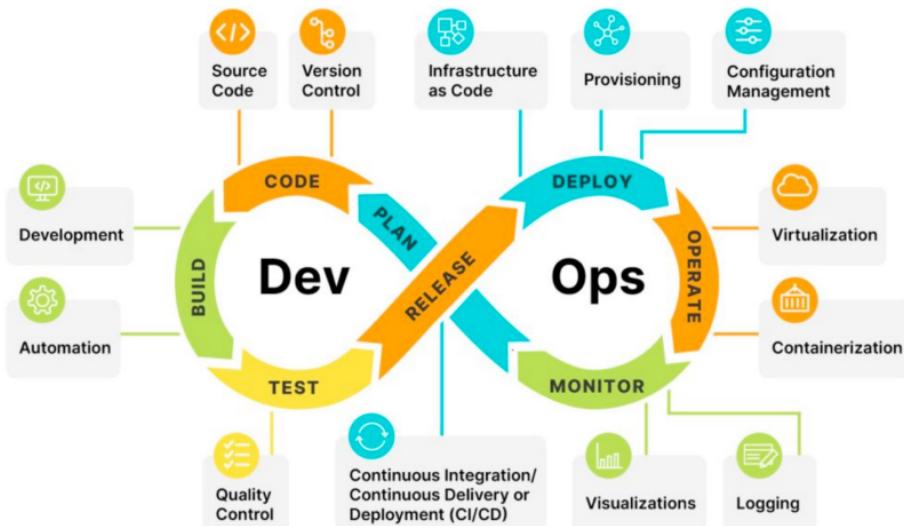
Lo sviluppo procede pertanto con una sequenza di iterazioni dette sprint, intervallati da incontri regolari (stand-up meetings) tra i componenti del team per effettuare la pianificazione a breve termine e per tenersi aggiornati sugli avanzamenti.



Devops

Nata per favorire la comunicazione tra sviluppatori e reparto IT che non accadeva dell'Agile (incentrate sulla comunicazione fra soli sviluppatori).

Crea pipeline per automatizzare task ripetitivi dal software development al rilascio finale. Utilizza i microservizi incentivando la sperimentazione.

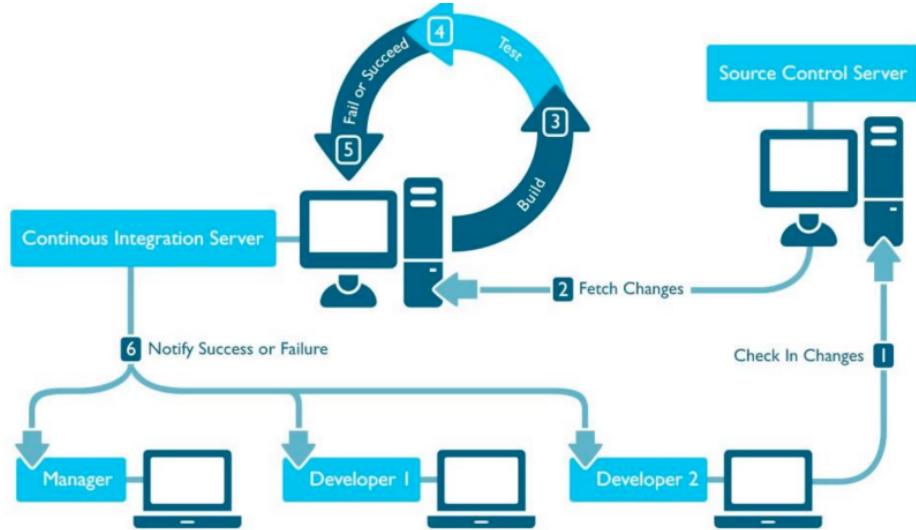


CI/CD

L'insieme di CI e CD riduce i rischi semplificando l'individuazione dei bug e permettendo di ridurre al minimo il time to market

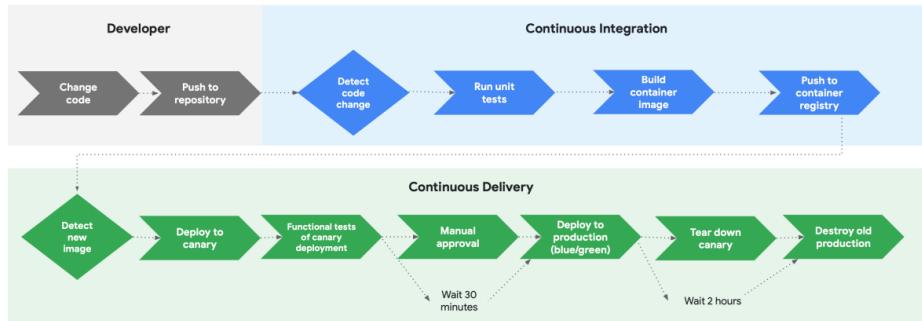
CI

La Continuous Integration (CI) è un processo che prevede di integrare il codice degli sviluppatori in un repository condiviso, più volte al giorno. Ogni check-in di codice è poi verificato tramite una build automatica in modo da rilevare prima possibile eventuali problemi.



CD

Il Continuous Deployment (CD) è strettamente collegato al processo di CI e si occupa di rilasciare in produzione il software che ha superato le fasi del processo di integrazione; tramite strategie di deploy, notifica e eventuali autorizzazioni manuali.



Su Google Cloud Platform

GCP fornisce due strumenti di CI/CD:

- Cloud Build: strumento di CI con alcune possibilità di deployment, elabora una repository applicandogli degli step di build riportati in un file YAML.
- Cloud Deploy: strumenti di CD a partire da build già pronte, automatizza la distribuzione delle app su alcuni ambienti.

Entrambi serverless ed è sufficiente configurarli utilizzando un file YAML.

Monitoraggio e Logging

Logging

Quando si registrano eventi e informazioni da un app/sistema per estrarre informazioni sulle attività per comprendere meglio l'uso o errori del sistema.

I log si possono analizzare in real time o salvati in file di testo o DB, infatti i log hanno una forma strutturata con timestamp e gravità, e un contesto/contenuto chiaro e ben definito.

Alcuni use case dei log potrebbero essere:

- Debug
- Auditing: cioè registrare chi fa cosa.
- Performance monitor: per identificare colli di bottiglia e degrado delle performance.
- Trend Analysis: salvataggio di informazioni per creare serie temporali.

Timestamp

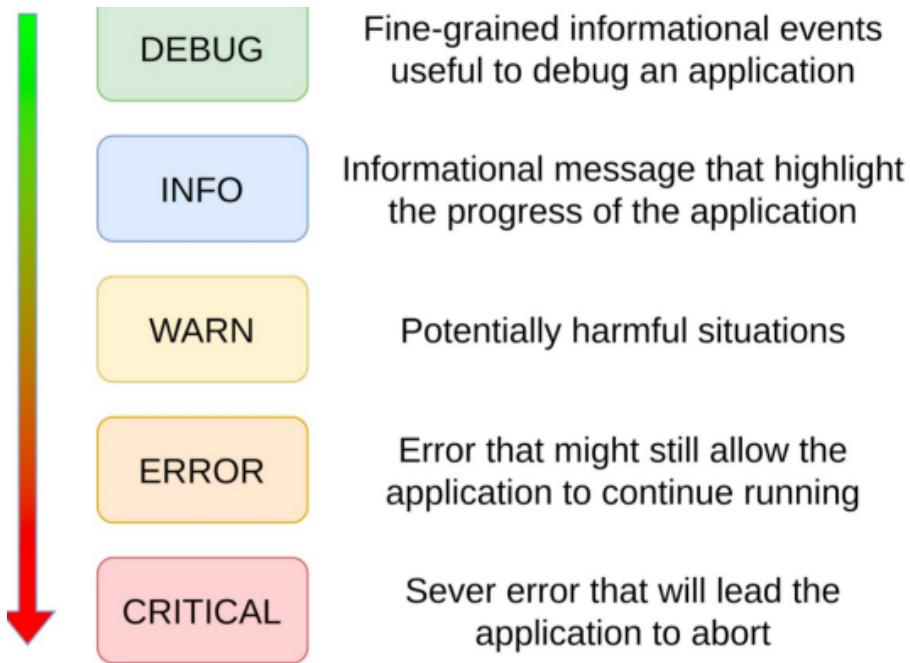
Esatto momento in cui il log si è generato, si usa l'orologio di sistema per avere la data e l'ora precisa e per questo va sempre tenuto sincronizzato.

I due formalismi principali sono:

- ISO 8601 → 2024-04-23T14:16:49.272Z. Dove T separa data e ora e Z indica che il tempo è riferito al fuso orario di Greenwich.
- Unix Time → 1713881809. Tempo espresso in numero di secondi trascorsi dal 1 gennaio 1970.

Severity

La gravità serve per dare priorità ad eventi gravi ed effettuare analisi e filtraggio una scala potrebbe essere:



Strumenti

Alcuni strumenti per gestire, aggregare e analizzare i log possono essere:

- ELK Stack: composto da tre prodotti. Un motore di ricerca (Elasticsearch), un gestore di log (Logstash) e un frontend di visualizzazione (Kibana)
- Splunk: soluzione all-in-one che indicizza e processa i dati ricevuti per visualizzarli tramite avanzate modalità di ricerca (anche uso di ML per individuare anomalie)

Entrambi i prodotti possono essere installati on-premises o su cloud provider. Sono inoltre disponibili come SaaS.

GCP Cloud Logging

Servizio completamente gestito di gestione dei log, compatibile "out of the box" con tutti i servizi GCP; può raccogliere, memorizzare, filtrare e visualizzare i log raccolti dalla Cloud Logging API e consegnati ad una destinazione (sink)

I sink possono essere di tre tipi:

- Required log sink: Per memorizzare attività amministrative ed eventi di sistema. Gli eventi sono memorizzati per 400 giorni e non c'è possibilità di modifica.

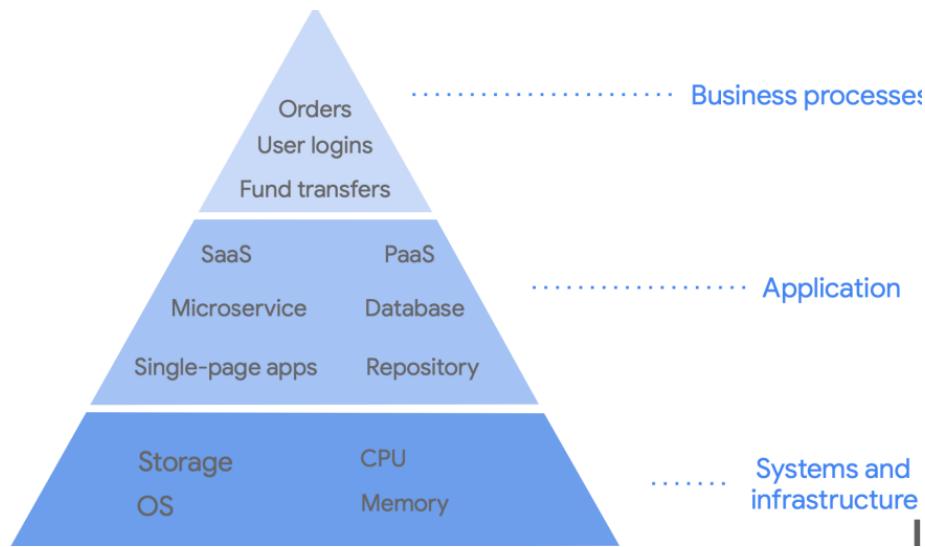
- Default log sink: Per tutti gli eventi che non sono inviati al required sink.
La retention di default è di 30 giorni ma può essere modificata.

I log possono essere esportati dall'archivio centrale verso altre tipologie di storage, più adatte alle specifiche esigenze:

- Cloud Storage: per retention a lungo termine, risparmiando sui costi
- BigQuery: per analisi approfondite, per correlazioni e per identificazione dei trend
- Pub/Sub: per inviarli a strumenti di terze parti, registrati come subscriber di determinati topic

Monitoring

Processo di raccolta ed osservazione di dati operativi e di performance dei sistemi, delle reti e delle applicazioni; con l'obiettivo di mantenere determinati livelli prestazionali e individuando tempestivamente le situazioni problematiche.



Metriche

Quelle che il monitoraggio raccoglie, ovvero misure specifiche collezionate, aggregate ed analizzate nel tempo per fornire informazioni sul livello attuale di performance.

Sono:

- Legate all'utilizzo di risorse: uso CPU, occupazione memoria, spazio disco disponibile, banda di trasmissione occupata...

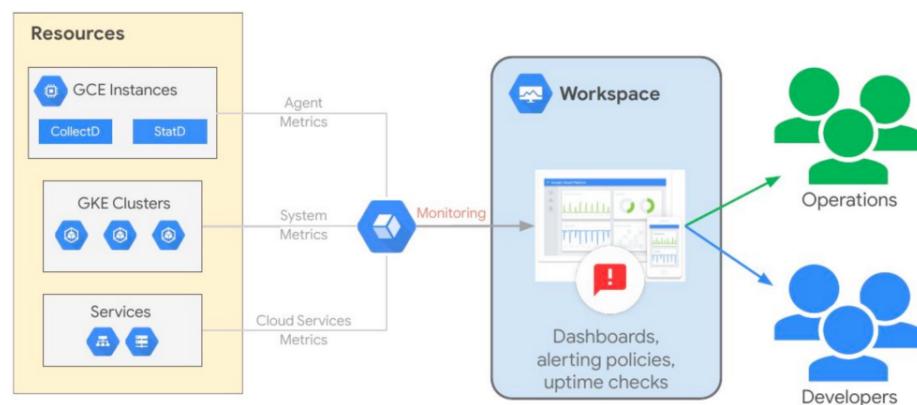
- Legate alle applicazioni: tempi di risposta, latenze, tassi di errore
- Legate al business: Numero di ordini, di utenti attivi, di interazioni...

Processo

Cioè selezionare le sorgenti per cui si vogliono monitorare determinate metriche, che invieranno i dati allo strumento di monitoraggio per memorizzarli e visualizzarli,

GCP Cloud Monitoring

Strumento completamente gestito per raccogliere metriche relative alle risorse cloud.

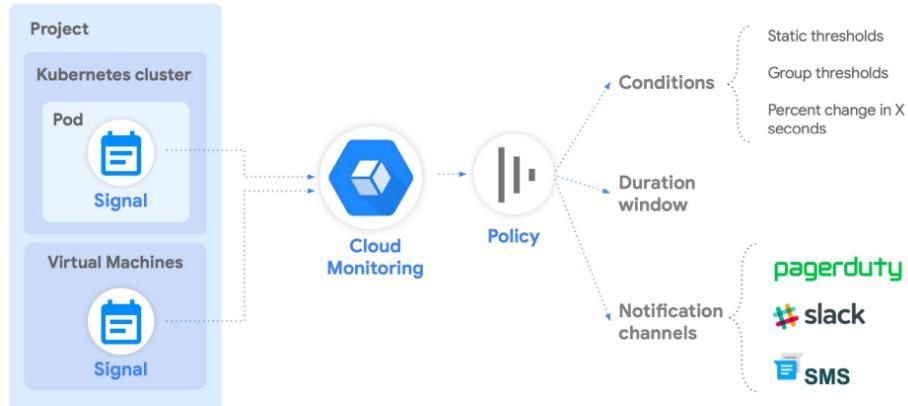


I dati, raccolti ad intervalli regolari e forniti in forma aggregata, possono portare alla generazione di alert (inviai a diversi destinatari e su canali differenti) se i valori non fossero conformi con le policy create.

La raccolta può avvenire direttamente in alcuni servizi, in altri bisogna installare un Agente.

Le policy, definite su una specifica metrica, contengono una condizione che può essere:

- Threshold condition: Specifica un valore sopra il quale (o sotto il quale) deve essere generata la notifica
- Metric absence: Specifica di inviare la notifica quando una metrica non ha dati per un certo intervallo di tempo

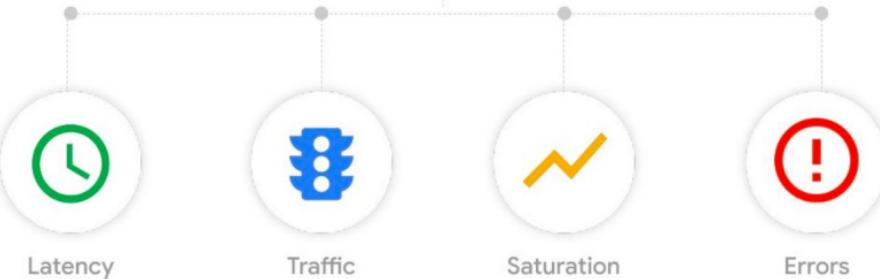


Per le notifiche va trovato un compromesso, infatti con troppe c'è l'alert fatigue (con il rischio di essere ignorate) e troppe poche possono non allertare di eventi importanti.

Site Reliability Engineering

Dopo la rilevazione e analisi bisogna porsi degli obiettivi di performance, affidabilità e disponibilità.

Performance e affidabilità possono essere determinate valutando 4 grandezze:



Latenza

Misura il tempo che il sistema impiega per restituire una risposta, importante perché:

- Ha effetto sulla User Experience degli utenti
- Un aumento può essere collegato ad un problema
- Un aumento può essere collegato ad un aumento di richieste
- Può essere usata per misurare l'impatto, in termini di miglioramento/peggioramento, generato da modifiche al sistema

Misurata nei seguenti modi:

- Tempo di caricamento di una pagina web
- Numero di richieste in coda – Tempo di esecuzione di una query sql
- Durata di una transazione
- Tempo impiegato per generare la prima risposta
- Tempo impiegato per restituire tutti i dati

Traffico

Misura il numero di richieste che arrivano al sistema. È importante perché:

- È un indicatore della domanda che arriva al sistema, quindi del carico di lavoro richiesto
- Il suo andamento storico può essere utilizzato per dimensionare le risorse e per anticipare necessità di scaling
- È utilizzabile per prevedere il costo dell'infrastruttura cloud

Misurata nei seguenti modi:

- Numero di richieste http per secondo
- Numero di richieste per contenuti statici vs numero di richieste per contenuti dinamici
- Network I/O
- Numero di sessioni concorrenti
- Numero di transazioni al secondo
- Numero di operazioni di lettura / scrittura

Saturazione

Misura quanto un sistema è vicino alla sua capacità massima. È importante perché:

- È un indicatore che rileva quanto un sistema è a pieno carico
- È un indicatore che considera le risorse più scarse, ovvero quelle che si saturano per prime
- Il suo aumento è spesso legato ad un degrado delle prestazioni dell'intero sistema

Misurata nei seguenti modi:

- % di memoria utilizzata
- % di thread pool utilizzata
- % di cache utilizzata
- % disco occupato
- % di cpu

Errori

Misura il numero di eventi che identificano una situazione di errore o di fallimento. È importante perché:

- Può indicare qualcosa che non sta funzionando a dovere. Un errore software o un hardware prossimo alla rottura
- Può indicare problemi di totale saturazione di una parte del sistema (p.e. disco pieno)

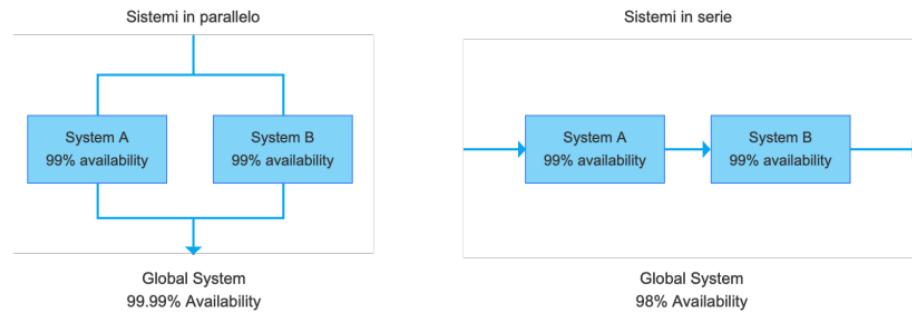
Misurata nei seguenti modi:

- Numero di risposte non corrette
- Numero di risposte HTTP con codice 400/500
- Numero di eccezioni nel codice – Numero di connessioni conclusive prematuramente

Availability

La disponibilità di un servizio è misurata come percentuale del tempo in cui il sistema è attivo e funzionante.

La disponibilità totale di un sistema composto da più sottosistemi dipende dalla disponibilità dei singoli sottosistemi e dal modo in cui essi sono collegati:



Service Level Indicator

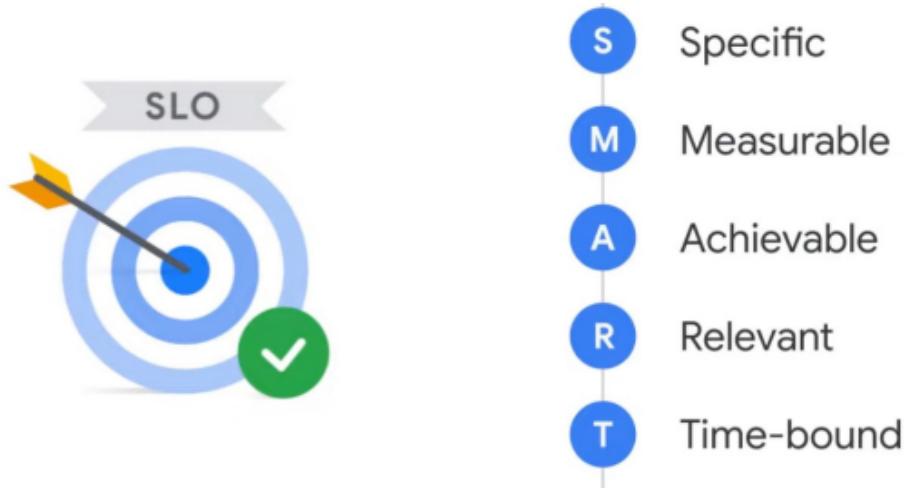
SLI, indicatore per valutare l'affidabilità di un sistema, calcolato come l'availability (dividendo il numero di eventi "positivi" per il numero di eventi totali).

Normalmente espressi in %, ma la tipologia di evento da considerare è libera e va scelta in modo rappresentativo per il sistema che si sta monitorando.

Service Level Object

SLO, target che si vuole raggiungere in un SLI, anche questi riferiti ad un intervallo di tempo.

Per definire un buon obiettivo si usa la regola SMART:



Error budget

Livello accettabile di non disponibilità che ogni SLO determina, trattato come un budget di "anomie" ammissibili su una metrica.

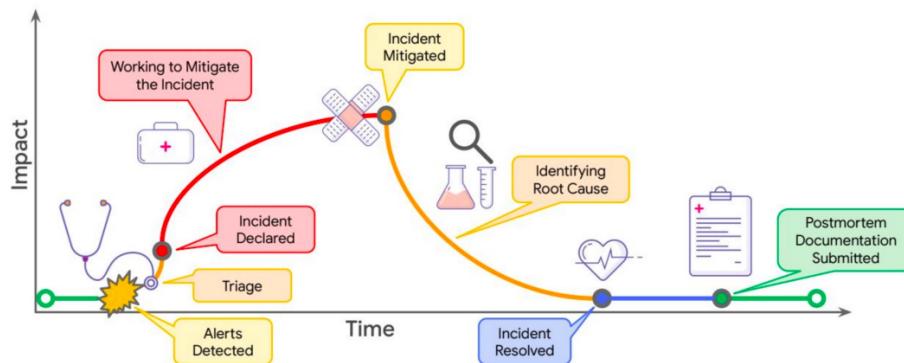
Service Level Agreement

SLA, contratto che rappresenta il livello di SLO promesso e garantito sotto forma di contratto con il cliente; con eventuale rimborso in caso di mancato rispetto.

Incident Management Process

Metodologia che indica una strategia d'azione per rispondere velocemente ed in maniera costruttiva agli eventi negativi, per evitare che un incidente di sistema possa avere seri impatti sugli utilizzatori del servizio.

Va definito a priori e periodicamente manutenuto e pianificato.

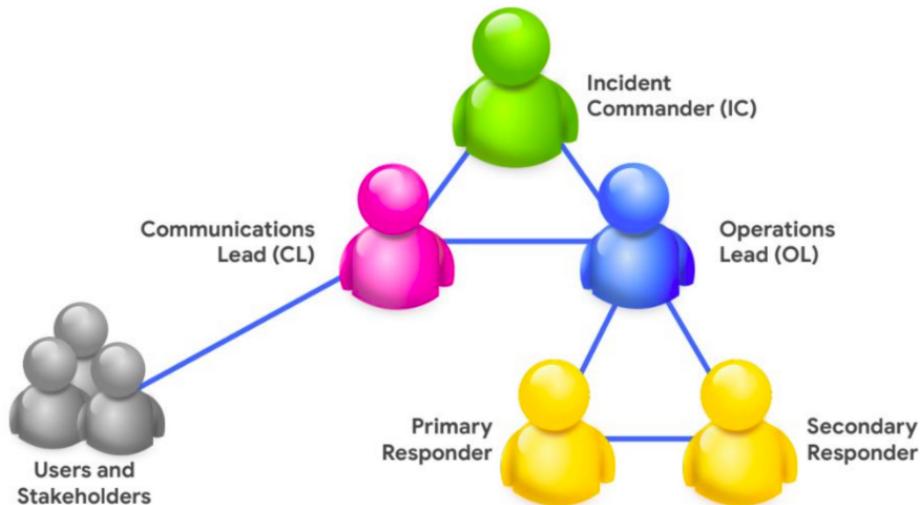


Le fasi sono:

- Mitigazione: applicazione di misure correttive per risolvere il problema o per mitigarne gli effetti
- Root cause analysis: ricerca e analisi delle cause che hanno portato alla nascita del problema (incentivando il principio della “blameless culture”, in cui non si cercano colpevoli ma solo problemi da risolvere)
- Resolved: individuata la causa e sistemata, l'incidente si considera risolto
- Post Mortem Documentation: documentazione dell'accaduto e delle azioni intraprese per risolvere l'incidente, in modo tale da migliorare l'intero processo per diminuire la probabilità che eventi simili accadano nuovamente e per aumentare la velocità di reazione nel caso riaccadano.

Esistono anche ruoli (Incident Management Roles):

- Incident Commander: figura apicale del team di gestione degli incidenti. Assembra e coordina il team e la gestione dell'incidente. Tipicamente non ha responsabilità tecniche.
- Communications Lead: tiene informate persone terze al team (utenti e ogni altro interessato) in maniera chiara e puntuale
- Operations Lead: gestisce il lavoro di risoluzione coordinando la comunicazione di IC e CL con il team tecnico
- Primary/secondary responder: svolgono le operazioni coordinati dall'OL



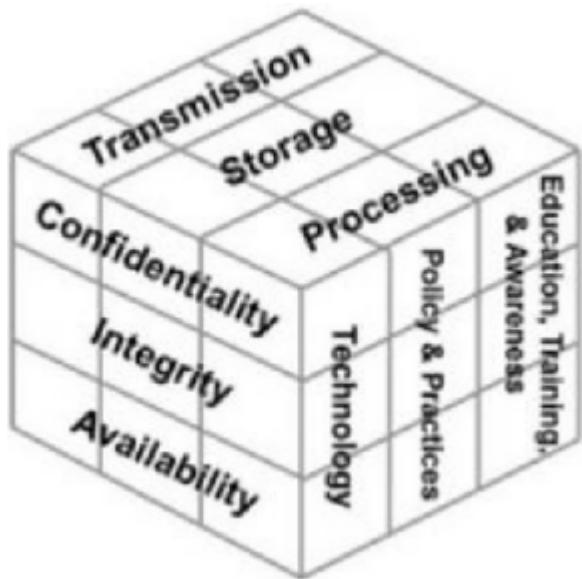
Cloud Security

Iniziamo definendo cos'è la sicurezza informatica, fare sicurezza informatica significa proteggere gli asset preservandone:

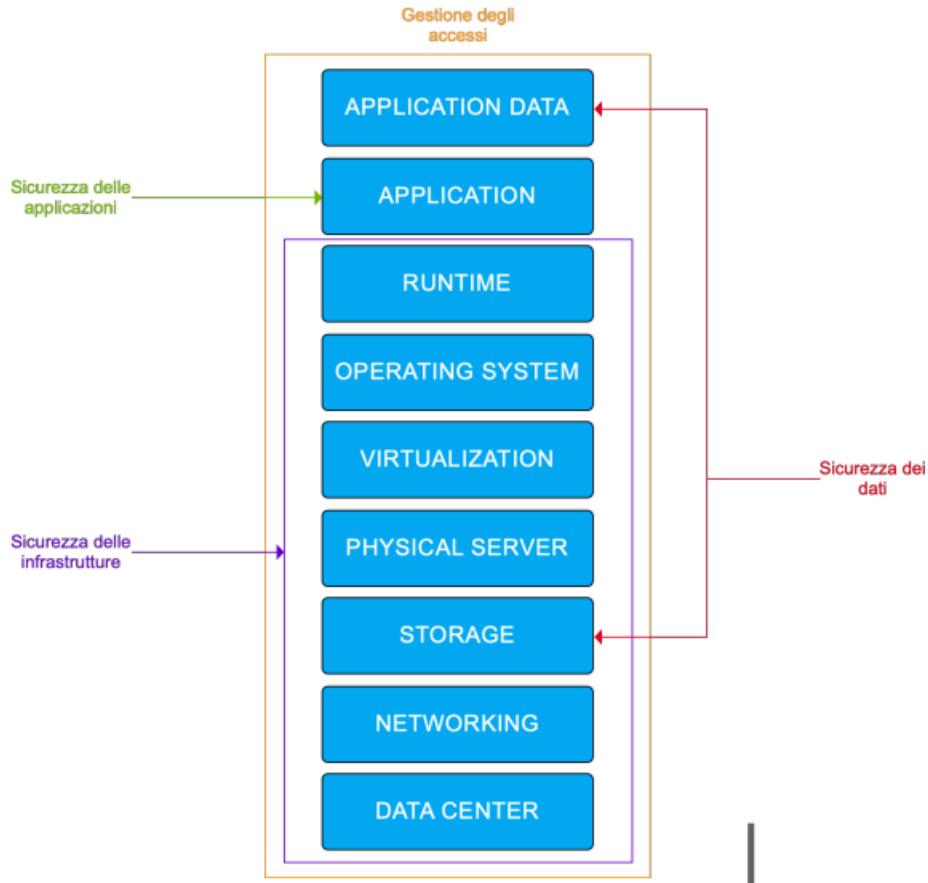
- Integrità: Non deve essere possibile effettuare modifiche non autorizzate.
- Disponibilità: Gli asset devono sempre essere disponibili per chi ne ha diritto.
- Riservatezza: L'accesso agli asset deve essere possibile solo a chi ne ha diritto.



Oltre e queste 3 realtà il modello McCumber introduce altre 2 facce: "gli stati delle informazioni" e i "sistemi di salvaguardia":



Quindi anche gli asset del cloud vanno messi in sicurezza, la security del cloud segue il modello della shared responsibility, dove chi è responsabile di un livello è responsabile anche della sua messa in sicurezza.



Ora vediamo i livelli nel dettaglio.

Gestione degli accessi

Autenticazione

Processo di verifica dell'identità di un "attore" che vuole accedere ad una risorsa.

Autenticazione ≠ Autorizzazione

Basata su un sistema a "fattori" che l'utente deve fornire per dimostrare la sua identità. Divisi in 3 categorie:

1. Fattori di conoscenza: ciò che l'utente conosce. Per esempio una password o la risposta ad una domanda di sicurezza
2. Fattori di possesso: ciò che l'utente possiede. Per esempio una tessera magnetica, un token fisico o uno smartphone

3. Fattori biometrici: sono rappresentati dalle caratteristiche fisiche dell'utente. Per esempio impronte digitali, scansione del viso...

I tipi di autenticazione, che si dividono per numero di fattori o tipo di fattore, sono:

- Basic Authentication: username e password. Usa un unico fattore facile da attaccare. Sconsigliata in molte casistiche
- Biometric Authentication: un solo dato biometrico (p.e. immagine del volto). Usa un unico fattore più difficile da attaccare
- Multi Factor Authentication (MFA): due fattori di tipo differente e che viaggiano su canali distinti (p.e. una password + un codice ricevuto sullo smartphone). Aumentano notevolmente il livello di sicurezza.

Autorizzazione

Processo che stabilisce a quali risorse un utente può accedere e quali tipologie di operazioni può eseguire. Si sa chi accede a cosa grazie ruoli e policy.

Le policy e i diritti di accesso alle risorse devono essere definiti e gestiti nel rispetto di alcuni principi fondamentali:

- Separation of duties: ogni policy ed ogni ruolo permettono di svolgere solamente un compito ben definito, senza mai andare in conflitto con gli altri.
- Least privilege: Ogni utente deve godere dei permessi minimi necessari per svolgere il suo compito, nulla di più per evitare abusi
- Centralizzazione: le policy devono essere gestite in modo centralizzato per agevolare e facilitare tutte le operazioni

GCP Resource Hierarchy

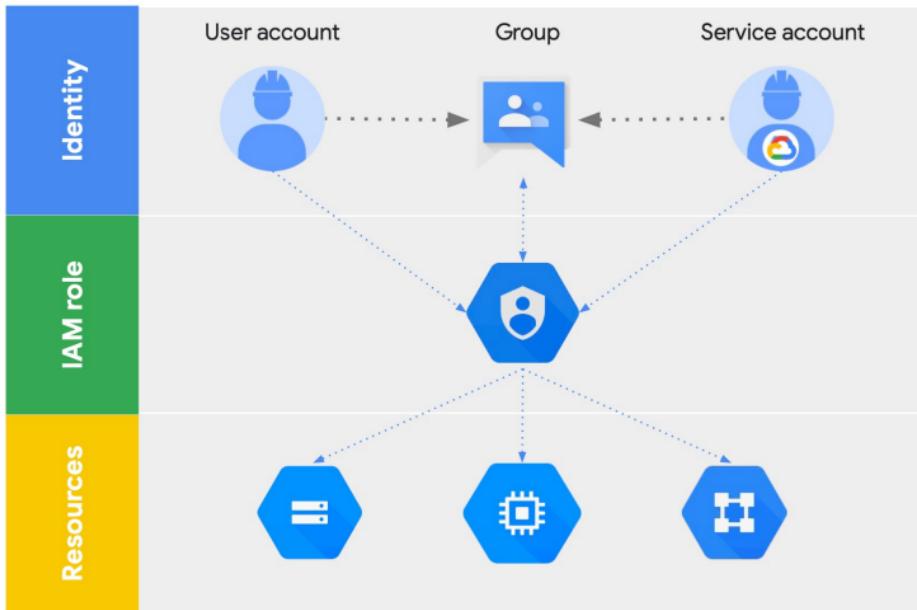
L'organizzazione gerarchica agevola la definizione delle policy applicate a qualsiasi livello e sono ereditate da tutti i livelli sottostanti

GCP IAM

GCP IAM (Identity and Access Management) è il sistema di GCP per la gestione dell'autorizzazione.

Si basa sui concetti di identità, di ruoli e di risorse

Una identità (un utente, un gruppo di utenti, un service account) possiede uno o più ruoli che determinano i diritti di accesso sulle risorse



GCP Service accounts

Utenze speciali utilizzate per identificare un servizio o un'applicazione che opera sull'ambiente cloud, in modo da poter accedere a risorse cloud, sia per utilizzarle, che per gestirle.

Trattati anche come risorse, per cui una particolare utenza può avere i permessi per accedervi ed impersonificarlo.

Sicurezza delle infrastrutture

Coinvolge i livelli che vanno dal data center fisico fino ai sistemi operativi virtualizzati, con attività differenti e specifiche per ogni livello.

- Sicurezza fisica: il data center ha molteplici livelli di sicurezza, sia per il controllo degli accessi (sorveglianza / videosorveglianza, scanner biometrici...) che per la prevenzione di eventi potenzialmente dannosi (antincendio, generatori ausiliari...)
- Sicurezza di rete: il traffico di rete, in entrata e in uscita, è messo in sicurezza tramite isolamento delle reti (VPN), firewall e connessioni cifrate (VPN)
- Virtualizzazione: le tecnologie di virtualizzazione sono protette con tecniche di isolamento degli hypervisor, segmentazione di rete per evitare che l'ambiente virtuale di un cliente non possa interferire con gli altri

- Secure boot: ci si assicura che solamente sistemi operativi fidati possano essere avviati per evitare che il software installato possa compromettere la sicurezza del sistema

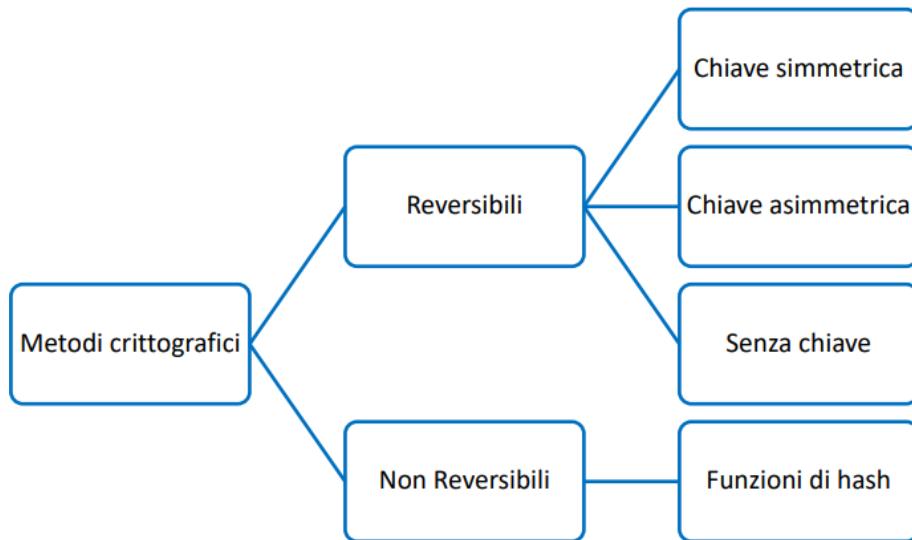
Sicurezza dei dati

La sicurezza dei dati si ottiene tramite cifratura (encryption), in modo da renderli utilizzabili solamente da chi è in possesso della chiave di decifratura, può avvenire a più livelli:

- A livello applicativo: esempio TLS
- A livello di sistema operativo: esempio file system cifrati
- A livello hardware: esempio dischi cifrati

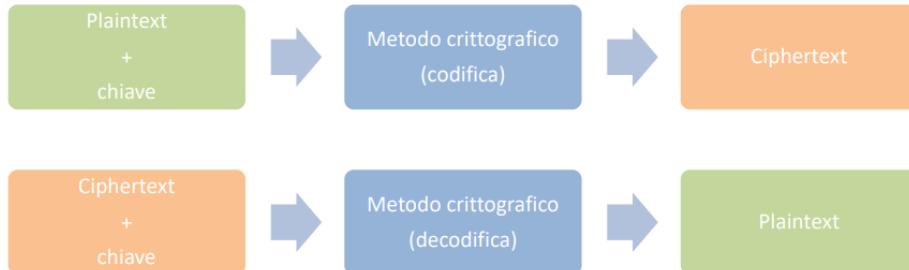
E può avvenire At rest o In transit.

Crittografia



Reversibili

Il processo di cifratura/decifratura è il seguente, con chiave simmetrica:



Esempio: AES o ECB

Se invece la chiave è asimmetrica, quindi c'è ne una per cifrare e una per decifrare e anche se si conosce una l'altra è impossibile da risalire il processo è:



Esempio: RSA o ECC

Ad una delle chiavi è assegnato il ruolo di CHIAVE PRIVATA e all'altra chiave è assegnato il ruolo di CHIAVE PUBBLICA. Solamente la chiave privata deve essere tenuta segreta, la pubblica può essere condivisa "liberamente".

Infatti uno dei principi fondamentali è:

Un metodo crittografico non deve basarsi sulla segretezza dell'algoritmo. Il metodo deve essere sicuro anche nel caso in cui l'algoritmo fosse noto. L'unica informazione da mantenersi segreta è la chiave di cifratura, da scegliersi in maniera completamente randomica

L'asimmetria delle chiavi permette di ottenere due applicazioni distinte:

- Riservatezza delle informazioni: Cifrando con la chiave pubblica solamente il possessore della relativa chiave privata può decifrare il messaggio
- Certezza del mittente: Cifrando con la propria chiave privata il ricevente è sicuro che sia stato io ad inviare il messaggio

Normative

Sono:

- GDPR (General Data Protection Regulation): Regolamento dell'unione europea che stabilisce come trattare in modo lecito i dati personali dei

cittadini.

- PCI DSS: (Payment Card Industry Data Security Standard): standard per assicurare un livello uniforme della sicurezza dei pagamenti effettuati con carte di credito
- HIPAA (Health Insurance Portability and Accountability Act): legge federale degli Stati Uniti che stabilisce come mantenere private e sicure informazioni sulla salute degli individui (healthcare providers, health plans)
- COPPA (Children's Online Privacy Protection Act): legge federale degli Stati Uniti che stabilisce come proteggere la privacy dei minori di 13 anni, anche in termini di consenso dei genitori per l'iscrizione a servizi online

Sicurezza applicazioni

Per avere una applicazione sicura dobbiamo basarci su questi due aspetti:

- Sicurezza dei componenti di terze parti
- Sicurezza del codice

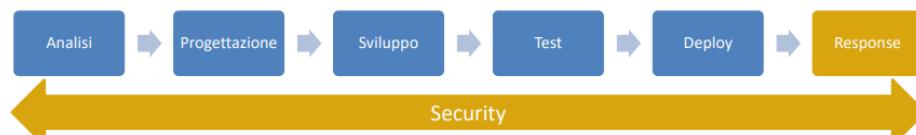
La sicurezza applicativa si differenzia rispetto ad altri settori della cybersecurity in quanto:

- Non è affrontabile con un prodotto
- Le Web application sono raggiungibili da chiunque: complicato applicare il principio di minimizzazione della superficie d'attacco

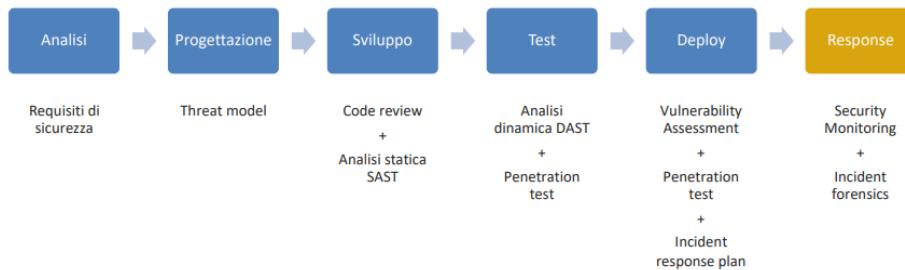
Quindi la sicurezza applicativa è:

- Il risultato di un processo di progettazione e sviluppo sicuro (SDLC)
- L'adozione di approcci di security by design (e non di security patching)
- L'esecuzione di test mirati ad identificare problemi nel codice

Quindi il ciclo di vita del software ora sarà così:



Dove ogni fase ha un'attività:



Alcune di queste attività nel dettaglio:

Vulnerability Assessment

O VA, processo AUTOMATICO di ricerca di vulnerabilità NOTE.

Con i Vulnerability Scanner è possibile analizzare l'asset da testare raccogliendo informazioni utili ad indicare la presenza di una vulnerabilità. Per funzionare si prendono le librerie con le versioni e si confrontano con un database che contiene tutte le vulnerabilità note.

Penetration Test

O PT, insieme di attività manuali volte ad identificare vulnerabilità utilizzando le stesse tecniche che utilizzerebbe un attaccante reale (ethical hacking).

PT In base all'asset testato:

- Network PT (NPT): Test dei servizi esposti su rete pubblica o privata
- Web App PT (WAPT): Test specifico di applicazioni web
- Mobile PT (MPT): Test specifico di applicazioni mobile
- Wi-Fi PT: Test specifico di infrastrutture Wi-Fi

PT In base alle regole di ingaggio:

- PT basato su linee guida (o semplicemente PT): Viene presa come riferimento una linea guida, per esempio OWASP WSTG, e vengono eseguiti tutti i controlli previsti.
- Bug Bounty (BB): Non c'è una linea guida di riferimento ma solo un perimetro di test. Il tester può effettuare le attività che ritiene più opportune rimanendo però all'interno del perimetro.
- Red team (RT): Non ci sono linee guida e il perimetro è appena accennato. Il tester può utilizzare qualsiasi tecnica, esattamente come un attaccante reale.

Analisi SAST

Static Application Security Testing, effettua la code review tramite sistemi

automatici, che prendono in input il codice sorgente e forniscono in output indicazioni sui punti del codice in cui sono presenti problemi di sicurezza.

Integrabili nella pipeline CI/CD.

Vulnerabilità applicative

- Injections
- Broken authentication
- Broken access control
- Broken session management
- XML external entities
- Security misconfiguration
- Cross site scripting (XSS)
- Uso di componenti con vulnerabilità note
- Cross site request forgery (CSRF)
- Server site request forgery (SSRF)
- Clickjacking
- Cross-origin resource sharing (CORS)
- HTTP request smuggling
- Insecure deserialization
- Directory traversal
- Business logic vulnerabilities

Injection

Quando non si gestiscono gli input provenienti dall'esterno, e quando inseriti nell'applicazione ne modificano il comportamento.

Le condizioni per renderla possibile:

1. Presenza di Source: input proveniente dall'esterno e manipolabile
2. Presenza di un Sink: parser che interpreta l'input e lo tramuta in azioni

Una volta identificati è possibile determinare il punto, denominato filter, nel quale effettuare le operazioni di gestione dell'input per renderlo "innocuo". Il Filter può fare:

- Validation: verificare l'input, deve essere solamente della tipologia che mi aspetto al contrario lo scarto. Deve essere fatta in modalità white list, ovvero esplicitando l'input valido e identificando tutto il resto come non valido (secure by default)
- Sanitization: Modifico l'input affinché possa essere utilizzato dal sink senza generare effetti indesiderati. Tipicamente si utilizza l'encoding dei dati, in modo da rendere "innocui" caratteri speciali che invece non lo sarebbero.

I tipi di injection si basano sul parser, per esempio la SQL Injection:

Tipologia di injection che più frequentemente troviamo nelle applicazioni web ed accade quando le query sql sono create concatenando l'input proveniente dall'esterno con parti fisse di sql.

WAF

Web Application Firewall, sistema di protezione che ispeziona il traffico diretto ad una web application, filtrando/sanificando quello potenzialmente dannoso. Implementato via hardware o software.

Non deve sostituire la scrittura di codice sicuro però può essere molto utile per proteggere applicazioni legacy, già scritte e difficili da cambiare.

GCP Cloud Armour

Sistema gestito per la protezione di applicazioni e siti web ospitate sull'infrastruttura cloud di Google che mette a disposizione un WAF e un sistema anti DDoS.

Rileva le vulnerabilità OWASP Top 10 e gestisce gli attacchi DDoS perimetrali.