



RAPPORTI TECNICI INGV

Progettazione e implementazione
di un Cluster Kubernetes on-premis



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

496

Direttore Responsabile

Daniela VERSACE

Editor in Chief

Milena MORETTI (editorinchief.collane-editoriali@ingv.it)

Editorial Board

Laura ALFONSI (laura.alfonsi@ingv.it)
Christian BIGNAMI (christian.bignami@ingv.it)
Simona CARANNANTE (simona.carannante@ingv.it)
Viviana CASTELLI (viviana.castelli@ingv.it)
Luca COCCHI (luca.cocchi@ingv.it)
Luigi CUCCI (luigi.cucci@ingv.it)
Lorenzo CUGLIARI (lorenzo.cugliari@ingv.it)
Alessia DI CAPRIO (alessia.dicaprio@ingv.it)
Roberto DI MARTINO (roberto.dimartino@ingv.it)
Domenico DI MAURO (domenico.dimauro@ingv.it)
Domenico DORONZO (domenico.doronzio@ingv.it)
Filippo GRECO (filippo.greco@ingv.it)
Alessandro IAROCCI (alessandro.iarocci@ingv.it)
Mario MATTIA (mario.mattia@ingv.it)
Daniele MELINI (daniele.melini@ingv.it)
Anna NARDI (anna.nardi@ingv.it)
Lucia NARDONE (lucia.nardone@ingv.it)
Marco OLIVIERI (marco.olivieri@ingv.it)
Pierangelo ROMANO (pierangelo.romano@ingv.it)
Maurizio SOLDANI (maurizio.soldani@ingv.it)
Sara STOPPONI (sara.stopponi@ingv.it)
Umberto TAMMARO (umberto.tammaro@ingv.it)
Andrea TERTULLIANI (andrea.tertulliani@ingv.it)
Stefano URBINI (stefano.urbini@ingv.it)

Ufficio Editoriale

Francesca DI STEFANO Coordinatore - Segreteria di Redazione

Produzione e grafica-redazionale

Barbara ANGIONI
Massimiliano CASCONI
Francesca DI STEFANO
Patrizia PANTANI

REGISTRAZIONE AL TRIBUNALE DI ROMA N.174 | 2014, 23 LUGLIO

© 2014 INGV Istituto Nazionale di Geofisica e Vulcanologia | Rappresentante legale: Carlo DOGLIONI

Sede: Via di Vigna Murata, 605 | Roma



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

RAPPORTI TECNICI INGV

Progettazione e implementazione di un Cluster Kubernetes on-premis

Design and implement a Kubernetes Cluster on-premis

Gianpaolo Sensale

INGV | Istituto Nazionale di Geofisica e Vulcanologia, Amministrazione Centrale

Accettato 25 marzo 2025 | Accepted 25 March 2025

Come citare | How to cite Sensale G., (2025). Progettazione e implementazione di un Cluster Kubernetes on-premis. Rapp. Tec. INGV, 496: 1-34,
<https://doi.org/10.13127/rpt/496>

In copertina Esempio del flusso di sincronizzazione tra il repository Git e il cluster Kubernetes tramite Argo CD | Cover Example of the
synchronization flow between the Git repository and the Kubernetes cluster via Argo CD

496

INDICE

Riassunto	7
<i>Abstract</i>	7
Introduzione	8
1. Cloud Computing	9
1.1 Infrastructure as a Service (IaaS)	9
1.1.1 Kubernetes e IaaS	9
1.2 Platform as a Service (PaaS)	9
1.2.1 Kubernetes e PaaS	10
1.3 Software as a Service (SaaS)	10
1.3.1 Kubernetes e SaaS	10
1.4 Kubernetes nel contesto delle infrastrutture cloud	10
1.5 Aggiornamenti sulle tendenze attuali	11
1.6 Vantaggi e sfide del cloud computing	11
1.7 Sfide del cloud computing: sicurezza e privacy dei dati	12
1.8 Normative sulla sicurezza e privacy nel cloud computing	12
1.9 Considerazioni sulla conformità normativa	13
2. Cluster Kubernetes	13
2.1 Terminologia	14
2.2 Architettura Cluster Kubernetes On-Premis	14
2.3 MAAS (Metal As A Service)	15
2.4 Server QEMU	15
2.5 Nodo Master	16
2.6 Nodo ETCD	17
2.7 Nodo Worker	18
2.8 SAN (Storage Area Network)	19
3. L'algoritmo RAFT	20
3.1 Concetti chiave di RAFT	20
3.2 Funzionamento del protocollo RAFT	20
4. Confronto tra sistemi deployment	21
4.1 Sistemi tradizionali su server fisici	21
4.2 Virtualizzazione	22
4.3 Container	22
4.3.1 Scalabilità dei container	23
4.3.2 Scalabilità Orizzontale (Scaling Out)	23
4.3.3 Scalabilità Verticale (ScalingUp)	24
5. Sicurezza informatica	25
6. Continuous Delivery mediante Argo CD	25

7. Risultati Ottenuti	26
7.1 Sviluppi futuri	27
8. Conclusioni	28
Bibliografia	29
Sitografia	29
Acronimi	29

Riassunto

Questo articolo descrive la progettazione e l'implementazione di un'infrastruttura *Cluster Kubernetes on-premise* presso l'Istituto Nazionale di Geofisica e Vulcanologia (INGV) di Roma. Il progetto è nato dall'esigenza di modernizzare l'infrastruttura esistente, basata su una *WebFarm* monolitica, che mostrava crescenti limitazioni in termini di scalabilità, sicurezza e gestione delle risorse IT. La precedente architettura richiedeva continui interventi manuali, risultava inefficiente nella gestione degli indirizzi IP (*Internet Protocol*) e vulnerabile a frequenti attacchi informatici. L'obiettivo principale è stato la migrazione di applicazioni e servizi *web* a una nuova architettura *cluster* che garantisca una gestione automatizzata, maggiore affidabilità e flessibilità operativa. Il sistema *Kubernetes* è stato scelto per le sue capacità di orchestrazione dei *container*, migliorando sia la scalabilità orizzontale che verticale, oltre a incrementare la sicurezza e ottimizzare l'uso delle risorse, esso è in grado di rilevare e gestire guasti in modo autonomo. Se un *container* fallisce o non risponde, *Kubernetes* lo sostituirà automaticamente, assicurando la disponibilità continua dell'applicazione. Il sistema gestisce anche il bilanciamento del carico tra le istanze di servizio in modo efficiente. Viene inoltre presentato un confronto tra diverse architetture di *deployment*, e si analizzano le soluzioni implementate per la gestione dei *cluster Kubernetes* e per la protezione delle applicazioni.

È stato implementato un sistema di *Continuous Delivery* (*ArgoCD*) che automatizza il processo di *deployment*, riducendo il lavoro manuale e migliorando la coerenza tra i vari ambienti. Le applicazioni vengono automaticamente distribuite quando viene rilevata una modifica nel *repository Git*. In questo modo, ogni aggiornamento viene applicato in modo controllato e ripetibile.

Infine, vengono discussi gli sviluppi futuri, che includono la federazione di più *cluster* geograficamente distribuiti per aumentare ulteriormente la resilienza e la disponibilità dei servizi.

Abstract

This paper describes the design and implementation of an on-premise Kubernetes Cluster infrastructure at the National Institute of Geophysics and Volcanology (INGV) in Rome. The project was born from the need to modernize the existing infrastructure, based on a monolithic WebFarm, which showed increasing limitations in terms of scalability, security and IT resource management. The previous architecture required continuous manual interventions, was inefficient in managing IP addresses and vulnerable to frequent cyber-attacks. The main objective was to migrate applications and web services to a new cluster architecture that ensures automated management, greater reliability and operational flexibility. The Kubernetes system was chosen for its container orchestration capabilities, improving both horizontal and vertical scalability, as well as increasing security and optimizing resource use, it is able to detect and manage failures autonomously. If a container fails or becomes unresponsive, Kubernetes will automatically replace it, ensuring continuous availability of the application. The system also manages load balancing between service instances efficiently.

A comparison between different deployment architectures is also presented, and the solutions implemented for Kubernetes cluster management and application protection are analyzed.

A Continuous Delivery system (ArgoCD) has been implemented that automates the deployment process, reducing manual work and improving consistency between environments. Applications are automatically deployed when a change is detected in the Git repository. In this mode, each update is applied in a controlled and repeatable manner. Finally, future developments are discussed, including the federation of multiple geographically distributed clusters to further increase the resilience and availability of services.

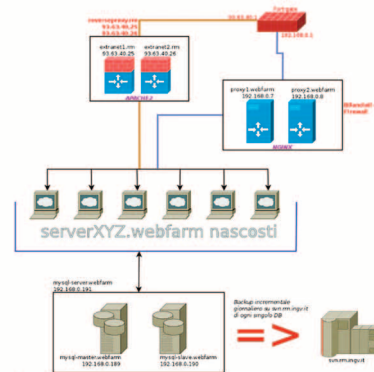
Introduzione

L'Istituto Nazionale di Geofisica e Vulcanologia (INGV [1]) ha sviluppato e gestito per anni una sottorete informatica denominata *WebFarm*, presso la sede centrale di Roma (Figura 1). Questa infrastruttura è stata creata per ospitare una vasta gamma di servizi *web*, rispondendo a due necessità principali:

1. Isolamento dei servizi *web* pubblici. I server responsabili della distribuzione di servizi *web* al pubblico erano frequentemente soggetti a tentativi di attacco informatico. Creare una sottorete dedicata ha permesso di proteggere questi server da potenziali minacce esterne, migliorando la sicurezza informatica complessiva.
2. Ottimizzazione dell'uso degli indirizzi IP (*Internet Protocol*). In precedenza, a ogni sito *web* ospitato sulla rete era associato un indirizzo IP distinto. Questo approccio non era sostenibile nel lungo termine, poiché il numero di IP disponibili era limitato. La *WebFarm* ha contribuito a ridurre significativamente il consumo di indirizzi IP, centralizzando i servizi *web* e riducendo la dipendenza da risorse IP uniche.

Figura 1 Schema dell'architettura *WebFarm* dell'INGV, con protezione tramite server di frontiera e politiche di accesso basate su ACL (*Access Control List*) e autenticazione.

Figure 1 Diagram of the INGV *WebFarm* architecture, with protection via border servers and access policies based on ACLs (*Access Control List*) and authentication.



Con l'aumento delle minacce alla sicurezza informatica e la necessità di un'architettura IT più flessibile e scalabile, l'INGV ha riconosciuto i limiti della propria infrastruttura *WebFarm*, che non era più in grado di supportare adeguatamente le crescenti esigenze operative. La migrazione verso *Kubernetes* [2] è stata guidata dal bisogno di automatizzare la gestione delle risorse, migliorare la protezione dei servizi *web* e ottimizzare l'uso delle risorse *hardware* e di rete, permettendo una gestione più efficiente delle applicazioni critiche.

A seguito della migrazione dei servizi alla *WebFarm*, l'INGV ha implementato una struttura di protezione e restrizione dei server utilizzando server di frontiera (*reverse proxy*). Questo ha permesso di ridurre il carico operativo sui server principali e sui rispettivi amministratori di sistema. Le soluzioni adottate per la protezione dei server erano standardizzate, facili da gestire e applicabili a tutti i servizi ospitati (vedi Figura 1). Le misure di sicurezza implementate si dividevano in tre categorie:

1. Servizi pubblici senza restrizioni: accessibili a tutto il pubblico, senza necessità di autenticazione.
2. Servizi con accesso ristretto tramite *Access Control List* (ACL): l'accesso era limitato a specifiche entità autorizzate, identificate tramite indirizzo IP.
3. Servizi con autenticazione obbligatoria: l'accesso era riservato a utenti con credenziali valide, garantendo che solo le entità autorizzate potessero accedere ai servizi.

Inoltre, ogni sito ospitato poteva richiedere uno o più *database* per gestire i propri dati, con l'accesso controllato tramite il sistema di autenticazione interno del *database management system* (DBMS). Con l'avanzare delle esigenze dell'ente e del contesto tecnologico, è emersa la necessità di

un'infrastruttura più moderna, capace di supportare lo sviluppo di *software* basato su architetture a microservizi. In questo contesto di trasformazione digitale, l'INGV ha avviato un processo di riprogettazione delle sue applicazioni, molte delle quali basate su un'architettura monolitica e ormai obsoleta. L'obiettivo è passare a un modello di microservizi, che consente uno sviluppo più rapido e modulare, favorendo l'innovazione e accelerando il rilascio di nuove funzionalità sul mercato. I microservizi, per la loro natura distribuita, vengono eseguiti all'interno di *container*, e *Kubernetes* è stato scelto come l'orchestratore ideale per gestire questi *container*. *Kubernetes* consente di automatizzare e ottimizzare l'utilizzo delle risorse IT, migliorando l'efficienza operativa e garantendo la scalabilità necessaria per supportare le crescenti esigenze dell'ente.

1. Cloud Computing

Il *Cloud Computing* è un modello di erogazione di servizi informatici tramite *Internet*. Invece di ospitare *software*, dati e risorse su un *computer* o *server* locale, le aziende possono accedere a questi servizi tramite *Internet* da parte di *provider* di servizi *cloud*.

Questo approccio consente alle aziende di risparmiare sui costi di infrastruttura, scalare facilmente le risorse in base alle esigenze e accedere a una vasta gamma di servizi e applicazioni.

Ci sono vari tipi di servizi di *cloud computing*:

- *Infrastructure as a Service* (IaaS);
- *Platform as a Service* (PaaS);
- *Software as a Service* (SaaS).

1.1 Infrastructure as a Service (IaaS)

IaaS fornisce alle aziende l'accesso a risorse di infrastruttura virtualizzate, come *server*, *storage* e reti, in un ambiente *cloud*. Questo modello offre la massima flessibilità, permettendo alle organizzazioni di creare e gestire le proprie applicazioni su macchine virtuali, con la possibilità di scalare le risorse in base alle esigenze.

1.1.1 Kubernetes e IaaS

In un ambiente IaaS, *Kubernetes* viene utilizzato per orchestrare i *container* distribuiti su macchine virtuali fornite da *provider* IaaS come AWS, Microsoft Azure o Google Cloud. *Kubernetes* ottimizza l'allocazione delle risorse, distribuendo i carichi di lavoro tra diversi nodi e automatizzando il *provisioning*, il ridimensionamento e la gestione delle risorse IT, migliorando l'efficienza complessiva. Ad esempio, su AWS, *Kubernetes* può essere integrato con EC2 per creare *cluster* elastici in grado di scalare dinamicamente in base al traffico delle applicazioni.

Con l'evoluzione delle infrastrutture *cloud*, *Edge Computing* sta diventando una tecnologia emergente complementare all'IaaS. L'*Edge Computing* estende le risorse di elaborazione vicino ai dispositivi periferici, riducendo la latenza e migliorando l'efficienza per applicazioni in tempo reale come *Internet of Things* (IoT) e reti 5G. *Kubernetes* è stato esteso anche per supportare ambienti *edge*, facilitando il *deployment* di applicazioni containerizzate in scenari distribuiti.

1.2 Platform as a Service (PaaS)

PaaS fornisce un ambiente completo di sviluppo e gestione delle applicazioni senza che le

aziende debbano preoccuparsi della gestione dell'infrastruttura sottostante. Questo modello consente agli sviluppatori di concentrarsi sullo sviluppo e l'implementazione di *software*, lasciando la gestione dell'*hardware*, del sistema operativo e delle piattaforme di sviluppo al *provider PaaS*.

1.2.1 Kubernetes e PaaS

In un ambiente PaaS, *Kubernetes* si integra per orchestrare i *container* e gestire le risorse necessarie per le applicazioni. Molte piattaforme PaaS moderne, come *OpenShift* di *Red Hat*, si basano su *Kubernetes* per fornire un ambiente di sviluppo e distribuzione di applicazioni "containerizzate". *Kubernetes* semplifica l'integrazione con CI/CD (*Continuous Integration/Continuous Delivery*), automatizzando il *deployment* delle applicazioni in ambienti PaaS.

Una delle tecnologie emergenti nel PaaS è il *Serverless Computing*, dove il codice viene eseguito in modo dinamico senza che gli sviluppatori debbano gestire l'infrastruttura sottostante. *Kubernetes* può essere utilizzato per gestire architetture *serverless*, fornendo scalabilità e orchestrazione automatica delle funzioni *serverless*. Esempi di piattaforme *serverless* che utilizzano *Kubernetes* includono *Knative* e *Kubeless*.

1.3 Software as a Service (SaaS)

SaaS è il modello più accessibile e diffuso di *cloud computing*, in cui le applicazioni *software* vengono fornite direttamente tramite *Internet*. Gli utenti accedono alle applicazioni tramite un *browser*, senza la necessità di installare o gestire il *software* localmente. L'intera gestione dell'applicazione, dall'aggiornamento alla sicurezza, è a carico del *provider SaaS*.

1.3.1 Kubernetes e SaaS

Kubernetes facilita l'implementazione di applicazioni SaaS grazie alla sua capacità di gestire applicazioni containerizzate distribuite e altamente scalabili. Le aziende che offrono servizi SaaS utilizzano *Kubernetes* per garantire la disponibilità continua delle loro applicazioni, scalando automaticamente i *container* in base al carico e migliorando la resilienza del sistema. Ad esempio, piattaforme SaaS come *Dropbox* [3] o *Slack* [4] utilizzano *container* orchestrati da *Kubernetes* per gestire milioni di utenti simultanei con elevate prestazioni e affidabilità.

Nel contesto SaaS, tecnologie come *AI-as-a-Service* stanno emergendo come un nuovo paradigma nel *cloud computing*. Piattaforme SaaS integrate con l'Intelligenza Artificiale (AI) consentono alle aziende di accedere a potenti strumenti di intelligenza artificiale, come il riconoscimento vocale e l'analisi dei dati, tramite servizi *cloud*. *Kubernetes* può essere utilizzato per orchestrare i modelli di intelligenza artificiale distribuiti, facilitando l'elaborazione di grandi dataset in tempo reale.

1.4 Kubernetes nel contesto delle infrastrutture cloud

Kubernetes è progettato per essere altamente versatile e adattabile, rendendolo uno strumento ideale per orchestrare applicazioni in una vasta gamma di infrastrutture *cloud*, inclusi ambienti *on-premise* e *cloud ibrido*.

- **On-Premise:** In ambienti *on-premise*, *Kubernetes* viene utilizzato per gestire *container* su infrastrutture locali, consentendo alle organizzazioni di sfruttare i benefici del *cloud computing* senza dover dipendere da *provider cloud esterni*. Questa configurazione è particolarmente utile per settori che richiedono il controllo completo dei dati, come finanza e sanità, dove la conformità normativa è critica.
- **Cloud Ibrido:** *Kubernetes* consente una perfetta integrazione tra ambienti *on-premise* e *cloud pubblico*, supportando configurazioni di *cloud ibrido*. Le aziende possono distribuire carichi di lavoro in ambienti diversi, mantenendo la flessibilità di spostare i carichi di lavoro tra *on-premise* e *cloud pubblico*, a seconda delle esigenze di scalabilità e dei costi. Con strumenti come *Google Anthos* o *Azure Arc*, *Kubernetes* permette di gestire *cluster* distribuiti in modo centralizzato, indipendentemente dal fatto che si trovino su *cloud pubblico* o infrastrutture private.

1.5 Aggiornamenti sulle tendenze attuali

Oltre a *Kubernetes*, altre tendenze emergenti nel *cloud computing* includono:

- **Multicloud.** L'utilizzo di più *provider* di servizi *cloud* (come *AWS*, *Azure*, *Google Cloud*) è sempre più comune. *Kubernetes* facilita la gestione *multicloud*, permettendo alle aziende di distribuire i loro *container* su diversi *provider cloud* per ridurre i rischi e migliorare la resilienza.
- **Sicurezza nel Cloud.** La sicurezza rimane una priorità fondamentale, con soluzioni come *Zero Trust Security* che diventano più prevalenti. *Kubernetes* supporta politiche di sicurezza avanzate, come il controllo degli accessi basato sui ruoli (*RBAC*) e l'isolamento dei *container* tramite *network policies*.
- **Automation and AI Ops.** L'automazione è cruciale per la gestione delle infrastrutture *cloud*. *Kubernetes* si integra con strumenti di *AI Ops* che automatizzano la gestione operativa attraverso intelligenza artificiale e *machine learning*, migliorando l'efficienza e riducendo gli errori umani.

1.6 Vantaggi e sfide del cloud computing

Il *cloud computing* offre numerosi vantaggi chiave che hanno rivoluzionato il modo in cui le organizzazioni gestiscono le loro risorse IT. Questi vantaggi sono trasversali ai modelli di *IaaS*, *PaaS* e *SaaS*, e includono:

- **Flessibilità:** le organizzazioni possono adattare rapidamente le loro risorse IT alle necessità del momento. Ad esempio, nel modello *IaaS*, l'infrastruttura può essere scalata dinamicamente, consentendo di aumentare o ridurre la capacità di calcolo senza investimenti in *hardware* fisico.
- **Scalabilità:** i modelli *PaaS* e *SaaS* garantiscono che le applicazioni possano essere scalate in base alle richieste, riducendo gli sprechi di risorse e ottimizzando le prestazioni. *Kubernetes*, come discusso nel Paragrafo 2.4, facilita questo processo automatizzando l'orchestrazione dei *container* per gestire le applicazioni scalabili in ambienti *cloud* e *on-premise*.
- **Riduzione dei costi:** nel *cloud computing*, le aziende pagano solo per le risorse effettivamente utilizzate. Questo è particolarmente rilevante nei modelli *IaaS*, dove le macchine virtuali e lo *storage* sono erogati *on-demand*. *Kubernetes* permette anche di ottimizzare ulteriormente l'uso delle risorse IT in ambienti containerizzati, riducendo i costi associati all'infrastruttura.

- Accesso globale ai dati e ai servizi: i servizi SaaS offrono la possibilità di accedere a dati e applicazioni da qualsiasi parte del mondo, senza che l'utente debba preoccuparsi della gestione dell'infrastruttura sottostante.

Tuttavia, nonostante questi vantaggi, il *cloud computing* presenta anche delle sfide significative, che le organizzazioni devono affrontare per garantire la sicurezza e la conformità normativa.

1.7 Sfide del cloud computing: sicurezza e privacy dei dati

Indipendentemente dal modello di *cloud* adottato (IaaS, PaaS, SaaS), le sfide legate alla sicurezza, alla *privacy* dei dati e alla gestione delle risorse sono comuni a tutte le infrastrutture *cloud*, sia *on-premise* che ibride. Come evidenziato nel Paragrafo 2.4, *Kubernetes* facilita l'orchestrazione delle risorse in *cloud* ibridi, ma è comunque essenziale che le aziende adottino misure di sicurezza adeguate.

- Sicurezza dei dati: le aziende devono proteggere i dati sensibili nel *cloud*, soprattutto quando utilizzano infrastrutture IaaS, dove sono responsabili della sicurezza della configurazione dei *server* virtuali. Questo include la protezione delle reti, del sistema operativo e delle applicazioni eseguite su queste piattaforme. *Kubernetes*, integrato con soluzioni di sicurezza come RBAC (*Role-Based Access Control*) e politiche di rete, può aiutare a proteggere le applicazioni containerizzate.
- Normative sulla *privacy* dei dati: l'adozione del *cloud* richiede conformità a normative locali e internazionali sulla *privacy* dei dati. Le normative come il Regolamento Generale sulla Protezione dei Dati (GDPR) [5] nell'unione europea influenzano l'adozione di soluzioni SaaS e PaaS, poiché le organizzazioni devono garantire che i dati personali siano trattati in conformità con le leggi applicabili. *Kubernetes*, in ambienti ibridi, permette di mantenere il controllo dei dati *on-premise* o di scegliere con attenzione la localizzazione geografica dei dati nel *cloud* pubblico per rispettare queste normative.

1.8 Normative sulla sicurezza e privacy nel cloud computing

Le normative sulla sicurezza e la *privacy* variano da paese a paese e devono essere considerate attentamente dalle aziende quando decidono di adottare soluzioni *cloud*. Alcune delle principali normative e regolamenti includono:

- Regolamento generale sulla protezione dei dati (GDPR): applicabile a tutte le organizzazioni che trattano dati personali di cittadini dell'UE, impone regole rigorose sulla protezione e sulla gestione dei dati. Questo è particolarmente rilevante per i modelli SaaS e PaaS, dove i dati degli utenti sono archiviati e gestiti nel *cloud*.
- ISO/IEC 27001: questa norma internazionale riguarda la gestione della sicurezza delle informazioni e viene spesso richiesta dai provider di servizi *cloud* per dimostrare che le loro infrastrutture, sia in ambito IaaS che PaaS, sono conformi agli standard di sicurezza.
- Leggi sulla localizzazione dei dati: alcuni Paesi richiedono che i dati dei cittadini siano archiviati all'interno dei confini nazionali, il che può avere un impatto sull'adozione di soluzioni IaaS e PaaS globali. Questo è un problema particolarmente complesso da gestire in ambienti *cloud* ibrido, dove *Kubernetes* può essere utilizzato per orchestrare applicazioni che rispettano le normative locali sulla localizzazione dei dati.

1.9 Considerazioni sulla conformità normativa

Le aziende che utilizzano il *cloud* devono tenere conto delle normative sulla sicurezza e sulla *privacy* dei dati, che si applicano sia a livello locale che globale. Le responsabilità in merito alla conformità variano a seconda del modello di servizio adottato (IaaS, PaaS, SaaS):

- Nei modelli IaaS, la responsabilità della sicurezza dell'infrastruttura ricade sull'organizzazione che gestisce i server virtuali. *Kubernetes* può essere utilizzato per orchestrare i carichi di lavoro in un ambiente sicuro, distribuendo automaticamente i *container* su più nodi per garantire la resilienza e la protezione dei dati.
- Nei modelli SaaS, la responsabilità è principalmente del *provider* del servizio, che deve garantire la conformità alle normative come GDPR, *Health Insurance Portability and Accountability Act* (HIPAA) e *California Consumer Privacy Act* (CCPA), a seconda della localizzazione e del tipo di dati trattati.
- Le soluzioni PaaS richiedono che le aziende rispettino la sicurezza a livello di applicazione, garantendo che il codice sia protetto e che i dati sensibili siano criptati. *Kubernetes*, con l'integrazione di soluzioni di sicurezza avanzate, può facilitare la protezione delle applicazioni distribuite su infrastrutture PaaS.

La conformità normativa nel *cloud computing* è un processo continuo. Le leggi e i regolamenti possono evolversi nel tempo, e le organizzazioni devono essere pronte a rivedere le loro pratiche per garantire la protezione dei dati e la conformità a lungo termine.

2. Cloud Kubernetes On-Premis

Kubernetes [2] è stato creato da Google come progetto *open source* per l'orchestrazione dei *container* interna ai *Googleplex*. Dal 2015 *Kubernetes* è stato ceduto alla *Cloud Native Computing Foundation*, nata all'interno della *Linux Foundation Kubernetes*, ottimizzando l'uso delle risorse IT con conseguenti vantaggi dal punto di vista di tempi e costi. Il nome "*Kubernetes*" trae origine da una parola greca che significa timoniere o pilota; infatti, il logo ufficiale è un timone (Figura 2).

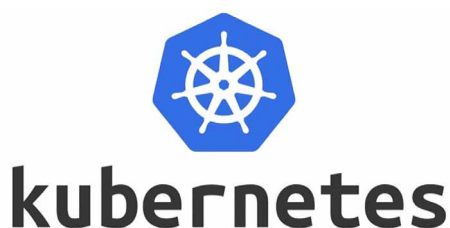


Figura 2 Logo ufficiale *Kubernetes*.
Figure 2 Official *Kubernetes* logo.

L'abbreviazione di *Kubernetes*, K8s, deriva dal conteggio delle otto lettere che separano la "K" dalla "s". Mediante questo orchestratore lo sviluppo, rilascio e distribuzione diventano più semplici e rapidi [Carrión, 2022], mentre l'allocazione dinamica delle risorse permette di ridurre gli sprechi e i costi. Lo *scale-down* automatico, infatti, evita l'utilizzo di risorse non necessarie. In caso di picchi di traffico, l'*autoscaling* aumenta la disponibilità e migliora le prestazioni dei servizi, garantendo continuità operativa. *Kubernetes* è inoltre indispensabile per sfruttare al meglio soluzioni di *multi-cloud* o *cloud* ibrido, poiché garantisce che le applicazioni funzionino in qualsiasi ambiente, pubblico o privato, senza perdite di funzionalità o prestazioni.

Le caratteristiche principali di *Kubernetes* sono:

- orchestrare i *container* su *host* multipli;
- ottimizzare l'uso dell'*hardware* per massimizzare le risorse;

- controllare e automatizzare il *deployment* e gli aggiornamenti delle applicazioni;
- montare e aggiungere *storage* per eseguire app *stateful*;
- gestire rapidamente la scalabilità delle applicazioni containerizzate e delle loro risorse;
- monitorare lo stato di integrità delle applicazioni e correggere problemi attraverso il riavvio, la replica e lo *scaling* automatico.

2.1 Terminologia

Nel contesto di *Kubernetes*, è importante comprendere i seguenti termini:

- *Master*: la macchina che controlla i nodi *Kubernetes*. È il punto di origine di tutte le attività assegnate.
- *Nodi/Worker*: queste macchine eseguono le attività assegnate richieste. Sono controllate dal nodo *master* di *Kubernetes*.
- *Pod*: un gruppo di uno o più *container* distribuiti su un singolo nodo. Tutti i *container* presenti in un *Pod* condividono indirizzo IP, *inter-process communication* (IPC), nome *host* ed altre risorse. I *Pod* astraggono la rete e lo *storage* dal *container* sottostante, consentendoti di spostare i *container* nei *cluster* con maggiore facilità.
- *Controller* di replica: controlla il numero di copie identiche di un *Pod* che devono essere eseguite in un punto preciso sul *cluster*.
- *Kubelet*: questo servizio viene eseguito sui nodi, legge i *manifest* del *container* e garantisce che i *container* definiti vengano avviati ed eseguiti.
- *Kubectl*: lo strumento di configurazione della riga di comando di *Kubernetes*.

2.2 Architettura Cluster Kubernetes On-Premis

L'architettura del *Cluster Kubernetes* realizzato *on-premise* presso l'INGV è composta da:

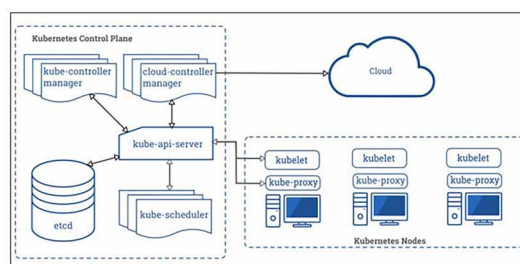
- N. 1 Server MAAS;
- N. 2 Server QEMU;
- N. 3 Nodi *Master*;
- N. 5 Nodi ETCD;
- N. 16 *Worker*;
- N. 1 SAN (*Storage Area Network*).

L'infrastruttura è stata progettata per garantire l'alta affidabilità del sistema e la resilienza ai guasti dei nodi.

La Figura 3 mostra la configurazione dell'architettura *on-premise* di *Kubernetes* presso l'INGV, con la distribuzione dei server MAAS, QEMU e i nodi *master* e *worker*.

Figura 3 Viene mostrata l'architettura generale di *Kubernetes*, evidenziando i componenti principali e il flusso di gestione dei *container* tra i nodi *master* e *worker*.

Figure 3 Shows the general architecture of *Kubernetes*, highlighting the main components and the container management flow between the master and worker nodes.



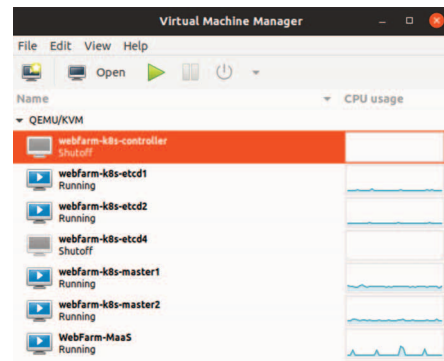
molte risorse e quindi per ottimizzare le risorse è stata sfruttata una sola macchina.

Per garantire un'alta affidabilità sono stati installati 2 server QEMU in modo tale che se uno non dovesse funzionare, l'altro si prenderà in carico l'esecuzione delle macchine presenti nell'altro. QEMU può essere gestito e monitorato sia mediante riga di comando sia mediante un'interfaccia grafica.

La Figura 5 mostra come QEMU viene utilizzato per gestire le macchine virtuali nel cluster Kubernetes.

Figura 5 Utilizzo di QEMU per la gestione delle macchine virtuali nel cluster Kubernetes on-premise, con ridondanza tra due server per garantire continuità operativa.

Figure 5 Using QEMU to manage virtual machines in the on-premise Kubernetes cluster, with redundancy between two servers to ensure operational continuity.



2.5 Nodo master

Il nodo *master* (chiamato anche *Control Plane*) è colui che gestisce i nodi *Worker*. Si preoccupa di mantenere il cluster in uno stato desiderato (cioè, verifica che il cluster rispetti i settaggi secondo dei file *manifest* che creiamo).

L'utente per gestire i *container* e altre risorse di Kubernetes, utilizzerà la macchina *master* (ad esempio per leggere i log di un *container*, vedere lo stato di un *container* o in generale lo stato del cluster).

I componenti più importanti che formano il *Control Plane* sono:

- *kube-apiserver*: è il *frontend* del *control plane*, accessibile sia tramite API REST che tramite linea di comando con *kubectl*.
- *kube-scheduler*: si occupa di capire se il cluster è in grado di ospitare altri *Pod*, monitorando le risorse dei nodi, e in caso positivo, decide su che nodo avviare il *Pod*.
- *kube-controller-manager*: si preoccupa di mantenere lo stato desiderato del cluster, come ad esempio verificare che sia rispettato il numero di repliche dei *Pod*. Per fare ciò, comunica con i nodi *slave* tramite il *kube-apiserver* che a sua volta utilizza un *database* distribuito, *etcd*.
- *etcd*: *database* distribuito di tipo *chiave-valore*, che contiene informazioni di configurazione del cluster come indirizzi IP dei vari nodi.
- *cloud-controller-manager*: aggiunge logica di controllo specifiche per *Cloud Provider*. Permette di collegare il cluster con le API del *cloud provider*. In un'installazione *on premise*, questo componente non è presente.

Questo nodo non richiede molte risorse *hardware* e per questo motivo è stato installato su una macchina virtuale ospitata sul server QEMU.

Allo scopo di tenerla in alta affidabilità ne sono state installate 3 e messe dietro un bilanciatore *hardware* che si occupa di smistare le richieste sui nodi attivi.

2.6 Nodo ETCD

ETCD è il *database* distribuito chiave-valore utilizzato da *Kubernetes* per memorizzare tutte le informazioni sullo stato del *cluster* [Larsson et al., 2020], comprese le configurazioni dei *Pod*, gli ingressi e le risorse. ETCD è fondamentale per garantire la consistenza e l'affidabilità del *cluster*. Il nome ETCD ha origine da due idee, la cartella *unix* “/etc” e i sistemi “d”*distributed*. La cartella “/etc” è un luogo in cui archiviare i dati di configurazione per un singolo sistema, mentre *etcd* archivia le informazioni di configurazione per sistemi distribuiti su larga scala.

ETCD è un componente chiave all'interno del *Cluster*. Infatti, *Kubernetes* archivia in esso i dati di configurazione dei servizi e la gestione dei *cluster*.

La coerenza di ETCD è fondamentale per pianificare e gestire correttamente i servizi. Il *server API Kubernetes* mantiene lo stato del *cluster* in *etcd*. Utilizza l'*API watch* di *etcd* per monitorare il *cluster* e implementare modifiche critiche alla configurazione.

ETCD è un *database* Chiave/Valore, che utilizza tre concetti chiave:

- capi (*leader*)
- elezioni (*elections*)
- periodi (*terms*)

Utilizza un algoritmo di consenso basato su *Raft*, dove il *cluster* elegge un *leader* per un determinato periodo di tempo. Questo gestisce tutte le richieste di archiviazione che ottengono il consenso del *cluster*.

Le richieste che non richiedono il consenso del *cluster*, come gli accessi in lettura, possono essere risposte indipendentemente da qualsiasi membro del *cluster*.

Se il *leader* accetta un cambiamento, *etcd* si assicura che l'informazione venga replicata ai nodi successivi. Una volta che questi ne hanno confermato la ricezione, il *leader* attua il cambiamento.

Se il *leader* cessa di funzionare o non risponde in modo prolungato ai comandi, trascorso un determinato intervallo di tempo i nodi rimanenti del *cluster* eleggono un nuovo *leader*. Il lasso di tempo che intercorre dal momento in cui un nodo chiede di eleggere un nuovo *leader* e si autodesigna come candidato varia da nodo a nodo ed è determinato per mezzo di un “cronometro” che ogni nodo possiede. Ciò fa sì che i nodi siano in grado di andare a sostituire il *leader* il più rapidamente possibile.

Affinché si possa sempre raggiungere una maggioranza, il numero di nodi nel *cluster* deve essere dispari. Per motivi di prestazioni, è raccomandabile non utilizzare un *cluster* che abbia più di sette nodi.

Nell'architettura *Kubernetes on-premise*, sono stati configurati cinque nodi ETCD per garantire alta disponibilità e ridondanza; infatti, come mostrato nella Figura 6, con cinque nodi c'è una tolleranza di *Failure* di 2 nodi. Il protocollo di consenso RAFT è utilizzato da ETCD per mantenere la coerenza dei dati tra i nodi del *cluster*. Anche se uno o più nodi ETCD falliscono, il sistema rimane operativo grazie alla replica delle informazioni tra i nodi rimanenti.

Cluster Size	Majority	Failure Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

Figura 6 Tabella di tolleranza ai guasti.

Figure 6 Fault tolerance table.

ETCD può essere installato con un'architettura interna (Figura 7) o esterna al *Cluster* (Figura 8), in questo progetto è stata implementata l'architettura esterna, inquanto consentiva maggiore flessibilità nella personalizzazione della configurazione e nelle procedure di *backup*.

Figura 7 Schema di una topologia in HA con l'utilizzo di un ETCD interno.

Figure 7 Diagram of a HA topology using an internal ETCD.

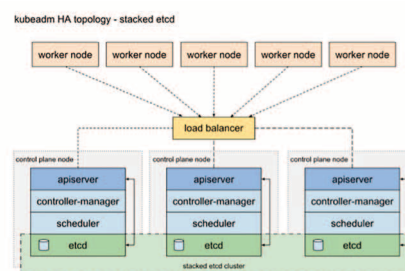
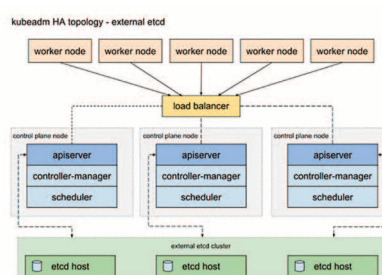


Figura 8 Schema di una topologia in HA con l'utilizzo di un ETCD esterno.

Figure 8 Diagram of a HA topology using an external ETCD.



2.7 Nodo worker

I nodi *worker*, detti anche nodi *slave*, eseguono le applicazioni containerizzate e ricevono i *workload* assegnati dal *master*. Ogni nodo *worker* esegue uno o più *Pod*, che sono le unità fondamentali di esecuzione di *Kubernetes*.

Ogni nodo *worker* contiene i seguenti componenti:

- *kube-proxy*: un *proxy* eseguito su ogni nodo è responsabile della gestione delle regole di rete del nodo per la comunicazione all'interno e all'esterno del *cluster*. Comunica costantemente col *control plane* attraverso il *kube-apiserver*, che gli comunica quali *Service* (un oggetto *Kubernetes* che vedremo più avanti) deve aggiungere o rimuovere.
- *container runtime*: è il componente che permette di eseguire i *container*, *Kubernetes* supporta diversi *container runtime* da qualche anno ha deprecato l'utilizzo di *Docker*. Come *container runtime* ho deciso di *containerd*.
- *kubelet*: *agent* che comunica col *master* e verifica che i *container* vengono eseguiti correttamente nel *Pod*. Esegue anche vari tipi di controlli sui *container*, come *readinessProbe* e *livenessProbe*.

Nell'architettura *Kubernetes on-premise* dell'INGV, sono stati configurati 16 nodi *worker* per un totale di risorse utilizzabili:

- 1760 CORE
- 20 Tera di RAM
- FIBER-CHANNEL 40 GB
- RETE A 40 GB

Ciascun nodo *worker* esegue un'istanza del servizio *Kubelet*, che si occupa di garantire che i *container* nei *Pod* siano in esecuzione come previsto. I *worker* possono essere ridimensionati dinamicamente in base al carico di lavoro grazie agli strumenti di *autoscaling* forniti da *Kubernetes*.

2.8 Storage Area Network (SAN)

La SAN (*Storage Area Network*) è una rete di storage centralizzata che fornisce uno *storage* condiviso per tutti i nodi *worker*. La SAN garantisce che i dati persistenti siano accessibili da qualsiasi nodo *worker*, permettendo alle applicazioni di mantenere lo stato tra diversi *container* e supportando strategie di *failover*.

Con una soluzione di *storage* condiviso come SAN, puoi garantire che il volume persistente sarà accessibile da qualsiasi nodo dell'infrastruttura, consentendo la mobilità del carico di lavoro nel cluster *Kubernetes*.

Nell'infrastruttura *Kubernetes on-premise*, la SAN è essenziale per gestire applicazioni *stateful* che richiedono l'accesso a dati persistenti, come i *database*. Inoltre, la SAN supporta il *backup* e il ripristino dei dati in caso di guasti o disastri.

Se i dati fossero stati allocati localmente ai nodi non sarebbe stato possibile effettuale uno *scaling* orizzontale, in quanto i dati non sarebbero stati accessibili dagli altri nodi e inoltre in caso di rottura del nodo si sarebbero persi i dati contenuti in esso.

Con l'avanzamento del *cloud computing*, le SAN sono state integrate in questo ambiente per sostenere una vasta gamma di carichi di lavoro basati sul *cloud*.

Ecco come vengono utilizzate le SAN nel *cloud computing*:

- Archiviazione di dati persistente: le SAN vengono spesso utilizzate nei servizi *cloud* per fornire archiviazione di dati persistente e affidabile. Questo è particolarmente importante per le applicazioni che richiedono dati persistenti tra riavvii o scalabilità orizzontale. Le SAN consentono di separare l'archiviazione dei dati dalla macchina virtuale (VM), rendendo i dati disponibili su più VM, se necessario.
- Supporto per *database*: le SAN sono spesso impiegate per supportare *database* nel *cloud*. I *database* richiedono prestazioni elevate e affidabilità e una SAN può offrire livelli di prestazioni e ridondanza necessari per gestire *database* critici per l'azienda.
- *Disaster Recovery* (DR) e *Backup*: le SAN sono utilizzate per implementare soluzioni di *disaster recovery* e *backup* nel *cloud*. I dati vengono replicati tra *data center* geograficamente separati tramite SAN, garantendo la continuità operativa in caso di guasto del *data center* primario.
- Condivisione dei dati tra VM: in molte configurazioni *cloud*, è necessario condividere dati tra diverse macchine virtuali. Le SAN consentono la condivisione di dati efficiente tra VM senza dover copiare fisicamente i dati da una VM all'altra.
- Supporto per carichi di lavoro ad alte prestazioni: applicazioni che richiedono prestazioni ad alte prestazioni, come il *rendering* 3D, il calcolo scientifico o le applicazioni finanziarie, possono beneficiare delle prestazioni offerte da una SAN. In questo caso, le SAN possono essere configurate per fornire prestazioni ottimali e latenza ridotta.
- Isolamento e controllo: le SAN offrono un maggiore isolamento e controllo sui dati rispetto all'archiviazione locale. Questo è importante in ambienti multilivello o con requisiti rigorosi di sicurezza e conformità.
- Elasticità e scalabilità: le SAN possono essere configurate per essere altamente scalabili ed elastiche, il che significa che è possibile aumentare o ridurre le risorse di archiviazione in base alle esigenze senza dover apportare modifiche significative all'infrastruttura. Tuttavia, con l'evoluzione del *cloud computing*, sono emerse nuove tecnologie di storage distribuito e servizi di archiviazione *cloud* gestiti che offrono opzioni di archiviazione altamente scalabili e gestite, spesso a costi più contenuti rispetto alle SAN tradizionali. Pertanto, la scelta di utilizzare una SAN dipenderà dalle specifiche esigenze dell'applicazione e dei carichi di lavoro, così come dalla strategia complessiva di archiviazione nel *cloud* dell'azienda.

3. L'algoritmo Replication Atomicity and Fault Tolerance

Nel mondo dei sistemi distribuiti, la progettazione di algoritmi di consenso è un aspetto fondamentale per garantire l'affidabilità e la coerenza dei dati tra più nodi.

Replication Atomicity and Fault Tolerance (RAFT) è uno degli algoritmi di consenso più innovativi e ampiamente adottati, sviluppato nel 2013 da Diego Ongaro e John Ousterhout [2014] che mira a semplificare il problema della ripartizione del lavoro tra nodi in un sistema distribuito. Il suo nome deriva dall'idea di "*rafting*" (*rafting*), poiché il processo di elezione di un *leader* è simile a una gara in cui i nodi cercano di emergere come *leader* del *cluster*.

3.1 Concetti chiave di Replication Atomicity and Fault Tolerance

L'obiettivo principale di RAFT è garantire che tutti i nodi in un *cluster* raggiungano un consenso sulla sequenza di operazioni da eseguire.

I componenti principali di RAFT sono:

- *Leader*: uno dei nodi nel *cluster* assume il ruolo di *leader* e coordina le operazioni di scrittura nel sistema. Il *leader* riceve tutte le richieste di aggiornamento e replica i cambiamenti sugli altri nodi.
- *Follower*: i nodi che non sono *leader* agiscono come *follower* e replicano i dati ricevuti dal *leader*. I *follower* non possono accettare nuove operazioni di scrittura, ma possono leggere lo stato replicato.
- *Candidato*: in caso di fallimento del *leader*, uno dei *follower* si propone come candidato per diventare il nuovo *leader*. Il processo di elezione del *leader* è gestito tramite votazioni tra i nodi del *cluster*.

In *Kubernetes*, ETCD utilizza RAFT per replicare i dati del *cluster* tra i nodi, garantendo che le modifiche allo stato del *cluster* (come l'aggiunta o la rimozione di *Pod*, configurazioni o ingressi) siano persistenti e coerenti su tutti i nodi *ETCD*.

3.2 Funzionamento del protocollo Replication Atomicity and Fault Tolerance

Il protocollo RAFT è suddiviso in tre fasi principali:

Prima fase. Elezione del *leader*

- All'inizio di ogni ciclo di consenso, uno dei nodi si candida per diventare *leader*.
- I *follower* votano per il candidato e, se il candidato riceve la maggioranza dei voti, diventa il *leader*.
- L'elezione di un nuovo *leader* avviene in caso di fallimento del *leader* corrente, o se il *leader* smette di rispondere.

Seconda fase. Replica del *log*

- Una volta eletto, il *leader* inizia a ricevere le richieste di scrittura dai *client* (ad esempio, aggiornamenti delle configurazioni o dei *Pod* nel *cluster*).
- Il *leader* registra ogni richiesta in un *log* locale e invia la richiesta agli altri nodi *follower* per replicare il *log*.
- Una volta che la maggioranza dei *follower* conferma la replica del *log*, il *leader* applica l'aggiornamento allo stato del *cluster* e risponde al *client*.

Terza fase. Persistenza e *failover*

- Se uno o più nodi del *cluster* falliscono o diventano inattivi, il *leader* continua a gestire le richieste e a replicare i dati sui nodi attivi.
- In caso di fallimento del *leader*, uno dei *follower* verrà eletto come nuovo *leader*, e il log delle transazioni sarà replicato su di esso per garantire la continuità delle operazioni.

Quando il *leader* riceve conferma dalla maggioranza dei *follower* che un'operazione è stata replicata con successo, considera l'operazione "*committed*" e invia una notifica a tutti i nodi del *cluster*: questo processo garantisce che tutte le scritture siano persistenti e coerenti su tutti i nodi.

Il meccanismo di *commit* di RAFT è essenziale per mantenere la coerenza dei dati in un sistema distribuito. Quando il *leader* conferma il *commit* di un'operazione, significa che l'operazione è stata applicata al suo stato locale e che gli altri nodi del *cluster* possono fare altrettanto. Una volta ricevuta la notifica, ogni nodo può applicare l'operazione al proprio stato locale, garantendo che tutte le operazioni siano applicate nello stesso ordine su tutti i nodi. Questo meccanismo assicura la coerenza globale dei dati.

Grazie alla sua semplicità e chiarezza concettuale, RAFT ha guadagnato popolarità come meccanismo affidabile per garantire la coerenza dei dati in ambienti distribuiti. La chiave del successo di RAFT risiede nella sua capacità di gestire in modo coerente ed affidabile l'elezione del *leader*, la replicazione dei dati e il meccanismo di *commit*. Questi elementi fondamentali lavorano insieme per fornire una base solida per la coerenza dei dati all'interno di un *cluster* distribuito.

Nel corso degli anni, RAFT è stato adottato da numerosi sistemi distribuiti ed è diventato uno *standard de facto* per il consenso distribuito. La sua robustezza e semplicità lo rendono uno strumento prezioso per gli ingegneri che si trovano ad affrontare sfide legate alla distribuzione e gestione dei dati in ambienti complessi.

4. Confronto tra sistemi di deployment

In questo capitolo, come rappresentato in Figura 9, andremo a confrontare e paragonare *deployment* eseguiti su sistemi tradizionali, ambienti virtualizzati e *container*. Ogni sistema presenta vantaggi e svantaggi specifici, che possono influenzare le prestazioni, la gestione delle risorse e la scalabilità.

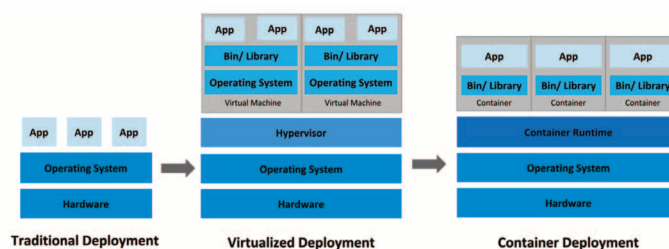


Figura 9 Rappresentazione di tre modalità di *deploy*.

Figure 9 Representation of three deployment modes.

4.1 Sistemi tradizionali su server fisici

All'inizio, le applicazioni venivano eseguite direttamente su *server* fisici, senza alcuna astrazione o isolamento tra loro. In questo approccio, le risorse di calcolo, come CPU, memoria e *storage*,

venivano condivise senza una chiara separazione tra le applicazioni, portando a problemi nella gestione delle risorse.

Quando più applicazioni venivano eseguite contemporaneamente su un singolo *server* fisico, una di esse poteva monopolizzare la maggior parte delle risorse, compromettendo le prestazioni delle altre. La soluzione di eseguire ciascuna applicazione su un *server* dedicato si è rivelata inefficiente in termini di utilizzo delle risorse e costosa, poiché richiedeva la gestione e il mantenimento di numerosi *server* fisici.

4.2 Virtualizzazione

Per risolvere i problemi legati alla gestione delle risorse sui *server* fisici, è stata introdotta la virtualizzazione, che consente l'esecuzione di più VM su un singolo *server* fisico. La virtualizzazione isola le applicazioni in ambienti separati, ciascuno con il proprio sistema operativo, garantendo un maggiore controllo sulle risorse e migliorando la sicurezza, poiché le informazioni di una VM non possono essere facilmente accessibili da un'altra.

Vantaggi della virtualizzazione:

- Miglior utilizzo delle risorse: le risorse *hardware* possono essere condivise tra più VM, riducendo i costi *hardware* e aumentando l'efficienza.
- Scalabilità: è possibile aggiungere nuove VM o aggiornare le risorse di quelle esistenti senza modificare l'infrastruttura fisica sottostante.
- Isolamento: ogni VM esegue un sistema operativo completo, garantendo un isolamento totale tra le applicazioni.

Tuttavia, la virtualizzazione introduce un certo *overhead*, poiché ogni VM deve eseguire un intero sistema operativo, aumentando il consumo di risorse rispetto ad approcci più leggeri, come i *container*.

4.3 Container

I *container* appresentano una forma di virtualizzazione più leggera rispetto alle VM, in cui il sistema operativo sottostante è condiviso tra le applicazioni, ma ciascun *container* mantiene un ambiente isolato per garantire la separazione delle risorse.

A differenza delle VM, i *container* condividono il *kernel* del sistema operativo, rendendoli più efficienti in termini di utilizzo delle risorse e tempi di avvio. I *container* isolano le applicazioni a livello di *file system*, CPU, memoria, PID e altre risorse, garantendo che ogni applicazione operi in modo indipendente.

L'uso dei *container* è diventato sempre più popolare grazie a una serie di vantaggi distintivi:

- Creazione e distribuzione agili di applicazioni: la creazione di immagini *container* è più efficiente rispetto all'uso di immagini VM, consentendo una distribuzione rapida e agevole delle applicazioni.
- Pratiche di sviluppo/test/rilascio continuativo: I *container* consentono la creazione e la distribuzione frequente di immagini affidabili, facilitando *rollback* rapidi e agevoli grazie all'immutabilità delle immagini stesse.
- Separazione tra Dev e Ops: le immagini *container* vengono create durante la fase di sviluppo dell'applicativo, anziché al momento del rilascio. Questo disaccoppia le applicazioni dall'infrastruttura sottostante.
- Osservabilità avanzata: i *container* non forniscono solo informazioni e metriche del sistema operativo ma anche segnali relativi allo stato di salute delle applicazioni, rendendo la

gestione e il monitoraggio più approfonditi.

- Coerenza tra ambienti: i *container* funzionano in modo uniforme sia su *computer* portatili che su ambienti *cloud*, garantendo la coerenza tra sviluppo, test e produzione.
- Portabilità tra ambienti eterogenei: La portabilità dei *container* è un vantaggio distintivo, in quanto possono essere eseguiti su vari sistemi operativi e ambienti *cloud*, garantendo la compatibilità e la flessibilità.
- Gestione centrata sulle applicazioni: i *container* alzano il livello di astrazione dall'infrastruttura sottostante, consentendo l'esecuzione delle applicazioni su un sistema operativo utilizzando risorse logiche.
- Architettura a microservizi: i *container* agevolano la suddivisione delle applicazioni in componenti più piccoli e indipendenti, favorendo la distribuzione dinamica e la gestione agilità. Non ci sono più *stack* monolitici che dipendono da una singola macchina.
- Isolamento delle risorse: grazie all'isolamento delle risorse, le prestazioni delle applicazioni containerizzate sono prevedibili e gestibili in modo ottimale.
- Efficienza nell'uso delle risorse: i *container* sono noti per l'efficienza e la densità nell'uso delle risorse, il che consente di ottenere il massimo dalle risorse *hardware* disponibili.

In sintesi, i *container* rappresentano una soluzione versatile e potente per lo sviluppo, il test e l'esecuzione di applicazioni, con vantaggi che spaziano dalla facilità di gestione all'efficienza delle risorse e dalla portabilità tra diversi ambienti. La loro adozione ha rivoluzionato il modo in cui le organizzazioni affrontano la distribuzione delle applicazioni e l'infrastruttura IT.

4.3.1 Scalabilità dei container

La scalabilità è un aspetto fondamentale nella gestione di *cluster Kubernetes* in quanto consente di adattare le risorse del sistema in base ai requisiti crescenti delle applicazioni e ai carichi di lavoro dinamici. In questo capitolo esamineremo i concetti fondamentali della scalabilità, soffermandoci sulle principali strategie di scalabilità orizzontale e scalabilità verticale, oltre alle best practice per garantire un'espansione efficace del *cluster*.

4.3.2 Scalabilità Orizzontale

La scalabilità orizzontale (o *scaling out*) implica l'aggiunta di nuove istanze di risorse, come Pod o nodi, per aumentare la capacità. L'Horizontal Pod Autoscaler (HPA) (Figura 10) è una delle funzionalità chiave di Kubernetes che permette di regolare automaticamente il numero di repliche dei Pod in un deployment o in un replicaset in base al carico di lavoro attuale. In altre parole, l'HPA consente di scalare orizzontalmente le applicazioni Kubernetes in modo dinamico per adattarsi alle fluttuazioni del carico di lavoro, garantendo così prestazioni affidabili ed efficienti.

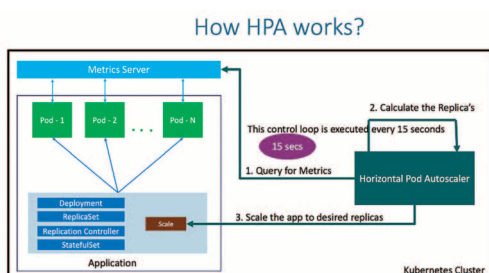


Figura 10 Funzionamento dell'HPA, che regola automaticamente il numero di Pod in base all'utilizzo delle risorse come CPU e memoria.

Figure 10 HPA works, which automatically adjusts the number of Pods based on resource usage such as CPU and memory.

Ecco alcuni concetti chiave relativi all'HPA:

- **Metriche di utilizzo delle risorse:** l'HPA basa le decisioni di scaling sulle metriche di utilizzo delle risorse, come la CPU e la memoria.
- **Target delle metriche:** quando si crea un HPA, bisogna specificare un valore *target* per le metriche di utilizzo delle risorse. Ad esempio, si può definire un obiettivo di utilizzo della CPU del 50 l'HPA cercherà di mantenere l'utilizzo della CPU di ciascun *Pod* vicino a questo obiettivo.
- **Scala automatica:** quando l'utilizzo delle risorse supera o scende al di sotto del *target* specificato, l'HPA avvia un processo di *scaling* automatico. Se l'utilizzo supera il *target*, l'HPA aumenterà il numero di repliche dei *Pod*. Se l'utilizzo scende al di sotto del *target*, diminuirà il numero di repliche.
- **Hysteresis:** per evitare oscillazioni indesiderate, l'HPA utilizza una "finestra" di tolleranza (*hysteresis*). Questo significa che non inizierà immediatamente a ridurre le repliche quando l'utilizzo scende appena al di sotto del *target*. È possibile configurare questa finestra per adattarla alle esigenze specifiche.
- **Metriche personalizzate:** oltre alle metriche di utilizzo delle risorse predefinite, è possibile definire metriche personalizzate per l'HPA. Ad esempio, potresti essere necessario scalare in base al numero di richieste HTTP al secondo o a metriche specifiche dell'applicazione.
- **Unità di misura:** le metriche di utilizzo delle risorse possono essere misurate in unità come *milliCPU* e *mebibyte* di memoria, consentendo una granularità fine nella definizione degli obiettivi di utilizzo.
- **Legame con i Deployment e i ReplicaSets:** l'HPA è solitamente associato a un *deployment* o a un *replicaset*. Quando l'HPA decide di aumentare o ridurre il numero di repliche, influisce direttamente sul numero di *Pod* gestiti da questi *controller*.
- **Monitoraggio e metriche:** per far funzionare correttamente l'HPA, è necessario disporre di un sistema di monitoraggio e raccogliere metriche sulle risorse e le prestazioni dell'applicazione. Strumenti come *Prometheus* sono spesso utilizzati per questo scopo.

In sintesi, l'HPA è uno strumento cruciale per garantire la scalabilità dinamica delle applicazioni *Kubernetes* in risposta alle fluttuazioni del carico di lavoro. Per configurare e utilizzare l'HPA in modo efficace, è necessario definire metriche di utilizzo delle risorse appropriate e comprendere come le impostazioni di *scaling* influiscono sul comportamento complessivo del cluster *Kubernetes*.

4.3.3 Scalabilità Verticale

La scalabilità verticale (o *scaling up*) comporta l'aumento delle risorse di una singola istanza, ad esempio l'aggiornamento delle dimensioni di una VM o di un *Pod*. *Kubernetes* offre funzionalità di *auto-scaling*, che consentono di regolare automaticamente il numero di *Pod* in base alle metriche di utilizzo delle risorse o a criteri specifici.

Kubernetes supporta la *Vertical Pod Autoscaler* (VPA), uno strumento che consente di regolare automaticamente le risorse allocate ai *Pod* in base al loro consumo effettivo di CPU e memoria. Questa modalità di *scaling* è particolarmente utile quando le applicazioni richiedono un aumento di risorse per gestire un carico di lavoro maggiore, ma senza la necessità di replicare ulteriori *Pod*. Vantaggi della scalabilità verticale:

- **Ottimizzazione delle risorse:** consente di migliorare l'efficienza delle applicazioni esistenti, aumentando le risorse per *Pod* che già gestiscono correttamente il carico di lavoro.
- **Facile gestione:** non richiede la creazione o il bilanciamento di nuovi *Pod*, ma si limita ad aggiornare quelli esistenti.

Svantaggi della scalabilità verticale:

- Limiti fisici: la quantità di risorse che possono essere assegnate a una singola istanza è limitata dall'*hardware* disponibile. Se una singola applicazione ha bisogno di più risorse di quante ne siano disponibili su un nodo, sarà comunque necessario ricorrere alla scalabilità orizzontale.
- Impatto sull'interruzione del servizio: l'aumento delle risorse in un *Pod* potrebbe richiedere un riavvio, causando potenzialmente un'interruzione temporanea.

5. Sicurezza informatica

La sicurezza informatica è un elemento fondamentale in un ambiente basato su *Kubernetes* per garantire la protezione dei dati, delle applicazioni e delle risorse, infatti, implementare pratiche di sicurezza adeguate aiuta a prevenire attacchi e garantire la continuità operativa del *cluster*.

Ecco alcune *best practice* che sono state implementate per garantire la sicurezza all'interno del *Cluster Kubernetes*:

- Accesso e autenticazione: è stato usato un sistema di autenticazione robusto come *Kubernetes RBAC (Role-Based Access Control)* per gestire l'accesso alle risorse del *cluster*. Configurare l'autenticazione a più fattori (MFA) per gli utenti e implementare una politica di accesso minimo privilegio.
- Isolamento dei servizi: sono stati usati *Namespaces* per isolare i servizi e i *team*. Ciò permette di separare applicazioni e risorse, riducendo il rischio di interferenze tra di esse.
- Gestione delle immagini dei *container*: Si sono utilizzate immagini dei *container* sicure, aggiornate e autenticate. Utilizzando *registries* di *container* sicuri e *scanner* per rilevare vulnerabilità nelle immagini.
- Politiche di rete: sono state configurate le politiche di rete utilizzando *Network Policies* per controllare il traffico tra i *Pod*, limitando l'accesso ai servizi a solo fonti autorizzate.
- Monitoraggio e registrazione: sono stati implementati sistemi di monitoraggio e *log* centralizzati come *Prometheus* e *Grafana* per tenere traccia delle attività e delle metriche del *cluster*.
- Sicurezza dei nodi: i nodi del *cluster* sono stati adeguatamente protetti utilizzando sistemi di protezione come *firewall*, *IDS/IPS* e *antivirus*.

La sicurezza in un ambiente basato su *Kubernetes* è un processo continuo che richiede la collaborazione tra *team* di sviluppo, amministratori di sistema e specialisti della sicurezza. Implementare un approccio di sicurezza olistico è essenziale per proteggere l'ambiente *cloud Kubernetes* da potenziali minacce.

6. Continuous Delivery mediante Argo CD

Argo CD è uno strumento utilizzato per implementare la *Continuous Delivery (CD)* automatizzata, garantendo che le applicazioni vengano distribuite in modo affidabile e sicuro negli ambienti *Kubernetes*.

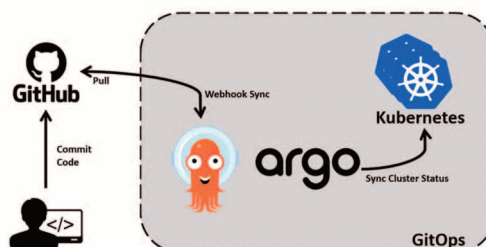
Alcuni concetti chiave associati alla CD includono:

- *Pipeline* di CI/CD: un flusso di lavoro automatizzato che coinvolge l'integrazione continua (CI) e la consegna continua (CD) per consegnare in modo affidabile le modifiche al *software* agli ambienti di produzione.
- *Deployment* automatizzato: l'automazione della distribuzione di nuove versioni di *software* in ambienti specifici senza intervento umano.

- *Rollback* automatizzato: la capacità di *Argo CD* di tornare rapidamente a una versione precedente in caso di problemi durante il rilascio.
- Revisioni delle applicazioni: un registro delle revisioni di ogni applicazione gestita da *Argo CD*, che consente il monitoraggio delle modifiche e la facilità di *rollback*.
- Politiche di sincronizzazione: regole configurabili per determinare quando e come sincronizzare lo stato delle applicazioni con le specifiche delle applicazioni.

Figura 11 Esempio del flusso di sincronizzazione tra il repository *Git* e il cluster *Kubernetes* tramite *Argo CD*.

Figure 11 Example of the synchronization flow between the *Git* repository and the *Kubernetes* cluster via *Argo CD*.

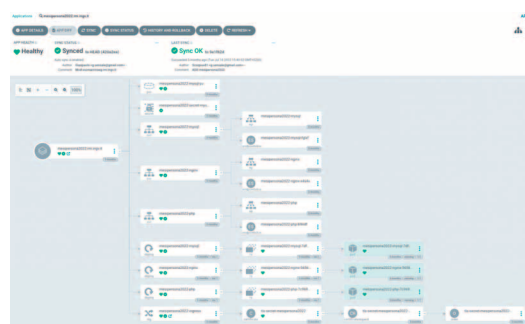


Per implementare la CD, è stato necessario configurare un repository *Git* per contenere le definizioni delle applicazioni *Kubernetes*. In Figura 11 è mostrato il flusso di sincronizzazione tra il repository *Git* e il cluster *Kubernetes*. Questo repository fungerà da “sorgente di verità” per *Argo CD*. Inoltre, si è reso necessario stabilire le politiche di sincronizzazione per le applicazioni *Argo CD*. Queste politiche determinano quando e come *Argo CD* deve sincronizzare le risorse *Kubernetes* con le definizioni delle applicazioni nel repository *Git*. Ad esempio, è possibile pianificare aggiornamenti periodici o sincronizzazioni in risposta a eventi specifici. *Argo CD* mantiene un registro delle revisioni di ogni applicazione.

Questo registro permette di monitorare tutte le modifiche apportate alle applicazioni nel tempo. In caso di problemi o errori durante il rilascio di una nuova versione di un’applicazione, *Argo CD* consente il *rollback* (Figura 12) automatico a una versione precedente stabile. Questa capacità di *rollback* automatico contribuisce a garantire la disponibilità continua delle applicazioni.

Figura 12 Schema che mostra il processo di *rollback* automatizzato con *Argo CD*, che garantisce la continuità delle applicazioni in caso di errori.

Figure 12 Diagram showing the automated *rollback* process with *Argo CD*, which ensures application continuity in case of errors.



7. Risultati Ottenuti

La progettazione e implementazione di un cluster *Kubernetes* rappresentano un investimento significativo in termini di tempo e risorse. Tuttavia, i benefici ottenuti confermano l’efficacia di questo approccio per la gestione delle applicazioni containerizzate.

Uno dei principali risultati osservati è stata l’elevata automazione delle operazioni legate all’implementazione, al rilascio e alla scalabilità delle applicazioni. Questo ha portato a una drastica riduzione delle attività manuali, riducendo al minimo gli errori umani che in precedenza

rappresentavano una delle maggiori cause di inefficienza. Di conseguenza, i tempi di *deployment* sono stati significativamente ridotti, consentendo di distribuire nuove versioni delle applicazioni in modo più rapido e affidabile.

L'utilizzo delle risorse del *cluster* è diventato molto più efficiente, grazie all'allocazione dinamica delle risorse in base alle esigenze reali delle applicazioni. Questo approccio ha eliminato gli sprechi, garantendo che le risorse fossero utilizzate solo quando necessario, e ha portato a una riduzione complessiva dei costi operativi. Le politiche di scalabilità automatica implementate, hanno permesso alle applicazioni di gestire picchi di traffico senza subire degrading nelle prestazioni, garantendo un'esperienza utente ottimale anche durante periodi di carico elevato. Finora, sono stati migrati circa 50 applicativi, suddivisi tra servizi interni all'ente e servizi rivolti al pubblico. Questo processo di migrazione ha confermato la flessibilità e l'affidabilità del cluster *Kubernetes* nel gestire una vasta gamma di applicazioni con requisiti diversi.

Un altro risultato significativo riguarda la resilienza del sistema. Sono state condotte prove di spegnimento simulato dei nodi per verificare la capacità del cluster di gestire guasti. Durante queste prove, si è osservato che i *Pod* ospitati sui nodi disattivati sono stati automaticamente trasferiti agli altri nodi funzionanti, grazie al meccanismo di *failover* integrato in *Kubernetes*. Questo ha confermato la capacità del *cluster* di continuare a operare senza interruzioni anche in caso di guasti ai nodi, garantendo così l'alta disponibilità delle applicazioni e una continuità operativa.

In sintesi, l'adozione di *Kubernetes* ha portato a:

- Automazione delle operazioni di *deployment* e scalabilità, con riduzione degli errori umani.
- Utilizzo efficiente delle risorse del cluster, riducendo sprechi e costi operativi.
- Miglioramento delle prestazioni delle applicazioni, capaci di gestire picchi di traffico senza degrado.
- Resilienza e *failover* automatizzato in caso di guasti dei nodi.
- Migrazione di circa 50 applicazioni, tra cui servizi interni e pubblici, senza impatti significativi sull'operatività.

Questi risultati confermano il successo dell'implementazione di *Kubernetes* come piattaforma centrale per la gestione delle applicazioni. containerizzate, offrendo vantaggi tangibili in termini di efficienza, affidabilità e riduzione dei costi.

7.1 Sviluppi futuri

Un obiettivo strategico per i futuri sviluppi è la federazione di due o più *cluster Kubernetes* distribuiti geograficamente. Questo approccio consentirebbe di migliorare ulteriormente la resilienza, la scalabilità e la disponibilità dei servizi, distribuendo le risorse in modo più efficiente tra diverse regioni o *data center*.

Una delle sfide principali nella federazione dei *cluster Kubernetes* è garantire un'interoperabilità fluida tra i *cluster*. Nel prossimo futuro, si prevede un maggiore sviluppo di *standard* e protocolli che semplificano l'integrazione e la comunicazione tra *cluster* distribuiti. Ad esempio, l'adozione di *standard* come il *Kubernetes Service Discovery* (KSD) potrebbe semplificare la scoperta e la comunicazione tra *cluster* federati, migliorando così l'orchestrazione tra diverse regioni geografiche.

La sicurezza rimane una priorità fondamentale nella federazione dei *cluster Kubernetes*; infatti, si potrebbero includere un'architettura più avanzata per gestire in modo centralizzato le politiche di sicurezza, come l'accesso, l'autorizzazione e la crittografia, su un insieme di *cluster* federati.

Le organizzazioni si orientano sempre più verso la distribuzione geografica dei *cluster Kubernetes*, non solo per aumentare la resilienza e migliorare la disponibilità dei servizi, ma anche per ridurre il rischio di perdita di dati e gestire meglio i picchi di carico regionali. In questo scenario i futuri

sviluppi potrebbero includere strumenti avanzati per la gestione della distribuzione geografica dei dati e delle applicazioni tra *cluster* federati introducendo meccanismi di replica dei dati tra regioni diverse, nonché la gestione avanzata del *failover* tra *data center*, garantendo la continuità operativa anche in caso di guasti regionali.

L'automazione, inoltre, rimane uno degli aspetti critici nella gestione di *cluster Kubernetes* federati richiedendo l'adozione di strumenti più sofisticati per l'automazione della federazione, comprese le operazioni di rilevamento e correzione automatica dei problemi tra i cluster federati.

Infine, la gestione delle risorse e il bilanciamento del carico tra *cluster* federati sono aspetti chiave per ottimizzare le prestazioni e l'efficienza del sistema. L'integrazione di strategie avanzate per la distribuzione delle risorse consentirà di ottenere un utilizzo ottimale dell'infrastruttura, migliorando le prestazioni complessive delle applicazioni.

In conclusione, la federazione di *cluster Kubernetes* rappresenta una direzione strategica nell'evoluzione delle infrastrutture *cloud-native* offrendo opportunità significative ma presentando al contempo nuove sfide. Gli sviluppi futuri in questo ambito richiederanno una continua ricerca e sperimentazione per massimizzare i benefici di questa tecnologia emergente e trasformarla in uno *standard* consolidato per la gestione delle infrastrutture distribuite.

8. Conclusioni

In questo articolo abbiamo esplorato in modo approfondito la progettazione, implementazione e gestione di un'infrastruttura *Kubernetes on-premise* motivando le scelte progettuali analizzando i vari vantaggi, sfruttando come caso di studio la migrazione di diversi siti *web/applicativi* da una infrastruttura *bast* su *webserver* su macchine virtuali a *container deployati* su un *cluster Kubernetes*, si è dimostrato come questa pratica possa portare a risultati significativamente positivi per le organizzazioni.

Durante il corso di questa ricerca, abbiamo identificato i molteplici benefici nell'adozione di *Kubernetes*, tra cui la scalabilità migliorata, il miglioramento delle prestazioni, la riduzione dei costi operativi e l'aumento della disponibilità.

L'automazione delle operazioni di *deployment* e *scalabilità* ha ridotto drasticamente l'intervento manuale, minimizzando al contempo il rischio di errori umani. La gestione dinamica delle risorse ha consentito un utilizzo ottimizzato del *cluster*, portando a una riduzione dei costi operativi e a un aumento della resilienza e delle prestazioni delle applicazioni, capaci di adattarsi in tempo reale alle fluttuazioni dei carichi di lavoro.

Kubernetes è una piattaforma potente e versatile che semplifica la gestione dei *container*, migliorando l'efficienza operativa, la resilienza, e la scalabilità delle applicazioni. Se hai bisogno di gestire ambienti distribuiti e complessi, *Kubernetes* è spesso una scelta ideale.

Le trasformazioni delle applicazioni in *microservizi* che girano su *container* rappresenta una pietra miliare nella modernizzazione delle infrastrutture IT.

Tuttavia, non dobbiamo dimenticare che la migrazione sul *cloud* non è un processo privo di sfide. La complessità della migrazione, la gestione dei costi e le questioni di sicurezza e conformità richiedono una pianificazione accurata e una gestione attenta. È fondamentale che le organizzazioni comprendano appieno le proprie esigenze e le dipendenze delle applicazioni prima di intraprendere il processo di migrazione.

Infine, il successo di questa trasformazione digitale è strettamente legato alla capacità delle organizzazioni di adattarsi a nuove tecnologie e di sviluppare competenze specifiche per la gestione dei sistemi distribuiti. La formazione continua del personale tecnico e l'adozione delle *best practice* emergenti nel settore *cloud-native* saranno elementi determinanti per mantenere competitività e garantire la piena realizzazione del potenziale offerto da *Kubernetes* e dalle architetture cloud moderne.

In conclusione, la migrazione dei siti *web* sul *cloud* con l'implementazione di un'infrastruttura *Kubernetes on-premise* rappresenta un passaggio fondamentale nel percorso di innovazione tecnologica, consentendo alle organizzazioni di beneficiare di un ambiente altamente scalabile, automatizzato e resiliente. Con una corretta pianificazione, gestione e adattamento alle sfide, le organizzazioni possono trarre vantaggio da un'infrastruttura *cloud* moderna e preparata per il futuro; ciò rappresenta una parte essenziale del percorso di innovazione e trasformazione digitale che le aziende stanno attraversando in questo secolo.

Bibliografia

- Carrión C., (2022). *Kubernetes as a Standard Container Orchestrator - A Bibliometric Analysis*. <https://doi.org/10.1007/s10723-022-09629-8>
- Larsson L., Tärneberg W., Klein C., Elmroth E., Kihl M., (2020). *Impact of etcd Deployment on Kubernetes. Istio, and Application Performance*. <https://arxiv.org/abs/2004.00372>
- Ongaro D., & Ousterhout J., (2014). *In Search of an Understandable Consensus Algorithm*. In *Proceedings of the 2014 USENIX Annual Technical Conference* (pp. 305-319). USENIX Association.

Sitografia

- [1] <https://www.ingv.it/it/>
- [2] <https://kubernetes.io/>
- [3] <https://www.dropbox.com>
- [4] <https://slack.com>
- [5] <https://www.garanteprivacy.it/>

Acronimi

- AKS: *Azure Kubernetes Service* - Servizio gestito *Kubernetes* di *Azure*
- API: *Application Programming Interface* - Interfaccia per interagire con *Kubernetes*
- ASG: *Auto Scaling Group* - Raggruppamento di istanze che si adatta automaticamente al carico
- CCM: *Cloud Controller Manager* - Integra *Kubernetes* con i servizi del *cloud provider*
- CNCF: *Cloud Native Computing Foundation* - L'organizzazione che mantiene *Kubernetes*
- CNI: *Container Network Interface* - Sistema per gestire il *networking* nei *cluster*
- CNI: *Container Network Interface* - Specifica per la gestione della rete dei *container*
- CRD: *Custom Resource Definition* - Definisce nuove risorse personalizzate in *Kubernetes*
- CRI: *Container Runtime Interface* - Interfaccia per il *runtime* dei *container*
- CSI: *Container Storage Interface* - Interfaccia *standard* per il collegamento dello *storage* nei *container*
- CSI: *Container Storage Interface* - Specifica per la gestione dello *storage* dei *container*
- DC: *Data Center* - Struttura che ospita i *server* e le risorse di calcolo
- DR: *Disaster Recovery* - Piano per il ripristino dei sistemi dopo un guasto
- EKS: *Elastic Kubernetes Service* - Servizio gestito *Kubernetes* di *AWS*
- FT: *Fault Tolerance* - Tolleranza ai guasti, capacità di un sistema di continuare a funzionare anche in caso di errori
- GKE: *Google Kubernetes Engine* - Servizio gestito *Kubernetes* di *Google Cloud*
- HA: *High Availability* - Alta disponibilità, garantisce continuità del servizio

HDFS: Hadoop Distributed File System - File system distribuito usato nei cluster Hadoop
HPA: Horizontal Pod Autoscaler - Scala automaticamente il numero di pod
HPC: High Performance Computing - Cluster ad alte prestazioni per calcoli complessi
IAM: Identity and Access Management - Sistema per la gestione delle identità e dei permessi
K8S: Abbreviazione di Kubernetes (K + 8 lettere + s)
KAPI: Kubernetes API - L'interfaccia principale per interagire con il cluster
KCM: Kube Controller Manager - Coordina i vari controller per mantenere lo stato del cluster
KCM: Kube Controller Manager - Coordina vari controller nel cluster
KCP: Kubernetes Control Plane - Gestisce lo stato e il ciclo di vita del cluster
KSA: Kubernetes Service Account - Account per l'autenticazione interna nei cluster Kubernetes
KSA: Kubernetes Service Account - Account usato per l'autenticazione all'interno del cluster
KSM: Kube State Metrics - Espone metriche sullo stato delle risorse Kubernetes
LB: Load Balancer - Bilanciamento del carico tra più nodi o server
MPI: Message Passing Interface - Standard per la comunicazione tra processi nei cluster
OIDC: OpenID Connect - Protocollo per l'autenticazione degli utenti
PDB: Pod Disruption Budget - Definisce il numero minimo di pod sempre attivi durante le manutenzioni
PV: Persistent Volume - Risorsa di storage all'interno del cluster
PV: PersistentVolume - Volume persistente disponibile nel cluster
PVC: Persistent Volume Claim - Richiesta di spazio di storage da parte di un pod
PVC: PersistentVolumeClaim - Richiesta di storage persistente
RBAC: Role-Based Access Control - Controllo degli accessi basato su ruoli
RBAC: Role-Based Access Control - Controllo degli accessi basato sui ruoli
TLS: Transport Layer Security - Protocollo per la sicurezza delle comunicazioni
TLS: Transport Layer Security - Protocollo per la sicurezza delle comunicazioni
VPA: Vertical Pod Autoscaler - Scala automaticamente le risorse (CPU/RAM) di un pod
VPC: Virtual Private Cloud - Rete privata virtuale all'interno del cloud
YARN: Yet Another Resource Negotiator - Sistema per la gestione delle risorse nei cluster Hadoop

QUADERNI di GEOFISICA

ISSN 1590-2595

<https://istituto.ingv.it/it/le-collane-editoriali-ingv/quaderni-di-geofisica.html/>

I QUADERNI DI GEOFISICA (QUAD. GEOFIS.) accolgono lavori, sia in italiano che in inglese, che diano particolare risalto alla pubblicazione di dati, misure, osservazioni e loro elaborazioni anche preliminari che necessitano di rapida diffusione nella comunità scientifica nazionale ed internazionale. Per questo scopo la pubblicazione on-line è particolarmente utile e fornisce accesso immediato a tutti i possibili utenti. Un Editorial Board multidisciplinare ed un accurato processo di peer-review garantiscono i requisiti di qualità per la pubblicazione dei contributi. I QUADERNI DI GEOFISICA sono presenti in "Emerging Sources Citation Index" di Clarivate Analytics, e in "Open Access Journals" di Scopus.

QUADERNI DI GEOFISICA (QUAD. GEOFIS.) welcome contributions, in Italian and/or in English, with special emphasis on preliminary elaborations of data, measures, and observations that need rapid and widespread diffusion in the scientific community. The on-line publication is particularly useful for this purpose, and a multidisciplinary Editorial Board with an accurate peer-review process provides the quality standard for the publication of the manuscripts. QUADERNI DI GEOFISICA are present in "Emerging Sources Citation Index" of Clarivate Analytics, and in "Open Access Journals" of Scopus.

RAPPORTI TECNICI INGV

ISSN 2039-7941

<https://istituto.ingv.it/it/le-collane-editoriali-ingv/rapporti-tecnici-ingv.html/>

I RAPPORTI TECNICI INGV (RAPP. TEC. INGV) pubblicano contributi, sia in italiano che in inglese, di tipo tecnologico come manuali, software, applicazioni ed innovazioni di strumentazioni, tecniche di raccolta dati di rilevante interesse tecnico-scientifico. I RAPPORTI TECNICI INGV sono pubblicati esclusivamente on-line per garantire agli autori rapidità di diffusione e agli utenti accesso immediato ai dati pubblicati. Un Editorial Board multidisciplinare ed un accurato processo di peer-review garantiscono i requisiti di qualità per la pubblicazione dei contributi.

RAPPORTI TECNICI INGV (RAPP. TEC. INGV) publish technological contributions (in Italian and/or in English) such as manuals, software, applications and implementations of instruments, and techniques of data collection. RAPPORTI TECNICI INGV are published online to guarantee celerity of diffusion and a prompt access to published data. A multidisciplinary Editorial Board and an accurate peer-review process provide the quality standard for the publication of the contributions.

MISCELLANEA INGV

ISSN 2039-6651

https://istituto.ingv.it/it/le-collane-editoriali-ingv/miscellanea-ingv.html

MISCELLANEA INGV (MISC. INGV) favorisce la pubblicazione di contributi scientifici riguardanti le attività svolte dall'INGV. In particolare, MISCELLANEA INGV raccoglie reports di progetti scientifici, proceedings di convegni, manuali, monografie di rilevante interesse, raccolte di articoli, ecc. La pubblicazione è esclusivamente on-line, completamente gratuita e garantisce tempi rapidi e grande diffusione sul web. L'Editorial Board INGV, grazie al suo carattere multidisciplinare, assicura i requisiti di qualità per la pubblicazione dei contributi sottomessi.

MISCELLANEA INGV (MISC. INGV) favours the publication of scientific contributions regarding the main activities carried out at INGV. In particular, MISCELLANEA INGV gathers reports of scientific projects, proceedings of meetings, manuals, relevant monographs, collections of articles etc. The journal is published online to guarantee celerity of diffusion on the internet. A multidisciplinary Editorial Board and an accurate peer-review process provide the quality standard for the publication of the contributions.

Coordinamento editoriale

Francesca DI STEFANO
Istituto Nazionale di Geofisica e Vulcanologia

Progetto grafico

Barbara ANGIONI
Istituto Nazionale di Geofisica e Vulcanologia

Impaginazione

Barbara ANGIONI
Patrizia PANTANI
Massimiliano CASCONI
Istituto Nazionale di Geofisica e Vulcanologia

©2025

Istituto Nazionale di Geofisica e Vulcanologia
Via di Vigna Murata, 605
00143 Roma
tel. +39 06518601

www.ingv.it



Creative Commons Attribution 4.0 International (CC BY 4.0)

