Alma Mater Studiorum · Università di Bologna

FACOLTÁ DI INGENGERIA
Corso di Laurea Magistrale in Ingegneria Informatica

GESTIONE DI UN CLUSTER HPC TRAMITE L'UTILIZZO DI MICROSERVIZI E KUBERNETES

Relatore:
Prof. Bartolini Andrea

Presentata da:
Mantovani Leonardo

Correlatore:
Ing. Galetti Daniela

Anno Accademico 2021/2022
Sessione V - Straordinaria

Abstract

Fino ad oggi la maggior parte dei sistemi HPC, distribuiscono e gestiscono i software tramite installazioni "bare-metal". Negli ultimi anni si stanno affermando tecnologie di containerizzazione che permettono di adottare un approccio a microservizi. Esistono diversi strumenti che permettono l'approccio a queste tipologie di architetture. Kubernetes é uno di questi.

Kubernetes é un software open-source per l'automazione del deployment, scalabilitá, e gestione di applicativi distribuiti tramite i container. Stanno diventando sempre piú popolari sistemi per la gestione di cluster HPC che lo implementano. Fornisce una piattaforma con caratteristiche che possono aiutare a semplificare l'implementazione e la gestione di carichi di lavoro di calcolo distribuito su larga scala come:

- Orchestrazione automatizzata
- Scalabilitá e ridondanza
- Gestione semplificata delle risorse
- Rolling updates e rollback
- Gestione delle configurazioni
- Monitoraggio e logging
- Vasto ecosistema di software e servizi

Fornendo una piattaforma unificata per la gestione delle risorse, l'automazione dei flussi di lavoro e l'ottimizzazione delle prestazioni, Kubernetes puó aiutare gli amministratori HPC a migliorare l'efficienza, ridurre i costi e migliorare le prestazioni complessive dei sistemi.

Con questa tesi é stata studiata e progettata una implementazione Kubernetes per un sistema di gestione di un cluster HPC. Se ne studieranno i vantaggi e le maggiori

criticitá a cui é possibile andare incontro durante le varie fasi operative: progettazione, installazione, configurazione e produzione.

Indice

Elenco delle figure

Introduzione

CINECA é un consorzio interuniversitario formato dai maggiori enti pubblici italiani é il piú grande centro di calcolo nello stato e uno dei maggiori ed importanti a livello mondiale. Offre supporto alla comunitá scientifica con soluzioni per il supercalcolo e le sue applicazioni.

L'azienda ospita e gestisce numerosi sistemi di calcolo ad altre prestazioni e ha da poco inaugurato il quarto supercomputer piú potente al mondo, LEONARDO [1].

Il campo del calcolo scientifico e di calcolo ad alte prestazioni (HPC) svolge un ruolo fondamentale nello sviluppo e nell'avanzamento di molteplici discipline scientifiche e ingegneristiche. L'HPC offre la possibilitá di svolgere calcoli complessi, modellare fenomeni naturali, analizzare grandi quantitá di dati e simulare sistemi complessi. L'evoluzione dell'HPC ha portato a risultati rivoluzionari in campi come l'astrofisica, la medicina, la chimica, l'ingegneria e molti altri settori.

Tradizionalmente, i cluster HPC sono stati gestiti utilizzando soluzioni monolitiche, che richiedono una configurazione complessa e sono meno adattabili alle mutevoli esigenze delle applicazioni. L'utilizzo di un'architettura a microservizi e container puó consentire una maggiore modularitá, facilitando la distribuzione e la gestione di applicazioni complesse in un ambiente HPC.

Negli ultimi anni, l'adozione di tecnologie come Kubernetes hanno rivoluzionato il modo in cui le applicazioni vengono gestite e distribuite su scala aziendale. Kubernetes é un sistema open-source di orchestrazione dei container che facilita la gestione, la distribuzione e la scalabilitá delle applicazioni containerizzate. É progettato per automatizzare molte delle attivitá complesse coinvolte nella gestione dei container, consentendo agli sviluppatori e agli operatori di concentrarsi sulle logiche di business senza dover preoccuparsi dei dettagli infrastrutturali.

La combinazione di un'architettura a microservizi, l'isolamento dei processi tramite container e la potenza di orchestrazione di Kubernetes offrono nuove opportunitá nella gestione di un cluster HPC, consentendo una maggiore flessibilitá, scalabilitá e efficienza

delle risorse.

L'obiettivo di questa tesi é quello di progettare un sistema di gestione per un cluster implementando i software necessari tramite l'utilizzo di Kubernetes. Verrá progettato e implementato in tutte le sue componenti, un cluster composto da dieci nodi. Verranno discusse le modalitá di progettazione della rete, della distribuzione e installazione dei nodi e dello stack software necessario allo scopo.

La tesi é cosí suddivisa:

Capitolo 1 si parlerá dei sistemi HPC e di come sono composti i loro sistemi di gestione. Si introdurranno i concetti di containerizzazione e Kubernetes, spiegando come questi possano portare benefici.

Capitolo 2 verrá introdotto in dettaglio Kubernetes

Capitolo 3 si espone in dettaglio la fase di progettazione e di realizzazione del cluster di gestione con Kubernetes

Capitolo 4 si parla delle caratteristiche finali del sistema e dell'overhead introdotto da Kubernetes. Si discuterá di alcune considerazioni riguardo all'utilizzo di questo approccio.

Capitolo 1

High Performance Computing - (HPC)

Obiettivi: *in questo capitolo verrá fatta una breve introduzione su cosa sono i cluster HPC e dei loro casi d'uso. Si parlerá poi dei sistemi di gestione, della loro composizione software e di come l'utilizzo di container e di Kubernetes possa portare benefici in questo tipo di sistemi*

- *High Performance Computing*
- *Architetture alta disponibilitá*
- *Gestione di un cluster HPC*
- *Hardware in un cluster HPC*
- *Stack software in un cluster HPC*
- *Container nell'HPC*
- *HPC e Kubernetes*

1.1 High Performance Computing

Con High Performance Computing (HPC) ci si riferisce generalmente all'elaborazione di calcoli complessi ad alta velocitá su piú server in parallelo. Questi gruppi di server sono noti come cluster e sono composti da centinaia o addirittura migliaia di server di elaborazione che sono stati collegati tramite una rete. Ciascun server viene anche chiamato nodo di calcolo, spesso abbreviato in nodo.

1.1.1 Perché l'HPC é importante?

É attraverso i dati e l'esecuzione di modelli matematici che vengono fatte scoperte scientifiche rivoluzionarie, vengono alimentate innovazioni e la qualitá della vita migliora per miliardi di persone in tutto il mondo. L'HPC é la base per i progressi scientifici, industriali e sociali.

Attualmente, i sistemi HPC sono ampiamente utilizzati per la ricerca scientifica, le operazioni militari, l'astrofisica, l'analisi dei big data, la finanza, la sicurezza informatica, il meteo, solo per citarne alcune delle sue applicazioni.

Con l'evolversi di tecnologie come l'Internet of Things (IoT), l'intelligenza artificiale (AI) e l'imaging 3D, le dimensioni e la quantitá di dati con cui le organizzazioni devono lavorare stanno crescendo in modo esponenziale. Per molti scopi, come lo streaming di un evento sportivo dal vivo, il monitoraggio di una tempesta in arrivo, il test di nuovi prodotti o l'analisi delle tendenze azionarie, la capacitá di elaborare i dati in tempo reale é fondamentale. Per stare un passo avanti rispetto alla concorrenza, le organizzazioni hanno bisogno di infrastrutture IT velocissime e altamente affidabili per elaborare, archiviare e analizzare enormi quantitá di dati.

Nel complesso, l'HPC é importante perché consente ai ricercatori di risolvere problemi complessi, fare progressi scientifici e sviluppare nuove tecnologie a vantaggio della societá. L'HPC é diventato uno strumento indispensabile in molte discipline scientifiche e ingegneristiche e la sua importanza é destinata a crescere man mano che i dati diventano sempre piú importanti in tutti gli aspetti della vita moderna [2].

1.1.2 Cos'é un cluster HPC

Un cluster di computer é un insieme di computer che lavorano come un unico sistema. Con il termine HPC (High Performance Computing) si intendendo i sistemi di calcolo, Cluster, per creare sistemi di elaborazione in grado, tipicamente tramite il calcolo parallelo, di raggiungere prestazioni molto superiori alla potenza raggiungibile da un singolo nodo.

Con con Cluster HPC si definisce quindi un tipo di sistema complesso e altamente performante, composto da una combinazione di hardware e software specializzati configurati per gestire una grandissima quantitá di dati con velocitá e alta disponibilitá di servizio. I cluster HPC sono anche conosciuti come supercomputer[3]. Il progetto TOP500 classifica e dettaglia i 500 sistemi informatici non distribuiti piú potenti al mondo.

I cluster HPC sono spesso costituiti da un gruppo di server; questi server sono generalmente indicati come nodi di calcolo.

Alcuni cluster possono essere piccoli composti da pochi nodi mentre altri invece possono averne centinaia o persino migliaia, tutti collegati attraverso una rete in modo che possano lavorare simultaneamente per risolvere carichi di lavoro avanzati.

Queste reti utilizzano interconnessioni ad alta velocitá e a bassi tempi di latenza. Queste caratteristiche sono necessarie affinché le comunicazione tra i vari processi durante le fasi di lavoro piú intenso, avvenga in modo immediato ed efficiente.

Un carico di lavoro HPC é un'attivitá altamente complessa e ad uso intensivo di dati che viene distribuita su risorse di calcolo, ciascuna delle quali esegue parti dell'attivitá in parallelo. I carichi di lavoro non sono altro che le esecuzioni degli applicativi da parte degli utenti. I carichi di lavoro hanno specifiche differenti e richiedono diversi livelli di CPU e memoria riservata per completare le attivitá, che dipendono dallo sforzo richiesto: la sua durata, gli intervalli e l'entitá. Al livello piú elementare, un carico di lavoro, o query, raccoglie l'input (I) e produce l'output (O).

I carichi di lavoro hanno spesso requisiti di archiviazione significativi, in termini di dimensioni dei dati o velocitá effettiva. Pertanto, é comune realizzare sia sistemi di archiviazione ad alte prestazioni, spesso indicati come scratch utilizzati per l'archiviazione di calcolo *"in-fly"*, sia una soluzione generica per i dati degli utenti, le applicazioni e l'archiviazione.

Per utilizzare tutte queste risorse nel modo piú efficace possibile e in parallelo, viene utilizzata un'interfaccia di programmazione basata sullo scambio di messaggi generalmente denominata MPI.

I carichi di lavoro nei cluster HPC generalmente vengono eseguiti in batch e sono gestiti da uno scheduler batch. Lo scheduler é una componente vitale dei cluster HPC, tiene traccia delle risorse disponibili di un cluster, mette in coda i carichi di lavoro quando non ci sono risorse computazionali sufficienti e quando le risorse computazionali diventano disponibili lo scheduler assegna in modo efficiente i carichi di lavoro alle risorse disponibili.

I cluster HPC sono quindi composti da:

- Un sistema di provisioning che garantisce l'omogeneitá dei nodi;
- Un'insieme di server;
- Uno scheduler che gestisce i carichi di lavoro in base alle risorse disponibili;
- La rete per la comunicazione tra i nodi;
- Uno storage generale per salvare i dati degli utenti;
- Un sistema di storage ad alte prestazioni per immagazzinare i dati computazionali;
- Gestione delle identitá per mantenere coerente l'accesso degli utenti in tutto il cluster;
- Un insieme di software per monitorare lo stato e le risorse del cluster.

1.2 Architetture ad alta disponibilitá

I cluster HPC sono progettati per elaborare grandi quantitá di dati ed eseguire attivitá complesse in un breve lasso di tempo. Questi sistemi sono spesso utilizzati in applicazioni critiche dove l'accuratezza e la velocitá dei risultati sono cruciali. In tali applicazioni critiche, i tempi di inattivitá del sistema possono avere conseguenze significative. L'indisponibilitá di un sistema di gestione HPC puó comportare una perdita di produttivitá, scadenze non rispettate e perdite finanziarie. Pertanto, é importante che un sistema di gestione HPC sia altamente disponibile per garantire che il sistema possa continuare a funzionare anche in caso di guasti hardware o software. Per questo é vengono progettati seguendo specifiche che ne garantiscano l'alta disponibilitá. Ció garantisce che il sistema di gestione HPC possa continuare a funzionare senza interruzioni, anche in caso di guasto di uno o piú componenti.

Un architettura si dice **High Available (HA)** quando in un sistema, tutti i componenti, moduli e servizi, lavorano insieme per mantenere prestazioni ottimali a scapito di picchi di carico. Piú semplicemente, queste architetture permettono di mantenere attivo un'ambiente di produzione senza interruzioni. Il livello di **High Availability** viene misurato come valore percentuale del tempo che i servizi rimangono attivi e funzionanti; un sistema puó essere considerato **HA** nel caso il valore percentuale non sia inferiore al **99.999%**

I cluster ad alta disponibilitá sono un gruppo di nodi che si uniscono in un unico sistema per evitare tempi di inattivitá. Se un server in un cluster ad alta disponibilitá si arresta, l'applicazione mission-critical viene immediatamente trasferita su un altro

server non appena viene rilevato l'errore. Nessun sistema é immune ai guasti e i cluster ad alta disponibilitá garantiscono il mantenimento di livelli di prestazioni ottimali indipendentemente dai guasti. Di conseguenza, questi tendono ad essere utilizzati per le applicazioni, i siti Web e i sistemi di elaborazione delle transazioni piú critici.

Un cluster ad alta disponibilitá utilizzerá piú sistemi che sono giá integrati, quindi se un errore causa il malfunzionamento di un sistema, un altro puó essere sfruttato in modo efficiente per mantenere la continuitá del servizio o dell'applicazione in uso. Il cluster di load-balancing ad alta disponibilitá svolge un ruolo cruciale nella prevenzione dei guasti del sistema. Avere un sistema di load-balancing essenzialmente distribuisce il traffico su diversi nodi Web che servono gli stessi utenti del sito Web o dell'applicazione. Ció riduce la pressione su qualsiasi server, consentendo a ciascun cluster di funzionare in modo ottimale e consentendo al traffico di essere inviato solo a server integri.

Ci sono diversi requisiti che questo tipo di architetture devono avere per massimizzare la durabilitá e l'alta disponibilitá.

Load Balancing

Il Load Balancing é fondamentale per qualsiasi architettura a disponibilitá elevata. La sua funzione principale é distribuire il traffico tra i server back-end per trasmettere i dati in modo piú efficiente e prevenire i sovraccarichi del server. Un prerequisito di qualsiasi sistema di load-balancing é identificare quale processo di failover deve essere eseguito in caso di errore del nodo.

Data Scalability

La capacitá di ridimensionare i database o le unitá di archiviazione su disco deve essere presa in considerazione da tutte le architetture a disponibilitá elevata. Esistono due soluzioni tra cui scegliere per ottenere la scalabilitá:

- utilizzare il database principale dell'architettura e utilizzare la replica o il partizionamento per renderlo altamente disponibile;
- garantire che le singole istanze dell'applicazione siano in grado di mantenere il proprio archivio di dati.

Backup e Ripristino

Nonostante tutta la sua coerenza, le architetture ad alta disponibilitá saranno sempre soggette a qualche tipo di malfunzionamento che puó interrompere il servizio. Pertanto,

in caso di interruzione di un servizio, le aziende devono disporre di una strategia di ripristino disponibile per ripristinare l'intero sistema il piú rapidamente possibile.

Questo é spesso indicato come **disaster recovery**: un insieme di politiche e procedure progettate per restituire un servizio alla piena funzionalitá in caso di un evento dirompente.

1.3 Gestione di un cluster HPC

La gestione di un cluster é un processo complesso che coinvolge diverse attivitá per garantire il corretto funzionamento, l'efficienza, la sicurezza e l'affidabilitá dell'infrastruttura distribuita, al fine di supportare le esigenze degli utenti.

Le attivitá di gestione di un cluster HPC possono includere la pianificazione e l'installazione del software e dell'hardware necessari, la configurazione delle risorse del sistema, l'amministrazione degli account utente, la gestione della sicurezza e della privacy dei dati, il monitoraggio delle prestazioni del sistema e la risoluzione dei problemi tecnici. Inoltre, si deve collaborare con gli utenti e gli stakeholder per garantire che il sistema risponda alle loro esigenze e alle loro aspettative.

1.3.1 Cluster bare-metal

Un cluster HPC gestito bare-metal é un sistema di calcolo ad alte prestazioni progettato per fornire agli utenti l'accesso ad un ambiente dedicato e completamente gestito, ottimizzato per i carichi di lavoro di calcolo scientifico. In un cluster HPC gestito bare-metal, le risorse hardware vengono fornite direttamente all'utente, senza la necessitá di un hypervisor o di un livello di virtualizzazione.

Uno dei principali vantaggi di un cluster HPC gestito bare-metal é che offre agli utenti un elevato livello di prestazioni e personalizzazione. Con risorse hardware dedicate, gli utenti possono essere sicuri di ottenere tutta la potenza del sistema, senza alcun sovraccarico di prestazioni dovuto alla virtualizzazione o ad altri livelli di astrazione.

1.4 Hardware in un cluster HPC

I vantaggi dell'HPC vengono dal riuscire ad utilizzare parallelamente molti nodi per eseguire un lavoro, oltre che alla continua disponibilitá dei servizi offerti. I soli nodi adibiti ad eseguire i workload di un cluster, peró non riuscirebbero a fare tutto se non ci fossero altri nodi che svolgono tutte quelle funzioni necessarie perché i requisiti vengano rispettati.

All'interno di un cluster HPC, possiamo trovare diversi tipi di nodi, ognuno con una funzione specifica. Di seguito, una panoramica dei principali tipi di nodi che si possono trovare in un cluster HPC.

1.4.1 Nodi di login

I nodi di login sono le interfaccia di accesso per gli utenti che devono utilizzare il cluster. É qui che gli utenti interagiscono con l'input e l'output dei loro carichi di lavoro e ottengono l'accesso ai sistemi di archiviazione locali disponibili per quel cluster. É anche il luogo in cui pianificano i loro carichi di lavoro. Lo scheduler, a sua volta, esegue i processi sui nodi di calcolo.

I nodi di login spesso fungono da ponte tra gli utenti e il resto del cluster. Di conseguenza, devono avere una configurazione adeguata per consentire agli utenti di accedere ai servizi del cluster e di utilizzare strumenti di sviluppo e di gestione del software.

1.4.2 Nodi master

Sono i nodi di controllo dei cluster. Su queste macchine sono presenti tutti i software e gli strumenti necessari a monitorare e raccogliere informazioni sullo stato del cluster, eseguono il software di gestione del cluster e coordinano il lavoro tra i nodi di calcolo e i nodi di archiviazione. Qui gli amministratori di sistema hanno tutti gli strumenti necessari ad intervenire in caso di guasti o effettuare operazioni di manutenzione su tutte le macchine del cluster senza dover accedere ad ognuna di esse.

1.4.3 Nodi computazionali

I nodi di calcolo o di elaborazione sono la componente di elaborazione di un cluster HPC. Sono responsabili dell'esecuzione del carico di lavoro utilizzando risorse locali, come CPU, GPU e altre unitá di elaborazione. Questi usano anche altre risorse sul nodo di calcolo per l'elaborazione, ad esempio la memoria, l'archiviazione e la scheda di rete. Ogni nodo computazionale é composto da una serie di componenti hardware, tra cui CPU, RAM, dispositivi di archiviazione e schede di rete ad alta velocitá.

I nodi accelerati sono nodi che incorporano acceleratori hardware, come ad esempio schede grafiche (GPU) o processori specializzati (ASIC), che consentono di eseguire operazioni specifiche in modo piú veloce rispetto ai processori tradizionali. Questi nodi sono spesso utilizzati per le applicazioni di machine learning, data analytics o simulazione fisica.

1.4.4 Storage

Lo storage in un cluster HPC é il piú delle volte basato su file con supporto POSIX. Esistono essenzialmente due tipologie di archiviazione, locale e distribuita.

I nodi di storage in un sistema HPC sono quei nodi dedicati alla gestione e all'immagazzinamento dei dati utilizzati dai job che vengono eseguiti sui nodi computazionali. Il loro compito principale é quello di fornire uno spazio di archiviazione condiviso e ad alta velocitá a tutto il sistema.

Lo storage locale in generale due principali utilizzi:

- l'archiviazione delle applicazioni e tutto ció che é necessario a farle funzionare. Questo perché é importante che tutti i file binari e le librerie siano coerenti in tutto il cluster durante l'esecuzione di un'applicazione, rendendo l'archiviazione centrale conveniente;

- per le home directory e i dati degli utenti.

Lo storage distribuito invece viene utilizzato per fornire l'accesso alle risorse archiviate da piú client contemporaneamente. Ció offre ai clienti l'accesso diretto ai dati archiviati, che, a sua volta, riduce i costi generali evitando l'astrazione, con conseguente bassa latenza e prestazioni elevate.

1.4.5 Network

I carichi di lavoro HPC dipendono fortemente dalla comunicazione tra processi. Quando quella comunicazione avviene all'interno di un nodo di calcolo, viene semplicemente passata da un processo all'altro attraverso la memoria di quel nodo di calcolo (shared memory). Ma quando un processo comunica con un processo su un altro nodo computazionale, quella comunicazione deve passare attraverso la rete. Questa comunicazione tra processi potrebbe essere abbastanza frequente. In tal caso, é importante che la rete abbia una bassa latenza per evitare ritardi di comunicazione tra i processi.

La rete di un sistema complesso come un cluster HPC viene gestita utilizzando diversi dispositivi.

Switch di rete : sono dispositivi di rete che consentono di connettere i vari nodi del cluster tra di loro.

Router : questi nodi di rete consentono di connettere il sistema HPC alla rete esterna, come Internet o altre reti aziendali.

Gateway : questi nodi di rete consentono di accedere alle risorse del sistema HPC da remoto, ad esempio tramite SSH o VPN.

1.5 Stack software cluster HPC

Uno dei punnti chiave per gestire un cluster HPC é disporre del giusto software di gestione HPC. Ció include metodi per distribuire i nodi di calcolo, mantenere aggiornati i sistemi operativi e altro software e monitorare l'hardware. Lo stack software é forse la parte piú importante per un sistema di calcolo HPC. A partire dalla scelta del sistema operativo, lo stack software determina non solo il funzionamento del sistema, ma anche le sue prestazioni.

La scelta dell'insieme dei software che andranno a definire il funzionamento del sistema non é univoco, ma viene scelto in base anche all'utilizzo che se ne dovrá fare. Ad esempio l'insieme dei software di un sistema che dorá fornire un servizio cloud sará diverso da uno adibito all'erogazione di servizi HPC.

1.5.1 Installazione e distribuzione dei nodi

La distribuzione e l'installazione di un cluster HPC possono essere compiti molto complessi e richiedono una buona conoscenza delle tecnologie di rete, del sistema operativo e del software di cluster.

Esistono diversi approcci possibili.La scelta dell'approccio dipende dalle esigenze specifiche del cluster e dalle risorse disponibili. In generale, si deve prestare attenzione a tutte le varie componenti che vanno a comporre il sistema finale.

xCAT

xCAT (eXtreme Cluster Administration Toolkit) [4] é un software open source che fornisce uno strumento di amministrazione e gestione automatizzata di cluster di computer. É stato progettato per semplificare il processo di distribuzione e installazione dei nodi in un sistema complesso, oltre a fornire strumenti per la gestione continua del cluster.

xCAT utilizza un approccio basato su immagini *(osimage)* per l'installazione del sistema operativo su tutti i nodi del cluster, utilizzando una varietá di protocolli di rete, come HTTP, FTP e TFTP. In questo modo, l'amministratore puó creare un'immagine personalizzata per il sistema operativo, con tutti i software, i driver e le configurazioni necessarie, e poi distribuirlo in modo uniforme su tutti i nodi del cluster.

Inoltre, xCAT include una vasta gamma di strumenti di gestione del cluster, tra cui la gestione del boot, la gestione dei nodi, la gestione dei servizi e la gestione della rete.

1.5.2 Sistema Operativo

Quando si tratta di High-Performance Computing (HPC), la scelta del sistema operativo é spesso guidata dai requisiti specifici dell'applicazione, nonché dall'ecosistema hardware e software su cui é costruito il sistema HPC. Detto questo, Linux é ampiamente considerato il sistema operativo piú popolare e affidabile per HPC. Questo perché Linux fornisce una piattaforma stabile, personalizzabile e open source che puó essere messa a punto per ottenere le massime prestazioni e buoni livelli di scalabilitá. Molti sistemi HPC utilizzano una versione di Linux ottimizzata per i carichi di lavoro HPC, come Red Hat Enterprise Linux, SUSE Linux Enterprise Server o CentOS. Queste distribuzioni in genere vengono fornite con strumenti e librerie specializzati progettati per sfruttare le caratteristiche uniche dell'hardware HPC, come l'elaborazione parallela e le interconnessioni ad alta velocitá. Altri sistemi operativi, come Microsoft Windows e macOS, vengono utilizzati anche negli ambienti HPC, ma sono generalmente meno comuni a causa del loro sovraccarico piú elevato e delle prestazioni inferiori sull'hardware HPC. Nel complesso, mentre altri sistemi operativi possono avere i propri punti di forza e vantaggi, Linux é generalmente considerato la scelta migliore grazie alla sua flessibilitá, prestazioni e forte supporto della comunitá.

1.5.3 Monitoraggio

I software di monitoraggio in un cluster HPC servono a tenere sotto controllo le prestazioni del sistema e identificare eventuali problemi che potrebbero causare un rallentamento o un malfunzionamento del cluster. Questi software raccolgono dati sulle prestazioni dei nodi del cluster, come l'utilizzo della CPU, la memoria utilizzata, la larghezza di banda di rete e altri parametri critici, e li presentano in modo chiaro e comprensibile per l'amministratore di sistema.

I software di monitoraggio consentono anche di identificare i problemi nel sistema e di intervenire rapidamente per risolverli. Ad esempio, se si rileva un elevato utilizzo della CPU su un nodo, l'amministratore di sistema puó intervenire per individuare l'applicazione che sta causando il problema e risolverlo prima che il sistema si blocchi. Inoltre, possono aiutare gli amministratori di sistema a ottimizzare le prestazioni del cluster, identificando i carichi di lavoro che richiedono piú risorse e distribuendoli in modo equilibrato sui nodi disponibili.

Per il monitoraggio di sistemi complessi esistono diversi software, di seguito ne verranno riportati alcuni

Nagios

Nagios [5] é un software di monitoraggio open source che puó essere utilizzato per il monitoraggio di un cluster HPC. É in grado di monitorare diversi aspetti del sistema, come ad esempio la disponibilitá dei servizi, lo stato dei nodi di calcolo e l'utilizzo delle risorse di sistema.

Il suo funzionamento si basa su una configurazione centralizzata in cui un server di Nagios monitora gli host e i servizi specificati nella sua configurazione. Quando un host o un servizio diventa inattivo o incontra problemi, Nagios invia notifiche agli amministratori di sistema tramite vari canali, come ad esempio e-mail o SMS.

Nagios supporta l'uso di plugin predefiniti ma permette anche la definizione di plugin personalizzati, che possono essere scritti in vari linguaggi di programmazione, per monitorare specifici aspetti del sistema.

Ganglia

Ganglia [6] é un software di monitoraggio open source progettato per il monitoraggio di cluster di computer di grandi dimensioni, tra cui i cluster HPC. é stato sviluppato dal Computational Research Laboratories (CRL) dell'Universitá del Tennessee e della University of California, Berkeley.

Il suo si basa sulla creazione di un demone che raccoglie le metriche di utilizzo delle risorse di sistema, come la CPU, la memoria, la rete e l'utilizzo del disco, da ogni nodo del cluster. Queste metriche vengono quindi trasmesse al server Ganglia, dove vengono elaborate e visualizzate in grafici e report.

Ganglia é progettato per essere altamente scalabile e distribuito, e puó essere utilizzato su cluster con migliaia di nodi. é in grado di visualizzare le informazioni sullo stato del cluster in tempo reale e puó inviare notifiche agli amministratori di sistema in caso di problemi.

Prometheus e Grafana

Prometheus e Grafana sono due software molto popolari utilizzati per il monitoraggio di cluster HPC. Di seguito ti fornisco una breve descrizione di entrambi:

Prometheus : é un sistema di monitoraggio open source che raccoglie e archivia le metriche di sistema [7]. é stato progettato per la scalabilitá, la facilitá d'uso e la

flessibilitá. Utilizzando un modello di dati basato su key-value, Prometheus raccoglie metriche dai target di monitoraggio, come ad esempio i nodi di calcolo del cluster, e le archivia in un database a series temporali. Grazie alla sua architettura, é in grado di gestire un gran numero di target di monitoraggio e di supportare la scoperta automatica dei target. Inoltre, Prometheus fornisce una ricca interfaccia utente web per la visualizzazione delle metriche, la definizione di allarmi e la creazione di grafici.

Grafana : é un software open source che consente di creare e condividere grafici interattivi e dashboard [8]. Grafana é in grado di integrarsi con numerosi data source, tra cui Prometheus, e consente di creare grafici personalizzati, visualizzazioni delle metriche in tempo reale e allarmi. Grafana supporta anche la definizione di pannelli di dashboard personalizzati, la creazione di query e l'integrazione con strumenti di allarme esterni.

Insieme, Prometheus e Grafana costituiscono un'ottima soluzione per il monitoraggio di un cluster HPC. Prometheus raccoglie le metriche dai nodi di calcolo del cluster e le archivia in un database, mentre Grafana consente di visualizzare le metriche in modo interattivo, creare dashboard personalizzate e definire allarmi in caso di problemi. In questo modo, gli amministratori di sistema possono monitorare le prestazioni del cluster e intervenire tempestivamente in caso di problemi.

1.5.4 Gestione della rete veloce

In un sistema HPC, l'interconnessione di rete é fondamentale per ottenere le prestazioni di calcolo e comunicazione necessarie per le applicazioni eseguite.

La gestione della rete veloce é fondamentale per un cluster HPC in quanto la comunicazione tra i nodi deve essere il piú rapida possibile per garantire prestazioni ottimali.

In particolare, l'InfiniBand é una tecnologia di rete ad alte prestazioni utilizzata in molti cluster HPC moderni. La gestione della rete InfiniBand é un aspetto critico nella configurazione e nell'ottimizzazione del sistema.

La gestione della rete veloce include la configurazione e il tuning della rete stessa, l'installazione dei driver di rete corretti sui nodi e l'implementazione di un protocollo di comunicazione adatto alle esigenze dell'applicazione in esecuzione sul cluster.

1.5.5 Storage e Filesystem

In un sistema HPC, lo storage é un elemento cruciale per l'efficienza e le prestazioni generali del sistema. I dati sono una componente essenziale delle applicazioni HPC e la capacitá di accedere rapidamente a grandi quantitá di dati é fondamentale per la scalabilitá delle prestazioni.

L'utilizzo di file system distribuiti e paralleli é fondamentale. Il parallelismo consente di suddividere il lavoro su piú nodi di elaborazione e, di conseguenza, di migliorare le prestazioni del sistema.

Un file system distribuito é un sistema di gestione dei dati che consente di memorizzare e accedere ai file distribuiti su piú nodi del cluster. In pratica, rende trasparente al programma applicativo il fatto che i file sono distribuiti su piú nodi, consentendo l'accesso come se fossero locali.

Un file system distribuito parallelo, invece, é una variante del file system distribuito che consente l'elaborazione parallela dei dati in modo distribuito su piú nodi del cluster.

CephFS [9] é un esempio di filesystem distribuito utilizzato nelle architetture di sistemi ad alte prestazioni. CephFS é basato su Ceph, un sistema di storage distribuito che utilizza la replicazione e la frammentazione dei dati per fornire alta disponibilitá e ridondanza. CephFS consente di distribuire i dati su un cluster di server, creando un filesystem unico e condiviso accessibile a tutti i nodi del cluster. Grazie alla sua architettura distribuita, offre prestazioni elevate e una scalabilitá orizzontale senza limiti. Inoltre, offre funzionalitá di snapshot e cloning dei dati, semplificando la gestione dei dati e la creazione di ambienti di test e sviluppo.

1.5.6 Scheduler

Gli scheduler sono strumenti fondamentali per la gestione delle risorse di un sistema HPC, in quanto consentono di pianificare e allocare in modo efficiente i processi di calcolo sui nodi disponibili.

Uno dei principali obiettivi degli scheduler é quello di massimizzare l'utilizzo delle risorse del sistema, minimizzando i tempi di attesa per i job e garantendo un equilibrato carico di lavoro sui nodi del cluster. Per fare ció, gli scheduler utilizzano algoritmi di scheduling complessi che considerano una serie di fattori, tra cui la prioritá del job, le risorse richieste, lo stato di utilizzo del nodo e le dipendenze tra i job.

Tra i principali scheduler utilizzati in un sistema HPC ci sono:

SLURM (Simple Linux Utility for Resource Management) : uno dei piú diffusi e utilizzati scheduler, in grado di gestire l'allocazione delle risorse in modo efficiente e flessibile; [10]

PBS (Portable Batch System) : un altro scheduler molto diffuso, in grado di gestire la pianificazione e l'allocazione delle risorse in modo efficiente e scalabile. [11]

1.5.7 Archiviazione e Backup

Nei sistemi HPC é fondamentale disporre di un sistema di gestione di backup e ripristino affidabile per garantire la disponibilitá e l'integritá dei dati critici utilizzati dalle applicazioni. In un ambiente HPC, la quantitá di dati generati e utilizzati puó essere molto elevata e pertanto il processo di backup e ripristino puó essere complesso e richiedere molte risorse.

Questi sistemi devono essere in grado di effettuare copie di backup dei dati in modo efficiente e affidabile, minimizzando l'impatto sulle prestazioni dell'applicazione. Inoltre, devono fornire una gestione flessibile delle politiche di backup e ripristino, consentendo di pianificare, eseguire e monitorare i processi di backup in modo centralizzato.

Tra i software di gestione di backup e ripristino utilizzati nei sistemi HPC, si puó citare IBM Tivoli Storage Manager (TSM). IBM Tivoli Storage Manager [12] é un software di backup e ripristino sviluppato da IBM per gestire grandi quantitá di dati in ambienti enterprise, tra cui sistemi HPC. TSM é stato progettato per funzionare su diverse piattaforme e sistemi operativi e supporta diversi metodi di backup, tra cui backup incrementale, differenziale e completo. Nel contesto di un sistema ad alte prestazioni, TSM puó essere utilizzato per eseguire backup e ripristino dei dati su cluster di grandi dimensioni, garantendo l'integritá e la sicurezza dei dati. TSM supporta anche la crittografia dei dati per garantire la privacy e la sicurezza delle informazioni salvate.

1.6 Container nell'HPC

I carichi di lavoro HPC sono in genere di natura monolitica. Tradizionalmente, le applicazioni HPC vengono eseguite per set di dati di grandi dimensioni nel data center. Il vantaggio principale di avere la containerizzazione delle applicazioni risiede nelle caratteristiche di portabilitá dei container. Possiamo impacchettare le applicazioni HPC ed eseguirle sui cluster per gestire un ampio set di dati.

In modo simile, i carichi di lavoro delle applicazioni HPC possono ottenere il vantaggio di una gestione isolata che include scalabilitá e sviluppo. Questa capacitá di

ridimensionamento dei container é importante, poiché i carichi di lavoro HPC possono trovarsi di fronte a una situazione in cui si verifica un picco nei requisiti di elaborazione dei dati e dovrebbero essere eseguiti senza tempi di inattivitá dei servizi. Le applicazioni HPC distribuite nei container possono scalare in modo indipendente per far fronte a tali picchi.

Inoltre, quando i container vengono utilizzati nell'architettura dei microservizi, vengono utilizzati per la velocitá, la scalabilitá e la modularitá. L'HPC coinvolge container con un ampio set di diverse librerie e componenti software insieme a dipendenze complesse per eseguire applicazioni di fascia alta. I container, in questo caso, aiutano i carichi di lavoro HPC a nascondere la complessitá e semplificare la distribuzione.

Inizialmente, i container sono stati designati come incompatibili per i flussi di lavoro HPC. Ora, con lo sviluppo di alcuni progetti open source, sono stati inventati nuovi modi per utilizzare i container per i carichi di lavoro HPC. Alcuni di questi progetti sono Singularity, Charliecloud, Shifter e Podman, tra gli altri.

Attualmente, i container sono principalmente orchestrati da Kubernetes, che puó eseguire i container in modo continuo. I carichi di lavoro HPC vengono eseguiti e completati per eseguire attivitá assegnate, ad esempio una simulazione per dati finanziari o flussi di lavoro genomici. L'esecuzione di carichi di lavoro HPC nei container avvantaggia i container stessi per eseguire regolarmente l'attivitá e richiama i container quando richiesto dai sistemi HPC.

1.7 HPC e Kubernetes

Kubernetes [13] é una piattaforma di orchestrazione di container open source progettata per automatizzare la distribuzione, il ridimensionamento e la gestione delle applicazioni containerizzate. Sebbene Kubernetes sia stato originariamente progettato per la gestione di applicazioni cloud-native, viene sempre piú utilizzato negli ambienti HPC per gestire carichi di lavoro di calcolo distribuito su larga scala. Ecco alcuni motivi per cui Kubernetes sta diventando popolare per la gestione dei sistemi HPC:

Scalabilitá : Uno dei principali vantaggi di Kubernetes é la sua scalabilitá. Kubernetes fornisce una piattaforma scalabile in grado di gestire un numero elevato di nodi di calcolo e container. Questo lo rende ideale per i carichi di lavoro HPC, che spesso richiedono la gestione di migliaia di nodi di calcolo.

Gestione delle risorse : Kubernetes fornisce sofisticate funzionalitá di gestione delle risorse che possono aiutare a ottimizzare l'uso delle risorse di calcolo. Puó allocare

dinamicamente le risorse in base ai requisiti del carico di lavoro, il che puó aiutare a migliorare le prestazioni e ridurre i costi.

Portabilitá : Kubernetes é progettato per essere indipendente dalla piattaforma, il che significa che puó essere utilizzato per gestire i carichi di lavoro su una varietá di piattaforme hardware, inclusi sistemi on-premise, cloud pubblici e ambienti cloud ibridi.

Automatizzazione : Kubernetes fornisce una potente piattaforma di automazione che puó aiutare a semplificare l'implementazione e la gestione dei carichi di lavoro. Ció puó aiutare a ridurre il carico di lavoro e migliorare l'efficienza dei flussi di lavoro. Si possono automatizzare le attivitá ripetitive, come l'implementazione di nuovi nodi o l'aggiornamento del software, liberando tempo per attivitá piú critiche.

Flessibilitá : Kubernetes é altamente personalizzabile e puó essere personalizzato per soddisfare i requisiti specifici di un ambiente HPC. Supporta un'ampia gamma di runtime dei container, che possono contribuire a migliorare le prestazioni e fornire una maggiore flessibilitá nella gestione dei carichi di lavoro. Con Kubernetes, é quindi possibile creare ambienti personalizzati ottimizzati esecuzioni specifiche, migliorando le prestazioni e riducendo i costi.

Kubernetes sta diventando sempre piú popolare per la gestione dei sistemi perché fornisce una piattaforma con caratteristiche che semplificano l'implementazione e la gestione dei sistemi di calcolo. Fornendo una piattaforma unificata per la gestione delle risorse, l'automazione dei flussi di lavoro e l'ottimizzazione delle prestazioni, Kubernetes puó aiutare gli amministratori di sistema a migliorare l'efficienza, ridurre i costi e migliorare le prestazioni complessive dei sistemi.

.

I cluster HPC sono sistemi complessi progettati per gestire grandi quantitá di dati e carichi computazioni intensivi. Questi sistemi devono grarantire efficienza, affidabilitá, disponibilitá e sicurezza. Perché queste caratteristiche vengano mantenute, servono sistemi di gestione. Fino a poco tempo fa la maggior parte di questi sistemi si sono basati su installazioni e distribuzioni bare-metal, dove tutto lo stack software necessario era distribuito attraverso installazioni dirette sulla macchina. Questo approccio richiedeva una grande quantitá di tempo e sforzo per configurare e gestire il cluster, rendendo difficile l'aggiornamento del software e l'implementazione di nuove funzionalitá.

Con l'avvento della virtualizzazione e dei container, é diventato possibile separare l'infrastruttura hardware dal software di sistema, consentendo una maggiore flessibilitá nella gestione dei cluster. I container consentono di isolare le applicazioni dal sistema operativo ospite, semplificando la gestione delle applicazioni e la distribuzione su diversi nodi del cluster.

Kubernetes, é un sistema di orchestrazione di container che automatizza la distribuzione, la scalabilitá e la gestione delle applicazioni su un cluster di server. Kubernetes offre una gestione centralizzata delle risorse del cluster e consente di distribuire facilmente applicazioni containerizzate su nodi del cluster. Questa nuova metodologia di gestione dei cluster offre molti vantaggi, tra cui la gestione centralizzata delle risorse, l'automatizzazione dei processi e la scalabilitá delle applicazioni.

All'interno del mio progetto di tesi ho esposto l'uso di Kubernetes per la progettazione di un'infrastruttura di gestione di un cluster HPC. Nel prossimo capitolo verrá presentato nel dettaglio Kubernetes.

Capitolo 2

Containerizzazione e Kubernetes

Obiettivi: *Partendo dalla definizione di container e dal concetto di architettura a microservizi, in questo capitolo verranno esposti concetti e definizioni riguardanti il mondo di Kubernetes.*

- *Containerizzazione*
- *Architettura a microservizi*
- *Kubernetes*

2.1 Containerizzazione

Con **containerizzazione** si intende il raggruppamento di codice software e di tutti i componenti necessari, come librerie, framework e altre dipendenze, in modo che risultino isolate in un proprio contenitore che chiamiamo **container**.

Questo metodo consente di essere spostare ed eseguire in modo uniforme il software o l'applicazione contenuta nel container in qualsiasi ambiente e infrastruttura, indipendentemente dal sistema operativo eseguito. Il container costituisce una sorta di bolla, un ambiente di elaborazione circostante all'applicazione, che la mantiene indipendente da ció che la circonda. Si tratta in sostanza di un ambiente di elaborazione portatile e completamente funzionale.

I container sono un'alternativa al coding in una piattaforma o sistema operativo quando il codice potrebbe non essere compatibile con il nuovo ambiente e lo spostamento dell'applicazione risulta quindi difficile, causando bug, errori e difficoltá che devono essere superate, ma che richiedono tempo, rallentano la produttivitá e aumentano la frustrazione.

Raggruppando tutti gli elementi di un'applicazione in un container che puó essere spostato tra piattaforme e infrastrutture, l'applicazione potrá essere utilizzata dove necessario. La caratteristica di leggerezza o portabilitá dei container deriva dalla capacitá di condividere il kernel del sistema operativo dell'host, evitando la necessitá di disporre di un sistema operativo distinto per ogni container e consentendo l'esecuzione dell'applicazione su qualsiasi infrastruttura (bare-metal, cloud) anche in macchine virtuali.

Inoltre, gli sviluppatori possono avvalersi degli stessi strumenti quando lavorano con i container in diversi ambienti host, il che semplifica nettamente lo sviluppo e il deployment di app containerizzate su diversi sistemi operativi.

I container sono spesso utilizzati per raggruppare singole funzioni che eseguono attivitá specifiche, noti come microservizi. Nei microservizi i componenti dell'applicazione sono suddivisi in servizi piú piccoli e specializzati, che consentono agli sviluppatori di lavorare su una specifica area dell'applicazione senza incidere sulle sue prestazioni complessive.

In questo modo l'app é sempre in esecuzione, anche durante gli aggiornamenti o la correzione dei problemi, accelerando le migliorie, i test e i deployment.

Caratteristiche quali portabilitá, compatibilitá e scalabilitá rendono microservizi e container simili e funzionali.

2.2 Architettura a microservizi

I microservizi sono sia un'architettura che un approccio alla scrittura di software. Con i microservizi, le applicazioni vengono scomposte nei loro elementi piú piccoli, indipendenti gli uni dagli altri ognuno dei quali comunica attraverso interfacce comuni come API e interfacce REST. Rispetto al tradizionale approccio monolitico, per cui ogni componente viene creato all'interno di un unico elemento, i microservizi interagiscono per completare le stesse attivitá, restando indipendenti gli uni dagli altri. Ciascun componente, o processo, costituisce un microservizio. Poiché particolarmente leggero, questo tipo di approccio allo sviluppo del software promuove la granularitá e consente di condividere processi simili tra piú app.

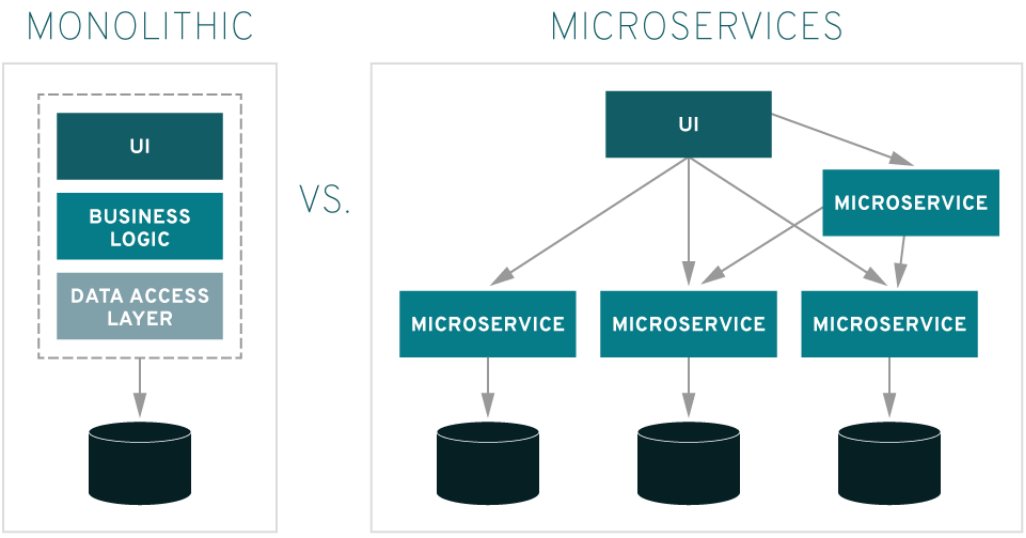

Figura 2.1: Architettura Monolitica vs Architettura a Microservizi

I microservizi rientrano nella categoria dei sistemi distribuiti. Per definizione, un sistema distribuito é una raccolta di programmi informatici che utilizzano risorse di elaborazione su piú nodi di calcolo separati per raggiungere un obiettivo comune e condiviso. I sistemi distribuiti contribuiscono a migliorare l'affidabilitá e le prestazioni dei sistemi e ad agevolare la scalabilitá.

I nodi di un sistema distribuito forniscono ridondanza, poiché se un nodo presenta errori, ci sono altri nodi pronti a sostituirlo. Ogni nodo dispone di scalabilitá orizzontale e verticale, il che si traduce in un miglioramento delle prestazioni. Se sul sistema é presente un carico elevato, é possibile ripartirlo aggiungendo altri nodi.

2.3 Kubernetes

Kubernetes o (**K8s**) é una piattaforma portatile, estensibile e open-source per la gestione di carichi di lavoro e servizi containerizzati, in grado di facilitare sia la configurazione dichiarativa che l'automazione. La piattaforma vanta un grande ecosistema in rapida crescita. Servizi, supporto e strumenti sono ampiamente disponibili nel mondo Kubernetes.

Con i container si riesce ad ottenere una soluzione per distribuire ed eseguire le applicazioni. In un ambiente di produzione, é necessario gestire i container che eseguono le applicazioni e garantire che non si verifichino interruzioni dei servizi. In questo caso con i container, in caso di fallimento, si dovrebbe provvedere tempestivamente ad avviarne uno nuovo. Kubernetes offre un framework per far funzionare sistemi distribuiti, occupandosi di scalabilitá, failover e distribuzione delle applicazioni.

2.3.1 Architettura

Un cluster Kubernetes é un insieme di macchine, chiamate nodi, che eseguono container gestiti da Kubernetes. Un cluster ha necessariamente un worker node. I worker node sono i nodi che ospitano i POD i quali eseguono le applicazioni (workload) degli utenti. I Control Plane sono i nodi che gestiscono i worker node e tutto ció che accade all'interno del cluster. Per garantire alta disponibilitá e la possibilitá di failover del cluster, dovrebbero essere utilizzati piú Control Plane. Questo non é obbligatorio, anche solo disponendo di un solo Control Plane il cluster sará operativo.

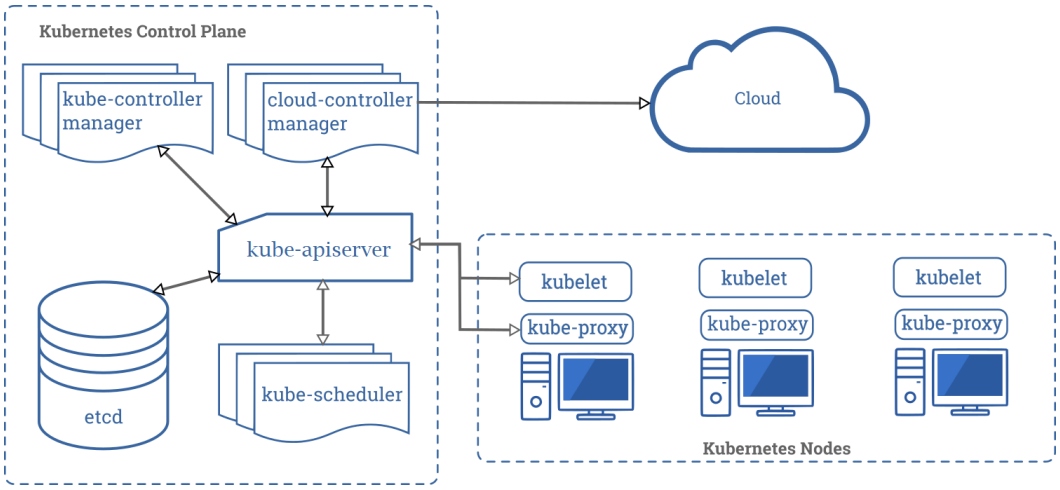

Figura 2.2: Diagramma componenti cluster Kubernetes

Control Plane - Master Nodes

Come detto i Control Plane sono i nodi responsabili della gestione e delle decisioni globali sul cluster. Si occupano ad esempio di eseguire lo scheduling e di reagire agli eventi rilevati nel cluster. I componenti del Control Plane possono essere eseguiti su qualsiasi nodo del cluster. Per semplicitá peró al momento dell'esecuzione degli script di installazione i componenti del Control Plane vengono fatti eseguire su una singola macchina, separando cosí le operazioni di gestione dai workload degli utenti.

kube-apiserver é la principale componente dei Control Plane. É una REST API che espone tutte le funzionalitá offerte da Kubernetes. Tutte le comunicazioni e le operazioni tra i componenti del Control Plane e i client esterni, come kubectl, vengono tradotte in chiamate API RESTful gestite dal server API. Tutte le richieste che

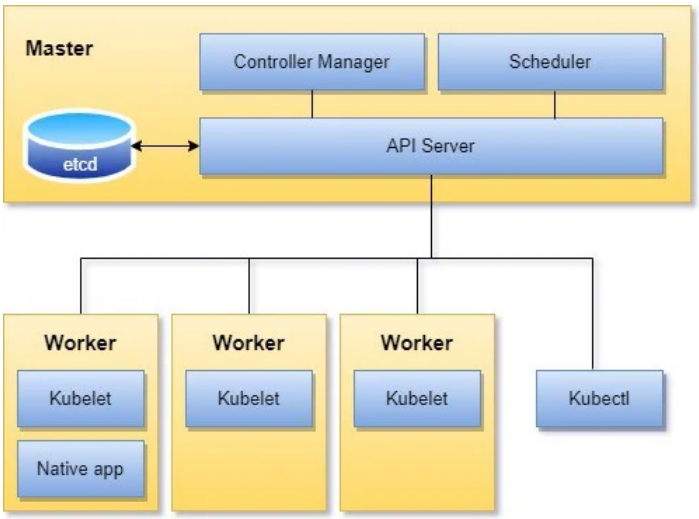

Figura 2.3: Funzionamento *kube-apiserver*

riceve vengono autenticate e autorizzate. Dopo che la richiesta é stata autenticata e autorizzata, passa ai moduli di controllo di ammissione. Questi moduli possono modificare o rifiutare le richieste. Se la richiesta sta solo tentando di eseguire un'operazione READ, ignora questa fase; ma se sta tentando di creare, modificare o eliminare, verrá inviata al plug-in del controller che ne deciderá l'ammissibilitá. Al termine delle modifiche, le nuove configurazioni, verranno salvate all'interno del database **etcd**.

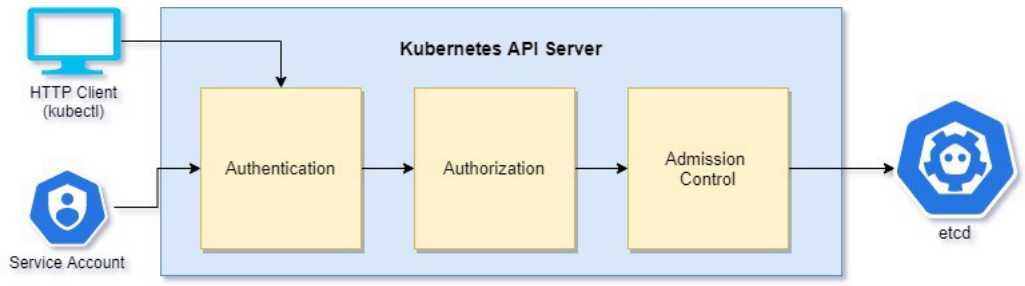

Figura 2.4: Flusso di richiesta HTTP Kubernetes

etcd é un database basato su **key-value** che funge da *"memoria"*. Viene utilizzato da un cluster Kubernetes per salvare tutte le informazioni. L'installazione della componente etcd puó essere locale sul Control Plane oppure essere installata come componente separata su un'altro nodo e acceduta via rete. É l'unica componente statefull e mantiene persistenti tutte le configurazioni e gli stati del cluster.

kube-scheduler questa componente é adibita allo scheduling dei POD appena creati e che non hanno un nodo assegnato. I fattori presi in considerazione per individuare un nodo a cui assegnare l'esecuzione di un POD si basano innanzitutto valutando la quantitá di risorse che il POD richiede, dal livello di carico presente sui nodi, da eventuali vincoli hardware/software, interferenze tra i diversi tipi di workload e alla disponibilitá dello storage.

kube-controller-manager In Kubernetes, i controller sono cicli di controllo che monitorano continuamente lo stato di un insieme di risorse e apportano modifiche per garantire che lo stato attuale corrisponda allo stato desiderato.

Il kube-controller-manager é la componente che gestisce i controller presenti nel Control Plane. Ogni controller é un processo separato ma per ridurre la complessitá questi vengono raggruppati in un unico container ed eseguiti in un singolo processo. Ogni controller di Kubernetes svolge un singolo compito, ad esempio, il controllore dei nodi del cluster, ha il compito di monitorare lo stato dei nodi nel cluster e prendere decisioni quando uno di questi non é piú disponibile; il replication controller invece ha il compito di mantenere costante il corretto numero di POD per ogni ReplicaSet del sistema. Ogni controller tipicamente esegue in background e osserva periodicamente la **kube-apiserver** in attesa di cambiamenti; seguono questo comportamento in modo da verificare che lo stato corrente del cluster sia quello desiderato.

In sintesi un Control Plane lavora come cervello del cluster, dove si prendono decisione e si fanno controlli sullo stato del sistema. Tutte queste funzionalitá lavorano come un singolo componente, ma in relatá si tratta di un insieme di piccoli servizi specializzati.

Nodi

I Nodi o Worker Nodes, sono i nodi che eseguono effettivamente il lavoro e dove vengono eseguite i container degli utenti. Svolgono essenzialmente tre attivitá:

- Osservano l'API server per nuovo lavoro;
- Eseguono il lavoro assegnato;
- Riportano tutti i risultati al Control Plane.

kubelet ogni nodo del cluster esegue un istanza di questa componente. Ha il ruolo di agente e si occupata di tutte le comunicazioni con il Control Plane, piú precisamente con il **kube-apiserver**. La **kubelet** lavora in termini di **PodSpec** ovvero dei file di configurazione che descrivono il contenuto e il funzionamento dei POD. Con l'utilizzo di questi file la **kubelet** si assicura che i container indicati siano presenti e che lavorino correttamente. Altra funzione di questa componente é registrare il nodo come partecipante del cluster.

kube-proxy é un proxy eseguito su ogni nodo ed é responsabile della gestione dei **Kubernetes Service**. Mantiene le regole di networking sui nodi, le quali permettono la comunicazione verso gli altri nodi del cluster o verso l'esterno. **kube-proxy** utilizza librerie del sistema operativo quando possibile, in caso contrario gestisce il traffico autonomamente. Ad esempio assicura ad ogni nodo di avere il proprio indirizzo IP e implementa regole di firewall locali o regole di routing per gestire il load-balancing di vari tipi di traffico.

container runtime é una componente fondamentale, in quanto é responsabile dell'esecuzione e della gestione dei container sui nodi del cluster. Il Container Runtime é al centro di ogni ambiente Kubernetes. Fondamentalmente é il componente dell'architettura che organizza le risorse hardware, esegue e arresta i container e si assicura che i container ricevano le risorse di cui hanno bisogno per funzionare in modo ottimale.

Quando Kubernetes é stato rilasciato per la prima volta, utilizzava il runtime Docker come impostazione predefinita hardcoded. Con la crescita della piattaforma,

é cresciuta anche la necessitá di supportare runtime alternativi. Per rendere Kubernetes piú indipendente dal runtime, é stata creata la CRI (Container Runtime Interface).

Kubernetes supporta qualsiasi runtime di container conforme a CRI. Quando si configura un cluster Kubernetes, si deve installare un runtime del container in ogni nodo del cluster in modo che i POD possano essere eseguiti su quel nodo.

2.3.2 Kubernetes Networking

Le applicazioni in un cluster Kubernetes raramente lavorano in solitudine. In un'architettura a microservizi, l'insieme di applicazioni che eseguono nei rispettivi Pod ha la necessitá di lavorare insieme. Kubernetes dá la possibilitá di creare connessioni *POD-to-Service* e connessioni che permettono l'accesso ai servizi dall'esterno.

Kubernetes é progettato come un sistema operativo per gestire la complessitá di dati e risorse distribuite. Il carico di lavoro puó essere distribuito sull'insieme dei nodi che compongono il cluster. La struttura della rete di Kubernetes é progettata per garantire diversi requisiti:

- **Comunicazione Container-to-Container**;
- **Comunicazione POD-to-POD**;
- **Comunicazione POD-to-Service**;
- **Comunicazione Node-to-Node**.

Ad ogni POD in un cluster viene assegnato il proprio indirizzo IP univoco a livello di cluster. Ció significa che non é necessario creare in modo esplicito collegamenti tra i POD e non é quasi mai necessario occuparsi della mappatura delle porte del container alle porte dell'host.

Kubernetes impone i seguenti requisiti fondamentali su qualsiasi implementazione di rete (escludendo qualsiasi policy di segmentazione di rete):

- i POD possono comunicare con tutti gli altri POD su qualsiasi altro nodo senza agenti NAT;
- su un nodo (ad es. demoni di sistema, kubelet) possono comunicare con tutti i POD eseguiti in quel nodo.

Questo modello non é solo meno complesso, ma é compatibile con il l'idea di consentire, in modo semplice e senza la creazione di criticitá, il porting delle applicazioni

dalle macchine virtuali ai container. Ogni macchina virtuale ha il proprio indirizzo IP e puó comunicare con le altre macchine virtuali presenti nel sistema. In generale i POD seguono lo stesso modello di funzionamento.

Gli indirizzi IP di Kubernetes esistono nell'ambito del POD: i container all'interno di un POD condividono i propri spazi dei nomi di rete, inclusi l'indirizzo IP e l'indirizzo MAC. Ció significa che tutti i container all'interno di un POD possono raggiungere le rispettive porte su localhost. Ció significa anche che i container all'interno di un POD devono coordinare l'utilizzo delle porte, ma questo non é diverso dai processi in una macchina virtuale.

CNI - Container Network Interface

CNI (Container Network Interface), consiste in una specifica ed in un'insieme di librerie per la scrittura di plug-in per configurare le interfacce di rete nei container Linux, insieme a una serie di plug-in supportati. CNI si occupa solo della connettivitá di rete dei container e della rimozione delle risorse allocate quando il container viene eliminato.(14)

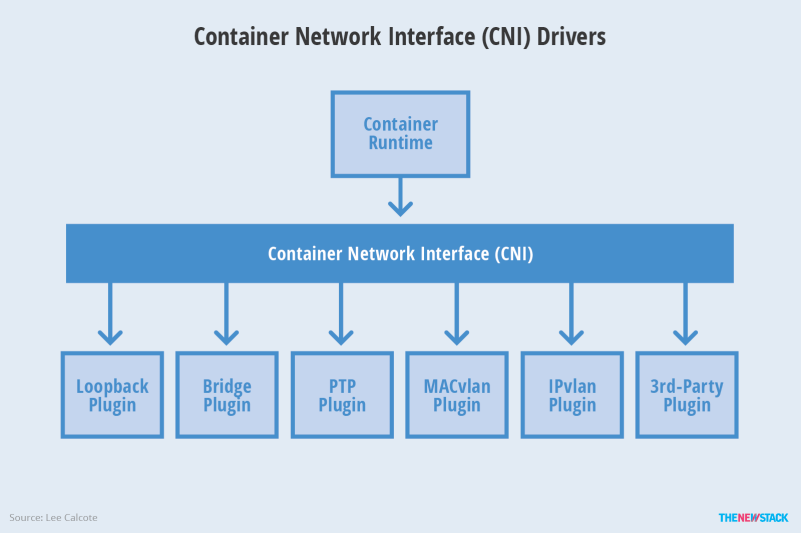

Figura 2.5: CNI - Container Network Interface

Nel contesto Kubernetes, il CNI si integra perfettamente con la componente **kubelet** per consentire la configurazione automatica della rete tra i POD utilizzando una rete underlay o overlay. Una rete underlay é definita a livello fisico del livello di rete composto

da router e switch. Al contrario, la rete overlay utilizza un'interfaccia virtuale come VXLAN per incapsulare il traffico di rete.

Una volta specificato il tipo di configurazione di rete, il runtime definisce una rete per i container da unire e chiama il plugin CNI per aggiungere l'interfaccia nello spazio dei nomi del container e allocare la sottorete collegata e le route effettuando chiamate al plugin IPAM (Gestione degli indirizzi IP).

2.3.3 *kubectl* e sintassi YAML

kubectl é lo strumento a riga di comando usato da Kubernetes. É un client HTTP, ottimizzato per interagire con Kubernetes e consente di inviare comandi al cluster Kubernetes. Dal momento in cui le *kube-apiserver* non sono altro che delle API HTTP, ogni client HTTP sarebbe in grado di interagire con esse e quindi con il cluster Kubernetes. Se prendessimo alla lettera quanto detto, sarebbe quindi possibile interagire con il cluster utilizzando ad esempio CURL. Questo oltre che aumentare la difficoltá delle operazioni non sarebbe nemmeno del tutto corretto. **kubectl** infatti non gestisce solo la parte che riguarda la connessione con le API ma anche altri aspetti legati all'autenticazione con il livello di autenticazione di Kubernetes, gestisce i contesti e altro.

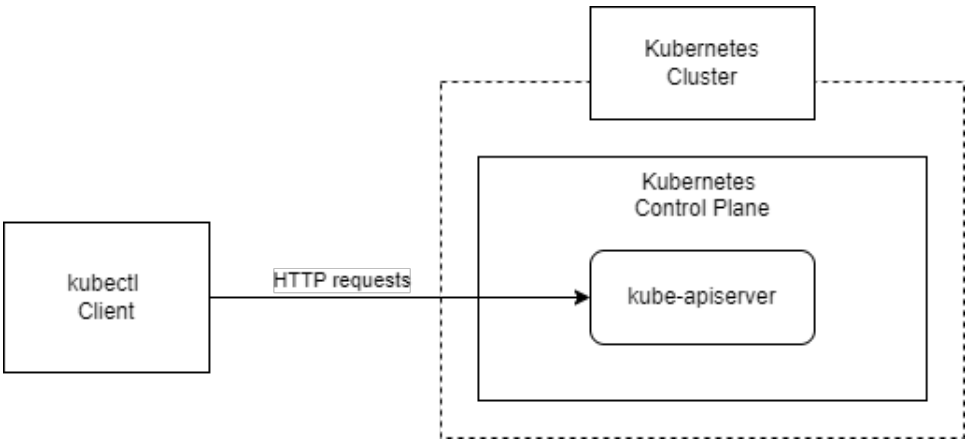

Figura 2.6: Interazione ***kubectl*** con le API Kubernetes

Questo in generale é il workflow con Kubernetes: si interagisce con il cluster utilizzando *kubectl*. Quando le API ricevono una richiesta HTTP valida, si va a leggere o ad aggiornare lo stato nel database **etcd**.

kubectl supporta due tipi di sintassi:

- La sintassi imperativa

- La sintassi dichiarativa

Sintassi imperativa

La forma imperativa esegue utilizzando i comandi kubectl tramite linea di comando. Passando i corretti sotto-comandi e argomenti, si possono costruire istruzioni per le API.

```
#crea un POD chiamato my-POD basato sull'immagine busybox:latest
 $ kubectl run my-POD --image busybox:latest
#si stampano a video i servizi presenti nel namespace myns
 $ kubectl get services -n myns
#elimina il POD my-POD
 $ kubectl delete pods my-POD
```

La sintassi imperativa ha molteplici benefici. Se si conoscono i comandi da mandare a Kubernetes per effetuare una modifica, utilizzando questa sintassi si é molto piú veloci. Oppure, per richiedere informazioni sullo stato del cluster, la sintassi imperativa é l'unico metodo possibile, con la sintassi dichiarativa non si é in grado di effettuare questo tipo di richieste.

Presenta peró un grande problema. Se si utilizza questa sintassi é molto complicato tenere traccia di tutto cio che si é fatto precedentemente nel cluster. Se ad esempio capita che si debba ripristinare il cluster da zero e si sono utilizzati solo comandi imperativi, é impossibile recuperare informazioni che descrivano le precedenti configurazioni.

Sintassi dichiarativa

La sintassi dichiarativa viene applicata scrivendo file YAML per poi applicare la configurazione tramite l'utilizzo di kubectl. In realtá anche il formato JSON é supportato, ma per semplicitá si utilizza YAML. Questo formato non é altro che un insieme di coppie *key:value* e rappresentano i dati della configurazione delle risorse che si vogliono creare.

```
#esempio file configurazione per la creazione di un POD
```

```
apiVersion: v1
kind: POD
metadata:
    name: my-POD
spec:
    containers:
    - name: busybox-container
      image: busybox:latest
```

Il file verrá salvato come file YAML (ad esempio *POD.yaml*) e per applicare le modifiche si dovrá eseguire il seguente comando

```
#comando per creare un POD a partire da una file di configurazione YAML
$ kubectl create -f POD.yaml
```

Il risultato finale é analogo al comando di esempio riportato per la sintassi imperativa.

Ogni file YAML creato per una configurazione Kubernetes, deve contenere obbligatoriamente quattro chiavi:

- `apiVersion`

  questo campo identifica quale versione delle API viene dichiarata. Ogni tipo di risorsa appartiene a un certo tipo di `apiVersion` e quindi deve essere indicato all'interno della configurazione

- `kind`

  questo campo indica il tipo di risorsa che si vuole creare. Nell'esempio riportato sopra indica che si sta definendo la configurazione di un `POD`

- `metadata`

  questo campo viene utilizzato per identificare in modo univoco la risorsa che si crea. Nell'esempio viene specificato il nome che assumerá il POD con la coppia `name: my-POD`

- `spec`

questo campo é dove viene definito come si vuole configurare effettivamente la risorsa e quindi se ne definisce il suo stato.

Con la sintassi dichiarativa, si riesce a mantenere traccia di tutto il lavoro effettuato sul cluster, creando di fatto una infrastruttura come codice **IaC (Infrastructure as Code)**. In questo modo nel caso di perdita dello stato del cluster, sará possibile ripristinare tutto in maniera veloce e pulita.

2.3.4 Oggetti Kubernetes

Gli oggetti Kubernetes sono entitá persistenti nel sistema Kubernetes. Kubernetes utilizza queste entitá per rappresentare lo stato del cluster. Nello specifico possono descrivere:

- Quali applicazioni containerizzate sono in esecuzione (e su quali nodi)
- Le risorse disponibili per tali applicazioni
- I criteri relativi al comportamento di tali applicazioni, ad esempio criteri di riavvio, aggiornamenti e tolleranza agli errori

Un oggetto Kubernetes é un insieme di specifiche esplicative su come si vuole che funzioni: una volta creato l'oggetto, il sistema Kubernetes lavorerá costantemente per garantire che l'oggetto esista e che sia esattamente come é stato dichiarato. Creando un oggetto, si sta effettivamente dicendo al sistema Kubernetes come si vuole creare e gestire il carico di lavoro del cluster; rappresenta quindi una descrizione dello stato che deve avere quell'ogggetto all'interno del cluster. Per lavorare con gli oggetti Kubernetes, per crearli, modificarli o eliminarli, si dovranno utilizzare le API Kubernetes. Utilizzando l'interfaccia della riga di comando kubectl, ad esempio, la CLI effettua le chiamate API Kubernetes necessarie.

Quasi tutti gli oggetti Kubernetes includono due campi oggetto nidificati che regolano la configurazione dell'oggetto: la specifica dell'oggetto e lo stato dell'oggetto. Per gli oggetti che hanno una `spec`, bisogna impostarla quando si crea l'oggetto, fornendo una descrizione delle caratteristiche che le risorse devono avere e quindi il suo stato desiderato.

Lo stato descrive la configurazione corrente dell'oggetto, fornito e aggiornato dal sistema Kubernetes e dai suoi componenti. Il piano di controllo Kubernetes gestisce continuamente e attivamente lo stato effettivo di ogni oggetto in modo che corrisponda allo alla configurazione fornita.

POD

I **POD** sono la piú piccola unitá di distribuzione in Kubernetes. Risiedono sui nodi del cluster e hanno il loro indirizzo IP, il che gli consente di comunicare attraverso il resto del sistema. Un singolo POD puó ospitare uno o piú container, fornire storage e risorse di rete.

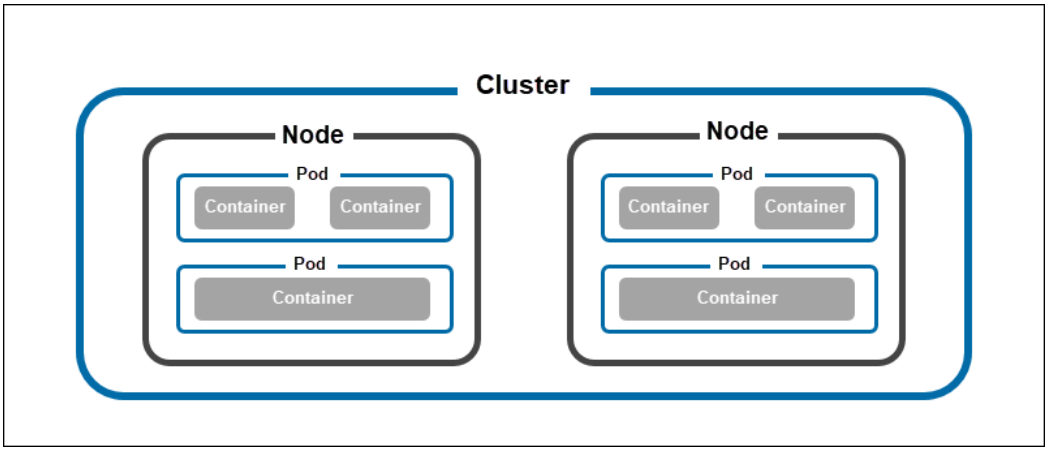

Figura 2.7: POD

Una delle caratteristiche chiave dei POD Kubernetes é che sono effimeri. In pratica, un POD puó guastarsi senza influire sul funzionamento del sistema. Kubernetes sostituisce automaticamente ogni POD non riuscito con una nuova replica del POD e mantiene il cluster in esecuzione.

Oltre ad essere wrapper di container, i POD memorizzano anche informazioni di configurazione che istruiscono Kubernetes su come eseguire i container.

Services

I servizi forniscono un modo per esporre le applicazioni in esecuzione nei POD. Il loro scopo é rappresentare un insieme di POD che eseguono la stessa funzione e impostare la policy per l'accesso a tali POD. Sebbene gli errori in un POD siano un evento previsto in un cluster, Kubernetes sostituisce il POD non funzionante con una replica avente un indirizzo IP diverso. Questo peró crea problemi di comunicazione tra POD che dipendono l'uno dall'altro. Utilizzando il processo kube-proxy che viene eseguito su ogni nodo del cluster, Kubernetes esegue il mapping dell'indirizzo IP virtuale del servizio agli indirizzi IP del POD. Questo processo consente una rete interna piú semplice, ma consente anche di esporre la distribuzione a reti esterne tramite tecniche come il bilanciamento del carico.

Esistono diverse tipologie di servizio. Kubernetes permette di decidere se si vuole esporre il servizio sono all'interno del cluster oppure se esporlo in modo che sia raggiungibile dall'esterno.

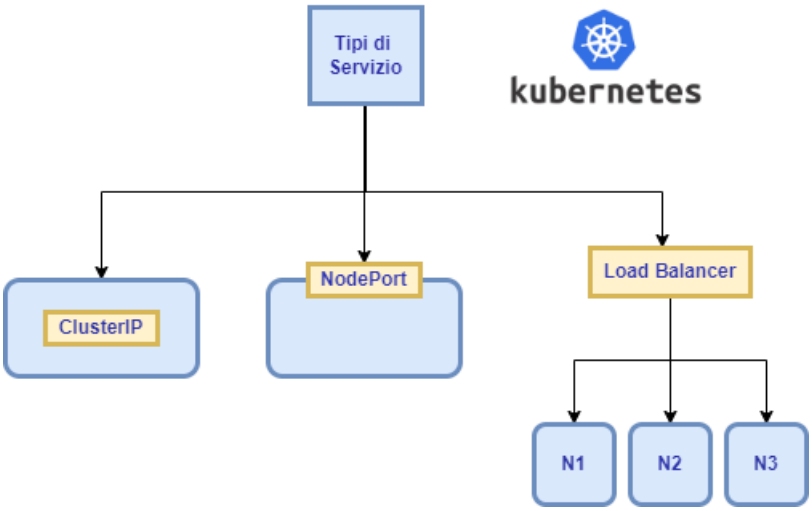

Figura 2.8: Tipi di Servizio

- `ClusterIP`: Espone il servizio su un IP interno al cluster. La scelta di questo valore rende il servizio raggiungibile solo dall'interno del cluster. Questo é il valore predefinito utilizzato se non si specifica in modo esplicito un tipo per un servizio.
- `NodePort`: Espone il servizio sull'IP di ciascun nodo su una porta statica (la NodePort). Per rendere disponibile la porta del nodo, Kubernetes configura un indirizzo IP del cluster, come se avessi richiesto un servizio di tipo: ClusterIP. Le porte del nodo su cui é possibile mappare i servizi sono di default comprese tra **30000** e **32767**.
- `LoadBalancer`: Espone il servizio esternamente utilizzando un load balancer del fornitore cloud.

Deployments

I Deployments sono oggetti controller che forniscono istruzioni su come Kubernetes deve gestire i POD che ospitano un'applicazione containerizzata. Utilizzando le distribuzioni, gli amministratori possono: ridimensionare il numero di repliche di POD;

implementare il codice aggiornato; eseguire rollback alle versioni precedenti del codice. Una volta creato, il controller di distribuzione monitora l'integritá dei POD e dei nodi. In caso di guasto, distrugge i POD guasti e ne crea di nuovi. Puó anche bypassare i nodi malfunzionanti, consentendo all'applicazione di rimanere funzionante anche quando si verifica un errore hardware.

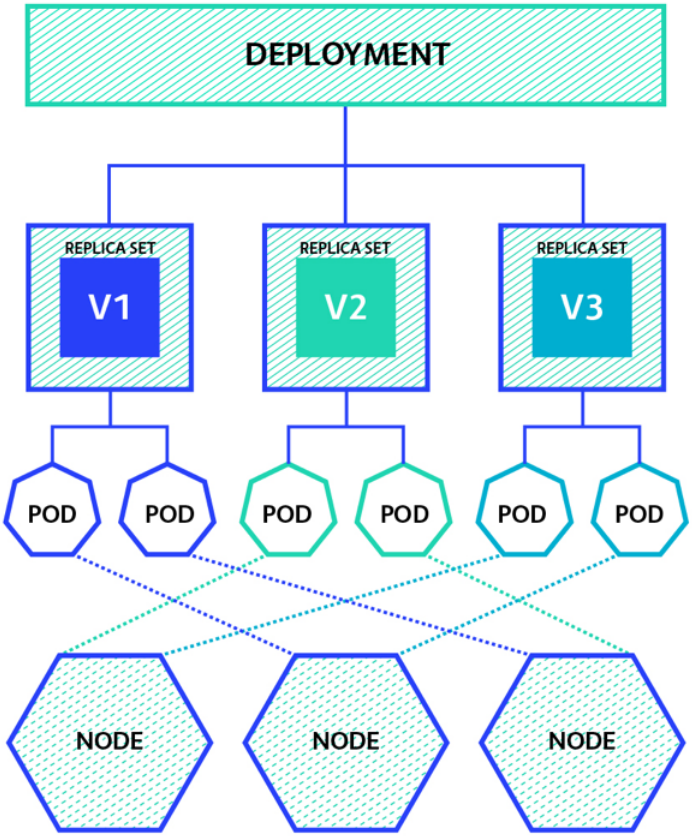

Figura 2.9: Come lavorano i Deployment

ReplicaSets e ReplicationControllers

I ReplicationControllers e i ReplicaSets assicurano che il numero corretto di repliche di POD sia sempre in esecuzione sul cluster. Quando si crea uno di questi oggetti, si

deve specificare il numero desiderato di POD. Il controller ne mantiene quindi stabile la quantitá stabilita, creando POD aggiuntivi e terminando quelli extra quando necessario.

DaemonSets

I DaemonSet sono oggetti controller il cui scopo é garantire che POD specifici vengano eseguiti su nodi specifici (o tutti) nel cluster. Lo scheduler Kubernetes ignora i POD creati da un DaemonSet, quindi quei POD durano finché esiste il nodo. Questo oggetto é particolarmente utile per configurare i demoni che devono essere eseguiti su ciascun nodo, come quelli utilizzati per l'archiviazione del cluster, la raccolta dei registri e il monitoraggio dei nodi. Per impostazione predefinita, un DaemonSet crea un POD su ogni nodo del cluster.

Namespaces

Lo scopo dell'oggetto Namespace é fungere da separatore di risorse nel cluster. Un singolo cluster puó contenere piú namespace, consentendo di organizzare meglio il cluster e semplificare l'allocazione delle risorse. Un cluster viene fornito con piú namespace creati per scopi di sistema e il namespace predefinito per gli utenti. Si possono quindi creare un qualsiasi numero di namespace aggiuntivi, ad esempio per lo sviluppo o per i test.

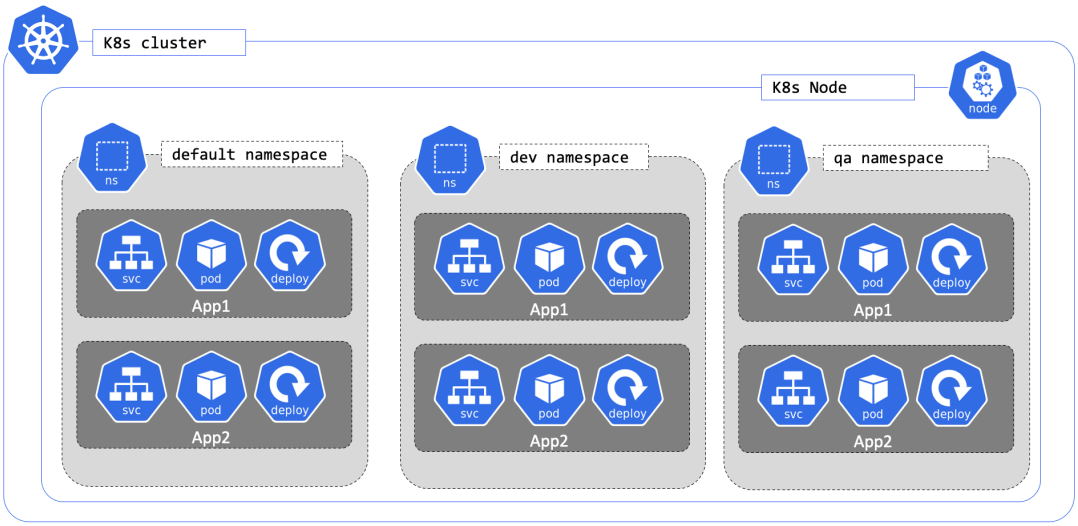

Figura 2.10: Namespaces

Volumes

I volumi sono oggetti il cui scopo é fornire spazio di archiviazione ai POD. Esistono due tipi fondamentali di volumi in Kubernetes:

- I volumi temporanei persistono solo durante la durata del POD a cui sono collegati.
- Volumi persistenti, che non vengono distrutti quando il POD si arresta in modo anomalo. Kubernetes utilizza le PVC per eseguire il provisioning dei volumi, che fungono quindi da collegamenti tra i POD e lo storage fisico.

Ingress

Gli Ingress sono una risorsa API che consente di esporre i servizi HTTP e HTTPS al traffico esterno. Gli Ingress fungono da livello di ingresso per il traffico di rete in un cluster Kubernetes, definendo le regole di routing per il traffico HTTP e HTTPS in ingresso dal mondo esterno ai servizi all'interno del cluster. Sono composti da regole di routing che specificano come gestire il traffico in ingresso, come l'URL, la porta e il servizio di destinazione per ogni richiesta. Possono anche includere altre opzioni, come la configurazione della sicurezza HTTPS e le regole di bilanciamento del carico.

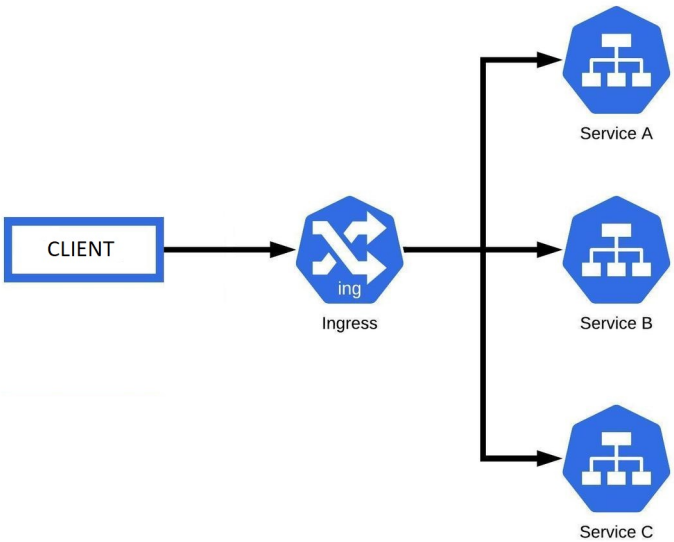

Figura 2.11: Ingress

Gli Ingress sono supportati da molti controller Ingress di terze parti, come NGINX, i quali forniscono funzionalitá aggiuntive come il bilanciamento del carico e la configurazione di sicurezza avanzata. Inoltre, gli Ingress possono essere configurati per funzionare con certificati SSL/TLS per garantire la sicurezza delle comunicazioni.

2.3.5 Architettura HA in Kubernetes

Per configurare un cluster Kubernetes che rispetti le regole per garantire l'alta disponibilitá sono essenzialmente due.

- Con nodi **Control Plane** stacked, ovvero dove le componenti **etcd** sono collocate insieme ai nodi del piano di controllo
- Con le componenti etcd esterne, dove etcd viene eseguito su nodi separati dal **Control Plane**

Topologia etcd stacked

Un cluster HA stacked é una topologia in cui il cluster di storage dei dati distribuito fornito da **etcd** é sovrapposto al cluster formato dai nodi gestiti da kubeadm che eseguono i componenti del piano di controllo.

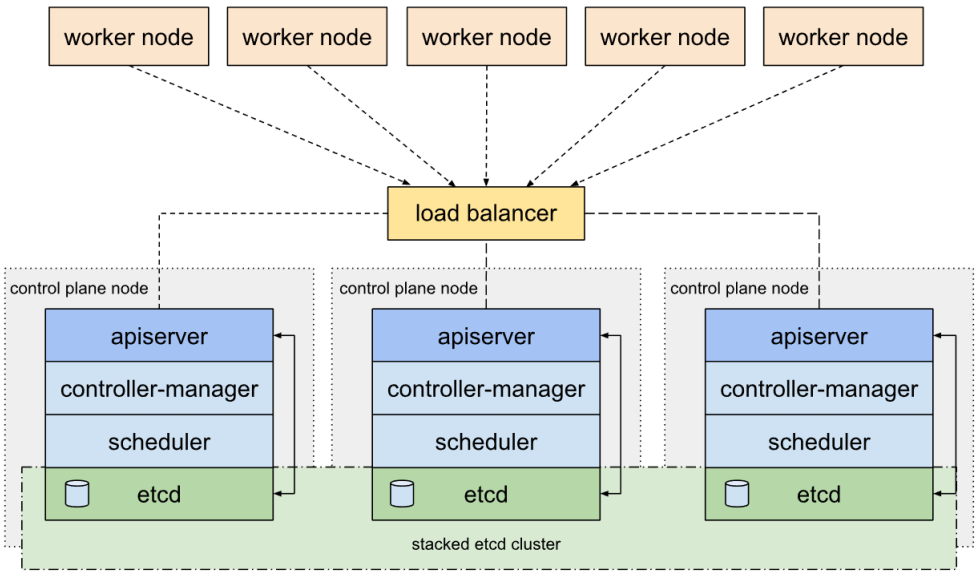

Figura 2.12: Topologia **etcd** stacked

Ogni Control Plane esegue un'istanza di kube-apiserver, kube-scheduler e kube-controller-manager. Il kube-apiserver é esposto ai nodi di lavoro utilizzando un load balancer.

Ogni nodo del Control Plane crea un membro etcd locale e questo membro etcd comunica solo con il kube-apiserver di questo nodo. Lo stesso vale per le istanze locali di kube-controller-manager e kube-scheduler.

Questa topologia accoppia i Control Plane e i membri etcd sugli stessi nodi. é piú semplice da configurare rispetto a un cluster con nodi etcd esterni e piú semplice da gestire per la replica.

Tuttavia, un cluster stacked corre il rischio di accoppiamento non riuscito. Se un nodo si interrompe, sia un membro etcd che un'istanza del Control Plane vengono persi e la ridondanza viene compromessa. Si puó questo rischio aggiungendo piú nodi Control Plane.

Si dovrebbe quindi quindi eseguire un minimo di tre nodi del piano di controllo in pila per un cluster HA. Questa é la topologia predefinita in kubeadm. Un membro etcd locale viene creato automaticamente sui nodi del Control Plane quando si utilizzano kubeadm init e kubeadm join –Control Plane.

Topologia etcd esterno

Un cluster HA con etcd esterno é una topologia in cui il cluster di archiviazione dati distribuito fornito da etcd é esterno al cluster formato dai nodi che eseguono i componenti del Control Plane.

Analogamente alla topologia etcd in stack, ogni nodo del piano di controllo in una topologia etcd esterna esegue un'istanza di kube-apiserver, kube-scheduler e kube-controller-manager. E il kube-apiserver é esposto ai nodi di lavoro utilizzando un load balancer. Tuttavia, i membri etcd vengono eseguiti su host separati e ciascun host etcd comunica con il kube-apiserver di ciascun nodo del Control Plane.

Questa topologia disaccoppia il Control Plane e la componente etcd. Fornisce quindi una configurazione HA in cui la perdita di un'istanza del Control Plane o di una componente etcd. In questo caso si ha un impatto minore e non influisce sulla ridondanza del cluster tanto quanto la topologia HA stacked.

Tuttavia, questa topologia richiede il doppio del numero di host rispetto alla topologia HA stacked. Per un cluster con questa topologia sono richiesti almeno tre host per i nodi del Control Plane e tre host per i nodi etcd.

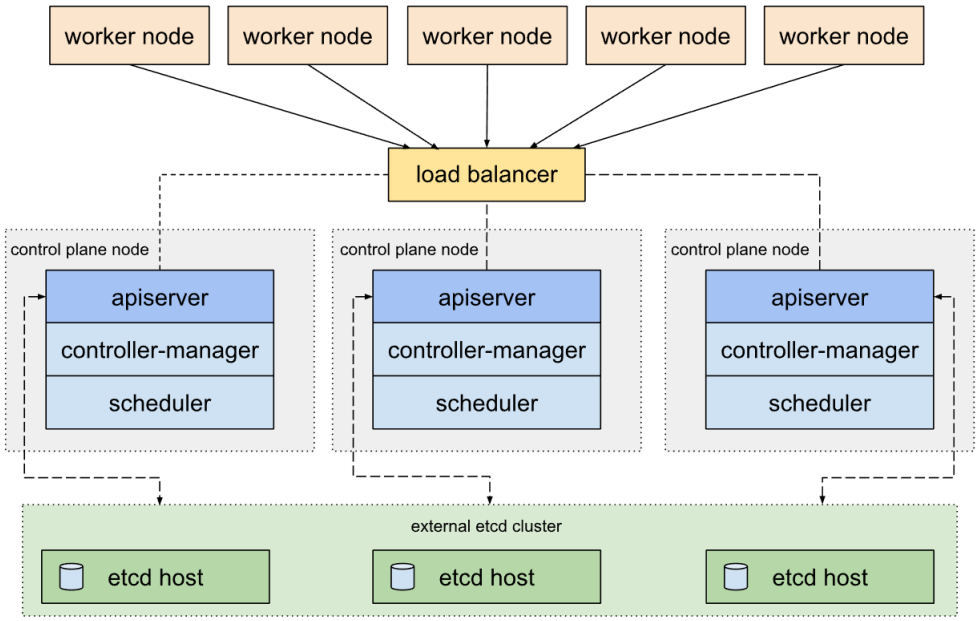

Figura 2.13: Topologia **etcd** esterno

Capitolo 3

Realizzazione di un cluster Kubernetes per la gestione di un cluster HPC

***Obiettivi**: in questo capitolo verranno esposte nel dettaglio le scelte progettuali e la fase operativa di studio e implementazione del sistema e del cluster Kubernetes.*

- *Scelte Progettuali*
- *Il Cluster*
- *xCAT ed installazione dei nodi*
- *Installazione Kubernetes e configurazioni nodi load-balancer*

3.1 Scelte progettuali

Prima di procedere con la parte operativa, é necessario discutere della progettazione e dei requisiti che il sistema dovrá rispettare. Si puó riassumere questa parte in diverse fasi, necessarie a capire e a decidere la distribuzione e il comportamento finale del sistema:

Conoscenza dell'hardware : conoscere l'hardware e i dispositivi che si utilizzano. Conoscere quindi la conformazione dei nodi, dei dispositivi di rete e i cavi per le connessioni;

Progettazione della rete : studiare il modo con cui le macchine si connettono tra di loro, con la rete CINECA e con l'esterno;

Distribuzione dei nodi : la struttura sotware dei nodi é fondamentale per poi costruire il cluster Kubernetes e permettere di implementare le soluzioni che garantiscono l'alta disponibilitá. Tramite gli strumenti di distribuzione dei nodi é possibile scrivere la configurazione che le macchine devono avere dopo l'installazione;

Implementazione del software : infine si procede con l'implementazione di tutto l'ecosistema, quindi l'installazione e la configurazione del cluster Kubernetes e i server LoadBalancer.

3.1.1 Nodi del cluster

Nella configurazione finale del cluster si avranno dieci nodi interconnessi. I nodi si compongono di due master, con il ruolo di macchine di controllo, 6 nodi compute (***m511n[01-06]***), su cui verrá installato il cluster Kubernetes e due nodi che fungeranno da load-balancer (***m511n[07-08]***).

Ogni nodo sará collegato con due tipi di connessione. Una connessione ad 1Gb/s come rete lan di gestione del cluster e una connessione tramite rete a 10/25Gb/s che verrá utilizzata per lo scambio di dati, come accesso ai servizi e come accesso allo storage distribuito.

I nodi master saranno i nodi sui quali verranno salvate le configurazioni di distribuzione e di installazione dei nodi di calcolo, oltre che essere il punto di accesso per i sistemisti alle macchine del cluster. Dai nodi master si potrá quindi avere accesso ad ogni singola componente ed informazione del sotware e dell'hardware presente nel sistema. Uno dei nodi che verrá utilizzato come master ha giá presente al suo interno un'installazione appartenente ad un precendente sistema ormai dismesso.

La definizione dell'immagine di installazione dei master del nostro sistema é stata configurata a partire da questo nodo. Al suo interno era giá presente Podman. Podman é un motore di container open-source che consente di creare, gestire e distribuire container Linux. Come alternativa a Docker, Podman offre un'esperienza di containerizzazione simile ma con alcune differenze chiave, ad esempio non richiede un daemon per funzionare e non richiede privilegi di root per eseguire i container. Sfruttando il software é stato installata una distribuzione xCAT containerizzata. Questa decisione é stata presa in modo da poter trasferire questa installazione sui nuovi master senza avere problemi di compatibilitá.

Sul nuovo master (da ora verrá chiamato ***master01***) verrá implementata l'istallazione xCAT basata sulla stessa distribuzione e sará utilizzata per la definizione delle immagini utili per il deploy di tutte le macchine presenti nel cluster.

Il master su cui era stata fatta partire la prima installazione (da ora verrá chiamato ***master02***), verrá ricreato partendo dall'immagine creata per il nodo master01.

Sui nodi master ci sará un sistema di distribuzione e sincronizzazione di file comuni ai vari gruppi di macchine. Il sotware si chiama **NodeTools** ed é stato sviluppato dagli amministratori di sistema di CINECA. Il sistema permette, tramite l'utilizzo delle informazioni presenti nel database xCAT, di ditribuire e sincronizzare i file di configurazione sui vari nodi. Il programma permette di utilizzare una specifica sintassi con la quale definire cartelle e nomi dei file. Tramite questa sintassi é possibile definire i percorsi di destinazione e l'insieme dei nodi ai quali sono destinati.

Il resto dei nodi verrá installato con un comparto software minimale. Le componenti principali, dato che l'obiettivo é avere un sistema Kubernetes, saranno lo strumento di deploy del cluster Kubernetes, lo strumento di controllo del cluster e l'ambiente di runtime dei container. Il resto delle funzionalitá verrá implementato tramite Kubernetes e i suoi strumenti. Unica eccezione saranno i nodi che dovranno implementare il sistema di load balancing. Su questi nodi saranno installati due strumenti necessari a costruire un sistema di load balancing ovvero **Keepalived** e **HAProxy**.

3.1.2 Sistema Operativo

Per tutti i nodi del cluster é stato scelto come sistema operativo **Rocky Linux 8.7** [15]. L'installazione dei nodi master utilizza la versione completa. Questo per garantire la presenza dei software necessari subito dopo l'installazione. Per i nodi compute invece si utilizzerá la distribuzione minimale.

Rocky Linux é un sistema operativo gratuito basato su Red Hat Enterprise Linux (RHEL). La nascita di questo sistema operativo corrisponde con l'annuncio un'annuncio

sulle prossime distribuzione CentOS. Le future distribuzioni di CentOS (a partire dalla versione 8) saranno rilasciate come distribuzioni "upstream" e non riceveranno piú i test e le verifiche necessarie per un prodotto di utilizzo aziendale.

Le aziende sono molto piú interessate alla stabilitá, ad un fase di test approfondita e alla compatibilitá hardware. Rocky Linux si propone come sistema progettato per essere bug-for-bug compatibile con RHEL e quindi garantendo le caratteristiche che servono in un ambiente Enterprise. Per ogni release Rocky linux garantisce aggiornamenti regolari e un supporto di 10 anni.

3.1.3 Progettazione della rete

La rete del sistema si divide essenzialmente in due parti, rete gigabit e rete veloce. La rete gigabit deve garantire la comunicazione tra i nodi, accedere agli switch e come rete di configurazione delle BMC. La rete veloce invece é stata pensata come canale di cominicazione per lo scambio di dati, per tutto quello che riguarderá l'accesso al cluster Kubernetes e ai vari servizi implementati e come punto di ingresso per i filesystem distribuiti.

Le connessioni gigabit confluiranno tutte su un switch interno al rack. Le reti veloci invece saranno collegate e configurate su uno switch con porte 10/25GbE esterno al sistema principale.

Per ogni tipo di connessione non si avrá una singola lan, ma per esigenze di amministazione dovranno essere presenti anche varie VLAN per garantire l'accesso per l'amministazione delle componenti di rete e delle BMC da parte del master.

Anche le reti veloci presentano piú reti implementate su una singola connessione. In tutto ci saranno quattro VLAN. Due VLAN sono state pensate per garantire le prestazioni nessessarie a non creare colli di bottiglia al cluster Kubernetes e per l'accesso agli storage distribuiti. Due VLAN invece non saranno utilizzate nel progetto. Sono state create in previsione di un ampliamento del sistema per la creazione di un sistema oVIRT, di cui peró non si parlerá in questa tesi.

Master

Le macchine che hanno funzione di **master** devono avere accesso a ogni singola rete prensente nel sistema, oltre che avere accesso ad internet e alla varie reti di gestione e di amministrazione.

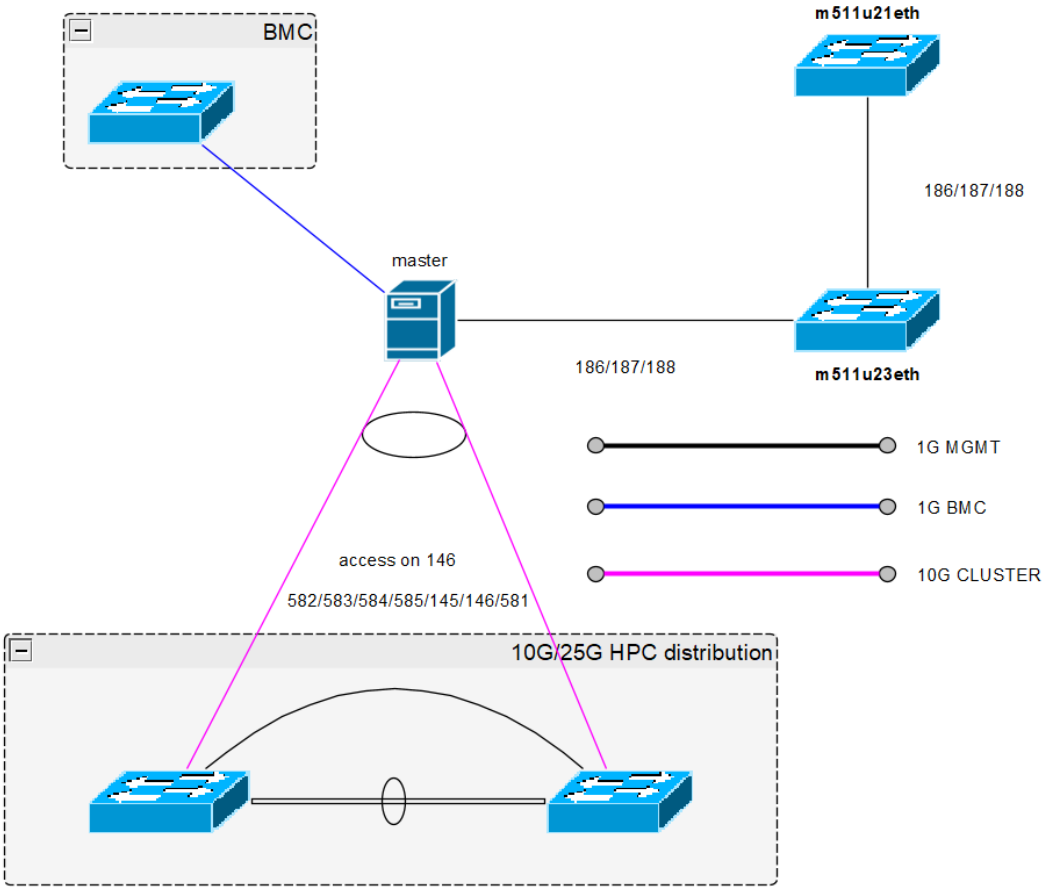

Figura 3.1: Architettura rete nodi master

Sulla rete interna, ovvero collegata sullo switch **m511n23** abbiamo bisogno di tutte e tre le VLAN per accedere alle macchine, alle interfacce di gestione degli switch e infine per poter accedere a tutte le BMC dei nodi computazionali.

La rete veloce é in ridondanza tramite l'implementazione di un bond tra le due interfacce collegate. Su questo bond sono presenti sia le VLAN interne al cluster, sia tutte quelle necessarie al collegamento con le reti di gestione e amminstrazione di CINECA, sia la rete necessaria a garantire l'accesso ad internet.

Fare il bonding di una rete é quel processo che permette di raggruppare due o piú interfacce di rete in una singola interfaccia. Questa tecnica permette di incrementare le performance e la ridondanza, aumentando la portata e la larghezza di banda della rete.

Se un'interfaccia é guasta oppure é stata semplicemente scollegata la connessione sará comunque operativa, essendo l'altra ancora online. Questa tecnica risulta ottimale nel caso in cui si voglia garantire alla rete una tolleranza ai guasti e un buon bilanciamento del carico. L'insieme delle porte combinate prende il nome di **Link Aggregation Group (LAG)**

A differenza dei nodi computazionali, la BMC dei nodi master, non deve essere collegata internamente, ma deve essere accessibile da una rete esterna. Questo é necessario perché nel caso si verifichino guasti interni al cluster, sia possibile verificare lo stato della macchina da remoto attraverso una rete differente e con caratteristiche di alta disponibilitá.

Compute

I nodi computazionali sulla rete gigabit hanno solo configurato l'indirizzo della rete interna per la gestione delle macchine. Per questi nodi non si ha la necessitá di configurare le reti che permettono l'accesso al resto dei dispositivi.

La rete veloce, come sui master, é ridondata tramite l'implementazione di un bond. Si puó notare che sono presenti due gruppi diversi di macchine, ogni gruppo ha come default una rete diversa. Questo é stato fatto in previsione di eseguire prove di installazione di un diverso sistema, per separarlo dall'area di servizio del cluster Kubernetes.

3.1.4 Distribuzione e installazione dei nodi

Per l'installazione e la distribuzione dei nodi si utilizza xCAT. Una volta scelto il sistema operativo e scaricata la relativa immagine si procede con la customizzazione tramite la creazione di un'immagine xCAT *(osimage)*.

Si procede anche con la definizione dei nodi che compongono il nostro sistema. Per ogni nodo é possibile configurarne ogni singola componente. Particolare attenzione é stata posta sulle configurazioni della rete e delle NIC.

Nella distribuzione del sistema operativo scelto non sono presenti molti dei componenti che servono per l'implementazione di tutti gli strumenti necessari. Durante la fase di modifica della **osimage** é possibile creare una propria repository scaricando dalla rete pacchetti delle installazioni necessarie. Sono quindi stati scaricati e aggiunti alla repository i software per l'implementazione delle macchine con funzione di load-balancing e le componenti necessarie all'inizializzazione del cluster Kubernetes **kubeadm**, **kubelet** e il container runtime **CRI-O**[16].

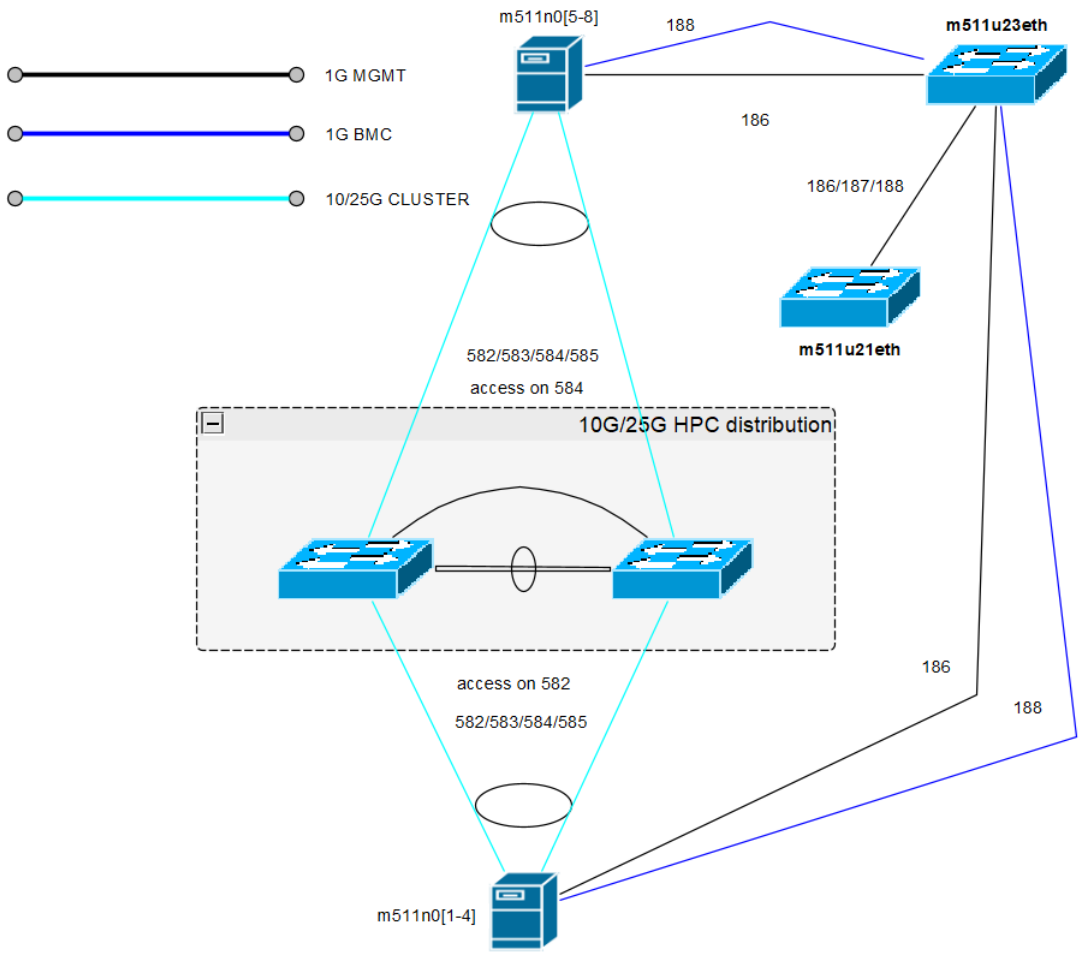

Figura 3.2: Architettura rete nodi computazionali

3.1.5 Deploy di Kubernetes

Esistono diversi metodi e strumenti per l'implementazione di un cluster Kubernetes. Il metodo piú complesso prevede l'installazione manuale di tutte le componenti necessarie al funzionamento del cluster, come lo scheduler, gli apiserver e i vari controller oltre che alla distribuzione manuale di tutte le chiavi e i certificati necessari a far parlare tra di loro i vari software. Per un'installazione automatica, in cui si vuole arrivare il prima possibile ad un livello di produzione, esistono strumenti che tramite la defizione di regole e comportamenti, permettono di automatizzare tutto il processo senza bisogno di implementare manualmente. Ad esempio, é possibile definire i plugin che garantiscono

funzionalitá come la rete e l'archiviazione.

Si é scelta una soluzione ibrida, **kubeadm**. Kubeadm é uno strumento di provisioning di cluster Kubernetes che permette di creare un cluster completamente funzionante in modo automatizzato e ripetibile. Kubeadm fornisce dei comandi che permettono di effettuare un'installazione minimale di un cluster Kubernetes. In base alla progettazione, si preoccupa solo del bootstrap, non del provisioning delle macchine. Allo stesso modo, l'installazione di vari componenti aggiuntivi utili, come Kubernetes Dashboard, soluzioni di monitoraggio e componenti aggiuntivi come storage distribuito o soluzioni di networking, non rientra nell'ambito.

Kubeadm é stato sviluppato per semplificare il processo di creazione di cluster Kubernetes da zero. É possibile utilizzarlo per installare un cluster con un solo comando fornendo semplicemente alcune informazioni di configurazione. Inoltre, é altamente configurabile, consentendo agli amministratori di personalizzare l'installazione in base alle proprie esigenze specifiche.

Avremo a disposizione sei macchine per l'installazione del cluster Kubernetes. Volendo un'approccio ad alta disponibilitá, verranno utilizzati tre dei sei nodi come nodi master e i rimanenti tre come nodi worker. Non si avrá l'accesso diretto ai nodi master, ma le richieste verranno effettuate attraverso un indirizzo ip virtuale configurato all'interno di due macchine proxy con funzione di load-balancing, questo per permettere la distribuzione equa delle richieste e delle operazioni. Le macchine worker ospiteranno i servizi che si vogliono testare.

Plugin di rete

Come plugin di rete si é scelto **CALICO**

CALICO é una soluzione di rete e sicurezza open source che puó essere utilizzato come plug-in di rete in un cluster Kubernetes. Fornisce l'applicazione delle policy di rete, la gestione degli indirizzi IP e le funzionalitá di isolamento della rete essenziali per l'esecuzione di un cluster Kubernetes sicuro e affidabile.

Esistono in realtá una moltitudine di plug-in di rete per kubenetes. In previsione della creazione di un sistema piú complesso si é scelto di utilizzare CALICO, perché permette, a scapito di una maggiore complessitá, di definire regole relative alle policy e alla sicurezza della rete e garantisce ottime performance.

Monitoraggio del sistema

Come sistema di monitoraggio si é scelto una combinazione dei software **Prometheus** e **Grafana**. Prometheus sará il software incaricato della raccolta di tutte le informazioni e le metriche che indicano lo stato e le caratteristiche del sistema. Grafana invece ci fornirá l'ambiente che riguarda la produzione di grafici, tabelle e indicatori. I software verranno installati come distribuzione Kubernetes.

Altro software che si vorrebbe testare, sempre come distribuzione Kubernetes é la componente **Kubernetes dashboard**. Kubernetes Dashboard é un'interfaccia utente basata sul Web che fornisce una rappresentazione grafica di un cluster Kubernetes. Consente agli utenti di visualizzare e gestire varie risorse in un cluster, come POD, servizi, distribuzioni e nodi, da un'unica interfaccia unificata.

3.1.6 Architettura HA e load balancer

Utilizzando kubeadm come strumento di installazione del cluster, serve installare manualmente un sistema di load balancing che permetta di ottenere un cluster Kubernetes che rispetti le regole per l'alta disponibilitá.

Si é scelto di procedere adottando una soluzione di load balancing con una combinazione di due applicazioni:

- HAProxy [17]
- keepalived [18]

Il compito di keepalived é quello di eseguire un controllo continuo che dinamicamente e adattivamente mantiene attivi i server di load balancing in rapporto alla loro salute. Keepalived implementa VRRP (Virtual Router Redundancy Protocol) su un sistema Linux e gestisce la configurazione di Linux Virtual Server. Keepalived puó implementare configurazioni ad alta disponibilitá (attiva/passiva) e bilanciamento del carico (attivo/attivo) che possono essere rese reattive a diversi fattori personalizzabili.

HAProxy fornisce un load balancer ad alta disponibilitá e un reverse proxy per applicazioni basate su TCP e HTTP che distribuiscono le richieste attraverso piú server.

L'obiettivo sará quindi di utilizzare gli indirizzi IP virtuali creati con keepalived come punto di accesso per i servizi eseguiti sul cluster Kubernetes e utilizzare HAProxy che intercetta le richieste fatte utilizzando porte e indirizzi IP definiti per redirigere il traffico su una delle macchine su cui esegue realmente il servizio.

Nell'implementazione del sistema si avranno quindi due macchine, una con stato MASTER e una con stato BACKUP, con prioritá maggiore nel primo caso e minore nel

secondo. Questo significa che all'avvio del servizio l'indirizzo IP virtuale sará presente sul nodo con prioritá maggiore. Perché l'IP sia sempre attivo, su una delle macchine, bisogna spiegare a keepalived quali controlli effettuare e quali condizioni si devono rispettare. Nel caso di HAProxy, si vuole controllare che il demone del servizio sia sempre attivo. Nel caso in cui sorgano problemi, il controllo implementato dará un risultato negativo e si avvierá il processo di migrazione dell'indirizzo sull'altro nodo. Cosí facendo non si ha nessun rischio di avere l'indirizzo presente contemporaneamente su tutte le macchine. Per ogni indirizzo IP che si definisce é possibile configurare un diverso valore di prioritá e stato. Non é quindi obbligatorio che una macchina sia solo MASTER o solo BACKUP per tutti gli indirizzi.

Quando si configura un load balancer utilizzando HAProxy, ci sono due tipi di nodi che devono essere definiti: frontend e backend. Il frontend é il nodo attraverso il quale HAProxy ascolta le connessioni. I nodi back-end sono quelli tramite i quali HAProxy puó inoltrare le richieste. Le richieste vengono inoltrate a uno dei nodi di backend utilizzando uno degli algoritmi di schedulazione messi a disposizione (l'algoritmo default é la schedulazione roundrobin).

3.1.7 Storage e Filesystem Distribuito

Ci sarano due tipi di storage distribuito.

Uno coprirá tutte le esigenze del cluster Kubernetes, sará quindi implementato per garantire uno storage persistente ai POD del sistema. Si vuole provare ad implementare questo tramite la distribuzione Kubernetes del filesystem distribuito CephFS e dell'operatore Rook.

Rook é un orchestratore di storage nativo del cloud open source, che fornisce la piattaforma, il framework e il supporto per lo storage Ceph per l'integrazione nativa con gli ambienti nativi del cloud. Ceph é un sistema di archiviazione distribuito che fornisce archiviazione di file, blocchi e oggetti ed é distribuito in cluster di produzione su larga scala.

La scelta di far gestire tutti i demoni responsabili al funzionamento del filesystem da Kubernetes ha i suoi pro e i suoi contro.

Per questo motivo si é scelto di implementare anche un secondo filesystem distribuito, con il compito di immagazzinare tutti i dati ottenuti tramite l'esecuzione dei software di monitoraggio.

Il motivo di questa scelta é semplice: nel caso in cui il sistema di controllo e gestione del filesystem CephFS non sia piú disponibile, sará comunque possibile accedere alle

informazioni di monitoraggio del cluster Kubernetes e del sistema, cosí da essere in grado di effettuare i controlli e risolvere gli eventuali problemi.

3.2 Il Cluster

Nelle prime fasi del progetto si sono svolte tutte le attivitá necessarie a capire le macchine a disposizione per implementare il cluster. CINECA ha messo a disposizione due ex cluster con macchine fuori produzione. Le macchine che si hanno a disposizione comprendono:

- due nodi master
- un nodo di login
- sette nodi di storage
- Switch Lenovo Rack Switch G8052

3.2.1 Caratteristiche Hardware e installazione fisica

Dato che le macchine provengono da due sistemi differenti hanno specifiche hardware non omogenee. Nella tabella 3.1 viene riportato un riassunto delle loro specifiche

Il rack scelto per installare i nodi é quello in cui era giá presente uno dei sistemi messi a disposizione da CINECA. Il Rack é di marca IBM da 42 RU e 19 pollici di larghezza. Non essendo prevista l'aggiunta di nuovi nodi, é stato necessario installare ulteriori PDU (Power Distribution Unit).

Le PDU (Power Distribution Unit) sono dispositivi elettrici utilizzati per distribuire l'alimentazione elettrica ai dispositivi elettronici presenti in un rack o in un data center. Le PDU possono essere considerate come "prese di corrente intelligenti" in quanto forniscono una serie di funzionalitá per il controllo e la gestione dell'alimentazione.

3.2.2 Configurazione BMC

Un Baseboard Manager Controller (BMC) é un processore di servizio specializzato che monitora lo stato fisico di un computer, server di rete o altro dispositivo hardware utilizzando sensori e comunicando con l'amministratore di sistema tramite una connessione indipendente. Il BMC fa parte dell'Intelligent Platform Management Interface (IPMI) e di solito é contenuto nella scheda madre o nel circuito principale del dispositivo da monitorare. I sensori di un BMC misurano variabili fisiche interne come temperatura,

Nodo	CPU	RAM	Storage	Net	Dim
master01	2xIntel Xeon 8-Core	6x16GB	3x2TB HDD	4x1GbE 2x25GbE	1 RU
master02	2xIntel Xeon 8-Core	4x16GB	2x500GB HDD	4x1GbE 2x40GbE 2x10GbE	2 RU
m511n08	2xIntel Xeon 10-Core	8x16Gb	3x1TB HDD	4x1GbE 2x25GbE	2 RU
m511n [01-03]	2xIntel Xeon 8-Core	8x8GB	2x1TB HDD 2x1TB SSD 5x11TB HDD	2x1/10GbE 2x25GbE	2 RU
m511n [04-06]	2xIntel Xeon 8-Core	8x8GB	2x1TB HDD 2x1TB SSD 5x11TB HDD	2x1/10GbE 4x25GbE	2 RU
m511n07	2xIntel Xeon 8-Core	6x16GB	2x2TB HDD 10x12TB HDD	2x1/10GbE 2x25GbE	2 RU

Tabella 3.1: Specifiche Hardware dei nodi

umiditá, tensione di alimentazione, velocitá della ventola, parametri di comunicazione e funzioni del sistema operativo (OS).

Operazione fondamentale per poter poi gestire l'intero cluster, soprattutto nelle fasi di installazione, é stata la configurazione dei BMC. Su ogni nodo a disposizione é installato un BMC, il quale permette di collegarsi e controllare da remoto le macchine.

Per poter effettuare tali modifiche, considerando che i nodi erano giá configurati con le specifiche dei vecchi sistemi e che non erano presenti tutte le informazioni necessarie per accedere, si é dovuto ricorrere all'utilizzo di uno strumento che permettesse di accedere ai BMC direttamente dalla macchina.

Lo strumento in questione é chiamato IPMItool. IPMItool é uno strumento che permette per la gestione e la configurazione dei dispositivi che supportano Intelligent Platform Management Interface (IPMI). Di seguito viene riportata la procedura di modifica dell'indirizzo IP di una BMC:

```
#Controllare che ipmitool possa connettersi con la BMC

    $ ipmitool bmc info

#determinare il numero del canale utilizzato da IPMI su LAN. Per questo step si
procede per tentativi, partendo dal canale 1

    $ ipmitool lan print 1

#Quando troviamo il canale dove avviene la connessione, bisogna attivare la
connessione su questo canale

    $ ipmitool -I bmc lan set 1 access on

#Una volta che é possibile accedere alla BMC con ipmitool, settare un indirizzo IP
statico
#Non si deve solo modificare l'indirizzo IP ma anche i valori della netmask e del
gateway di default

    $ ipmitool -I bmc lan set 1 ipaddr 10.33.1.1
    $ ipmitool -I bmc lan set 1 netmask 255.255.255.0
    $ ipmitool -I bmc lan set 1 defgw ipaddr 10.33.1.250
```

Questa procedura é possibile solo nel caso in cui sul sistema sia installato un Sistema Operativo con installato il tool di gestione *ipmitool*.

Se nessun sistema operativo é installato sulle macchine, si puó ricorrere ad altri metodi, ad esempio tramite l'utilizzo dell'interfaccia specifica presente nel BIOS di setup all'avvio. L'interfaccia offre la possiblitá di effettuare le modifiche tramite l'utilizzo dello standard IPMI, ed é quindi possibile quindi definire le configurazioni desiderate.

3.2.3 Configurazione switch di Rete

Lo switch che garantisce le connessioni all'interno del cluster e su cui sono configurate le connessioni di management é un **Lenovo Rack Switch G8052**. Il G8052 include 48 porte Gigabit Ethernet (GbE) RJ-45 e 4 porte SFP+ da 10 GbE.

Il Lenovo Rack Switch G8052 [19] utilizza un'interfaccia di linea di comando (CLI) chiamata ISCLI (Intelligent Stacking Command Line Interface) per la gestione e la configurazione dello switch. L'ISCLI é una CLI basata su comandi testuali e consente di configurare e gestire lo switch tramite comandi da tastiera. Fornisce un metodo diretto

```
[root@m511n01 ~]# ipmitool lan print 1
Set in Progress         : Set Complete
Auth Type Support       : NONE MD2 MD5 PASSWORD
Auth Type Enable        : Callback : MD2 MD5 PASSWORD
                        : User     : MD2 MD5 PASSWORD
                        : Operator : MD2 MD5 PASSWORD
                        : Admin    : MD2 MD5 PASSWORD
                        : OEM      : MD2 MD5 PASSWORD
IP Address Source       : Static Address
IP Address              : 10.33.1.1
Subnet Mask             : 255.255.0.0
MAC Address             : ac:1f:6b:b7:5f:82
SNMP Community String   : public
IP Header               : TTL=0x00 Flags=0x00 Precedence=0x00 TOS=0x00
BMC ARP Control         : ARP Responses Enabled, Gratuitous ARP Disabled
Default Gateway IP      : 10.33.1.250
Default Gateway MAC     : 00:00:00:00:00:00
Backup Gateway IP       : 0.0.0.0
Backup Gateway MAC      : 00:00:00:00:00:00
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
RMCP+ Cipher Suites     : 1,2,3,6,7,8,11,12
Cipher Suite Priv Max   : XaaaXXaaaXXaaXX
                        :     X=Cipher Suite Unused
                        :     c=CALLBACK
                        :     u=USER
                        :     o=OPERATOR
                        :     a=ADMIN
                        :     O=OEM
Bad Password Threshold  : 0
Invalid password disable: no
Attempt Count Reset Int.: 0
User Lockout Interval   : 0
```

Figura 3.3: Esempio configurazione di rete BMC

per raccogliere informazioni sullo scambio ed eseguire lo scambio configurazione. Utilizzando un terminale, l'ISCLI consente di visualizzare informazioni e statistiche sullo switch ed eseguire qualsiasi configurazione necessaria.

Prima di parlare delle configurazioni, é necessario approfondire le VLAN e della differenza tra porte in access e porte in trunk.

Virtual Local Area Network - VLAN

Una VLAN (Virtual Local Area Network) é una rete locale virtuale che consente di suddividere una rete fisica in piú reti logiche separate. Ció consente di separare il traffico di rete in diverse VLAN, migliorando la sicurezza e l'efficienza della rete. Le VLAN sono utilizzate principalmente per isolare il traffico di rete e migliorare le prestazioni della rete, riducendo il traffico di broadcast e multicast.

É possibile configurare le VLAN in diversi modi e, a seconda della tipologia, viene applicata una tecnologia diversa. Nella pratica ritroviamo due tipi di applicazione: VLAN

Figura 3.4: Lenovo Rack Switch G8052

basate su porta o tagged VLAN ("VLAN taggate"). In molti casi gli amministratori di rete realizzano le installazioni e le assegnazioni implementando entrambe le tipologie.

VLAN basata su porta All'interno di uno switch ogni partecipante di rete viene indirizzato verso una porta, ovvero una presa all'interno della quale viene inserito il relativo cavo di rete collegato al computer di turno.

Tagged VLAN Nel caso della tagged VLAN l'assegnazione alla VLAN é piú dinamica: a garantire l'assegnazione é un tag nel frame del pacchetto, che sostituisce la permanente impostazione nello switch. Nel tag é contenuta l'informazione che indica in quale VLAN ci si trova al momento. Uno switch riconosce in quale segmento avviene la comunicazione e in base a questo inoltra il messaggio.

Porte in access - Porte in trunk

Le porte in access e le porte in trunk sono due tipi di configurazioni di porte utilizzati nei switch di rete per gestire il traffico di rete.

Una porta in access é una porta che é configurata per connettere un singolo dispositivo di rete, come un computer o un server. Quando una porta é configurata in access, il traffico che passa attraverso la porta appartiene a una singola VLAN, e la porta non trasmette informazioni sulle VLAN ad altre porte. In altre parole, il traffico che passa attraverso una porta in access non viene marcato o taggato in modo da indicare a quale VLAN appartiene.

D'altra parte, una porta in trunk é una porta configurata per trasmettere il traffico di piú VLAN attraverso la stessa connessione. Quando una porta é configurata in trunk,

la porta utilizza protocolli specifici per trasmettere informazioni sulle VLAN ad altre porte. Ció consente di separare il traffico di rete in diverse VLAN, senza dover utilizzare una porta separata per ogni VLAN.

Sullo switch sono state configurate le VLAN necessarie per la gestione del sistema. Ogni VLAN rappresenta la rete per l'accesso ai dispositivi. Ad esempio, la rete di gestione delle BMC é la VLAN 188. Sulle porte a cui verranno collegate le schede di rete delle BMC saranno configurate in access sulla VLAN 188.

```
#Esempio configurazione di una porta in access sulla VLAN 188

    interface port 1
        description "m511n01-bmc"
        switchport access vlan 188
        exit
```

Sulle porte dove sono collegati i nodi master invece deve essere possibile collegarsi con ogni VLAN. Questo perché come giá accennato, i nodi master devono avere accesso ad ogni dispositivo presente nel sistema.

```
#Esempio configurazione di una porta in trunk sulle VLAN 186, 187 e 188

    interface port 20
        switchport mode trunk
        switchport trunk allowed vlan 186-188
        switchport trunk native vlan 186
        exit
```

3.3 xCAT ed installazione dei nodi

In questa sezione verrá discussa la procedura utilizzata per la definizione dei nodi e la loro installazione. Come strumento di distribuzione é stato utilizzato il software xCAT. Non é stata utilizzata peró una versione bare-metal, ma si é optato per la versione containerizzata.

3.3.1 Installazione Podman e container di xCAT

Come runtime per i container é stato scelto Podman 3.4.7 e come versione del software di distribuzione é stato scelto xCAT 2.16.3.

Installazione Podman

Prima di eseguire l'installazione del software é stato necessario definire i file di configurazione, come i file di policy e dei registri. I file sono necessari a definire il comportamento del container runtime quando deve scaricare le immagini dei container. Con il file di policy si dichiarano le chiavi di collegamento ai registri proprietari, come il registro RedHat, o come si deve comportare nel caso di collegamenti che non utilizzano un certificato. Con il file dei registri si elencano i registri consultabili per il downalod delle immagini, ad esempio *quay.io*, *docker.io*.

Sistemate le configurazioni e installate le dipendenze, si puó procedere con l'installazione di Podman. In questo caso é stata eseguita un'installazione manuale, in quanto la versione offerta dalle repository del sistema operativo non era aggiornata e quindi non copatibile con la versione decisa.

Configurazione CNI Podman

Prima di procedere con l'installazione del container di xCAT si devono configurare le interfacce necessarie al software per poter comunicare successivamente con tutti i nodi e con le loro BMC.

Prima di configurare le interfacce dei container devono essere configurati i bridge sulle interfacce fisiche del nodo. Non serve configurare ogni interfaccia presente sui nodi, ma solo quelle necessarie ai fini dell'intallazione, ovvero la rete di management (186) e la rete delle BMC (188).

Un **Bridge** di rete é un dispositivo di rete che connette due o piú reti o segmenti di rete e consente ai dispositivi su ciascuna rete di comunicare tra loro. Il bridge funziona a livello di collegamento (layer 2) del modello OSI e utilizza l'indirizzo MAC dei dispositivi connessi per instradare i pacchetti tra le reti. Quando un pacchetto arriva su una delle interfacce del bridge, il bridge esamina l'indirizzo MAC di destinazione del pacchetto e determina su quale interfaccia deve essere trasmesso per raggiungere il dispositivo di destinazione.

Di seguito si mostra l'implementazione del bridge di rete sull'interfaccia VLAN eno1.188 utilizzando il client **nmcli** del software **NetworkManager**

```
#Creare l'interfaccia

    $ nmcli connection add type bridge con-name br188 \
      ifname br188

#Assegnare il nuovo bridge come interfaccia master della VLAN eno.188

    $ nmcli connection modify eno1.188 master br188

#Configurazione e avvio dell'interfaccia

    $ nmcli connection modify br188 ipv4.addresses '10.33.1.253/24'
    $ nmcli connection modify br188 ipv4.method manual
    $ nmcli connection modify br188 connection.autoconnect-slaves 1
    $ nmcli connection up br188
```

Terminata la configurazione dei bridge si puó procedere con la creazione delle interfacce dei container. Di seguito la configurazione dell'interfaccia configurata sulla rete della VLAN 188.

```
{
  "cniVersion": "0.4.0",
  "name": "xcat_br188",
  "interface_name": "eth2",
  "plugins": [
    {
      "type": "bridge",
      "bridge": "br188",
      "ipam": {
        "type": "host-local",
        "routes": [{ "dst": "0.0.0.0/0" }],
        "ranges": [
          [
            {
              "subnet": "10.33.1.0/24",
              "gateway": "10.33.1.251",
```

```
              "rangeStart": "10.33.1.253",
              "rangeEnd": "10.33.1.253"
            }
          ]
        ]
      }
    },
    {
      "type": "tuning",
      "capabilities": {
        "mac": true
      }
    }
  ]
}

#Al termine delle configurazioni Podman riconosce le interfacce create

    $ podman network ls

    NETWORK ID    NAME        DRIVER
    2f259bab93aa  podman      bridge
    8def10af566b  xcat_br186  bridge
    b71c7097e893  xcat_br188  bridge
```

Installazione container xCAT

Il container di xCAT viene creato manualmente utilizzando l'immagine del sistema operativo RockyLinux e i dockerfile creati ad hoc per l'installazione. L'utilizzo dei dockerfile permette di modificare la struttura interna del container, come la creazione delle cartelle necessarie e l'installazione di pacchetti e software.

Per costruire l'immagine modificata si utilizza il comando **podman build**. Il comando consente di creare una nuova immagine di container partendo da un Dockerfile o un altro file di specifica. In sostanza, il comando **podman build** esegue una serie di istruzioni definite nel file Dockerfile per creare l'immagine del container.

I dockerfile necessari permettono di configurare per prima cosa l'immagine del sistema operativo scelto, permettendo l'installazione dei software di xCAT. Partendo dall'immagine creata, con un altro file di configurazione Dockerfile, si eseguono le operazioni a creare l'abiente runtime di xCAT.

```
#Creazione dell'immagine base

    $ podman build --cap-add CAP_MKNOD --build-arg=arch=x86_64 \
    --build-arg=xcat_version=2.16.3 -v \
    /install/custom/repository/sw/xcat-2.16.3-rpms:/rpms \
    --build-arg=xcat_baseos=rh8 \
    --build-arg=xcat_firewall=1 \
    -t xcatbase -f Dockerfile.xcatbase .

#Modifica dell'immagine base creando l'ambiente runtime di xCAT

    $ podman build --cap-add CAP_MKNOD --build-arg=arch=x86_64 \
    --build-arg=xcat_version=2.16.3 -v \
    /install/custom/repository/sw/xcat-2.16.3-rpms:/rpms \
    --build-arg=xcat_baseos=rh8 \
    --build-arg=xcat_firewall=1 \
    -t xcatruntime -f Dockerfile.xcatruntime .

#Esecuzione del container

    $ podman run --name=xcat -d --net=xcat_br186,xcat_br188 \
    --hostname masterdk --privileged=true --pids-limit=8192 \
    --cgroups=no-conmon \
    --replace -e FINAL_INSTALL=/xcatdata/install \
    -v /xcatpod/xcatdata:/xcatdata \
    -v /xcatpod/xcatlogs:/var/log/xcat \
    -v /xcatpod/rootimage:/opt/rootimage \
    -v /etc/localtime:/etc/localtime:ro \
    -v /xcatpod/xcatdata/install:/install \
    -v /xcatpod/xcatlogs/consoles:/var/log/consoles \
    -v /install/rockylinux8.7:/rockylinux8.7 \
    --dns=none --no-hosts=true xcatruntime

#Comando per accedere all'interno del container
```

```
$ poman exec -it xcat bash
```

3.3.2 Definizione osimage

In xCAT, le OS image (abbreviazione di Operating System image) sono immagini di sistema operativo che vengono utilizzate per installare, configurare e gestire nodi nel cluster. Un'OS image contiene un'immagine del sistema operativo, i driver necessari per il hardware sottostante e altri pacchetti software correlati.

Partendo dalla stessa immagine del sistema operativo, ovvero l'immagine di RokyLinux 8.7, sono state create due osimage, una definita per l'installazione dei nodi master e una per l'installazione dei nodi computazionali come si vede in figura 3.5.

Le differenze sostanziali tra le due definizioni sono nei pacchetti installati. Mentre sui master non sono necessarie installazioni particolari, il loro stak sotware si limita a strumenti per fornire il servizio NTP o per implementare il server DNS, necessari al funzionamento dell'intero cluster. I nodi di calcolo invece, come giá accennato necessitano dei software per l'installazione del sistema di load-balancing e degli strumenti di Kubernetes.

L'immagine del sistema operativo scaricato contiene giá al suo interno tutti i software disponibili nelle sue repository. É quindi possibile installare i software presenti direttamente durante la fase di creazione del nodo, senza bisgno di una connessione diretta ad internet. Per indicare i pacchetti che si vogliono installare basta modificare il file *pkglist*.

Se si hanno esigenze di particolari software non presenti nelle repository di default, si puó creare una repository personale all'interno della cartella *otherpkgdir*. É necessario prima scaricare i pacchetti all'interno della cartella e poi creare la repository eseguendo il comando:

```
$ createrepo .
```

Perché durante l'installazione peró avvenga l'installazione, come nel caso precedente, bisogna indicare all'interno del file *otherpkglist* i loro nomi.

```
[root@master01 ~]# lsdef -t osimage compute-node-install
Object name: compute-node-install
    imagetype=linux
    osarch=x86_64
    osdistroname=rocky8-x86_64
    osname=Linux
    osvers=rocky8
    otherpkgdir=/install/custom/osimages/compute/22.12.0/otherpkgdir
    otherpkglist=/install/custom/osimages/compute/22.12.0/otherpkglist
    partitionfile=/install/custom/osimages/compute/22.12.0/partitionfile
    pkgdir=/install/custom/osimages/compute/22.12.0/pkgdir
    pkglist=/install/custom/osimages/compute/22.12.0/pkglist
    profile=storage
    provmethod=install
    synclists=/install/custom/osimages/compute/22.12.0/synclist
    template=/install/custom/osimages/compute/22.12.0/template
[root@master01 ~]# lsdef -t osimage management-22.12.0-install
Object name: management-22.12.0-install
    imagetype=linux
    osarch=x86_64
    osdistroname=rocky8-x86_64
    osname=Linux
    osvers=rocky8
    otherpkgdir=/install/custom/osimages/management/22.12.0/otherpkgdir
    otherpkglist=/install/custom/osimages/management/22.12.0/otherpkglist
    partitionfile=/install/custom/osimages/management/22.12.0/partitionfile
    pkgdir=/install/custom/osimages/management/22.12.0/pkgdir
    pkglist=/install/custom/osimages/management/22.12.0/pkglist
    profile=management
    provmethod=install
    synclists=/install/custom/osimages/management/22.12.0/synclist
    template=/install/custom/osimages/management/22.12.0/template
```

Figura 3.5: Caratteristiche e percorsi dei file di configurazione delle osimage

Infine é necessario, bisogna definire la configurazione dei dischi e delle partizioni. Il file incaricato a contenere queste informazioni é il file *partitionfile*.

```
zerombr
bootloader --append=" crashkernel=auto" --location=mbr --boot-drive=sda
clearpart --all --initlabel

part     biosboot      --fstype=biosboot --size=1     --ondisk=sda --asprimary
part     /boot         --fstype="xfs"    --size=1024    --ondisk=sda
part     pv.166        --fstype="lvmpv" --size=1024 --grow --ondisk=sda
part /boot/efi --fstype="efi" --ondisk=sda --size=250 --fsoptions="defaults,uid=0,gid=0,umask=0077,
shortname=winnt" --label=EFI

volgroup system_vg     --pesize=4096 pv.166
logvol   /             --fstype="xfs"    --size=102400  --name=root_lv        --vgname=system_vg
logvol   /usr          --fstype="xfs"    --size=51200   --name=usr_lv         --vgname=system_vg
logvol   /opt          --fstype="xfs"    --size=51200   --name=opt_lv         --vgname=system_vg
logvol   /home         --fstype="xfs"    --size=51200   --name=home_lv        --vgname=system_vg
logvol   /tmp          --fstype="xfs"    --size=102400  --name=tmp_lv         --vgname=system_vg
logvol   /var          --fstype="xfs"    --size=102400  --name=var_lv         --vgname=system_vg
logvol   /cinecalocal  --fstype="xfs"    --size=10240   --name=cinecalocal_lv --vgname=system_vg
logvol   swap          --fstype="swap"   --size=4096    --name=swap_lv        --vgname=system_vg
```

Figura 3.6: Configarazione *partitionfile*

3.3.3 Definizione dei nodi

Le definizioni delle componenti dei nodi puó essere fatta in due modi:

- per ogni singolo nodo;
- tramite la definizione di gruppi.

Per ogni singolo nodo le parti fondamentali cui prestare particolare attenzione sono:

ipmi : la configurazione degli indirizzi e delle credenziali delle BMC dei nodi é fondamentale per potersi collegare durante la fase di installazione e per controllare la gestione dell'energia (riavvio, spegnimento, accensione);

nic : In questa sezione vengono definite le caratteristiche delle interfacce di rete;

network : definisce le reti del sistema, fornisce informazioni riguardo alla subnet, al gateway e in caso si utilizzi l'assegnamento dinamico degli indirizzi con DHCP, anche il range dove scegliere il valore;

prescripts, postscripts e postbootscripts : questi file fanno parte della distribuzione di xCAT e permettono l'esecuzione di operazioni fondamentali. Ad esempio le operazioni di prescript vengono eseguite prima che l'installazione abbia inizio, postscript subito dopo la fine dell'installazione e postbootscript subito dopo il primo riavvio. Tra gli script fondamentali troviamo quello che implementa le configurazioni delle interfacce di rete, quello che installata i pacchetti indicati nelle definizioni delle osimage e lo script che crea gli utenti. É possibile anche creare script personali;

hosts : definisce l'hostname della macchina, fondamentale per la creazione delle informazioni del DNS;

osimage : assegna l'osimage di installazione ai nodi.

```
"m511n05","eno1!|m511n0(\d+).*$|10.29.1.($1)|,bond0!|m511n0(\d+).*$|10.47.19.($1+67)|,bond0.585!|m511n0(\d
+).*$|10.47.20.($1+3)|,bond0.583!|m511n0(\d+).*$|10.47.19.($1+3)|,bond0.582!|m511n0(\d+).*$|10.47.18.($1+
3)|",,,"eno1!ethernet,bond0!bond,ens2f0!ethernet,enp217s0f0!ethernet,bond0.585!vlan,bond0.583!vlan,bond0.5
82!vlan",,"eno1!HPC-IG-CLUSTER,bond0!HPC-IG-INT1,bond0.585!HPC-IG-ST,bond0.583!HPC-IG-ST1,bond0.582!HPC-I
G-INT",,"bond0!BONDING_OPTS=lacp_rate=1;miimon=100;mode=802.3ad;xmit_hash_policy=layer3+4 GATEWAY=10.47.19
.67,bond0.585!VLAN=yes GATEWAY=10.47.20.3,bond0.583!VLAN=yes GATEWAY=10.47.19.3,bond0.582!VLAN=yes GATEWA
Y=10.47.18.3","bond0!ens2f0|enp217s0f0,bond0.585!bond0,bond0.583!bond0,bond0.582!bond0",,,
```

Figura 3.7: Configurazione **nics** del nodo **m511n05**

3.3.4 Sincronizzazione dei nodi

Per la sincronizzazione e la distribuzione dei file sui nodi si utilizza uno strumento sviluppato in CINECA. Lo strumento in questione si chiama **NodeTools**. Nodetools é un software che fornisce strumenti di gestione dei nodi di un cluster. Nato principalmente per la sincronizzazione di file mette a disposizione anche librerie che possono essere incluse in script aggiuntivi. Tramite l'utilizzo delle informazioni definite con xCAT, in modo particolare i nomi dei nodi e i gruppi, NodeTools permette, tramite una specifica sintassi, di specificare il nome e il percorso dei file da distribuire e a chi distribuirli.

```
================================================================================
source
================================================================================
+TAG=cont_policy/etc/containers/policy.json+@kubernetes
+TAG=cont_policy/etc/containers/registries.conf+@kubernetes
+TAG=haproxy/etc/haproxy/haproxy.cfg+@lb_node
+TAG=hosts/etc/hosts+@all
+TAG=keepalived/etc/keepalived/keepalived.conf+@m511n07
+TAG=keepalived/etc/keepalived/keepalived.conf+@m511n08
+TAG=rsyslog/etc/sysconfig/rsyslog+@kubernetes
+TAG=sysctl/etc/sysctl.d/99-sysctl.conf+@kubernetes
+TAG=kubernetes/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf+@m511n01
+TAG=kubernetes/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf+@m511n02
+TAG=kubernetes/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf+@m511n03
+TAG=kubernetes/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf+@m511n04
+TAG=kubernetes/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf+@m511n06
+TAG=kubernetes/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf+@m511n05
================================================================================
```

Figura 3.8: Lista file da sincronizzare con la sintassi di NodeTools

3.4 Installazione Kubernetes e configurazioni nodi load-balancer

Per logica i primi nodi da configurare saranno quelli adibiti al load-balancing delle richieste. Una volta funzionanti si puó prodere con l'installazione del cluster Kubernetes.

3.4.1 Nodi load-balancer

I nodi, come da specifiche dell'installazione xCAT, sono giá dotati dei software HA-Proxy e Keepalived. Quello che ora si deve fare é fornire un IP virtuale in grado di cambiare ospite in caso di malfunzionamenti (Keepalived) e un sistema in grado di rimanere in ascolto su un determinato indirizzo e una determinata porta e inoltrare il traffico alle macchine realmente in grado di fornire una risposta.

Keepalived

Nella prima sezione in figura 3.9 viene configurato lo script di controllo dello stato del processo. Il corretto funzionamento del VIP implementato si basa sulla presenza del processo controllato, se questo non é presente l'ip fornito diventa inutile. Se questo succede avvia le procedure di migrazione verso l'altra macchina.

```
vrrp_script chk_haproxy {
  script "sudo /usr/bin/killall -0 haproxy"
  interval 3
  weight -2
  fall 10
  rise 2
}

vrrp_instance haproxy-vip {
  state MASTER ###BACKUP
  priority 101 ###100
  interface bond0.582                       # Network card
  virtual_router_id 60
  advert_int 1
  authentication {
    auth_type PASS
    auth_pass
  }
  unicast_src_ip 10.47.18.10 ###11      # The IP address of this machine
  unicast_peer {
    10.47.18.11 ###10                       # The IP address of peer machines
  }

  virtual_ipaddress {
    10.47.18.250                 # The VIP address
  }

  track_script {
    chk_haproxy
  }
}
```

Figura 3.9: Configurazione Keepalived

L'intera istanza del VIP viene descritta nella sezione successiva. Nell'immagine 3.9 é riportata la configurazione del nodo che é stato scelto per essere il principale. Lo si nota dalla voce **state** settata con il valore *MASTER*. Nella sezione **virtual_ipaddress** viene indicato il valore effettivo dell'indirizzo virtuale che si vuole utilizzare. Le altre voci indicano le informazioni di controllo, come lo script da utilizzare e gli indirizzi IP delle macchine che vengono incluse in questa configurazione.

Finita la configurazione possiamo abilitare all'avvio il processo e attivarlo. Se si controlla qualche secondo dopo si noterá che sull'interfaccia scelta verrá indicato un'indirizzo IP secondario, lo stesso che si é scelto nella configurazione.

```
11: bond0.582@bond0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9216 qdisc noqueue state UP
    inet 10.47.18.10/24 brd 10.47.18.255 scope global noprefixroute bond0.582
    inet 10.47.18.250/24 scope global secondary bond0.582
```

Figura 3.10: Virtual IP configurato sull'interfaccia

HAProxy

Nella configurazione di HAProxy si devono definire sostanzialemente due parti. La parte frontend, che specifica cosa deve aspettare la macchina ospite. In questo caso si é specificato che dovrá aspettarsi delle richieste in arrivo sulla porta **6443** e su una qualsiasi intefaccia (in una configurazione futura dove potrebbero essere configurati piú VIP e servizi, si indicherá in modo preciso l'indirizzo su cui le richieste dovrebbero arrivare). Le richieste in arrivo dalle macchine che vogliono interagire con il cluster Kubernetes utilizzano il protocollo HTTP (livello 7 modello ISO/OSI). Con HAProxy sarebbe possibile dire di attendere richieste di quel tipo, ma si é deciso di essere piú generici, indicando TCP (livello 4 ISO/OSI) come protocollo.

```
#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------
frontend kube-apiserver
    bind *:6443
    mode tcp
    option tcplog
    default_backend kube-apiserver

#---------------------------------------------------------------------
# round robin balancing between the various backends
#---------------------------------------------------------------------
backend kube-apiserver
    mode tcp
    option tcplog
    option tcp-check
    balance     roundrobin
    default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250 maxqueue 256 weight 100
    server  kube-master1 10.47.18.4:6443 check
    server  kube-master2 10.47.18.5:6443 check
    server  kube-master3 10.47.18.6:6443 check
```

Figura 3.11: Configurazione HAProxy

La parte di backend definisce, una volta ricevuto un messaggio, come si deve comportare nell'inoltro. Anche in questocaso viene scelto il tipo di protocollo. Si deve poi indicare come si vuole che il sistema agisca nella scelta del server a cui inoltrare la richiesta, indicando il tipo di algoritmo. Non essendoci particolari esigenze, si é scelto di mantenere l'algoritmo roundrobin. Infine si devono indicare gli endpoint disponibili per l'inoltro. Nel caso riportato sono presenti i tre server Control Plane di Kubernetes.

3.4.2 Cluster Kubernetes

Sono stati svolti diversi tentativi prima di arrivare ad un'installazione che rispettasse le specifiche che si avevano in mente per l'implementazione del cluster.

Come detto, in fase di progettazione si é deciso di procedere con l'installazione del cluster tramite l'utilizzo del tool kubeadm. Funzionalitá utile di questo tool é che permette di tornare ad uno stato "pre-installazione" solo eseguendo il comando **kubeadm reset**. L'unico svantaggio é che non funziona per tutti i servizi esterni. In sintesi, se si ha un'installazione completa, con plugin e software installati successivamente, questi dovranno essere rimossi manualmente prima di poter eliminare il cluster. L'utilizzo di questo strumento quindi ne semplifica l'operazione, ma per installazioni complesse viene richiesta attenzione per rimuovere a mano le configurazioni esterne. Per questo motivo si é dovuto procedere alla reinstallazione completa dei nodi e ripartire da uno stato iniziale.

Prima di inizializzare il cluster con l'utilizzo di **kubeadm** bisogna assicurarsi che i nodi abbiano attivi:

- net.ipv4.ip_nonlocal_bind = 1
- net.ipv4.ip_forward = 1
- E abbiano il modulo kernel br_netfilter attivo

Se non si danno indicazioni a **kubeadm**, durante l'installazione, sceglie in modo autonomo la rete su cui verranno configurati i nodi Kubernetes. La rete scelta corisponderá alla rete principale del sistema. Con kubeadm questa impostazione non é configurabile direttamente, ma é necessario indicare alla componente **kubelet** la rete che si intende utilizzare.

Le reti dei POD e dei servizi se non indicate in fase di inizializzazione vengono scelte tra quelle di default e sono:

- 10.85.0.0/16 per la rete dei POD
- 10.96.0.0/12 per la rete dei servizi

Non avendo necessitá particolari, si é deciso di mantenere le reti di default.

Il comando utilizzato per l'installazione é il seguente:

```
#Comando di inizializzazione del cluster Kubernetes
```

```
$ kubeadm init --apiserver-advertise-address "10.47.18.4" \
  --Control Plane-endpoint "10.47.18.250:6443" \
  --upload-certs
```

Al termine dell'installazione, vengono forniti i comandi di join, i quali permettono di aggiungere nodi Control Plane o nodi worker al cluster.

```
#Comando di join per l'aggiunta di un nuovo Control Plane

$ kubeadm join 10.47.18.250:6443 \
  --token xxx \
  --discovery-token-ca-cert-hash sha256:xxx \
  --Control Plane \
  --certificate-key xxx \
  --apiserver-advertise-address "10.47.18.5"

#Comando di join per l'aggiunta di un worker node

$ kubeadm join 10.47.18.250:6443 \
--token xxx \
--discovery-token-ca-cert-hash sha256:xxx
```

3.4.3 Plugin di rete

Prima di installare il plugin di rete si sono stati realizzati alcuni tentativi per capire per quale motivo fosse necessario l'utilizzo di uno strumento esterno. L'interfaccia CNI sui nodi viene fornita dal container runtime, quindi tecnicamente la rete dovrebbe funzionare anche senza il supporto di un plugin.

Si é proceduto con l'installazione di un semplice software, ovvero la dashboard Kubernetes. L'installazione prevede la creazione di varie componenti tra cui il servizio che ne permette l'accesso. Questo servizio sarebbe di default configurato per essere di tipo **ClusterIP** e quindi essere accessibili solo all'interno della rete del cluster. É stata quindi modificata la definizione del servizio in modo da ottenerne uno di tipo **NodePort**, cosí da poter eseguire l'accesso anche dall'esterno.

A seguito di diverse prove si é notato che procedendo senza plugin il servizio era si acccessibile, ma solo utilizzando l'indirizzo del nodo sul quale il POD stava eseguendo.

Stando alla definizione di servizio NodePort di Kubernetes, tramite le funzioni offerte dal proxy, i servizi dovrebbero essere accessibile utilizzando l'indirizzo di qualsiasi nodo del cluster.

L'installazione del plugin di rete deve essere effettuata prima dell'aggiunta dei nodi worker. La procedura non é coplessa. Per un imlementazione basilare basta eseguire gli strumenti forniti sul sito ufficiale del plugin. L'unico accorgimento é che prima di completare l'installazione bisogna indicare nei file, quali sono le reti scelte per i POD e i servizi. Se si é scelto di mantenere i valori di default si devono indicare quelli.

```
#Procedura di installazione CALICO #Per prima cosa bisogna installare l'operatore

    $ kubectl create -f tigera-operator.yaml

#Scaricare le CustomResource necessarie per configurare CALICO #Scaricato il file
modificare i valori che si vogliono utilizzare

    $ cat custom-resources.yaml

    # This section includes base Calico
    #installation configuration.
    apiVersion: operator.tigera.io/v1
    kind: Installation
    metadata:
    name: default
    spec:
    # Configures Calico networking.
    calicoNetwork:
        # Note: The ipPools section cannot
        # be modified post-install.
        ipPools:
        - blockSize: 26
        cidr: 10.85.0.0/16
        encapsulation: VXLANCrossSubnet
        natOutgoing: Enabled
        nodeSelector: all()

    ---
```

```
    # This section configures the Calico API server.
    apiVersion: operator.tigera.io/v1
    kind: APIServer
    metadata:
    name: default
    spec: {}


#Applicare la configurazione cosí da installare CALICO

    $ kubectl create -f custom-resources.yaml
```

3.4.4 Storage distribuito - Rook+CephFS

Prima di installare le componenti necesarie ad installare il sistema Rook+Ceph, bisogna assicurasi di che i requisiti siano rispettati.

- Una versione di Kubernetes uguale o maggiore alla v1.21
- Un sistema amd64/x86_64 o arm64
- I dispositivi di storage devono appartenere ad una delle seguenti specifiche
  - Dispositivi senza partizioni e formattati
  - Con partizioni senza filesystem
  - Volumi logici LVM
  - Volumi persistenti disponibili da una classe di archiviazione in modalitá *block*
- Il kernel deve possedere il modulo RBD

Per prima cosa bisogna clonare la repository contenente i file YAML contenenti le regole di installazione degli oggetti che andranno a comporre il sistema.

Applicando le configurazioni giá presenti, il sistema sarebbe giá funzionante. Tutte le operazioni di scelta dei dischi da utilizzare, la creazione dei demoni di monitoraggio sono giá definite e configurate. Volendo peró creare un'installazione "custom", si andranno a modificare i parametri che definiscono i nodi su cui costruire il sistema, i dischi da utilizzare per creare il filesystem, il disco per l'archiviazione dei metadati e si andranno

ad attivare tutte le voci che permetteranno il funzionamento del monitoraggio. Anche CephFS ha una sua dashboard e verrá installata automaticamente durante il deploy del servizio.

Modifiche per attivare il monitoraggio del filesystem

```
#Abilitare la componente dashboard all'interno del file cluster.yaml

    $ vim cluster.yaml

    [...]
    spec:
    dashboard:
        enabled: true


#Permettere alla dashboard di ottenere informazioni relative ai dischi fisici

    [...]
    mgr:
    modules:
      - name: rook
        enabled: true

#Permettere alla dashboard di ottenere informazioni relative ai dischi fisici

    $ vim operator.yaml

    [...]
    data:
    ROOK_ENABLE_DISCOVERY_DAEMON: true
    ROOK_DISCOVER_DEVICES_INTERVAL: 60

#Creazione di un servizio per esporre verso l'esterno la dashboard per permettere
l'accesso dai nodi

    $ vim dashboard-external-https.yaml

    apiVersion: v1
```

```
kind: Service
metadata:
  name: rook-ceph-mgr-dashboard-external-https
  namespace: rook-ceph
labels:
  app: rook-ceph-mgr
  rook_cluster: rook-ceph
spec:
    ports:
    - name: dashboard
      port: 8443
      protocol: TCP
      targetPort: 8443
    selector:
      app: rook-ceph-mgr
      rook_cluster: rook-ceph
    sessionAffinity: None
    type: NodePort
```

Configurazione nodi e dischi del cluster

#Indicare l'IP dei nodi che su cui installare il sistema e indicare quali dischi al loro interno si vogliono utilizzare
#Considerando che si specificano i nodi e i dischi da utilizzare, assegnare alle voci *useAllNodes* e *useAllDevices* a false
#Il dischi che vogliono essere utilizzati come dispositivi di archiviazione dei metadati é consigliato che non siano dischi rotativi ma dischi allo stato solido

```
$ vim cluster.yaml

[...]
storage: # cluster level storage configuration and selection
useAllNodes: false
useAllDevices: false
config:
```

```
      metadataDevice: "sdb"
    nodes:
    - name: "10.47.18.7"
      devices:
      - name: "sdd"
      - name: "sde"
      - name: "sdf"
      - name: "sdg"
      - name: "sdh"
    - name: "10.47.18.8"
      devices:
      - name: "sdd"
      - name: "sde"
      - name: "sdf"
      - name: "sdg"
      - name: "sdh"
    - name: "10.47.18.9"
      devices:
      - name: "sdd"
      - name: "sde"
      - name: "sdf"
      - name: "sdg"
      - name: "sdh"
```

Installazione Rook+CephFS

Configurati i file che descrivono le specifiche del sistema, si puó procedere con l'installazione.

```
#Tutti i file di installazione si trovano all'interno della cartella rook/deploy/examples

    $ kubectl create -f crds.yaml -f common.yaml -f operator.yaml

#verificare che l'operatore rook-ceph sia nello stato "Running" prima di procedere

    $ kubectl -n rook-ceph get pod
```

```
#Ora che l'operatore Rook é in esecuzione, possiamo creare il cluster Ceph

    kubectl create -f cluster.yaml
```

Elemento essenziale per il controllo, il debugging e la gestione del filesystem é l'installazione del **Rook Toolbox**. Questo elemento inizializza un POD che mette a disposizione tutti i comandi necessari per gestire un sistema CephFS.

```
#Installare il deploy della componente Rook Toolbox

    $ kubectl create -f toolbox.yaml

#Una volta installata, accedere all'iterno del container

    $ kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash

#Verificare lo stato dell'installazione

    $ ceph status
```

```
[root@master01 deploy]# kubectl -n rook-ceph exec -it  deploy/rook-ceph-tools -- bash
bash-4.4$ ceph status
  cluster:
    id:     f8c79348-467d-4976-b40e-9c3f168b5b49
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum a,b,c (age 2d)
    mgr: b(active, since 8h), standbys: a
    mds: 1/1 daemons up, 1 hot standby
    osd: 15 osds: 15 up (since 2d), 15 in (since 2d)

  data:
    volumes: 1/1 healthy
    pools:   3 pools, 65 pgs
    objects: 27 objects, 1.6 MiB
    usage:   3.3 TiB used, 164 TiB / 167 TiB avail
    pgs:     65 active+clean

  io:
    client:   853 B/s rd, 1 op/s rd, 0 op/s wr
```

Figura 3.12: Schermata stato del sistema

Creazione filesystem distribuito ed esempio di utilizzo del filesystem

Se il sistema non presenta errori e tutto funziona si puó procedere con la creazione del filesystem distribuito. Un filesystem distribuito puó essere montato ed utilizzato per operazioni di lettura/scrittura da piú POD.

In questo caso, giusto per fare qualche test, sono stati utilizzati i file di configurazione forniti, senza cambiare nessuna impostazione. Il file di installazione del filesystem di default crea due pool, una per i metadati e una per i dati, ognuna replicata tre volte.

```
#Decisa la configurazione, inizializzare il filesystem

    $ kubectl create -f filesystem.yaml

#Per confermare che il filesystem é configurato, verificare lo stato dei POD mds:

    $ kubectl get pod -n rook-ceph | grep mds

    rook-ceph-mds-myfs-a-d88bc665-lw2nf      2/2  Running
    rook-ceph-mds-myfs-b-66775dbb86-zklwz    2/2  Running
```

Prima che Rook possa iniziare il provisioning dell'archiviazione, é necessario creare una StorageClass basata sul filesystem. Ció é necessario affinché Kubernetes interagisca con il driver CSI per creare volumi permanenti.

```
    $ kubectl create -f storageclass.yaml
```

Per eseguire alcune prove e vedere se il filesystem creato sia funzionante, sono stati creati un PersistentVolumeClaim, per richiedere un determinato spazio al filesystem e un Deployment che usa e monta il volume creato.

```
    $ kubectl create namspace prova
    $ vim prova.yaml

    apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
    name: cephfs-pvc
    namespace: prova
spec:
    accessModes:
    - ReadWriteMany
    resources:
    requests:
    storage: 10Gi
    storageClassName: rook-cephfs
---
apiVersion: apps/v1
kind: Deployment
metadata:
    [...]
spec:
  [...]
  containers:
  [...]
    volumeMounts:
      - name: image-store
        mountPath: /prova
  volumes:
  - name: image-store
    persistentVolumeClaim:
    claimName: cephfs-pvc
    readOnly: false

$ kubectl create -f prova.yaml
```

Per verificare il corretto funzionamento dello storage, una volta all'interno del container di prova, si sono elaborati diversi file nella cartella scelta come mount point del filesystem. A seguito di cancellazioni e riavvii del pod, ogni volta schedulato ed eseguito su un diverso nodo, i file creati in precedenza sono sempre rimasti disponibili.

3.4.5 Software di monitoraggio

Si avranno in tutto due dashboard di monitoraggio del sistema.

- Kubernetes Dashboard
- Prometheus e Grafana

Kubernetes Dashboard

L'installazione della dashboard di Kubernetes, essendo uno strumento proprietario é pittosto semplice e non richiede modifiche alle configurazioni. L'unica modifica che é stata applicata, é stata cambiare il tipo di servizio, il quale definisce il metodo di accesso alla dashboard. Di base il tipo di servizio esposto é di tipo *ClusterIP*, quindi accessibile sono all'interno del cluster. Perché sia possibile accedere anche dall'esterno é stato cambiato con un servizio di tipo *NodePort*

```
#Per prima cosa bisogna eseguire il download del file di installazione della dashboard
#Dopo aver scaricato il file, applicare le modifiche al tipo di servizio

    $ vim dashboard.yaml

    kind: Service
    apiVersion: v1
    metadata:
    labels:
        k8s-app: dashboard-metrics-scraper
    name: dashboard-metrics-scraper
    namespace: Kubernetes-dashboard
    spec:
    ports:
        - nodePort: 30123 #porta scelta tra i valori permessi. Se non
          port: 8000      #idicata, questa viene scelta casualmente
          targetPort: 8000
    selector:
        k8s-app: dashboard-metrics-scraper
    type: NodePort

#Ora si puó installare la componente
```

```
$ kubectl apply -f dashboard.yaml
```

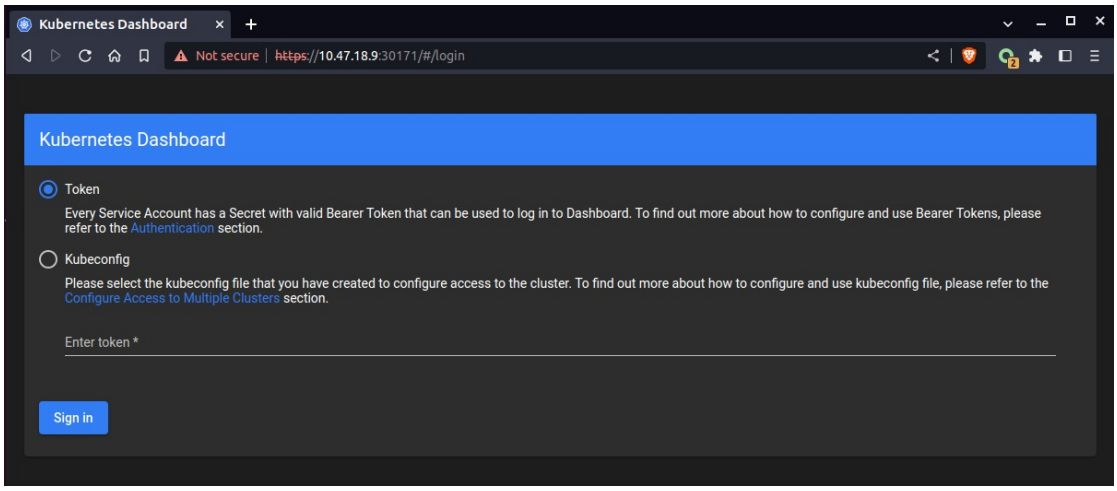

Figura 3.13: Pagina di accesso Dashboard Kubernetes

Per proteggere i dati del cluster, la Dashboard viene distribuito con una configurazione RBAC minima per impostazione predefinita. Attualmente, Dashboard supporta solo l'accesso con un Bearer Token.

Per poter accedere é stato quindi creato un semplice utente con i permessi necessari.

```
#Come prima cosa bisogna creare un ServiceAccount nel namespace dove é in esecuzione la dashboard

    apiVersion: v1
    kind: ServiceAccount
    metadata:
        name: admin-user
        namespace: Kubernetes-dashboard

#Si crea un ClusterRoleBinding per associare all'acount i permessi necessari

    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRoleBinding
    metadata:
```

```
        name: admin-user
    roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: cluster-admin
    subjects:
    - kind: ServiceAccount
        name: admin-user
        namespace: Kubernetes-dashboard

#Infine generiamo il token

    $ kubectl -n Kubernetes-dashboard create token admin-user

    eyJhbGciOiJSUzI1NiIsImtpZCI6IlQxNUkyOVJFSWJPRUZZRDdQWndtQjA
    1ek9fczJGYzRHYmZuZTIzLTJwckUifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJ
    lcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNj
    gyODYxMDAyLCJpYXQiOjE2ODI4NTc0MDIsImlzcyI6Imh0dHBzOi8va3ViZ
    cy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlcm5ldGVz...
```

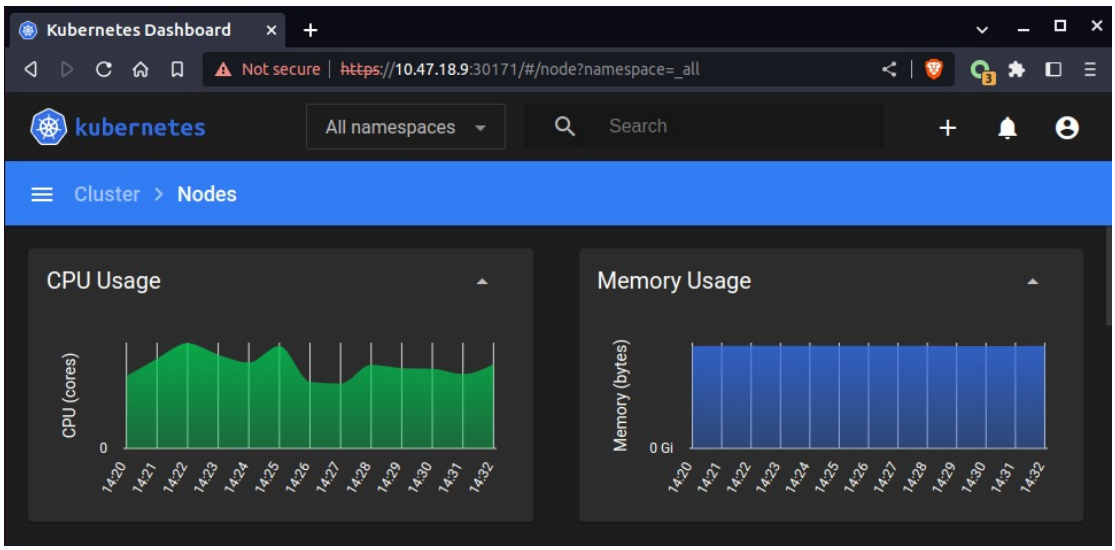

Figura 3.14: Overview consumo risorse dei nodi

Prometheus e Grafana

Per l'installazione di Prometheus e Grafana é stata utilizzata una distribuzione di prometheus-operator, la quale raccoglie file manifest Kubernetes, la dashboard Grafana e regole Prometheus combinate con script per fornire un monitoraggio del cluster Kubernetes end-to-end utilizzando l'operatore Prometheus.

Nella sua configurazione base, é possibile monitorare lo stato dei nodi e tutte le risorse appartenenti al cluster Kubernetes.

Come per la dashboard, anche in questo caso é stato necessario modificare gli oggetti che espongono i servizi, configurati come *ClusterIP*. Sono stati modificati in servizi di tipo *NodePort* in modo che si riesca ad accedere anche dall'esterno.

```
#Clonare la reposiory git contenente i file necessari all'installazione #Creare il namespace e le CustomResourceDefinitions richieste

    $ kubectl create -f manifests/setup

#Una volta confermato che l'operatore Prometheus é in esecuzione, si puó procedere e distribuire lo stack di monitoraggio Prometheus

    $ kubectl create -f manifests/
```

Esistono regole di rete che impediscono l'accesso tramite servizi LoadBalancer o NodePort. I criteri di rete che limitano l'accesso ai componenti vengono aggiunti per impostazione predefinita e, per poter quindi accedere anche dall'esterno, é opportuno rimuoverli.

```
    $ kubectl -n monitoring delete \
      networkpolicies.networking.k8s.io --all

#Ora si puó procedere alle modifiche dei servizi

    $ kubectl --namespace monitoring patch svc \
      prometheus-k8s -p '{"spec": {"type": "NodePort"}}'
    $ kubectl --namespace monitoring patch svc \
      alertmanager-main -p '{"spec": {"type": "NodePort"}}'
    $ kubectl --namespace monitoring patch svc \
      grafana -p '{"spec": {"type": "NodePort"}}'
```

```
#Controllare che i servizi siano effettivamente diventati di tipo NodePort.

    $ kubectl -n monitoring get svc  | grep NodePort

    alertmanager-main NodePort 10.96.75.168   9093:31234,8080:32584
    grafana           NodePort 10.101.78.3    3000:32724
    prometheus-k8s    NodePort 10.103.249.195 9090:32693,8080:31978
```

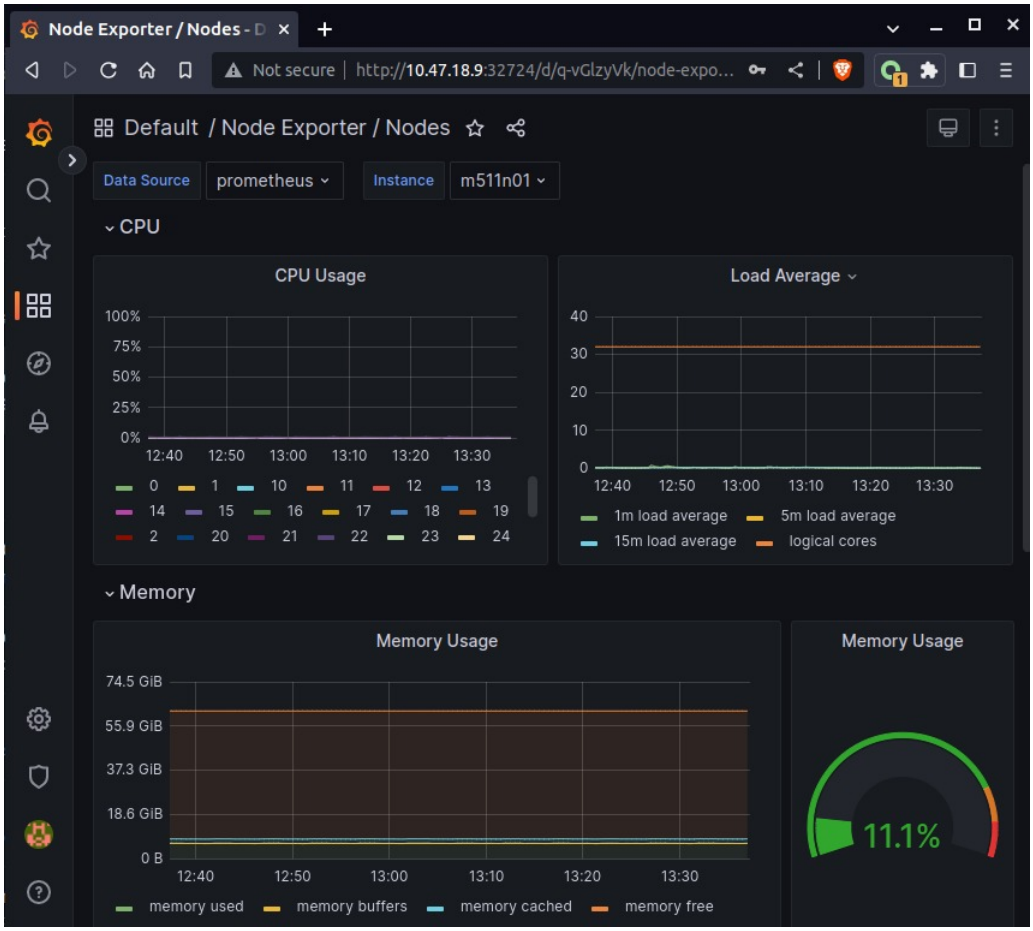

Figura 3.15: Dasboard di controllo del nodo m511n01

Capitolo 4

Test e considerazioni

***Obiettivi**: in questo capitolo si discute del peso che ha un'installazione Kubernetes e l'overhead che introduce sul sistema. Si espone poi qualche considerazione riguardante la fattibilitá e il costo in termini di tempo e risorse che richiede questo tipo di approccio.*

- *Cluster Kubernetes*
- *Consumo delle risorse*
- *Fattibilitá*

4.1 Cluster Kubernetes

Al termine delle prime fasi di progettazione, la configurazione del rack che ospita i nodi del cluster risulta come si vede in figura 4.1.

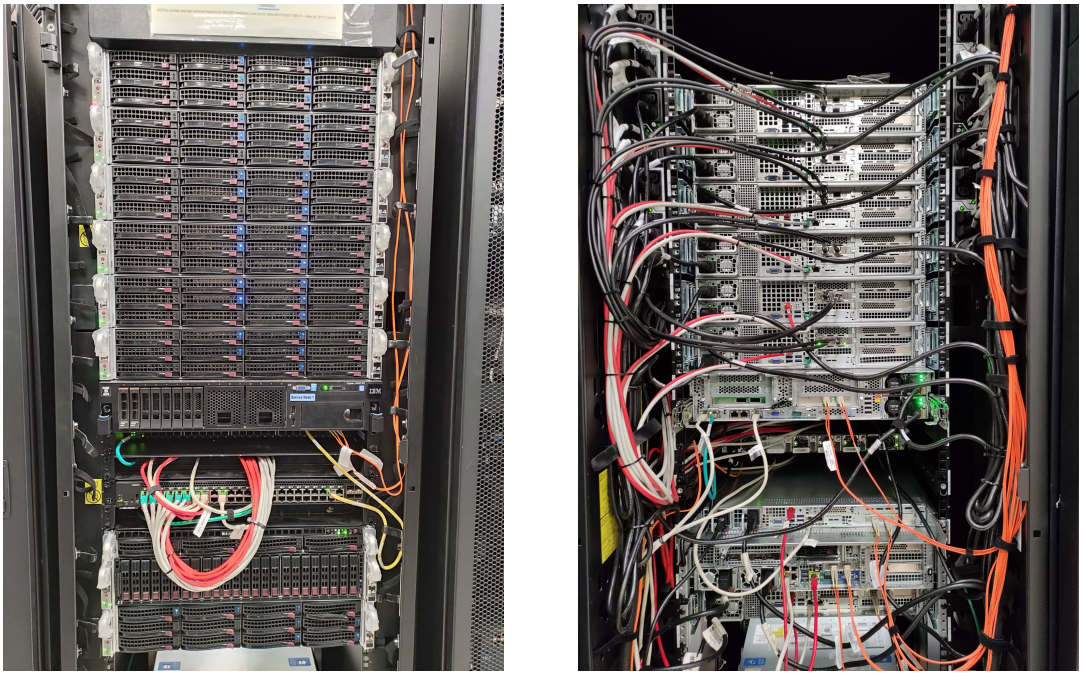

Figura 4.1: Rack server HPC-IG

L'installazione del sistema finale presenta una forma compatibile con l'architettura ad alta disponibilitá con il database **etcd** stacked, ovvero ospitato internamente ai nodi Control Plane e non distribuito esternamente 4.2. Si hanno tre nodi Control Plane sui quali eseguono solamente gli strumenti necessari a gestire la rete e il sistema Kubernetes, unica eccezione fatta per i Node Exporter, installati per permettere la raccolta delle metriche dei nodi utili a Prometheus per il monitoraggio del sistema. Responsabili all'esecuzione degli strumenti realmente utili alla gestione del cluster di produzione sono invece i nodi worker. Nella soluzione proposta sono stati installati solo alcuni degli strumenti necessari alla gestione e al monitoraggio di un cluster.

4.2 Consumo delle risorse

L'installazione di un sistema gestito con Kubernetes richiede risorse aggiuntive rispetto ad un'installazione bare-metal. Questo introduce un certo grado di overhead nel

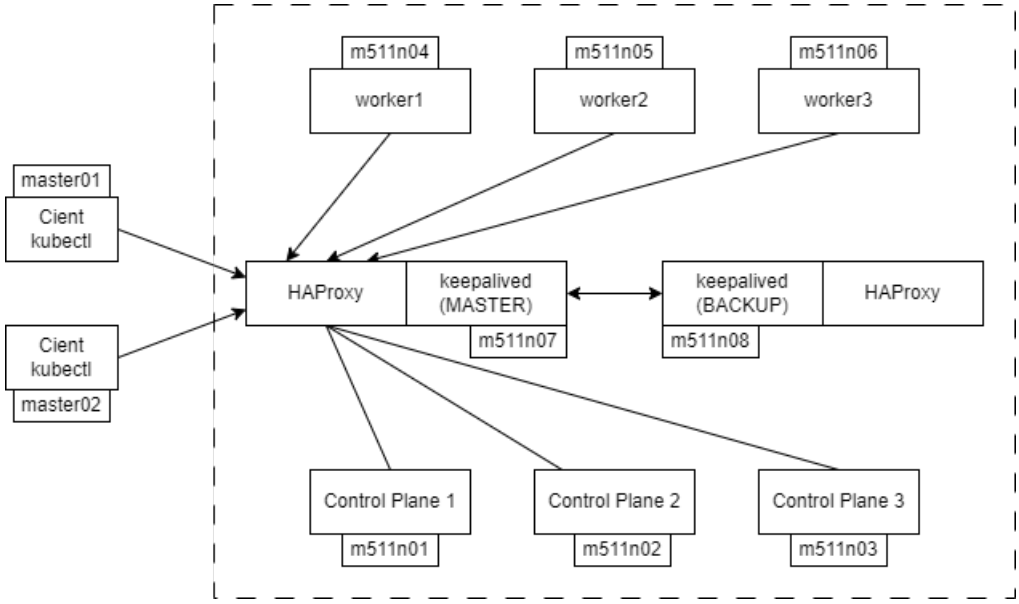

Figura 4.2: Architettura finale cluster Kubernetes

sistema. Le risorse aggiuntive richieste riguardano soprattutto all'allocazione di CPU, RAM e storage per l'esecuzione dei container che ospitano le applicazioni e l'esecuzione di tutti gli strumenti che servono a Kubernetes per gestirsi e funzionare. L'overhead cresce in base alla dimensione del cluster e al carico di lavoro che deve gestire.

Con la configurazione attuale, ogni nodo Control Plane esegue dai 9 agli 11 pod e ogni nodo worker esegue 21 pod. L'intero sistema Kubernetes, come si puó vedere in figura 4.3, per l'esecuzione di tutti i software installati richiede circa 13.4 GiB di memoria. In questo grafico sono peró esclusi servizi come la kubelet e il container runtime.

I processi che eseguono la kubelet e il container runtime (CRI-O), necessarie per la gestione delle risorse utilizzate dai container e della loro esecuzione, introducono ulteriore overhead. Ognuna di queste componenti é installata su ogni nodo e, in base al numero di oggetti che devono gestire, consumano maggiori risorse. Infatti sui nodi Control Plane, dove eseguono meno container, l'overhead introdotto sulla memoria é di circa 250 MiB per la kubelet e 250 MiB per il container runtime. Sui nodi worker invece, dove se ne eseguono un maggior numero, l'impatto é decisamente maggiore per il container runtime, che richiede circa 805 MiB di memoria per ogni nodo, e circa uguale per la kubelet che rimane su un consumo di circa 290 MiB.

Per quanto rigarda invece le risorse della CPU sull'intero sistema, l'impatto risulta minimo, avendo un consumo totale di circa 1.5%.

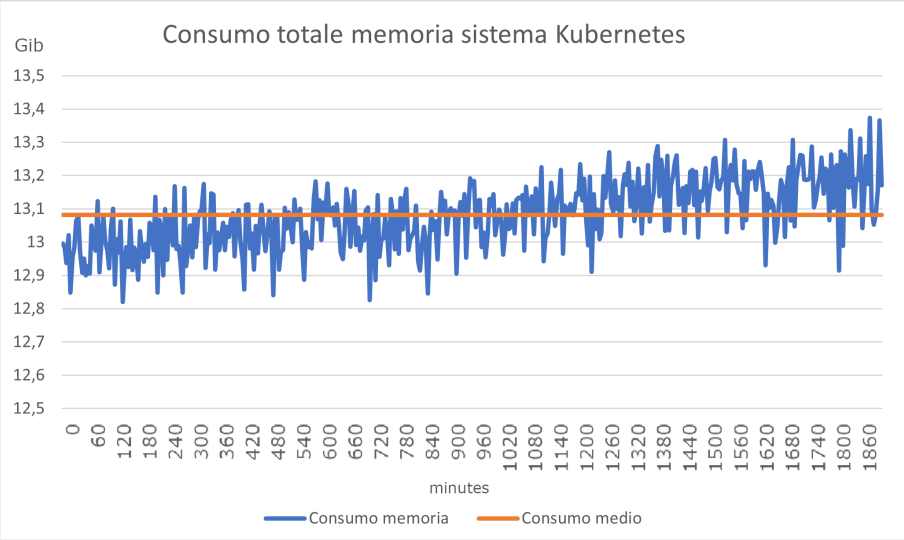

Figura 4.3: Consumo memoria cluster Kubernetes

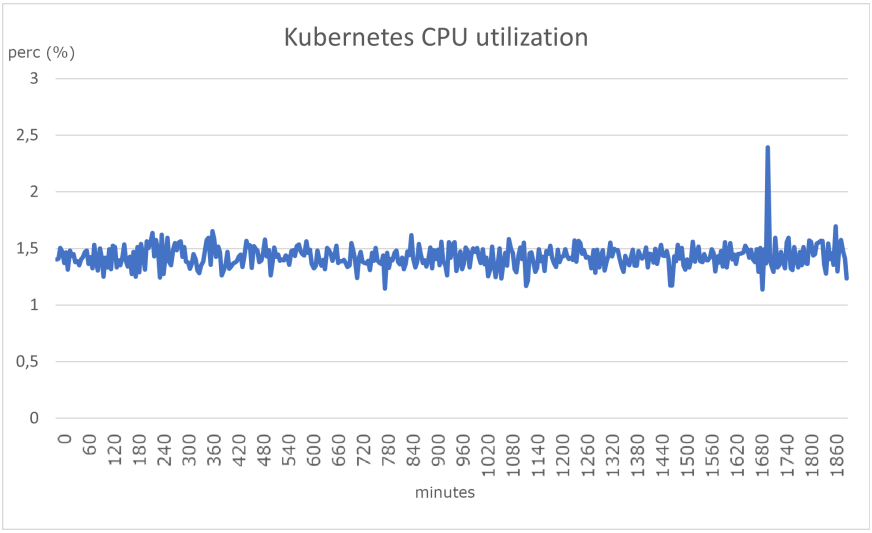

Figura 4.4: Consumo CPU cluster Kubernetes

La rete di Kubernetes é completamente virtuale, questa virtualizzazione introduce sicuramente dei peggioramenti nelle performance generali del sistema. Questo overhead é principalmente correlato al modello di rete, dalle operazioni fra i vari componenti per la gestione del cluster e al plug-in di rete utilizzato nel cluster. Tuttavia, per un sistema di gestione, le performance della rete sono secondarie, rispetto ad un cluster HPC in

produzione che richiede comunicazioni a bassa latenza durante l'esecuzione parallela dei carichi di lavoro.

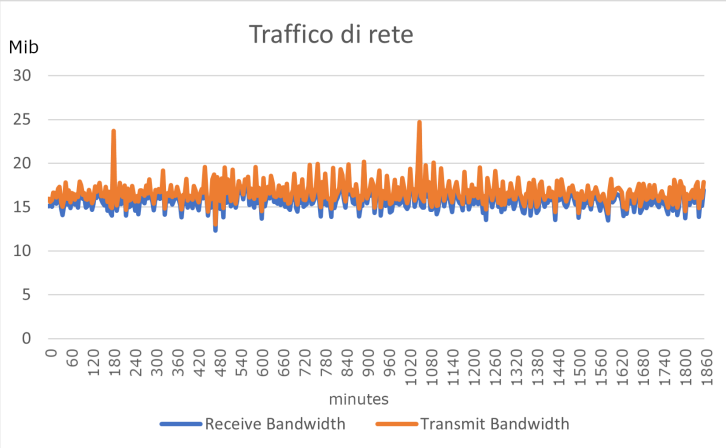

Figura 4.5: Consumo banda di rete in trasmissione e ricezione

La maggior parte del consumo si ha da parte dei software di monitoraggio e del sistema di storage distribuito. Per quanto riguarda invece il consumo di risorse da parte del plugin di rete e dei sistemi presenti nel sistema Kubernetes, il consumo é trascurabile, si parla di un consumo medio di 6 MB/s sia per la ricezione e di 7 MB/s per la trasmissione dei dati.

Ora peró si vogliono analizzare con maggiore dettaglio i pod che introducono il vero overhead nel sistema, ovvero la gestione della rete tramite il plugin CALICO e gli strumenti di sistema e i servizi centrali per il corretto funzionamento del sistema. Questo perché sono strumenti introdotti da Kubernetes e non sarebbero installati nel caso di un'installazione bare-metal.

4.2.1 kube-system

Come si puó vedere dall'immagine 4.6, l'impatto sull'intero sistema degli oggetti con il compito di gestire le risorse del cluster Kubernetes varia fra i 3.5 GiB e i 3.8 GiB con un'occupazione media di 3.67 GiB.

Se si osservano piú nel dettaglio le tipologie dei nodi, si puó notare che l'impatto maggiore degli oggetti di gestione del cluster é sui nodi di Control Plane, in quanto é qui che eseguono.

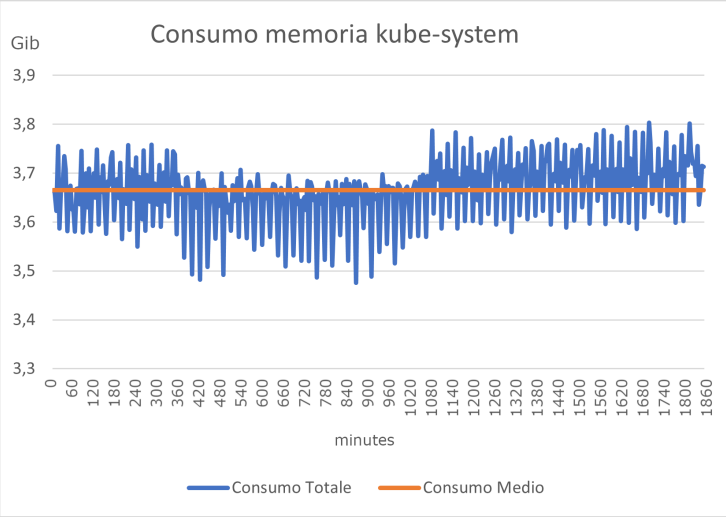

Figura 4.6: Consumo memoria del namspace kube-system

Infatti in figura 4.7, si puó vedere che su un consumo totale degli oggetti Kubernetes su un nodo Control Plane di 1.38 GB, il 72.6%, circa 1 GiB, é occupato dagli strumenti di gestione.

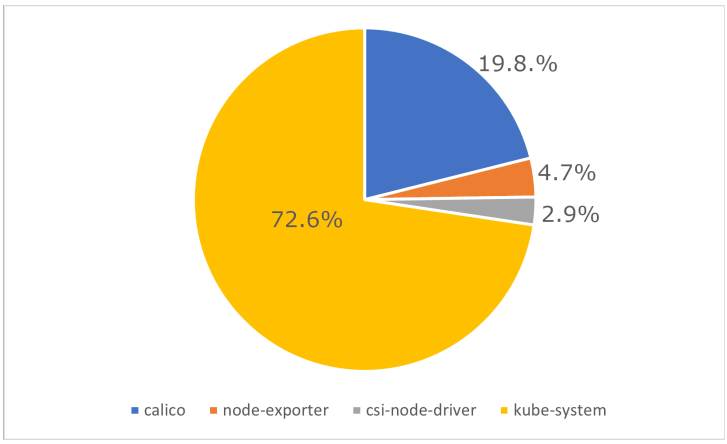

Figura 4.7: Consumo memoria del namspace kube-system rispetto al totale

L'overhead introdotto sulla memoria nodi worker invece risulta minimo. Infatti l'unica componente eseguita sui nodi worker appartenente al namespace kube-system é il kube-proxy che occupa solamente 40 MiB di memoria.

4.2.2 calico-system

Altro elemento che introduce overhead sulla memoria in un sistema Kubernetes é l'esecuzione del plugin per la gestione del firewall e della rete. Sull'intero sistema la gestione della rete consuma in media 2.42 GiB.

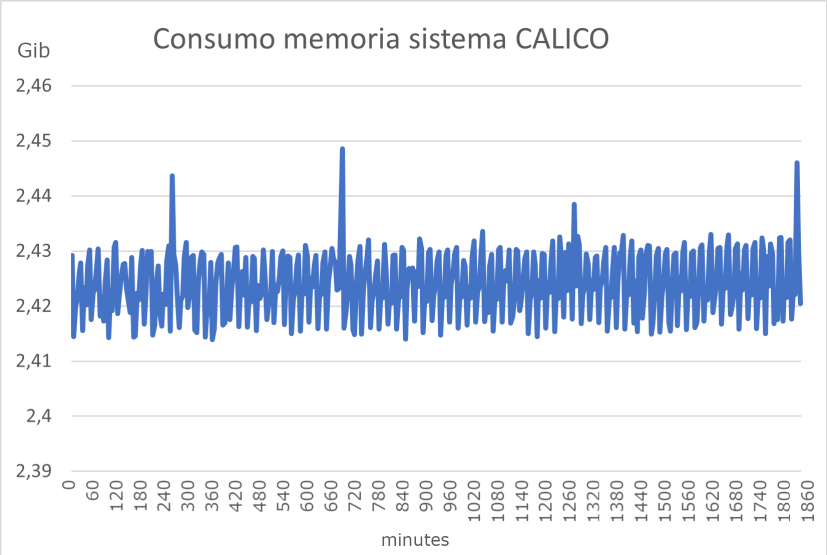

Figura 4.8: Consumo memoria del plugin di rete CALICO

Ognuno dei nodi esegue il plugin calico, che occupa mediamente 320 MiB di memoria. Sui nodi Control Plane peró sono esguiti gli operatori di calico, necessari per eseguire le modifiche sul sistema e per poter gestire la rete virtuale del cluster. In questo caso la richiesta di memoria é di 560MiB.

.

Le analisi fatte sul sistema sono da considerarsi per una situazione di IDLE, infatti il sistema non viene utilizzato attualmente per gestire un cluster di produzione.

In generale l'overhead introdotto da Kubernetes sui nodi del sistema, é di circa il 3% di memoria, su un totale di 64 GB e non si ha praticamente nessun impatto sulla CPU.

É importante notare che l'overhead di risorse introdotto da Kubernetes puó variare a seconda delle dimensioni del cluster, del carico di lavoro gestito e della configurazione specifica. Tuttavia, una pianificazione attenta delle risorse, l'ottimizzazione dei parametri di configurazione di Kubernetes e l'utilizzo di strumenti di monitoraggio possono

contribuire a mitigare l'impatto dell'overhead di risorse sulle prestazioni complessive del sistema.

4.3 Fattibilitá

Si é visto che creare un sistema di gestione di un cluster tramite l'implementazione di Kubernetes é fattibile. Sono peró da fare alcune osservazioni.

Il sistema proposto ha lo scopo di presentare una possibile soluzione ai problemi introdotti dall'utilizzo di questo tipo di approccio ed analizzare come in generale questo impatti sulle risorse del sistema. L'installazione di sistemi complessi tramite Kubernetes risulta semplice nel caso in cui non si abbiano esigenze particolari.

Se si approfondisce peró la questione, si puó notare come molte delle aziende, come CINECA debbano fare fronte ad alcune problematiche di integrazione dei sistemi giá presenti. Infatti, essendo fino ad ora l'approccio "bare-metal" quello principale, risulta oneroso a livello di tempo e di competenze, adattare i sistemi giá presenti all'utilizzo tramite l'implementazione con Kubernetes. Si richiede infatti non solo una profonda conoscenza di Kubernetes e di tutti i suoi funzionamenti, ma richiede anche una profonda conoscenza del software che si sta implementando. Infatti, la configurazione di tutti i software, non puó essere fatta runtime, ma deve essere studiata ed implementata preventivamente. Questo implica che l'amministratore di sistema incaricato a gestire il sistema conosca ogni elemento che sta eseguendo sul sistema.

In generale quindi Kubernetes introduce inizialmente complessitá, sia per quanto riguarda la curva di apprendimento sia per le fasi di progettazione del sistema.

C'é da considerare inoltre l'introduzione di overhead sulle risorse del sistema. Se con l'utilizzo di macchine con prestazioni elevate questo consumo non crea nesun disagio, per delle realtá minori, questo problema potrebbe tradursi in spese maggiori per l'equipaggiamento hardware. Per ridurre l'impatto della maggiore richiesta di risorse, é necessario definire correttamente le richieste e i limiti delle risorse per ogni container, al fine di evitare sovraffollamento o spreco di risorse. Questo richiede un'attenta pianificazione e monitoraggio delle risorse, cosí come un'analisi approfondita delle metriche di utilizzo. La configurazione errata delle risorse potrebbe comportare problemi di prestazioni, scalabilitá o disponibilitá.

Nonostante l'introduzione di maggiore complessitá rispetto ad un'approccio baremetal, l'utilizzo di Kubernetes porta, nel lungo termine, benefici maggiori. Se prima si é parlato della maggiore difficoltá di adattamento dei software, la loro definizione e

configurazione, una volta fatta, rimane sempre conpatibile con il sistema. Se ad esempio si crea la necessitá di una nuova installazione, basta l'esecuzione delle definizioni create precedentemente e l'intero sistema ritorna operativo, semplificando cosí il lavoro. Oppure, per la creazione di ridondanza dei software, in un caso standard sarebbe compito degli amministatori di sistema crearla e renderla funzionante. Con Kubernetes, tramite l'utilizzo dei **Deployment** o dei **Daemonset**, la creazione di ridondanza avviene automaticamente garantendo la stabilitá dello stato generale degli oggetti e delle risorse.

Conclusioni e lavori futuri

Negli ulimi anni, con l'avvento e la crescita di tecnologie come la containerizzazione e Kubernetes, l'approccio di distribuzione e gestione di cluster HPC é notevolmente cambiato. Si é infatti passati da cluster gestiti totalmente bare-metal, ad avere sistemi ibridi, dove alcuni software sono distribuiti con l'utilizzo di container, ad oggi, dove si studiano e si progettano sistemi di gestione totalmente comandati da Kubernetes.

L'intero mondo della produzione HPC si sta spostando verso questo approccio: un esempio é il nuovo supercalcolatore LEONARDO.

L'installazione di un infrastruttura di gestione per un ambiente HPC é sicuramente un'attivitá che richiede conoscenze a tutto tondo degli strumenti necessari, dei software che offrono i vari servizi, la rete che offre l'interconnessione del sistema, il tutto in modo che l'infrastruttura sia robusta e garantisca il massimo della disponibilitá. L'utilizzo di Kubernetes, risolve in modo autonomo molti dei lavori che altrimenti gli amministratori di sistema dovrebbero risolvere manualmente. Infatti offre tutti gli strumenti per poter configurare un sistema scalabile, tollerante ai guasti, efficiente, sicuro e flesibile.

Questi benefici peró non sono gratuiti. Per potersi approcciare a Kubernetes é sicuramente necessaria una fase di apprendimento. Bisogna capire a fondo i concetti e il funzionamento degli strumenti che vengono utilizzati.

Con questa tesi, si sono voluti portare alla luce quelli che sono i processi di progettazione di un sistema di gestione con l'utlizzo di Kubernetes, le criticitá e i vantaggi introdotti dall'utilizzo di questo approccio. A tal fine é stato implementato un cluster di gestione per un sistema HPC con l'utilizzo di Kubernetes.

Per la realizzazione del sistema si sono seguite essenzialmente quattro fasi:

Progettazione

Configurazione della rete e distribuzione

Implementazione del sistema Kubernetes

Test

Nella fase di progettazione é stata pensata l'architettura che il sistema avrebbe dovuto avere una volta operativo, quindi come connetterlo alla rete e quali software sarebbero stati necessari. La configurazione della rete ha richiesto la configurazione di dispositivi di rete e lo studio di concetti, come VLAN e bonding, fondamentali per topologie di rete che richiedono efficienza e robustezza. Per la distribuzione dei nodi é stato utilizzato xCAT, strumento fondamentale per la definizione e installazione in parallelo di molti nodi.

Una volta distribuiti i nodi, é stato implementato il cluster Kubernetes. Il sistema finale comprende, oltre ai componenti base, anche software di monitoraggio, con prometheus e grafana, e un filesystem distribuito basato su CephFS e Rook.

Infine si sono svolti alcuni test per capire quanto incidesse sul sistema l'utilizzo di Kubernetes. I servizi che introducono overhead sul sistema sono quelli necessari al funzionamento e alla gestione di Kubernetes. I servizi in questione sono: quelli responsabili dell'allocazione delle risorse dei container, kubelet e il container runtime CRI-O, quelli che gestiscono il cluster e il plugin di rete CALICO.

Dai test é emerso che l'impatto generale l'impatto di questi servizi sul sistema é del 3% sulla memoria e del 1.5% sulla CPU. Si ha anche un'impatto sulla rete, data l'introduzione di reti virtuali che gestioscono il traffico dei container in Kubernetes, ma é trascurabile per sistemi dove l'efficienza della rete non é fondamentale.

In conclusione si puó affermare che l'utilizzo di Kubernetes noffre numerosi vantaggi introducendo peró una certa complessitá nelle fasi iniziali. A seguito dei test si puó inoltre affermare che l'impatto dei servizi di gestione é minimo e non compromette le prestazioni generali del sistema.

.

Il sistema implementato é promettente ed in futuro potrá diventare operativo. Nei prossimi lavori, ci saranno sicuramente la creazione e l'adattamento di tutti quei servizi che ancora i nodi master stanno offrendo come: il servizio NTP per la sincronizzazione degli orologi dei nodi e il servizio di mail postfix per gestire al meglio gli allarmi dei sistemi di monitoraggio.

Per quanto riguarda i servizi installati, saranno sicuramente riconfigurati in modo che garantiscano maggiore sicurezza. Si seguirá con maggiore attenzione la gestione degli utenti e dei certificati perché ognuno possa accedere alle risorse prestabilite. Lo storage distribuito sará riconfigurato cosí che possa essere utilizzare anche all'esterno del cluster Kubernetes.

In futuro, il sistema potrebbe essere utilizzato per effettuare prove di installazione di RedHat OpenShift[20], una piattaforma di sviluppo e deployment di applicazioni containerizzate basata su Kubernetes.

Bibliografia

[1] LEONARDO supercomputer CINECA - https://leonardo-supercomputer.cineca.eu/

[2] Strategic Research Agenda - https://www.etp4hpc.eu/sra.html

[3] Top 500 - https://www.top500.org/lists/top500/2022/11/

[4] xCAT - https://xCAT.org/

[5] NAGIOS - https://www.nagios.org/

[6] Ganglia - http://ganglia.info/

[7] Prometheus - https://prometheus.io/

[8] Grafana - https://grafana.com/

[9] CephFS - https://docs.ceph.com/en/quincy/cephfs/index.html

[10] SLURM - https://slurm.schedmd.com/overview.html

[11] OpenPBS - https://openpbs.org/

[12] IBM Tivoli Storage Manager - https://www.ibm.com/docs/en/tsm/7.1.1?topic=servers-tivoli-storage-manager-overview

[13] Kubernetes - https://Kubernetes.io/docs/

[14] CNI - https://www.cni.dev/

[15] Rocky Linux - https://rockylinux.org/

[16] CRI-O - https://CRI-O.io/

[17] HAProxy - https://www.haproxy.com/

[18] Keepalived - https://www.keepalived.org/

[19] Lenovo RS G8052 - https://serveroption.lenovo.com/rack_switches/g8052/

[20] Red Hat OpenShift - https://www.redhat.com/it/technologies/cloud-computing/openshift