Esame di Virtualizzazione

1. Descrizione del progetto

1.1 Descrizione ed obbiettivo

L'obbiettivo di questo progetto è quello di creare un cluster kubernetes production ready e con high availability (HA).
Questo progetto si occupa solamente di dare una soluzione pronta all'uso per la creazione del solo cluster kubernetes,
senza quindi occuparsi di ulteriori aspetti sistemistici quali configurazione del Firewall, gestione della DMZ, gestione dei certificati SSL, eccetera.
Il cluster che si vuole creare è un cluster che risolva il problema dato da un scenario molto diffuso in ambito aziendale ovvero la gestione in container di un applicazione client server sviluppata a micro servizi che sfrutti al meglio le funzionalità che kubernetes offre mantenendo comunque semplice la gestione del cluster.
L'applicativo in questione avrà 3 micro servizi:

- applicazione client
- applicazione server
- database

Essendo lo scopo di questo progetto un'architettura high available, saranno presenti più nodi control plane, più worker node e almeno 2 load balancer per accedere ai control plane ed ai worker node.
Per lo sviluppo di questo progetto sono state usate solamente tecnologie Open Source

1.2 Challenge affrontate nel progetto

1. Lo sviluppo di applicazioni orientati ai micro servizi
2. La containerization dei vari applicativo con relativa gestione in cloud tramite un docker hub
3. La creazione di un cluster kubernetes con high availability
4. L'horizontal pod autoscaling dei micro servizi
5. La gestione di un DB Mysql con high availability (quindi la gestione dello storage condiviso in più nodi in modo da garantire le funzionalità anche in caso di spegnimento di un nodo)
6. Rendere Client e Server disponibile per gli utenti tramite distinti nomi DNS.

1.3 Descrizione dell'applicazione containerizzata

L'applicazione containerizzata consiste in:

- applicativo client: Applicazione lato client sviluppata con React JS, utilizzo di Nginx come HTTP server.
- applicativo server: Applicazione server sviluppata con Node JS utilizzando il framework Express JS.
- database: Database Mysql, utilizzo di Percona operator per Mysql per rendere Mysql High Available e distribuito su più nodi.
- load balancer: Utilizzo del load balancer HAProxy insieme all'applicativo Keepalived per gestire la ridondanza sui load balancer
- strumento di virtualizzazione: Per creare il cluster e renderlo il più simile possible ad un ambiente di produzione si fa uso di macchine linux ubuntu server create tramite il software LXD

1.4 Strumenti per il testing

Per testare la web app sarà sufficiente testare il corretto funzionamento collegandosi dal browser all'URL del client (default client.local, importante aggiungere l'host nel file `/etc/hosts`, facendolo puntare al Virtual IP del load balancer).
Per testare l'horizontal pod autoscaling verrà utilizzato il software Locust perfetto per load e stress testing.
Per testare invece la ridondanza tra i server e quindi il corretto funzionamento dell'high availability cluster basterà spegnere e riaccendere i nodi del cluster in momenti randomici.
Per testare il Database è inoltre presente un Deploy dell'immagine di PHP My Admin (usando phpmyadmin.local come ingress).

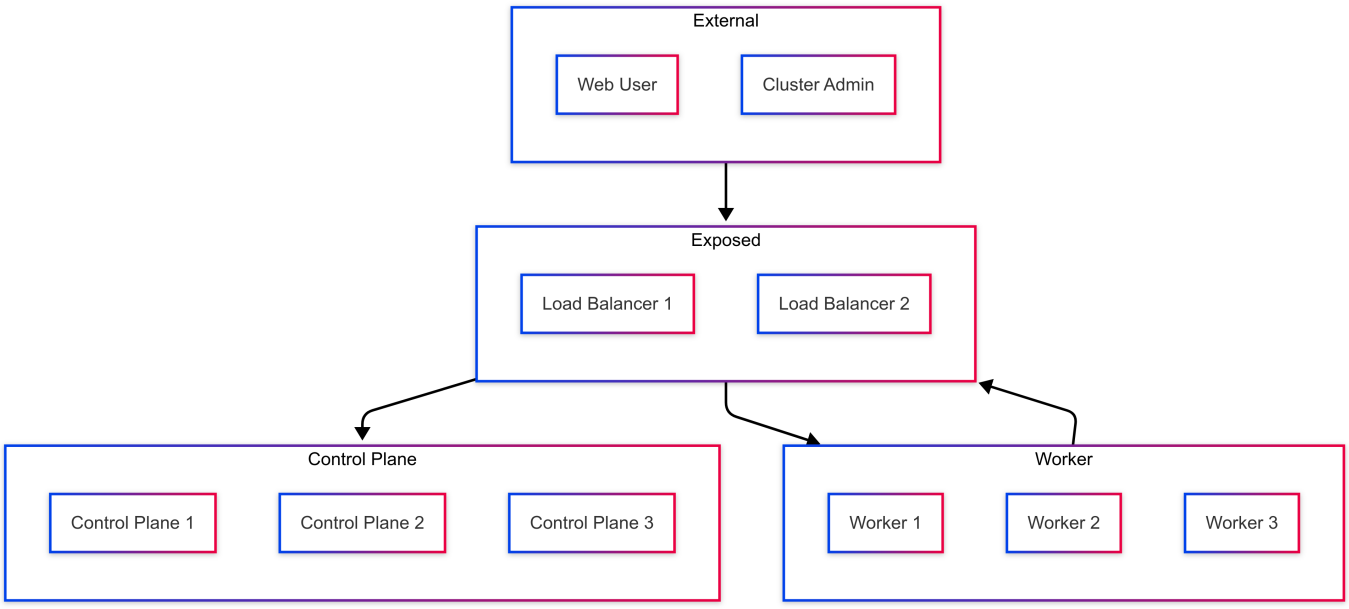
2. Infrastruttura

2.1 Descrizione del Cluster

Per questo progetto è stato scelto di realizzare un cluster kubernetes con high availability quindi con ridondanza di macchine server e con la totale assenza di single point of failure.
Il cluster è quindi composto da:

- 3 Control Plane: come descritto nella documentazione di Kubernetes, servono almeno 3 nodi control plane per creare un HA cluster. Per avere quorum nel sistema di consenso (etcd) e garantire la disponibilità anche in caso di guasto di un nodo.
- 3 Worker Node: sono stati scelti 3 nodi worker per garantire risorse sufficienti all'esecuzione dei carichi applicativi e per rispettare i requisiti minimi del Percona Operator, che richiede almeno 3 repliche per il corretto funzionamento del cluster database in modalità HA.
- 2 Load Balancer: vengono utilizzati per fornire un punto di accesso unificato al cluster, distribuendo il traffico verso i 3 nodi control plane e bilanciando gli accessi agli ingress delle web app tra i nodi worker. La presenza di due istanze evita un single point of failure; tramite Keepalived viene gestito un IP virtuale condiviso, che garantisce continuità di accesso al cluster anche in caso di guasto di uno dei due load balancer.

2.2 Diagramma del cluster



Implementazione

3.1 Sviluppo di applicazioni orientate ai micro servizi

Per utilizzare al meglio kubernetes e le sue funzionalità è preferibile sviluppare applicazione orientate ai micro servizi, per ottenere questo risultato ho creato 2 progetti distinti per client e server (in modo da poterli scalare singolarmente).
Inoltre, sempre per poter sfruttare al meglio lo scaling e la possibilità di distribuire i pod di una stesso micro servizio su nodi diversi, ho sviluppato sia l'applicativo client che l'applicativo stateless per non avere problemi di session in memoria o di storage condivisi.
Per quanto riguarda MySQL, invece, ho scelto di utilizzare il Percona Operator per MySQL che gestisce automaticamente la creazione di un cluster database altamente disponibile, con replica sincrona e failover automatico, facilitando così la gestione e la resilienza del database all'interno dell'ambiente Kubernetes.

3.2 Containerizzazione dei micro servizi

Ogni micro servizio ha la sua immagine creata tramite DockerFile che esegue la build del micro servizio e poi lo espone tramite una porta.
Queste immagini vengono successivamente pushate sul docker hub (un docker hub pubblico in questo caso) per poi essere pullate da kubernetes.
Entrando nella cartella di ogni progetto è possibile trovare il DockerFile contenente le istruzione per creare l'immagine del container.

3.3 Creazione del cluster kubernetes con High Availability

Per creare un cluster Kubernetes con high availability è necessario avere almeno 3 nodi control plane, questo rende possibile il funzionamento delle attività che deve svolgere il control plane anche in caso di rottura di uno dei 3 nodi. Questo è possibile perché la sincronizzazione dei nodi si basa su un sistema di consenso (etcd) che richiede una maggioranza (quorum) per garantire coerenza e disponibilità: con 3 nodi, il cluster può tollerare la perdita di uno di essi mantenendo il controllo operativo senza rischiare inconsistenze o interruzioni.
I load balancer vengono configurati per distribuire il traffico verso l'API server sui 3 nodi control plane, oltre a gestire il bilanciamento delle richieste HTTP e HTTPS (porte 80 e 443) inoltrate ai service NGINX Ingress presenti sui nodi worker.

3.4 Horizontal pod autoscaling

Viene utilizzato l'Horizontal Pod Autoscaling (HPA) per monitorare dinamicamente il carico di lavoro sull'applicazione server e adattare automaticamente il numero di pod in esecuzione. In questo modo, il sistema è in grado di scalare orizzontalmente le risorse in base al volume delle richieste ricevute, garantendo prestazioni ottimali anche in caso di picchi di traffico e ottimizzando l'utilizzo delle risorse del cluster.
Per rendere possibile ciò è stato inoltre installato il metric server.

3.5 High Availability con Mysql

Per garantire HA con MySQL ho utilizzato il Percona Operator per MySQL, una soluzione che permette la creazione di un cluster MySQL distribuito e resiliente. Il Percona Operator automatizza la creazione e il failover del database, assicurando che i dati siano sempre sincronizzati tra i nodi e che il sistema continui a

funzionare anche in caso di guasto di uno o più nodi. Per lo storage persistente ho scelto Rancher come storage provisioner, che fornisce volumi dinamici.

3.6 Creazione degli ingress

Per permettere l'accesso alle applicazioni client e server tramite domini personalizzati, ho configurato gli Ingress nel cluster Kubernetes. Utilizzando il controller NGINX Ingress, ho creato regole che instradano il traffico verso i rispettivi servizi in base ai domini client.local e server.local.