UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Sistemi Operativi**

Kubernetes : la piattaforma di Google per la gestione dei container

Anno Accademico 2015/2016

Candidato:

Mario Brescia

matr. N46001647

Indice

Introduzione

In informatica, con il termine virtualizzazione si fa riferimento a tutte quelle tecnologie che permettono l'astrazione delle componenti hardware (CPU,RAM e storage) degli elaboratori al fine di renderle disponibili al software sotto forma di risorsa virtuale.

Un approccio alternativo alla virtualizzazione è la cosiddetta virtualizzazione basata su container, nota anche come virtualizzazione a livello di sistema operativo. Il concetto chiave è quello di container, ovvero un' istanza isolata in grado di incapsulare al suo interno sia l'applicazione, sia le relative librerie.

Scopo di questo elaborato è fornire una panoramica generale su questa tecnologia presentandone le caratteristiche e i vantaggi che ne hanno favorito l'utilizzo nel mondo del cloud computing. Verranno quindi messi in evidenza i motivi per i quali nasce la necessità di avere strumenti che permettano di gestire i container ad alto livello.

Ci si concentrerà con maggiore attenzione su uno di questi strumenti , Kubernetes, che è divenuto negli ultimi anni uno dei più utilizzati tool di orchestrazione dei container. Verranno presentate quindi le sue caratteristiche e la sua architettura , mettendone in risalto le principali funzionalità offerte mostrando infine nell'ultima parte come questo strumento possa essere utilizzato per progettare una semplice applicazione multi-tier.

Capitolo 1: I Container

I container rappresentano spazi utente multipli e isolati direttamente in esecuzione sul livello di sistema operativo. Un container incapsula l'applicazione e tutte le sue dipendenze risultando indipendente dall'architettura sottostante. Sono utilizzati in alternativa alle macchine virtuali attraverso una tecnica di virtualizzazione nota come virtualizzazione basata su container. La differenza fondamentale rispetto all'approccio classico basato sull'utilizzo dell'hypervisor sta nella condivisione del medesimo kernel fra tutti i container in esecuzione. Quindi come mostrato in figura 1 [21] , non è possibile avere in esecuzione sullo stesso hardware ambienti virtualizzati con sistemi operativi diversi, cosa che è invece possibile utilizzando le macchine virtuali.

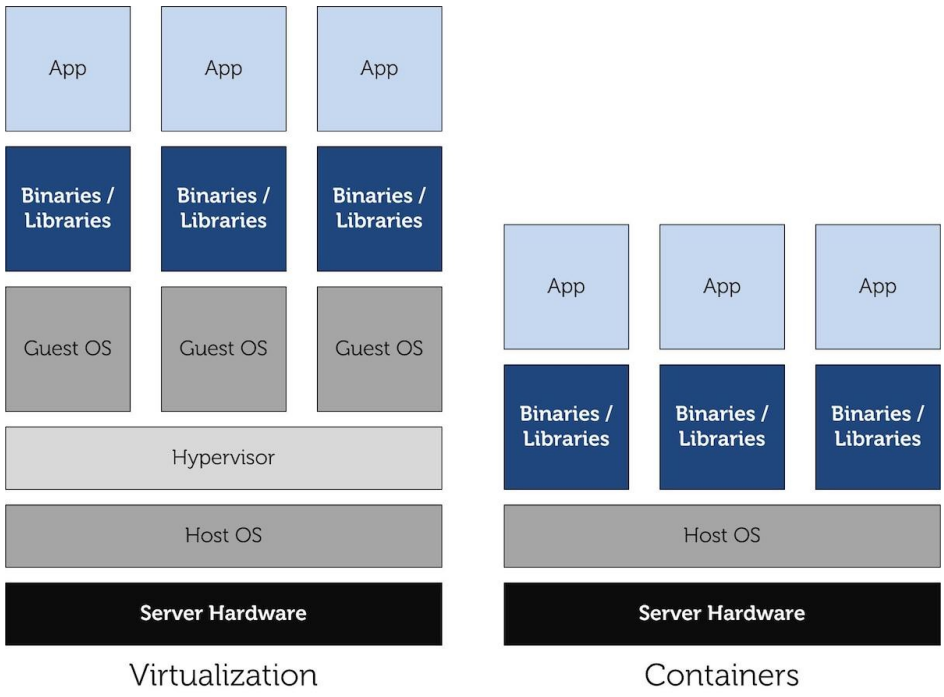

Figura 1 : Differenze tra virtualizzazione classica e virtualizzazione basata su container

1.1 Vantaggi nell'uso dei Container

Analizziamo nel dettaglio i vantaggi offerti da questa tecnologia :

- **isolamento** : al fine di garantire l'isolamento delle risorse tra il sistema host e i container in esecuzione su di esso, viene implementato un meccanismo del kernel noto come namespace per fare in modo che ogni processo abbia una propria visione del sistema , differente da quella degli altri processi. Le risorse sono quindi effettivamente condivise, ma ogni container vede le risorse (filesystem,memoria,processi e dispositivi) come se fossero interamente dedicate ad esso
- **migliore gestione delle risorse** : al fine di gestire l'accesso alle risorse , i container utilizzano un altro modulo kernel noto come cgroups, che permette sia di limitare l'accesso a CPU, memoria e I/O per alcuni container, sia di fare in modo che essi possano accedervi con maggiore priorità rispetto ad altri
- **sviluppo semplificato** : un container impacchetta l'applicazione in un singolo componente che può essere distribuito e configurato con una sola linea di comando, non bisogna quindi preoccuparsi dell'eventuale configurazione dell'ambiente di esecuzione
- **maggiore disponibilità** : uno sviluppatore può ospitare sul proprio computer miriadi di container. Un container è infatti molto più leggero di una macchina virtuale che può arrivare anche a pesare alcuni Gb. L'esecuzione di più macchine virtuali contemporaneamente è comunque possibile ma andrebbe a impattare pesantemente con le prestazioni del sistema, cosa che non accade invece con i container grazie alla loro dimensione ridotta
- **tempi di avvio più rapidi** : dal momento che si virtualizza solo il sistema operativo e non l'intera macchina, il container può essere avviato in un tempo che è molto più breve di quello necessario per avviare una macchina virtuale
- **portabilità e consistenza** : è probabilmente il vantaggio più importante che questa tecnologia è in grado di offrire grazie al suo formato che consente l'esecuzione applicativa su diversi host. Grazie alla standardizzazione è poi facile spostare velocemente i carichi di lavoro lì dove sono eseguiti in maniera più economica e rapida, evitando i problemi di compatibilità legati alle particolari caratteristiche delle singole piattaforme dei provider

1.2 Utilizzo dei container nel cloud computing

Nonostante i numerosi vantaggi, i container hanno sofferto nei primi anni di vita, la mancanza di supporto che ne stimolasse l'evoluzione. Con il passare del tempo però, come affermato dal CRO di Rackspace (azienda da sempre attiva nella gestione del cloud) "sempre più casi d'uso si sono rivelati calzanti per l'utilizzo dei container, come ad esempio la distribuzione in modo rapido e il porting delle applicazioni tra diversi tipi di infrastrutture, la creazione e la gestione di servizi di cloud più accurati ed efficienti, lo sviluppo di metodi di trasporto e così via"[16].

Questi sono fondamentalmente i motivi per cui numerose aziende hanno iniziato poi a supportare lo sviluppo di questa tecnologia. Oltre LXC infatti abbiamo diverse soluzioni che implementano i container, da Core OS, a Docker, passando per Mesos fino a Kubernetes, il cuore di questo elaborato.

Il vero punto di forza dei container sta proprio nella minor potenza di calcolo richiesta, nella leggerezza, nella portabilità, nell'isolamento e soprattutto nella capacità di fare a meno dell'hypervisor. Essendo poi basati su tecnologie open-source rappresentano un'ottima soluzione alle richieste di maggiore agilità, risparmio e flessibilità nel contesto delle risorse informatiche che vengono utilizzate per il cloud pubblico,privato o ibrido. Il cloud può essere visto come una galassia di server molto eterogenei, ognuno con delle proprie caratteristiche e risulta quindi chiaro perché molte aziende, tra cui Google in primis, usano i container per far girare in maniera efficiente i workload su questi server. Ogni applicazione nell'ecosistema Google viene infatti eseguita all'interno di container, da Gmail a Maps, fino alla ricerca Web e sono circa 2 miliardi i container avviati ogni settimana da Google.

Il container rappresenta l'elemento perfetto per gestire le complessità soprattutto perché, oltre a possedere una forte componente di automatizzazione dei suoi processi di creazione,avvio e gestione, consente anche in modo molto semplice la riorganizzazione tramite crescenti livelli di astrazione.

L'utilizzo di questa tecnologia permette poi di allontanarsi ancora di più dai vincoli

hardware permettendo di avere un sistema che presenta una maggiore flessibilità, compattezza e funzionalità sia per le macchina host, sia per i fornitori di servizi, sia per gli utilizzatori finali del sistema stesso.

1.3 Strumenti per la gestione dei container

Al crescere del numero dei container in esecuzione su vari host, diventa fondamentale avere degli strumenti appositi per gestirli in maniera semplice, diretta e con un livello di astrazione anche abbastanza elevato. Tra le varie soluzioni presenti nel mercato odierno abbiamo Docker e Kubernetes che operano a differenti livelli di astrazione.

Docker è un progetto open-source che estende la tecnologia LXC (Linux Container), introducendo una API di alto livello che permette di automatizzare la distribuzione del software in ambiente sicuro e riproducibile. Fornisce quindi tutte le funzionalità necessarie per costruire,caricare,scaricare,avviare e arrestare un container. È pensato per la gestione di questi processi soprattutto in ambienti single-host con un numero abbastanza piccolo di container.

Quando le applicazioni sono scalate attraverso più host, diventa fondamentale gestire ogni host e astrarne la sua piattaforma sottostante in maniera tale da eliminare eventuali complessità. Con il termine *orchestration* intendiamo proprio la schedulazione dei container, la gestione del cluster e il concetto di approvvigionamento. È in questo contesto che andremo ad introdurre Kubernetes.

La schedulazione si riferisce alla capacità dell'amministratore di caricare su un host un servizio che stabilisce come eseguire un dato container.

Per gestione del cluster si intende invece il processo di controllo di gruppi di host. Questo può includere l'aggiunta o la rimozione di host dal cluster, l'acquisizione di informazioni sullo stato corrente di host e container , l'avvio e l'interruzione di processi.

Una delle maggiori responsabilità dello schedulatore è la selezione dell'host : se un amministratore decide di eseguire un servizio (un container) sul cluster, solitamente lo schedulatore seleziona l'host in modo automatico. L'amministratore può però

opzionalmente imporre dei vincoli sulla schedulazione in accordo con i suoi bisogni e desideri, ma alla fine è sempre lo schedulatore a decidere se rispettarli o meno.

Gli schedulatori più avanzati mettono a disposizione una funzionalità che permette la gestione di gruppi di container per consentire all'amministratore di trattare un insieme di container come una singola applicazione, mantenendo i vantaggi derivanti dall'uso dei container e riducendo l'overhead addizionale di gestione.

Un concetto legato alla gestione del cluster è quello di approvvigionamento**,** ovvero quel processo mediante il quale un amministratore di sistema può inserire nuovi host nella rete e configurarli in maniera semplice, così che essi possano essere usati per eseguire delle operazioni.

Nonostante il risultato dell'approvvigionamento sia sempre la comparsa di un nuovo host online disponibile per eseguire nuove operazioni, la metodologia con cui ciò viene fatto dipende dal tool e dal tipo di host usati. L'approvvigionamento può infatti essere eseguito dall'amministratore, oppure essere inglobato nei tool di gestione del cluster per lo scaling automatico. Questo secondo metodo comporta la definizione di un processo per richiedere host addizionali così come la definizione delle condizioni che devono verificarsi affinché questa richiesta venga inoltrata. Ad esempio, se la nostra applicazione soffre di un sovraccarico del server, sarebbe desiderabile che il nostro sistema avviasse nuovi host e scalasse i container attraverso l'infrastruttura al fine di alleviare la congestione.

Alcuni progetti che offrono gestione del cluster e schedulazione basilari sono :

- fleet , marathon (componente di Mesosphere) e Swarm

Tra i progetti che invece offrono funzionalità di schedulazione avanzate e gestione di gruppi di container come una singola unità abbiamo :

- kubernetes e docker's compose

Capitolo 2: Kubernetes

Kubernetes è un progetto open-source rilasciato da Google nel Giugno del 2014 volto a incrementare lo sforzo dell'azienda nel voler condividere i vantaggi della sua infrastruttura e tecnologia con tutta la comunità su grande scala. Il progetto nasce fondamentalmente dalle perplessità di Google circa i limiti mostrati da Docker nel coordinare, gestire e distribuire i container e i carichi di lavoro man mano che le risorse disponibili nell'infrastruttura sottostante erano consumate e soprattutto al crescere del numero di host e container . Google asserisce con fermezza che tutti questi problemi possono essere superati con l'utilizzo di Kubernetes che sin dalla sua prima versione ha subito un rapido sviluppo dovuto soprattutto all'enorme contributo di tutta la comunità open-source, da Red Hat a Vmware. La piattaforma è utilizzata per automatizzare lo sviluppo , la scalabilità e le operazioni di applicazioni containerizzate all'interno di gruppi di host, fornendo un'infrastruttura basata sui container. In una rete di computer Kubernetes è in grado di gestire lo scheduling sui nodi e i carichi di lavoro in modo attivo per assicurare che vengano soddisfatte le esigenze degli utenti. Per garantire una più semplice gestione dei container, Kubernetes utilizza i concetti di label e pod. I pod sono insiemi di container co-locati, i label sono invece usati per organizzare e selezionare gruppi di oggetti. Il funzionamento di Kubernetes è basato sul ciclo di controllo mostrato in figura 2 [19] :

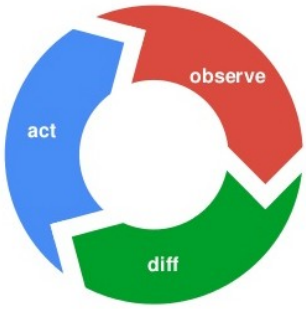

Figura 2 : Ciclo di controllo

esso si basa sui concetti di *stato desiderato e stato reale* che rappresentano la chiave di gestione dell'intero cluster e dei suoi carichi di lavoro. Tutti i componenti in Kubernetes lavorano infatti in modo continuo per monitorare lo stato reale e sincronizzarlo con lo stato desiderato definito dall'amministratore.

2.1 Vantaggi

Kubernetes si presenta come un tool molto utile nella gestione di più container in esecuzione su host diversi all'interno di un cluster. Partendo dalle funzionalità base di Docker, le estende fornendo un livello di astrazione ancora più elevato e grazie all'automatizzazione dello sviluppo e della replicazione dei container, ottenibili lanciando semplici istruzione da riga di comando, è possibile rispondere alle esigenze sempre crescenti di scalabilità delle nostre applicazioni. L'attenzione dello sviluppatore grazie all'utilizzo di questo tool, è completamente rivolta all'applicazione e non più alla piattaforma su cui essa verrà eseguita. I vantaggi sono evidenti anche per gli amministratori del cluster : nel momento in cui un' applicazione richiede più risorse, è possibile in maniera semplice e intuitiva redistribuire i vari container (raggruppati in pod) in modo da ripartire i carichi di lavoro tra i nodi correntemente attivi del nostro cluster. Anche l'aggiornamento di un'applicazione diventa un'operazione rapida e invisibile all'utente effettivo del servizio che la nostra applicazione fornisce. Lo stesso accade quando un container viene eliminato : l'operazione di rimpiazzamento viene eseguita in tempi rapidissimi grazie all'implementazione del ciclo di controllo e nel momento in cui lo stato reale del sistema non rispecchia quello desiderato dall'amministratore, tutti i componenti si attivano per far si che la situazione torni alla normalità. Il tool è utilizzato inoltre per gestire il networking dei servizi containerizzati e realizzare dei metodi di self-healing del cluster. Le potenzialità sono talmente elevate che la stessa Google afferma che Kubernetes può gestire cluster contenenti fino a 100 nodi con 30 pod per nodo e con 1-2 container per pod. Tutto questo garantendo un controllo più accurato sull'utilizzo delle risorse come CPU, memoria e spazio su disco .

2.2 Architettura

Kubernetes è un vero e proprio ecosistema in cui coesistono numerosi componenti che interagiscono tra di loro per orchestrare i container presenti in un cluster . I componenti possono essere classificati in due macro gruppi , quelli appartenenti al Master che rappresenta il “cervello” dell’intero ecosistema, e quelli appartenenti ai nodi worker che rappresentano le unità elaborative su cui vengono distribuiti i vari container . I componenti principali sono mostrati in figura 3 e saranno analizzati uno alla volta nei paragrafi successivi.

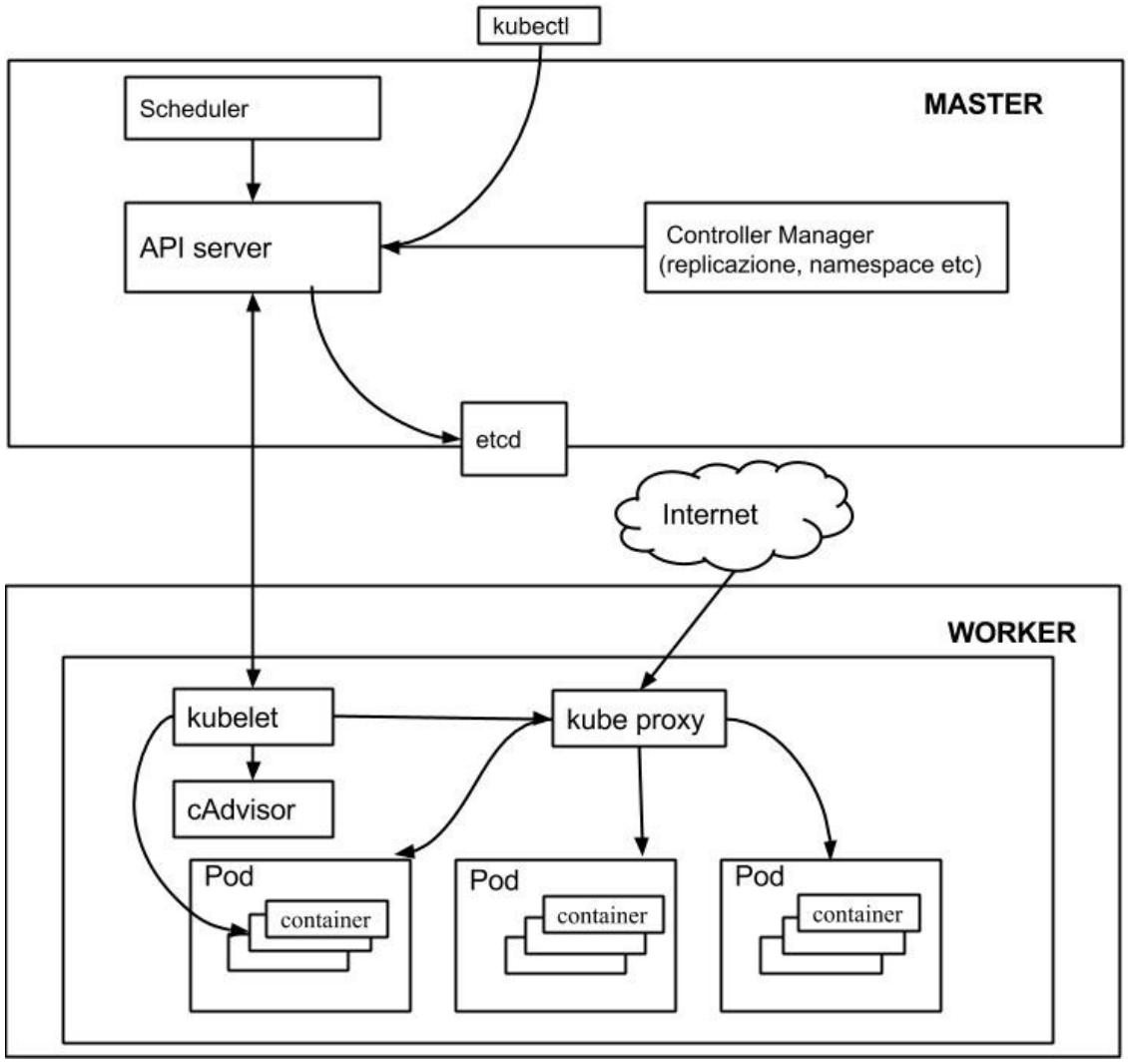

Figura 3 : Architettura interna di Kubernetes

2.2.1 Worker Node

Il nodo worker (precedentemente conosciuto come Minion) rappresenta una macchina fisica o virtuale che esegue una serie di operazioni. Ogni nodo è caratterizzato da uno stato (che può essere pronto o non raggiungibile) , un indirizzo IP esterno per essere visibile al di fuori del cluster e un indirizzo IP interno per essere raggiungibile all'interno del cluster stesso. Il nodo è poi caratterizzato da una certa capacità che definisce il numero di risorse disponibili su quel nodo, ovvero CPU, memoria e il numero massimo di pod che possono essere schedulati su di esso. Normalmente la capacità è impostata in automatico dallo scheduler, uno dei componenti più importanti del Master, in maniera tale che la somma delle risorse richieste dai vari container su un nodo non ecceda la capacità del nodo stesso. A differenza dei pod e dei servizi un nodo non è direttamente creato da Kubernetes ma è fornito da un cloud provider , ad esempio il Google Compute Engine, oppure è una macchine fisica o virtuale. Questo significa che quando Kubernetes crea il nodo , in realtà sta solo creando un oggetto che rappresenta lo stato interno del nodo. Creato il nodo vengono effettuati poi dei controlli di validità per verificare che esso funzioni nel modo corretto. Ogni nodo worker possiede i servizi necessari per eseguire i pod ed è gestito dal nodo Master. I servizi posseduti dal nodo includono Docker (che si occupa ovviamente di eseguire i container), kubelet e kube-proxy. Li analizzeremo ora nel dettaglio.

- **Kubelet** : uno dei servizi di base che gestisce i pod e i loro container, le loro immagini , i loro volumi etc. Rappresenta una sorta di supervisore che viene eseguito all'interno di ogni nodo. Kubelet lavora seguendo le direttive definite nel file di configurazione del pod, un oggetto in formato YAML o JSON, noto anche come manifesto. Quindi kubelet riceve un insieme di file di configurazione , ad esempio attraverso l'API server, e assicura che i container descritti in questi oggetti siano in esecuzione e funzionino correttamente. Interagisce poi con l'API server per aggiornare lo stato dei nodi e avviare nuovi carichi di lavoro assegnati allo

scheduler.

- **Kube-proxy** : un servizio di proxy che opera anche come load balancer e redirige il traffico destinato a specifici servizi sul pod appropriato nel cluster.
- **Pod** : è la più piccola unità che può essere creata e gestita in Kubernetes. Formalmente è un insieme di uno o più container che cooperano e quindi sono raggruppati logicamente. I pod sono sempre co-locati e schedulati insieme, e vengono eseguiti in un contesto condiviso che è un insieme di namespace,cgroups e altri strumenti in grado di garantire l'isolamento. I container che si trovano nello stesso pod condividono un indirizzo IP e un numero di porta e possono localizzarsi l'uno con l'altro tramite localhost. Inoltre possono comunicare usando i meccanismi standard di comunicazione tra processi. Le applicazioni all'interno di un pod hanno anche accesso a volumi condivisi, che sono quindi parte del pod e sono pronti per essere montati all'interno del filesystem di ogni applicazione.

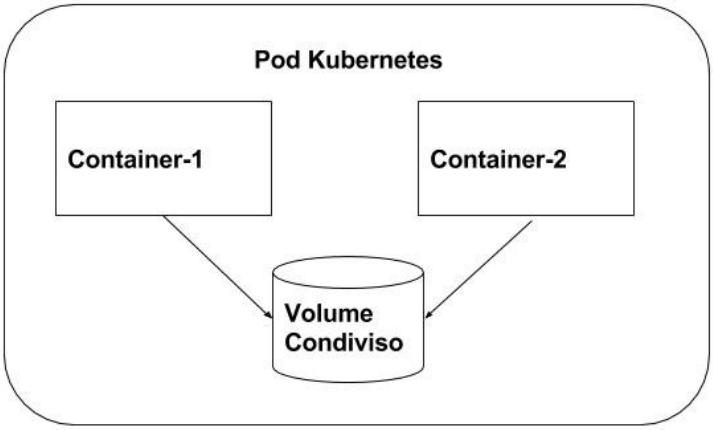

Figura 4 : Architettura di un pod

I pod hanno un ciclo di vita abbastanza breve: sono creati, viene assegnato loro un ID univoco e sono poi schedulati sui nodi dove rimangono fino alla terminazione (in accordo con determinate politiche) o alla cancellazione. Se un nodo muore, i pod schedulati da quel nodo vengono eliminati dopo la scadenza di un certo timeout. Dato un pod con un certo ID , esso non può essere rischedulato da un

nuovo nodo, piuttosto esso può essere replicato dando vita ad un nuovo pod che può avere anche lo stesso nome ma un nuovo ID. Questo è ciò che fa il controller di replicazione che vedremo successivamente. Naturalmente se un pod viene eliminato per qualche motivo anche i volumi a esso collegati saranno distrutti.

- **cAdvisor** : il suo compito è monitorare le performance e le risorse utilizzate dai container. A questo proposito esplora tutti i container nella macchina e colleziona le statistiche di utilizzo di memoria, filesystem, rete e CPU.

2.2.2 Master Node

Essenzialmente il master è il cervello del nostro cluster. Esso ospita tutti quei componenti che cooperano per fornire una visione unificata della rete e formano nel loro insieme il cosiddetto pannello di controllo di Kubernetes. In particolare abbiamo :

- **Scheduler** : in generale uno schedulatore di container ha lo scopo principale di far partire l'esecuzione dei container sull'host appropriato e di connetterli tra loro. Deve inoltre gestire i fallimenti in modo automatico e deve essere in grado di scalare i container quando ci sono troppi dati da processare ed elaborare su una singola istanza. Nel caso di Kubernetes quindi lo scheduler ha l'obiettivo principale di assegnare i pod non schedulati ai nodi. Lo scheduler utilizza predicati e priorità per stabilire su quale nodo un pod dovrebbe essere eseguito. I predicati sono regole obbligatorie per schedulare un nuovo pod sul cluster. Se nessuna macchina corrisponde ai predicati richiesti dal pod, allora esso rimarrà in uno stato di attesa finché non ci sarà una macchina che possa soddisfarli. Se poi lo scheduler trova più macchine che soddisfano i predicati richiesti, allora verranno utilizzate le priorità per capire quale macchina è più adatta ad eseguire quel determinato pod.
- **API server** : rappresenta la application programming interface di Kubernetes. É pensato per fornire le funzionalità CRUD (Create,Read,Update,Delete). Processa poi per lo più le cosiddette operazioni REST (delete,get,head,post,put), le convalida e aggiorna gli oggetti riferiti da queste operazioni che sono presenti in etcd. È bene

notare che le richieste di modifiche ad un nodo non vengono inviate direttamente al nodo stesso , ma sempre al nodo master sotto forma di query

- **Controller manager** : è responsabile del monitoraggio dei controller di replicazione e della creazione dei pod corrispondenti. Utilizza la API per rilevare nuovi controller e per creare/eliminare i pod
- **Controller di replicazione** : coopera con l'API server per assicurare che in ogni istante di tempo un pod o un insieme di pod siano sempre attivi e disponibili. Ogni controller di replicazione mantiene uno stato desiderato che è gestito dallo sviluppatore dell'applicazione e quando questo stato cambia, il controller tenta di modificare lo stato corrente affinché si riconcili con quello desiderato. Per esempio se incrementiamo il numero di istanze desiderate da tre a quattro, il controller di replicazione rileverà la necessità di una nuova istanza, la creerà e la lancerà da qualche parte nel cluster. Il controller di replicazione mantiene il template di un pod per creare nuovi pod se necessario. Fa poi uso dei label per gestire solo i pod di cui è responsabile. Ad esempio il controller di replicazione di una webapp frontend è responsabile di tutti i pod che hanno il campo app=webapp e il campo role = frontend. Il rilascio di aggiornamenti e modifiche può essere eseguito utilizzando i controller di replicazione e usando il comando kubectl rolling-update.
- **Etcd** : rappresenta un database basato su chiave distribuita che fornisce un metodo affidabile per memorizzare dati attraverso un cluster di macchine. Tutte le elaborazioni del nodo master che richiedono persistenza sono memorizzate in un'istanza di etcd.

2.3 Funzionalità Principali

2.3.2 Networking

Kubernetes permette di gestire quattro tipi di comunicazione :

1. Comunicazione tra container : ogni pod possiede un proprio indirizzo IP, ma poiché il container esiste nell'ambito del pod stesso, condividerà con esso l'indirizzo IP. Quindi i container all'interno di un pod possono comunicare con gli altri container tramite localhost.
2. Comunicazione tra pod : come detto ogni pod in Kubernetes possiede un proprio indirizzo IP reale che si suppone essere raggiungibile da tutti gli altri host e pod presenti nel cluster. Questa comunicazione può essere implementata attraverso strumenti ausiliari, tra cui Google Compute Engine, Calico e Flannel.
3. Comunicazione tra pod e servizio : un servizio è un'astrazione che definisce un insieme logico di pod e una politica con cui accedervi (ad esempio di bilanciamento del carico). I servizi sono fondamentali per la gestione di un cluster Kubernetes : dal momento che i pod possono essere creati, distrutti e rischedulati dinamicamente (ad esempio scalando l'applicazione o rilasciando modifiche) potrebbe accadere che l'indirizzo IP di un pod mantenuto dal relativo controller di replicazione che lo gestisce, non sia più lo stesso. Occorre quindi un meccanismo in grado di identificare in ogni istante il gruppo di pod con cui si sta comunicando Ciò viene realizzato attraverso l'uso dei servizi che assegnano a un insieme di pod un indirizzo IP che rimane stabile e fisso durante tutto il ciclo di vita del servizio. Il gruppo di pod coperti dal servizio è specificato usando un selettore di label.
4. Comunicazione tra servizio e mondo esterno : fino ad ora si è parlato di come accedere ad un pod o ad un servizio all'interno del cluster stesso. Se si vuole accedere a un pod dall'esterno del cluster la situazione diventa leggermente più complicata. Questa comunicazione è in genere implementata attraverso meccanismi di bilanciamento del carico esterni che permettono di raggiungere tutti i nodi del cluster. Quando ad un nodo giunge una richiesta, essa viene riconosciuta come

parte di un particolare servizio e instradata al pod appropriato

2.3.1 Creazione e scalabilità dei pod

Per creare i pod abbiamo due possibilità : creare un oggetto di tipo Deployment se abbiamo un solo container all'interno del pod oppure creare direttamente il pod se abbiamo più container al suo interno.

Il comando *run* crea un Deployment che va a monitorare i pod, avviandone di nuovi se qualcuno di essi fallisce mantenendo sempre coerente il numero di pod con quelli definiti nello stato desiderato del sistema. Quando creiamo il pod tramite Deployment va specificata l'immagine Docker da usare per il container, il numero di porta da esporre, il numero di repliche del pod da creare (se non specificato verrà creata una replica) ed eventuali label da agganciare al pod per identificarlo. Si noti che se i pod sono gestiti con i Deployment, per eliminarli definitivamente bisogna eliminare il Deployment corrispondente che li gestisce.

Nel caso in cui non volessimo usare i Deployment (ad esempio se il pod non produce dati persistenti che devono sopravvivere a un riavvio, o il pod è pensato per avere vita breve) è possibile creare il pod direttamente tramite il comando *create* passandogli un file di configurazione in formato YAML o JSON che descrive le caratteristiche del pod. Nel file di configurazione oltre a specificare il nome del container e dell'immagine Docker, è possibile specificare le politiche di riavvio dei container all'interno del pod e una lista di volumi che possono essere utilizzati dai container appartenenti allo stesso pod.

Se creiamo i pod direttamente diventa obbligatorio creare i controller di replicazione per gestirli al meglio e poter effettuare lo scaling dei pod attraverso il cluster. Questa è fondamentalmente la differenza tra i due approcci : dichiarativo se usiamo i Deployment, imperativo se usiamo i controller di replicazione.

Indipendentemente dall'utilizzo di un controller di replicazione o di un Deployment, è possibile incrementare o decrementare automaticamente il numero di pod in base ad alcune metriche, ad esempio la percentuale di utilizzo della CPU. L'autoscaling è

orizzontale se si scala il numero di istanze, verticale se si scala il numero di risorse utilizzate da un'istanza. In particolare l'autoscaler orizzontale viene implementato tramite un controller che periodicamente modifica il numero di repliche in un controller di replicazione o in un Deployment per far si che la percentuale media di utilizzo della CPU sia uguale a quella specificata dall'utente. La percentuale di utilizzo della CPU viene calcolata come rapporto tra la frazione di CPU utilizzata da un pod e la somma delle richieste di CPU da parte dei container che si trovano nel pod.

2.3.3 Health checking

Piuttosto che provare a scrivere codice privo di errori, un approccio migliore è usare un sistema di gestione degli errori che esegua periodicamente un controllo sullo stato dell'applicazione e che eventualmente la ripari. In questo modo è un sistema esterno alla nostra applicazione a essere responsabile del monitoraggio dell'applicazione stessa e di eseguire operazione per risolvere eventuali problemi. É importante che il sistema sia esterno, perché se fosse interno, qualora l'applicazione fallisse, fallirebbe anche il sistema di monitoraggio. In kubernetes il componente responsabile del monitoraggio è kubelet che interroga costantemente il Docker daemon per verificare se un certo container è in esecuzione e se così non è viene riavviato.

2.3.4 Namespace

Per separare utenti, gruppi o applicazioni all'interno del cluster si utilizzano i namespace. Il namespace è un meccanismo utilizzato per partizionare le risorse create dagli utenti in gruppi. Un singolo cluster infatti potrebbe essere utilizzato da più utenti o gruppi di utenti. Ogni utente però, avendo le proprie risorse (pod,servizi,rcs) e politiche (chi può o non può eseguire operazioni nel suo ambiente) vorrebbe avere la possibilità di lavorare in un ambiente isolato non accessibile agli altri utenti . Il namespace permette proprio di realizzare ciò fornendo un meccanismo di naming delle risorse (per evitare collisioni) e la possibilità di limitare il consumo di risorse per un certo gruppo di utenti. Questo

meccanismo è molto utile per gli operatori del cluster perché si possono ospitare più community di utenti su un singolo cluster. Altro vantaggio per gli operatori del cluster è la possibilità di limitare il numero di risorse che ogni community può utilizzare in modo da evitare che il loro consumo eccessivo possa danneggiare o limitare l'utilizzo del cluster per altre community. D'altro canto ovviamente, lo stesso utente del cluster trae vantaggi dall'utilizzo di questo meccanismo avendo la possibilità di interagire con risorse che sono attinenti alla sua community garantendo al contempo il loro isolamento.

Kubernetes prevede due namespace iniziali , default (per oggetti che non hanno altri namespace) e kube-system (per oggetti creati da Kubernetes). Ogni namespace può poi avere due stati : active se il namespace è in uso, o terminating se il namespace è in fase di cancellazione.

2.3.5 Scalabilità del cluster

Una delle più importanti funzionalità offerta da Kubernetes è la possibilità di gestire e scalare facilmente verso l'alto o verso il basso il numero di pod.

Cosa succede se a un certo punto i nodi sono saturi e sono richieste più risorse per schedulare nuovi pod per i nostri carichi di lavoro ? Al momento della creazione del cluster possiamo definire il numero iniziale di nodi (minions) che per default è impostato a quattro. Questo valore è mantenuto in una variabile di ambiente e ogni modifica a questa variabile dopo l'avvio del cluster non avrà effetto.

Per scalare il cluster rapidamente è possibile usufruire delle potenzialità offerte dal Google Cloud Engine che rappresenta un vero e proprio motore che supporta Kubernetes nelle sue elaborazioni.

Il Google Compute Engine fornisce una semplice e intuitiva interfaccia grafica che permette in modo rapido la scalabilità del cluster aggiungendo o rimuovendo dei nodi per schedulare nuovi pod e workload. L'unica nota da tenere in considerazione è che nel momento in cui effettuiamo un'operazione di scale down il Google Cloud Engine rimuove un nodo che non è però necessariamente l'ultimo aggiunto. In ogni caso

comunque i pod assegnati a quel nodo verranno rischedulati sui rimanenti nodi. Di seguito è mostrato il funzionamento di questo meccanismo , in particolare di una operazione di scaling verso il basso.

Supponiamo di avere inizialmente quattro nodi e che lo stato desiderato del sistema, monitorato dal controller manager e mantenuto dal controller di replicazione, coincida con quello attuale.

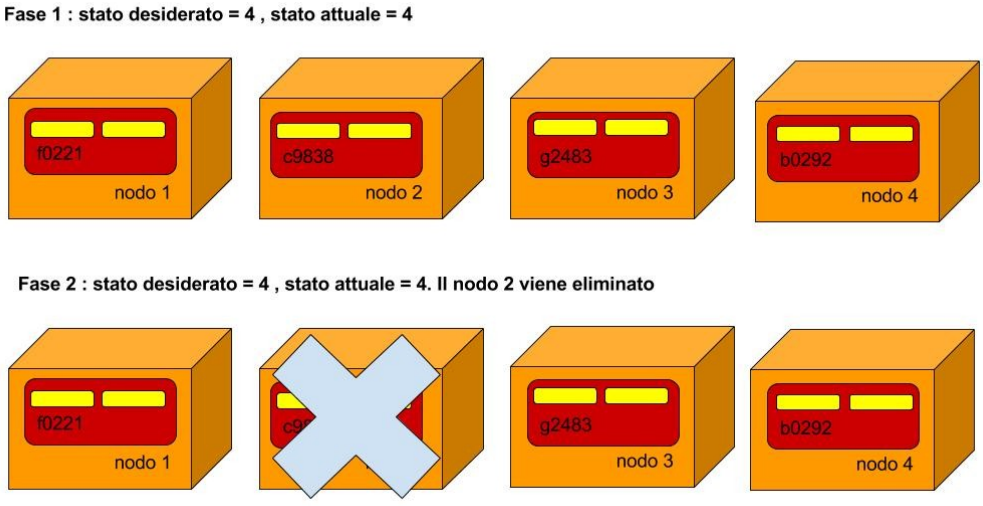

Figura 5 : stato iniziale del cluster e successiva eliminazione di un nodo

Supponiamo che in un certo istante uno dei nodi venga eliminato come mostrato in figura 5. In questo caso lo stato attuale sarà diverso da quello desiderato. Il controller manager che esegue il ciclo di controllo in modo continuativo, rileva il cambiamento e invoca il controller di replicazione che va a replicare il pod che era stato schedulato sul nodo eliminato come mostrato in figura 6.

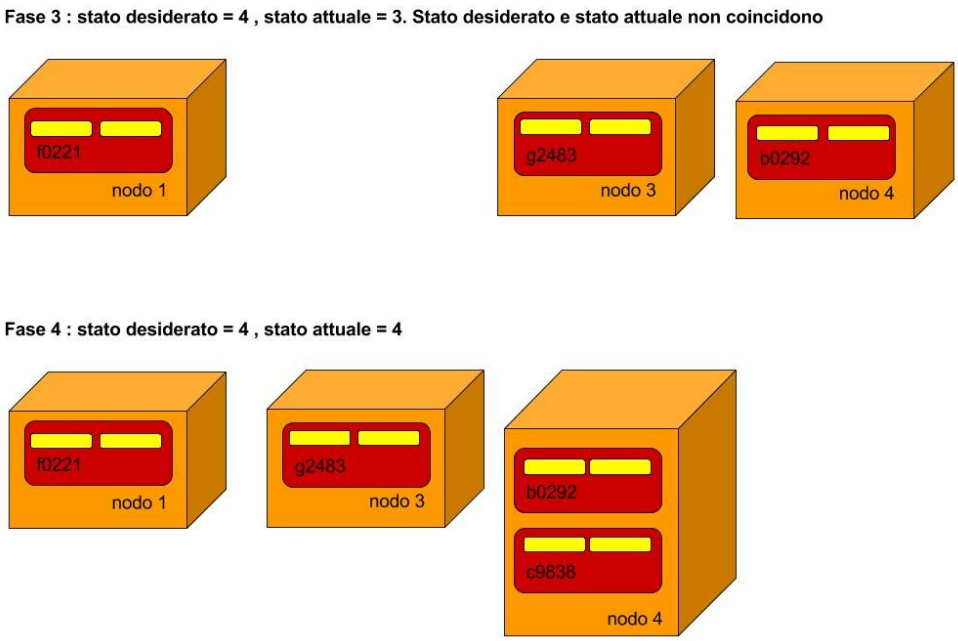

Figura 6 : Stato Intermedio del cluster e implementazione del ciclo di controllo

Capitolo 3: Progettazione di un'applicazione con Kubernetes

Verranno ora illustrati i passi chiave per la progettazione di una semplice e generica applicazione multi-tier costituita da front-end web-server e back-end database.

In particolare si supponga di voler realizzare un'applicazione costituita da tre container front-end e due container back-end. L'utente presumibilmente accederà tramite browser all'applicazione web e richiederà un certo servizio . Partirà quindi una chiamata al web-server che manderà a sua volta una richiesta al servizio di back-end per erogare la funzionalità richiesta.

Il primo passo è ovviamente scrivere i vari livelli dell'applicazione e successivamente inserirli in dei container Docker. Per fare ciò dobbiamo creare per ogni livello un Dockerfile, un vero e proprio file di testo che contiene tutti i comandi utili a Docker per creare immagini in modo automatico a partire da un'immagine base. É bene quindi capire subito cos'è un'immagine e qual è la differenza con un container.

Un' immagine è costituita dal filesystem e dai parametri da usare a tempo di esecuzione, non ha uno stato e non cambia mai. Un container invece è un'istanza di una certa immagine che viene mandata in esecuzione.

Generalmente in un Dockerfile compaiono alcune keywords base che permettono di creare un'immagine :

- **FROM** : comunica a Docker su quale immagine è basata la nostra immagine. Se l'immagine non viene trovata sull'host, Docker la cerca in un indice contenente tutte le varie immagini e ne effettua il download.
- **RUN** : prende un comando come suo argomento e lo esegue per costruire l'immagine.
- **CMD** : può essere usato per eseguire uno specifico comando

- **EXPOSE** : è usato per associare una porta specifica in modo tale da abilitare la connessione di rete tra il processo in esecuzione all'interno del container e il mondo esterno (ad esempio l'host).
- **VOLUME** : è usato per abilitare l'accesso del container a una directory sulla macchina host.

Dopo l'esecuzione dei Dockerfile avremo sostanzialmente dei veri e propri container che possono essere mandati in esecuzione o arrestati. Dati i container bisogna poi caricarli con un'operazione di push sul Google Container Registry , un repository privato utilizzato per memorizzare le nostre immagini di container Docker sulla piattaforma cloud di Google per un recupero e uno sviluppo semplice e scalabile.
Dopo il push, se tutto va a buon fine, le diverse immagini saranno disponibili nel Container Registry pronte per essere orchestrate da Kubernetes.
Una volta che i container possono essere acceduti e gestiti dalla nostra piattaforma, bisogna creare un cluster Kubernetes, ovvero un Master e dei nodi Worker su cui poi andare a distribuire i vari pod che incapsuleranno i vari container. Anche la creazione del cluster, grazie alla semplicità di utilizzo di Kubernetes, può avvenire semplicemente eseguendo un'apposita istruzione di creazione da riga di comando del tipo :

```
gcloud container clusters create NAME
```

In alternativa l'operazione può essere eseguita attraverso l'interfaccia web di Kubernetes.
Dopo la creazione dovremmo avere un cluster con tre nodi pronti a ricevere i nostri container come mostrato in figura 7.

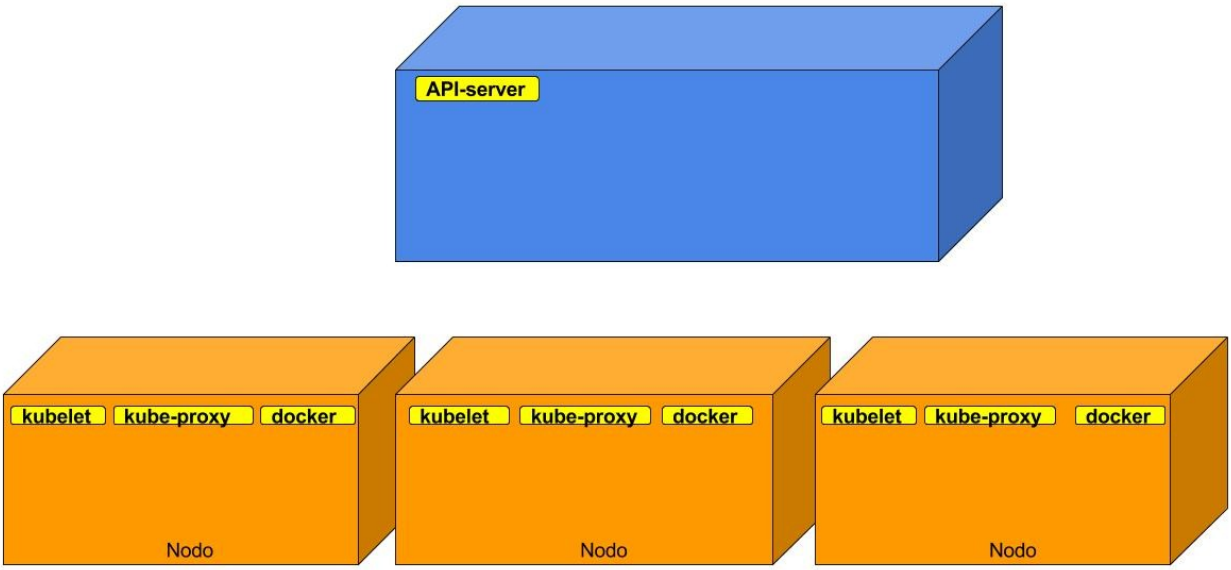

Figura 7 : Fase iniziale della costruzione di un cluster Kubernetes

Bisogna ora creare i controller di replicazione che, contenendo i template dei vari pod, permettono di gestirli in modo rapido. Nello specifico occorrono due controller, uno per il front-end che gestirà tre pod e uno per il back-end che gestirà due pod. Per creare un controller di replicazione si può usare il seguente comando :

```
kubectl create -f <replication controller filename>
```

Per visualizzare invece i controller presenti nel cluster basta digitare :

```
kubectl get rc
```

Dopo la creazione dei due controller avremo la situazione mostrata in figura 8 con un totale di cinque pod.

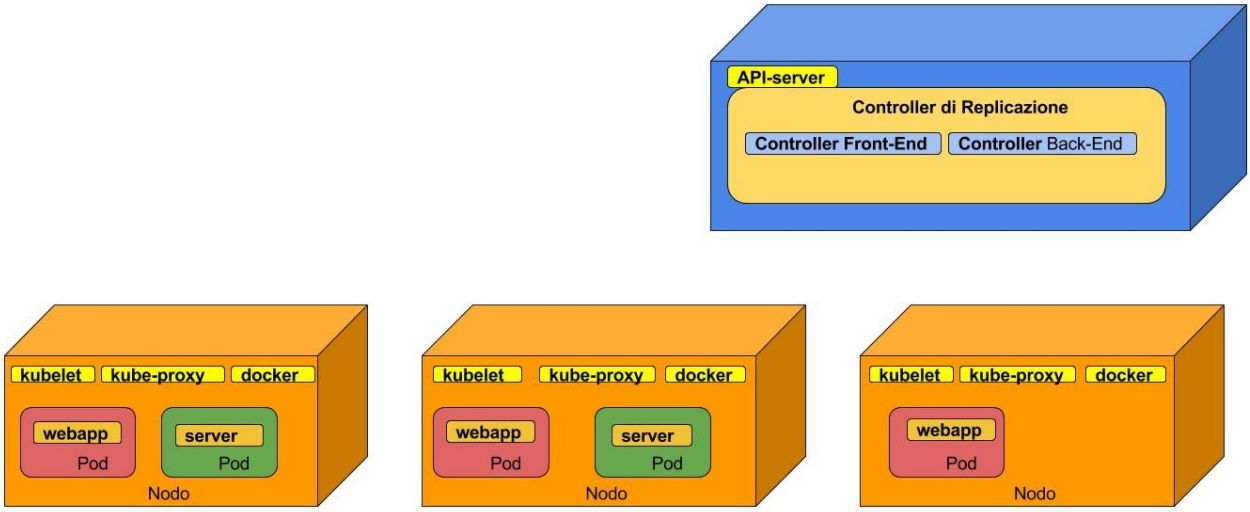

Figura 8 : Fase intermedia della creazione di un cluster Kubernetes

I container non sono pensati per memorizzare uno stato persistente, infatti quando un container va in crash o viene riavviato, il suo filesystem locale viene cancellato. Se c'è la necessità di mantenere questo stato , bisogna utilizzare dei volumi persistenti. Dal momento che ogni cosa in Kubernetes è basata sui pod, i volumi devono essere definiti all'interno dei pod stessi. L'uso di un volume è comunque limitato al nodo che contiene quel determinato pod e sopravvive a crash e riavvii ma non ai fallimenti del nodo o dell'host. É chiaro quindi che bisogna ancora effettuare la duplicazione o il back-up di dati importanti.

Abbiamo quindi fatto in modo che i nostri container siano sotto il controllo di Kubernetes ma dobbiamo ancora renderli accessibili al mondo esterno.

Di default, i pod sono accessibili solamente all'interno del cluster Kubernetes attraverso il proprio IP interno. Per far si che essi siano invece accessibili anche all'esterno della nostra rete virtuale, bisogna configurare i pod come dei servizi . In seguito a questa procedura di "expose" i nostri pod , avranno oltre ad un indirizzo IP che li identifica all'interno del

cluster, anche un IP esterno affinché possano essere raggiunti da host esterni alla nostra rete. Per creare un servizio :

```
kubectl create -f -myservice.yaml
```

Dopo l'esposizione saremo quindi in grado di raggiungere i nostri due servizi di front-end e back-end semplicemente inserendo il loro IP esterno nella barra di ricerca.
Con l'introduzione dei servizi il nostro cluster è completo almeno nelle sue funzionalità di base e si presenta come in figura 9 :

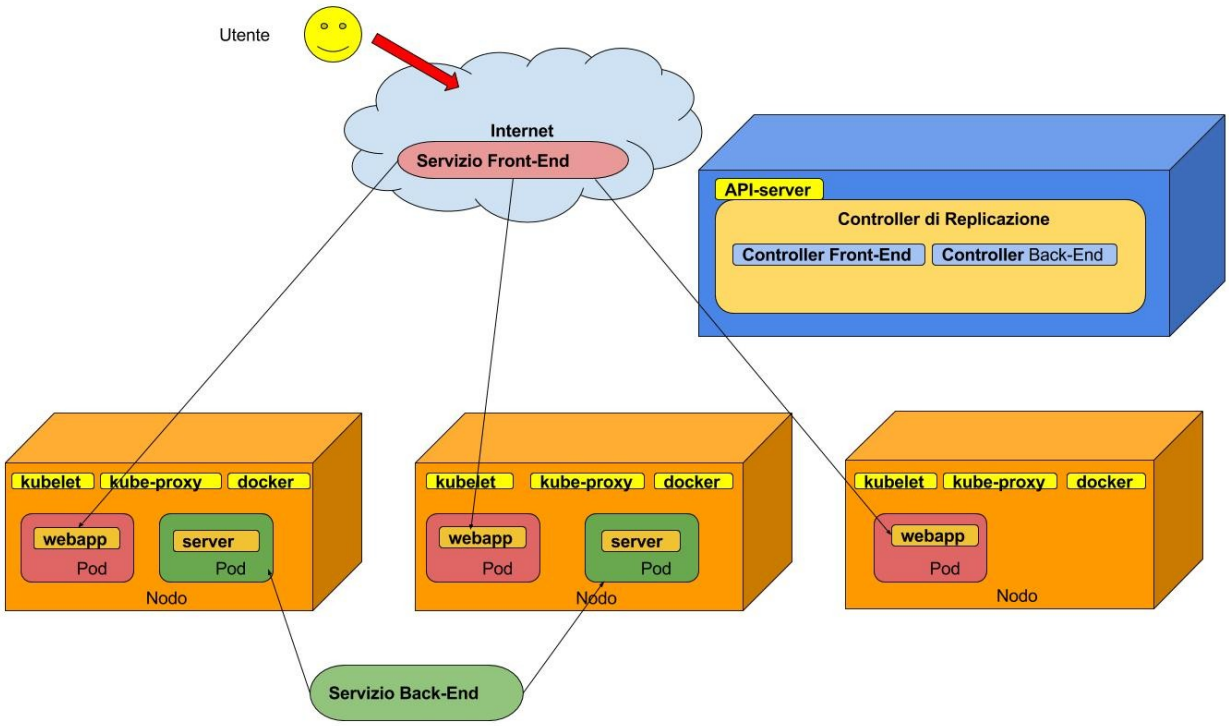

Figura 9 : Fase finale della creazione di un cluster Kubernetes

Se ci si accorga che l'applicazione necessita di più risorse, o magari che essa può essere gestita anche con meno risorse di quelle impiegate , invece di rimuovere l'intero controller di replicazione e i pod associati, Kubernetes offre la possibilità di scalare facilmente l'applicazione. Supponendo di avere due repliche in esecuzione, se volessimo averne altre due per bilanciare il carico di lavoro basterebbe semplicemente scrivere :

```
kubectl get pods -l name=node-js-scale
kubectl scale –replicas=2 rc/node-js-scale
```

Se lo scaling va a buon fine, semplicemente apparirà la parola **scaled** sul nostro terminale. Ogni pod può poi utilizzare uno o più "segreti" per gestire informazioni sensibili come password o credenziali per accedere alla API. Tutti i segreti possono essere elencati tramite :

```
kubectl get secrets
```

Come già detto poi Kubernetes permette di monitorare in tempo reale modifiche alle risorse. Tutte queste modifiche sono accessibili tramite gli eventi :

```
kubectl get events
```

Infine per separare utenti, gruppi o applicazioni si utilizzano i namespace. Per creare un nuovo namespace diverso da quelli di default dopo aver creato un apposito file di configurazione basta eseguire :

```
kubectl create -f ./my-namespace.yaml
```

Conclusioni

Kubernetes si rivela essere uno strumento molto valido per la gestione del cluster e la schedulazione delle operazioni, due attività che costituiscono una parte chiave dell'implementazione di servizi containerizzati su insiemi di host distribuiti. In particolare utilizzando questi strumenti di gestione dei container numerosi sono i vantaggi sia per gli amministratori di sistema, sia per gli utenti finali del servizio. Grazie poi al rapido sviluppo che sta attraversando la tecnologia dei container a discapito delle macchine virtuali, saranno sempre più numerosi i casi d'uso cui questa tecnologia si presterà andando a sancire il definitivo salto in avanti di questi cosiddetti schedulatori di container. Tutto questo grazie a numerose aziende , Google in primis, che da sempre hanno supportato questo progetto e che stanno estendendo i propri servizi cloud affinché supportino nativamente i container.

Bibliografia

[1] Omer Dawelbeit http://omerio.com/2016/01/02/getting-started-with-kubernetes-on-google-container-engine/

[2] Jonathan Baier, Getting Started with Kubernetes, Packt Publishing, 2015.

[3] Kamal Marhubi, http://kamalmarhubi.com/blog/2015/08/27/what-even-is-a-kubelet/

[4] Kubernetes http://kubernetes.io/docs

[5] CloudTalk http://www.cloudtalk.it/container-cosa-sono-come-funzionano

[6] CWIpedia http://www.cwi.it/container-cosa-funzionano-virtualizzazione_86516,

[7] Nune Isabekya http://x-team.com/2016/07/introduction-kubernetes-architecture

[8] Antonio Dini http://www.corrierecomunicazioni.it/tlc/34884_il-futuro-del-cloud-e-nei-container.htm

[9] CloudTalk http://www.cloudtalk.it/container-docker-kubernetes/

[10] The Docker Ecosystem : Scheduling and Orchestration https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-scheduling-and-orchestration

[11] Armand Grillet, Comparison of container scheduler https://medium.com/@ArmandGrillet/comparison-of-container-schedulers-c427f4f7421#.c62744hdo

[12]DigitalOcean https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images

[13] Docker https://docs.docker.com/

[14] CoreOS https://coreos.com/kubernetes/docs/latest/services.html

[15] Kubernetes Namespace http://kubernetes.io/docs/admin/namespaces/

[16] CloudTalk http://www.cloudtalk.it/i-container-sono-maturi-per-il-cloud-a-sostenerlo-rackspace

[17]Github https://github.com/kubernetes/kubernetes/blob/master/docs/design/networking.md

[18] Openshift https://docs.openshift.com/enterprise/3.1/dev_guide/pod_autoscaling.html

[19]http://www.slideshare.net/SatnamSingh67/2015-0605-cluster-management-with-kubernetes

[20] Wikipedia https://it.wikipedia.org/wiki/Virtualizzazione

[21] Vineet Badola http://cloudacademy.com/blog/container-virtualization/