SBDIO I4.0
Big Data
for Industry

O1.5

Prototipo finale del laboratorio dimostrativo federato SBDIO I4.0 (secondo ciclo di sviluppo)

T3LAB

Code	O1.5
Data	30-06-2021
Type	Confidential
Participants	T3LAB
Authors	Luca Padovan
Corresponding authors	Mirko Falavigna

POR FESR EMILIA-ROMAGNA 2014/2020
FSC Fondo per lo Sviluppo e la Coesione
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
Regione Emilia-Romagna

SBDIO I4.0
Big Data
for Industry

Sommario

1 Obiettivi

Gli obiettivi di questo secondo ciclo di sviluppo erano i seguenti:

- realizzare un cloud Openstack che consentisse ai vari partner di progetto di hostare le applicazioni realizzate durante lo svolgimento del progetto, in modo tale da renderle disponibili alle aziende;
- sperimentare nell'ambito dell'offerta IaaS del laboratorio SBDIO I4.0 la possibilità di utilizzare architetture a container, ed in particolare l'utilizzo dell'orchestratore di container Kubernetes;
- definire un modello di interazione con i laboratori che si occupano dei servizi applicativi nel momento in cui un servizio SaaS dovesse essere offerto tramite infrastruttura Kubernetes;
- implementare la possibilità di effettuare un SSO (Single Sign On), cioè la possibilità di effettuare il processo di autenticazione ai vari cloud federati tramite un Identity Provider esterno, in modo che un utente potesse accedere a tutti i cloud federati con soltanto una coppia di credenziali;
- implementare la possibilità di migrare macchine virtuali da un cloud all'altro in modo completamente automatizzato da parte di un utente che non abbia conoscenze pregresse della struttura e del funzionamento del cloud Openstack o di programmazione e scripting.

Tutto il progetto è quindi basato sull'utilizzo di Openstack per la realizzazione dei singoli nodi del cloud federato di progetto (in particolare i nodi T3LAB e UniBo) e sull'uso di Kubernetes per il deploy delle applicazioni containerizzate realizzate dai partner di progetto.

Questo documento viene distribuito con una serie di video in allegato. In questi video è possibile vedere in modo dettagliato tutti i passi che il T3LAB ha seguito per realizzare l'infrastruttura del cloud federato, la configurazione dell'autenticazione federata, il deploy di Kubernetes e il processo di migrazione delle macchine virtuali da un cloud all'altro, nonché le procedure di deploy su Kubernetes di alcune delle applicazioni realizzate da altri partner del progetto.

2 Modelli di interazione e ruoli dei laboratori

L'utilizzo di una infrastruttura basata su container ha portato a rivedere le politiche di deployment che erano state definite in sez. 2 di [1].

Il modello precedentemente utilizzato può essere così riassunto:

- T3LAB in quanto gestore di un nodo cloud basato su un insieme di risorse fisiche (T3LAB-IaaS) definisce la configurazione funzionale del nodo (quali servizi di OpenStack devono essere installati su quali macchine fisiche) e installa congruentemente OpenStack su tutte le macchine fisiche coinvolte;
- T3LAB-IaaS in quanto gestore di un nodo cloud crea un tenant al quale è allocato un pool di risorse virtuali che rappresenteranno la base di definizione della sua infrastruttura (ovviamente il nodo cloud potrà ospitare più tenant). Il tenant è creato sulla base di una richiesta di T3LAB-PaaS, che vuole supportare una piattaforma cloud per lo sviluppatore di applicazioni;
- Sulla base delle necessità dello sviluppatore dell'applicazione, T3LAB-PaaS compie due operazioni:

  - definisce nel contesto del tenant una infrastruttura virtuale di computazione (un insieme di macchine e LAN virtuali) compatibile con il pool di risorse allocato al tenant stesso e adeguato per il supporto dell'applicazione finale;
  - sull'infrastruttura così definita installa e configura la piattaforma middleware (emulando una situazione PaaS) sulla quale verrà poi realizzata l'applicazione finale;
- lo sviluppatore, sulla piattaforma middleware, crea e rende disponibile l'applicazione finale (emulando una situazione SaaS).

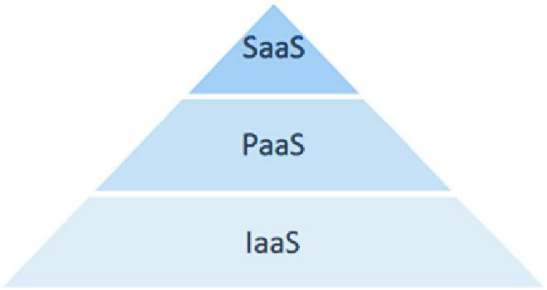

Figura 1 Stack dei possibili servizi cloud

Il modello di interazione utilizzato durante la prima parte del progetto, in cui si sono seguiti tutti i possibili livelli di servizio cloud, non è risultato però conveniente nel momento in cui l'offerta del servizio SaaS è basata su un cluster Kubernetes, anche nel momento in cui Kubernetes stesso è inserito in un contesto cloud/OpenStack.
Gli ultimi due punti del modello di interazione descritto in precedenza sono quindi stati modificati come segue:

- l'utente che richiede i servizi cloud sviluppa l'applicazione o la libreria finale utilizzando la opportuna piattaforma middleware;
- T3LAB-IaaS offre all'utente il servizio di containerizzazione e deployment della sua applicazione su un cluster Kubernetes che si appoggia alle risorse computazionali offerte dal nodo cloud T3LAB. A questo scopo T3LAB-IaaS:
  - definisce nel contesto del tenant una infrastruttura virtuale di computazione (un insieme di macchine e LAN virtuali) compatibile con il pool di risorse allocato al tenant e adeguato per il supporto dell'applicazione finale;
  - installa su questa infrastruttura virtuale di computazione un cluster Kubernetes;
  - effettua la containerizzazione docker dell'applicazione finale
  - effettua il deployment dell'applicazione finale containerizzata sul cluster Kubernetes

Nel nuovo modello di deployment basato su Kubernetes T3LAB non gioca quindi più il ruolo di fornitore di servizi PaaS ma viene a potenziare il suo ruolo di fornitore di sevizi IaaS e assume il ruolo di fornitore dei servizi di containerizzazione e deployment su Kubernetes dell'applicazione finale.

3 Packstack

Come riferito in [1] l'installazione di Openstack sull'infrastruttura fisica è stata originariamente effettuata tramite script manuali.

In questa seconda fase, in accordo e con la collaborazione di CiriICT, per effettuare l'installazione di OpenStack su un nodo (sulle macchine fisiche di un nodo) del cloud federato si è deciso di utilizzare uno strumento più di alto livello quale Packstack. Utilizzando Packstack il deployment di Openstack non avviene tramite script imperativi ma fornendo una descrizione non procedurale (dichiarativa) della topologia fisica del cloud e dei servizi di Openstack che si vogliono installare su ciascuna delle sue macchine. Non solo il lavoro di installazione risulta enormemente semplificato, ma risulta anche possibile raggiungere risultati difficilmente ottenibili tramite script manuali.
L'utilizzo di Packstack facilita anche la gestione dinamica del nodo cloud: risulta ad esempio molto più semplice l'aggiunta di nuove macchine fisiche all'infrastruttura (specificando quali sono i servizi OpenStack che vanno installati su di essa) così come la loro rimozione. Senza dubbio l'impiego di Packstack semplifica l'installazione e la gestione dell'infrastruttura cloud una volta ottenute le informazioni giuste da inserire nel file di configurazione; d'altra parte Packstack pone dei vincoli sulla configurazione della macchine fisiche che compongono il cloud. Packstack, per esempio, è supportato soltanto da macchine fisiche che usano CentOS o RHEL. Inoltre l'answerfile con le configurazioni che desideriamo per il nostro cloud è molto dettagliato e non è sempre di facile comprensione.

4 Kubernetes

L'utilizzo di Kubernetes nel contesto del progetto SBDIO I4.0 è stato studiato inizialmente da CiriICT. In base alle indicazioni di CiriICT T3LAB si è quindi occupato di fare evolvere la propria infrastruttura cloud affinché essa potesse supportare anche cluster Kubernetes.
Questo capitolo è basato su [3]. Kubernetes è un progetto sviluppato inizialmente da Google e reso open-source nel 2014. Lo scopo principale di Kubernetes è quello di facilitare sia la configurazione dichiarativa che l'automazione di applicazioni containerizzate. I principali punti di forza di Kubernetes sono dati dall'utilizzo dei container per il deploy delle applicazioni:

- i container presentano uno modello di isolamento più leggero di quello delle macchine virtuali, garantendo comunque la non interferenza tra container differenti e quindi tra le applicazioni;
- coerenza di ambiente tra sviluppo, test e produzione: i container funzionano allo stesso modo sia su un computer portatile che sulle macchine di un cloud;
- portabilità tra sistemi cloud e sistemi operativi differenti;
- microservizi liberamente combinabili, distribuiti e ad alta scalabilità: le applicazioni sono suddivise in componenti più piccoli e indipendenti tra loro, che possono essere distribuiti e gestiti dinamicamente;
- in un container le risorse sono isolate dagli altri container, ciò consente di prevedere le prestazioni delle applicazioni.

Kubernetes quindi ci permette di creare dei cluster (composti da macchine fisiche e/o virtuali, nel nostro caso macchine virtuali in un ambiente cloud OpenStack) sui quali possiamo effettuare il deploy di applicazioni containerizzate. Inoltre Kubernetes non si limita a gestire singoli container di applicazioni, infatti è un vero e proprio orchestratore di container: si occupa della scalabilità, del failover e della distribuzione dei container che formano le nostre applicazioni.
Le principali funzionalità di Kubernetes sono:

- **Scoperta dei servizi e bilanciamento del carico**: è possibile esporre un container (o un gruppo di container) verso l'esterno usando un nome DNS o un semplice indirizzo IP. Kubernetes è in grado monitorare il traffico verso un determinato container e reindirizzarlo verso altri in modo che il servizio rimanga disponibile e stabile;
- **Rollout e rollback automatici**: possiamo descrivere quale vogliamo che sia lo stato finale desiderato per i nostri container e Kubernetes si occuperà di effettuare le operazioni necessarie per portarli allo stato che abbiamo indicato. Possiamo dire a Kubernetes di creare nuovi container per un nuovo servizio, di aggiornare le immagini di tutti i container di una determinata applicazione e di effettuare un rollback nel caso i cambiamenti non siano andati a buon fine;
- **Ottimizzazione dei carichi**: se forniamo un cluster di nodi sul quale è possibile eseguire i container e indichiamo di quanta CPU e RAM gli stessi hanno bisogno, allora Kubernetes allocherà i container sui nodi in modo tale da ottimizzare l'uso delle risorse;

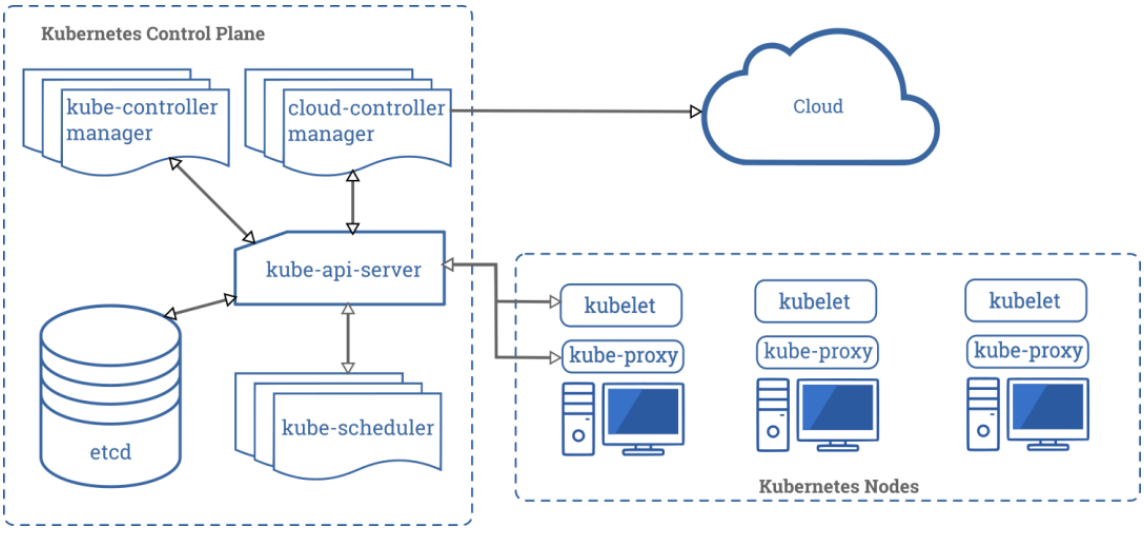

Figura 2 Diagramma di un cluster Kubernetes e delle interazioni dei suoi componenti

- **Self-healing**: Kubernetes elimina i container che si bloccano e li sostituisce, elimina i container che non rispondono agli health checks e evita di far arrivare traffico ai container che non sono pronti per gestire richieste.

4.1 I componenti di Kubernetes

Una volta fatto il deploy di Kubernetes, otteniamo un cluster. Un cluster Kubernetes non è altro che un insieme di macchine (fisiche e/o virtuali) che chiamiamo nodi, che eseguono container gestiti e coordinati da Kubernetes. Ogni cluster deve avere almeno un Worker Node e un nodo (che prende il nome di Master Node) che ospita il Control Plane che si occuperà di coordinare le operazioni del cluster. I Worker Node ospitano i Pod che eseguono i Workload dell'utente (vedi sez. 4.2). Il Control Plane gestisce i Worker Node. In Figura 2 viene riportato un diagramma dei componenti di un cluster Kubernetes e di come interagiscono tra di loro.

4.1.1 Componenti del Control Plane

I componenti del Control Plane sono responsabili delle decisioni che riguardano tutto il cluster:

- **kube-api-server**: espone le Kubernetes APIs. È il frontend del Control Plane di Kubernetes ed è completamente scalabile orizzontalmente (è possibile cioè eseguire più istanze e bilanciare il carico tra esse);
- **etcd**: è un database key-value usato da Kubernetes per salvare tutte le informazioni del cluster;
- **kube-scheduler**: si occupa di assegnare i Pod appena creati ad un nodo del cluster;
- **kube-controller-manager**: si occupa della gestione dei controller. I controller si occupano di supervisionare il cluster e il suo stato. Esistono diversi tipi di controller per diversi scopi, tra cui i più importanti sono il Node Controller che monitora lo stato di salute dei nodi del cluster e il Replication Controller, responsabile per il mantenimento dei corretto numero di Pod per ogni ReplicaSet (il numero di container che vogliamo per ogni componente dell'applicazione) presente nel sistema;
- **cloud-controller-manager**: permette di collegare il cluster alle API del cloud provider e separare quindi le componenti che interagiscono con la piattaforma cloud dai componenti che interagiscono solamente con il cluster.

4.1.2 Componenti dei nodi

I componenti dei nodi vengono eseguiti su ogni nodo del cluster, mantenendo i Pod in esecuzione e fornendo l'ambiente di runtime a Kubernetes:

- **kubelet:** un agent eseguito su ogni nodo del cluster. Si assicura che tutti i container siano eseguiti su un Pod, funzionino correttamente e siano sani;
- **kube-proxy**: gestisce un servizio di proxy su ogni modo che interagisce con le singole sotto reti della macchina host ed espone i servizi al mondo esterno. Esegue anche l'inoltro delle richieste ai Pod destinatari situati nei vari nodi del cluster;
- **Container Runtime**: software responsabile per l'esecuzione dei container. Kubernetes supporta diversi container runtime, anche se quello maggiormente utilizzato è sicuramente Docker (che è quello che è stato utilizzato nel progetto).

4.2 Concetti chiave di Kubernetes

Per poter utilizzare Kubernetes, è necessario comprendere alcuni concetti e astrazioni di base che vengono utilizzati per rappresentare lo stato del sistema e del nostro cluster. Di seguito vengono elencate le più importanti per i nostri scopi.

4.2.1 Pod

I nodi worker del cluster eseguono i Pod. I Pod sono i componenti più semplici e basilari di Kubernetes. Ogni Pod rappresenta una singola istanza di un'applicazione o di un processo in esecuzione su Kubernetes e consiste di uno (solitamente) o più container. Kubernetes si occupa di eseguire, interrompere o replicare tutti i container in un Pod trattandoli come un gruppo.
I Pod situati nei Worker Node vengono dunque creati e distrutti a seconda dello stato desiderato dall'utente, e rappresentano anche l'unità di scaling di Kubernetes: se dovesse servire scalare

orizzontalmente un'applicazione si aggiungono nuovi Pod sui quali verranno eseguiti nuovi container.

4.2.2 Service

Come detto precedentemente, i Pod sono unità effimere e volatili, e quindi non affidabili. Un Pod potrebbe per esempio essere smantellato in qualsiasi momento secondo politiche sia interne che esterne a Kubernetes. Se un Pod dovesse interrompersi, Kubernetes non si occuperà di riportarlo in vita: semplicemente lo distruggerà e ne creerà uno nuovo. Il nuovo Pod potrebbe anche sembrare lo stesso del precedente, ma non è così: avrà infatti un nuovo indirizzo IP ed uno stato differente. Per far fronte a questo problema, Kubernetes introduce il concetto di Service, il cui scopo è quello di fornire (a livello di networking) un punto di accesso stabile ad un insieme volatile di Pod che forniscono uno stesso servizio.
I Service quindi forniscono un frontend stabile costituito da un nome DNS o da un IP ed una porta, bilanciando il carico tra diversi Pod. Inoltre monitorano lo stato di salute di ogni singolo Pod che si ritrovano a gestire, in modo tale da non indirizzare richieste ad un Pod non in grado di servirle.

4.2.3 Deployment

Una delle caratteristiche più importanti che un'applicazione cloud nativa deve garantire è la resilienza.
I Pod, di per sé, non sono in grado di gestire eventuali malfunzionamenti, non sono in grado di scalare autonomamente e non forniscono nessun meccanismo di gestione di aggiornamenti e rollback delle immagini dei container. In Kubernetes questo ruolo viene ricoperto dai Deployment.
Di fatto i Pod vengono messi in esecuzione sempre attraverso dei Deployment, ognuno dei quali è responsabile della gestione di un singolo tipo di Pod (si pensi ad un'applicazione web con backend e frontend, dove dovranno essere utilizzati due tipi di Pod diversi).
A livello pratico, un Deployment descrive lo stato desiderato di un insieme di Pod, basandosi su un file YAML [4], e si assicura che lo stato descritto dal file venga raggiunto da parte dei Pod. Un Deployment per esempio fornisce la descrizione di come è strutturata un'applicazione a livello di container, specificando le regole di strutturazione della stessa e la politica di replicazione dei Pod.

5 Cloud T3LAB

Come detto precedentemente, il cloud T3LAB è stato realizzato con l'ausilio di PackStack. In Figura 3 è illustrata la sua struttura.

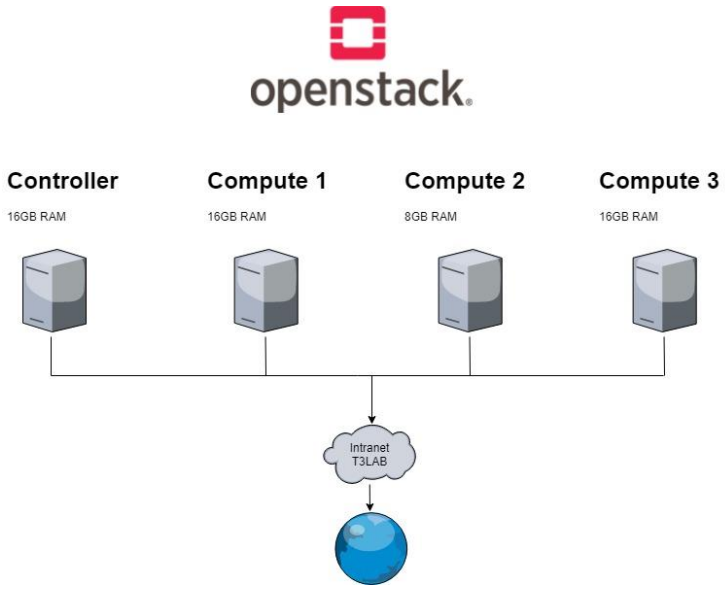

Figura 3 Struttura del cloud OpenStack federato T3LAB

Il cloud è ora composto da 4 nodi fisici, uno che svolge la funzione di Controller (ma anche di Compute) e tre nodi di calcolo. Inoltre i tre nodi di calcolo hanno anche caratteristiche hardware diverse tra di loro: il nodo Compute 3 ha a disposizione una CPU molto più recente rispetto alle altre, oltre che un SSD che ci consente di svolgere in maniera più performante determinate operazioni, soprattutto quelle che hanno a che fare con le tecniche di Machine Learning o Deep Learning. Questa situazione si presenta comunemente nei cloud esistenti: non tutte le macchine che li compongono non hanno tutte le stesse caratteristiche, ma si tratta di un insieme di macchine estremamente eterogeneo. Per una procedura dettagliata del processo di installazione di Openstack tramite Packstack si rimanda ai video 1.mp4 allegato a questo documento.

6 Cluster Kubernetes T3LAB

Una volta realizzata l'infrastruttura del cloud T3LAB, abbiamo effettuato il deploy del cluster Kubernetes. Una procedura dettagliata è visibile nel video 6.mp4 e 7.mp4. Inoltra nel video 8.mp4 è possibile vedere il processo di deployment della dashboard di Kubernetes.
Per realizzare il nostro cluster Kubernetes, abbiamo realizzato tre macchine virtuali su Openstack. A partire da queste VM, abbiamo realizzato il nostro cluster utilizzando uno tool di Kubernetes che prende il nome di *kubeadm*.
Il risultato è un cluster con un nodo master e due worker node, sul quale successivamente abbiamo effettuato il deploy delle applicazioni realizzate dagli altri partner di progetto. La struttura finale del cluster realizzato è visibile in Figura 4.

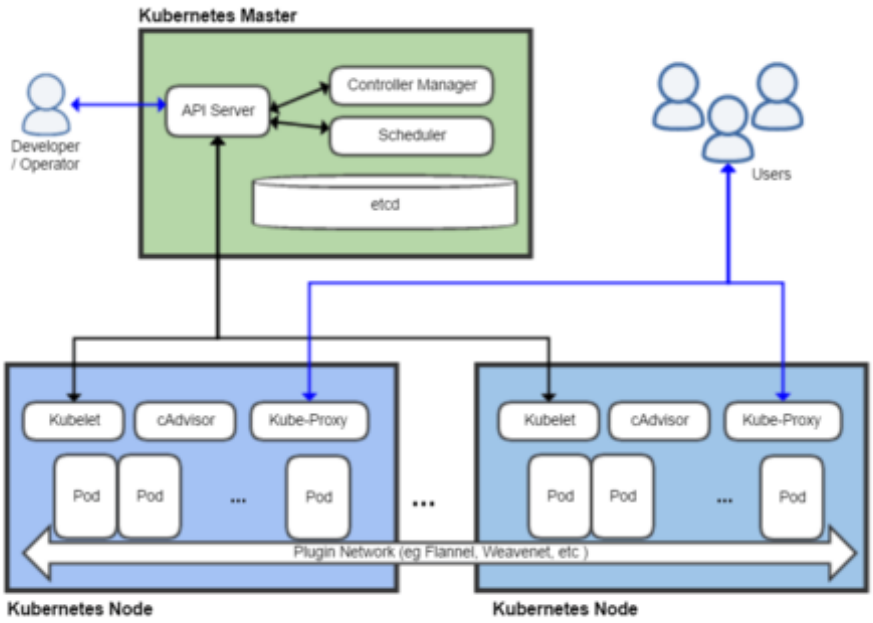

Figura 4 Struttura del cluster Kubernetes T3LAB

7 Autenticazione federata

Uno degli obiettivi principali del progetto SBDIO I4.0 era di consentire un sistema di autenticazione federata tra i vari cloud del progetto. In pratica, si voleva rendere possibile ad un utente accedere a tutti i cloud del progetto utilizzando soltanto una coppia di credenziali, delegando il processo di autenticazione ad un Identity Provider (IdP) esterno.

Per fare ciò ci siamo appoggiati ad un IdP che prende il nome di JumpCloud [13].

JumpCloud è un servizio di Directory as a Service che consente la gestione degli utenti e i loro sistemi Windows, Mac o Linux fornendo accesso a risorse locali, cloud o ad applicazioni come Microsoft 365 e Google Workspace.

Permette di implementare SSO (Single Sign On), scadenza password centralizzata, LDAP, RADIUS, per assolvere con semplicità e sicurezza a tutti gli obblighi previsti, tra l'altro, dal GDPR.

Questo IdP può essere utilizzato gratuitamente fino a 10 utenti. Ci consente di creare diversi account che vengono associati a dei gruppi distinti. Questi gruppi possono poi essere associati a diversi Service Provider (SP). La funzionalità di SSO, come abbiamo visto, non è l'unica funzionalità offerta da JumpCloud, ma è l'unica che è stata presa in considerazione per la realizzazione di questo progetto.

Un'altra tecnologia che abbiamo utilizzato lato cloud Openstack per implementare questa funzionalità è Shibboleth. Shibboleth è un sistema di SSO (login) per reti informatiche. Consente di autenticarsi su sistemi differenti, permettendo di effettuare il login su reti di organizzazioni o istituzioni diverse, utilizzando una sola identità.

Shibboleth è stato sviluppato dal consorzio no-profit Internet2, che ha creato un'architettura e un'implementazione open source per l'Identity Management e l'autenticazione con identità federata. L'infrastruttura è basata sullo standard aperto Security Assertion Markup Language

(SAML). Shibboleth è open source ed è rilasciato con licenza Apache 2. Nel video 10.mp4 è possibile vedere i passi seguiti per configurare l'autenticazione federata con JumpCloud.

8 Migrazione di macchine virtuali

Un altro obiettivo che il progetto voleva raggiungere è quello di consentire ad un utente di migrare macchine virtuali da un cloud Openstack ad un altro. Inoltre questa operazione non doveva presupporre nessuna conoscenza da parte dell'utente che avrebbe lanciato la migrazione. Per raggiungere questo obiettivo abbiamo scritto diverse componenti che collaborano tra loro e utilizzato diverse tecnologie.
Prima di entrare nel dettaglio delle applicazioni che abbiamo realizzato è necessario però illustrare brevemente cosa intendiamo per migrazione di una macchina virtuale. Per migrazione si intende appunto lo spostamento di uno o più macchine virtuali da un cloud ad un altro, senza la perdita di dati, informazioni o di funzionalità dell'applicazione (o applicazioni) che la macchina virtuale eventualmente ospita. Sul cloud sorgente ovviamente non devono restare tracce della macchina virtuale che abbiamo appena migrato, in modo da non consumare inutilmente risorse che potrebbero essere destinate ad altri.

Questo risultato si ottiene in diversi passi:

- Creare uno snapshot della macchina virtuale che vogliamo migrare;
- Eliminare la macchina virtuale dal cloud sorgente;
- Scaricare localmente lo snapshot che abbiamo appena creato e una volta fatto eliminare lo snapshot dal cloud sorgente;
- Caricare sul cloud destinazione lo snapshot, creando un'immagine;
- Creare sul cloud destinazione una nuova instanza di macchina virtuale con lo snapshot che abbiamo caricato con le impostazioni corrette.

Questi passi vanno ripetuti tante volte quante sono le macchine virtuali che vogliamo creare.
Chiarito cosa significa migrare una macchina virtuale possiamo passare ad illustrare i vari applicativi che abbiamo realizzato, e spiegare come collaborano tra loro: l'infrastruttura che consente la migrazione è suddivisa sostanzialmente in quattro macro componenti. Il primo, che si occupa di interagire direttamente con i cloud openstack, è composto da un insieme di script che effettuano effettivamente la migrazione di una macchina virtuale. Questi script vengono eseguiti dal secondo componente, che non è altro che un backend che espone delle API. Le API del backend vengono chiamate da due diversi frontend, uno web e uno che consiste in un'applicazione mobile, per consentire ad un utente di effettuare una migrazione comodamente dal proprio smartphone. Nei prossimi paragrafi parleremo in dettaglio delle singole componenti. In Figura 5 viene mostrato lo

schema di interazione tra le diverse componenti che gestiscono la migrazione delle macchine virtuali.
Nel video 11.mp4 è possibile vedere queste componenti all'opera mentre viene effettuata una migrazione.

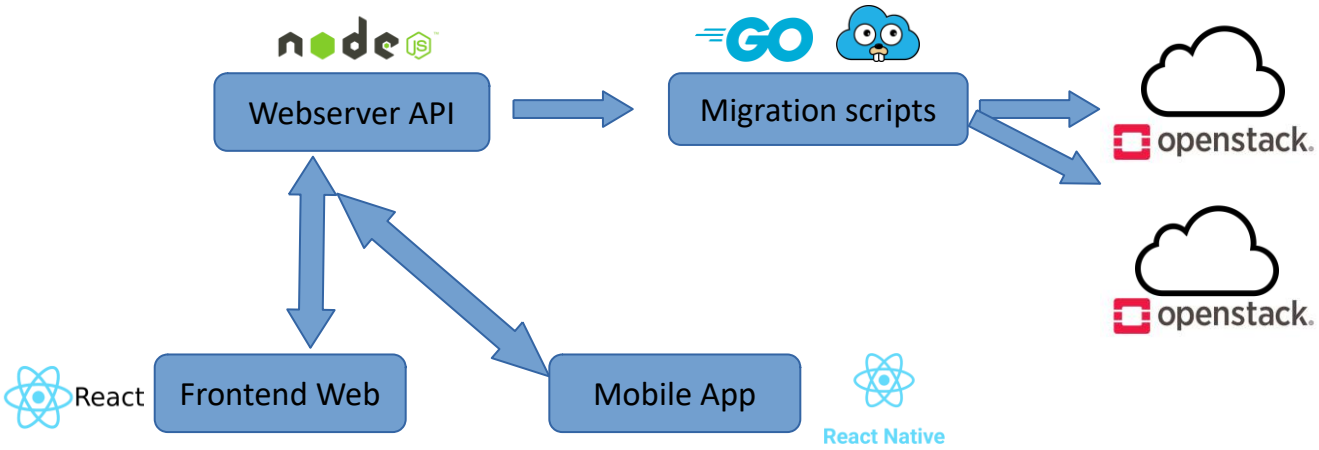

Figura 5: Schema di interazione tra i componenti della migrazione

8.1 Migration scripts

I migration script sono un insieme di script scritti in GO che utilizzano la libreria GopherCloud. Vengono eseguiti dal backend NodeJS e interagiscono direttamente con i vari cloud coinvolti nell'operazione di migrazione. Abbiamo scelto di separare questi due componenti per sfruttare da una parte la potenza di NodeJS nel realizzare backend in javascript, dall'altra per sfruttare la potenza e la flessibilità della libreria Gophercloud che è stata scritta appositamente per GO.

8.2 Webserver API

Il Webserver altro non è che un componente scritto in NodeJS e ExpressJS che espone delle API che i vari frontend possono chiamare per effettuare delle migrazioni o per avere informazioni sui vari cloud presenti nel sistema. Espone API che tramite autenticazione consentono di inserire, modificare o eliminare cloud dal sistema, avviare migrazioni e ottenere lo stato di eventuali migrazioni in corso.

8.3 Frontend Web

Il frontend web è stato realizzato in ReactJS e ha il solo compito di interfacciarsi con l'utente. Una volta fatto ciò si occuperà di chiamare le API che il backend espone per avviare la migrazione. Inoltre informerà l'utente sullo stato corrente della migrazione in corso.

8.4 Mobile App

Il frontend mobile è stato scritto con il framework React Native ed ha sostanzialmente gli stessi compiti del frontend web. Avendo utilizzato react native il frontend mobile e' potenzialmente disponibile sia per dispositivi iOS che per dispositivi Android.

9 Altre funzionalità di Openstack e Kubernetes utilizzate

Durante lo svolgimento del progetto ci siamo imbattuti in diverse funzionalità interessanti di Openstack e Kubernetes che abbiamo pensato valesse la pena approfondire, di seguito riportiamo le principali.

9.1 Host Aggregates

Gli Host Aggregates sono una funzionalità che Openstack mette a disposizione degli amministratori del cloud. Sono un meccanismo che consente di partizionare gli host di un cloud Openstack, basandosi su caratteristiche arbitrarie. Ad esempio un amministratore potrebbe voler creare un gruppo di host sul quale creare macchine virtuali con caratteristiche hardware addizionali o performance superiori ad altri. Gli host aggregates sono completamente trasparenti agli utenti finali, è l'amministratore che, tramite un meccanismo di labeling, deve associare dei flavor a degli specifici host aggregates. Lo scheduler di Openstack successivamente proverà a creare le macchine virtuali con un determinato flavor su un host aggregates che soddisfi le label di quel flavor. Gli host inoltre possono far parte di più di un host aggregate.
Questo meccaniscmo ci è tornato particolarmente utile durante il deploy dell'applicazione Anomaly Detection di UniMoRe: tali librerie utilizzavano TensorFlow, che richiede che la CPU del PC host supportasse il set di istruzioni AVX o AVX2. Tramite il meccanismo degli host aggregates siamo riusciti a fare in modo che tale applicazione fosse deployata su un host fisico la cui CPU supportasse questo set di istruzioni.

9.2 Stack

La gestione degli Stack tramite il servizio Heat di Opestack è un'altra funzionalità che abbiamo approfondito durante il progetto, anche se non è stata utilizzata direttamente durante lo svolgimento dello stesso. Lo scopo di questo servizio è di orchestrare le cloud applications. In pratica, consente di creare, sia a mano che tramite un comodo editor visuale) dei file chiamati templates in formato YAML che descrivono un'infrastruttura (composta da macchine virtuali, reti, volumi e sostanzialmente tutti gli oggetti che Openstack è in grado di gestire) che può essere successivamente passata a Heat. Una volta ricevuto il file, Heat si occuperà di realizzare sul cloud l'infrastruttura descritta dallo stesso.

In pratica, grazie ad Heat, è possibile replicare a partire da un file un'intera infrastruttura su un cloud, senza dover intervenire manualmente. Questa funzionalità è sicuramente molto utile nel momento in cui si dovesse migrare un'infrastruttura molto complessa tra diversi cloud.

9.3 Node Affinity

La Node Affinity, insieme ad altri meccanismi che non saranno approfonditi in questo documento, è un meccanismo di Kubernetes che consente di assegnare dei Pods a specifici nodi del cluster. In pratica è possibile vincolare un pod in modo che possa essere eseguito solo su un particolare insiemi di nodi del cluster. Esistono diversi modi per eseguire questa operazione e tutti gli approcci consigliati utilizzano selettori di etichette per facilitare la selezione. Generalmente tali vincoli non sono necessari, in quanto lo scheduler effettuerà automaticamente un posizionamento ragionevole (ad esempio distribuirà i pod tra i nodi in modo da non posizionare il pod su un nodo con risorse libere insufficienti) ma ci sono alcune circostanze in cui vorremmo voler controllare su quale nodo viene distribuito il pod, ad esempio per garantire che un pod finisca su una macchina con un SSD collegato o per collocare insieme i pod di due servizi diversi che comunicano spesso tra di loro. Il meccanismo è molto semplice: a un determinato nodo vengono associate delle coppie chiave-valore. Nel file di deployment del pod vengono inserite delle indicazioni per indicare a Kubernetes che quel pod deve essere ospitato da un nodo del cluster che abbia un determinato valore per una determinata chiave.
Durante il progetto abbiamo utilizzato le node affinity in combinazione con gli host aggregates di Openstack per il deploy dell'applicazione Anomaly Detection di UniMoRe.

10 Bibliografia

[1] T3LAB, "SBDIO I4.0 - O1.1".

[2] T3LAB, "SBDIO I4.0 - O1.2".

[3] «Kubernetes Documentation | Kubernetes» [Online]. Available: https://kubernetes.io/it/docs/home/.

[4] «The Official YAML Web Site » [Online]. Available: https://yaml.org

[5] «Communicate Between Containers in the Same Pod Using a Shared Volume» [Online]. Available: https://kubernetes.io/docs/tasks/access-application-cluster/communicate-containers-same-pod-shared-volume/

[6] https://wiki.openstack.org/wiki/Packstack.

[7] «OpenStack Docs: OpenStack Compute (Nova),» [Online]. Available: https://docs.OpenStack.org/nova/latest/.

[8] «OpenStack Docs: Welcome to Glance's documentation!,» [Online]. Available: https://docs.OpenStack.org/glance/latest/.

[9] «OpenStack Docs: Keystone, the OpenStack Identity Service,» [Online]. Available: https://docs.OpenStack.org/keystone/latest/.

[10] «OpenStack Docs: Horizon: The OpenStack Dashboard Project,» [Online]. Available: https://docs.OpenStack.org/horizon/latest/.

[11] «OpenStack Docs: Welcome to Neutron's documentation!,» [Online]. Available: https://docs.OpenStack.org/neutron/latest/.

[12] «OpenStack Docs: OpenStack Block Storage (Cinder) documentation,» [Online]. Available: https://docs.OpenStack.org/cinder/latest/.

[13] «Active Directory and LDAP Reimagined - JumpCloud,» [Online]. Available: https://jumpcloud.com.