O1.2

Prototipo parziale del laboratorio dimostrativo federato di SBDIOI40 (primo ciclo di sviluppo)

Code	O1.2
Data	24-07-2020
Type	Confidential
Participants	T3LAB
Authors	Luca Padovan
Corresponding authors	Claudio Salati

Indice

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

1 Obiettivi

Gli obiettivi di questa seconda parte del progetto sono i seguenti:

1. valutare la possibilità di gestire in modo più conveniente l'infrastruttura fisica di un nodo del cloud federato. In particolare, questo dovrà consentire di gestire più facilmente l'aggiunta e la rimozione di macchine fisiche ;
2. introdurre nell'ambito dell'offerta IaaS del laboratorio SBDIOI40 la possibilità di utilizzare architetture a container, ed in particolare Kubernetes.
   L'obiettivo è di ottenere un servizio IaaS capace di supportare scaling e load balancing delle applicazioni che si appoggiano all'infrastruttura.
   L'infrastruttura (il cluster Kubernetes) deve anche risultare facilmente espandibile.
3. definire un modello di interazione con i laboratori che si occupano dello sviluppo dei servizi applicativi nel momento in cui un servizio SaaS dovesse essere offerto tramite una infrastruttura Kubernetes.
4. sperimentare questo modello di interazione.
5. sperimentare la capacità di Kubernetes di supportare scaling e load balancing per un servizio SaaS supportato oltre che la possibilità di modificare dinamicamente l'insieme delle risorse di calcolo utilizzate

Tutto il progetto è 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 la gestione dinamica delle risorse di un tenant.

2 Packstack

Come riferito in [1] l'installazione di Openstack sull'infrastruttura fisica è stata originariamente effettuata tramite script manuali.
In questa seconda fase, e in accordo e cin 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.
Il deployment originale di OpenStack prevedeva ad esempio che le macchine fisiche del nodo cloud dovessero essere interconnesse tra loro tramite 2 reti Ethernet fisiche (vedi ***Figura 1***) mentre operando con Packstack è stato facile rimuovere il vincolo ed utilizzare una topologia fisica basata sull'utilizzo di una sola Ethernet (le macchine non necessitano quindi più di essere dual-homed, vedi ***Figura 2***).
Figura 2 illustra anche come Packstack debba essere installato solo sulla macchina Controller del cloud.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

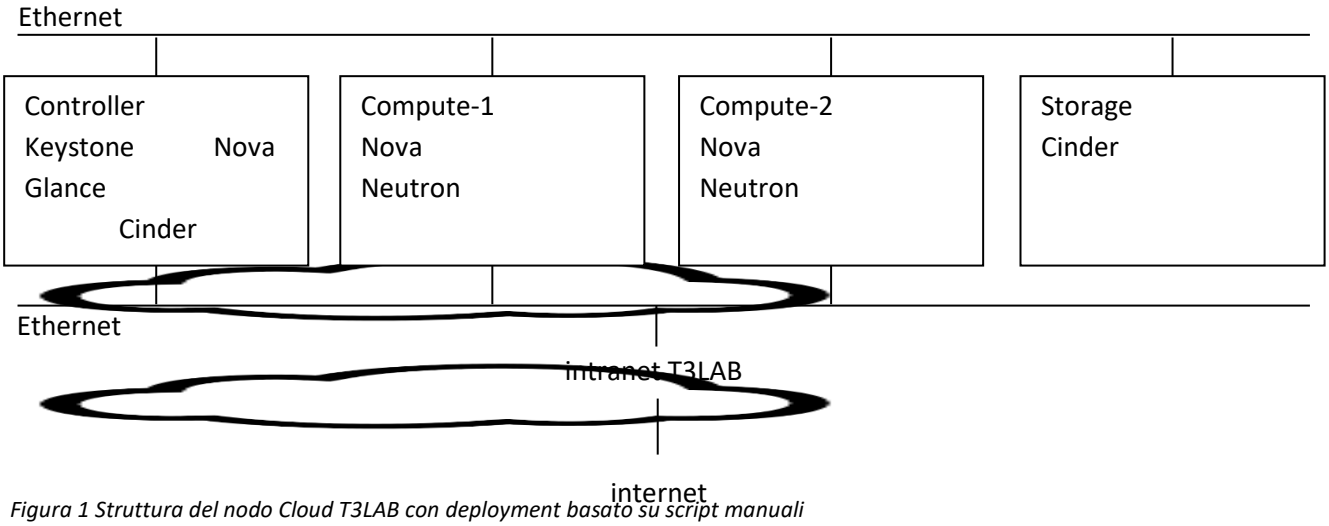

Figura 1 Struttura del nodo Cloud T3LAB con deployment basato su script manuali

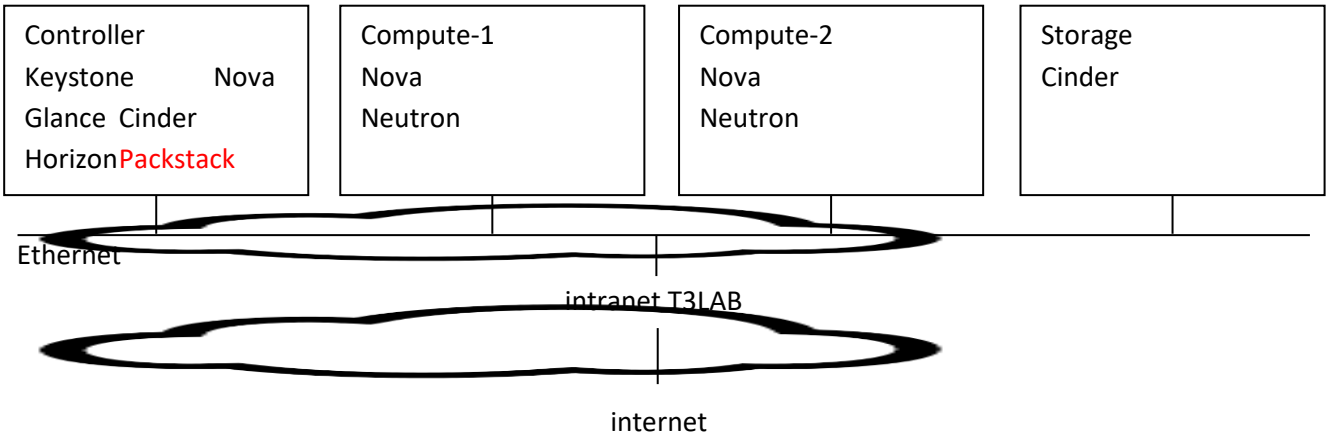

Figura 2 Struttura del nodo Cloud T3LAB con deployment basato su Packstack

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'answer file con le configurazioni che desideriamo per il nostro cloud è molto dettagliato e non è sempre di facile comprensione. Questa situazione si aggrava ulteriormente se consideriamo che la documentazione di Packstack è praticamente inesistente o errata.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

3 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 [2]. 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;

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

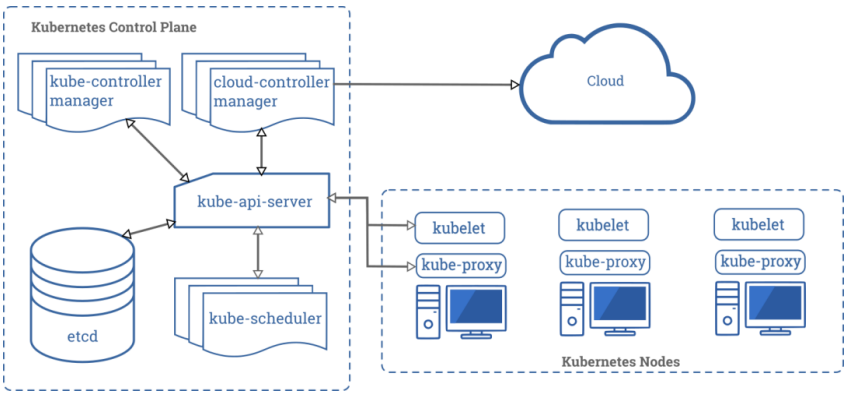

Figura 3 Diagramma di un cluster Kubernetes e delle interazioni dei suoi componenti

- **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.

3.1 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. 3.2). Il Control Plane gestisce i Worker Node. In ***Figura 3*** viene riportato un diagramma dei componenti di un cluster Kubernetes e di come interagiscono tra di loro.

3.1.1 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.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

3.1.2 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).

3.2 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.

3.2.1 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.

3.2.2 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.

3.2.3 Deployment

Una delle caratteristiche più importanti che un'applicazione cloud nativa deve garantire è la resilienza.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

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 [3], 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.

4 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, basandosi sull'esempio legato al supporto dell'applicazione finale in corso di sviluppo presso l'Università di Ferrara:

- 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 UniFe;
- Sulla base delle necessità di UniFe 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, nel caso particolare lo stack ELK [1];
- UniFe, sulla piattaforma middleware, crea e rende disponibile l'applicazione finale (emulando una situazione SaaS).

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

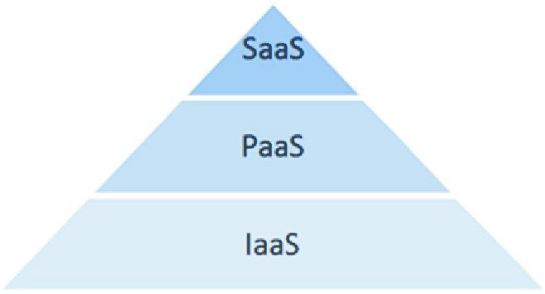

Figura 4 Stack dei possibili servizi cloud

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:

- UniFe sviluppa la applicazione finale utilizzando la opportuna piattaforma middleware;
- T3LAB-IaaS offre a UniFe 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 UniFe
    - effettua il deployment dell'applicazione finale UniFe 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.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

5 Containerizzazione e deployment di applicazioni su Kubernetes

5.1 Struttura del cluster Kubernetes configurato

La struttura del cluster Kubernetes destinato a supportare l'applicazione UniFe è costituita di 3 macchine, nel nostro caso 3 macchine virtuali (VM), così come illustrato nella figura seguente:

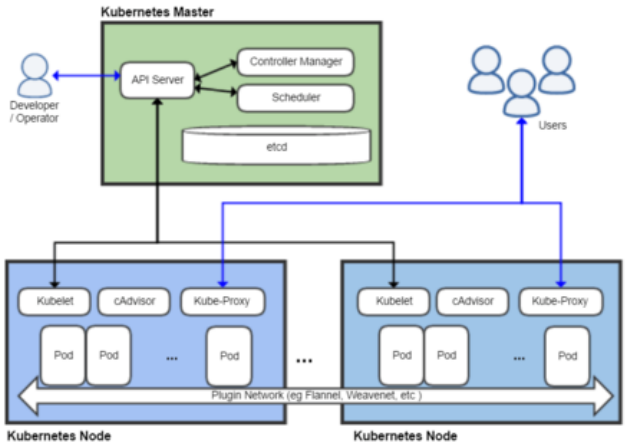

Figura 5 Stack Cluster Kubernetes per l'applicazione finale UniFe

Il deployment di Kubernets, essendo basato sull'utilizzo di VM in una infrastruttura cloud, prevede ovviamente che questa infrastruttura virtuale sia stata precedentemente creata all'interno del tenant OpenStack.

Questa operazione è assolutamente analoga a quanto fatto nella fase 1 del progetto [1] ma in questo caso la creazione dell'infrastruttura è stata effettuata interattivamente tramite la dashboard di OpenStack (servizio Horizon).

5.2 Containerizzazione dell'applicazione UniFe

5.2.1 Strumenti per la containerizzazione

Mentre è possibile effettuale la containerizzazione di una applicazione finale o di una piattaforma middleware creando un opportuno insieme di Dockerfile, è anche possibile recuperare delle containerizzazioni già disponibili in rete per molte delle applicazioni/piattaforme più utilizzate. I repository di container si presentano in due forme:

- repository di Dockerfile e file YAML che definiscono la struttura containerizzata dell'applicazione finale in Kubernetes (Helm offre un repository di questo tipo);
- repository di immagini docker (Docker Hub rappresenta un repository di questo tipo).

Una immagine docker è comunque costruibile a partire da un Dockerfile tramite il suo (docker) build.

L'applicazione UniFe è basata sulla piattaforma middleware Elastic Stack: dato il livello di diffusione di questo piattaforma, essa è disponibile in forma containerizzata nel repository Helm (per i dettagli vedi sez. 5.2.2).

Nota che quello che mette a disposizione il repository Helm è la containerizzazione della sola piattaforma Elastic Stack: ovviamente, per l'applicazione finale, bisogna considerare di includere

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

nella containerizzazione anche gli sviluppi applicativi di UniFe; questo è possibile attraverso tre diverse strategie:

1. modificare i Dockerfile messi a disposizione da Helm (trattandoli quindi come Dockerfile proprietari);
2. arricchendo i container della piattaforma durante l'installazione su Kubernetes;
3. in realtà la containerizzazione fornita da Helm è predisposta per interagire con un'applicazione finale costituita da file di configurazione dello stack Elastic allocati in un volume condiviso tra i container. Ovviamente il deploy dell'applicazione, oltre che effettuare il deloy dello stack Elastic containerizzato, dovrebbe effettuare anche il deloy dei file di configurazione.

L'applicazione UniFe è ancora in corso di sviluppo e questo aspetto non è quindi stato ancora considerato in questa fase del progetto.
Per altre applicazioni finali sviluppate da T3LAB si è invece proceduto attraverso la scrittura diretta di un Dockerfile dedicato.

5.2.2 Containerizzazione dell'applicazione UniFe

La struttura di containerizzazione della piattaforma Elastic Stack utilizzata da Helm coincide in realtà con quella utilizzata da T3LAB: un container per ciascuno dei componenti della piattaforma Logstash (+PostgreSQL), ElasticSearch, Kibana.
Questa struttura è in qualche modo analoga a quella utilizzata in fase 1 del progetto per il deployment della piattaforma su macchine virtuali (una macchina virtuale per ciascun componente).

5.2.3 Il problema della memoria stabile condivisa tra i Pod

Uno dei motivi che portano ad utilizzare architetture basate su container, e quindi Kubernetes, anziché basarsi direttamente su macchine virtuali per il deployment di una applicazione, è legato alla capacità di Kubernetes di supportare lo scaling dell'applicazione stessa.
Questo ovviamente implica che uno stesso container potrà essere istanziato più di una volta (in terminologia Kubernetes più Pod eseguiranno uno stesso container).
Considerando la piattaforma Elastic Stack, in particolare, dobbiamo considerare il caso che più pod eseguano il container ElasticSearch: è ovvio che tutti questi pod dovranno condividere una stessa base dati.
Per risolvere il problema, Kubernetes utilizza i cosiddetti volumi condivisi [4]. In pratica Kubernetes definisce dei volumi per i Pod, e li monta ad uno specifico path all'interno di ogni container che lo richiede. Normalmente il ciclo di vita di un generico volume è legato al ciclo di vita del Pod che lo utilizza, in questi casi invece il volume è indipendente dal Pod a cui è collegato, ottenendo così un volume persistente che può essere condiviso e riutilizzato tra diversi Pod.

5.3 Sviluppo e containerizzazione dell'applicazione UniMoRe

Questa parte del progetto è frutto della collaborazione tra T3LAB, Dmitrij David Padalino Montenero di CIRI-ICT e Francesco Del Buono di UniMoRe. L'obiettivo è stato quello di realizzare la prima iterazione della piattaforma applicativa cloud-native di SBDIO I4.0 per fornire servizi di analisi as-a-service orientati all'Industria 4.0. L'applicazione è composta da due microservizi:

- Il Front-End, scritto in React a cura del CIRI-ICT di Bologna;

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

- Il Back-End sviluppato tramite il framework Django da parte del T3LAB.

I servizi offerti dalla piattaforma sono stati sviluppati da UniMoRe e consistono in alcuni modelli di Machine Learning per eseguire inferenze su dati in forma di serie temporali che consentono di riconoscere variazioni nei dati specificando l'istante temporale dell'inserimento dei settings e andando a individuare l'istante di tempo in cui il sistema raggiunge la condizione di equilibrio. Il Back-End di T3LAB espone una serie di API per richiamare l'esecuzione di tali inferenze, mentre il Front-End realizzato da CIRI-ICT espone un'interfaccia utente, invoca queste API con gli opportuni parametri di ingresso e consente di visualizzare e consultare i dati in ingresso e in uscita.
Una volta realizzati e containerizzati sia Front-End che Back-End, CIRI-ICT ha redatto i file YAML che descrivono i Deployment di Kubernetes per l'intera applicazione. In questo caso, il Deployment consiste nella creazione di due container per il Front-End e di uno per il Back-End, oltre ad un servizio per l'esposizione del Front-End e del Back-End al di fuori del cluster. In
Figura ***6*** viene riportato uno screenshot dell'applicazione finale.

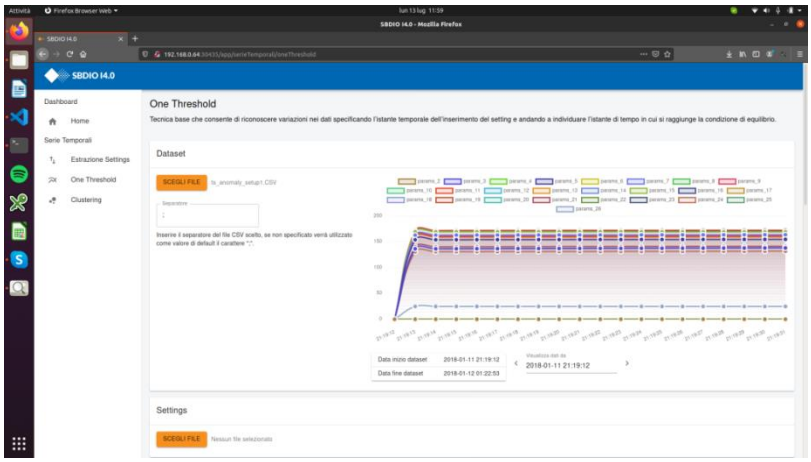

Figura 6 Screenshot dell'applicazione realizzata tramite una collaborazione con UniMoRe-CiriICT-T3LAB

5.4 Containerizzazione di altre applicazioni T3LAB

Come già detto, durante questa fase del progetto è stata anche effettuata, per prova, la containerizzazione manuale di due applicazioni sviluppate in T3LAB.
L'operazione ha comportato due passi:

- creazione del Dockerfile dell'applicazione;
- creazione dell'immagine docker dell'applicazione (comando build).

Per rendere le immagini docker più direttamente accessibili in fase di installazione su Kubernetes esse sono state versate nel repository Docker Hub.
Entrambe queste applicazioni sono semplici e non si basano sull'utilizzo di piattaforma middleware. Sono mappate ciascuna in un container.
Mentre la prima applicazione, costituita da un sito web per la visualizzazioni di informazioni relative alla pandemia COVID-19, è stata realizzata come un esercizio a sé stante, la seconda, costituita da un semplice webserver che risponde a query predefinite, è stata sviluppata per sperimentare le capacità di gestione di scaling e load balancing di Kubernetes.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

5.5 Deployment di applicazioni containerizzate su Kubernetes

5.5.1 Strumenti per il deployment su Kubernetes

Anche il deployment di una applicazione containerizzata su Kubernetes può essere effettuato in diversi modi:

1. attraverso script di kubectl (la command line interface di Kubernetes);
2. attraverso script del package manager per Kubernetes Helm (detti chart);
3. utilizzando la piattaforma Rancher per la gestione di Kubernetes (che a sua volta può appoggiarsi ad esempio a Helm).

5.5.2 Deployment su Kubernetes dell'applicazione UniFe

Il deployment dell'applicazione UniFe (in realtà della piattaforma Elastic Stack che la supporta) è stato fatto utilizzando Rancher.
Tra le indicazioni date a Kubernetes in questa fase tramite un Deployment c'è il numero massimo di pod attivabili per l'esecuzione di ciascun container che costituisce l'applicazione, nel nostro caso:

- 1 per il container Logstash,
- 3 per il container ElasticSearch,
- 1 per il container Kibana.

Utilizzando Rancher è anche facile modificare questi valori durante la vita dell'applicazione, così come i parametri fondamentali di Kubernetes stesso (e.g. il numero delle macchine slave che fanno parte del cluster Kubernetes).

5.5.3 Deployment su Kubernetes dell'applicazione UniMoRe

Il deployment del prototipo della piattaforma applicativa cloud-native di SBDIO I4.0 è stato effettuato attraverso degli script realizzati dal CIRI-ICT. In particolare sono stati creati degli script per il rilascio e la rimozione dell'applicazione da Kubernetes, sfruttando lo strumento kubectl e fornendo a quest'ultimo i file YAML con le configurazione desiderata per la nostra applicazione.

5.5.4 Deployment su Kubernetes di altre applicazioni T3LAB

Rancher è stato utilizzato anche per il deployment dell'applicazione sito web-COVID-19 mentre per dell'applicazione webserver si sono utilizzato uno script kubectl.
In ogni modo, in questi due casi l'installazione delle applicazioni è stata fatta a aprtire da immagini docker presenti su Docker Hub, mentre nel caso dell'applicazione UniFe l'installazione è stata effettuata a partire da Dockerfile presenti nel repository Helm.

6 Sperimentazione di scaling e load balancing

Una delle applicazioni di cui è stato effettuato il deploy in Kubernetes è costituita semplicemente da un web server che risponde alle richieste che riceve indicando l'identificativo del Pod (stringa alfanumerica) sul quale viene processata la richiesta stessa, con un ritardo fissato (un secondo, nel nostro caso) tramite una semplice istruzione di sleep. Lo scopo di questa applicazione è di verificare le potenzialità dello scaling e del load balancing dei Pods di Kubernetes.
Per verificare il funzionamento del load balancing (effettuato automaticamente da Kubernetes tra i diversi container di una stessa applicazione) basta verificare che a fronte di diverse richieste al web server, la risposta proviene sempre da un web server ospitato su un Pod differente (verificabile analizzando la risposta del web server, che indica l'ID del Pod). Siamo stati in grado di verificare e confermare questo comportamento da parte di Kubernetes.
Per quello che riguarda lo scaling, per verificarne il funzionamento ci siamo appoggiati a un tool open source di load testing, Locust [5]. Locust e' un framework scritto in Python che ci consente di definire il comportamento di un generico utente (le richieste che esegue, a quali URL, il tempo che intercorre tra una chiamata e l'altra) e di simularne un determinato numero, specificando anche il loro tasso di nascita.
Una volta lanciato il test e modificando in tempo reale tramite kubectl il numero di repliche del nostro web server, possiamo consultare diversi indici statistici, tra cui i più importanti sono:

- il totale delle richieste fatte,
- il numero dei fallimenti,
- il tempo medio di risposta e
- il numero di richieste per secondo eseguite.

In Figura 7 vengono riportati i dati relativi alle richieste per secondo eseguite con successo (in verde, da ora in avanti RPM) da Locust. Iniziando con un ReplicaSet pari a 1, RPM si attesta intorno a 1, mentre aumentando il numero di ReplicaSet (prima portandolo a 10, e successivamente a 25) cresce fino a stabilizzarsi nuovamente. Interessante notare che in prossimità dello scaling dell'applicazione, il numero di richieste fallite aumenta momentaneamente, segno che potrebbero esserci delle momentanee e brevi interruzioni del servizio quando Kubernetes scala un'applicazione.

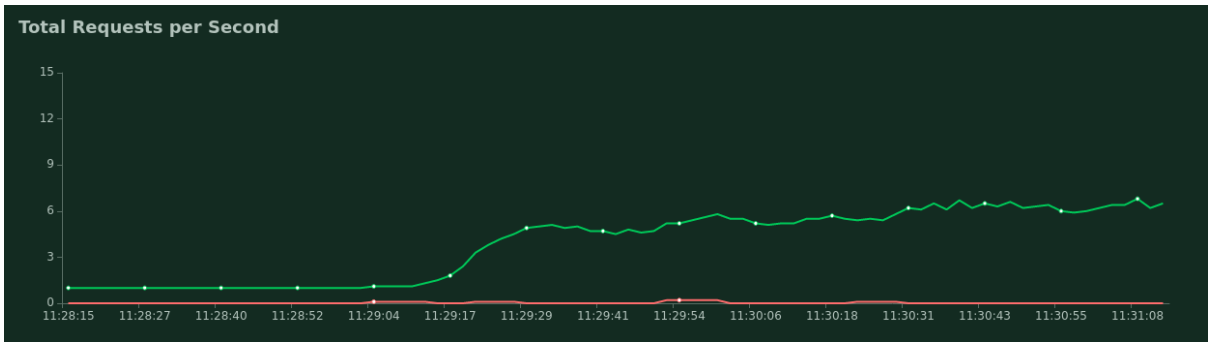

Figura 7 Richieste per secondo (in verde) e fallimenti per secondo (in rosso) rilevati da Locust durante il test

In Figura 8 invece viene riportato il tempo medio di risposta in millisecondi (in verde) e il valore del 95esimo percentile sempre in millisecondi (in giallo). Anche in questo caso possiamo vedere come il tempo medio di risposta si abbassi all'aumentare del numero di Pod che eseguono il web server.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

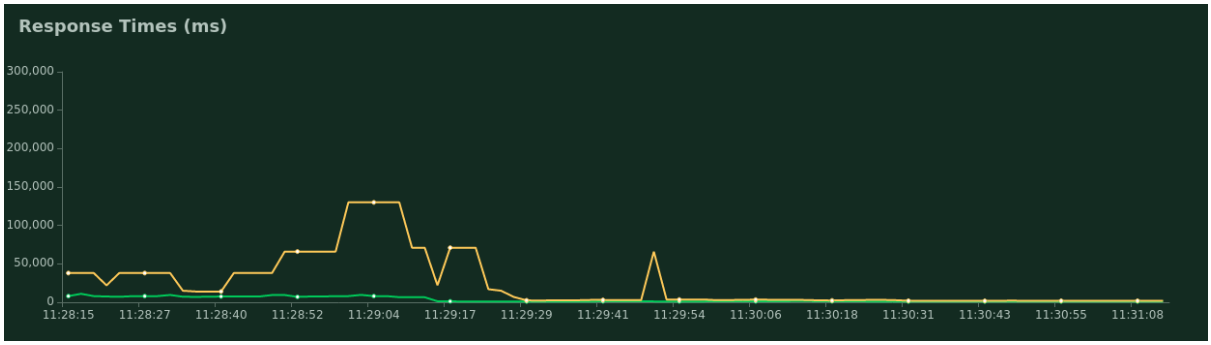

Figura 8 Tempo medio di risposta in ms (in verde) e valore del 95esimo percentile (in giallo)

7 File di configurazione Packstack del nodo cloud T3LAB del laboratorio dimostrativo di SBDIOI40

Il file dichiarativo che contiene la descrizione dell'installazione OpenStack che si vuole realizzare (o answer file) comprende diversi tipi di informazione:

1. Quali sono i servizi OpenStack che si voglio installare sull'infrastruttura cloud che si sta creando, nel nostro caso

- OpenStack Compute (nome in codice Nova);
- OpenStack Image Service (nome in codice Glance);
- OpenStack Identity (nome in codice Keystone);
- OpenStack Dashboard (nome in codice Horizon);
- OpenStack Networking (nome in codice Neutron);
- OpenStack Block Storage (nome in codice Cinder).

Esempio:
CONFIG_CINDER_INSTALL=y
CONFIG_HORIZON_INSTALL=y
CONFIG_NEUTRON_INSTALL=y
CONFIG_GLANCE_INSTALL=y
CONFIG_NOVA_INSTALL=y
CONFIG_MAGNUM_INSTALL=n
CONFIG_CELIOMETER_INSTALL=n

2. La lista delle macchine fisiche che entreranno a far parte dell'infrastruttura cloud e quale ruolo ciascuna di esse è destinata a giocare in tale infrastruttura.

Nella nostra infrastruttura sono stati utilizzati i ruoli controller, compute e storage mentre non ci sono macchine network.
Di ogni macchina viene indicato in particolare l'indirizzo IP.
Packstack installerà su ciascuna macchina quelli tra i servizi OpenStack indicati al punto precedente che sono congruenti con il ruolo svolto dalla macchina nell'infrastruttura cloud.
Esempio:
CONFIG_CONTROLLER_HOST=192.168.0.30
CONFIG_COMPUTE_HOST=192.168.0.31,192.168.0.32

CONFIG_STORAGE_HOST=192.168.0.33
CONFIG_NETWORK_HOST=192.168.0.30

3. Il tipo di rete che sarà utilizzato per connettere tra loro le macchine virtuali di ciascuna delle infrastrutture virtuali di calcolo che saranno realizzate sull'infrastruttura cloud.

Si può ad esempio basarsi su VLAN che si appoggiano a switch fisici o a VXLAN che si appoggiano a switch virtuali.
Se si fa uso di reti virtuali (VXLAN, come nel nostro caso) bisogna anche indicare il tipo di switch virtuali che si vuole utilizzare per implementarle.
Esempio:
CONFIG_NEUTRON_ML2_TYPE_DRIVERS=local,flat,vxlan
CONFIG_NEUTRON_ML2_TENANT_NETWORK_TYPES=local,vxlan

4. Tutte le informazioni di security necessarie per il funzionamento dei diversi servizi di OpenStack (ogni servizio è dotato di una password specifica).
   CONFIG_KEYSTONE_ADMIN_PW=xxxxxxxxxx

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

8 Script realizzati

8.1 Script per la configurazione del cluster Kubernetes

Nome script	Descrizione
init_cluster.sh	Inizializza un cluster Kubernetes sul nodo master. Stampa l'istruzione che deve essere lanciata sugli altri nodi affinché' si registrino al cluster come Worker Node.

8.2 Dockerfile delle applicazioni T3LAB

Nome script	Descrizione
deploy_covid.sh	Effettua il deploy dell'applicazione web per la consultazione dei dati sulla pandemia COVID-19

8.3 Script kubectl per il deployment dell'applicazione webserver sul cluster Kubernetes

Nome script	Descrizione
deploy_webserver.sh	Effettua il deploy dell'applicazione web server per il testing di load balancing e scaling.

8.4 Script kubectl per il deployment dell'applicazione UniMoRe sul cluster Kubernetes

Script realizzati dal CIRI-ICT per il deployment e la rimozione del prototipo della piattaforma applicativa cloud-native di SBDIO I4.0 su un cluster Kubernetes.

Nome script	Descrizione
startApplication.sh	Effettua il deploy dell'applicazione sul cluster Kubernetes, esponendo Front-End e Back-End tramite un Service.
stopApplicationAndCleanEverything.sh	Rimuove l'applicazione dal cluster Kubernetes.

Questi due script sono accompagnati da due file YAML: sbdio_front_end_deployment.yaml e sbdio_back_end_deployment.yaml, utilizzati dallo script startApplication.sh per indicare a Kubernetes come il deployment deve essere effettuato e come i container di Front-End devono comunicare con quelli di Back-End.

POR FESR EMILIA-ROMAGNA 2014/2020
UNIONE EUROPEA
Fondo europeo di sviluppo regionale
RegioneEmilia-Romagna

9 Bibliografia

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

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

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

[4] «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/

[5] «Locust – A modern load testing framework» [Online]. Available: https://locust.io

[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.

[14] http://www.gazlene.net/demystifying-keystone-federation.html