SBDIO I4.0
Big Data
for Industry

O1.4

Prototipo processo tecnologico innovativo accoppiato a proactive diagnostic e addictive manufacturing per la realizzazione di parti e componenti di ricambio

T3LAB

Code	O1.4
Data	29-06-2021
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
RegioneEmilia-Romagna

SBDIO I4.0
Big Data
for Industry

Sommario

1 Obiettivi

L'obiettivo di questa parte del progetto era di replicare l'infrastruttura per l'analisi dei dati vibrazionali provenienti da un prototipo ospitato dal CIRI MAM realizzata dal MechLav di UniFe sul cloud del CIRI ICT sul cloud del T3LAB.

2 Cloud OpenStack T3LAB

Il cloud OpenStack di T3LAB è stato realizzato con l'utilizzo di PackStack. Il funzionamento di PackStack, le sue potenzialità e le sue criticità sono state ampiamente trattate in [2] e non verranno ulteriormente approfondite in questo documento. In Figura 1 è illustrata la sua attuale struttura.

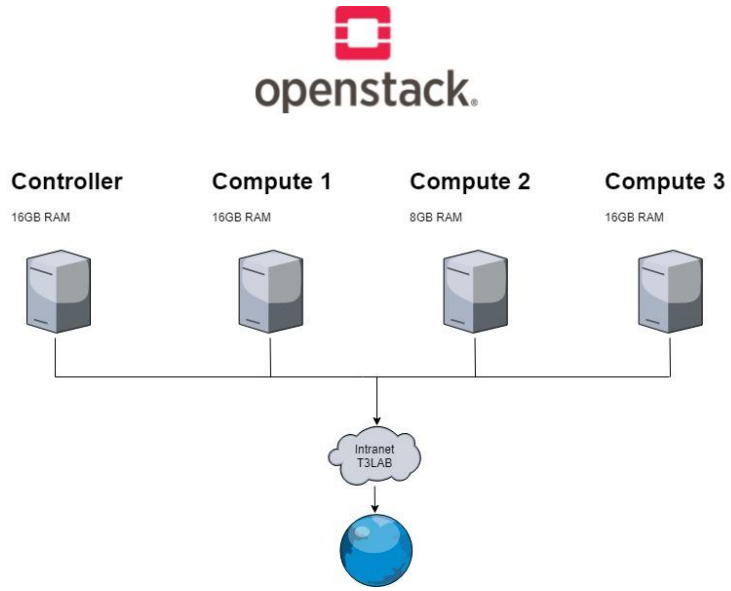

Figura 1 Struttura del cloud OpenStack federato T3LAB

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.

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 [3]. Kubernetes è un progetto sviluppato inizialmente da Google e reso open-source nel 2014. Lo scopo principale di Kubernetes è 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;

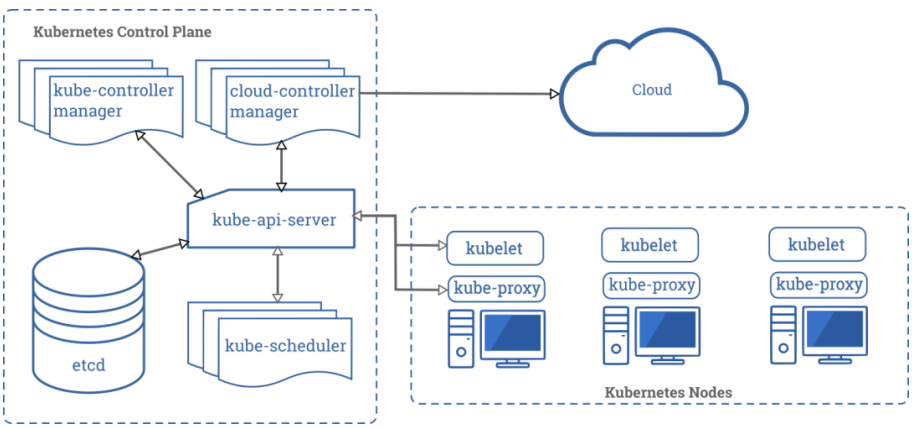

Figura 2 Diagramma di un cluster Kubernetes e delle interazioni dei suoi componenti

- **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.

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 2 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.

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.
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.

4 Tecnologie utilizzate

Di seguito vengono riportate le principali tecnologie con le quali abbiamo dovuto lavorare per questa parte del progetto.

4.1 Elastic Stack

Elastic Stack è un set di strumenti open source che nasce con lo scopo di semplificare l'acquisizione, la gestione, l'elaborazione, l'archiviazione, l'analisi e la visualizzazione di enormi quantità di dati. Comunemente chiamato ELK Stack (acronimo dei tre principali componenti che compongono lo stack: Elasticsearch, Logstash e Kibana) include anche una ricca raccolta di integratori noti come Beats per l'invio di dati a Elasticsearch. In Figura 3 viene riportato uno schema che illustra in modo semplificato il funzionamento dello stack. I Beats sono servizi specializzati per raccogliere dati da fonti eterogenee, e inviano i dati direttamente a Elastic se questi dati non necessitano di un'ulteriore elaborazione, in caso contrario vengono inviati a Logstash. Quest'ultimo componente si occupa di manipolare i dati in ingresso secondo determinate pipelines, per poi inviarli a Elastic che provvederà ad immagazzinarli nel modo migliore secondo determinate politiche di gestione predefinite. L'ultimo componente dello stack, Kibana, non fa altro che prendere i dati da Elastic, aggregarli nel modo giusto per presentarli all'utente finale. Nei prossimi paragrafi tratteremo in modo più approfondito i tre componenti principali dello stack.

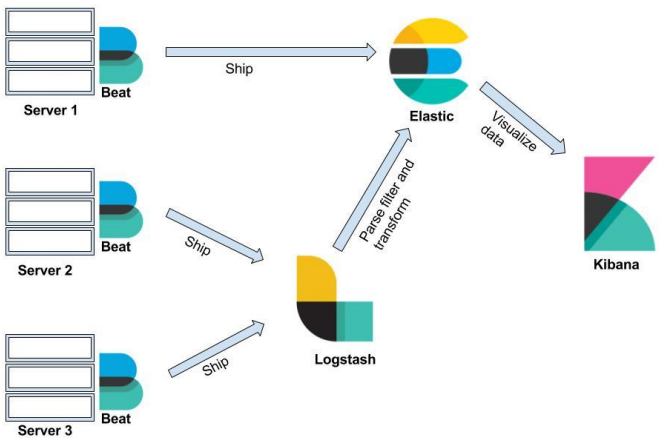

Figura 3 Schema di funzionamento dello stack ELK

4.1.1 Elasticsearch

Si tratta del componente dello stack che si occupa di salvare e indicizzare i dati provenienti da Logstash o dai Beats, per fornire un accesso rapido agli stessi da parte di Kibana al fine di creare le opportune visualizzazioni da mostrare all'utente finale. L'archiviazione dei dati su Elasticsearch si

basa sugli index. Un index in Elasticsearch è una raccolta di documenti correlati tra loro. Elasticsearch archivia i dati come documenti JSON. Ogni documento correla una serie di chiavi (nomi di campi o proprietà) con i valori corrispondenti (stringhe, numeri, valori booleani, date, matrici di valori, geo localizzazione o altri tipi di dati). Durante il processo di indicizzazione, Elasticsearch archivia i documenti e crea un indice per rendere i dati del documento ricercabili quasi in tempo reale. L'indicizzazione viene avviata con le API dell'index, tramite le quali è possibile aggiungere o aggiornare un documento JSON in un indice specifico.

4.1.2 Logstash

Logstash è uno dei prodotti principali di Elastic Stack, viene utilizzato per aggregare ed elaborare i dati e inviarli a Elasticsearch. Logstash è una pipeline di elaborazione dati lato server che consente di acquisire dati da più origini contemporaneamente, arricchirli, eventualmente filtrarli e trasformarli prima che vengano indicizzati in Elasticsearch. Esistono numerosissimi plugin per Logstash che consentono di effettuare manipolazioni sui dati anche molto complesse.

4.1.3 Kibana

Kibana è uno strumento di visualizzazione e gestione dei dati per Elasticsearch che fornisce istogrammi in tempo reale, grafici a linee, grafici a torta e mappe. Kibana include anche applicazioni avanzate come Canvas, che consente agli utenti di creare infografiche dinamiche personalizzate in base ai propri dati ed Elastic Maps per la visualizzazione dei dati geo spaziali.

4.2 Kafka e Kafka Connect

Apache Kafka è un motore di *stream processing* distribuito per costruire pipeline di dati in real-time e applicazioni in streaming. Permette di ricevere dati da diversi tipi di sorgenti, elaborandoli all'interno della sua architettura e rendendoli disponibili ai riceventi. All'interno dell'architettura è possibile costruire applicazioni attraverso la libreria Kafka Streams per operazioni di filtraggio e arricchimento dati. In Figura 4 viene riportato lo schema di funzionamento di Kafka semplificato.

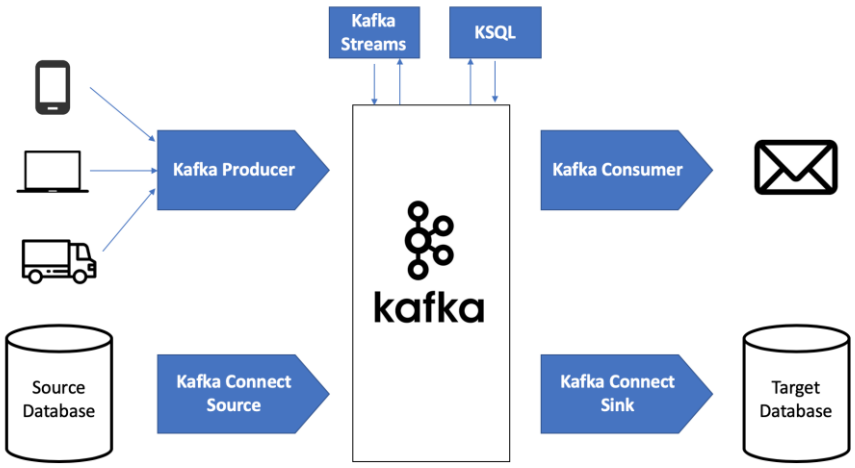

Figura 4 Schema di funzionamento di Apache Kafka

Apache Kafka viene eseguito come cluster su uno o più server, che possono comprendere diversi centri di calcolo. I singoli nodi di un cluster, denominati broker, salvano e categorizzano i flussi di dati in entrata nei cosiddetti topic. Qui vengono suddivisi in partizioni, replicati e distribuiti nel cluster e contrassegnati con una marca temporale. In questo modo la piattaforma di streaming garantisce un'alta disponibilità e un accesso rapido. Apache Kafka differenzia i topic in *normal topic* e *compacted topic*. Nei normal topic Kafka può cancellare i messaggi una volta superato il periodo o il limite di salvataggio, mentre le voci contenute nei compacted topic non hanno alcun limite, né di tempo né di spazio.
Le applicazioni che scrivono dati in un cluster di Kafka vengono definiti produttori, mentre tutte le applicazioni che leggono i dati di un cluster di Kafka si definiscono invece consumatori. La componente centrale da cui attingono i produttori e i consumatori per l'elaborazione di flussi di dati è una biblioteca Java chiamata Kafka Streams. Il supporto di una scrittura transazionale garantisce che i messaggi vengano trasmessi una volta sola (senza doppioni), fenomeno definito anche come *exactly-once delivery*.
Un altro componente molto importante di Kafka è Kafka Connect: è un servizio che consente a Kafka di effettuare l'importazione o l'esportazione di dati da e verso un qualsiasi sistema esterno, ad esempio MySQL, ed altri sistemi di archiviazione dati. Nel corso di questo progetto Kafka Connect ha utilizzato i plugin per collegarsi con Elasticsearch, RabbitMQ e MQTT.

5 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).

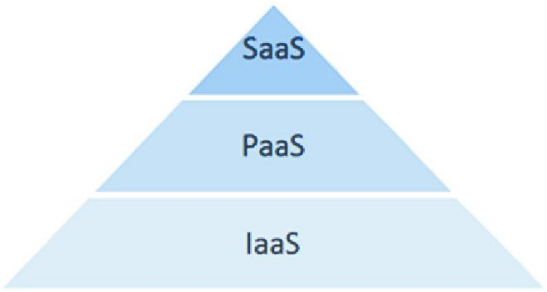

Figura 5 Stack dei possibili servizi cloud

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.

6 Containerizzazione e deploy

In questo capitolo illustreremo il processo che abbiamo seguito per effettuare la containerizzazione e il deploy dell'infrastruttura di cui il MechLav di UniFe aveva bisogno per la realizzazione della sua applicazione.

6.1 Descrizione dell'applicazione

L'applicazione finale consiste in diverse componenti, dislocate in diversi siti. Per prima cosa, un prototipo presso il CIRI MAM di UniBo genera dei dati vibrazionali, che vengono elaborati da una macchina tramite script MatLab e inviati all'esterno tramite un gateway Python. Il resto dell'infrastruttura è stata sviluppata internamente al T3LAB, e consiste in tre componenti principali:

- Kafka e Kafka Connect: i punti di ingresso della nostra infrastruttura. Il gateway Python di UniBo invia i dati a Kafka, che si occuperà di elaborarli, eventualmente arricchirli e successivamente di inviarli al prossimo componente della nostra infrastruttura.
- Elasticsearch: riceve i dati da Kafka, li indicizza e li rende disponibili per la ricerca da parte di terzi.
- Kibana: l'ultimo componente ospitato dal cloud T3LAB. Interroga Elasticsearch e visualizza i dati provenienti da Elasticsearch all'utente finale.

Il T3LAB quindi si è occupato soltanto di realizzare l'infrastruttura descritta nei tre punti precedenti, dal momento che non aveva la possibilità di avere la licenza MatLab e un prototipo per generare i dati vibrazionali da analizzare.

6.2 Containerizzazione e deploy su Kubernetes

Il deploy sul cluster Kubernetes del T3LAB è stato realizzato tramite dei file YAML e la collaborazione del CIRI ICT nella persona del Dott. Lorenzo Patera. I file YAML realizzati non saranno riportati per semplicità. Utilizzando questi file tramite la CLI di Kubernetes (kubectl) effettuiamo il deploy di quattro deployments: kafka, kafka-connect, elasticsearch e kibana. Inoltre creiamo anche quattro servizi che consentono a Kubernetes di esporre all'esterno i quattro componenti di cui abbiamo fatto il deployment, in modo tale che possano accedervi sia il MechLav possa accedervi per implementare la logica di business, sia un eventuale utente finale per poter consultare e visualizzare i dati.

7 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.