Rapporto sulla progettazione della architettura a microservizi della piattaforma SBDIOI40

Code	O2.1
Date	24/1/2020
Type	Confidential
Participants	UNIBO
Authors	Lorenzo Civolani (UNIBO), Dmitrij David Padalino Montenero (UNIBO)
Corresponding Authors	Luca Foschini

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

Sommario

1 Introduzione

Questo documento descrive gli esiti delle attività svolte al 22/01/2020 dal Centro Interdipartimentale di Ricerca Industriale ICT - CIRI dell'Alma Mater Studiorum - Università di Bologna per il progetto POR-FESR Emilia-Romagna 2014-2020 dal titolo "Servizi Big Data In e Out per Industria 4.0: da shop-floor a post-vendita - SBDIOI40" per il soddisfacimento del Risultato 1 del progetto.

Tra le attività previste dal Risultato 1 figurano:

- **Piattaforma cloud federata SBDIOI40**: progettazione e realizzazione dell'architettura di riferimento di SBDIOI40 per abilitare l'integrazione delle diverse realtà aziendali coinvolte a livello di soluzioni 'in' (es. diversi protocolli IoT, e soluzioni MOM e DCS) e 'out' abilitando l'interazione con i prodotti che si intende monitorare e gestire da remoto, e le sorgenti dati esterne. Per consentire la scalabilità elastica di questo supporto su larga scala si impiegheranno tecnologie cloud-enabled già sperimentate come modelli a microservizi e su container (Kubernetes) a livello di supporto al computing, code di messaggi Kafka/Confluent accoppiate al sistema storage NoSQL MongoDB per lo storage, mentre per il supporto a big data processing si impiegheranno Spark/Spark Streaming e soluzioni Google (TensorFlow). Ciò consente di abilitare il supporto a servizi in e out.
- **Analisi di architetture innovative a microservizi** per una scalabilità elastica dei servizi e l'orchestrazione automatica di container attraverso tecnologie Docker/Kubernetes.

Il lavoro descritto nel seguente documento illustra i progressi fatti al fine di perseguire gli obiettivi descritti dal Risultato 1 riassumibili in due punti:

- "Piattaforma cloud federata SBDIOI40": Installazione e configurazione di un cluster OpenStack con implementazione, in collaborazione con T3LAB, di una soluzione federata per consentire l'autenticazione a tutti i partner del progetto alla piattaforma cloud SBDIOI40.
- "Analisi di architetture innovative a microservizi": Installazione e configurazione di un cluster Kubernetes ibrido.

Prima di procedere alla descrizione dettagliata delle attività svolte si farà una panoramica del contesto applicativo, il cloud computing, e delle tecnologie utilizzate per l'implementazione dei cluster, OpenStack e Kubernetes, citate nei due punti precedenti.

2 Cloud computing

Il cloud computing è la soluzione tradizionale per fornire servizi ai clienti attraverso una piattaforma accessibile tramite la rete Internet. Grazie al cloud computing, gli utenti sono in grado di accedere alle risorse di cui hanno bisogno indipendente dal luogo in cui essi si trovano, perché queste risorse sono ospitate da una terza parte nel cloud. Le organizzazioni che offrono servizi di cloud computing vengono solitamente indicate col termine generico *cloud provider*.

Tipicamente, un cloud provider offre servizi che si distinguono in tre principali categorie:

- *Infrastructure-as-a-Service* (o *IaaS*) comprende servizi grazie ai quali l'utente è in grado di predisporre ed eseguire qualunque tipo di software. Il cloud provider mette a disposizione dell'utente semplice potenza di calcolo. L'utente può installare il proprio sistema operativo ed eseguire qualunque applicazione, alla stregua di una macchina fisica disponibile localmente.
- *Platform-as-a-Service* (o *PaaS*) indica servizi grazie ai quali un cliente può eseguire il deployment di applicazioni all'interno di un ambiente di esecuzione che supporta un insieme di linguaggi di programmazione, librerie, servizi e strumenti prestabiliti. Il cliente non ha la possibilità di configurare direttamente il sistema operativo e le sue impostazioni. Al contrario, il cliente può personalizzare solamente il proprio ambiente di esecuzione.
- *Software-as-a-Service* (o *SaaS*) indica uno schema di interazione in cui il cliente utilizza un'applicazione sviluppata e gestita dal cloud provider. Solitamente, queste applicazioni sono "pronte all'uso" e sono accessibili tramite programmi client "leggeri", come un semplice web browser. Esempi di piattaforme SaaS comunemente utilizzate includono la suite office di Google (documenti, presentazioni, fogli di calcolo), applicazioni web per la gestione della posta elettronica, piattaforme per l'archiviazione di file come Google Drive, e così via.

L'adozione del paradigma cloud computing comporta numerosi vantaggi. In primo luogo, grande potenza di calcolo. In secondo luogo, gli utenti possono concentrarsi sulla logica di business della loro applicazione, piuttosto che su configurazione e manutenzione dell'hardware necessarie per mantenere il servizio attivo e funzionante. Anche la convenienza economica riveste senza dubbio un ruolo di primo piano nel determinare il successo di questa architettura: la possibilità di pagare soltanto per le risorse di calcolo effettivamente utilizzate, evitando i costi fissi di un'infrastruttura server locale, risulta estremamente attraente per i clienti. Il cloud computing mostra anche vantaggi in termini di scalabilità. Per incrementare la capacità di calcolo di del sistema, non è necessario acquistare, installare e configurare nuovo hardware localmente all'azienda ma è sufficiente richiedere maggiori risorse al cloud provider. Di conseguenza, fornire supporto a numero crescente di utenti e clienti diventa un problema meno complesso.

Le infrastrutture di cloud computing si suddividono essenzialmente in due macro categorie: cloud pubblici e cloud privati. Quando si utilizza un **cloud pubblico**, si sceglie di affidarsi ai servizi offerti da un cloud provider. Alcuni tra i cloud provider più popolari sono Google Cloud Platform, Amazon Web Services e Microsoft Azure. In questo caso, è il provider ad essere responsabile della gestione e della manutenzione dell'infrastruttura computazionale fisica. Il vantaggio principale dell'utilizzo di un cloud

pubblico è proprio il fatto che l'onere della gestione e della manutenzione delle macchine fisiche è delegato a un'entità terza. D'altro canto, questo schema produce inconvenienti non trascurabili in termini di sicurezza: molte aziende non sono disposte a perdere il controllo diretto sui propri dati sensibili.

Al contrario, una soluzione di **cloud privato** viene gestita internamente. In questo caso, si sceglie di dedicare un certo numero di macchine locali alla costruzione di una piattaforma cloud i cui servizi sono solitamente disponibili internamente all'organizzazione. Una delle piattaforme più popolari in questo caso è OpenStack. L'utilizzo di un cloud privato può comportare costi e tempi di gestione rilevanti. Tuttavia, i vantaggi offerti in termini di sicurezza della gestione dei dati sono un aspetto di grande interesse in molti contesti aziendali.

2.1 Virtualizzazione hardware: le macchine virtuali

Storicamente, all'aumentare della potenza computazionale e della capacità di archiviazione dei server, le applicazioni bare metal non erano più in grado di sfruttare la nuova quantità di risorse. Per questo motivo nacquero le **macchine virtuali**, progettate come software da eseguire su server fisici per emulare un particolare sistema hardware. Un *hypervisor*, chiamato anche virtual machine monitor, è un software, firmware o hardware che consente la creazione, la gestione e l'esecuzione di macchine virtuali. L'hypervisor è quel componente situato tra l'hardware e la macchina virtuale necessario alla virtualizzazione del server.

All'interno di ogni macchina virtuale esegue un unico sistema operativo ospite. Macchine virtuali munite di diversi sistemi operativi possono eseguire sullo stesso server fisico. Ogni macchina virtuale possiede i propri file, librerie e applicazioni e la macchina virtuale può arrivare ad avere una dimensione nell'ordine di diversi gigabyte.

L'introduzione della virtualizzazione del server ha portato diversi benefici, primo tra tutti la possibilità di eseguire più di una applicazione su un singolo sistema, ciascuna in un ambiente di esecuzione separato. La virtualizzazione ha comportato anche una riduzione dei costi grazie ad un'ottimizzazione delle risorse impiegate a livello hardware, alla possibilità di creare server più velocemente e ad una migliorata gestione del disaster recovery.

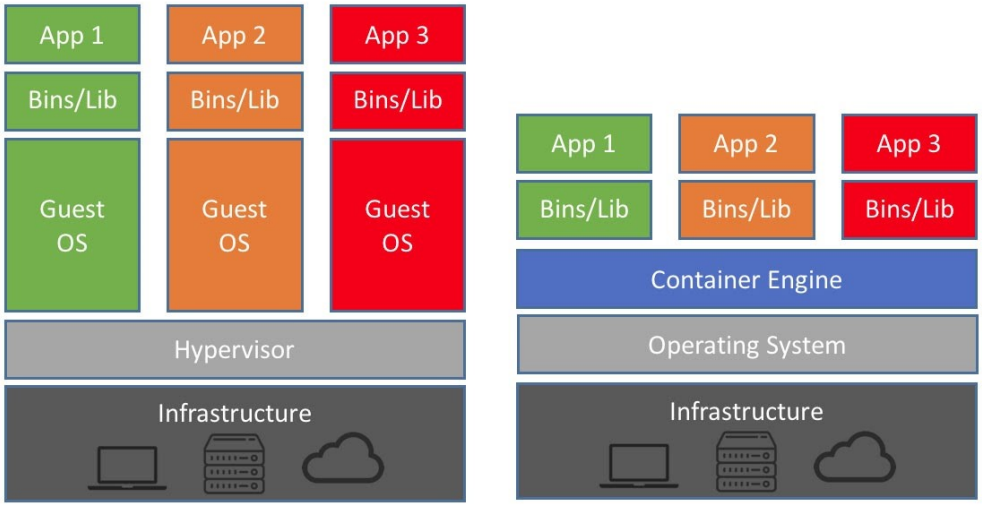
Anche lo sviluppo del software ha tratto vantaggio da questa tecnica perché la riduzione delle risorse hardware utilizzate ha consentito il riutilizzo dei server nelle fasi di sviluppo e di QA.

Questo approccio tuttavia ha anche i suoi aspetti negativi. Ogni macchina virtuale contiene un'immagine del sistema operativo separata, che aggiunge overhead in termini di spazio di archiviazione e di spazio in memoria centrale. Infine questo approccio limita pesantemente la portabilità delle applicazioni tra cloud pubblici, cloud privati e data center tradizionali.

2.2 Virtualizzazione a livello di sistema operativo: i container

Nonostante le macchine virtuali siano ad oggi il metodo di virtualizzazione più utilizzato in ambito cloud computing la ricerca nel campo ha portato alla nascita di un nuovo approccio basato sul concetto di containerizzazione.

La virtualizzazione a livello di sistema operativo, nota anche come **containerizzazione**, sta spopolando negli ultimi anni perché garantisce predicibilità e correttezza all'esecuzione di software applicativo quando viene trasferito da un ambiente server ad un altro.



Machine Virtualization Containers

La containerizzazione è una funzionalità del kernel di un sistema operativo che permette di far coesistere istanze multiple di spazio utente. Queste istanze, chiamate container, partizioni o ambienti virtuali appaiono come veri e propri calcolatori per i programmi che eseguono al loro interno. Un'applicazione in esecuzione all'interno di un tradizionale sistema operativo ha visibilità di tutte le risorse hardware sottostanti, mentre un'applicazione in esecuzione all'interno di un container ha visibilità solo delle risorse assegnate a quel container.

I container risiedono al di sopra del sistema operativo che li ospita (vedi Figura 1). Non dovendo inglobare tutte le risorse di un server, in particolare il kernel del sistema operativo, i container sono molto più leggeri delle macchine virtuali, richiedono poche risorse computazionali e possono essere attivati in tempi ridottissimi. Questo li rende particolarmente adatti alle situazioni in cui il carico di elaborazione da sostenere è fortemente variabile nel tempo e con picchi poco prevedibili.

Da un punto di vista pratico l'impiego dei container nel cloud ha dato vita ad un nuovo modo di concepire l'architettura di un'applicazione. Il nuovo approccio consiste nel collocare i differenti servizi che costituiscono un'applicazione in container separati, e successivamente effettuare il deployment di questi container in cluster di macchine fisiche o virtuali.

La containerizzazione sta diventando sempre più popolare perché i container sono:

- Flessibili: anche le applicazioni più complesse possono essere containerizzate.
- Leggeri: i container sfruttano e condividono il kernel del sistema operativo che li ospita.
- Scambiabili: è possibile rilasciare aggiornamenti e upgrade on-the-fly.
- Portabili: è possibile fare la build del software in locale, rilasciarla nel cloud, ed eseguirla ovunque.
- Scalabili: è possibile aumentare e automaticamente distribuire repliche di container.
- Organizzabili in stack: è possibile organizzare i servizi in stack verticali e on-the-fly.

Le attuali applicazioni distribuite sono composte da decine o centinaia di componenti containerizzati opportunamente abbinati che devono lavorare insieme per consentire a una data applicazione di funzionare come previsto. L'**orchestrazione** dei container fa riferimento al processo di organizzazione del funzionamento di singoli componenti e livelli dell'applicazione.

L'orchestrazione dei container consente agli utenti di controllare l'avvio e l'interruzione dei container, di raggrupparli in cluster e di coordinare tutti i processi che compongono un'applicazione. Gli strumenti di orchestrazione dei container consentono agli utenti di guidare la distribuzione dei container e automatizzare gli aggiornamenti, il monitoraggio dello stato e le procedure di failover.

3 Tecnologie per il cloud computing

3.1 Macchine virtuali: OpenStack

OpenStack è una piattaforma software per il cloud computing che offre agli utenti risorse di calcolo e di archiviazione virtuali. La piattaforma software consiste in una serie di moduli software correlati che controllano un bacino di risorse fisiche eterogenee, come ad esempio risorse di calcolo (computer), di archiviazione e di rete. Grazie alle API offerte da OpenStack, uno sviluppatore può ottenere e gestire risorse virtuali in modo semplice e veloce, senza l'intermediazione di un amministratore di sistema.

OpenStack è un sistema modulare, diviso in una serie di di progetti software. Ognuno di questi componenti di OpenStack, chiamati servizi, esegue una funzione specifica e collabora con gli altri. Per avere una piattaforma funzionante, non è necessario installare tutti i componenti; al contrario, è sufficiente installare 5 servizi fondamentali. I servizi fondamentali per ottenere una piattaforma cloud utilizzabile sono:

- La **dashboard** (Horizon), che offre all'utente un portale web con cui interagire con i servizi della piattaforma.
- Il servizio di **compute** (Nova), che realizza la gestione delle macchine virtuali.
- Il servizio di **networking** (Neutron), che si occupa delle software-defined network, assicurandosi, ad esempio, che le macchine virtuali possano comunicare fra di loro.
- L'**image service** (Glance), che si occupa della archiviazione e del recupero delle immagini per le macchine virtuali (ad esempio, file .iso).
- L'**identity service** (Keystone), che controlla l'autenticazione e l'autorizzazione degli utenti prima che essi accedano agli altri servizi.

Gli utilizzatori della piattaforma hanno essenzialmente 3 possibilità per accedere alle risorse:

1. utilizzare la dashboard, un pannello di controllo intuitivo accessibile con un semplice browser web;
2. impartire comandi testuali tramite l'interfaccia a riga di comando (CLI);
3. inviare richieste alle API REST esposte dai singoli servizi, meccanismo che abilita l'interazione anche da parte di un software.

Solitamente, l'architettura di OpenStack distingue i nodi fisici secondo tre categorie in base alla suddivisione dei servizi tra i nodi.

Il nodo **controller** è quello in cui la gran parte dei servizi comuni della piattaforma, oltre ad altri strumenti generici come la CLI, sono installati. Questo nodo ospita la dashboard, il servizio di image store e quello di autenticazione. Inoltre, ospita gli API server di tutti i servizi della piattaforma. Il nodo **compute** è quello in cui le macchine virtuali vere e proprio sono in esecuzione. Il nodo **network** ospita il servizio di networking per realizzare le funzionalità di rete per le istanze virtuali. Switch e router virtuali, così come anche i server DHCP, sono esempi di risorse create e gestite nel nodo network.

Per quanto riguarda la scelta dei nodi fisici cui assegnare i ruoli, lo schema di un'installazione OpenStack può essere stabilito con un forte grado di libertà. Ad esempio, si potrebbe prevedere un numero di nodi compute proporzionato al carico di lavoro previsto per il cluster. Nel caso di questo progetto, come si vedrà in seguito, si è scelto di seguire, per lo meno in questa fase iniziale, un criterio di semplicità.

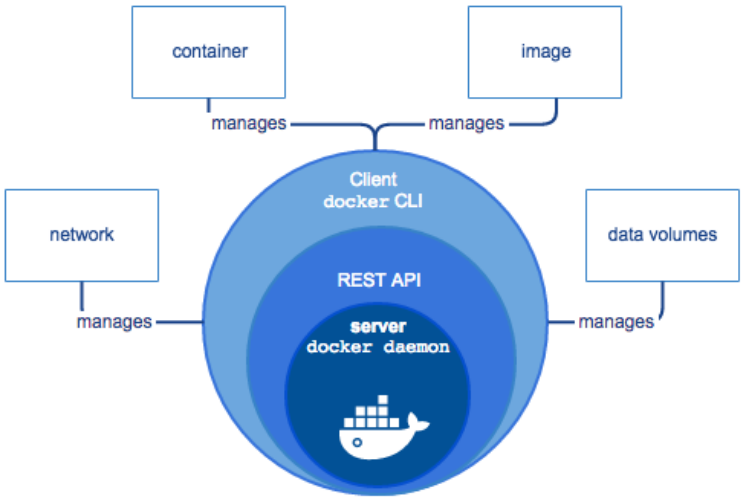
3.2 Container engine: Docker

Docker è una piattaforma software per sviluppatori e amministratori di sistema per sviluppare, rilasciare ed eseguire applicazioni sfruttando la containerizzazione. Docker consente di separare le applicazioni dall'infrastruttura in modo da poter rilasciare software più velocemente. Con Docker, è possibile gestire l'infrastruttura nello stesso modo con cui si gestiscono le applicazioni. Grazie a questa piattaforma si può ridurre drasticamente il gap temporale tra la scrittura del codice del software e il suo rilascio in produzione.

3.2.1 Architettura

Docker è un'applicazione client-server con i seguenti componenti principali descritti come in Figura:

- Un **server** implementato con un processo daemon (chiamato dockerd, Docker daemon).
- Delle **API REST** che specificano le interfacce che i programmi possono utilizzare per comunicare con il daemon per indicargli cosa si deve fare.
- Un **cliente** nella forma di un'interfaccia a linea di comando (CLI).



Il cliente comunica con il daemon, che si occupa di creare, eseguire e distribuire i container Docker. Il cliente e il daemon possono trovarsi nello stesso sistema oppure è possibile connettere un cliente Docker ad un daemon Docker remoto. Il cliente e il daemon comunicano per mezzo delle API REST, su socket UNIX oppure su un'interfaccia di rete.

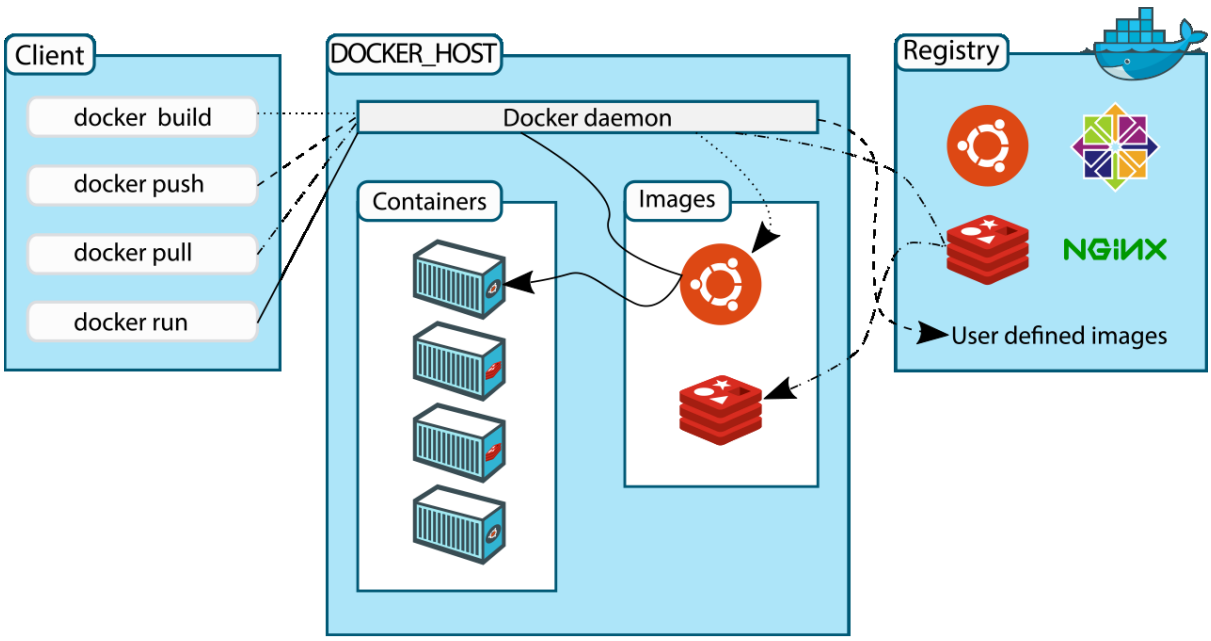
3.2.2 Concetti chiave di Docker

Daemon Docker

Il Docker daemon (dockerd) è un processo il cui compito è ascoltare le richieste inoltrate utilizzando le API e gestisce oggetti come immagini, container, reti e volumi. Un daemon può anche comunicare con altri daemon per gestire i servizi Docker.

Docker Client

Il Docker client (docker) è la modalità utilizzata dalla maggior parte degli utenti che utilizzano questa piattaforma. Quando si utilizzano i comandi come ad esempio docker run, il cliente invia questi comandi al daemon dockerd, che si occupa di eseguirli. Il comando docker utilizza le API Docker. Il cliente può comunicare con più di un daemon.



Registri Docker

Un registro Docker mantiene in memoria le immagini. DockerHub e Docker Cloud sono registri web-based che chiunque può utilizzare, e Docker è configurato a default per cercare le immagini nel DockerHub. Per fornire un esempio concreto, quando uno sviluppatore inizia a sperimentare con Docker è utile come operazione preliminare creare un nuovo account Docker su DockerHub poiché dà la possibilità di accedere ad un registro da utilizzare per ospitare le immagini Docker che incapsulano il software applicativo. Questi registri possono essere pubblici o privati.

Quando si utilizza il comando docker pull o docker run, le immagini richieste verranno prelevate dal registro configurato. Quando si usa il comando docker push, l'immagine viene inserita nel registro.

Oggetti Docker

Quando si utilizza Docker, si parte dal codice eseguibile di una applicazione, per esempio un file jar, lo si impacchetta in un oggetto immagine, si esegue l'immagine all'interno di un oggetto container ed eventualmente si organizzano i container come oggetti servizi.

Immagini

Un'immagine è un template a sola lettura contenente le istruzioni per creare un container Docker. Spesso, un'immagine è basata su un'altra immagine, con alcune personalizzazioni addizionali. Prendendo come esempio un caso reale, si immagini di voler containerizzare un'applicazione web sviluppata con Spring. L'applicazione Spring per eseguire ha bisogno di una Java Virtual Machine e una Java Virtual Machine necessita di un sistema operativo su cui essere installata. Di conseguenza l'immagine dell'applicazione Spring che si dovrà creare necessiterà di un ambiente di esecuzione dato dal sistema operativo e da una Java Virtual Machine. L'immagine da creare si configurerà quindi come una personalizzazione di un'immagine di Java che a sua volta è una personalizzazione di un'immagine di un sistema operativo, per esempio Linux. Per ridurre i tempi deployment nel DockerHub sono già disponibili le immagini delle tecnologie più diffuse pertanto creare un'immagine di questo tipo è molto semplice.

È possibile creare delle immagini personali come nell'esempio appena mostrato oppure è possibile solamente utilizzare immagini create da terzi reperibili da un registro. Per creare un'immagine, bisogna scrivere un file di testo chiamato Dockerfile che grazie ad una semplice sintassi permette di definire i passaggi necessari alla creazione di un'immagine e alla sua esecuzione. Ogni istruzione nel Dockerfile crea un nuovo strato nell'immagine. Quando si modifica un Dockerfile e si ricrea l'immagine, solo gli strati modificati sono ricostruiti. Questa tecnica è parte di quello che rende le immagini così leggere, di dimensioni contenute e veloci da eseguire, se comparate con altre tecnologie di virtualizzazione.

Container

Un container è un'istanza eseguibile di un'immagine. Si può creare, eseguire, interrompere, spostare o cancellare un container utilizzando le API Docker o l'interfaccia a linea di comando. Si può connettere un container ad una o più reti, è possibile assegnargli spazio di archiviazione o anche creare una nuova immagine basata sul suo stato corrente.

A default, un container è relativamente ben isolato dagli altri e dalla macchina che li ospita. È possibile controllare il livello di isolamento della rete, dello spazio di archiviazione e di altri componenti del container rispetto ad altri e alla macchina host.

Servizi

I servizi permettono di scalare i container orizzontalmente su diversi daemon Docker, che insieme lavorano come uno sciame, in gergo **swarm**, di numerosi nodi manager e nodi worker. Ogni membro di uno swarm è un daemon Docker e tutti i daemon comunicano per mezzo delle API Docker. Un servizio consente di definire lo stato desiderato, come ad esempio il numero di repliche di un servizio

che deve essere disponibile in un dato istante. A default, il carico del servizio è bilanciato tra tutti i nodi worker. Per un consumatore di servizi di un'applicazione il servizio Docker appare come una singola applicazione.

3.3 Orchestrazione di container: Kubernetes

Kubernetes, abbreviato in letteratura con k8s, è un potente strumento di orchestrazione open-source sviluppato da Google per la gestione di micro-servizi e applicazioni containerizzate in esecuzione su un cluster e a livello industriale rappresenta lo standard de facto per l'orchestrazione di applicazioni cloud native.

Ad alto livello non fa altro che raggruppare le risorse come CPU, RAM e storage messe a disposizione da tutti i nodi di un data center e nasconde la complessità della loro gestione fornendo all'utente un corposo insieme di API REST. Kubernetes è stato progettato per garantire portabilità, infatti può essere impiegato in diverse piattaforme cloud pubbliche o private come AWS, GCP, Azure, OpenStack o Apache Mesos.

3.3.1 Kubernetes e Docker

Kubernetes non va considerato un sostituto di Docker. È meglio definire Kubernetes un sostituto per alcune tecnologie di alto livello che sono emerse grazie a Docker. Una di queste tecnologie è Docker Swarm, l'orchestratore fornito con Docker. K8s è decisamente più complesso di Swarm e richiede più lavoro per essere configurato. Fortunatamente questo lavoro è finalizzato a fornire nel lungo termine un'infrastruttura dell'applicazione più facile da gestire e resiliente. Di conseguenza per applicazioni containerizzate destinate ad essere ospitate in data center su cui saranno eseguiti pochi container, Docker Swarm rappresenta la scelta più semplice.

3.3.2 Funzionalità

Kubernetes fornisce astrazioni di alto livello per gestire gruppi di container che permettono all'utente di concentrarsi maggiormente su come le applicazioni siano eseguite più che sui dettagli di implementazione.

Queste sono alcune delle funzionalità che Kubernetes fornisce:

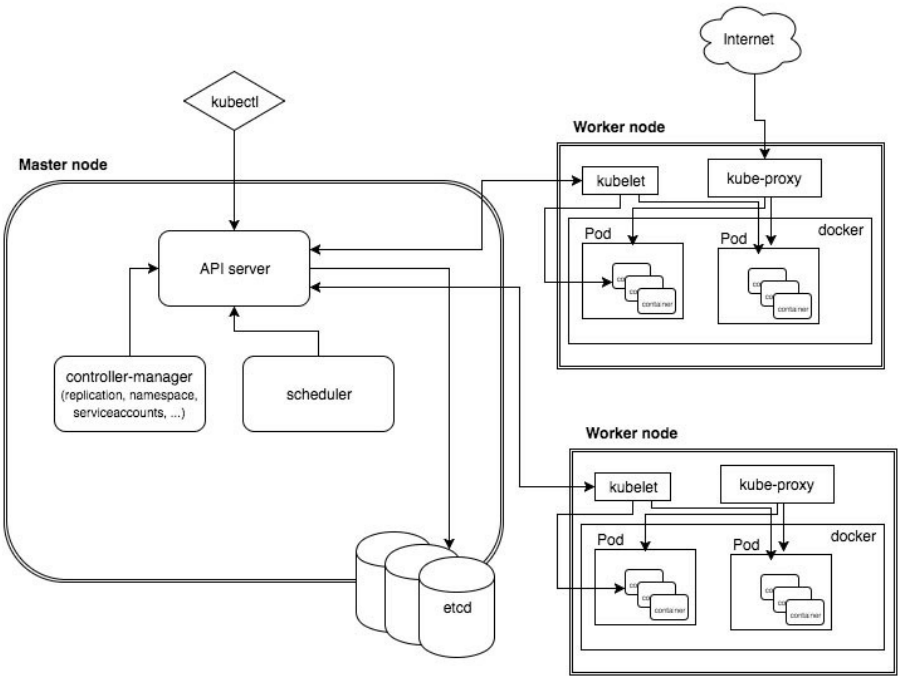
- Deployment di applicazioni multi-container.
- Scaling automatico di applicazioni containerizzate.
- Aggiornamento live di un'applicazione senza interruzione del servizio.
- Provisioning di risorse di rete, spazio di archiviazione e servizi di discovery.
- Indipendenza dall'infrastruttura sottostante.

3.3.3 Architettura

Kubernetes sfrutta un'architettura master-slave. L'astrazione di più alto livello che k8s introduce è il concetto di cluster. Il cluster si riferisce ad un gruppo di macchine che eseguono Kubernetes e ai container che Kubernetes si occupa di gestire. Un cluster deve avere almeno un nodo master, il quale comanda e controlla tutte le altre macchine Kubernetes all'interno del cluster. I componenti di Kubernetes si possono dividere in quelli che gestiscono un singolo nodo e quelli che fanno parte del control plane realizzato dal nodo master, vedi Figura 9.

Il cluster è composto da un master e da un insieme di nodi worker. Il master ha la responsabilità di gestire e di controllare l'intero cluster, questo significa che spettano a lui le decisioni di scheduling. Inoltre tra le numerose mansioni effettua attività di monitoraggio, riflette i cambiamenti indicati dall'utente, risponde agli eventi. È possibile avere una configurazione con più di un master in caso di cluster in cui si voglia garantire high-availability tuttavia a default è presente un solo master che funge da nodo di controllo e unico punto di accesso al cluster. Per le motivazioni appena elencate spesso ci si riferisce al master come al control plane.

I nodi worker sono il luogo in cui eseguono le applicazioni cloud native, per questo motivo a volte ci si riferisce a loro come al data plane. Il loro compito oltre all'esecuzione di applicazioni è quello di inviare continuamente informazioni al master sul loro stato e di restare costantemente in attesa per nuovi carichi di lavoro.



3.3.4 Control plane

Un master Kubernetes è una collezione di servizi di sistema che insieme compongono il control plane del cluster. Di seguito si dà una breve descrizione di questi servizi.

API Server

L'API Server è l'entità di gestione centrale che riceve tutte le richieste REST per la modifica delle entità presenti nel cluster come ad esempio i pod e i servizi, e di fatto rappresenta il front-end del cluster.

Cluster Store Etcd

È un semplice database distribuito con struttura dati di tipo chiave-valore utilizzato per memorizzare le informazioni del cluster. Per motivi di sicurezza è accessibile solamente dall'API server. Rappresenta l'unico componente stateful del control plane e mantiene in modo persistente l'intera configurazione e l'intero stato del cluster. Il cluster store è attualmente basato su Etcd, un popolare database distribuito.

Controller Manager

Esegue in background diversi processi di controllo per regolare lo stato distribuito del cluster e si occupa delle operazioni di routine. Quando un utente apporta una modifica nella configurazione, il controller si accorge di tale modifica e si occupa di portare il cluster nel nuovo stato desiderato. Nel concreto, il controller manager è un controller di diversi controller e viene fornito come un componente monolitico. Tuttavia, anche se esegue come un processo singolo, gestisce numerosi cicli di controllo tra loro indipendenti che osservano lo stato del cluster e rispondono agli eventi. Lo scopo di questo gestore è far sì che lo stato corrente del cluster corrisponda a quello desiderato specificato dall'utente, questo obiettivo è alla base dell'approccio dichiarativo di Kubernetes.

Scheduler

Supporta lo scheduling dei pod (container) sui vari nodi. Legge i requisiti dei servizi richiesti dall'utente e ne fa lo scheduling sul nodo più adatto. Se ad esempio un'applicazione ha bisogno di 1GB di RAM e di una CPU a 2 core, allora il pod per quell'applicazione sarà eseguito sul nodo che soddisfa i requisiti minimi richiesti. Lo scheduler è in esecuzione ogni volta che c'è il bisogno di effettuare lo scheduling di un pod. Lo scheduler deve conoscere il numero totale di risorse disponibili e il numero di quelle già allocate sui diversi nodi. È importante sottolineare che lo scheduler non si occupa di mettere in esecuzione un nuovo container ma solo di indicare su quali nodi il container dovrà essere eseguito.

Cloud Controller Manager

È responsabile della gestione dei processi di controllo che interagiscono con il cloud provider sottostante.

3.3.5 Data plane

I nodi sono i worker di un cluster Kubernetes. Ad alto livello svolgono tre compiti fondamentali:

- Restano in ascolto per nuovi carichi di lavoro comunicati dall'API Server.
- Eseguono i carichi di lavoro.
- Riportano costantemente le informazioni sul loro stato al control plane.

Kubelet

È l'agente principale di Kubernetes ed esegue su tutti i nodi del cluster. Quando un nodo entra a far parte di un cluster, l'operazione di join prevede l'installazione del Kubelet che provvederà alla registrazione vera e propria del nodo. È durante questo processo che il nodo invia al control plane le informazioni sulle sue risorse di CPU, RAM e storage che entreranno a far parte del pool di risorse del cluster gestite dal control plane. Uno dei compiti principali del Kubelet è quello di osservare l'API server per la ricezione di nuovi incarichi di lavoro. Ogni volta che arriva una nuova richiesta di lavoro, esegue il task e mantiene un canale di comunicazione per il report al control plane. In caso un Kubelet non fosse in grado di eseguire un particolare task informa il master e sarà il control plane a decidere la migliore azione da intraprendere per risolvere la situazione.

Inoltre, è il Kubelet il componente che si occupa di gestire tutte le operazioni relative ai container come ad esempio scaricare un'immagine dal Docker Hub e in seguito eseguire o interrompere l'esecuzione di un container. Per farlo ha bisogno di comunicare con un container engine come Docker, Kubernetes grazie ad un modello a plugin chiamato Container Runtime Interface (CRI) è in grado di supportare diversi container engine.

Kube Proxy

È un servizio proxy presente su ogni nodo worker 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.

Kubectl

Kubectl è lo strumento client a linea di comando che interagisce con l'API Server e che consente all'amministratore del cluster di inviare comandi al nodo master. Ogni comando è convertito in una chiamata alle API di Kubernetes. Grazie a Kubectl è possibile effettuare il deployment e la gestione delle applicazioni che eseguono su Kubernetes. Con questo strumento si può ad esempio ispezionare le risorse del cluster e creare, eliminare e aggiornare gli oggetti di Kuberentes.

3.3.6 Concetti chiave di Kubernetes

Utilizzare Kubernetes richiede la comprensione di diverse astrazioni utilizzate per rappresentare lo stato del sistema. Di seguito si menzionano per brevità solo i costrutti più rilevanti.

Pod

I nodi worker eseguono i pod, i componenti Kubernetes più elementari che possono essere creati o gestiti. Ogni pod rappresenta una singola istanza di un'applicazione o di un processo in esecuzione in Kubernetes e consiste in uno o più container. Kubernetes esegue, interrompe e replica tutti i container in un pod trattandoli come un gruppo. Il concetto di Pod è stato introdotto in modo che l'utente si concentri sull'applicazione più che sui singoli container.

I pod situati nei nodi vengono creati e distrutti a seconda dello stato specificato dall'utente nella definizione del pod. Ciascuno di essi ha al suo interno i container di un'applicazione, risorse in termini di spazio di archiviazione, un identificatore univoco di rete e altre configurazioni su come eseguire i container. I pod rappresentano anche l'unità di scaling in Kubernetes, in caso fosse necessario scalare un'applicazione si aggiungono o si rimuovono dei Pod.

Service

I pod sono delle entità volatili e per tale motivo non sono affidabili, ovvero Kubernetes non garantisce che un determinato pod fisico sia tenuto sempre in vita. In caso un pod dovesse inaspettatamente interrompersi, k8s non si occuperebbe di riportarlo in vita ma ne creerà uno nuovo al suo posto. Anche se il nuovo pod apparirà a tutti gli effetti come quello precedente in realtà non lo è, avrà infatti un nuovo indirizzo IP. Per far fronte a questo problema k8s introduce il concetto di Service che si occupa di fornire a livello di networking un punto di accesso stabile ad un insieme volatile di pod.

I service forniscono un front-end stabile costituito da un nome DNS, un indirizzo IP e una porta mentre dietro le quinte bilanciano il carico per un insieme dinamico di pod. I service sono in grado di osservare la dinamicità del ciclo di vita dei pod e di conseguenza si aggiornano automaticamente al fine di fornire un endpoint di rete stabile a tutti gli utenti che hanno bisogno dei servizi offerti da container in esecuzione nei pod.

Volume

Kubernetes ha un potente ed avanzato sistema di gestione dello spazio di archiviazione denominato Persistent Volume Subsystem. Indipendentemente dal tipo o dalla provenienza, tutte le risorse di storage che si interfacciano con di k8s sono modellate come dei *volume*.

In Kubernetes il collante che connette le risorse di storage esterne con l'orchestratore è il Container Strage Interface (CSI). Questa interfaccia, analogamente a quanto visto con il Container Runtime Interface, è uno standard aperto che utilizza un modello a plugin per agganciare risorse di storage di qualunque natura.

Il Persistent Volume Subsystem è un insieme di API che consente ad un'applicazione containerizzata di utilizzare risorse di archiviazione. Ad alto livello, i *Persistent Volume* (PV) rappresentano il mapping di una risorsa di dati esterna all'interno del cluster e i *Persistent Volume Claims* è il meccanismo di autorizzazione che consente ad un container di utilizzare un Persistent Volume.

Namespace

Kubernetes supporta la divisione logica di un cluster fisico in diversi cluster virtuali. Un namespace pertanto rappresenta un cluster virtuale, è pensato per isolare ambienti con utenti diversi sparsi in diversi team o progetti. Le risorse all'interno di un namespace devono essere univoche e non possono essere accessibili da un altro namespace. Inoltre ad un namespace può essere allocata una quota di risorse per evitare che occupi tutte le risorse condivise con gli altri namespace del cluster.

Deployment

Una delle caratteristiche chiave da garantire per le applicazioni cloud native è la resilienza. I pod da soli non sono in grado di risolvere eventuali malfunzionamenti, non scalano, e non forniscono nessun meccanismo per la gestione degli aggiornamenti e dei rollback. In K8s sono i *deployment* ad occuparsi di tutto questo e per questo motivo nella pratica i pod sono messi in esecuzione sempre attraverso dei deployment. Ogni deployment è responsabile della gestione di un singolo tipo di pod, per esempio in caso di un'applicazione web composta da un pod per il servizio di front-end e un pod per quello di back-end si dovranno utilizzare due pod.

A livello pratico un deployment descrive lo stato desiderato di un pod o di un insieme di repliche in un file con estensione YAML similmente al file docker-compose. Il deployment controller applica poi al cluster le modifiche specificate dal file di deployment in modo che lo stato corrente del sistema rispecchi quello desiderato.

4 L'infrastruttura SBDIOI40

4.1 Cluster OpenStack

L'architettura del cluster è a singolo nodo, con la possibilità di aggiungere una ulteriore macchina nell'immediato futuro. Il nodo ricopre i ruoli *controller*, *compute* e *network*. Il modello della macchina fisica è un **Acer AR585F1** con le seguenti caratteristiche hardware:

- 32 CPU
- 32GB di memoria RAM
- 15TB di memoria di massa
- 4 interfacce di rete

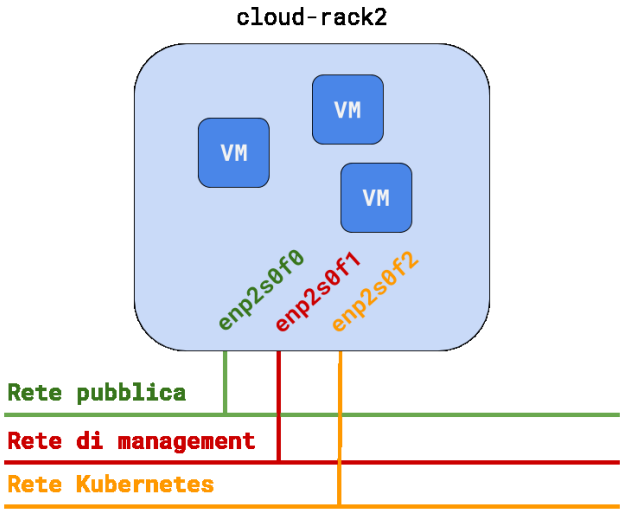
Per quanto riguarda la scelta del sistema operativo installato sulle macchine dell'infrastruttura, la famiglia dei sistemi Linux sembra una direzione naturale per alcune ragioni fondamentali: innanzitutto, tali sistemi sono disponibili gratuitamente; in secondo luogo, Linux è un sistema operativo estremamente diffuso in ambito server; infine, la piattaforma OpenStack richiede la presenza di questo tipo di sistema operativo per poter essere installata con successo. Per quanto riguarda quest'ultimo punto, in particolare, la documentazione ufficiale di OpenStack suggerisce l'installazione della piattaforma su una delle seguenti distribuzioni Linux:

- openSUSE e SUSE Linux Enterprise Server
- Red Hat Enterprise Linux e CentOS
- Ubuntu

Per questioni di efficienza, stabilità e affidabilità, la nostra scelta finale è ricaduta su **CentOS 7**. Questa distribuzione Linux è estremamente efficiente in termini di consumo di risorse perché solo i programmi essenziali sono installati. Una caratteristica primaria di CentOS è un ciclo di aggiornamento del sistema e delle applicazioni piuttosto lungo. I vantaggi sono estremamente rilevanti: preserva compatibilità con molti software server-side, rende l'ambiente non solo efficiente ma anche stabile. Il lato negativo sta nel fatto che, spesso, le applicazioni di CentOS risultano "datate"; tuttavia, per i programmi fondamentali, il problema non è significativo. Infine, CentOS è un sistema estremamente longevo in termini di durata del supporto da parte di Red Hat (CentOS 7, ad esempio, riceverà aggiornamenti di manutenzione fino al 2024). Il risultato è un sistema leggero e affidabile, senza dubbio adatto per un uso su server.

La configurazione del sistema operativo in ottica preparatoria all'installazione di OpenStack di CentOS ha riguardato principalmente la disabilitazione dei servizi di sistema che possono interferire con la piattaforma e la configurazione delle interfacce di rete. I servizi disabilitati sono: firewalld, che rischia di interferire con iptables, lo strumento utilizzato da OpenStack per attuare politiche di filtraggio del traffico; NetworkManager, la cui natura dinamica non è compatibile con Neutron; SELinux, le cui politiche di controllo degli accessi possono ostacolare il funzionamento di OpenStack.

Per quanto concerne le impostazioni di rete, la macchina utilizza 3 interfacce fisiche. Una prima interfaccia offre l'accesso alla rete pubblica (colore verde, nel diagramma), rendendo possibile gestire l'infrastruttura virtuale da remoto ed esporre i servizi su Internet. Una seconda interfaccia allaccia il computer alla rete di gestione del cluster CIRI-ICT (colore rosso). Si tratta di una rete locale che viene utilizzata sia per accedere alle macchine per operazioni di manutenzione, sia come canale di comunicazione per il traffico di controllo fra i nodi di OpenStack, possibilità che diventa fondamentale nel momento in cui si scelga di procedere con l'estensione del cluster a 2 nodi. La terza interfaccia di rete fornisce l'accesso alla rete dedicata al traffico di controllo e applicativo della piattaforma Kubernetes (colore arancio). L'architettura di Kubernetes sarà descritta più avanti.



Per la procedura di installazione, abbiamo utilizzato Packstack. Packstack è uno strumento di installazione che guida l'amministratore di sistema nell'installazione e nella configurazione di OpenStack su distribuzioni Linux della famiglia Red Hat. L'implementazione di Packstack consiste in una serie di moduli Puppet che realizzano il deployment e la configurazione dei componenti sui server multipli attraverso connessioni ssh. L'utente specifica le specifiche del cluster da realizzare all'interno di un file di configurazione chiamato "answer file". Packstack avvierà l'esecuzione per raggiungere l'obiettivo richiesto.

La stesura di un answer file adeguato alle caratteristiche del cluster desiderato rappresenta il punto cruciale di un'installazione di OpenStack effettuata tramite Packstack. Un answer file suddiviso in alcune sezioni principali contenenti la specifica di parametri relativi ai vari aspetti della piattaforma. Di seguito riportiamo in maniera concisa i parametri principali e le motivazioni per la loro impostazione.

Una delle prime sezioni è quella dedicata alla scelta dei servizi da installare. Questa sezione è composta da parametri del tipo seguente (* indica il nome di un servizio):

```
*_INSTALL
```

Con l'obiettivo di ridurre il più possibile la complessità della piattaforma, abbiamo scelto di installare solo i componenti fondamentali:

- MariaDB, il database SQL utilizzato da tutti i servizi per conservare lo stato corrente
- Glance (Image service)
- Nova (Compute service)
- Neutron (Networking service)
- CLI, utile per interagire con OpenStack via riga di comando
- Horizon (Dashboard), utile per interagire con un'interfaccia grafica intuitiva

Una seconda sezione cruciale per Packstack consiste nella specifica degli indirizzi dei nodi fisici del cluster. Essendo il nostro cluster composto da un nodo singolo, specifichiamo il medesimo indirizzo per tutti e tre i ruoli.

```
CONTROLLER_HOST = XXX.XXX.XXX.XXX

COMPUTE_HOST = XXX.XXX.XXX.XXX

NETWORK_HOST = XXX.XXX.XXX.XXX
```

Un'ultima parte fondamentale è quella dedicata alla configurazione di Neutron, servizio che necessita di un'attenta regolazione. Di seguito riportiamo un estratto dei parametri, seguito da una breve discussione.

```
ML2_TYPE_DRIVERS = flat,local,vxlan

ML2_TENANT_NETWORK_TYPES = vxlan

ML2_MECHANISM_DRIVERS = openvswitch

ML2_VNI_RANGES = 10:100

L2_AGENT = openvswitch

OVS_BRIDGE_MAPPINGS = extnet:br-ex,kubenet:br-kub

OVS_BRIDGE_IFACES = br-ex:enp2s0f0,br-kub:enp2s0f2

OVS_BRIDGE_COMPUTE

OVS_EXTERNAL_PHYSNET = extnet

PROVISION_OVS_BRIDGE = y
```

Come driver per la realizzazione delle reti virtuali si è scelto di adoperare le VXLAN per via della flessibilità offerta da questa tecnologia. Con le VXLAN, il traffico delle reti virtuali viene incapsulato

all'interno di tunnel che sono stabiliti tra due terminali dell'infrastruttura di rete fisica sottostante. Di conseguenza, nel momento in cui un nuovo nodo compute venisse aggiunto al cluster OpenStack, resterebbero aperte varie possibilità circa la scelta della rete fisica cui delegare il carico del traffico applicativo: la rete di management del cluster; la rete dei dati applicativi del cluster; la rete dedicata alla piattaforma Kubernetes. Inoltre, l'uso delle VXLAN al posto delle VLAN rimuove l'obbligo di instaurare e mantenere una sincronizzazione tra lo switch fisico cui i server sono collegati e alcuni parametri di configurazione di OpenStack.

La configurazione degli switch virtuali necessari per la realizzazione delle reti dei tenant in OpenStack segue un criterio di massima semplicità. Le interfacce di rete fisiche del server che realizzano l'accesso alla rete pubblica e alla rete Kubernetes (enp2s0f0, enp2s0f2) sono collegate a due switch virtuali denominati rispettivamente "br-ex" e "br-kub". Alle reti cui questi due componenti danno accesso si associano rispettivamente i nomi logici "extnet" e "kubenet". Questi, in altre parole, sono i nomi che l'utente dovrà specificare qualora volesse collegare una V.M. (o un altro componente virtuale gestito da OpenStack, come ad esempio un router) a una rete che esiste fisicamente in laboratorio.

4.2 Federazione degli accessi

In generale, con il termine *federazione* si intende la condivisione di un singolo meccanismo di autenticazione tra sistemi multipli. Nel caso di OpenStack, la federazione consiste in una autenticazione unica fra molteplici cloud.

I vantaggi dell'approccio federato all'autenticazione sono molteplici. In primo luogo, è possibile delegare l'onere dell'autenticazione a una entità terza, scaricando i singoli cloud privati di questa responsabilità. Inoltre, diventa possibile sfruttare una sorgente di identità unica e condivisa: con le stesse credenziali è possibile accedere ed utilizzare i servizi di diverse piattaforme. Il risultato è che grazie a un meccanismo di autenticazione federata si ha la possibilità di connettere, di fatto, molteplici cloud insieme.

La sezione successiva illustra i concetti legati alla federazione degli accessi attraverso la discussione di SAML, lo standard che è stato studiato e utilizzato come protocollo di autenticazione tra due prototipi di piattaforma cloud all'interno di questo progetto.

4.2.1 Lo standard SAML

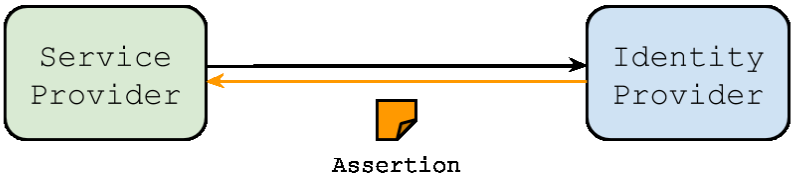
SAML (Security Assertion Markup Language) è uno standard per la descrizione e lo scambio di informazioni di sicurezza tra le più parti. Questo standard è comunemente utilizzato per realizzare meccanismi di single sign-on (SSO), autenticazione federata, e così via. SAML propone un protocollo di comunicazione basato su XML per trasmettere dei "token" di sicurezza, chiamati *assertion*, tra un'autorità che li emette (l'Identity Provider) e un consumatore che li utilizza (il Service Provider). Queste assertion contengono informazioni fondamentali sull'utente che è in fase di autorizzazione.

All'interno dello standard SAML, e pure in generale nel contesto della federazione, si distinguono, come già accennato, 3 attori fondamentali: il client, il Service Provider (SP) e l'Identity Provider (IdP).

Il **client**, anche detto *principal*, rappresenta il soggetto che desidera accedere a una risorsa per la quale sia richiesto autenticarsi. Tipicamente, il client è un utente umano; in altri casi, il client potrebbe coincidere con una applicazione, che, nel caso di OpenStack, potrebbe voler accedere ai servizi della piattaforma cloud attraverso le sue API.

Il **service provider** (SP) rappresenta l'entità che offre un servizio protetto e che ha la necessità che il cliente si autentichi prima di accedere. Un esempio di SP potrebbe essere un sito web che ha bisogno dell'autenticazione degli utenti per decidere se autorizzare (o negare) l'accesso a una risorsa. Il SP delega l'autenticazione a un Identity Provider, ricevendo come risposta da quest'ultimo un documento chiamato "assertion". Nel caso di OpenStack, Keystone ricopre il ruolo del service provider. Come illustrato brevemente in precedenza, Keystone è il servizio che si occupa dell'autenticazione dei clienti. Il servizio offerto da Keystone è il rilascio dei **token**, una speciale stringa di testo che il cliente può utilizzare per accedere agli altri servizi della piattaforma (ad esempio, per lanciare una nuova macchina virtuale).

L'**identity provider** (IdP) indica l'entità centralizzata e affidabile che mantiene una base dati di informazioni sugli utenti ed è incaricata di procedere alla loro autenticazione. L'IdP riceve una richiesta SAML da un SP, gestisce il processo di autenticazione ed invia una risposta SAML indietro al SP. Il protocollo SAML non specifica in modo preciso il metodo di autenticazione che l'IdP deve mettere in pratica. Ad esempio, un IdP potrebbe chiedere delle semplici credenziali (utente e password), oppure realizzare un'autenticazione multi-fattore. è importante notare che l'IdP è responsabile per l'autenticazione (chi è l'utente) ma non per l'autorizzazione (che cosa l'utente può fare), perché le scelte di autorizzazione dipendono dal SP.



Oltre ai 3 ruoli fondamentali (principal, SP e IdP), SAML definisce anche: struttura delle asserzioni, protocolli, *binding*, e *profili*. Una **asserzione**, meglio nota con l'inglese *assertion*, è una dichiarazione formale rilasciata dall'IdP per il SP a seguito di una autenticazione completata con successo. Una assertion è scritta in formato XML, contiene vari attributi dell'utente ed è tipicamente firmata e criptata prima di essere spedita attraverso la rete. Attributi comuni contenuti in un'asserzione includono: id, nome, ruolo, indirizzo e-mail, numero di telefono, eccetera. SAML non dispone una definizione precisa della tipologia di attributi: ogni IdP archivia ed emette un proprio insieme di attributi. Il SP deve quindi sapere cosa aspettarsi.

Un **protocollo** descrive in che modo gli elementi dello standard (come le asserzioni) siano impacchettati all'interno dei messaggi di richiesta e di risposta. Inoltre, definisce le modalità secondo

cui questi messaggi vengono scambiati fra le parti. SAML è, sostanzialmente, un semplice protocollo del tipo richiesta-risposta.

I **binding** definiscono il meccanismi di trasporto a "basso livello" dei messaggi SAML. Ad esempio, nel profilo Web Browser SSO, i binding utilizzati comunemente sono HTTP Redirect e HTTP POST.

I **profili** descrivono i casi d'uso comuni dello standard. Un profilo descrive e vincola il contenuto delle asserzioni, dei protocolli e dei binding al fine di supportare un caso d'uso specifico. Ad esempio, due profili molto utilizzati e anche rilevanti per il caso di OpenStack sono *Web Browser SSO* e *Enhanced Client Proxy* (ECP).

4.2.2 Federazione in OpenStack

Per quanto concerne l'identity provider, OpenStack offre sostanzialmente due possibilità: "Keystone as a SP" e "Keystone to Keystone" (K2K). Nella configurazione con **Keystone as a SP**, Keystone delega il processo dell'autenticazione a un IdP esterno (Google, ad esempio) e consuma le assertion da lui emesse al termine della procedura. Questo schema risulta di particolare utilità se l'organizzazione ha già una base dati per le identità. In questo modo, si evita di creare un gruppo di identità separate per l'accesso ai servizi cloud.

La configurazione **Keystone to Keystone (K2K)** realizza la federazione senza avvalersi di entità esterne. Lo schema prevede che si selezioni uno tra i servizi Keystone delle piattaforme da federare: quel servizio svolgerà il ruolo di IdP. Di conseguenza, gli altri Keystone si rivolgeranno a lui per la validazione dei dati di accesso.

In ultimo, per quanto riguarda lo scambio delle assertion fra IdP e SP, il supporto di OpenStack per i protocolli di autenticazione varia a seconda dello schema di autenticazione prescelto. Nel caso di un IdP esterno (con Keystone nel ruolo di SP) il supporto include SAML2.0, OpenID Connect, o un qualunque altro protocollo, a condizione di fornire a OpenStack il relativo plugin. Nel caso di uno schema K2K, invece, il supporto si limita al protocollo SAML2.0.

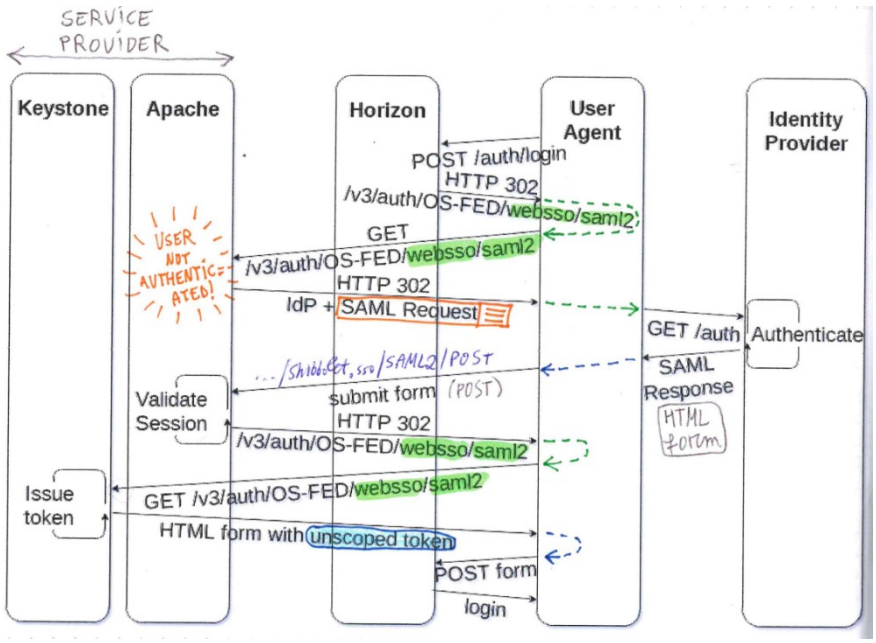
4.2.3 La federazione dei cloud CIRI-ICT e T3LAB

Per un primo prototipo di federazione degli accessi tra le piattaforme OpenStack del CIRI-ICT e del T3LAB si è scelto di adottare lo schema con Identity Provider esterno e Keystone come semplice Service Provider. Per quanto riguarda l'IdP, la scelta è ricaduta su JumpCloud. JumpCloud è un servizio di directory che gli utenti possono utilizzare per effettuare procedure di autenticazione gestendo utenti e applicazioni multiple. Il vantaggio principale di JumpCloud è che il servizio offerto resta gratuito fino a un massimo di 10 utenti.

Scelto l'IdP, si è passati alla configurazione di Keystone affinché deleghi la procedura di autenticazione a JumpCloud. Come primo passo, occorre predisporre Horizon affinché mostri, nella pagina di accesso, un menu addizionale che consenta di scegliere il metodo di autenticazione desiderato tra quello locale e quello federato. Poi, si passa alla configurazione di Keystone (il servizio di autenticazione). Facendo

uso delle sue API, abbiamo creato tre distinte nuove risorse interne a Keystone: un Identity Provider, che modella JumpCloud; un mapping, che consiste in un insieme di regole di traduzione degli attributi specificati dall'IdP verso attributi che Keystone è in grado di comprendere; un protocollo di federazione, che definisce la scelta di SAML e mette in relazione la risorsa IdP con il mapping. Una volta create le risorse, abbiamo installato Shibboleth, un modulo del servizio di Horizon (il servizio di dashboard) che gestisce il processo dell'autenticazione federata lato SP. Quando un utente prova a richiedere il rilascio di un token tramite l'endpoint dedicato all'autenticazione federata, è compito di Shibboleth assicurarsi che egli sia autorizzato prima di inoltrare la richiesta a Keystone. A questo punto occorre uno scambio di Entity ID e metadati tra SP e IdP, affinché ciascuno conosca l'identità dell'altro e sia quindi in grado di stabilire una relazione di fiducia. In particolare, Shibboleth deve essere parametrizzato con l'Entity ID e il file di metadati di JumpCloud; sulla piattaforma JumpCloud, invece, occorre eseguire l'upload di un Entity ID che identifichi univocamente il SP, e del file di metadati generato dal modulo Shibboleth.

Il flusso di comunicazione tra Keystone, Horizon e JumpCloud durante autenticazione federata è rappresentato nel diagramma seguente.



La configurazione dell'accesso federato appena descritta è stata ripetuta sia sul cluster CIRI-ICT sia su quello T3LAB. Sulla piattaforma JumpCloud, si è proceduto a un'appropriata configurazione di utenti, applicazioni e permessi. Come atteso, il risultato è che con le medesime credenziali è possibile accedere e utilizzare, al momento tramite dashboard, entrambe le piattaforme cloud. Si prevede di poter estendere questa possibilità anche all'accesso tramite API.

4.3 Cloud ibrido Kubernetes

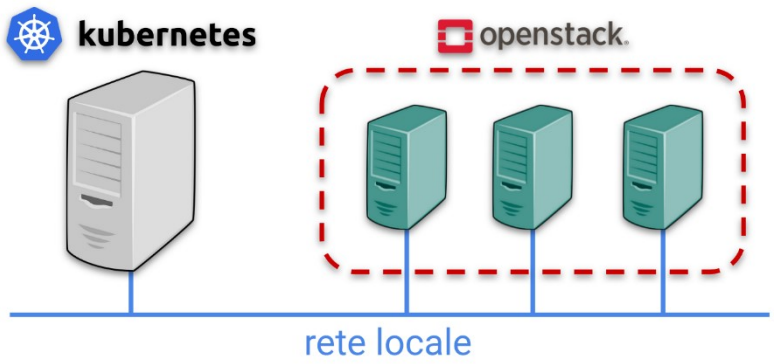
Il cluster Kubernetes realizzato è il risultato dell'attività di analisi di architetture innovative a microservizi del progetto SBDIOI40. Nel contesto di questa attività per *cloud ibrido* si intende la natura

dei nodi che compongono il cluster Kubernetes. Essendo k8s indipendente dall'infrastruttura sottostante, si è deciso infatti di utilizzare Openstack come piattaforma IaaS per dare la possibilità a Kubernetes di utilizzare nodi in forma di macchina virtuale e dei tradizionali server da utilizzare come nodi fisici. Di conseguenza il cluster ibrido è composto sia da nodi fisici che da nodi virtuali. In particolare, l'attuale configurazione, facilmente estendibile, è composta da tre nodi virtuali che comprendono il master e due nodi worker e un nodo worker fisico ad alte prestazioni dalla grande capacità di calcolo.

L'approccio ibrido adottato garantisce i vantaggi di entrambi i paradigmi: da un lato, la possibilità di estendere il cluster in qualunque momento con nodi virtuali addizionali conferisce al sistema una notevole flessibilità; dall'altro lato, la presenza di un nodo fisico consente comunque di mantenere i vantaggi prestazionali derivanti dall'uso di un hardware reale piuttosto che esclusivamente virtuale.

4.3.1 Architettura

L'architettura della soluzione ibrida alla realizzazione del cluster Kubernetes è riportata nel seguente diagramma. Il nodo di dimensione maggiore rappresenta un server fisico, mentre i nodi più piccoli rappresentano macchine virtuali generate e gestite tramite OpenStack. Ogni componente si affaccia sulla medesima rete locale dedicata.



Configurazione hardware

A livello hardware sono stati utilizzati i seguenti server

Hostname	Modello	Core CPU	Memoria	Disco	Rete
cloud-rack-1	Acer AR585F1	32	32GB	15TB	4 x 1Gb
cloud-rack-2	Acer AR585F1	32	32GB	15TB	4 x 1Gb

Sulla macchina cloud-rack-2 è stata effettuata un'installazione all-in-one di OpenStack per il provisioning dei nodi virtuali del cluster mentre il nodo cloud-rack-1 è stato destinato a nodo worker fisico di Kubernetes.

La specifica dei nodi del cluster è la seguente

Hostname	Ruolo	Tipo Nodo	Core CPU	Memoria	Disco	Rete
cloud-rack-1	Worker	Fisico	32	32GB	15TB	1 x 1GB
master.novalocal	Master	VM	2	4GB	40GB	1 x 1GB*
worker-1.novalocal	Worker	VM	2	4GB	40GB	1 x 1GB*
worker-2.novalocal	Worker	VM	2	4GB	40GB	1 x 1GB*

* I nodi master.novalocal, worker-1.novalocal e worker-2.novalocal utilizzano la stessa scheda di rete fisica.

Per quanto riguarda il networking dei vari nodi sono state utilizzate le seguenti configurazioni di rete

Hostname	MAC	IP privato
cloud-rack-1	00:25:90:5A:71:36	192.168.17.101
master.novalocal	FA:16:3E:D8:54:70	192.168.17.151
worker-1.novalocal	FA:16:3E:0D:16:4B	192.168.17.152
worker-2.novalocal	FA:16:3E:EF:A4:6E	192.168.17.153

4.3.2 Configurazione software

Come accennato nella descrizione della configurazione hardware sul server cloud-rack-2 è stata effettuata un'installazione all-in-one di OpenStack per il provisioning delle macchine virtuali master.novalocal, worker-1.novalocal, worker-2.novalocal ed enterprise.

Le configurazioni software delle macchine fisiche e virtuali è la seguente:

Hostname	Sistema Operativo	Note
cloud-rack-1	CentOS 7.7.1908 (Core)	-
cloud-rack-2	CentOS	Openstack Stein
master.novalocal	CentOS 7.7.1908 (Core)	VM
worker-1.novalocal	CentOS 7.7.1908 (Core)	VM

worker-2.novalocal	CentOS 7.7.1908 (Core)	VM
enterprise	CentOS 7.7.1908 (Core)	Ansible e Rancher Server

Per quanto riguarda la configurazione software di Kubernetes, su tutti i nodi ad eccezione della macchina *enterprise* sono stati installati i seguenti pacchetti:

- docker-ce-18.09.1
- docker-ce-cli-18.09.1
- containerd.io
- kubeadm v1.17
- kubelet
- kubectl v1.17

4.3.3 Installazione

Per la creazione e l'avvio del cluster è stato utilizzato **kubeadm**, strumento che consente di installare un cluster Kubernetes conforme ai Kubernetes Conformance Test. Kubeadm fornisce le azioni necessarie per l'inizializzazione di un cluster e per la sua esecuzione. I tre comandi principali sono:

- kubeadm init per avviare un nodo Kubernetes di control plane.
- kubeadm join per avviare un nodo Kubernetes worker e aggiungerlo ad un cluster esistente.
- kubeadm upgrade per aggiornare il cluster Kubernetes ad una nuova versione.

L'installazione è stata facilitata grazie all'utilizzo di **Ansible**, un tool open-source di Red Hat per automatizzare il provisioning di software, la gestione della configurazioni e il deployment di applicazioni in ambito IT. Fornisce inoltre il supporto ad un linguaggio dichiarativo proprio ,conforme alla sintassi YAML, per descrivere le configurazioni di sistema.

Ansible basa le proprie comunicazioni con i nodi target sul protocollo SSH e pertanto risulta un tool particolarmente leggero e che non richiede l'installazione di alcun tipo di agent sui nodi target della configurazione. L'unico accorgimento adottato è stato quello di disporre ,tramite OpenStack, una macchina virtuale indipendente di nome *enterprise* collegata a tutti i nodi su cui è stato installato l'Ansible Server e da cui è stata effettuata la configurazione dell'intero cluster. Attraverso l'utilizzo di script Ansible, chiamati in gergo *playbook*, è stato possibile eseguire per tutti i nodi l'installazione simultanea dei pacchetti software citati precedentemente.

I playbook utilizzati per l'installazione di Kubernetes sono disponibili al seguente indirizzo:

https://github.com/lorciv/kubernetes-ansible

Come add-on per il supporto alla rete inter-Pod è stato installato **Flannel**.

4.3.4 Note aggiunitve

Rancher

Al fine di facilitare l'amministrazione del cluster si è deciso di impiegare Rancher. Questo strumento è un progetto open source che fornisce una piattaforma di gestione di container pensato per le organizzazioni che decidono di utilizzare la containerizzazione in produzione.

Una caratteristica interessante di questo strumento è la possibilità di gestire tramite la sua dashboard web-based un cluster Kubernetes pre-esistente. Pertanto al termine dell'installazione manuale del cluster k8s, è stato installato il Rancher Server sulla stessa macchina virtuale utilizzata per effettuare il deployment di Kubernetes tramite Ansible e infine è stata effettuata la procedura di importazione del cluster k8s. Grazie a questa operazione diventa possibile gestire il cluster direttamente dalla dashboard di Rancher rendendo di fatto superflua l'installazione della dashboard di Kubernetes.

Nonostante Rancher, tra le sue funzionalità, consenta di installare un cluster Kubernetes in modo automatico si è deciso di optare per una procedura di installazione manuale. Il vantaggio di questo approccio è duplice: da un lato il completo controllo dei componenti software installati su ciascun nodo e dell'altro ottenere, con Rancher, tutti i benefici di una piattaforma di controllo per Kubernetes production-ready.

Il Rancher Server da cui è possibile controllare il cluster è disponibile al seguente indirizzo:

https://disi057151.ing.unibo.it/

Disaster Recovery

Si è deciso di utilizzare un control plane composto da un singolo nodo che di fatto rappresenta l'unico punto di fallimento dell'intera piattaforma. Essendo il master l'unico partecipante del cluster ad avere una componente stateful, per far fronte ad eventuali fallimenti o interruzioni inaspettate è stato realizzato un sistema di backup periodico.

Essendo il nodo master una macchina virtuale fornita da OpenStack, il sistema di disaster recovery è stato realizzato utilizzando le funzionalità di backup di macchine virtuali di OpenStack.

5 Bibliografia

- https://docs.openstack.org/keystone/latest/admin/federation/configure_federation.html
- http://www.gazlene.net/demystifying-keystone-federation.html
- https://www.youtube.com/watch?v=jwXenfEOSOk
- https://www.oreilly.com/library/view/identity-authentication-and/9781491941249/
- https://www.youtube.com/watch?v=Td3wOqj0Hqw
- https://cloud.garr.it/support/kb/cloud/federated_auth/
- https://en.wikipedia.org/wiki/SAML_2.0
- https://blog.netapp.com/blogs/containers-vs-vms/
- https://www.cwi.it/data-center/virtualizzazione/container-cosa-funzionano-virtualizzazione-86516
- https://www.infoworld.com/article/3268073/what-is-kubernetes-your-next-application-platform.html
- https://www.hpe.com/it/it/what-is/container-orchestration.html
- https://docs.docker.com/
- https://kubernetes.io/docs/home/
- https://rancher.com/docs/
- https://docs.ansible.com/
- https://github.com/coreos/flannel
- The Complete Kubernetes Guide, Jonathan Baier, Gigi Sayfan, Jesse White ISBN 978-1-83864-734-6
- The Kubernetes Book, Nigel Poulton, Pushkar Joglekar ISBN: 978-1-83898-438-0