ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Dipartimento di Informatica - Scienza e Ingegneria
Corso di Laurea in Ingegneria e Scienze Informatiche

Setup automatizzato di un cloud privato con OpenStack

Tesi di laurea in
Virtualizzazione E Integrazione Di Sistemi

Relatore
Prof. Vittorio Ghini

Candidato
Sauro Camagni

Quarta Sessione di Laurea
Anno Accademico 2021-2022

Indice

Capitolo 1

Introduzione

1.1 Il cloud computing

Il cloud computing è una tecnologia che consente l'erogazione su richiesta di servizi di elaborazione dati, archiviazione, esecuzione di applicazioni e altre tipologie di servizi, il tutto attraverso internet. Questo aspetto implica che i dati e le applicazioni non sono più situati su singoli dispositivi in locale, ma sono dislocati su vari server destinati a funzionare come un unico ecosistema, accessibile da qualsiasi parte del mondo da qualsiasi dispositivo compatibile con una connessione internet.

Oggigiorno sempre più aziende si approcciano all'uso di questa tecnologia, e in base alle loro necessità ed esigenze devono valutare a quale tipologia di cloud computing affidarsi. Esistono tre principali tipologie di cloud computing: pubblico, privato e ibrido.

Cloud pubblico. Il cloud pubblico è un insieme di servizi di elaborazione offerti su una rete pubblica da fornitori di terze parti, i cloud provider, come ad esempio Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform (GCP). Un uso molto comune di un cloud pubblico è per l'utilizzo di applicazioni non critiche, ovvero tutte quelle applicazioni che non conservano dati sensibili o troppo importanti, come siti web o applicazioni mobile. Fortemente scalabile, il cloud pubblico è tra le tipologie più economiche, in quanto non si acquistano componenti hardware bensì l'utilizzo di queste. Ciò comporta che è possibile richiedere quasi immediatamente l'aumento o la diminuzione delle risorse che si necessita in un particolare periodo, senza dover affrontare i costi relativi all'acquisto, alla gestione, alla manutenzione, etc.

Tra i vantaggi più importanti nell'uso di un cloud pubblico si trovano i bassi costi, in quanto non vengono acquistati dispositivi hardware e non bisogna preoccuparsi della gestione, la flessibilità e la scalabilità, dato è in grado di

adattarsi alle esigenze degli utenti e la facilità d'accesso, poiché è sufficiente un dispositivo compatibile ed una connessione internet.

Gli svantaggi più rilevanti sono quelli riguardanti la sicurezza dei dati, dato che non se ne ha un controllo completo e la forte dipendenza dal fornitore del servizio cloud, il quale gestisce le risorse.

Cloud privato. Il cloud privato è la tipologia di cloud computing in cui l'azienda gestisce i propri servizi di cloud direttamente all'interno della propria rete, in genere *on-premise*. Quindi viene implementato un cloud non accessibile dal pubblico generale ma solamente da alcuni utenti selezionati, come ad esempio i vari dipendenti dell'azienda.

Anche in questa tipologia di cloud sono garantiti la maggior parte dei vantaggi del cloud pubblico, come la scalabilità e la flessibilità e in più si ha una personalizzazione aggiuntiva su tutti gli aspetti del cloud. In oltre i cloud privati offrono in media un alto livello di sicurezza e di privacy sui dati, necessari quando bisogna rispettare determinate normative di sicurezza. In generale le prestazioni risultano essere più elevate rispetto ad un cloud pubblico, dato che le risorse possedute sono condivise solamente con il pool di utenti selezionato e non con la miriade di cliente che un cloud provider pubblico potrebbe avere.

La maggior parte dei lati negativi sono incentrati sull'alto costo che necessita mantenere tale infrastruttura, partendo dal semplice acquisto dell'hardware dedicato fino al costo del personale specializzato, della gestione e della manutenzione; infatti è completa responsabilità dell'azienda mantenere funzionante e aggiornato tutto il sistema. Infine, il grado di scalabilità non è così elevato come in un cloud pubblico, poiché essa dipende dalla capacità dell'hardware presente in loco.

Cloud ibrido. Il cloud ibrido è la soluzione che combina gli elementi del cloud pubblico e privato, ottenendo così una soluzione flessibile e personalizzabile, in grado di adattarsi alle esigenze dell'azienda che ne fa uso. Infatti, con il cloud ibrido si ha sia l'implementazione on-premise del cloud privato, sia l'accesso ad un cloud pubblico; in questo modo è possibile separare i servizi che un azienda utilizza e fornisce in base loro grado di riservatezza. Per esempio, le applicazioni e i dati più sensibili resteranno nel cloud privato, come i dati d'accesso degli o i conti corrente, mentre le applicazioni e i dati non critici, come siti web o immagini ad uso pubblico, possono essere gestiti attraverso il cloud pubblico. Questa tecnica consente un risparmio economico nel gestire quegli aspetti su cui non è necessario l'uso di un cloud privato.

I cloud ibridi consentono anche di effettuare la tecnica del *bursting* [3] per gestire i picchi nella domanda IT. Se un'azienda che usa un cloud privato

satura la sua capacità in termini di risorse, il traffico in eccesso viene indirizzato verso un cloud pubblico senza alcuna interruzione dei servizi. Questo aspetto è molto importante quando si devono valutare dei picchi di domanda, dovuti per esempio ad alcuni periodi dell'anno (come offerte particolari o l'apertura di nuovi servizi); quindi, invece di acquistare dell'hardware solo per breve periodi, si demanda il traffico verso un cloud pubblico, pagando le risorse solo quando sono veramente necessarie. In questo modo si è in grado di accendere, aumentare o spegnere servizi e risorse in brevissimo tempo.

Gli svantaggi maggiori sono tutti quegli aspetti complessi relativi alla gestione dei due sistemi in contemporanea, come ad esempio la sincronizzazione tra i due ambienti.

Community Cloud. Il community cloud non è una vera e propria tipologia di cloud computing; si tratta di una condivisione di un'infrastruttura cloud, tipicamente privata o ibrida, tra varie società che hanno esigenze ed obiettivi in comune. Questa infrastruttura cloud può essere gestita da internamente da una di queste società o da un ente terzo e ospitata sempre internamente o esternamente. Esempi di community cloud si possono trovare nella pubblica amministrazione, in settori sanitari e governativi,

In questo modello vi sono numerosi vantaggi, tra cui la condivisione delle spese di gestione dell'infrastruttura tra i vari enti, e una condivisione dei dati e delle risorse molto più immediata.

Gli svantaggi più importanti sono relativi alla sicurezza e la privacy di alcuni dati delle singole aziende che non si vogliono condividere, e ciò spesso porta a dover implementare parallelamente dei cloud privati. Un altro aspetto negativo è quello di dover gestire al negoziazione e accordi tra le varie organizzazioni.

1.2 Progetto di tesi

In questo documento di tesi si affronteranno tutti gli aspetti riguardanti l'implementazione e l'utilizzo di un **cloud privato**, concentrandosi sull'automatizzazione di tale procedure. Il cloud privato verrà implementato utilizzando la piattaforma di cloud computing open source **OpenStack**. In primo luogo, per poter gestire il deployment delle macchine fisiche verrà utilizzato **MAAS**, uno strumento open source per il provisioning di server e host bare-metal. Successivamente per l'installazione e la gestione lato software dei singoli componenti di OpenStack verrà utilizzato **Juju**, un sistema open source per l'automazione, configurazione e deploy di infrastrutture e software per cloud. Sia MAAS che Juju permettono un alto livello di automazione in tutte le fasi, limitando

il numero di operazioni che l'amministratore di sistema dovrà affrontare. A cloud installato, verranno mostrati dei semplici casi d'uso, in primis attraverso la linea di comando e successivamente attraverso lo strumento **Terraform**; quest'ultimo permette la gestione di numerose risorse attraverso del codice di configurazione, bypassando interfacce e terminali.

Questo documento di tesi è strutturato nel seguente modo:

Obiettivi - Definizione dei vari obiettivi intermedi e finali da raggiungere.

Progettazione del cluster - Descrizione dell'architettura fisica e software dell'intero sistema.

Prerequisiti: MAAS e Juju - Descrizione approfondita dei due sistemi, con la loro installazione all'interno del cluster.

OpenStack - Descrizione della piattaforma OpenStack e dei suoi componenti per la creazione di cloud privati.

Installazione di OpenStack - Provisioning del cloud OpenStack attraverso Juju e configurazione da parte dell'amministratore.

Utilizzo di OpenStack - Descrizione sui concetti di base per poter utilizzare OpenStack, con le relative istruzioni per eseguire le operazioni principali.

Terraform - Descrizione completa dello strumento Terraform e gestione delle risorse di OpenStack tramite esso, sia dal punto di vista dell'amministratore sia da quello degli utenti.

Conclusioni e sviluppi futuri - Riassunto del progetto e descrizione dei possibili sviluppi futuri.

Questo progetto di tesi è stato realizzato dal lavoro di *Sauro Camagni* (l'autore di questa tesi) e di *Matteo Bambini*. Tutte le parti riguardanti la messa in atto del cloud privato, dalla progettazione del cluster all'utilizzo di MAAS e Juju fino al deploy di OpenStack, sono state eseguite congiuntamente da ambedue le parti.

A cloud dispiegato, l'autore si è concentrato sul funzionamento di Terraform e su come è possibile creare con esso infrastrutture che usino OpenStack, sia per gli amministratori che per gli utenti.

Matteo Bambini invece ha concentrato i suoi studi sulla completa automazione delle fasi di deploy di OpenStack e sull'implementazione delle tecniche di load balancing. Inoltre, ha utilizzato anch'egli Terraform per avviare istanze di macchine virtuali verificando così il corretto funzionamento del load balancer implementato.

Capitolo 2

Obiettivi

In questo capitolo verranno descritti i principali obiettivi di questo progetto di tesi.

Lo scopo principale è creare un cloud privato OpenStack che si avvicini il più possibile come funzionalità ad uno che potrebbe essere usato in produzione, evitando di utilizzare macchine e apparati di rete di ambiente server, privilegiando quindi computer desktop e attrezzatura economica, senza tuttavia compromettere le performance del cloud. A cloud funzionante, studiare in che modo Terraform si integri con OpenStack e quali funzionalità mette a disposizione.

In questo documento, alcuni concetti verranno dati per scontato mentre altri saranno approfonditi e spiegati nel dettaglio nei vari capitoli.

2.1 Progettazione del cluster

Come prima cosa sarà necessario identificare l'hardware a disposizione e, considerato il vasto numero di componenti software da installare, realizzare un piano di deployment che sfrutti al massimo le risorse presenti. Si cercherà quindi di utilizzare un certo numero di macchine AMD64 e, dove possibile, ripiegare su hardware a basso costo, eventualmente anche di architettura diversa (come ad esempio Raspberry Pi); questo ovviamente verrà fatto dopo aver verificato che non ci siano incompatibilità per via delle diverse architetture.

Per tutte le componentistiche di rete si cercherà di ridimensionare gli apparati in modo tale che siano economici e che abbiano performance adeguate allo scopo prefissato.

2.2 Installazione e configurazione dei sistemi di gestione del cloud

Il tipo di deployment che è stato scelto dopo un analisi di massima comporta l'utilizzo di sistemi avanzati di gestione del cloud, il cui scopo è quello di semplificare il provisioning delle macchine fisiche e l'installazione del cloud stesso; nello specifico, questi sistemi sono MAAS e Juju (approfonditi rispettivamente nei capitoli 4 e 5). Questi sistemi devono essere installati e configurati correttamente prima di procedere con la vera e propria installazione di OpenStack.

2.3 Installazione del cloud OpenStack

Una volta che i sistemi MAAS e Juju saranno operativi, sarà possibile procedere con l'installazione di OpenStack. Anche questa sarà effettuata attraverso Juju, il quale consente di installare sia un componente alla volta che tutti i componenti in maniera completamente automatizzata. Come scritto nella guida all'installazione di OpenStack [32], è consigliato effettuare i primi deploy installando un componente alla volta, in modo tale da comprendere a pieno quali componenti formeranno il cluster e in che modo questi interagiranno tra loro.

Quando il cloud sarà operativo, si configurerà l'ambiente attraverso il profilo d'amministratore e si andrà ad effettuare un primo utilizzo attraverso un utente base.

2.4 Integrazione tra Terraform e OpenStack

In ultimo, dopo aver ottenuto un cloud OpenStack perfettamente funzionante, si andrà a studiare lo strumento Terraform e che tipo di supporto offre verso i cloud provider, nello specifico verso OpenStack. Quindi si verificheranno quali funzionalità saranno disponibili e tramite queste si proveranno ad effettuare le stesse configurazioni realizzate a cloud operativo dall'amministratore di sistema e dall'utente base.

Capitolo 3

Progettazione del cluster

All'interno di questo capitolo verranno descritte quelle che sono state le scelte di progettazione del cluster riguardanti architettura hardware, metodologie di deployment e architettura di rete e i motivi per cui sono state fatte determinate scelte.

3.1 Hardware

L'hardware utilizzato per lo svolgimento di questa tesi è descritto in tabella 3.1.

Tabella 3.1: Hardware utilizzato per il progetto.

Oggetto	**Quantità**
Raspberry Pi 4 8 GB	2
PC con architettura AMD64	6
HDD 1TB	12
Switch layer 2	1
Adattatore Ethernet USB	1
Cavi ethernet	9

Nello specifico i PC con architettura AMD64 hanno un processore Intel Core i5-4460S, 8 GB di RAM e due HDD da 1 TB ciascuno. Inizialmente uno solo dei due HDD era collegato alla scheda madre (il secondo era comunque già all'interno della macchina), quindi è stato necessario aprire ciascun computer e collegare gli hard disk alla scheda madre con un cavo SATA. Per fare questo è stato necessario scollegare il lettore CD perché tutte le schede madri delle macchine a disposizione hanno solamente due porte SATA.

Il motivo per cui è stato necessario utilizzare due HDD su ciascuna macchina è che Ceph, ovvero il servizio che fornisce lo storage, necessita di un hard disk fisico dedicato per poter funzionare e non può condividerlo con il sistema operativo.

Per quanto riguarda i Raspberry Pi invece, era previsto di utilizzarne solamente uno ma, successivamente alla scelta della metodologia di deployment (descritta nella sezione 3.2), è stato deciso di utilizzarne uno aggiuntivo su cui installare MAAS, in modo da avere una macchina in più da poter dedicare effettivamente a OpenStack.

3.2 Metodologia di deployment

Nel momento in cui sono iniziati i lavori per questo progetto di tesi l'ultima release di OpenStack era quella denominata *Yoga* e supportava le seguenti metodologie di deployment [31]:

- Charms Deployment
- Ansible in Docker Containers
- OpenStack-Ansible (con container LXC o Bare Metal)
- TripleO

Charms Deployment. Il Charms Deployment prevede l'utilizzo di MAAS per gestire il deployment delle macchine fisiche e di Juju per l'installazione e la gestione dei singoli componenti di OpenStack all'interno di ciascuna macchina. Il vantaggio di questo tipo di deployment è che l'installazione e la configurazione hanno un livello di automazione molto elevato e questo facilita tutto il processo di deployment e, dato che i servizi di OpenStack che vanno installati e configurati sono numerosi, i benefici di questo tipo di automazione non vanno sottovalutati.

Gli svantaggi principali di questa metodologia di deployment sono tre:

1. è necessaria una conoscenza abbastanza approfondita di MAAS e Juju oltre che di OpenStack
2. i parametri di configurazione dei charm Juju sono limitati e in caso di problemi è difficile eseguire il debug o modificare le configurazioni
3. MAAS e Juju richiedono ciascuno una macchina dedicata per funzionare, quindi c'è un overhead di macchine importante, soprattutto in situazione in cui si ha a disposizione una quantità di hardware limitata come in questo caso

Alla fine, nonostante gli svantaggi, questa è la metodologia di deployment scelta per lo svolgimento di questo progetto di tesi, e i suoi dettagli sono spiegati nella relativa documentazione [27].

Ansible in Docker Containers Questa metodologia di deployment consiste nell'utilizzare Kolla, un servizio di OpenStack, per installare ciascuno dei componenti necessari al cluster all'interno di un container Docker. È possibile sia installare tutto in una singola macchina che fare un deployment distribuito, quindi suddiviso tra più macchine; tutti i comandi necessari a installare e configurare i container vengono eseguiti attraverso un playbook Ansible. Il vantaggio di questo tipo di deployment è che è possibile dedicare tutte le macchine che si hanno a disposizione al cluster OpenStack mentre lo svantaggio principale è la maggiore difficoltà nella configurazione rispetto al Charms Deployment. Un altro problema è che i requisiti minimi di questo tipo di deployment impongono che ciascuna macchina abbia due schede di rete; questo vincolo ha reso impossibile utilizzare questa metodologia in questo progetto perché le macchine a disposizione hanno una sola interfaccia di rete e procurarsi ulteriori schede di rete avrebbe aumentato sia costi che i tempi per la realizzazione del progetto.

OpenStack-Ansible OpenStack-Ansible è molto simile a Ansible in Docker Containers come metodologia di deployment con l'unica differenza che i servizi invece che essere installati su container Docker sono installati direttamente sulla macchina fisica o su container LXC. Anche i vantaggi e gli svantaggi sono i medesimi e anche questa metodologia è stata scartata perché i requisiti minimi richiedevano due schede di rete su ciascuna macchina.

TripleO TripleO è un servizio di OpenStack che mira a installare, gestire e operare un cluster utilizzando l'infrastruttura cloud proprietaria di OpenStack. Ovviamente questo comporta dei costi visto che l'hardware viene fornito dal cloud provider a noleggio, quindi questa metodologia di deployment è stata scartata a priori senza ulteriori approfondimenti.

3.3 Architettura del cluster

In figura 3.1 è descritta l'architettura del cluster. Come enunciato in precedenza l'insieme dell'hardware a disposizione è composto da due Raspberry Pi, sei computer desktop, uno switch, un adattatore ethernet USB e cavetteria varia per i collegamenti. Tutti i Raspberry Pi e tutti i PC sonno stati collegati allo switch tramite cavo ethernet.

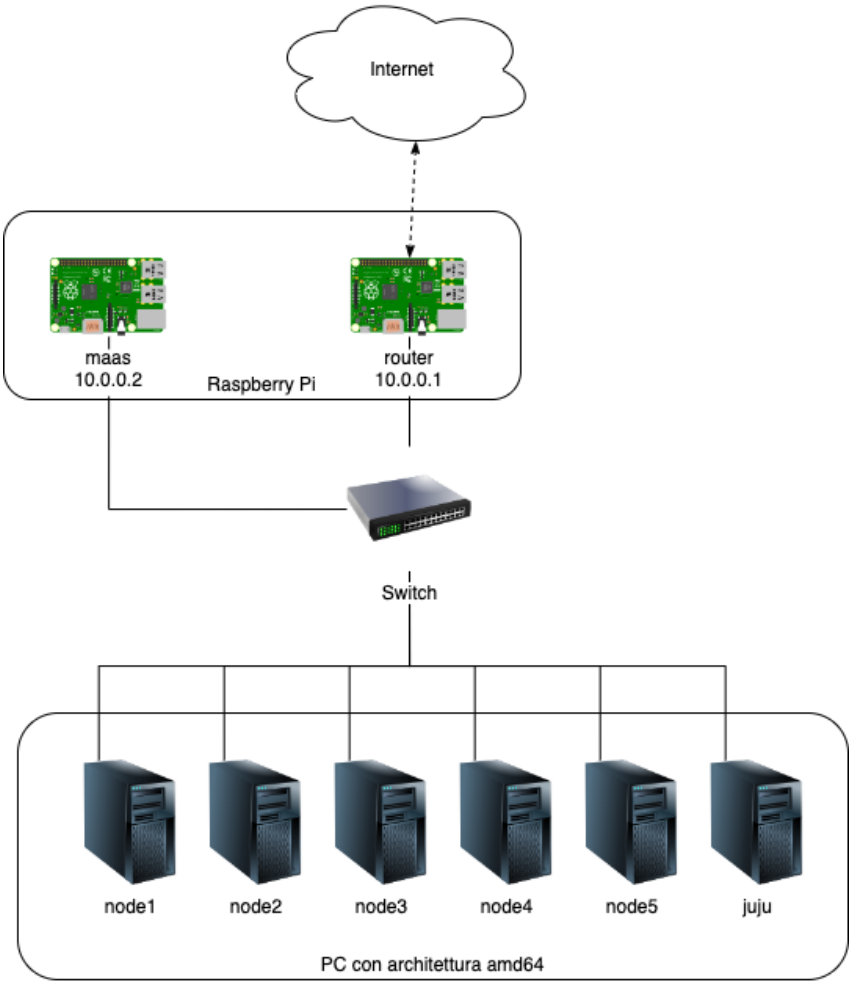


Figura 3.1: Architettura di rete del cluster.

Al Raspberry denominato nello schema *router* è stato collegato l'adattatore ethernet USB, fornendogli un'interfaccia di rete aggiuntiva; in questo modo è stato possibile collegarlo ad internet. Il ruolo di questo Raspberry Pi è esclusivamente quello di comportarsi come un dispositivo di routing, connettendo così le due reti. La configurazione di questo dispositivo non verrà trattata in questo documento.

Il secondo Raspberry, denominato *maas*, è quello su cui è stato installato l'intero sistema MAAS; la guida al deployment che è stata seguita prevedeva che MAAS fosse installato su una macchina simile a quelle utilizzate per il cluster ma, dopo un'attenta valutazione, è stato deciso di installarlo su un Raspberry Pi per avere una macchina aggiuntiva da dedicare a OpenStack.

Sul PC denominato *juju* è stato installato il Juju Controller, ovvero il

componente che gestisce e monitora tutti i charm che verranno installati sui nodi.

I PC denominati *nodeX* sono quelli dedicati al cluster vero e proprio, ovvero quelli su cui verranno installati tutti i componenti di OpenStack.

Inizialmente si era pensato di pianificare in quale macchina installare ciascun componente ma poi, dato che tutte le macchine sono uguali e che la guida di installazione non faceva distinzione tra le diverse macchine, si è deciso di non farlo. Inoltre la quinta macchina è rimasta inutilizzata nel deployment iniziale ma si è rivelata fondamentale durante l'installazione del load balancer per distribuire i servizi e non sovraccaricare le altre macchine (lavoro svolto da Matteo Bambini come spiegato nella sezione 1.2).

3.3.1 Networking

Grazie alla modalità di deployment scelta, la configurazione della rete è molto semplice e gli indirizzi IP assegnati appartengono alla subnet `10.0.0.0/24`. Come si può vedere dall'immagine in figura 3.1 ai Raspberry Pi *router* e *maas* sono stati assegnati rispettivamente gli indirizzi IP `10.0.0.1` e `10.0.0.2`; le altre macchine invece non hanno indirizzi IP statici perché è il MAAS controller che si occupa di assegnarli al momento del deployment.

La scelta della subnet e degli indirizzi IP da assegnare a ciascuna macchina è totalmente arbitraria; sono stati scelti questi perché sono gli stessi utilizzati nella guida all'installazione [32]. L'unico vincolo presente per la configurazione della rete è che almeno una delle subnet private di classe A rimanga inutilizzata; questo perché LXD, durante l'inizializzazione automatica, sceglie tra queste subnet una da utilizzare per le proprie interfacce di rete virtuali e questa scelta viene fatta in base a quelle che non sono raggiungibili dalla macchina host [19]. Se questo vincolo non viene rispettato l'inizializzazione automatica di LXD fallisce e, anche nel caso in cui si tenti di farla manualmente, i container creati non saranno raggiungibili tramite rete.

Nelle fasi iniziale di questo progetto la rete era stata configurata sulla subnet `10.0.0.0/8` e questo, come si può intuire, ha causato non pochi problemi durante l'installazione dei charm su container perché l'inizializzazione di LXD stesso falliva senza dare messaggi di errore esplicativi. Il primo approccio di risoluzione è stato quello di inizializzare LXD manualmente; questo ha permesso di avviare i container contenenti i charm, ma è stato da subito chiaro che la comunicazione via rete non funzionasse correttamente. Dopo ulteriori tentativi e ricerche è stato possibile individuare il problema legato alla subnet e, dopo una riconfigurazione totale della rete, ha funzionato tutto come previsto.

Capitolo 4

MAAS

MAAS (Metal-as-a-Service [20]) è uno strumento open source e gratuito di provisioning di server e di host bare-metal creato e sviluppato da Canonical. Ha come compito quello di aiutare, facilitare e automatizzare l'implementazione e il provisioning dinamico su ambienti di elaborazione iperscalabili (hyperscale computing enviroments) come cloud service o big data workloads. Per fare questo, MAAS collabora con diversi servizi come Juju per coordinare applicazioni e carico di lavoro, riuscendo così a distribuire hardware e servizi che possono scalare dinamicamente verso l'alto e verso il basso.

Permette quindi il monitoraggio e il rilevamento automatico dell'infrastruttura, la creazione di cloud bare metal con server on-demand, il deploy automatizzato di immagini anche con applicazioni preinstallate, la configurazione completa della rete e dello storage e il testing e commissioning dell'hardware.

4.1 Funzionamento

4.1.1 I Controller

La struttura di MAAS è suddivisa in due tipi di controller: un singolo region controller (regiond) e uno o più rack controller (rackd). Nell'esempio riportato in figura 4.1 viene mostrato uno scenario dove il region controller gestisce due rack controller.

Region Controller. Il region controller è il cuore di MAAS; gestisce i rack controller fornendo loro le immagini da utilizzare per il provisioning delle macchine. Utilizza un database PostgreSQL per mantenere lo stato dei nodi registrati al sistema e ospita alcuni dei servizi di rete principali, come ad esempio il server DNS o il proxy HTTP. Inoltre comunica con l'utente attraverso un'in-

terfaccia web e una serie di API REST, dalle quali è possibile configurare e gestire tutto il sistema.

Rack Controller. Il rack controller gestisce effettivamente le macchine collegate al sistema, occupandosi del deploy delle immagini fornite dal region controller. Gestisce la rete delle macchine, fornendo servizi come DHCP per l'assegnamento degli indirizzi IP, PXE per rendere possibile l'avvio da rete, TFTP per il trasferimento file, etc. Vengono chiamati rack controller perché idealmente gestiscono individualmente singoli armadi rack con le relative macchine. Inoltre ogni rack controller è collegato via rete ad un "fabric" (vedasi la sezione 4.1.3 per maggior dettagli).

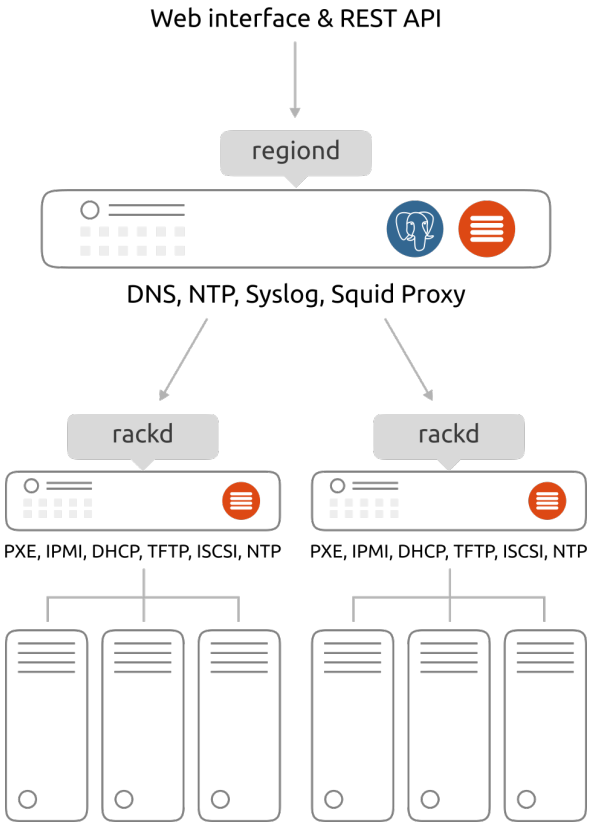


Figura 4.1: Configurazione con un region controller (regiond) e due rack controller (rackd) [22].

Per un sistema di piccole dimensioni è possibile collocare sia il region controller che il rack controller sulla stessa macchina. Durante lo svolgimento di questa tesi è stato scelto questo approccio, in quanto trattasi di un piccolo scenario di prova del sistema.

4.1.2 Risorse: i Nodi

I nodi sono gli oggetti registrati su MAAS e ve ne sono di tre tipi:

- *Controllers*: sono i nodi che assumono il ruolo di region controller e rack controller.
- *Machines*: sono i nodi che vengono gestiti tramite MAAS, ovvero quelli il cui deployment è gestito da MAAS.
- *Devices*: sono altri dispositivi collegati alla rete che non vengono gestiti da MAAS ma che sono stati rilevati, come ad esempio router.

Ad ogni *machine* è associata un etichetta che ne identifica lo stato attuale del lifecycle. I principali stati del lifecycle di una machine sono:

- *New*: quando una nuova risorsa viene collegata alla rete del rack controller, MAAS la rileva in automatico (fase di enlist); quindi l'aggiunge, registra il suo indirizzo MAC e gli associa lo stato *New*.
- *Commissioning*: è la fase durante la quale vengono raccolte e registrate le informazioni sulla configurazione dell'hardware della macchina, come quantità di RAM, numero di CPU, spazio sui dischi e altri device presenti sulla macchina stessa (e.g. GPU).
- *Ready*: una volta terminata con successo la fase di *Commissioning*, lo stato della macchina viene modificato in *Ready*.
- *Allocated*: questo stato indica che una macchina in *Ready* è stata allocata ad un utente ed è pronta per il deploy.
- *Deploying*: durante questa fase viene installato il sistema operativo in maniera completamente automatica, applicando le varie configurazioni che sono state scelte.
- *Deployed*: è lo stato che identifica una macchina come utilizzata, ovvero con il sistema operativo installato e in funzione.
- *Releasing*: se una macchina non è più necessaria è possibile eseguire il *release*, ovvero rilasciarla in modo che possa essere riutilizzata per altri scopi. Durante questa fase è possibile cancellare i dati dei dischi, scegliendo il grado di profondità dell'operazione.

Oltre a questi, esistono altri stati che permettono di identificare il malfunzionamento dei nodi o di un'azione intrapresa su di essi, come ad esempio *Failed testing*, *Failed Commissioning*, *Failed Deployment*, *Broken*, *Locked*, etc.

4.1.3 Gestione della rete

Una corretta gestione della rete di provisioning è un punto cruciale per il corretto funzionamento di MAAS. La rete viene gestita da MAAS tramite l'uso di *fabric* e di *space*.

Fabric. Il fabric concettualmente corrisponde ad uno switch o ad una combinazione di switch che utilizzano il trunking (VLAN Trunking Protocol, VTP [47]) per fornire accesso a specifiche VLAN (Virtual LAN). Il fabric racchiude un insieme di VLAN, a cui appartengono le sottoreti, che possono anche essere controllate da MAAS; in questo modo si rende possibile la comunicazione tra le varie VLAN appartenenti allo stesso fabric e ciò permette a MAAS di ricoprire anche il ruolo di server DHCP.

In una sottorete gestita in questo modo da MAAS è possibile amministrare gli indirizzi IP sia per riservarne un pool per usi diversi dal provisioning degli host, sia per gestirli dinamicamente per usarli e associarli automaticamente durante l'enlist, commission e deploy dei nodi.

Space. Le sottoreti possono essere raggruppate tra loro anche se appartengono a fabric differenti. In questi raggruppamenti, gli space, le sottoreti possono comunicare direttamente tra loro. Ciò è utile in caso si desideri separare i nodi in base all'utilizzo o per motivi di sicurezza. Ogni space ha un indirizzo IP e una subnet mask, e i nodi assegnati ad esso possono comunicare tra loro attraverso questa rete logica.

Un uso comune è lo space DMZ, usato per raggruppare le sottoreti che espongono un'interfaccia web verso la rete internet pubblica; dietro questa DMZ si possono trovare ad esempio le applicazioni che non possono interagire direttamente con l'utente ma che devono invece interagire con un'interfaccia Web all'interno dello space DMZ. Inoltre, gli space facilitano l'allocazione delle macchine per Juju.

Durante l'installazione, MAAS crea un fabric di default ("fabric-0", "fabric-1", etc) per ogni subnet rilevata, mentre non crea alcuno space. Nello scenario d'esempio di questo documento verrà utilizzata la subnet `10.0.0.0/24` e dunque MAAS assocerà a questa il "fabric-0", mentre non è stato creato nessuno space in quanto non si è rilevato necessario.

Il diagramma in figura 4.2 mostra un esempio generico di data center contenente due fabric, ognuno delle quali contiene due rack controller, diverse VLAN e uno space in comune tra i due fabric.

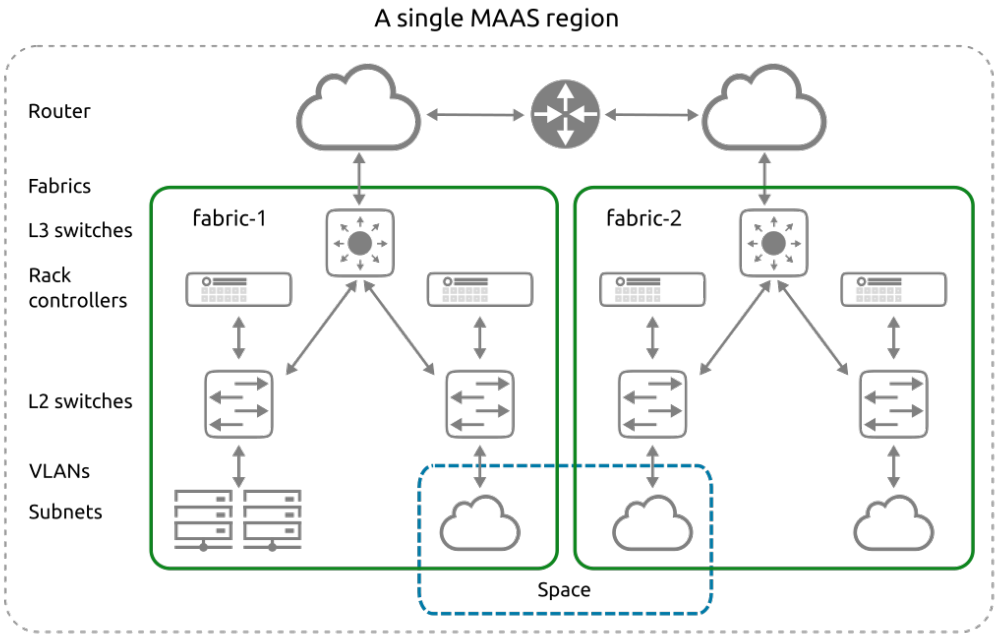


Figura 4.2: Esempio generico di configurazione di rete di un data center [21].

4.1.4 Requisiti del sistema MAAS

Il sistema che ospita MAAS non richiede un hardware eccessivamente prestante, tuttavia a seconda della configurazione finale i requisiti possono variare notevolmente. Di seguito verranno mostrati due scenari d'esempio con i relativi requisiti di sistema stimati da Canonical [26].

Test environment. Questo è uno scenario proof-of-concept ed è l'ideale per testare e provare le potenzialità di MAAS prima di procedere con la vera e propria messa in produzione. È un tipo di scenario minimale, quindi non è necessario disporsi di macchine estremamente performanti. Nella tabella 4.1 viene mostrata la stima delle risorse richieste da ogni componente per questo scenario.
Per questo tipo di scenario, ogni componente può essere eseguito sullo stesso host. In questo modo le risorse richieste sul singolo host diventano approssimativamente la somma delle singole specifiche: 2 GB di RAM, CPU da 2 GHz e 20 GB di spazio su disco.

Tabella 4.1: Risorse richieste per uno scenario proof-of-concept.

	TEST ENVIRONMENT		
	RAM (MB)	CPU (GHz)	DISK (GB)
Region controller (senza PostgreSQL)	512	0.5	5
Rack controller	512	0.5	5
PostgreSQL	512	0.5	5
Ubuntu Server	512	0.5	5

Production environment. Questo è il tipico scenario con cui poter approcciarsi nelle prime messe in produzione di un cloud gestito con MAAS.

Nella tabella 4.2 viene mostrata la stima delle risorse richieste da ogni componente per questo scenario. È bene evidenziare che le risorse richieste sono influenzate da diversi fattori, tra cui: complessità del sistema, numero di rack controller, numero dei nodi collegati a ciascun rack controller, carico sul region controller e numero di immagini da mantenere.

Tabella 4.2: Risorse richieste per un ipotetico scenario in produzione.

	PRODUCTION ENVIRONMENT		
	RAM (MB)	CPU (GHz)	DISK (GB)
Region controller (senza PostgreSQL)	2048	2.0	5
Rack controller	2048	2.0	20
PostgreSQL	2048	2.0	20
Ubuntu Server	512	0.5	5

In questo modo è possibile creare uno scenario in produzione con le seguenti caratteristiche:

- un region controller (incluso PostgreSQL) installato su un host avente 4.5 GB di RAM, CPU da 4.5 GHz e 45 GB di spazio su disco;

- un rack controller installato su un host avente 2.5 GB di RAM, CPU da 2.5 GHz e 40 GB di spazio su disco.

I requisiti descritti per questo sistema non coprono le specifiche dei nodi che verranno collegati al rack controller bensì solamente all'infrastruttura MAAS. In ultimo, un rack controller non dovrebbe gestire più di 1000 nodi, indipendentemente da come sono suddivisi tra le sottoreti.

4.2 Installazione

Preparazione hardware. Come anticipato nella sezione 3.1, il cloud progettato in questa tesi è costituito da:

- Una macchina dedicata all'installazione completa del sistema MAAS (sia il region che il rack controller come spiegato in fondo alla sezione 4.1.1).
- Una macchina dedicata al controller Juju (vedasi la sezione 5.2).
- Quattro nodi sui quali verrà effettivamente installato il cloud OpenStack più un quinto per sviluppi successivi.

In questo capitolo verrà trattata la sola installazione del sistema MAAS.

Date le piccole dimensioni del cloud e del sistema progettato in questa sede, e dato che il rack controller deve gestire un numero esiguo di macchine, non è necessario disporre di una potenza computazionale eccessivamente elevata. Si è quindi deciso di adottare come macchina per MAAS l'SBC Raspberry Pi 4 Model B (come spiegato nel capitolo 3.1) con le seguenti caratteristiche:

> 8 GB di RAM, CPU quad-core ARM64 da 1.5 GHz e 128 GB di storage su scheda micro SD (per ulteriori dettagli sulle specifiche, si veda [37]).

Nonostante la frequenza della CPU sia leggermente inferiore a quella consigliata da Canonical nei vari scenari d'esempio (sezione 4.1.4), durante tutto lo sviluppo non sono stati riscontrati né rallentamenti né altre problematiche all'interno del sistema MAAS.

Preparazione software. Il sistema operativo installato sul Raspberry è *Raspberry Pi OS* [36], basato sul sistema Debian 11 (la quale installazione in questa tesi non verrà trattata), mentre la versione di MAAS installata è la 3.1.0. Durante tutte le fasi di installazione, verrà utilizzato il package manager *snap* per l'installazione e gestione del software.

L'intero sistema è situato all'interno della sottorete `10.0.0.0/24`, il Raspberry ospitante MAAS ha l'indirizzo IP statico `10.0.0.2` (vedere capitolo 3.3.1 per maggiori dettagli). Inoltre il sistema MAAS sarà l'unico fornitore dei servizi DHCP e DNS all'interno della rete.

N.B. Rispetto ad un'installazione da manuale basata sul sistema operativo Ubuntu 20.04 con architettura AMD64, quella affrontata in questa sede differisce leggermente essendo basata sul sistema operativo Raspberry Pi OS (Debian 11) con architettura ARM64. Tutte le eventuali differenze riscontrate verranno spiegate e mostrate nei dettagli.

Per maggior informazioni sulle seguenti fasi di installazione, fare riferimento alla guida OpenStack [24] e alla documentazione MAAS [23].

4.2.1 Installazione del region e rack controller

Durante questa fase di installazione, verranno utilizzati comandi da terminale per installare MAAS sul Raspberry.

Come prima cosa, verrà installato MAAS 3.1.0.

```
1 sudo snap install maas --channel=3.1/stable
```

Listato 4.1: Installazione di MAAS.

Successivamente verrà scaricato e configurato il database PostgreSQL che verrà utilizzato da MAAS in questo scenario. Questo passaggio è opzionale e non è da eseguire se si desidera utilizzare un database PostgreSQL esterno.

```
1 sudo snap install maas-test-db
```

Listato 4.2: Installazione del DB PostgreSQL.

Dopo aver installato MAAS, bisogna inizializzare il sistema e configurare region e rack controller. Eseguire `sudo mass init --help` per maggiori dettagli.

```
1 sudo maas init region+rack --maas-url http://10.0.0.2:5240/MAAS --
    database-uri maas-test-db:///
```

Listato 4.3: Inizializzazione del region e rack controller del sistema MAAS.

- Indicando al comando `init` il valore `region+rack`, viene specificato che il sistema MAAS installato avrà il ruolo sia di region controller che di rack controller.

- Con l'argomento `--maas-url` viene specificato l'indirizzo URL dove è situato l'accesso per l'interfaccia utente web; essendo gestito dal region controller, va indicato il suo indirizzo IP. In questo caso è la stessa macchina Raspberry, avente indirizzo IP `10.0.0.2`.

  L'indirizzo IP all'interno dell'URL va quindi sostituito nel caso in cui si voglia configurare in maniera differente.

- Con l'argomento `--database-uri` viene specificato l'URI del database PostgreSQL che MAAS andrà ad utilizzare; nel sistema installato per questo progetto viene utilizzato il database di test fornito direttamente da MAAS (listato 4.2), ma è possibile specificare un URI che rimanda ad un database PostgreSQL esterno nel seguente formato:
  `postgresql://[user[:password]@][[host][:port]][/dbname]`
  `[?name=value[&...]]`

A questo punto, verranno create le credenziali di amministratore.

```
1 sudo maas createadmin --username admin --password ubuntu --email
    admin@example.com --ssh-import lp:<usernameLaunchpad>
```

Listato 4.4: Creazione delle credenziali d'amministratore.

- Con l'argomento `--email` viene indicato l'indirizzo e-mail dell'account amministratore; tuttavia non è necessario che l'indirizzo e-mail esista veramente, in quanto in realtà non viene utilizzato da MAAS.

- Con l'argomento `--ssh-import` è possibile inserire la propria chiave ssh pubblica all'interno di MAAS importandola dal proprio profilo GitHub[1] o Launchpad[2]. Per importare la chiave da Launchpad è necessario inserire il nome del profilo preceduto da "`lp:`", per GitHub va sempre inserito il nome del profilo ma questa volta preceduto da "`gh:`". Questa chiave potrà poi essere inserita all'interno dei nodi in fase di deploy e servirà per poter accedere alle singole macchine una volta che il sistema operativo verrà installato da MAAS. È possibile saltare questa configurazione omettendo l'argomento e gestire ulteriori chiavi da interfaccia in un secondo momento (si veda il punto 4 e il paragrafo successivo nella sezione 4.3).

[1] GitHub: `https://github.com`, ultimo accesso 13 Gennaio 2023.
[2] Launchpad: `https://launchpad.net/`, ultimo accesso 13 Gennaio 2023.

In ultimo verrà copiata la chiave API dell'utente "admin" e memorizzata in un file a parte; questa servirà nei passaggi successivi per l'installazione di Juju (listato 5.4 nella sezione 5.2). Nel caso in cui non si voglia salvare questa chiave su file, è possibile successivamente ricavarla da interfaccia web (si veda il terzo paragrafo della sezione 4.3).

```
1 sudo maas apikey --username admin > ~/admin-api-key-file
```

Listato 4.5: Salvataggio della chiave API.

In appendice A.1 è possibile visionare l'elenco dei comandi precedentemente descritti in un unico raggruppamento.

4.3 Configurazione

Giunti a questo punto, è possibile accedere all'interfaccia utente web attraverso l'uso del browser. L'URL dell'interfaccia web a cui collegarsi è quello inserito nel listato 4.3 nell'argomento `--maas-url`, in questo caso:

URL: `http://10.0.0.2:5240/MAAS`

Le credenziali da immettere sono quelle inserite nel listato 4.4, ovvero:

Username: `admin`

Password: `ubuntu`

Al primo accesso verranno presentate delle schermate di benvenuto; da queste è già possibile configurare vari parametri del sistema MAAS che si andrà ad utilizzare.

In questo scenario sono state inserite le configurazioni seguenti:

1. **Region name.** Questa stringa identifica il nome del sottodominio dei nodi secondo la nomenclatura FQDN [4]. In questa sede è stato scelto *oscluster.unibo.it*; ad esempio il node1 sarà identificato come *node1.oscluster.unibo.it*.

2. **DNS forwarder.** Qui vanno indicati gli indirizzi IP dei server DNS esterni che si vogliono utilizzare. Questi serviranno per risolvere i domini che non sono gestiti da MAAS. In questo caso sono stati utilizzati gli indirizzi `8.8.8.8` e `8.8.4.4`, ovvero i server DNS di Google.

3. **Ubuntu.** In questa sezione è possibile scegliere le immagini di Ubuntu da importare selezionandole per sorgente, versione e architettura. Nel caso di questo progetto sono state scaricate dalla sorgente maas.io le immagini 20.04 LTS e 22.04 LTS per architettura AMD64 (come richiesto dalle istruzione di installazione di OpenStack).

   Una volta scaricate le immagini scelte, premere su "*Continue*" per proseguire con la configurazione.

4. **SSH keys for admin.** In questa ultima sezione è possibile aggiungere varie chiavi ssh pubbliche in tre modi.

   I primi due, come menzionato nel listato 4.4, sono attraverso le piattaforme Launchpad e GitHub; una volta selezionata la piattaforma dalla quale si vuole importare la chiave, va indicato lo username preceduto da `lp:` per Launchpad o da `gh:` per GitHub (per esempio `lp:user1` o `gh:user2`).

   Il terzo approccio invece consiste nel caricare la chiave pubblica manualmente copiandola da un file sul proprio computer e cliccando sul bottone "*Import*" per salvarla.

   Una volta importate tutte le chiavi desiderate, premere su "*Go to the Dashboard*" per terminare fase di configurazione guidata.

Al termine della configurazione guidata si ha libero accesso al sistema. Se si vogliono modificare o visionare le impostazioni configurate fino ad ora, basterà premere "*Settings*" nel menù in alto. In "*admin*" sempre nel menù in alto è possibile aggiungere ulteriori chiavi ssh e visionare la chiave API nel caso non fosse stata salvata nel listato 4.5.

A questo punto è possibile aggiungere e gestire i vari nodi e procedere con il deploy delle immagini; tuttavia, prima di proseguire, è importante configurare e abilitare il server DHCP, altrimenti MAAS non sarà in grado di amministrare correttamente i nodi.

5. **DHCP.** Per abilitare il DHCP seguire i seguenti passaggi.

   (a) Premere su "*Subnets*" nel menù in alto; comparirà una schermata che mostra le varie sottoreti che MAAS ha rilevato.

   (b) Premere sulla VLAN desiderata, in genere denominata come *untagged*.

(c) In questa schermata verranno mostrate le informazioni riguardanti la VLAN scelta. Nel menù a tendina "*Take action*" premere su "*Provide DHCP*" per far comparire la schermata di abilitazione del DHCP.

(d) Spuntare "*MAAS provides DHCP*" e "*Provide DHCP from rack controller*". Dopodiché, selezionare la subnet a cui riservare un pool di indirizzi IP che il DHCP andrà ad utilizzare in fase di elinst delle macchine, quindi inserire l'IP di partenza in "*START IP ADDRESS*" e l'IP finale in "*END IP ADDRESS*".

(e) Infine, premere su "*Configure DHCP*".

In questa sede, come mostrato in figura 4.3, è stato scelto un pool di 14 indirizzi IP a partire dal `10.0.0.240` fino al `10.0.0.253`.

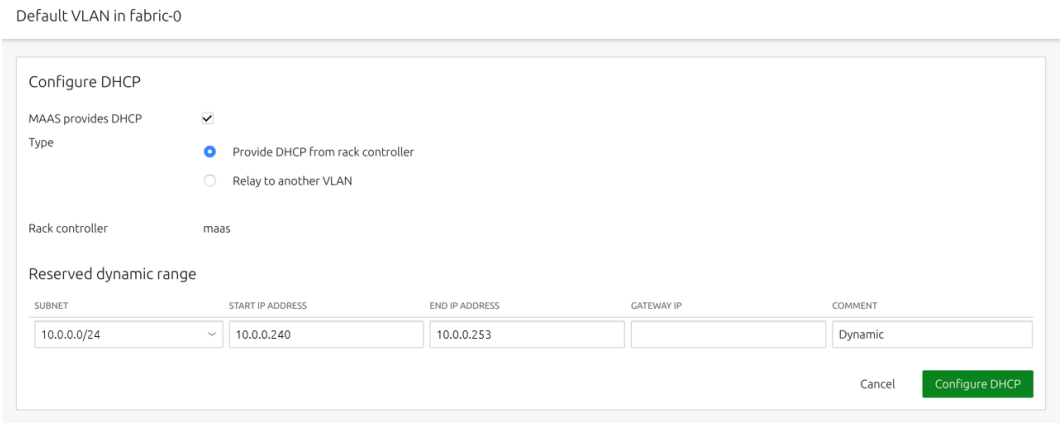


Figura 4.3: Finestra di configurazione del DHPC.

Verifiche finali. Ora il sistema MAAS è perfettamente funzionante. Per verificarlo basterà accedere alla schermata del controller appena configurato.

(a) Premere su "*Controllers*" nel menù in alto, poi premere sul nome del controller desiderato (in questo caso *maas.oscluster.unibo.it*). Verrà mostrata la schermata con le informazioni riguardanti il controller e lo status dei servizi in esecuzione.

(b) Verificare dunque che a fianco ai nomi di questi, come mostrato in figura 4.4, siano presenti le spunte verdi.

Come ultimo passaggio è consigliato verificare che le immagini selezionate nel punto 3 siano state scaricate correttamente.

(a) Premere su "*Images*" nel menù in alto e nella schermata che comparirà.

(b) Verificare che le immagini scelte abbiamo come status "*Synced*".

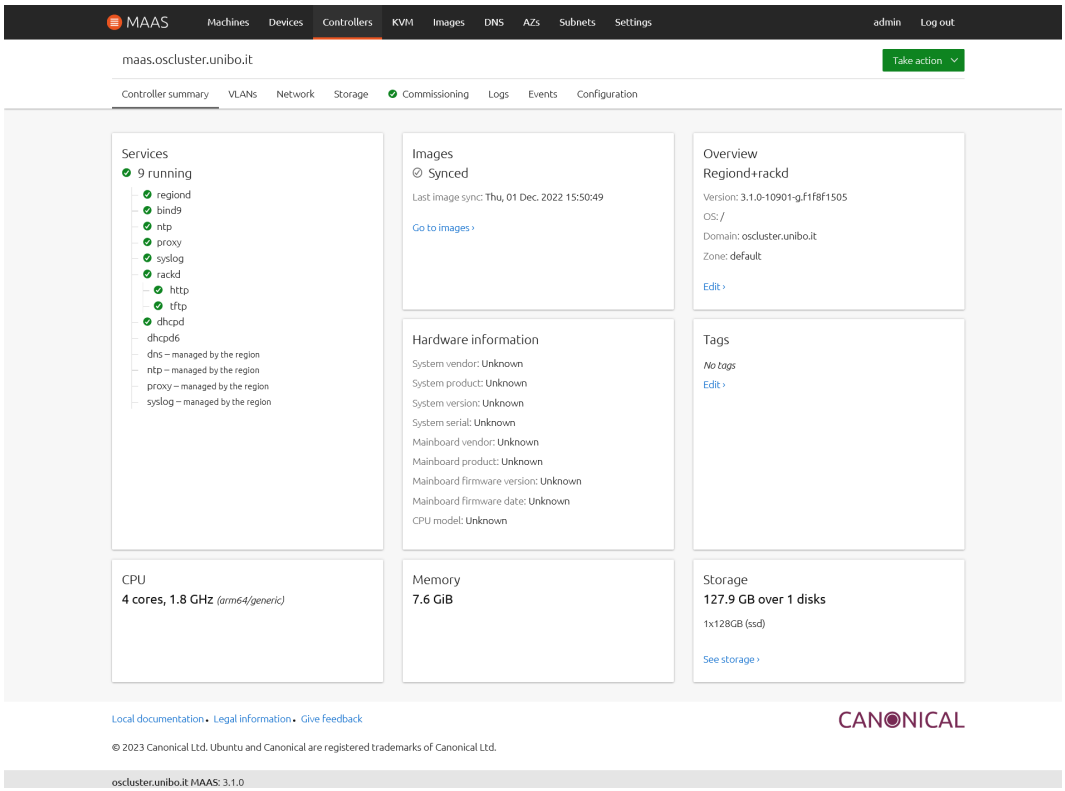


Figura 4.4: Schermata di sintesi del region+rack controller.

Durante lo svolgimento del progetto sono sorti dei problemi con il Proxy HTTP integrato all'interno di MAAS. In particolare esso non permetteva alle macchine di collegarsi a internet e quindi di scaricare i software da installare e gli aggiornamenti. Dato che questa funzionalità non è strettamente necessaria per il funzionamento di tutto il sistema, è stato deciso di disabilitarla tramite la seguente procedura:

1. Premere su "*Settings*" nel menu in alto; in questo modo comparirà la schermata da cui è possibile modificare la configurazione del controller.

2. Premere sul bottone "*Proxy*" nella sottosezione "*Network*".

3. Selezionare la voce "*Don't use a proxy*" e premere il tasto "*Save*" per salvare le modifiche.

4.4 Aggiunta dei nodi

Finalmente è possibile aggiungere le macchine al sistema. Prima di procedere è importante verificare che nel BIOS di ciascuna macchina sia abilitata l'opzione *PXE* (Preboot eXecution Environment), ovvero quella che permette il boot via rete e che sia in cima come priorità d'avvio. In caso di incertezze, consultare il manuale della scheda madre della macchina.

Nonostante la procedura per aggiungere i nodi possa essere eseguita simultaneamente, è consigliabile iniziare aggiungendo una macchina per volta in modo da prendere confidenza con i vari step e identificare correttamente le varie macchine.
Durante le seguenti fasi MAAS assocerà ai nodi uno status in base allo step del lifecycle; per maggiori dettagli si veda la sezione 4.1.2.

1. **Enlist dei nodi.** Collegare la macchina alla subnet del rack controller e accenderla. Una volta effettuato il boot da rete, MAAS la rileva e avvia la procedura di enlist. A processo terminato la macchina si spegnerà e sarà possibile visualizzare il nodo appena aggiunto dall'interfaccia utente web di MAAS premendo su "*Machines*" nel menù in alto. Il nodo appena aggiunto apparirà con status *News*.

2. **Configurazione power type.** La prima cosa da fare dopo aver aggiunto una macchina a MAAS è configurare il power type. Questo parametro indica la metodologia con la quale viene gestita l'alimentazione delle macchine. MAAS infatti ha la capacità di accendere e spegnere le macchine in autonomia nel caso queste siano predisposte (per esempio se sono dotate di una IPMI).

   È possibile configurare il power type di una macchina nel seguente modo:

   (a) Entrare nella schermata di visualizzazione delle macchine tramite il bottone "*Machines*" nella barra di navigazione e cliccare sul nodo che si vuole configurare

   (b) Aprire la scheda "*Configuration*" e premere sul secondo tasto "*Edit*" (quello riguardante la sezione *Power configuration*).

   (c) Selezionare il power type desiderato, configurarlo e al termine premere su "*Save changes*" per applicare le modifiche.

   In questo scenario, dato che le macchine in dotazione non supportano l'avvio e lo spegnimento automatico, è stata scelta la modalità *Manual*, che comporta una gestione manuale da parte dell'accensione e spegnimento delle macchine durante le varie fasi di setup.

Per maggiori dettagli sulle varie tipologie messe a disposizione da MAAS, consultare la documentazione a riguardo [25].

3. **Rinominare i nodi.** È possibile rinominare i nodi, in modo tale che siano facilmente riconoscibili. Per fare questo è sufficiente entrare nella pagina di gestione del nodo, premere in alto a sinistra sul nome corrente e inserire quello nuovo; per salvare i cambiamenti è sufficiente premere invio o cliccare sul tasto "*Save*". In questo scenario sono stati utilizzati nomi incrementativi per quanto riguarda i nodi, da *node1* fino a *node5*, mentre la macchina su cui verrà installato il controller Juju avrà semplicemente il nome *juju*.

Una volta caricati tutti i nodi nel sistema, è possibile proseguire contemporaneamente su tutte le macchine.

4. **Commission.** MAAS ora è pronto per raccogliere le informazioni dei vari nodi. Il tempo necessario al completamento di questa fase dipende da diversi fattori e potrebbe richiedere diversi minuti.

   (a) Dalla pagina "*Machines*" selezionare tutti i nodi spuntando la casella a sinistra del loro nome.

   (b) Premere sul pulsante verde "*Take action*" in alto a destra e poi premere su "*Commission...*".

   (c) A questo punto, se nel punto 2 il power type dei nodi è stato configurato su *Manual*, bisognerà accendere tutti i nodi manualmente. Una volta completata la fase di commission, i computer si spegneranno e i nodi passeranno in stato *Commissioned*.

5. **Tag dei nodi.** È possibile associare ai nodi delle etichette per facilitarne la selezione e gestione durante le fasi di installazione del cloud.

   Dalla scheda "*Configuration*" di un singolo nodo, premere sul primo tasto "*Edit*" e nella casella di testo "*Tags*" aggiungere i tag desiderati.

   In questo scenario sono stati utilizzati il tag *compute* per i nodi del cloud (eccetto per il node5 al quale non è stato associato alcun tag) e il tag *juju* per la macchina con il controller Juju.

6. **Creazione del OvS bridge.** Come ultimo passaggio, è importate creare uno switch virtuale su ogni nodo del cloud (non la macchina juju) che verrà successivamente utilizzato per poter collegare le VM alla rete esterna. Per maggiori dettagli su Open vSwitch, vedasi la sezione 6.2.2.

(a) Dalla pagina del nodo aprire la scheda "*Network*", spuntare la casella a fianco al nome della rete e premere sul tasto "*Create bridge*".

(b) A questo punto compare la finestra per la configurazione del OvS bridge; inserire un nome in "*Bridge name*" uguale per tutti i nodi, selezionare in "*Bridge type*" il valore *Open vSwitch (ovs)*, inserire i parametri di rete corretti (come il "*Fabric*", "*VLAN*", "*Subnet*") e in "*IP mode*" selezionare *Auto assign* per l'assegnamento automatico degli indirizzi IP.

In figura 4.5 viene mostrata la configurazione usata in questo scenario su uno dei quattro nodi del cloud; eccetto per il campo "*MAC address*", la configurazione risulta essere la medesima per ogni nodo.

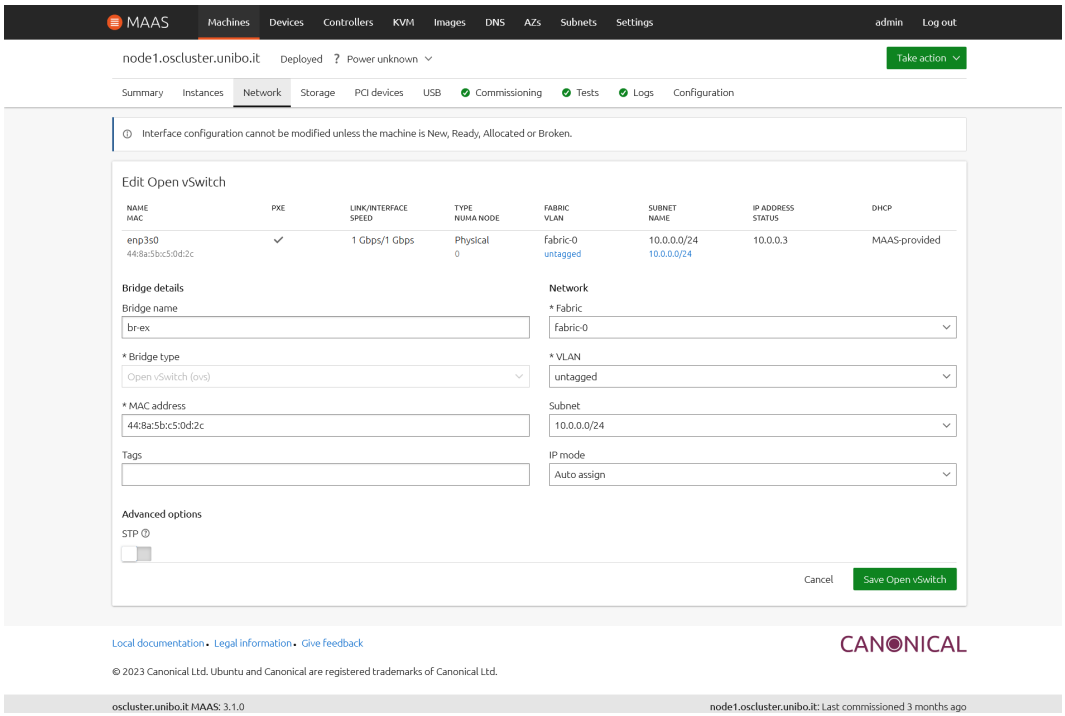


Figura 4.5: schermata di editing della configurazione Open vSwitch su un nodo.

L'installazione e la configurazione di MAAS è terminata. Il prossimo passo sarà quello di introdurre e installare il sistema Juju, il quale consentirà il deploy e la gestione lato software sui vari nodi.

Capitolo 5

Juju

Juju [5] è un sistema open source adibito all'automazione, configurazione e deploy di infrastrutture e software per cloud ibridi e non. Sviluppato anch'esso da Canonical, ha come compito quello di aiutare gli amministratori di sistema nel deploy e semplificare la gestione software del cloud, spostando l'attenzione dalla gestione delle configurazioni delle applicazioni alla gestione delle applicazioni stesse. In questo modo i progettisti di sistemi cloud possono concentrarsi maggiormente sull'amministrazione ad alto livello delle applicazioni e degli scenari, sviluppando modelli per gestire il ridimensionamento, il coordinamento e le dipendenze tra i vari servizi.

Aumento della scalabilità e della ridondanza, semplificazione nella gestione dell'infrastruttura e architettura del cloud e automazione nel deploy sono esempi dei vantaggi che si hanno utilizzando Juju nella creazione dei cloud.

Inoltre è possibile utilizzare Juju per gestire ambienti multi-cloud, ovvero l'utilizzo di più provider di cloud contemporaneamente, come ad esempio AWS (Amazon Web Services) per la parte computazionale e GCP (Google Cloud Platform) per lo storage.

Infine Juju può anche operare in ambienti diversi dai consueti cloud gestendoli comunque come se fossero dei veri e propri cloud, ad esempio server bare metal utilizzando MAAS o container utilizzando LXD. Questa tipologia di cloud viene anche definita *substrate* [9].

5.1 Componenti e funzionamento

5.1.1 Controller

Il controller [10] è il core di Juju nella gestione del cloud. Ha come compito quello di creare e gestire l'infrastruttura software del cloud ed è il responsabile dell'implementazione di tutte le modifiche richieste da parte del client Juju.

Ogni controller generalmente è situato su una macchina dedicata e gestisce un singolo cloud (figura 5.1), quest'ultimi sono descritti attraverso i modelli.

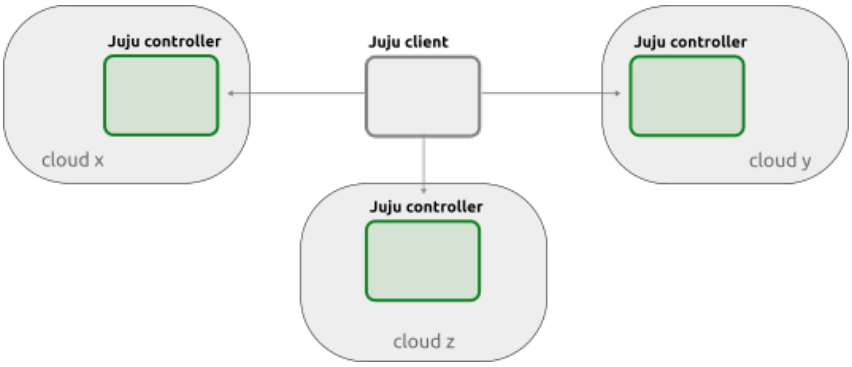


Figura 5.1: Schema del sistema Juju. Un Juju client può comunicare con più Juju controller [8].

Un **modello** [13] è una collezione di applicazioni distribuite all'interno di diverse macchine. Il suo scopo è quello di consentire il raggruppamento logico di applicazioni e infrastrutture che operano e collaborano assieme per fornire un determinato servizio. Un modello è gestito da un singolo controller, mentre un controller può gestire diversi modelli e quindi diversi set di applicazioni e macchine.

Alla creazione del controller vengono generati due modelli: il modello *controller*, che conterrà solamente la macchina del controller, e il modello *default*, un modello generico utilizzabile per distribuire applicazioni e macchine.

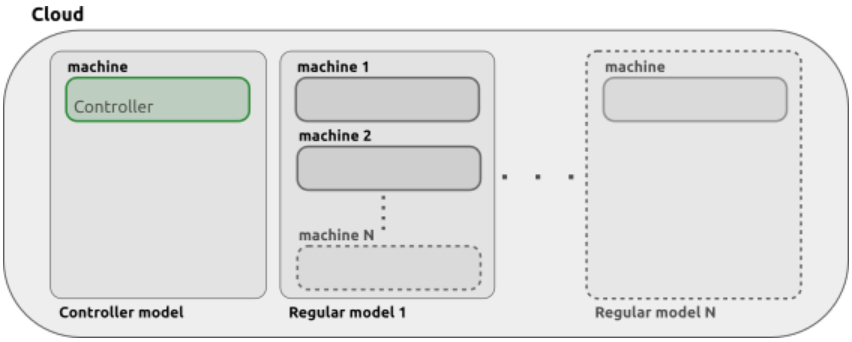


Figura 5.2: Rappresentazione del cloud in modelli. Un singolo Controller model e diversi Regular model su diverse macchine [13].

In sintesi, un cloud è composto da un controller che può gestire vari modelli e ogni modello distribuisce applicazioni su diverse macchine (figura 5.2).

5.1.2 Charmed Operator

Un Charmed Operator [7], o più semplicemente *Charm*, è un componente software open source che guida le fasi del ciclo di vita di una singola applicazione in tutti i suoi aspetti, come l'installazione, distribuzione, configurazione, aggiornamenti e interoperabilità con altre applicazioni. Ne gestisce inoltre le istanze, il ridimensionamento, l'ottimizzazione, il networking, semplificando la distribuzione dell'applicazione, rendendola facilmente riutilizzabile e condivisibile.

I charm sono un'espansione e una generalizzazione della nozione di operatore in Kubernetes, con la differenza che le applicazioni possono essere connesse ad altre applicazioni e possono essere distribuite non solo su cluster Kubernetes ma anche su container, VM, e macchine bare metal, sia su cloud pubblico che privato.

Esiste un'altra tipologia di charm, chiamato *subordinate charmed operator*, che aumenta le funzionalità di un altro charm già esistente (in questo contesto viene denominato come *principal charmed operator*). Quando un subordinate charm viene distribuito, non viene creata una nuova istanza dell'applicazione a cui fa riferimento il principal charmed ma ne estende direttamente le funzionalità.

I charm sono orchestrati dal componente integrato in Juju chiamato *Charmed Operator Lifecycle Manager (OLM)* [14] . Questo strumento ha il compito di distribuire e mantenere aggiornate le applicazioni che i charm descrivono, provvedendo al deploy delle varie istanze delle applicazioni nelle macchine del cloud. Quando vengono scelti in fase di deploy, i charm vengono scaricati dal marketplace *Charmhub* [2], il luogo dove è possibile sviluppare e condividere gratuitamente i propri charm.

Bundle. Un bundle [6] è la rappresentazione di un modello Juju in un file in formato yaml. Al suo interno sono elencati tutti i charm, le relazioni e le configurazioni per poter effettuare il deploy del modello che rappresenta. Infatti partendo da un bundle, è possibile effettuare il deploy in maniera automatizzata di un modello, rendendolo un potente strumento indispensabile per la distribuzione di sistemi grandi e complessi in modo semplice e ripetibile. Come per i charm, è possibile trovare bundle già costruiti da altri utenti in maniera veloce e gratuita nel marketplace *Charmhub* [2].

Overlay. Un overlay è un'estensione del bundle. Anch'esso in formato yaml, il suo compito è quello di permettere la personalizzazione di un bundle esistente senza dover applicare direttamente le modifiche sul file, lasciando quindi inalterato l'intero bundle. In questo modo è possibile utilizzare un bundle

più generico ed applicare delle personalizzazioni tramite overlay, per esempio aggiungendo charm o impostando dei vincoli personalizzati sulle macchine, o ancora modificando il numero di macchine su cui distribuire determinate applicazioni.

5.1.3 Funzionamento delle applicazioni

Unit. Juju definisce le unit [16] come le istanze di un'applicazione in esecuzione e ognuna di queste viene istanziata su macchine distinte. Durante il deploy è possibile decidere quante e su quali macchine installare le unit. Grazie alle unit è possibile dividere una singola applicazione in più istanze, garantendo in questo modo un set di repliche resiliente ai guasti e una suddivisione del carico di lavoro. Ad esempio, come mostrato in figura 5.3 è possibile distribuire il charm mongodb su tre macchine, specificando in fase di deploy sia il numero di unit sia le macchine di destinazione. Ogni unit viene contraddistinta da un numero, preceduto dal nome dell'applicazione; ad esempio la prima unit verrà denominata `nome_app/0`. Infine, tutte le unit della stessa applicazione condividono lo stesso codice del charm, le stesse relazioni e le stesse configurazioni, ma una di esse verrà battezzata come *leader* e sarà responsabile della gestione del lifecycle dell'intera applicazione.

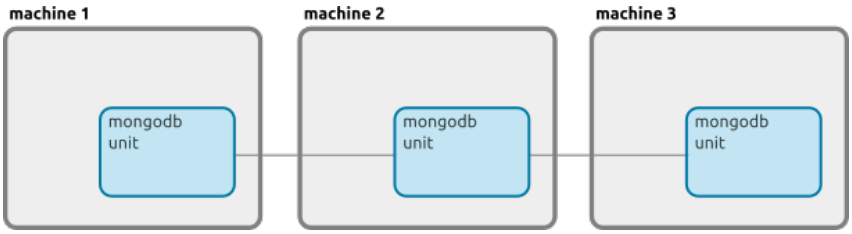


Figura 5.3: Suddivisione dell'applicazione MongoDB su tre unit [16].

Relation. Una relation [15] (integration dalla versione 3.0 di Juju) è un protocollo di Juju che facilita lo scambio di informazioni e configurazioni tra applicazioni. Le relation di un'applicazione sono definite all'interno del charm e vengono create connettendo i vari endpoint delle applicazioni coinvolte. I vari endpoint possono essere connessi solamente se sono dello stesso tipo e se supportano la stessa interfaccia. Per esempio, come mostrato in figura 5.4, il charm wordpress necessita di un database e fornisce un sito web, quindi espone rispettivamente le interfacce "mysql" e "http", alle quali è possibile (e necessario) collegare i charm mysql e apache attraverso le loro rispettive interfacce che a loro volta mettono a disposizione.

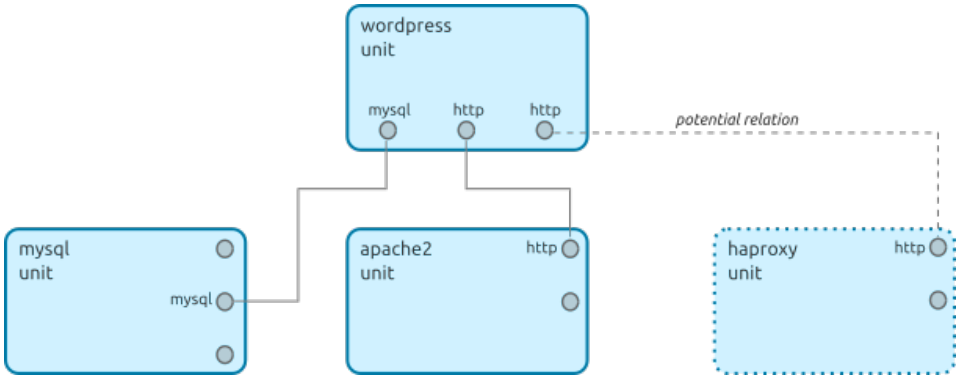


Figura 5.4: WordPress in relation con MySQL e Apache (eventualmente anche con HAProxy) [15].

Le relation possono essere facoltative o obbligatorie, a seconda delle dipendenze che il charm necessita e di come è stato definito dal suo sviluppatore.

Va evidenziato che le relation non sono connessioni dirette tra i vari charm, bensì sono una virtualizzazione delle connessioni per consentire lo scambio di informazioni di configurazioni. Infatti è il controller Juju che ha il ruolo di mediatore per queste connessioni virtuali, gestendo il vario flusso di informazioni tra dli accessi.

5.2 Installazione

Preparazione Hardware. Come anticipato nella sezione 3.1, il sistema cloud progettato in questa tesi è costituito da:

- Una macchina dedicata (Raspberry Pi) per il sistema MAAS (vedasi deploy nella sezione 4.2), a cui in questo capitolo verrà aggiunto il client Juju.
- Una macchina dedicata per il controller Juju.
- Quattro nodi sui quali verrà effettivamente installato il cloud OpenStack più un quinto per sviluppi successivi.

In questo capitolo verrà trattata la sola installazione del sistema Juju, composto dal client e dal controller.

Preparazione software. L'installazione di Juju affrontata in questa sede è affiancata dallo strumento di provisioning server MAAS (capitolo 4). Inoltre, è altre sì importante che la macchina dedicata al controller Juju sia stata

aggiunta all'elenco dei nodi gestiti da MAAS (fase di elinst e commission) Pertanto, prima di procedere, è necessaria effettuare l'installazione completa del suddetto strumento e l'aggiunta dei nodi. Si vedano le sezioni da 4.2 a 4.4 per maggiori dettagli.

La versione di Juju che è stata installata è la 2.9.29, aggiornata poi alla 2.9.37. Il comando che verrà utilizzato in fase di installazione però installerà l'ultima versione disponibile; ciò non dovrebbe risultare problematico, ma è bene prestare attenzione che alcuni aspetti potrebbero essere differenti se si vorrà installare una versione più aggiornata. È comunque possibile, come verrà descritto poi nel listato 5.1, decidere quale versione scaricare e installare.

N.B. Rispetto ad un'installazione da manuale basata sul sistema operativo Ubuntu 20.04 con architettura AMD64, quella affrontata in questa sede differisce leggermente essendo basata sul sistema operativo Raspberry Pi OS (Debian 11) con architettura ARM64. Tutte le eventuali differenze riscontrate verranno spiegate e mostrate nei dettagli.

Per maggior informazioni sulle seguenti fasi di installazione, fare riferimento alla guida OpenStack [12] e alla documentazione Juju [11, 17].

5.2.1 Installazione e configurazione del client Juju

Durante le fasi di installazione di Juju verranno utilizzati solamente comandi da terminale impartiti sulla macchina Raspberry Pi del sistema MAAS, diventando così anche client Juju.

Come prima cosa verrà installato il client Juju all'ultima versione.

```
1 sudo snap install juju --classic
```

Listato 5.1: Installazione del client Juju.

- Nel caso in cui si volesse installare un'altra versione, è possibile specificarla aggiungendo `--channel=<version/release>`. Per esempio, per la versione `2.9.37` si va ad aggiungere `--channel=2.9.37/stable`.

Fatto ciò, MAAS verrà collegato a Juju in modo tale che venga visto e gestito come se fosse un cloud. Prima verrà creato un file yaml di nome *mass-cloud.yaml* contenente le seguenti configurazioni del cloud MAAS.

```
clouds:
  maas-one:
    type: maas
    auth-types: [oauth1]
    endpoint: http://10.0.0.2:5240/MAAS
```

Listato 5.2: File yaml di configurazione del cloud MAAS.

- La dicitura `mass-one` indica il nome che assumerà il cloud. Se si vuole utilizzare un altro nome, bisogna cambiare la dicitura `mass-one` con il nome desiderato.
- In `type` viene indicato la tipologia del cloud; in questo caso `maas`.
- Con `auth-types` si intende il tipo di autenticazione del cloud (verrà aggiunta la chiave di autenticazione nei listati 5.4 e 5.5).
- `endpoint` invece è l'URL, il punto d'accesso di MAAS inserito in fase di installazione nel listato 4.3.

A questo punto è possibile aggiungere il cloud a Juju.

```
juju add-cloud --client -f maas-cloud.yaml maas-one
```

Listato 5.3: Aggiunta del cloud mass-one in Juju.

- Con l'opzione `--client` si va ad indicare a Juju di archiviare la definizione del cloud sulla macchina da cui si sta eseguendo il comando, ovvero il client Juju
- Con l'argomento `-f` viene indicato il nome del file di configurazione yaml salvato nel listato 5.2
- Il valore *mass-one* indica il nome del cloud corrispondente a quello scelto sempre nel listato 5.2.

Per verificare che il cloud sia stato correttamente aggiunto, si può eseguire `juju clouds --client`; l'output dovrebbe essere simile a quello mostrato in figura 5.5.

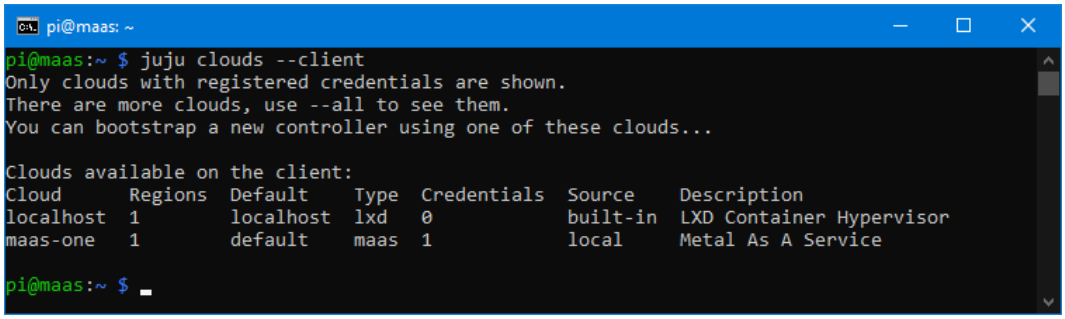


Figura 5.5: Elenco dei cloud presenti nel sistema Juju.

Una volta aggiunto il cloud MAAS, Juju ha bisogno delle credenziali per poter interagire con esso. Verrà creato un nuovo file yaml ad hoc di nome *maas-creds.yaml* avente le seguenti impostazioni.

```yaml
credentials:
  maas-one:
    anyuser:
      auth-type: oauth1
      maas-oauth: "admin-api-key-file"
```

Listato 5.4: File yaml contenente le credenziali del cloud MAAS.

- `mass-one` è il nome del cloud scelto nel listato 5.2.
- `anyuser` è il nome del nuovo utente che Juju andrà ad utilizzare.
- Con `auth-type` si indica la tipologia delle credenziali da inserire.
- Infine in `maas-oauth` va indicata la chiave API che è stata salvata nel listato 4.5; quindi sostituire *admin-api-key-file* con la corrispondente chiave.

Ora è possibile aggiungere le credenziali a Juju.

```
juju add-credential --client -f maas-creds.yaml maas-one
```

Listato 5.5: Aggiunta delle credenziali del cloud MAAS in Juju.

(Vedasi il listato 5.3 per la spiegazione degli argomenti del comando eseguito).

Anche in questo caso è possibile verificare che le credenziali siano state inserite correttamente. Per farlo, basterà eseguire `juju credentials --client --show-secrets --format yaml`; l'output dovrebbe essere simile a quello illustrato in figura 5.6.

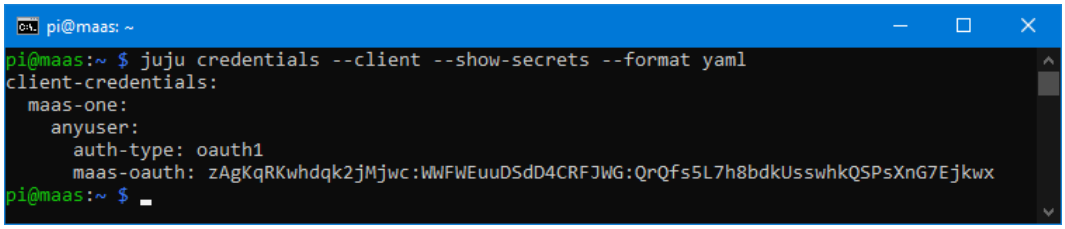


Figura 5.6: Elenco delle credenziali presenti nel sistema Juju.

5.2.2 Deploy del controller Juju e creazione del model per il cloud

Arrivati a questo punto, è possibile avviare il deploy automatizzato del controller Juju. Tale richiesta verrà demandata a MAAS, il quale si occuperà dell'installazione del sistema operativo e del controller.

```
1 juju bootstrap --bootstrap-series=focal --constraints "tags=juju arch=amd64
    " maas-one maas-controller
```

Listato 5.6: Deploy del controller Juju nella macchina con tag juju.

- Con l'opzione `--bootstrap-series` viene specificata la versione dell'immagine del sistema operativo da far installare da MAAS; questa verrà presa tra le immagini scaricate nel voce 3 nella sezione 4.3.

  In questo caso è stata installata Ubuntu Focal, corrispondente alla versione 20.04 LTS.

- Con l'opzione `--constraints` è possibile specificare in maniera precisa l'hardware a cui si sta facendo riferimento. In caso di più valori, questi devono essere inseriti all'interno del carattere doppio apice come mostrato nel listato 5.6.

  In questo caso è stato indicato col valore `tag=juju` che il comando deve essere eseguito per il nodo con il tag "juju".

Inoltre, in questo specifico scenario, si è rilevato indispensabile aggiungere anche il valore `arch=amd64`, in quanto il comando `bootstrap` viene richiesto dal client Juju, situato su Raspberry Pi avente architettura ARM64 e non AMD64 come per i restanti nodi.

Dunque in uno scenario diverso da quello presentato in questo documento, se tutte le architetture di tutte le macchine coinvolte saranno di tipo AMD64 non sarà necessario l'aggiunta del suddetto valore.

- *mass-one* è il nome del cloud scelto nel listato 5.2, mentre *maas-controller* sarà il nome che assumerà il controller Juju, ovvero il controller del cloud.

A comando avviato, dall'interfaccia utente web di MAAS (sezione 4.3) dalla schermata "*Machines*" sarà possibile verificare lo stato di avanzamento della fase di deploy.

Se durante la configurazione del power type dei nodi (voce 2 nella sezione 4.4) è stata scelta la modalità *Manual*, la macchina del controller Juju deve essere avviata manualmente. L'intera fase di deploy richiede all'incirca una decina di minuti. A deploy terminato, la macchina del controller Juju si spegnerà e apparirà nell'elenco dei nodi sotto lo status *Deployed*.

Per visualizzare l'elenco aggiornato dei controller noti al client Juju, eseguire il comando `juju controllers`, il cui output d'esempio viene mostrato in figura 5.7.

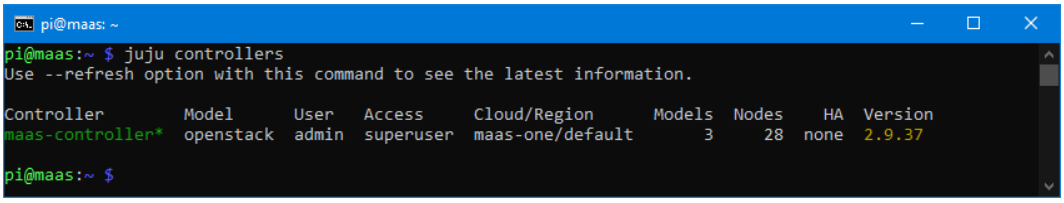


Figura 5.7: Elenco dei controller registrati nel client Juju.

L'ultimo passo è quello di creare il model del cloud, al quale successivamente verranno aggiunti i vari charm che daranno corpo all'istanza effettiva del cloud.

```
1 juju add-model --config default-series=jammy openstack
```

Listato 5.7: Creazione del model openstack per il cloud mass-one.

- Con l'opzione `--config` è possibile specificare varie configurazioni al modello. Con il valore `default-series=` viene indicata l'immagine di

default da utilizzare in fase di deploy dei nodi all'interno del cloud. In questo caso, l'immagine del sistema operativo è Ubuntu Jammy, corrispondente alla versione 22.04 LTS.

- Il valore *openstack* sta ad indicare il nome che assumerà il model.

Al termine, eseguendo il comando `juju status` è possibile verificare la creazione del cloud. Come mostrato in figura 5.8 l'output riassume il cloud appena creato.

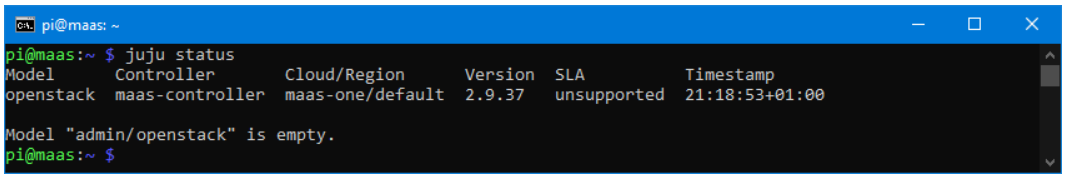


Figura 5.8: Status del cloud vuoto mass-one.

In appendice A.2 è possibile visionare l'elenco dei comandi precedentemente descritti in un unico listato.

5.2.3 Accesso alla dashboard

Una volta effettuata l'installazione di Juju, è possibile accedere alla dashboard grafica attraverso il browser web. Infatti, come per MAAS, è disponibile una web app, che permette di monitorare le varie applicazioni e i vari charm all'interno del model del cloud. Eseguendo quindi il comando `juju dashboard` verrà mostrato L'URL per accedere all'interfaccia web, lo username e la password da utilizzare. In questo caso, l'URL dell'interfaccia web è:

URL: `https://10.0.0.254:17070/dashboard`

Mentre le credenziali da immettere sono:

Username: `admin`

Password: `527460c1b6b309a6bfd565924bcaacf4`

L'uso della dashboard è del tutto superfluo, e in questo documento non verrà ulteriormente menzionata. Tuttavia, può risultare comoda nel momento in cui si voglia andare a configurare qualche charm nello specifico; questo perché vengono mostrati direttamente le possibili configurazioni e azioni che

questi possano avere, senza doverli cercare ne in documentazione ne a riga di comando.

In figura 5.9 viene mostrata la schermata d'accesso del cloud OpenStack con i charm già installati. Come è possibile vedere sono presenti sia i due model che vengono creati in automatico, ovvero il model per il *controller* e il model di *default*, che il model *openstack*, creato nel listato 5.7.

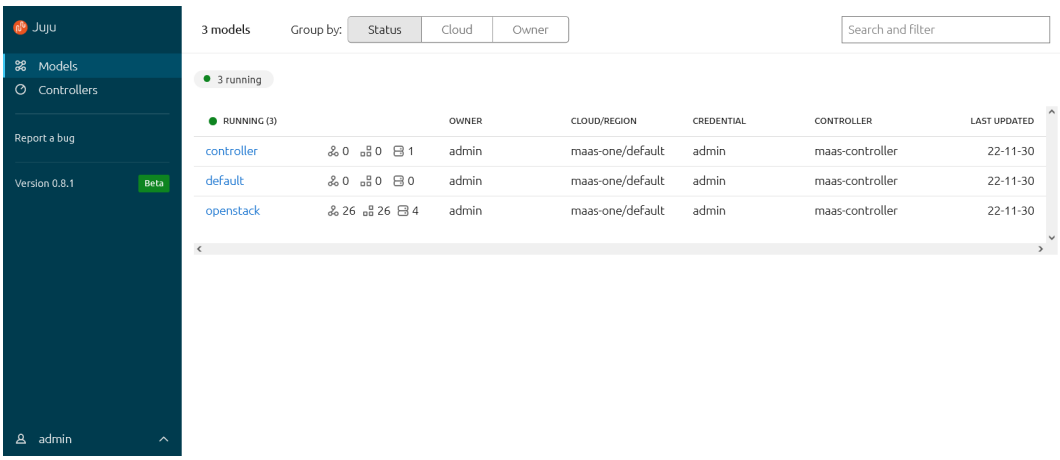


Figura 5.9: Esempio di schermata d'accesso della Juju dashboard.

Nei prossimi capitolo verrà trattata la piattaforma OpenStack, partendo da una sua descrizione e composizione, per poi passare alla completa installazione del cloud fino ad arrivare ad un suo semplice utilizzo.

Capitolo 6

OpenStack

OpenStack è una piattaforma di cloud computing completamente open source. Viene largamente utilizzata sia da aziende che vogliono costruirsi un proprio cloud privato che da cloud provider che offrono i loro servizi a terzi. Una caratteristica fondamentale di OpenStack è che non è un software monolitico, ma è composto da numerosissimi componenti che permettono un'ampia personalizzazione; è possibile infatti decidere quali componenti installare in base alle funzionalità che si desiderano e su quale macchina fisica installare ciascun componente in modo che si possano costruire macchine con caratteristiche diverse in base al componente che devono ospitare (e.g. la macchina che deve contenere il componente di block storage sarà sicuramente diversa rispetto ad una che deve contendere l'hypervisor). Oltre ai propri componenti, OpenStack utilizza anche software di terze parti, che verrà approfondito nel capitolo 6.2.

6.1 Componenti di OpenStack

Di seguito verranno descritti i componenti di OpenStack che sono stati installati durante questo progetto di tesi. Esistono numerosi altri componenti che implementano altre funzionalità ma, dato che non sono stati installati o approfonditi durante lo svolgimento del progetto, non verranno trattati.

6.1.1 Cinder

Cinder è il servizio di block storage di OpenStack e il suo compito è quello di fornire API di block storage sia agli utenti che agli altri componenti di OpenStack. Una sua caratteristica fondamentale è che virtualizza l'accesso ai dispositivi di storage in modo che i client possano utilizzare le API che espone senza sapere quale sia il backend utilizzato. Di conseguenza Cinder non implementa al suo interno la gestione fisica dello storage ma si collega a

sua volta ad altri servizi come ad esempio OpenStack Swift o Ceph (quello utilizzato in questo progetto).

6.1.2 Glance

Glance è il servizio di immagini di OpenStack, ovvero il servizio che si occupa di scoprire, registrare e fornire le immagini di macchine virtuali e i relativi metadati. Questo componente espone una serie di API che permettono di consultare i metadati di ciascuna immagine e di prelevare le immagini stesse. Glance ha la capacità di archiviare le immagini sia su storage locale che su block storage.

6.1.3 Horizon

Horizon è la dashboard predefinita di OpenStack. Fornisce un'applicazione web che si interfaccia con le API di tutti i componenti installati e permette di gestire il cloud tramite un'interfaccia grafica molto più semplice e intuitiva rispetto al tool a linea di comando.

6.1.4 Keystone

Keystone è l'identity service di OpenStack. Si occupa di: fornire le API per l'autenticazione dei client, rilevare i servizi e implementare l'autorizzazione multi-tenant. Supporta l'autenticazione tramite LDAP, OAuth, OpenID Connect, SAML e SQL.

6.1.5 Neutron

Neutron è il componente che gestisce tutta la parte di networking del cloud OpenStack. Nello specifico viene definito come un NaaS (Network as a Service) provider e permette di creare reti, sottoreti e router virtuali con lo scopo di far comunicare le macchine virtuali tra di loro con l'esterno. Gestisce anche l'assegnazione degli indirizzi IP pubblici (denominati *floating IP*) e include un servizio di firewall che permette di raggruppare le regole in *security groups* che poi possono essere assegnati alle macchine virtuali.

Neutron mette a disposizione una vasta scelta di plugin che permettono di scegliere quale backend utilizzare in base alle esigenze di ciascuna installazione. In questo progetto è stato utilizzato Open vSwitch perché è quello che viene consigliato di default.

6.1.6 Nova

Nova è il componente di OpenStack che permette di creare e gestire le macchine virtuali utilizzando l'hypervisor messo a disposizione dalla macchina host. Per poter funzionare ha bisogno di interfacciarsi con i seguenti componenti di OpenStack: Keystone, Glance, Neutron e Placement. Nel caso in cui si voglia uno storage persistente per le macchine virtuali è richiesto anche Cinder.

Nova supporta anche la gestione di server bare metal (tramite l'uso di Ironic) e ha un supporto limitato per i container, ma in questo caso specifico non abbiamo approfondito queste sue funzionalità.

6.1.7 Placement

Placement è il componente che si occupa di inventariare e tenere traccia dei *resource provider*. Un *resource provider* è un pool di risorse presenti nel cloud (e.g. nodi di calcolo, storage condivisi, pool di allocazione IP).

6.2 Componenti esterni a OpenStack

Come accennato in precedenza, OpenStack fa uso anche di componenti di terze parti; nello specifico, quelli utilizzati in questo progetto sono: MySQL, RabbitMQ, Vault, Open Virtual Network e Ceph. Alcuni di questi sono descritti più nello specifico nei prossimi paragrafi.

6.2.1 Vault

Vault è un sistema di gestione di *secrets* e crittografia basato sull'identità. Per *secret* si intende tutto ciò a cui si vuole controllare e restringere l'accesso, come ad esempio chiavi API, password, e certificati. Il compito di Vault è appunto fornire servizi di autenticazione a autorizzazione per accedere a queste risorse in maniera sicura.

In figura 6.1 è schematizzato il funzionamento di Vault. La prima cosa che ciascun client deve fare è autenticarsi e, durante questa procedura, Vault verifica l'identità attraverso il provider di autenticazione che è stato configurato. Una volta autenticato, il client può richiedere uno specifico *secret* e, una volta verificato che il suddetto client abbia le autorizzazioni necessarie, Vault restituisce il *secret* richiesto.

Come detto in precedenza, Vault verifica l'identità degli utenti tramite un provider di autenticazione. Sono disponibili numerosi provider che permettono di interfacciarsi con altrettanti servizi esterni (e.g. AWS, Azure, GitHub, Google Cloud, LDAP, Username e Password, ecc.), in questo modo si ha una

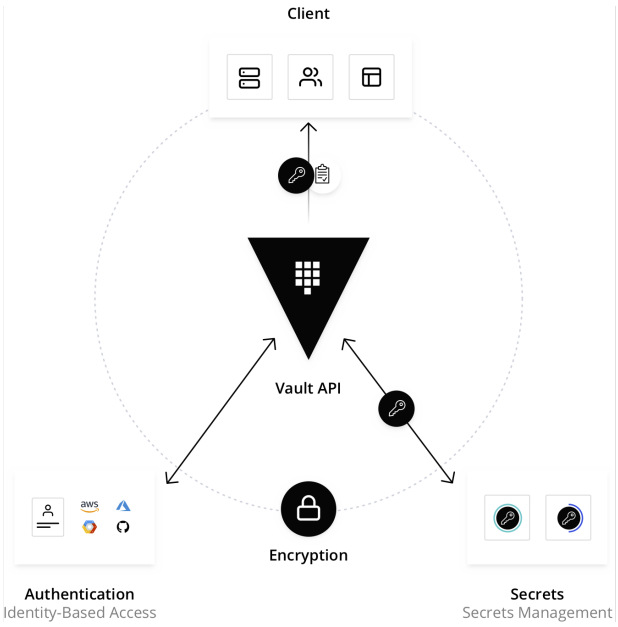


Figura 6.1: Schema di funzionamento di Vault [46].

vasta flessibilità. Il charm di Vault utilizza di default il provider basato sui Token [45].

6.2.2 Open vSwitch e Open Virtual Network

Open vSwitch. Open vSwitch (OvS) è un software open source che implementa uno switch virtuale. È progettato in modo da permettere un'ampia automazione della rete mantenendo al contempo il supporto a interfacce e protocolli di gestione standard. Supporta inoltre l'installazione distribuita su più macchine fisiche [34].

Open Virtual Network. Open Virtual Network (OVN) consiste in una serie di demoni che si appoggiano a Open vSwitch e implementano i layer di astrazione più alti; OVN permette infatti lavorare con router e switch logici ed è studiato per essere usato dai cloud management software (OpenStack nel nostro caso). Alcune delle sue funzionalità più rilevanti sono: router virtuali distribuiti, switch virtuali distribuiti, Access Control Lists (ACL), DHCP e DNS server [33].

6.2.3 Ceph

Ceph è una piattaforma open source di software-defined storage, ovvero una piattaforma software che gestisce l'archiviazione dei dati in modo indipendente dall'hardware sottostante. Una sua caratteristica fondamentale è che supporta i cosiddetti *storage cluster*, che consistono nell'aggregazione di più host fisici (potenzialmente anche migliaia) in un volume di storage unico che viene poi gestito da Ceph stesso in modo che i client non debbano preoccuparsi della posizione in cui archiviare o reperire i dati [1].
Ciascun *storage cluster* è composto da diversi componenti software:

- Ceph OSD Daemon (OSD): è il demone che interagisce con i volumi di storage su ciascuna macchina che compone il cluster e si occupa dell'immagazzinamento dei dati
- Ceph Monitor (MON): mantiene una copia della mappa dell'intero cluster
- Ceph Manager: funziona insieme a Ceph Monitor e il suo compito è quello di interfacciarsi con sistemi di monitoraggio esterni (non è fondamentale per il funzionamento del cluster)

Ceph mette a disposizione tre modalità di storage in un unico sistema: object storage, block storage e file storage.

Object storage. L'object storage è un formato di storage nel quale i dati vengono archiviati in unità separate chiamati oggetti. Ciascuno di questi oggetti possiede una chiave univoca che ne permette l'individuazione all'interno di un sistema distribuito.

Block storage. Il block storage è un formato di storage nel quale i dati vengono suddivisi in componenti di dimensione fissa detti blocchi, ciascuno dei quali dotato di un id univoco. Questo id, in modo analogo a quello che succede nell'obecjt storage, permette di individuare un singolo blocco all'interno di un sistema distribuito.

File storage. Il file storage è il formato di storage più conosciuto: i dati vengono archiviati in una struttura gerarchica di directory. Questo è il sistema che viene utilizzato ad esempio sui computer o sui NAS [35].

Capitolo 7

Installazione di OpenStack

Una volta creato il model con Juju (sezione 5.2.2), che rappresenta il contenitore software del cloud, arriva il momento di popolarlo con le applicazioni che costituiranno i servizi di OpenStack. Per fare ciò si utilizza sempre Juju, il quale in maniera più o meno automatizzata effettuerà il deployment dei charm sui vari nodi indicati.

Prima di procedere nell'installazione, è importate prestare attenzione ad alcuni aspetti, in particolare quelli riguardanti le versioni, in quanto essendo un progetto in rapido sviluppo le versioni utilizzate in questo documento diverranno con molta probabilità obsolete nel giro di una manciata di mesi.

7.1 Concetti preinstallazione

7.1.1 Tipologie di charm su OpenStack.

Su OpenStack esistono due tipologie di charm [29]: quelli che utilizzano i *channel* e quelli *legacy* che non lo utilizzano.

I charm di tipo **channel** sono quelli la cui release è dedicata ad una specifica combinazione del sistema operativo (detta anche *series*) e della versione di OpenStack; questa combinazione è chiamata *series-openstack* Questo implica che se un charm funziona per una recente combinazione di series-openstack, non è garantito che funzioni correttamente su una combinazione precedente. Inoltre, bisogna passare ad un channel diverso nel caso in cui si voglia eseguire l'aggiornamento ad una nuova versione di OpenStack. Per fare un esempio, durante il deploy di alcuni charm si è utilizzato l'argomento `--channel yoga/stable`; ciò implica la versione di OpenStack Yoga, che è funzionante sulla versione di Ubuntu Focal (20.04 LTS) o Jammy (22.04 LTS).

I charm di tipo **legacy** invece includono all'interno tutte le funzionalità delle revisioni precedenti e dunque funzionano anche con una combinazione

series-openstack antecedente. Tuttavia, il loro sviluppo si è fermato dalla versione 21.10 di OpenStack Charms (Ottobre 2021), di conseguenza l'ultimo supporto alla combinazione series-openstack è quella `focal-xena`.

In questo progetto sono stati utilizzati solo charms di tipo channel, e su ogni comando di deploy è stata specificata appunto la versione. In caso si volessero replicare i comandi su series-openstack più aggiornati, è necessario consultare le versioni adeguate dei channel nelle pagine di documentazione individuali dei singoli charm.

7.1.2 Versione di OpenStack.

All'inizio del progetto di questa tesi, era da poco uscita la versione di OpenStack chiamata *Yoga*, in data 2022-03-30, e pertanto tutta l'installazione è basata su questa versione. Durante la conclusione dei lavori progettuali, in data 2022-10-05 è uscita una nuova release, chiamata *Zed* , mentre nel prossimo futuro è prevista l'uscita di *Antelope* con data stimata 2023-03-22, e la successiva versione *Bobcat* con data stimata 2023-10-04. Nonostante il progetto OpenStack cresca velocemente, le release vengono mantenute a lungo, anche grazie al supporto della community; per esempio, la versione *Queens* uscita in data 2018-02-28 ha terminato solo da poco il suo ciclo di vita (2023-01-18), avendo quindi potuto godere di un supporto per quasi cinque anni.

7.1.3 Distribuzione dei charm all'interno delle macchine del cluster.

Come spiegato nella sezione 5.1, è compito di Juju di occuparsi del completo dispiegamento delle applicazioni nelle varie macchine collegate su MAAS. Le varie applicazioni saranno distribuite tra le quattro macchine a disposizione, in modo tale da avere una ripartizione equa in termini di risorse computazionali e di storage. In oltre, alcune di esse verranno replicate su più macchine in modo tale da creare ridondanza e suddivisione del carico di lavoro. Quindi, tutti gli aspetti inerenti al ciclo di vita delle applicazioni come l'installazione o la sincronizzazione tra le unit (le istanze effettive in esecuzione delle applicazioni), saranno gestiti completamente da Juju. Le uniche cose necessarie da definire sono quali charm installare e la loro versione, su quale macchine dispiegarli e risolvere le loro relazioni (integrazioni dalla versione 3.0 di Juju).

La decisione su quali charm usare e la loro suddivisione adottata in questo progetto è quella implementata da Canonical nei suoi esempi di Charms Deployment (sezione 3.2). Di seguito vengono elencate le applicazioni e i relativi charm suddivisi tra le quattro macchine (la quinta è stata successivamente utilizzata da Matteo Bambini).

- Ceph OSD (`ceph-osd`), OVN Central (`ovn-central`), MySQL InnoDB Cluster (`mysql-innodb-cluster`), Keystone (`keystone`), Ceph Mon (`ceph-mon`) e Ceph RADOS Gateway (`ceph-radogsw`).
- Ceph OSD (`ceph-osd`), Nova Compute (`nova-compute`), MySQL InnoDB Cluster (`mysql-innodb-cluster`), OVN Central (`ovn-central`), Cinder (`cinder`), Neutron API (`neutron-api`), Ceph Mon (`ceph-mon`).
- Ceph OSD (`ceph-osd`), Nova Compute (`nova-compute`), MySQL InnoDB Cluster (`mysql-innodb-cluster`), OVN Central (`ovn-central`), RabbitMQ Server (`rabbitmq-server`), OpenStack Dashboard (`openstack-dashboard`), Ceph Mon (`ceph-mon`).
- Ceph OSD (`ceph-osd`), Nova Compute (`nova-compute`), Nova Cloud Controller (`nova-cloud-controller`), Vault (`vault`), Glance (`glance`).

7.1.4 Monitoraggio del deploy.

Indipendentemente dalla tipologia di deployment scelta (un charm alla volta o in bundle), è possibile seguire ogni singolo passaggio che Juju effettua all'interno del cluster. Eseguendo il comando `juju status` quindi, verrà stampato lo stato attuale del cloud, elencando per ogni applicazione, unit e relazione varie informazioni a riguardo, come la macchina o il container sul quale verranno installati, i vari indirizzi IP e lo status ognuno di essi.

In particolare, è molto utile specialmente le prime volte per capire ed imparare i vari status delle applicazioni, nonché come primo punto di debug in caso di anomalie; questo perché verranno mostrati anche messaggi sulla condizione di ogni elemento mostrato, come è possibile vedere in figura 7.1.

Durante tutto il deployment del cloud, sarà normale notare messaggi come relazioni mancanti o unit bloccate, in quanto tali requisiti non possono essere risolti immediatamente fintanto che una successiva applicazione non viene installata. Un modo più conveniente per monitorare la progressione del deploy è quello di eseguire il comando `watch -n 5 -c juju status --color` in un terminale separato; questo comando eseguirà ogni `5` secondi il comando `juju status --color`, molto utile per avere un continuo aggiornamento sull'avanzamento dell'installazione.

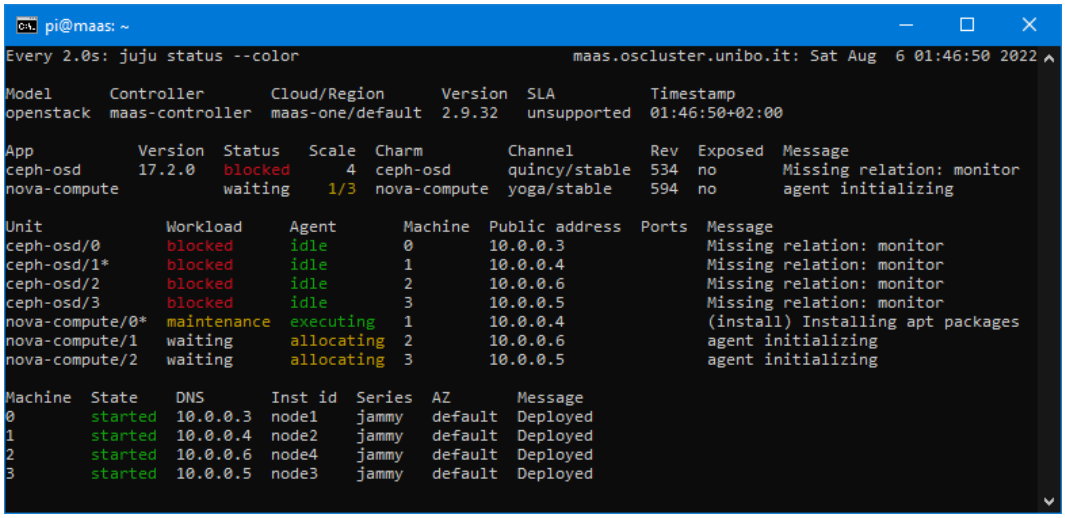


Figura 7.1: Esempio d'output di `juju status` durante le prime fasi di deploy dei charm.

7.2 Deploy semi-automatizzato di OpenStack

In questa sezione verrà effettuato il deploy del cloud OpenStack installando un charm alla volta. Tuttavia, Juju si occuperà dell'installazione ed effettiva configurazione di ogni singolo componente. L'intero processo di installazione richiede non poco tempo, e a seconda della velocità delle macchine e del tempo impiegato dall'operatore possono occorrere all'incirca 30/60 minuti per un'installazione veloce fino a 120/180 minuti per un'installazione più analitica. Per maggiori dettagli, si rimanda alla documentazione [32].

Come prima cosa, bisogna assicurarsi che i comandi verranno impartiti al Juju controller e al model corretto (sezione 5.2.2). Per farlo, basterà eseguire il comando mostrato nel listato 7.1.

```
1 juju switch maas-controller:openstack
```

Listato 7.1: Comando per selezionare il Juju controller e il corretto model.

Ora è possibile effettuare i deploy dei vari charm; tra l'esecuzione di un comando e l'altro non è necessario aspettare che Juju finisca nei vari procedimenti. Tuttavia, specialmente le prime volte, è consigliato eseguire un comando alla volta per apprendere al meglio le varie fasi di deploy.

Ceph OSD. Il primo charm che si andrà ad installare è quello inerente al gestore d'archiviazione dei dati, ceph-osd (sezione 6.2.3). Questo charm

bisogna configurarlo il maniera corretta, indicandogli dispositivi di storage da utilizzare. in questo progetto tutti i nodi montano gli stessi dispositivi di storage.

Per configurare i charm in maniera agile, si utilizzano i file di configurazione `YAML`; quindi è stato creato il file `ceph-osd.yaml` con le configurazioni mostrate nel listato 7.2.

```
ceph-osd:
  osd-devices: /dev/sda /dev/sdb
  source: distro
```

Listato 7.2: Configurazione di *ceph-osd* nel file *ceph-sod.yaml*.

Il charm ceph-osd verrà installato su tutti e 4 i nodi; essendo il primo deploy, sui nodi non è ancora presente il sistema operativo, quindi durante questa fase verrà installato anche quest'ultimo in maniera automatica, applicando le immagini scaricate attraverso MAAS. nel listato 7.3 viene mostrato il comando per il deploy di ceph-osd.

```
juju deploy -n 4 --series jammy --channel quincy/stable --config ceph-osd.
    yaml --constraints tags=compute ceph-osd
```

Listato 7.3: Deploy del charm *ceph-osd*.

- Con `-n` viene indicato su quanti nodi effettuare il deploy del charm, in questo caso su tutti e quattro i nodi.
- Con `--series` e con `--channel` viene indicato di fatto la versione del sistema operativo e del charm.
- Con `--config` è possibile specificare il file `YAML` per applicare delle configurazioni personalizzate.
- Con `--constraints` è possibile specificare in maniera accurata i requisiti hardware per le nuove macchine su cui effettuare il deploy. Questi requisiti possono essere ad esempio memoria ram, tag, space, etc. In questo caso specifico è stato indicato di effettuare il deploy su tutti i nodi aventi il tag *compute* (impostato durante le fasi di add dei nodi su MAAS nella voce 5 della sezione 4.4).

Nova Compute. Il prossimo charm da installare è nova-compute (sezione 6.1.6), usato per gestire le macchine virtuali. Anche in questo caso l'installazione del charm sarà personalizzata con l'inserimento di qualche impostazione salvata nel file `nova-compute.yaml`, contenente le configurazioni mostrate nel listato 7.4.

```
nova-compute:
  config-flags: default_ephemeral_format=ext4
  enable-live-migration: true
  enable-resize: true
  migration-auth-type: ssh
  virt-type: qemu
  openstack-origin: distro
```

Listato 7.4: Configurazione di *nova-compute* nel file *nova-compute.yaml*.

Quindi è possibile effettuare il deploy del charm nova-compute su tre macchine.

```
juju deploy -n 3 --to 1,2,3 --series jammy --channel yoga/stable --config
    nova-compute.yaml nova-compute
```

Listato 7.5: Deploy del charm *nova-compute*.

- Con `--to` è possibile indicare in maniera precisa su quali macchine andare ad effettuare i deploy del charm. Queste macchine devono essere già state aggiunte in Juju, e in questo caso è avvenuto con il primo comando di deploy. Il numero indicato rappresenta l'indice della macchina, ed è possibile prenderne nota attraverso il comando `juju status`.

MySQL InnoDB Cluster. Il charm che si occuperà della creazione e gestione del database per archiviare tutte le varie informazioni che gli altri charm utilizzeranno è mysql-innodb-cluster. Questo charm richiede sempre almeno tre unit, e saranno containerizzate tramite LXD nelle macchine 0, 1 e 2.

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --series jammy --channel 8.0/stable
    mysql-innodb-cluster
```

Listato 7.6: Deploy del charm *mysq-innodb-cluster*.

Vault. In questa fase verrà installato il charm vault (sezione 6.2.1) in una unica unit, il quale gestirà tutte quelle informazioni sensibili denominate come *secret*, come ad esempio i certificati TLS per la comunicazione crittografata tra le applicazioni cloud.

```
juju deploy --to lxd:3 --series jammy --channel 1.7/stable vault
```

Listato 7.7: Deploy del charm *vault*.

Questo charm deve essere messa in relazione con il database del cloud. Per farlo, è necessario installare una unit specifica del subordinate *mysql-router*, che si comporterà da tramite tra vault e mysql-innodb-cluster. Una volta installato, è possibile aggiungere le relazioni come mostrato in listato 7.8.

```
juju deploy --channel 8.0/stable mysql-router vault-mysql-router
juju add-relation vault-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation vault-mysql-router:shared-db vault:shared-db
```

Listato 7.8: Deploy del subordinate *mysql-router* per le relazioni con vault.

Configurazione di vault Finita l'installazione di vault, è necessario inizializzarlo e "aprirlo". Innanzitutto bisogna installare il client Vault attraverso snap.

```
sudo snap install vault
```

Listato 7.9: Installazione del client *Vault*.

Successivamente, si estrapola l'indirizzo IP del container sul quale è stato installato; l'indirizzo IP è possibile visionarlo anche attraverso `juju status`. L'indirizzo IP serve per inizializzare la variabile `VAULT_ADDR` con l'URI del charm vault, necessaria per le fasi successive.

```
IP_VAULT=$(juju status --format=yaml vault | grep public-address | awk '{
    print $2}' | head -1)
export VAULT_ADDR="http://${IP_VAULT}:8200"
```

Listato 7.10: Crezione della variabile d'ambiente `VAULT_ADDR`.

A questo punto è possibile inizializzare Vault, indicandogli di creare cinque chiavi e di necessitarne tre per la sua apertura. Queste chiavi poi sono state esportate sul file `vault-keys` per una miglior comprensione ed utilizzo. Queste chiavi sono salvate in chiaro, ed è consigliato conservarle in un luogo sicuro.

```
vault operator init -key-shares=5 -key-threshold=3 > vault-keys
```

Listato 7.11: Generazione delle chiavi d'apertua del vault.

Ora è possibile aprire Vault, utilizzando tre delle cinque chiavi generate nel listato 7.11.

```
vault operator unseal <key1>
vault operator unseal <key2>
vault operator unseal <key3>
```

Listato 7.12: Unseal del client Vault.

- Al posto di `<key1>`, `<key2>` e `<key3>`, bisogna inserire tre chiavi a piacere tra le cinque a disposizione.

Una volta aperto il client Vault, bisogna autorizzare il charm vault a poter interagire con esso e creare e gestire i secret. Per farlo, viene creato un token root dalla durata di 10 minuti.

```
export VAULT_TOKEN=<Initial Root Token>
vault token create -ttl=10m
```

Listato 7.13: Generazione del token temporaneo.

- Il `<Initial Root Token>` è possibile trovarlo nell'output (nel file se è stato salvato) del listato 7.11.

Infine, è possibile autorizzare il charm vault con il token appena generato nel listato 7.13. Con il comando `run-action`, è possibile far eseguire ai charm un'azione tra quelle che concedono, in base all'implementazione del charm stesso.

```
juju run-action --wait vault/leader authorize-charm token=<token>
```

Listato 7.14: Autorizzazione al charm *vault*.

- Al posto di `<token>` bisogna inserire il token temporaneo generato nel listato 7.13.

A charm autorizzato, è possibile creare un certificato autofirmato se non se ne possiede uno rilasciato da una CA.

```
juju run-action --wait vault/leader generate-root-ca > vault-ca.crt
```

Listato 7.15: Creazione del certificato autofirmato.

Come ultimo passaggio, verrà collegato il certificato di vault, creato nel listato 7.15, al DB del cloud attraverso l'aggiunta della relativa relazione.

```
juju add-relation mysql-innodb-cluster:certificates vault:certificates
```

Listato 7.16: Aggiunta della relazione per collegare il certificato a DB.

Neutron. Per implementare la rete con Neutron (sezione 6.1.5), verranno installati quattro charms:

- ovn-central per il controllo delle OVN (sezione 6.2.2), installato su tre unit.
- neutron-api fornisce il servizio di API di Neutron, installato in una unica unit.
- neutron-api-plugin-ovn subordinate di neutron-api.
- ovn-chassis subordinate di ovn-central.

Nel file `neutron.yaml` verranno inserite le configurazioni della rete che Neutron utilizzerà.

```
ovn-chassis:
  bridge-interface-mappings: br-ex:enp3s0
  ovn-bridge-mappings: physnet1:br-ex
neutron-api:
  neutron-security-groups: true
  flat-network-providers: physnet1
  openstack-origin: distro
ovn-central:
  source: distro
```

Listato 7.17: Configurazione di *Neutron* nel file *neutron.yaml*.

- `bridge-interface-mappings` indica la mappatura del *OvS bridge* creato nel voce 6 nella sezione 4.4 ed è composto dal *bridge name* seguito dai due punti e dal nome dell'interfaccia di rete. In questo caso, il *bridge name* dato è stato `br`, mentre le interfacce di rete erano nominate come `enp3s0`.
- `physnet1` è il nome che viene associato al provider di rete di tipo flat.

Quindi, si procede con il deploy dei quattro charm.

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --series jammy --channel 22.03/
    stable --config neutron.yaml ovn-central
juju deploy --to lxd:1 --series jammy --channel yoga/stable --config
    neutron.yaml neutron-api

juju deploy --channel yoga/stable neutron-api-plugin-ovn
juju deploy --channel 22.03/stable --config neutron.yaml ovn-chassis
```

Listato 7.18: Deploy dei quattro charm che comporranno *Neutron*.

Dopo il deploy, è possibile aggiungere le relazioni che i charm necessitano. Anche in questo caso viene installata una unit per il collegamento di Neutron con il DB (come nel listato 7.8)

```
juju add-relation neutron-api-plugin-ovn:neutron-plugin neutron-api:
    neutron-plugin-api-subordinate
juju add-relation neutron-api-plugin-ovn:ovsdb-cms ovn-central:ovsdb-cms
juju add-relation ovn-chassis:ovsdb ovn-central:ovsdb
juju add-relation ovn-chassis:nova-compute nova-compute:neutron-plugin
juju add-relation neutron-api:certificates vault:certificates
juju add-relation neutron-api-plugin-ovn:certificates vault:certificates
juju add-relation ovn-central:certificates vault:certificates
juju add-relation ovn-chassis:certificates vault:certificates

juju deploy --channel 8.0/stable mysql-router neutron-api-mysql-router
juju add-relation neutron-api-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation neutron-api-mysql-router:shared-db neutron-api:shared-db
```

Listato 7.19: Aggiunta delle varie relazioni per *Neutron*.

Keystone. Il charm keystone (sezione 6.1.4) è il componente che si occuperà di fornire le API per l'autenticazione dei client. Verrà installato in un'unica unit.

```
juju deploy --to lxd:0 --series jammy --channel yoga/stable keystone

juju deploy --channel 8.0/stable mysql-router keystone-mysql-router
juju add-relation keystone-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation keystone-mysql-router:shared-db keystone:shared-db

juju add-relation keystone:identity-service neutron-api:identity-service
juju add-relation keystone:certificates vault:certificates
```

Listato 7.20: Deploy del charm *keystone*.

RabbitMQ. RabbitMQ è il servizio che implementa il broker per il protocollo di messaggistica AMQP e il suo charm rabbitmq-server viene installato su un'unica unit.

```
juju deploy --to lxd:2 --series jammy --channel 3.9/stable rabbitmq-server
juju add-relation rabbitmq-server:amqp neutron-api:amqp
juju add-relation rabbitmq-server:amqp nova-compute:amqp
```

Listato 7.21: Deploy del charm *rabbitmq-server*.

Nova cloud controller. Questa applicazione implementa per conto di Nova tre servizi: uno inerente alle API, uno per il coordinamento e supporto per le query del DB con Nova ed infine uno per la selezione dei nodi durante la creazione di istanze delle macchine virtuali. Anche in questo caso è necessario aggiungere delle configurazioni attraverso il file dedicato `ncc.yaml`.

```
nova-cloud-controller:
  network-manager: Neutron
  openstack-origin: distro
```

Listato 7.22: Configurazione di *Nova Cloud Controller* nel file *ncc.yaml*.

Dopodiché è possibile installare il charm nova-cloud-controller su un'unica istanza, con annesso subordinate per il collegamento con il database.

```
juju deploy --to lxd:3 --series jammy --channel yoga/stable --config ncc.
    yaml nova-cloud-controller

juju deploy --channel 8.0/stable mysql-router ncc-mysql-router
juju add-relation ncc-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation ncc-mysql-router:shared-db nova-cloud-controller:
    shared-db

juju add-relation nova-cloud-controller:identity-service keystone:
    identity-service
juju add-relation nova-cloud-controller:amqp rabbitmq-server:amqp
juju add-relation nova-cloud-controller:neutron-api neutron-api:neutron-api
juju add-relation nova-cloud-controller:cloud-compute nova-compute:
    cloud-compute
juju add-relation nova-cloud-controller:certificates vault:certificates
```

Listato 7.23: Deploy del charm *nova-cloud-controller*.

Placement. Placement (sezione 6.1.7), attraverso il charm placement, si occuperà di inventariare le risorse del cloud e viene installato su un'unica unit. Anch'esso utilizza un subordinate per l'interfacciamento con il database.

```
juju deploy --to lxd:3 --series jammy --channel yoga/stable placement

juju deploy --channel 8.0/stable mysql-router placement-mysql-router
juju add-relation placement-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation placement-mysql-router:shared-db placement:shared-db

juju add-relation placement:identity-service keystone:identity-service
juju add-relation placement:placement nova-cloud-controller:placement
juju add-relation placement:certificates vault:certificates
```

Listato 7.24: Deploy del charm *placement*.

OpenStack dashboard. La dashboard e la relativa interfaccia grafica via web dell'intero cloud OpenStack è implementata dall'applicazione Horizon (sezione 6.1.3), installato attraverso il charm openstack-dashboard in un'unica unit e il subordinate per la connessione con il database.

```
juju deploy --to lxd:2 --series jammy --channel yoga/stable
    openstack-dashboard

juju deploy --channel 8.0/stable mysql-router dashboard-mysql-router
juju add-relation dashboard-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation dashboard-mysql-router:shared-db openstack-dashboard:
    shared-db

juju add-relation openstack-dashboard:identity-service keystone:
    identity-service
juju add-relation openstack-dashboard:certificates vault:certificates
```

Listato 7.25: Deploy del charm *openstack-dashboard*.

Glance. Glance (sezione 6.1.2) è il servizio di OpenStack avente il compito della gestione delle immagini per le macchine virtuali. Viene implementato attraverso il charm glance su un'unica istanza con il relativo subordinate per la comunicazione con il DB.

```
juju deploy --to lxd:3 --series jammy --channel yoga/stable glance

juju deploy --channel 8.0/stable mysql-router glance-mysql-router
juju add-relation glance-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation glance-mysql-router:shared-db glance:shared-db

juju add-relation glance:image-service nova-cloud-controller:image-service
juju add-relation glance:image-service nova-compute:image-service
juju add-relation glance:identity-service keystone:identity-service
juju add-relation glance:certificates vault:certificates
```

Listato 7.26: Deploy del charm *glance*.

Ceph (monitor). Il charm ceph-mon implementa il monitor per Ceph, ovvero quel componente che mantiene una copia della mappa dell'intero cluster. Anche questo viene configurato attraverso un file esterno, `ceph-mon.yaml`

```
ceph-mon:
  expected-osd-count: 4
  monitor-count: 3
```

```
  source: distro
```

Listato 7.27: Configurazione di *Ceph Mon* nel file *ceph-mon.yaml*.

Infine, viene installato su tre nodi.

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --series jammy --channel quincy/
    stable --config ceph-mon.yaml ceph-mon

juju add-relation ceph-mon:osd ceph-osd:mon
juju add-relation ceph-mon:client nova-compute:ceph
juju add-relation ceph-mon:client glance:ceph
```

Listato 7.28: Deploy del charm *ceph-mon*.

Cinder. Cinder (sezione 6.1.1) è l'ultimo componente che verrà configurato attraverso un file, `cinder.yaml`, per l'implementazione del servizio di block storage attraverso il charm cinder.

```
cinder:
  block-device: None
  glance-api-version: 2
  openstack-origin: distro
```

Listato 7.29: Configurazione di *Cinder* nel file *cinder.yaml*.

Quindi si installa il charm cinder, il subordinate per la comunicazione con il database e l'aggiunta delle relative relazioni.

```
juju deploy --to lxd:1 --series jammy --channel yoga/stable --config cinder
    .yaml cinder

juju deploy --channel 8.0/stable mysql-router cinder-mysql-router
juju add-relation cinder-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation cinder-mysql-router:shared-db cinder:shared-db

juju add-relation cinder:cinder-volume-service nova-cloud-controller:
    cinder-volume-service
juju add-relation cinder:identity-service keystone:identity-service
juju add-relation cinder:amqp rabbitmq-server:amqp
juju add-relation cinder:image-service glance:image-service
juju add-relation cinder:certificates vault:certificates
```

Listato 7.30: Deploy del charm *cinder*.

Infine, cinder necessita del subordinate cinder-ceph per poter interfacciarsi con Ceph; Infatti, sfrutta quest'ultimo come effettivo storage, limitandosi a fornire una virtualizzazione di essi. Nel listato 7.29 è possibile notare che attraverso l'opzione `block-device: None` non gli sono stati indicati i block device.

```
juju deploy --channel yoga/stable cinder-ceph
juju add-relation cinder-ceph:storage-backend cinder:storage-backend
juju add-relation cinder-ceph:ceph ceph-mon:client
juju add-relation cinder-ceph:ceph-access nova-compute:ceph-access
```

Listato 7.31: Deploy del charm *cinder-ceph*.

Ceph RADOS Gateway. L'ultimo componente che viene installato è Ceph RADOS Gateway, che attraveso il charm ceph-radosgw offre un gateway HTTP compatibile con S3 e Swift. Quest'ultimo charm viene installato su un'unica istanza.

```
juju deploy --to lxd:0 --series jammy --channel quincy/stable ceph-radosgw
juju add-relation ceph-radosgw:mon ceph-mon:radosgw
```

Listato 7.32: Deploy del charm *ceph-radosgw*.

Risultati finali. Dopo aver effettuato l'ultimo deploy, è necessario attendere che Juju porti a termine le varie operazioni. Una volta stabilizzato, l'output del comando `juju status` dovrebbe essere simile a come mostrato nelle figure da 7.2 a 7.4. Tutte le applicazioni e le unit avranno lo status *active* con i relativi agent in *idle*, e come message mostreranno che sono in *ready*.

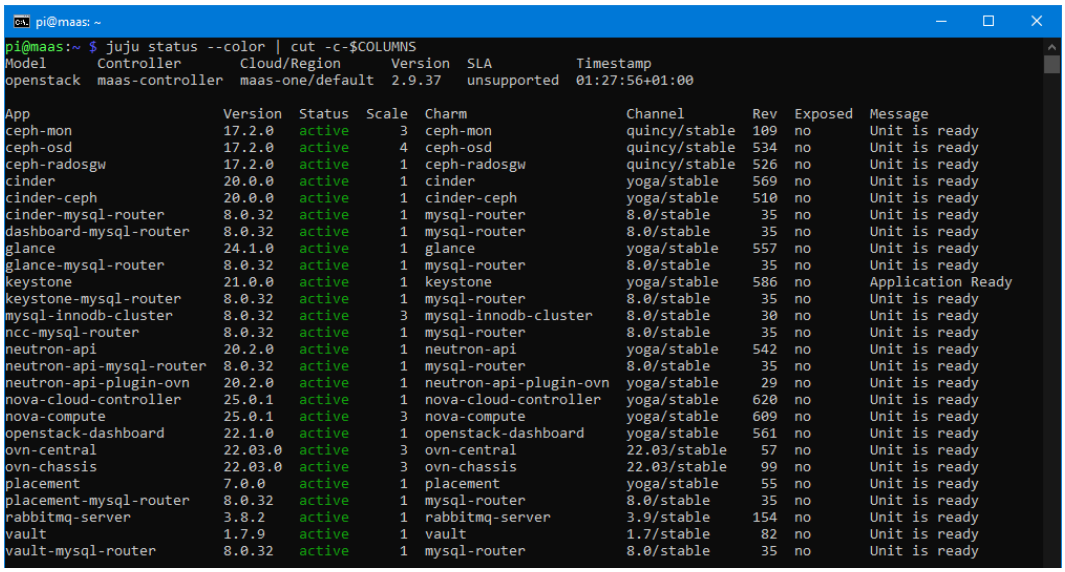
7.2.1 Accesso alla dashboard Horizon.

Ora è finalmente possibile utilizzare il cloud OpenStack. Come prima cosa, bisogna ricavare i dati per poter accedere alla dashboard del cloud. Per ricavare l'indirizzo IP della web app è possibile sia cercarlo manualmente sotto la unit `openstack-dashboard/0*` nell'output del comando `juju status` (come mostrato in figura 7.3) sia estrapolarlo utilizzando la combinazione di comandi mostrati nel listato 7.33.

```
juju status --format=yaml openstack-dashboard | grep public-address | awk
    '{print $2}' | head -1
```

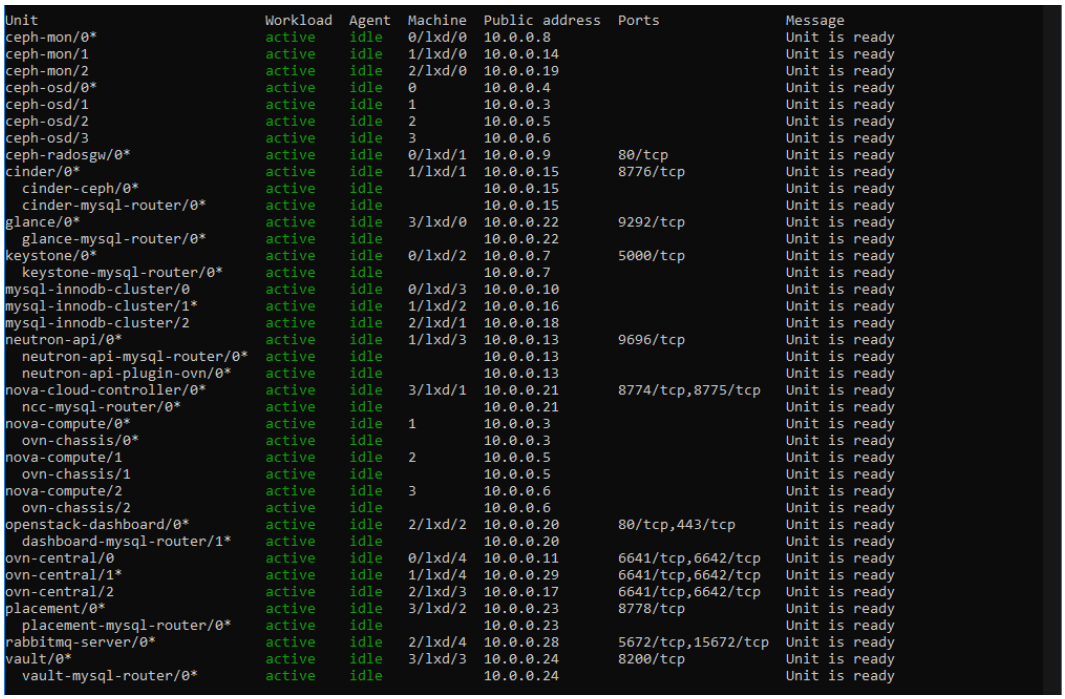
Listato 7.33: Comandi per estrapolare l'IP della dashboard Horizon.

In questo progetto di testi, l'indirizzo IP che è stato riservato in maniera automatica al charm openstack-dashboard è `10.0.0.20`.



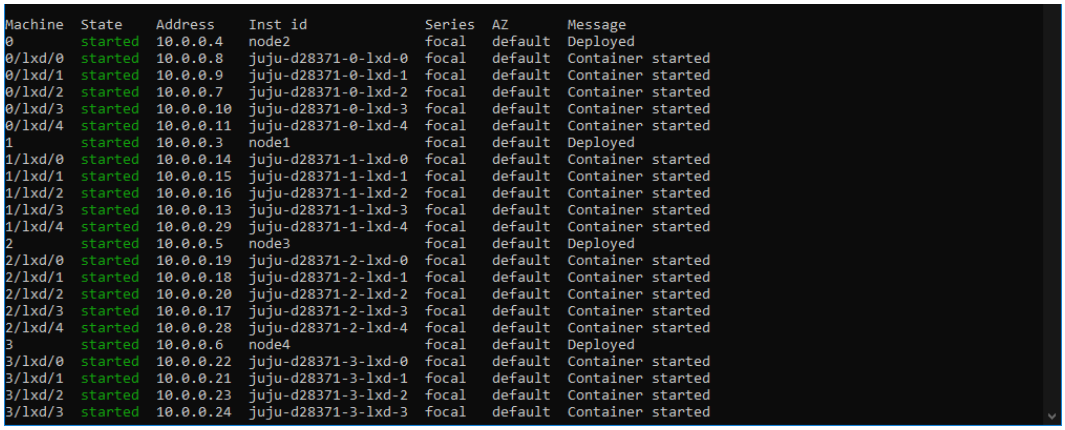
```
pi@maas: ~
pi@maas:~ $ juju status --color | cut -c-$COLUMNS
Model      Controller       Cloud/Region       Version  SLA          Timestamp
openstack  maas-controller  maas-one/default  2.9.37   unsupported  01:27:56+01:00

App                       Version  Status  Scale  Charm                     Channel        Rev  Exposed  Message
ceph-mon                  17.2.0   active      3  ceph-mon                  quincy/stable  109  no       Unit is ready
ceph-osd                  17.2.0   active      4  ceph-osd                  quincy/stable  534  no       Unit is ready
ceph-radosgw              17.2.0   active      1  ceph-radosgw              quincy/stable  526  no       Unit is ready
cinder                    20.0.0   active      1  cinder                    yoga/stable    569  no       Unit is ready
cinder-ceph               20.0.0   active      1  cinder-ceph               yoga/stable    510  no       Unit is ready
cinder-mysql-router       8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
dashboard-mysql-router    8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
glance                    24.1.0   active      1  glance                    yoga/stable    557  no       Unit is ready
glance-mysql-router       8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
keystone                  21.0.0   active      1  keystone                  yoga/stable    586  no       Application Ready
keystone-mysql-router     8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
mysql-innodb-cluster      8.0.32   active      3  mysql-innodb-cluster      8.0/stable      30  no       Unit is ready
ncc-mysql-router          8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
neutron-api               20.2.0   active      1  neutron-api               yoga/stable    542  no       Unit is ready
neutron-api-mysql-router  8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
neutron-api-plugin-ovn    20.2.0   active      1  neutron-api-plugin-ovn    yoga/stable     29  no       Unit is ready
nova-cloud-controller     25.0.1   active      1  nova-cloud-controller     yoga/stable    620  no       Unit is ready
nova-compute              25.0.1   active      3  nova-compute              yoga/stable    609  no       Unit is ready
openstack-dashboard       22.1.0   active      1  openstack-dashboard       yoga/stable    561  no       Unit is ready
ovn-central               22.03.0  active      3  ovn-central               22.03/stable    57  no       Unit is ready
ovn-chassis               22.03.0  active      3  ovn-chassis               22.03/stable    99  no       Unit is ready
placement                 7.0.0    active      1  placement                 yoga/stable     55  no       Unit is ready
placement-mysql-router    8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
rabbitmq-server           3.8.2    active      1  rabbitmq-server           3.9/stable     154  no       Unit is ready
vault                     1.7.9    active      1  vault                     1.7/stable      82  no       Unit is ready
vault-mysql-router        8.0.32   active      1  mysql-router              8.0/stable      35  no       Unit is ready
```

Figura 7.2: Status finale del cloud OpenStack post deployment con i charm: elenco delle App (i message sono tagliati per motivi di spazio).



```
Unit                            Workload  Agent  Machine  Public address  Ports              Message
ceph-mon/0*                     active    idle   0/lxd/0  10.0.0.8                          Unit is ready
ceph-mon/1                      active    idle   1/lxd/0  10.0.0.14                         Unit is ready
ceph-mon/2                      active    idle   2/lxd/0  10.0.0.19                         Unit is ready
ceph-osd/0*                     active    idle   0        10.0.0.4                          Unit is ready
ceph-osd/1                      active    idle   1        10.0.0.3                          Unit is ready
ceph-osd/2                      active    idle   2        10.0.0.5                          Unit is ready
ceph-osd/3                      active    idle   3        10.0.0.6                          Unit is ready
ceph-radosgw/0*                 active    idle   0/lxd/1  10.0.0.9        80/tcp            Unit is ready
cinder/0*                       active    idle   1/lxd/1  10.0.0.15       8776/tcp          Unit is ready
  cinder-ceph/0*                active    idle            10.0.0.15                         Unit is ready
  cinder-mysql-router/0*        active    idle            10.0.0.15                         Unit is ready
glance/0*                       active    idle   3/lxd/0  10.0.0.22       9292/tcp          Unit is ready
  glance-mysql-router/0*        active    idle            10.0.0.22                         Unit is ready
keystone/0*                     active    idle   0/lxd/2  10.0.0.7        5000/tcp          Unit is ready
  keystone-mysql-router/0*      active    idle            10.0.0.7                          Unit is ready
mysql-innodb-cluster/0          active    idle   0/lxd/3  10.0.0.10                         Unit is ready
mysql-innodb-cluster/1*         active    idle   1/lxd/2  10.0.0.16                         Unit is ready
mysql-innodb-cluster/2          active    idle   2/lxd/1  10.0.0.18                         Unit is ready
neutron-api/0*                  active    idle   1/lxd/3  10.0.0.13       9696/tcp          Unit is ready
  neutron-api-mysql-router/0*   active    idle            10.0.0.13                         Unit is ready
  neutron-api-plugin-ovn/0*     active    idle            10.0.0.13                         Unit is ready
nova-cloud-controller/0*        active    idle   3/lxd/1  10.0.0.21       8774/tcp,8775/tcp  Unit is ready
  ncc-mysql-router/0*           active    idle            10.0.0.21                         Unit is ready
nova-compute/0*                 active    idle   1        10.0.0.3                          Unit is ready
  ovn-chassis/0*                active    idle            10.0.0.3                          Unit is ready
nova-compute/1                  active    idle   2        10.0.0.5                          Unit is ready
  ovn-chassis/1                 active    idle            10.0.0.5                          Unit is ready
nova-compute/2                  active    idle   3        10.0.0.6                          Unit is ready
  ovn-chassis/2                 active    idle            10.0.0.6                          Unit is ready
openstack-dashboard/0*          active    idle   2/lxd/2  10.0.0.20       80/tcp,443/tcp    Unit is ready
  dashboard-mysql-router/1*     active    idle            10.0.0.20                         Unit is ready
ovn-central/0                   active    idle   0/lxd/4  10.0.0.11       6641/tcp,6642/tcp  Unit is ready
ovn-central/1*                  active    idle   1/lxd/4  10.0.0.29       6641/tcp,6642/tcp  Unit is ready
ovn-central/2                   active    idle   2/lxd/3  10.0.0.17       6641/tcp,6642/tcp  Unit is ready
placement/0*                    active    idle   3/lxd/2  10.0.0.23       8778/tcp          Unit is ready
  placement-mysql-router/0*     active    idle            10.0.0.23                         Unit is ready
rabbitmq-server/0*              active    idle   2/lxd/4  10.0.0.28       5672/tcp,15672/tcp  Unit is ready
vault/0*                        active    idle   3/lxd/3  10.0.0.24       8200/tcp          Unit is ready
  vault-mysql-router/0*         active    idle            10.0.0.24                         Unit is ready
```

Figura 7.3: Status finale del cloud OpenStack post deployment con i charm: elenco delle Unit (i message sono tagliati per motivi di spazio).



```
Machine  State    Address     Inst id              Series  AZ       Message
0        started  10.0.0.4    node2                focal   default  Deployed
0/lxd/0  started  10.0.0.8    juju-d28371-0-lxd-0  focal   default  Container started
0/lxd/1  started  10.0.0.9    juju-d28371-0-lxd-1  focal   default  Container started
0/lxd/2  started  10.0.0.7    juju-d28371-0-lxd-2  focal   default  Container started
0/lxd/3  started  10.0.0.10   juju-d28371-0-lxd-3  focal   default  Container started
0/lxd/4  started  10.0.0.11   juju-d28371-0-lxd-4  focal   default  Container started
1        started  10.0.0.3    node1                focal   default  Deployed
1/lxd/0  started  10.0.0.14   juju-d28371-1-lxd-0  focal   default  Container started
1/lxd/1  started  10.0.0.15   juju-d28371-1-lxd-1  focal   default  Container started
1/lxd/2  started  10.0.0.16   juju-d28371-1-lxd-2  focal   default  Container started
1/lxd/3  started  10.0.0.13   juju-d28371-1-lxd-3  focal   default  Container started
1/lxd/4  started  10.0.0.29   juju-d28371-1-lxd-4  focal   default  Container started
2        started  10.0.0.5    node3                focal   default  Deployed
2/lxd/0  started  10.0.0.19   juju-d28371-2-lxd-0  focal   default  Container started
2/lxd/1  started  10.0.0.18   juju-d28371-2-lxd-1  focal   default  Container started
2/lxd/2  started  10.0.0.20   juju-d28371-2-lxd-2  focal   default  Container started
2/lxd/3  started  10.0.0.17   juju-d28371-2-lxd-3  focal   default  Container started
2/lxd/4  started  10.0.0.28   juju-d28371-2-lxd-4  focal   default  Container started
3        started  10.0.0.6    node4                focal   default  Deployed
3/lxd/0  started  10.0.0.22   juju-d28371-3-lxd-0  focal   default  Container started
3/lxd/1  started  10.0.0.21   juju-d28371-3-lxd-1  focal   default  Container started
3/lxd/2  started  10.0.0.23   juju-d28371-3-lxd-2  focal   default  Container started
3/lxd/3  started  10.0.0.24   juju-d28371-3-lxd-3  focal   default  Container started
```

Figura 7.4: Status finale del cloud OpenStack post deployment con i charm elenco delle Machine.

Dopodiché, è necessario conoscere le credenziali dell'amministratore di sistema. La password può essere richiesta a Keystone come mostrato nel seguente listato listato 7.34.

```
1 juju run --unit keystone/leader leader-get admin_passwd
```

Listato 7.34: Richiesta a Keystone della password d'amministratore.

L'URL della web app a cui collegarsi quindi sarà `http://<IP>/horizon/` (o se si volesse usare il protocollo https `https://<IP>/horizon/`), in questo caso:

URL: `http://10.0.0.20/horizon/`

Una volta collegatosi da qualsiasi browser web all'indirizzo, apparirà la schermata di log-in come mostrato in figura 7.5.

Le credenziali dell'amministratore da immettere sono:

Username: `admin`

Password: `ejco3C6Wgxj2je9B`

Domain: `admin_domain`

A log-in effettuato, apparirà la schermata iniziale, da cui poter utilizzare il cloud (figura 7.6).

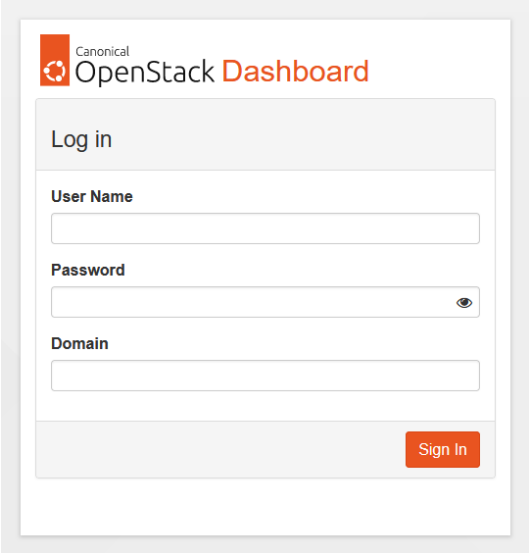


Figura 7.5: Schermata di login della dashboard di OpenStack.

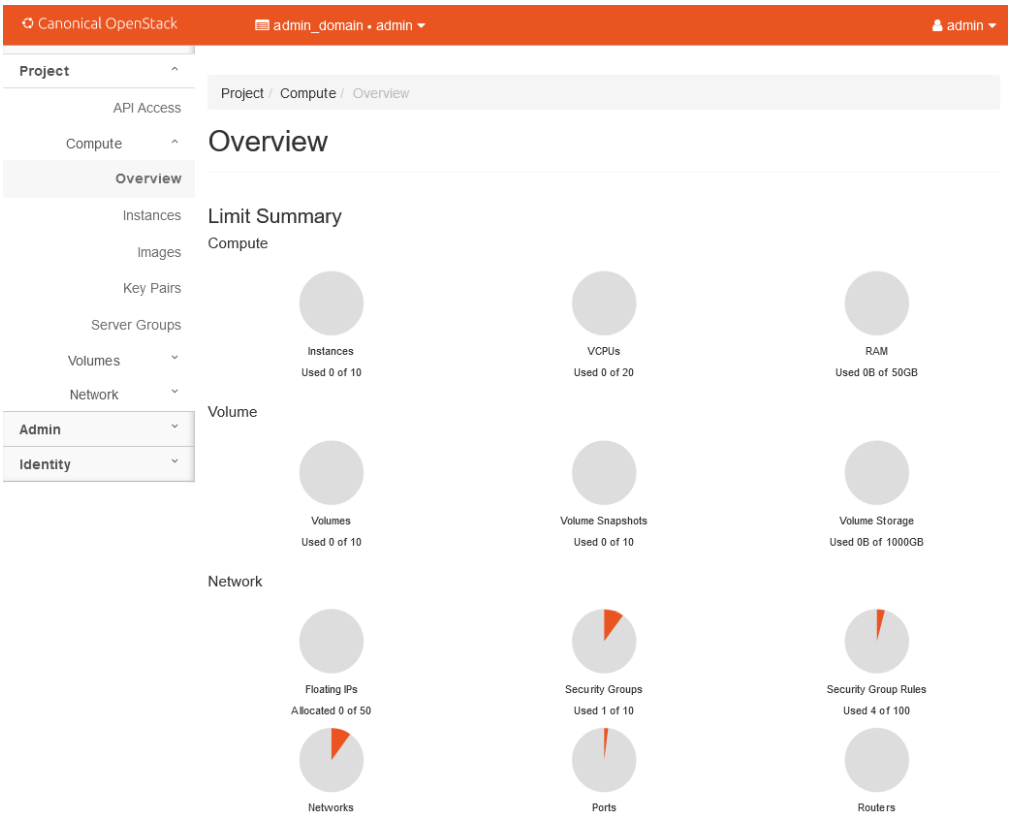


Figura 7.6: Schermata post log-in della dashboard di OpenStack con qualche risorsa in uso.

7.3 Configurazione iniziale

In questo capitolo verrà trattata la configurazione iniziale di un cloud OpenStack. Tutta la configurazione verrà fatta seguendo la guida di installazione [30] e di conseguenza utilizzando il tool a linea di comando, però è possibile anche eseguire tutte le operazione descritte in questo capitolo tramite l'interfaccia web. È possibile ottenere le credenziali dell'utente `admin` eseguendo il comando `juju run --unit keystone/leader leader-get admin_passwd` dall'host su cui è stato precedentemente installato e configurato il client Juju. Per maggiori dettagli su come ottenere l'accesso, consultare la sezione 7.2.1.

Per comprendere meglio le configurazioni utilizzate in questa sezione, sia lato amministratore che utente, si consiglia di visionare la capitolo 8 dove viene fornita una spiegazione dei concetti di base che permette di comprendere meglio quali sono le funzionalità di OpenStack e in che modo possono essere sfruttate.

Installazione del client e accesso al cloud. Per eseguire la configurazione tramite linea di comando è necessario installare il client OpenStack tramite il package manager (`snap` in questo caso) e clonare il repository openstack-bundles[1]. Per fare questo si devono eseguire i comando mostrati nel listato 7.35

```
1 sudo snap install openstackclients
2 git clone https://github.com/openstack-charmers/openstack-bundles ~/
    openstack-bundles
```

Listato 7.35: Installazione del client openstack e clonazione del repo.

In questo modo il repo verrà clonato nella home dell'utente e successivamente sarà sufficiente eseguire il comando nel listato 7.36 per impostare le variabili d'ambiente in modo da avere accesso al cloud con privilegi di amministrazione.

```
1 source ~/openstack-bundles/stable/openstack-base/openrc
```

Listato 7.36: Configurazione dell'accesso al cloud.

7.3.1 Configurazioni da parte dell'amministratore

Immagini e Flavor. Il primo passo è quello di creare la prima immagine e il primo flavor. Eseguendo i comandi nel listato 7.37 viene create la cartella `cloud-images` nella home e al suo interno viene scaricata un'immagine recente di Ubuntu Jammy direttamente dall'archivio delle cloud images di Canonical; in questo modo sarà poi possibile ricaricare l'immagine sul nuovo cloud.

[1] OpenStack Bundle repository: `https://github.com/openstack-charmers/openstack-bundles`, ultimo accesso 28 Febbraio 2023

```
mkdir ~/cloud-images
curl http://cloud-images.ubuntu.com/jammy/current/
    jammy-server-cloudimg-amd64.img --output ~/cloud-images/jammy-amd64.img
```

Listato 7.37: Download dell'immagine di Ubuntu Jammy server.

Una volta terminato il download è possibile creare una nuova immagine partendo da quella scaricata eseguendo il comando nel listato 7.38.

```
openstack image create --public --container-format bare --disk-format qcow2
     --file ~/cloud-images/jammy-amd64.img jammy-amd64
```

Listato 7.38: Creazione di un'immagine partendo dal file scaricato.

- `--public`: indica che l'immagine è pubblica, ovvero è visibile a tutti gli utenti del cloud
- `--container-format`: indica il formato del container dell'immagine, ovvero per quale tecnologia di virtualizzazione o containerizzazione è stata create l'immagine (per esempio `bare` per macchine virtuali, `docker` per container docker, ecc.)
- `--disk-format`: indica il formato dell'immagine
- `--file`: permette di specificare un file locale dal quale leggere l'immagine
- Come ultimo argomento deve essere inserito il nome dell'immagine

Successivamente è possibile creare un flavor eseguendo il comando nel listato 7.39 (per maggiori informazioni sui flavor si veda la sezione 8.4.1).

```
openstack flavor create --ram 2048 --disk 20 --ephemeral 20 m1.small
```

Listato 7.39: Creazione di un flavor.

- `--vcpus`: numero di CPU virtuali (di default impostato a 1)
- `--ram`: quantità di memoria RAM espressa in MB
- `--ephemeral`: dimensione dello storage storage temporaneo espresso in GB (utilizzato nel caso in cui non venga creato un volume per l'istanza)
- Come ultimo argomento deve essere inserito il nome del flavor

Rete pubblica. Dopo aver creato la prima immagine e il primo flavor è necessario configurare la rete pubblica, ovvero la rete che permette alle istanze di tutto il cloud di comunicare con l'esterno. Per fare questo è necessario prima di tutto creare la rete eseguendo i comandi nel listato 7.40.

```
openstack network create --external --share --provider-network-type flat --
    provider-physical-network physnet1 ext_net
```

Listato 7.40: Creazione di una rete pubblica.

- `--external`: indica che è una rete pubblica
- `--share`: indica che la rete è condivisa tra tutti gli utenti del cloud
- `---provider-network-type`: permette di specificare il meccanismo fisico con il quale la rete virtuale è implementata; in questo caso `flat` significa che utilizza una scheda di rete fisica configurata come bridge
- `--provider-physical-network`: nome della rete fisica sulla quale la rete virtuale si appoggia
- Come ultimo argomento deve essere inserito il nome della nuova rete.

Dopo aver creato la rete è possibile creare anche la subnet utilizzando il comando mostrato nel listato 7.41.

```
openstack subnet create --network ext_net --no-dhcp --gateway 10.0.0.1 --
    subnet-range 10.0.0.0/24 --allocation-pool start=10.0.0.40,end
    =10.0.0.99 ext_subnet
```

Listato 7.41: Creazione di una subnet pubblica.

- `--network`: nome della rete dentro la quale creare la subnet
- `--no-dhcp`: se specificato il DHCP viene disabilitato per la nuova subnet
- `--gateway`: indirizzo IP del gateway
- `--subnet-range`: permette di specificare il range di indirizzi IP appartenenti alla subnet (in formato CIDR)
- `--allocation-pool`: permette di specificare il pool di indirizzi IP assegnabili dinamicamente (nel formato `start_ip,end_ip`). Questo pool può risiedere anche nella sottorete di MAAS. Quindi bisogna riservare questi indirizzi in modo tale che non li utilizzi. Per farlo, bisogna connettersi alla dashboard di MAAS, preme in alto su *Subnets* e selezionare la VLAN. Poi, bisogna premere sul pulsante *Reserve Range* (NON *dynamic*) ed inserire i due indirizzi IP del pool.

- Come di consueto l'ultimo argomento deve essere il nome della nuova subnet.

Dominio, Progetto e Utente. A questo punto può considerarsi conclusa la configurazione delle risorse condivise ed è possibile iniziare a configurare le risorse per gli utilizzatori del cloud, ovvero domini, progetti e utenti.

Il primo passo è quello di creare un dominio, perché senza quest'ultimo non è possibile aggiungere progetti o utenti al di fuori del dominio di amministrazione. Successivamente, si possono creare il progetto, specificando il domain in cui inserirlo, e l'utente, specificando il progetto e il dominio di cui appartiene. Questi tre comandi sono esplicitati nel listato 7.42.

```
1 openstack domain create domain1
2 openstack project create --domain domain1 project1
3 openstack user create --domain domain1 --project project1 --password-prompt
      user1
```

Listato 7.42: Creazione di dominio, progetto e utente.

- Con `--password-prompt` la password verrà richiesta a terminale una volta eseguito il comando.

Dopo aver creato l'utente è necessario assegnargli un ruolo in base ai permessi che gli si vogliono consentire. In questo caso verrà assegnato il ruolo `Member` al nuovo utente utilizzando il comando mostrato nel listato 7.43; in questo modo che potrà effettuare tutte quelle operazioni che non richiedono permessi di amministrazione.

```
1 openstack role add --user 8b16e5335976418e99bf0b798e83e413 --project
     project1 Member
```

Listato 7.43: Assegnazione del ruolo al nuovo utente.

A questo punto è possibile iniziare ad utilizzare il cloud con il nuovo utente appena creato.

7.3.2 Configurazioni da parte dell'utente

Setup ambiente utente. Prima di iniziare ad utilizzare il nuovo utente, è necessario configurare le variabili d'ambiente del terminale per poter eseguire i comandi `openstack` con questi privilegi. Come prima cosa, bisogna prelevare l'URL di *Keystone* per l'autenticazione; è possibile ricavarlo tramite il comando `juju status` o più semplicemente, si può ricavare dall'ambiente

amministratore utilizzato in precedenza dalla variabile `OS_AUTH_URL`. Quindi basterà eseguire sul terminale `echo $OS_AUTH_URL` per avere in output l'URL.

A questo punto, si può creare un file con le configurazioni d'esempio mostrate nel listato 7.44. In questo caso il file verrà chiamato *project1-rc* (senza estensione).

```
export OS_AUTH_URL=https://10.0.0.7:5000/v3
export OS_USER_DOMAIN_NAME=domain1
export OS_USERNAME=user1
export OS_PROJECT_DOMAIN_NAME=domain1
export OS_PROJECT_NAME=project1
export OS_PASSWORD=ubuntu
```

Listato 7.44: File per l'ambiente utente *user1*.

- I valori delle variabili sono i medesimi creati per l'utente nel listato 7.42.
- Per la password, si è ipotizzato che è stata inserita *ubuntu*.

Una volta salvato il file, è possibile eseguire il comando `source project1-rc` per impostare le variabili d'ambiente per avere eccesso al cloud con l'utente scelto. Ora è possibile utilizzare il cloud con l'utente attraverso la riga di comando.

Rete Privata. Come prima cosa, verrà configurata la rete privata che verrà poi utilizzata dalle varie macchine virtuali. Si eseguano i comandi nel listato 7.45 per poterla creare.

```
openstack network create --internal user1_net

openstack subnet create --network user1_net --dns-nameserver 10.0.0.2 --
    subnet-range 192.168.0.0/24 --allocation-pool start=192.168.0.10,end
    =192.168.0.199 user1_subnet
```

Listato 7.45: Creazione di una rete privata con la relativa subnet.

- `--internal`: indica che si tratta di una rete privata
- `--dns-nameserver`: indica l'indirizzo del DNS utilizzato
- Come ultimo argomento dei due comandi deve essere inserito il nome della nuova rete e della nuova subnet

Creata la rete, è necessario creare un router virtuale per poter collegare la rete pubblica con la rete appena creata. I comandi mostrati in mostrano la creazione del router.

```
openstack router create user1_router
openstack router add subnet user1_router user1_subnet
openstack router set user1_router --external-gateway ext_net
```

Listato 7.46: Creazione di una router virtuale.

- `--external-gateway`: indica il nome della rete esterna creata nel listato 7.40
- Come ultimo argomento del primo comando deve essere inserito il nome del nuovo router

Import chiavi SSH e Security Group. Per poter accedere alle istanze delle macchine virtuali, è necessario importare una coppia di chiavi SSH. Nel listato 7.47 è mostrato come viene importata la chiave pubblica SSH all'interno di OpenStack.

```
openstack keypair create --public-key ~/.ssh/id_rsa.pub user1
```

Listato 7.47: Importazione delle chiavi SSH.

- `--public-key`: indica il path della chiave pubblica
- Come ultimo argomento deve essere inserito il nome da associare alla *Key Pairs*

Per consentire il passaggio del traffico SSH nelle future macchine virtuali, è necessario creare un *Security Group* come mostrato nel listato 7.48, avente le varie regole per consentire il traffico su determinati protocolli e porte.

```
openstack security group create --description 'Allow SSH' Allow_SSH
openstack security group rule create --proto tcp --dst-port 22 Allow_SSH
```

Listato 7.48: Creazione del Security Group per il traffico SSH.

- `--description`: è la descrizione del gruppo di regole da creare
- `--proto`: indica il protocollo della regola
- `--dst-port`: indica la porta della regola
- Come ultimo argomento dei due comandi deve essere inserito il nome da associare al *Security Group* e alla relativa regola

Creazione di una istanza. Finalmente è possibile creare una macchina virtuale. Per farlo è sufficiente eseguire il comando mostrato nel listato 7.49

```
openstack server create --image jammy-amd64 --flavor m1.small --key-name
    user1 --network user1_net --security-group Allow_SSH jammy-1
```

Listato 7.49: Creazione di una istanza della macchina virtuale.

- `--image`: viene indicata l'immagine da utilizzare
- `--flavor`: viene indicato il flavor da utilizzare
- `--key-name`: viene indicato il nome della key pair da utilizzare
- `--network`: indica il nome della rete privata su cui la macchina virtuale si collegherà
- `--security-group`: indica il gruppo di regole per consentire il traffico dati
- Come ultimo argomento deve essere inserito il nome da associare alla istanza della macchina virtuale

Al termine ella creazione della macchina virtuale, è possibile associargli un indirizzo pubblico per poter poi collegarsi in SSH. Anche questa volta, nel listato 7.50 verrà mostrato come poter eseguire questa fase.

```
FLOATING_IP=$(openstack floating ip create -f value -c floating_ip_address
    ext_net)
openstack server add floating ip jammy-1 $FLOATING_IP
```

Listato 7.50: Creazione e associazione di un floating ip alla vm.

Una volta eseguito, si avrà l'indirizzo IP pubblico all'interno della variabile `FLOATING_IP`, e pertanto è possibile utilizzarlo per connettersi in SSH alla macchina virtuale.

Si conclude qui l'installazione e l'uso base del cloud OpenStack. Nel capitolo 8 è possibile approfondire gli aspetti trattati in questa sezione, mentre nella capitolo 9 è stato utilizzato lo strumento Terraform per poter creare le varie risorse di OpenStack tramite del codice di configurazione.

Capitolo 8

Utilizzo di OpenStack

All'interno di questo capitolo verranno spiegati tutti i concetti base che servono per un primo utilizzo di OpenStack. Verranno anche date le istruzioni per eseguire le operazioni principali, come ad esempio creare una rete o una macchina virtuale, in modo che anche chi non ha mai utilizzato questa piattaforma possa orientarsi e farsi un'idea generale sulle funzionalità che offre.

8.1 Identity

8.1.1 Domains

Un dominio di OpenStack è un contenitore ad alto livello per progetti, gruppi e utenti. Ciascun dominio definito all'interno di un cloud OpenStack è completamente indipendente dagli altri; questo permette di avere entità (per esempio utenti, gruppi o progetti) con gli stessi nomi definiti in domini diversi. Inoltre gli utenti di domini diversi possono essere rappresentati da diversi backend di autenticazione e avere attributi diversi ma devono obbligatoriamente essere mappati sugli stessi set di ruoli e privilegi che sono stati utilizzati per definire le policy di sicurezza in modo da poter avere accesso ai servizi del cloud.

Per creare un dominio è necessario accedere con un utente amministratore. Una volta eseguito il login si deve aprire il menu a tendina denominato *Identity* e selezionare la voce *Domains*. A questo punto verrà visualizzata la pagina con tutti i domini presenti all'interno del cloud e per crearne uno nuovo sarà sufficiente cliccare sul pulsante *Create Domain* e inserire i dati richiesti. Se si vogliono aggiungere entità al dominio appena creato (per esempio utenti, progetti, ecc.) sarà necessario cliccare sul pulsante *Set Domain Context* accanto al nome del dominio; in questo modo il context della sessione verrà cambiato in modo da attivare il dominio selezionato e poter modificare le sua entità.

8.1.2 Groups

I gruppi di OpenStack sono dei contenitori ai quali appartengono degli utenti. Possono essere utilizzati per garantire l'accesso a progetti o semplicemente per raggruppare gli utenti.

Per creare un gruppo è necessario accedere all'interfaccia con un utente amministratore o con privilegi di amministrazione all'interno del dominio. Una volta eseguito l'accesso si deve aprire il menu a tendina *Identity* e selezionare la voce *Groups*. Cliccando sul pulsante *Create Group* verrà visualizzato un form tramite il quale sarà possibile creare il nuovo gruppo.

8.1.3 Roles

I ruoli sono delle entità associate agli utenti e, dato che OpenStack usa un approccio RBAS (Role-Based Access Control), servono per determinare quali azioni ciascun utente può compiere.

8.1.4 Projects

Un progetto è un contenitore di entità (utenti, istanze, ecc.) che a sua volta fa parte di un dominio. A differenza di un dominio il progetto possiede alcune entità univoche e altre che possono essere in comune con altri progetti; per esempio un'istanza può far parte solamente di un progetto, mentre un utente può far parte di più progetti.

Per creare un progetto è necessario eseguire l'accesso all'interfaccia con un utente amministratore o con privilegi di amministrazione all'interno del dominio. Una volta eseguito l'accesso si deve aprire il menu a tendina *Identity* e selezionare la voce *Projects*. Cliccando poi sul pulsante *Create Project* verrà aperto un form composto da tre schede:

- **Project Information**: permette di specificare le informazioni generiche del progetto (nome, descrizione, ecc.)
- **Project Members**: permette di selezionare gli utenti che hanno accesso al progetto
- **Project Groups**: permette di selezionare i gruppi che fanno parte del progetto

Se si utilizza Horizon (l'interfaccia Web di OpenStack) è possibile accedere solamente ad un progetto per volta, ovvero visualizzare le entità del progetto selezionato. Per cambiare la selezione del progetto è necessario cliccare sul nome del progetto attualmente selezionato nella barra di navigazione in cima

alla pagine (come mostrato in figura 8.1) e successivamente selezionare dal menu a tendina il progetto che si vuole utilizzare.

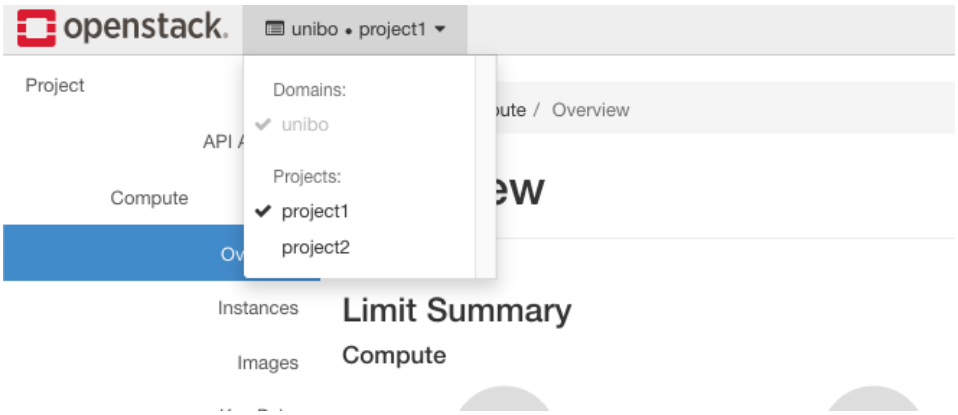


Figura 8.1: Selezione del progetto tramite l'interfaccia Web.

8.1.5 Users

Gli user, come ci su può aspettare, sono gli utenti che utilizzano il cloud OpenStack. Ciascun utente può essere membro di uno o più progetti, di uno o più gruppi e può avere uno o più ruoli. È possibile autenticare gli utenti in modi differenti utilizzando provider di autenticazione esterni (per esempio LDAP, SSO, ecc.); nel caso di questo progetto è stato utilizzato il provider di default di OpenStack.

Per creare un utente è necessario eseguire l'accesso all'interfaccia con un utente amministratore o con privilegi di amministrazione all'interno del dominio. Una volta eseguito l'accesso si deve aprire il menu a tendina *Identity* e selezionare la voce *Users*. In questo modo verrà aperta la pagine contenente tutti gli utenti del dominio che si sta visualizzando e, cliccando sul pulsante *Crete User* in alto a destra, sarà possibile crearne uno nuovo. È possibile inserire numerose informazioni riguardanti ciascun utente ma quelle fondamentali sono Username, Password e Primary Project; se non viene scelto nessun progetto verrà negato l'accesso al nuovo utente nonostante sia abilitato.

8.2 Network

8.2.1 Networks e Subnets

Una network in OpenStack è una rete virtuale che sta alla base della comunicazione tra macchine virtuali, container o altri dispositivi virtuali che comu-

nicano tramite rete. Ciascuna network può contenere al suo interno una o più subnet che hanno la stessa funzionalità delle VLAN in una rete fisica: servono per suddividere logicamente la rete in più sezioni separate, ciascuna delle quali possiede un proprio range di indirizzi IP ed è indipendente dalle altre.

Per creare una rete è necessario aprire il menu a tendina *Project* > *Network* e selezionare la voce *Networks*; questo aprirà la pagina dove vengono visualizzate tutte le reti. A questo punto, cliccando sul pulsante *Create Network*, verrà visualizzato il form per creare la nuova rete. Spuntato la casella *Enable Admin State* la rete verrà automaticamente abilitata dopo la creazione e, spuntando *Create Subnet*, viene data anche la possibilità di creare direttamente una subnet. Tramite il parametro *MTU* è possibile anche configurare la dimensione massima che possono avere i pacchetti che vengono inviati sulla rete.

Con un utente con privilegi di amministrazione è possibile anche creare una rete pubblica, ovvero una rete che possiede una subnet con indirizzi IP pubblici a cui tutti gli utenti possono collegarsi, in modo che gli host della rete possano essere raggiunti direttamente dall'esterno.

Per creare una subnet invece è necessario andare alla pagine dove vengono visualizzate tutte le reti e cliccare sul nome di quella che si vuole utilizzare; questo aprirà la pagina di dettaglio della rete. Una volte fatto questo si deve passare alla scheda denominata *Subnets* e cliccare sul pulsante *Subnets*; a questo punto verrà visualizzato un form suddiviso in due schede dove andranno inserite tutte le informazioni riguardanti la nuova subnet che si vuole creare. I campi richiesti sono i seguenti:

- **Subnet Name**: nome della subnet (arbitrario)
- **Network Address**: spazio degli indirizzi IP (in formato CIDR) che appartengono alla subnet (per esempio `10.0.0.0/24`)
- **IP Version**: versione del protocollo IP
- **Gateway IP**: indirizzo IP del gateway; se non si desidera aggiungere un gateway alla subnet è possibile spuntare la casella `Disable Gateway` per disabilitarlo
- **Enable DHCP**: se spuntato, il server DHCP viene abilitato per la nuova subnet
- **Allocation Polls**: permette di indicare i range di indirizzi IP che possono essere assegnati dinamicamente; si deve inserire un pool per ciascuna riga nel formato `start_ip,end_ip` (per es. `10.0.0.128,10.0.0.253`)
- **DNS Name Servers**: permette di specificare i server DNS per la subnet (uno per riga)

- **Host Routes**: permette di specificare delle regole di routing statiche che verranno applicate agli host; ciascuna entry deve essere nel formato `destination_cidr,nexthop` (per es. `192.168.1.0/24,10.0.0.1`)

8.2.2 Routers

Un router virtuale OpenStack ha esattamente le stesse funzioni di un router fisico in una rete fisica: permette agli host della rete di raggiungere subnet diverse dalla propria. Il router supporta sia il routing statico sia diversi protocolli di routing dinamico, come ad esempio OSPF e BGP. Ciascun router può avere una o più interfacce virtuali che possono essere collegate a diverse subnet appartenenti alla stessa rete o a reti diverse. I router possono avere un numero indefinito di interffacce di rete virtuali e quindi possono essere collegati a tutte le subnet desiderate; inoltre, se una delle interfacce viene collegata ad una rete pubblica, permettono a ciascuno degli host presenti sulle subnet private di accedere a internet utilizzando la tecnica del NAT.

Per creare un router è necessario aprire il menu a tendine *Project* > *Network* e selezionare la voce *Routers*; in questa pagina verranno mostrati tutti i router appartenenti al progetto corrente. Cliccando sul pulsante *Create Router* verrà mostrato il form che permette di creare il nuovo router con i seguenti campi che, oltre a campi simili a quelli visti in precedenza, contiene il campo *External Network* che permette di specificare una rete esterna a cui collegare il router. Questo campo non è obbligatorio ma se si desidera far accedere gli host delle subnet collegate al router a internet senza un indirizzo IP pubblico è necessario impostarlo. Dopo aver creato il router è possibile aggiungere le interfacce virtuali che lo collegano alle subnet; per fare questo si deve cliccare sul nome del router in modo da aprire la pagina con tutte le informazioni, aprire la scheda *Iterfaces* e cliccare sul pulsante *Add Interface*. A questo punto comparirà un form che permette di selezionare quale subnet collegare al router e opzionalmente l'indirizzo IP da assegnargli all'interno della subnet selezionata.

Nel caso in cui il router sia stato collegato ad una rete esterna, nella pagina di dettaglio con tutte le informazioni riguardanti il router è possibile visualizzare il suo indirizzo IP pubblico nella sezione *External Gateway*. Questo permette di verificare se il router funziona semplicemente facendo un ping all'indirizzo mostrato.

8.2.3 Security Groups

I *Security Groups* sono l'equivalente dei firewall virtuali che controllano il traffico in entrata e in uscita per ciascun host. Permettono di specificare

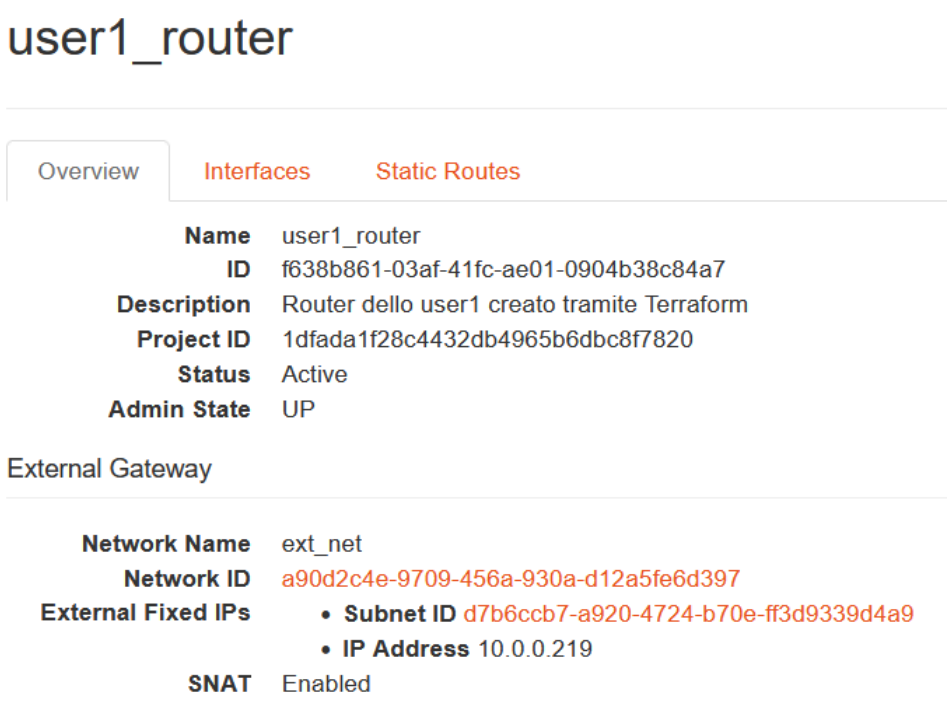


Figura 8.2: Pagina con le informazioni di un router.

regole secondo diversi criteri come ad esempio gli indirizzi IP della sorgente e del destinatario o le porte. Ciascun *Security Group* può avere un numero indefinito di regole e può essere assegnato a più di un host; ogni host può avere a sua volta più di un *Security Group*.

Per creare un *Security Group* da interfaccia grafica è necessario aprire il menu a tendina *Project* > *Network* e cliccare sulla voce *Security Groups*; in questo modo verrà aperta la pagina che mostra tutti i gruppi presenti all'interno del progetto corrente. Cliccando sul pulsante *Create Security Group* in alto a destra verrà aperto un form che permetterà di specificare nome e descrizione del nuovo gruppo. Per modificare le regole del *Security Group* appena creato si deve cliccare il pulsante *Manage Rules* sulla stessa riga del nome; a questo punto verrà caricata una pagina con la lista delle regole già presenti; per crearne una nuova è necessario cliccare sul pulsante *Add Rule* che farà comparire un form con i seguenti campi:

- **Rule**: permette di selezionare il protocollo che si vuole filtrare; sono già presenti numerosi protocolli standard (per esempio HTTP, HTTPS, SSH, ecc.) ma viene anche data la possibilità di definire regole personalizzate sia su protocollo TCP che su UDP

- **Direction**: permette di specificare se la regola deve essere applicata al traffico in entrata (Ingress) o in uscita (Egress)
- **Open Port**: permette di decidere se la regole deve essere applicata ad una sola porta, ad un range di porte o a tutte le porte per il protocollo selezionato
- **Port**: viene visualizzato solo se la regola viene applicata ad una sola porta e permette di specificare la suddetta porta
- **From Port** e **To Port**: vengono visualizzati solo se la regola viene applicata ad un range di porte e permettono di specificare rispettivamente la porta iniziale e quella finale
- **Remote**: permette di specificare che la regola deve filtrare per *CIDR* o *Security group*; nel primo caso verranno fatti passare i pacchetti che provengono da un indirizzo appartenente alla subnet definita mentre nel secondo caso verranno fatti passare solo i pacchetti che provengono da un host che possiede un *Security Group* definito
- **CIDR**: viene visualizzato solo se il campo *Remote* è impostato su *CIDR*; permette di impostare il range di indirizzi IP da filtrare in formato *CIDR*
- **Security Group** e **Ether Type**: vengono visualizzati solo se il campo *Remote* è impostato su *Security Group*; permettono di specificare rispettivamente il *Security Group* a cui è consentito l'accesso e la versione del protocollo IP

8.2.4 Floating IPs

I *Floating IPs* sono indirizzi IP pubblici che possono essere assegnati ad un host connesso ad una rete privata e che gli permettono di essere raggiunto anche dall'esterno. Ciascun IP può essere assegnato e rilasciato in qualsiasi momento e addirittura può essere assegnato a host differenti in momenti diversi.

Per ottenere un *Floating IP* da associare ad un host è necessario aprire il menu a tendina *Project* > *Network* e selezionare la voce *Floating IPs*. In questa pagina verranno visualizzati tutti gli indirizzi IP ottenuti per il progetto corrente. Per allocarne uno nuovo è necessario cliccare sul pulsante *Allocate IP To Project*; a questo punto comparirà il form di allocazione che permette di scegliere, tra le altre cose, il pool di indirizzi pubblici da cui prelevare il nuovo IP.

Per assegnare un *Floating IP* ad un'istanza è necessario accedere alla pagina con la lista delle istanza aprendo il menu a tendina *Project* > *Compute* e

selezionare la voce *Instances*. A questo punto si deve individuare l'istanza nella tabella e poi cliccare il pulsante *Associate Floating IP* nella colonna *Action*; in questo modo verrà visualizzato il form che permetterà di associare all'istanza un Floating IP esistente oppure di allocarne uno nuovo e poi assegnarlo.

8.3 Storage

8.3.1 Volumes

Un volume è un disco virtuale che può essere collegato ad un'istanza e su cui vengono salvati i dati. Ha esattamente la stessa funzione di un HDD o SSD all'interno di un computer fisico. I volumi possono essere creati al momento della creazione dell'istanza oppure tramite l'apposita pagina all'interno dell'interfaccia web.

Per crearne uno è necessario aprire il menu a tendina *Project* > *Volumes* e selezionare la voce *Volumes*; qui verranno visualizzati tutti i volumi appartenenti al progetto corrente. Cliccando sul pulsante *Create Volume* verrà mostrato il form che permette di creare un nuovo volume specificandone tutti i parametri compresa la dimensione massima.

8.3.2 Snapshots

Gli snapshot sono come delle fotografie dei volumi. Permettono di eseguire un'istantanea del volume in un qualsiasi momento e di salvarla in modo che in caso di problemi, come ad esempio un cancellazione involontaria di dati, si possa ripristinare il sistema esattamente com'era. Vengono spesso usati per creare backup o per clonare un sistema con lo scopo di eseguire test o di modificarlo.

8.4 Compute

8.4.1 Flavors

I *Flavors* sono configurazioni predefinite per le risorse da assegnare ad una istanza; al loro interno contengono il numero di CPU, la quantità di RAM e lo spazio di storage predefinito da assegnare ad un'istanza quando viene creata. I *Flavors* hanno il compito di creare dei modelli di istanze tra cui i client possono scegliere in base alle risorse di cui hanno bisogno e, all'interno dei cloud provider pubblici come ad esempio AWS, hanno anche lo scopo di definire il costo di una determinata istanza.

Per creare un *Flavor* è necessario eseguire il login con un utente con privilegi di amministrazione; una volta fatto questo si deve aprire il menu a tendina *Admin* > *Compute* e selezionare la voce *Flavors*. A questo punto si aprirà la pagina contente tutti i *Flavors* definiti all'interno del cloud; per crearne uno nuovo si deve cliccare il pulsante *Create Flavor* e compilare il form composto dai seguenti campi:

- **Name**: nome del flavor
- **ID**: id del flavor; inserendo la stringa *auto* viene generato in automatico
- **VCPUs**: numero di CPU virtuali a disposizione dell'istanza
- **RAM (MB)**: quantità in MB di RAM a disposizione dell'istanza
- **Root Disk (GB)**: dimensione in GB del volume principale
- **Ephemeral Disk (GB)**: dimensione in GB dello storage temporaneo non persistente dell'istanza
- **Swap Disk (MB)**: dimensione in MB della memoria swap
- **RX/TX Factor**: indica la larghezza di banda in percentuale che l'istanza può utilizzare; i valori devono essere espressi in un numero decimale tra 0 e 1

È possibile inoltre restringere l'accesso al *Flavor* a specifici progetti tramite la scheda *Flavor Access* presente nel form.

8.4.2 Key Pairs

Le coppie di chiavi (*Key Pairs*) sono coppie composte da una chiave privata ed una pubblica che servono per accedere alle istanze create tramite il protocollo SSH. È possibile sia importare una chiave pubblica precedentemente generata sia creare la coppia di chiavi direttamente dall'interfaccia web. Nel secondo caso la chiave pubblica verrà memorizzata sul cloud mentre quella privata verrà scaricata in automatico e immediatamente eliminata.

Per gestire le coppie di chiavi pubbliche è necessario aprire il menu a tendina *Project* > *Compute* e selezionare la voce *Key Pairs*. A questo punto per creare una nuova coppia di chiavi si deve cliccare sul pulsante *Create Key Pair*, mentre per importare una chiave pubblica già esistente si deve cliccare su *Import Public Key*.

8.4.3 Images

Le immagini sono template di macchine virtuali preconfigurate che fungono da base per la creazione di nuove istanze. Solitamente un'immagine è composta solamente dal sistema operativo di base configurato in modo da importare in automatico una chiave pubblica per consentire l'accesso alla nuova istanza tramite SSH senza l'utilizzo di password. Questo permette agli utilizzatori del cloud di avere piena autonomia nella costruzione delle loro istanze e quindi dell'infrastruttura che desiderano.

Per creare una nuova immagine da interfaccia grafica si deve aprire il menu a tendina *Project* > *Compute* e selezionare la voce *Images*. A questo punto cliccando sul pulsante *Create Image* verrà visualizzato il form che permette la creazione dell'immagine e che contiene i seguenti campi:

- **Image Name**: nome della nuova immagine
- **Image Description**: descrizione della nuova immagine
- **File**: file da caricare con l'immagine
- **Format**: formato dell'immagine
- **Architecture**: architettura del sistema operativo all'interno dell'immagine
- **Minimum Disk (GB)**: dimensione minima in GB del volume di storage
- **Minimum RAM (MB)**: dimensione minima in MB della memoria RAM
- **Visibility**: visibilità della nuova immagine agli altri utenti del cloud
- **Protected**: permette di proteggere l'immagine da cancellazioni accidentali

8.4.4 Instances

Un'istanza è una vera e propria macchina virtuale che viene istanziata partendo da un'immagine.

Per creare una nuova istanza è necessario aprire il menu a tendina a sinistra *Project* > *Compute*, selezionare la voce *Instances* e, una volta caricata la pagina con le informazioni su tutte le istanze del progetto, cliccare sul pulsante *Launch Instance*. In questo modo si aprirà un form con diverse schede da compilare con tutti i dati della nuova istanza; ciascuna di queste schede verrà trattata nei paragrafi seguenti.

Details.

- **Project Name**: nome del progetto; non è possibile selezionare un progetto diverso da quello attivo nella sessione, quindi se si vuole cambiare si deve fare prima della creazione dell'istanza
- **Instance Name**: nome dell'istanza
- **Description**: descrizione dell'istanza
- **Availability Zone**: nel caso in cui siano configurate più availability zones questo parametro permette di scegliere in quale di queste creare l'istanza
- **Count**: indica il numero di istanze da creare con la configurazione che si sta specificando

Source.

- **Select Boot Source**: permette di selezionare la sorgente dalla quale creare l'istanza; tale sorgente può essere un'immagine, un volume già esistente o uno snapshot
- **Create New Volume**: permette di specificare se creare un nuovo volume per l'istanza o se utilizzare un disco temporaneo; nel caso in cui si scelga come boot source un volume esistente questo parametro non viene mostrato
- **Volume Size**: permette di specificare la dimensione del nuovo volume (nel caso in cui venga creato)
- **Delete Volume on Instance Delete**: permette di specificare se il volume associato all'istanza (nel caso in cui esista) deve essere cancellato contemporaneamente all'istanza o meno
- **Allocated**: permette di selezionare la sorgente da cui viene creata la nuova istanza

Flavor. Questa scheda permette semplicemente di scegliere il flavor (ovvero la quantità di risorse) da assegnare alla nuova istanza

Networks. Questa scheda permettere di scegliere a quali reti collegare la nuova istanza; è possibile collegare un numero indefinito di reti alle istanze.

Security Groups. In questa scheda è possibile scegliere i security groups da associare alla nuova istanza.

Key Pair. Questa scheda permette di selezionare la chiave pubblica da importare all'interno dell'istanza in modo da poter eseguire l'accesso tramite SSH utilizzando la relativa chiave privata.

Configuration. All'interno di questa scheda è possibile inserire uno script che verrà eseguito all'avvio della macchina e che permette di configurare l'istanza in modo totalmente automatico. Permette inoltre di eseguire il partizionamento manuale del volume nel caso in cui ne sia stato aggiunto uno.

Capitolo 9

Terraform

Terraform è uno strumento open source multi-piattaforma per la creazione, configurazione e gestione delle infrastrutture di servizi cloud. Sviluppato nel linguaggio di programmazione Go da HashiCorp, il suo compito principale è quello di automatizzare la creazione di risorse, rendendo più semplice, prevedibile e ripetibile la gestione dell'intera infrastruttura. Grazie alla sua alta interoperabilità con molteplici provider di cloud, come ad esempio AWS di Amazon, GCP di Google e Azure di Microsoft, è possibile creare e gestire intere infrastrutture su più provider anche contemporaneamente (merito anche della forte capacità di concorrenza che Go fornisce). Usando in questo modo un unico strumento per la creazione delle risorse e delle infrastrutture è possibile accantonare il caotico utilizzo di diverse soluzioni e programmi proprietari durante le fase di provisioning, nonché il doversi districare nelle diverse interfacce grafiche.

Inoltre Terraform fornisce un'astrazione flessibile delle varie risorse dei provider; questa astrazione gli consente di gestire in modo semplice qualsiasi cosa, passando da hardware fisico a macchine virtuali e container, provider di posta elettronica e servizi DNS. Ciò che Terraform non offre, però, è il monitoraggio continuo delle risorse dell'infrastruttura, ma solamente le operazioni CRUD (create, read, update, delete). D'altro canto questo "svantaggio" lo rende uno strumento molto leggero e versatile e al tempo stesso molto veloce nell'applicare le configurazioni richieste; queste caratteristiche permettono a Terraform di poter esser eseguito su praticamente qualsiasi computer.

Infrastructure as Code. Terraform implementa l'idea di *Infrastructure as Code (IaC)*, ovvero la tecnica che consiste nel descrivere l'infrastruttura desiderata attraverso del codice ad alto livello. Lo scopo di questa tecnica è quella di rendere più veloce e ripetibile la creazione e la gestione delle infrastrutture e delle risorse, dato che è possibile descriverle attraverso del codice

manutenuti su file e non eseguendo comandi specifici da linea di comando o impostando configurazioni particolari da interfacce utente che, per quanto possano essere intuitive, non rendono una rappresentazione immediata delle configurazioni. Tramite questa tecnica è possibile condividere le configurazioni delle infrastrutture e risorse in rapidamente, garantendo un analisi e un responso immediato.

Per far fronte a queste esigenze, HashiCorp ha sviluppato un linguaggio di configurazione ad alto livello di sintassi simile a JSON chiamato *HashiCorp Configuration Language* o più comunemente *HCL*. Si tratta di un linguaggio dichiarativo e human-readable e serve per descrivere lo stato finale dell'infrastruttura che si vuole modellare. Contrariamente ai linguaggi procedurali infatti, non è possibile eseguire compiti in sequenza e controllare il flusso d'esecuzione. Aspetti come il calcolo dell'ordine di creazione o distruzione delle risorse nell'infrastruttura, ovvero il calcolo delle dipendenze delle risorse stesse, vengono controllati direttamente da Terraform; durante l'esecuzione dedurrà l'ordine corretto e creerà un diagramma delle dipendenze e le gestirà in maniera ottimale. Per esempio, se si volesse gestire una semplice infrastruttura OpenStack composta da una risorsa *domain* e una risorsa *user*, avendo quest'ultima una dipendenza necessaria con domain, prima verrà creato il domain desiderato e poi lo user; la scelta sull'ordine di esecuzione della creazione però è dedotta in prima parte da Terraform e non dal programmatore che crea l'infrastruttura.

Infine, un'altra caratteristica da non sottovalutare dello IaC è che si possa effettuare il versioning del codice utilizzando un *Distributed Version Control System (DVCS)*.

Immutable infrastructure. Terraform fornisce un'infrastruttura immutabile. Ciò implica che, ogni volta che si effettua un cambiamento della configurazione dell'infrastruttura, Terraform sostituisce tale configurazione con una nuova che tiene conto delle modifiche e poi applica un reprovisioning sull'intera infrastruttura che la nuova configurazione rappresenta. Le configurazioni precedenti comunque possono essere mantenute attraverso l'uso di un DVCS per poter effettuare rollback in caso di necessità.

Questo approccio è fondamentale per evitare l'insorgenza del *configuration drift*, ovvero quel fenomeno che si verifica quando la configurazione originale non corrisponde più all'infrastruttura che sta configurando. Questo può accadere con l'accumularsi delle modifiche, che a lungo andare possono portare a discrepanze tra la configurazione dell'infrastruttura e l'infrastruttura effettiva.

Edizioni. HashiCorp offre tre edizioni di Terraform per far fronte a varie esigenze che un team di sviluppo può incontrare.

Terraform Open Source (CLI) è la versione open source utilizzabile in locale sul proprio computer. Nonostante sia completamente gratuita, predispone di tutte le funzionalità per poter effettuare il provisioning delle infrastrutture su praticamente qualsiasi cloud provider. Terraform CLI è scaricabile come unico file binario con coi si interagisce dalla riga di comando.

Essendo che è completamente gratuita e che raggiunge il suo obiettivo principale, è l'edizione che è stata adottata in questo progetto di tesi.

Terraform Cloud è una piattaforma basata sul concetto di SaaS (Software-as-a-Service); Terraform viene eseguito in un ambiente remoto, dove l'infrastruttura e i dati sensibili vengono archiviati in maniera sicura. Include un'interfaccia utente molto semplice e completa, che può aiutare a comprendere meglio le operazioni e le risorse dell'infrastruttura. Si può anche connettere a viari servizi di DVCS.

Molte funzionalità sono gratuite per i piccoli team, per le organizzazioni più grandi invece diventa a pagamento.

Terraform Enterprise è la versione aziendale di Terraform Cloud, rivolto alle grandi organizzazioni con severi requisiti di sicurezza. Include funzionalità tra cui il controllo degli accessi basato su ruoli e l'auditing.

9.1 Funzionamento

Terraform crea e gestire le risorse sulle piattaforme cloud rispettivamente attraverso le loro API (Application Programming Interface). Queste API sono il punto d'accesso che i vari cloud provider devono garantire in generale per poter interagire coi servizi e le risorse che loro stessi offrono. Non esiste uno standard de facto sul design model delle API, e la loro implementazione e struttura è a completa discrezione del provider in base ai servizi e risorse che offrono; di conseguenza, un API di un provider risulterà diversa dalla API di un altro provider.

Per poter interfacciare correttamente Terraform e la moltitudine di API esistenti, è necessario l'utilizzo di vari componenti intermediari (plugin), chiamati *Terraform Provider* (figura 9.1), sviluppati talvolta dai cloud provider stessi, talvolta dalla community o da HashiCorp.

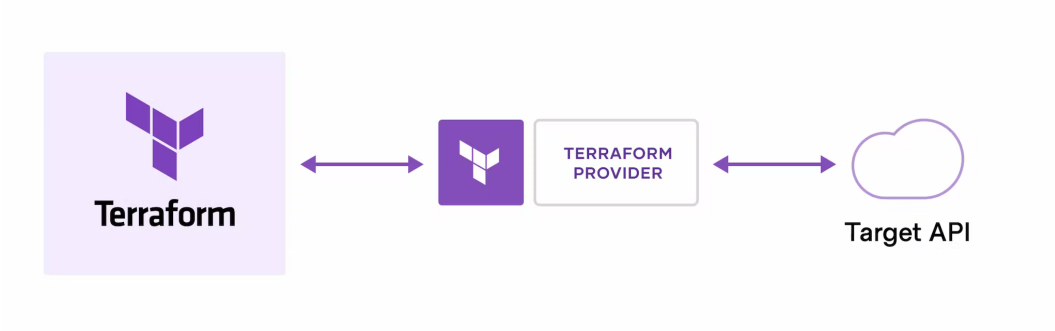


Figura 9.1: Comunicazione tra Terraform e i vari cloud provider [40].

Terraform dunque utilizza il concetto di plugin, ovvero a seconda del cloud provider con cui dovrà comunicare utilizzerà degli altri componenti esterni al momento del bisogno.

L'architettura di Terraform è così divisa logicamente in due parti: *Terraform Core* e *Terraform Plugins*.

9.1.1 Terraform Core

Terraform Core è il componente principale di Terraform. Si tratta di un piccolo file binario compilato staticamente scritto nel linguaggio di programmazione Go. Il codice sorgente è completamente open source ed è lo strumento che l'utente utilizza attraverso il comando `terraform` a riga di comando.

Si occupa di tutti gli aspetti più importanti per il provisioning con successo dell'infrastruttura desiderata, dalla lettura dei file di configurazione fino all'esecuzione e alla richiesta di creazione delle risorse.

- **Caricamento e interpretazione dei file di configurazione.** Terraform legge i vari file di configurazione creati dall'utente, che rappresentano le risorse dell'infrastruttura, su cui poi effettuerà le varie operazioni CRUD. Offre anche un piccolo servizio di validazione e formattazione dei vari file e in caso di errori li segnala all'utente.

- **Gestione del file dello stato dell'infrastruttura.** Terraform salva all'interno di un file in formato JSON una copia locale dello stato dell'intera infrastruttura e la mantiene sincronizzata con l'attuale infrastruttura presente nel cloud provider. Questa operazione gli consente di capire quali risorse sono state create e quali devono essere modificate, ed è necessaria per mantenere salvati i loro metadati, come ad esempio le varie dipendenze che hanno tra loro (informazioni utilizzate in fase

di eliminazione per determinare il corretto ordine di cancellazione delle risorse).

- **Costruzione del grafo delle risorse.** Il grafo delle risorse è un diagramma delle dipendenze che Terraform crea basandosi sulle risorse definite nei vari file di configurazione dell'infrastruttura. Viene generato durante la generazione o modifica dell'infrastruttura e viene percorso per poter generare o modificare nell'ordine corretto le varie risorse. In caso di eliminazione di una risorsa, Terraform calcolerà le sue dipendenze basandosi sul file di stato, in quanto la risorsa non sarà più presente all'interno dei file di configurazione e dunque non è possibile generare correttamente il diagramma delle dipendenze.

- **Esecuzione del plan.** Prima di effettuare il provisioning dell'infrastruttura che l'utente desidera, viene creato un piano di esecuzione, che mostra le azioni che Terraform intraprenderà per applicare la configurazione. A questo punto Terraform eseguirà il piano fino a quando tutte le azioni descritte non vengono eseguite o al verificarsi di un errore. In entrambi i casi, Terraform crea un nuovo file dello stato dell'infrastruttura. L'utente può anche salvare il piano in un file a parte ed avviare l'esecuzione in un secondo momento.

- **Comunicazione con i plugin.** I plugin sono anch'essi file binari compilati staticamente scritti nel linguaggio Go e sono separati da Terraform Core. Ogni plugin fornisce un'implementazione astratta dei servizi e delle risorse di un cloud provider specifico, interfacciandosi con le API che mette a disposizione; ad esempio il plugin per Azure implementa la gestione delle risorse soltanto del cloud Azure. Terraform Core richiama le funzioni di questi file binari attraverso le RPC (remote procedure calls).

  I plugin verranno trattati nei dettagli nella sezione 9.1.2

9.1.2 Terraform Plugins

Come accennato nell'ultimo punto nella sezione 9.1.1, i plugin sono file binari scritti in Go e richiamati da Terraform Core attraverso le RPC. Ogni plugin espone un'implementazione per un servizio specifico e ve ne sono di due tipi: i *Terraform Provider* per la gestione dei cloud provider e *Terraform Provisioner* per l'esecuzione di comandi, ad esempio script bash; entrambe le tipologie di plugin vengono eseguiti in un processo separato da Terraform Core ed espongono un'interfaccia per poter comunicare con esso.

Per garantire un alto livello di modularità, i plugin, in particolare i Terraform Provider, non comunicano direttamente con le API del cloud provider ma

passano attraverso delle librerie esterne a Terraform, che gestiscono la corretta comunicazione tra i plugin e il cloud provider (figura 9.2).

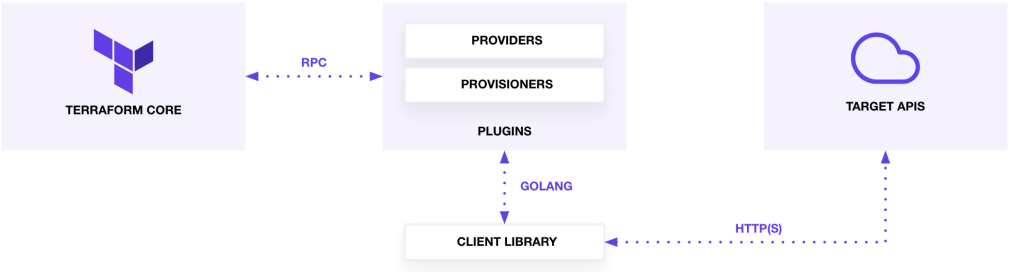


Figura 9.2: Approfondimento delle comunicazioni dal Terraform Core fino alle API dei cloud provider [39].

Di seguito i Terraform Provider e i Terraform Provisioner verranno trattati maggiormente nei dettagli.

Terraform Provider. I Terraform Provider sono un'astrazione delle API del cloud provider e delle risorse e servizi che esso offre. Sono necessari per poter creare l'infrastruttura desiderata, in quanto sono loro che si occupano di inviare le richieste al cloud provider che rappresentano. Ad ogni Terraform Provider corrisponde soltanto un cloud provider e non è dunque possibile utilizzare un Terraform Provider per un cloud provider a cui originalmente non è stato progettato.

Come per il Terraform Core, sono file binari compilati staticamente scritti nel linguaggio di programmazione Go. Sono separati dal Terraform Core e vengono scaricati dinamicamente dallo stesso, il quale, durante la fase di caricamento e lettura dei file di configurazione, è in grado di determinare quali plugin sono necessari.

Di solito, e come consigliato dalla stessa HashiCorp [39] nel caso in cui si voglia sviluppare il proprio Terraform Provider, questi plugin non interagiscono direttamente con le API dei cloud provider, ma effettuano delle chiamate a delle librerie indipendenti dei cloud provider; queste fanno da tramite tra i plugin e le API. Questo aspetto consente di mantenere una struttura modulare e di separare meglio i vari ruoli. Anche in questo caso, ogni cloud provider sviluppa la propria libreria con le proprie chiamate; ad esempio il provider AWS sfrutta l'SDK AWS per Go.

In sintesi, un Terraform Provider contiene tutto il codice necessario per autenticare un utente verso il cloud provider, connettersi ai vari servizi e gestire le risorse, nonché effettuare tutte le operazioni CRUD che l'utente richiede.

I Terraform Provider possono essere sviluppati e pubblicati sia da HashiCorp, sia dai fornitori dei servizi cloud stessi, sia dalla community, e possono essere cercati nel *Terraform Registry* [1].

HashiCorp, inoltre, fornisce un framework ad alto livello per poter sviluppare i Terraform Provider in modo che poi Terraform Core riesca ad usarli correttamente attraverso le RPC.

Terraform Provisioner. Sono plugin che vengono utilizzati per poter eseguire comandi o script su un host locale o remoto durante la creazione o eliminazione di risorse. Anch'essi, per poter adempire ai loro scopi, utilizzano librerie indipendenti esterne; per esempio per implementare l'esecuzione remota su un host Windows, viene utilizzata la libreria open source Windows Remote Management (WinRm) scritta in Go [2].

A differenza dei Terraform Provider, i Terraform Provisioner sono integrati all'interno di Terraform Core, e non è possibile crearne nuovi. Questo perché lo scopo principale di Terraform è descrivere un'infrastruttura come "modello dichiarativo", nascondendo e slegando i concetti di provisioning della stessa. Inoltre aggiungono una notevole quantità di complessità e incertezza nel corretto utilizzo di Terraform, dato che non è possibile modellare queste azioni. Infatti, HashiCorp sconsiglia l'utilizzo dei provisioner per qualsiasi casistica, raccomandando invece l'utilizzo delle normali tecniche di configurazione prima di utilizzarli.

Nonostante siano parte integrali del Terraform Core, vengono comunque eseguiti in processi separati e richiamati sempre attraverso le RPC.

9.2 I Progetti con Terraform

Un progetto Terraform rappresenta una singola configurazione di un'infrastruttura che si vuole modellare, che sia questa composta da un singolo cloud provider o da un inseme di essi.

Di seguito verranno approfonditi gli aspetti riguardanti lo sviluppo delle configurazioni dell'infrastrutture, passando dalle varie fasi nello sviluppo, alla struttura logica di file e directory fino al linguaggio vero e proprio HCL.

[1] Terraform Registry: `https://registry.terraform.io`, ultimo accesso 2 Febbraio 2023
[2] WinRm repo: `https://github.com/masterzen/winrm`, ultimo accesso 2 Febbraio 2023

9.2.1 Workflow principale nello sviluppo con Terraform

Il workflow consiste in una serie di fasi che accompagnano lo sviluppatore durante tutta la vita del progetto, dalla scrittura del codice Terraform, all'applicazione delle configurazioni fino ad una sua eventuale distruzione. Queste fasi sono progettate per essere ripetibili, scalabili e collaborative, rendendole adatte per la gestione di infrastrutture di qualsiasi dimensione. In figura 9.3 è mostrato in sintesi il workflow per quanto riguarda la creazione di un progetto Terraform.

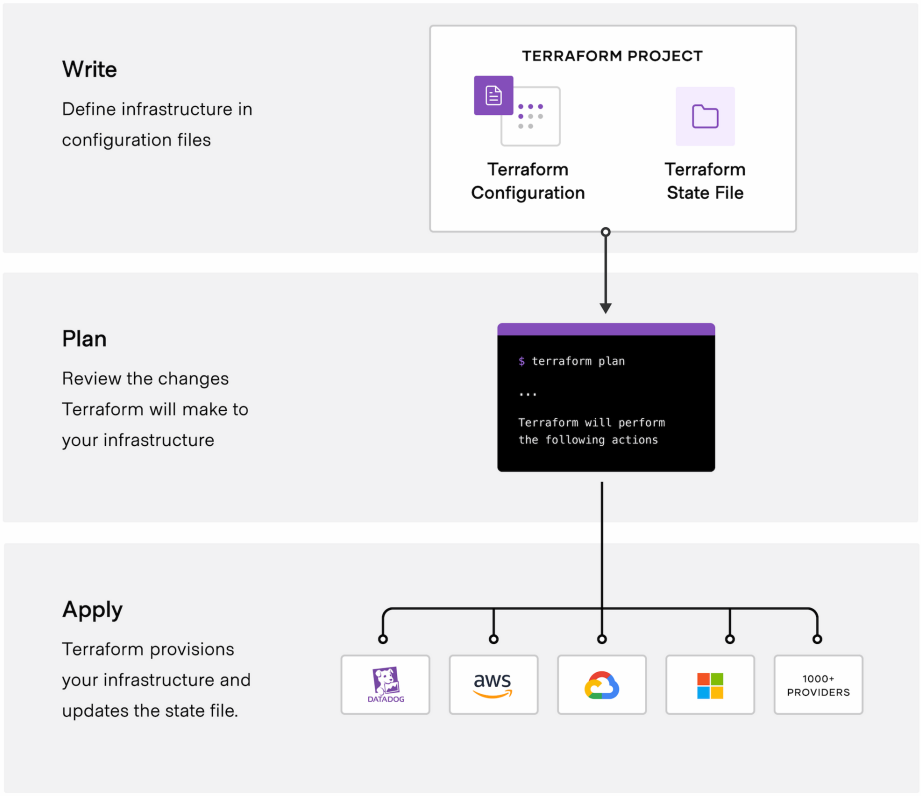


Figura 9.3: Punti focali del workflow di Terraform [40].

Write. Questa è la fase in cui l'utente modella e descrive l'intera infrastruttura attraverso la stesura del codice nei file di configurazione. Vengono definiti i vari provider dei servizi cloud, le risorse appartenenti all'infrastruttura e le varie variabili da utilizzare in input per rendere dinamica la configurazione. Come per ogni progetto in ambito ICT, è buona prassi affiancare tale processo con uno strumento di DVCS come Git, che permette di controllare le varie modifiche apportate all'infrastruttura. Dopo aver scritto i file di configurazione

dove vengono dichiarate la versione di Terraform Core e i vari plugin da utilizzare, è possibile inizializzare il progetto (workspace, sezione 9.2.2), attraverso il comando `terraform init`. Questo comando deve essere necessariamente eseguito ogni volta che si aggiungono dei plugin o dei moduli.

Terraform in questa fase fornisce due semplici strumenti, che affiancano l'utente ad una corretta stesura del codice e ad un'individuazione degli errori di sintassi.

- Con il comando `terraform fmt` è possibile formattare correttamente tutti i file di configurazione all'interno del progetto secondo le convenzioni di stile definiti da HashiCorp; è possibile anche indicare al comando di mostrare i cambiamenti che verranno applicate e di non modificare in automatico la formattazione.

- Con il comando `terraform validate` si effettua un check che verifica se i file di configurazione locali sono sintatticamente validi e internamente consistenti, indipendentemente dal valore dalle variabili fornite o dallo stato esistente. Questo comando richiede che la cartella del progetto sia stata inizializzata. Per maggiori dettagli sul comando, si veda la documentazione [43].

Ad ogni modifica apportata ai file di configurazione è buona prassi eseguire entrambi i comandi per una più facile gestione di eventuali problemi ed errori.

Plan. Plan, assieme ad Apply e Destroy fa parte delle fasi del provisioning dell'infrastruttura: questa è la fase dove viene deciso il piano d'azione. Qui Terraform caricherà tutti i file di configurazione scritti, eseguirà un'analisi dell'infrastruttura modellata e costruirà il grafo delle risorse. In assenza di errori, mostrerà un'anteprima delle azioni che prevederà di apportare per raggiungere lo stato desiderato, evidenziandone, se sono presenti, i futuri cambiamenti all'infrastruttura.

Il comando per avviare questa prima fase è `terraform plan`. Oltre a visionare il piano, è anche possibile salvarlo su file e caricarlo in un secondo momento durante la fase di applicazione del piano. Di per sé Terraform con questo comando non esegue le azioni mostrate nel piano; questa fase è un modo per verificare se le modifiche proposte corrispondono all'idea di infrastruttura pianificata.

Apply. Questa è la fase in cui Terraform esegue effettivamente il provisioning dell'infrastruttura in base ai file di configurazione creati, istanziando le risorse attraverso i Terraform Provider del progetto. All'esecuzione del comando `terraform apply`, Terraform crea prima il plan proprio come con

`terraform plan` e sotto conferma da parte dell'utente, esegue effettivamente le azioni mostrate; quindi costruirà il grafo delle risorse e le creerà o modificherà in ordine in base alle loro dipendenze. Il comando può anche accettare in input un file del piano salvato precedentemente.

Al termine, Terraform creerà un nuovo file rappresentante la copia locale dello stato dell'infrastruttura e l'eventuale file vecchio verrà archiviato.

Destroy. Con Terraform è possibile anche eliminare in modo efficiente e sicuro un'infrastruttura precedentemente organizzata. Eseguendo il comando `terraform destroy`, un alias del comando `terraform apply -destroy`, è possibile eseguire tale operazione. Anche in questa fase Terraform eseguirà le richieste attraverso i Terraform Provider del progetto, e al termine del processo aggiornerà il file di stato (verosimilmente non contenente più l'infrastruttura).

Come per la creazione dell'infrastruttura, è possibile organizzare un piano di distruzione tramite il comando `terraform plan -destroy` prima di eseguire la sua completa eliminazione; in questo modo è possibile visionare l'effettiva distruzione di tutte quante le risorse dell'infrastruttura prima della loro eliminazione. Infine, è anche possibile eliminare singole risorse, o cancellandole direttamente dai file di configurazione, o attraverso il comando `terraform destroy -target=<NOME_RISORSA>`

9.2.2 Struttura del workspace dei progetti con Terraform

Una configurazione Terraform è strutturata all'interno di un unica cartella, che rappresenta l'infrastruttura da modellare nonché il workspace del progetto. La struttura dei file è molto semplice ed è composta da pochi elementi suddivisibili in due categorie: quelli creati dall'utente e quelli creati da Terraform.

In figura 9.4 viene mostrato un esempio dell'organizzazione dei file di un semplice progetto. In questo progetto vengono creati due utenti in un cloud OpenStack pronto all'uso, e sarà utilizzato per le spiegazioni delle sezioni 9.2.2 e 9.2.3.

Creati dall'utente. I file che l'utente crea sono quelli riguardanti le configurazioni e le risorse dell'infrastruttura che si vuole gestire. Questi file hanno estensione `*.tf` e `*.tfvars` e saranno i file che Terraform leggerà per il corretto provisioning dell'infrastruttura. In figura 9.4 è possibile vedere questi file nel secondo riquadro.

Il modo in cui si organizzano i file che l'utente crea all'interno del workspace e i loro nomi è completamente soggettivo; tuttavia è possibile seguire alcune linea guida per strutturare i file in modo tale il progetto possa essere facilmente manutenuto e ridimensionato in caso di esigenza.

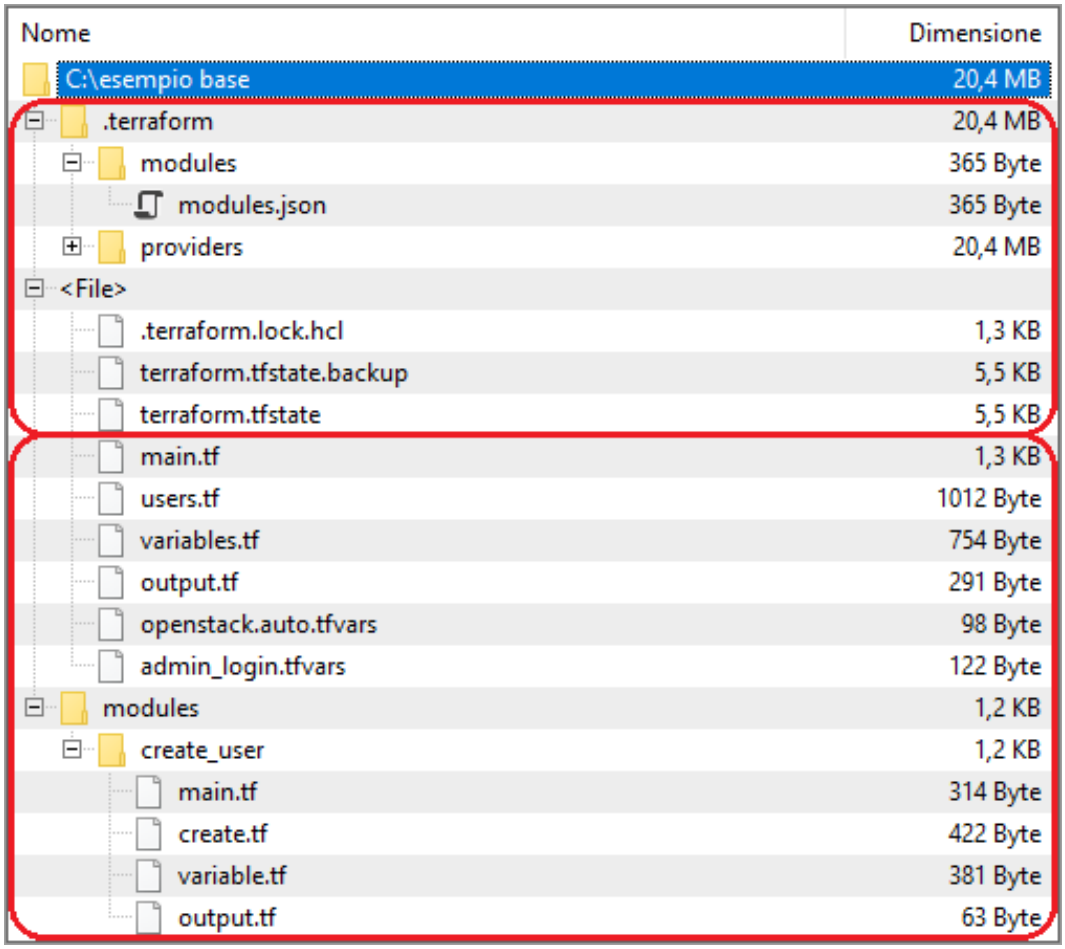


Figura 9.4: Esempio della struttura di un progetto Terraform, diviso tra file gestiti da Terraform (1° riquadro) e file creati dall'utente (2° riquadro).

- **Terraform e Providers.** Il file principale del progetto viene chiamato `main.tf`. Al suo interno vengono indicate le definizioni di Terraform e dei plugin provider da utilizzare. Se i provider sono numerosi, è possibile scorporare ulteriormente il file; in questo caso potrebbe essere una buona idea creare due file dai nomi `terraform.tf` e `provider.tf`, contenenti rispettivamente le definizioni necessarie per Terraform e le definizioni dei provider.

  Definiti Terraform e i provider da adoperare, è possibile utilizzare il comando `terraform init` per inizializzare il workspace.

- **Risorse.** Le risorse che condividono uno scopo o ambito è opportuno

raggrupparli in un unico file `*.tf`. Nell'esempio riportato in figura 9.4 è presente il file `users.tf` dentro al quale vengono definite le risorse per la gestione di due utenti.

- **Variabili.** Le variabili sono un aspetto molto importante nella descrizione dell'infrastruttura e anch'esse meritano di essere dichiarate in appositi file. Ve ne sono di tre tipi: **input**, **output** e **local**.

  Le variabili di tipo **input** sono utilizzate per poter inserire dinamicamente delle informazioni durante il plan dell'infrastruttura e se il loro valore non è definito, l'esecuzione del plan fallirà. La loro dichiarazione viene effettuata all'interno di un file di tipo `*.tf`, solitamente chiamato `variables.tf`. Per quanto riguarda l'assegnamento del loro valore invece sono possibili tre modalità a seconda del loro scopo: a caricamento automatico, indicando il file dove vengono definite oppure inserendo il loro valore a riga di comando. Nel primo e secondo caso, le variabili vengono definite all'interno dei file terminanti in `*.tfvars`. Per il primo caso, vengono caricate automaticamente tutte le variabili salvate all'interno dei file terminanti in `*.auto.tfvars`; anche le variabili all'interno del file `terraform.tfvars` vengono caricate automaticamente. Nell'esempio in figura 9.4, il file `openstack.auto.tfvars` contiene i valori per stabilire la connessione con il provider. Nel secondo caso invece è possibile passare durante il plan (e di conseguenza anche durante l'apply) uno o più file `*.tfvars`, aggiungendo al comando `terraform plan` o `terraform apply` il parametro `-var-file="<NOME_FILE>.tfvars"`. Come ultima casistica invece, verranno richieste a riga di comando tutte quelle variabili che non sono state definite nei modi precedenti.

  Le variabili di tipo **output** sono utilizzate per rendere disponibili le informazioni sull'infrastruttura dopo il suo provisioning. Per questa tipologia di variabile viene usato il file `output.tf`. A provisioning terminato, Terraform mostrerà il valore di queste variabili a riga di comando. In oltre, è possibile visionare il loro valore utilizzando il comando `terraform output`.

  Le variabili **local** invece vengono dichiarate e definite assieme alle risorse, per cui è più coerente posizionarli assieme ad esse. Sono molto utili per definire valori d'appoggio o temporanei.

- **Moduli.** I moduli sono un modo per riutilizzare configurazioni delle risorse o gruppo di risorse codificate anche per altre infrastrutture (sempre dello stesso provider). Ogni modulo non è altro che una raccolta di file `*.tf` e all'interno possono esser presenti risorse e variabili. Nel caso siano presenti variabili di input, il loro valore viene passato quando

viene chiamato il modulo. L'organizzazione in file di un modulo è molto simile a quella di un qualsiasi progetto Terraform; pertanto nel qual caso si vogliano utilizzare è opportuno mantenere tale struttura anche per i moduli. Per una maggior leggibilità, è consigliato creare una cartella `modules` all'interno del workspace principale, dentro al quale è possibile organizzare in varie sottocartelle i moduli che si andranno ad utilizzare. Nel file `main.tf` è sufficiente dichiarare i provider utilizzati all'interno del blocco, senza definirli esplicitamente.

Nel progetto d'esempio in figura 9.4 è stato utilizzato un modulo (cartella `create_user`) che permette la creazione di un utente, l'assegnazione di un ruolo e di un progetto. In questo modo, nella configurazione principale, è sufficiente richiamare due volte il modulo per la creazione dei due utenti, passando solamente qualche valore per le variabili di input che richiede.

Gestiti da Terraform. I file controllati da Terraform sono quelli inerenti alla gestione dei plugin e allo stato dell'infrastruttura. In figura 9.4 è possibile vedere questi file nel primo riquadro.

Eseguendo il comando `terraform init` sul workspace, Terraform leggerà tutti i file di configurazione ed interpreterà le definizioni dei provider (nell'esempio in figura 9.4 sono presenti nel file `main.tf`); da queste provvederà a scaricare i plugin dei provider se non sono stati forniti localmente e a salvarli all'interno della cartella `.terraform/providers`. Successivamente creerà il file `.terraform.lock.hcl` su cui memorizzerà le informazioni dei provider scaricati. Inoltre, se per l'infrastruttura sono stati utilizzati dei moduli, Terraform come per i plugin provvederà a scaricarli se non localmente presenti e li salverà all'interno della cartella `.terraform/modules`, aggiungendo all'interno di quest'ultima un file `modules.json` per tenerne traccia. Nell'esempio in figura 9.4 è stato utilizzato il plugin provider di OpenStack (si veda il paragrafo Plugin provider openstack nella sezione 9.3) e un modulo per la creazione degli utenti; il plugin viene scaricato da Terraform, mentre per il modulo, essendo presente in locale, viene solamente aggiunto il riferimento all'interno del file `modules.json`.

Dal primo provisioning eseguito con successo (effettuato attraverso il comando `terraform apply`), verrà creato il file `terraform.tfstate`; questo file rappresenta la copia locale in formato JSON dello stato corrente dell'intera infrastruttura che si sta modellando. Grazie a questo file, Terraform può tenere traccia di tutti i metadati dell'infrastruttura. Nei successivi provisioning, Terraform confronta il vecchio stato con l'infrastruttura nuova e applica le differenze; dopodiché il vecchio file di stato viene rinominato in

`terraform.tfstate.backup` e ne viene generato un nuovo che rispecchia fedelmente l'infrastruttura corrente.

Terraform fornisce anche la possibilità di salvare i file di stato in una locazione remota e sicura: quest'opzione è molto utile in ambienti collaborativi, dove più persone potrebbero modificare la configurazione dell'infrastruttura. In questo modo si evitano l'insorgenza di conflitti e discrepanze dovute alle varie versioni dei file di stato che ogni singolo membro del team potrebbe conservare. Inoltre, le informazioni custodite all'interno dei file di stato sono conservate in chiaro, anche i dati sensibili (come ad esempio i dati di login degli utenti) e quindi per progetti più organizzati sarebbe meglio non conservare tali file ne in locale ne alla consultazione di tutti. Questo aspetto non è stato affrontato in questa sede, pertanto se è necessaria la configurazione di tale funzionalità, si veda la documentazione [38].

9.2.3 HCL: costrutti base e funzionalità aggiuntive

HCL è stato sviluppato da HashiCorp con l'idea di creare un linguaggio di configurazione completo e al tempo stesso intuibile e facilmente interpretabile dall'uomo. Non si tratta solamente di un'insieme di regole che definiscono una sintassi, ma di un vero e proprio piccolo framework, in grado di fornire un insieme di costrutti e funzionalità che permettono il raggiungimento dello scopo.

I blocchi e gli argomenti Il linguaggio di Terraform è dichiarativo, descrive l'obiettivo piuttosto che i passaggi su come raggiungere quell'obiettivo. Quindi lo scopo principale è quello di dichiarare le risorse e le loro caratteristiche, che siano componenti hardware o software; queste rappresentano gli oggetti dell'infrastruttura. Per fare ciò, viene usata la nozione di **blocco**.

```
<BLOCK TYPE> ["<BLOCK LABEL>"] ["<BLOCK LABEL>"] {
  # Block body
  [<IDENTIFIER> = <EXPRESSION>] # Argument
}
```

Listato 9.1: Sintassi base dei blocchi su HCL.

Si prenda l'esempio in listato 9.1, dove viene esplicata la struttura base di un blocco. Un blocco è definito dalla sua tipologia (`BLOCK TYPE`) e, a seconda di questa, da zero, una o due etichette (`BLOCK LABEL`). Queste etichette possono rappresentare il nome generico e locale dell'oggetto. All'interno dei blocchi, racchiuse tra le parentesi `{` e `}`, si possono trovare un qualsiasi numero di argomenti che rappresentano in generale gli attributi di una risorsa (`IDENTIFIER`) ai quali viene assegnato un valore (`EXPRESSION`). Questo

valore può essere composto da valori singoli o da una combinazione di altri valori, anche da blocchi nidificati. Una nota molto importante è che l'ordine con cui vengono scritti i blocchi e gli argomenti al loro interno è del tutto superfluo per Terraform. Terraform considera solo le relazioni di dipendenza tra i vari oggetti e in fase di plan costruisce il diagramma delle dipendenze, da cui poi viene determinato l'ordine di esecuzione delle varie operazioni per il provisioning.

Nel HCL esistono una manciata di tipologie di blocchi di primo livello, ovvero quei blocchi che possono apparire all'esterno di qualsiasi altro blocco in un file di configurazione, e non è concesso crearne dei nuovi. Di seguito verranno elencati e spiegati in sintesi, prendendo come riferimento il codice d'esempio del progetto della figura 9.4.

Blocco `terraform`: è il blocco principale della configurazione. Non ha etichette e serve per dichiarare sia la versione di Terraform Core necessaria per eseguire il provisioning dell'infrastruttura sia i provider che si andranno ad utilizzare. Dei vari provider viene indicata la versione e la sorgente da dove Terraform, nel caso in cui questa non sia un indirizzo locale, scaricherà il plugin.

Nel progetto d'esempio questo blocco è stato inserito nel file `main.tf` e nel listato 9.2 si può vedere com'è formato: in particolare, è stato dichiarato il provider openstack con sorgente la locazione nel Terraform Registry.

```
terraform {
  required_version = ">= 1.3.4"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.49.0"
    }
  }
}
```

Listato 9.2: Sintassi del blocco `terraform` nel file `main.tf`.

Blocco `provider`: questo blocco serve per definire tutti quei valori che sono necessari al plugin provider per funzionare. Essendo specifico per un singolo provider, il suo contenuto può variare a seconda delle esigenze del plugin. Ha soltanto un etichetta dove viene indicato il nome del provider che si sta definendo. Con l'argomento `alias` è possibile associare un nome univoco alla configurazione del provider, così facendo è possibile creare più blocchi provider dello stesso plugin provider ma con configurazioni diverse. Questa pratica è utile nella creazione di risorse associate a configurazioni di provider differenti.

Anche questo blocco è stato inserito nel file `main.tf` e nel listato 9.3 è possibile vedere in linea di massima com'è organizzato; sono state richiamate delle variabili di tipo input (dicitura `var.`) e la loro spiegazione verrà trattata nei punti successivi.

```
provider "openstack" {
  region    = var.openstack_environment.region
  auth_url = var.openstack_environment.auth_url

  user_name   = var.user_name
  password    = var.user_password
  domain_name = var.user_domain
  tenant_name = var.user_project

  insecure = true
}
```

Listato 9.3: Sintassi del blocco `provider` nel file `main.tf` .

Blocco `resource` : il blocco resource rappresenta effettivamente la risorsa che l'utente desidera creare ed l'elemento più importante del linguaggio. HCL non impone limiti su ciò che può rappresentare e questo dipende esclusivamente dalle risorse che il plugin provider mette a disposizione. In genere ha due etichette; la prima rappresenta il nome generico (o tipo) della risorsa che si vuole utilizzare, la seconda invece rappresenta il nome locale che la risorsa avrà. Entrambi i nomi servono per identificare univocamente al risorsa all'interno dell'infrastruttura tramite la nomenclatura `<TIPO_RISORSA>.<NOME_LOCALE>` . Infine è possibile accedere agli argomenti delle risorse tramite la nomenclatura `<TIPO_RISORSA>.<NOME_LOCALE>.<NOME_ARGOMENTO>` .

Nel listato 9.4 è riportata la risorsa `openstack_identity_user_v3` , necessaria per il provisioning dell'utente. Questo blocco è stato inserito all'interno del file `users.tf` e, come si può vedere, è stata richiamata una variabile di tipo locale (dicitura `local`) e anche questa verrà spiegata nei punti successivi.

```
resource "openstack_identity_user_v3" "user_alice" {
  description = local.desc
  name = "alice"
  password = "123"
}
```

Listato 9.4: Sintassi del blocco `resource` nel file `users.tf` .

All'interno del blocco resource è possibile definire uno o più **meta-argomenti**. I meta-argomenti sono argomenti speciali che possono essere utilizzati con ogni tipo di blocco resource, e permettono a seconda della tipologia di estendere

l'espressività che la risorsa stessa offre. HCL implementa cinque tipi di meta-argomenti.

- Con `depends_on` è possibile indicare delle dipendenze aggiuntive che si vogliono far avere alla risorsa. Si dichiara come un argomento semplice, e il suo valore è un elenco di riferimenti ad altre risorse o moduli. In questo modo si dichiara esplicitamente a Terraform di completare tutte le azioni sugli oggetti di dipendenza prima di eseguire azioni sull'oggetto che dichiara le dipendenze.

- Con `count` e `for_each` Terraform offre la possibilità di poter rappresentare più oggetti dell'infrastruttura reale attraverso un unico blocco resource o module. In questo modo è possibile creare più istanze dello stesso blocco, ad ognuna di queste è associato un oggetto dell'infrastruttura distinto e le azioni che vengono effettuate su esso vengono effettuate separatamente dagli altri. Count accetta un numero intero e crea quel numero di istanze della risorsa o del modulo. All'interno dei blocchi dove count è impostato, è presente un oggetto aggiuntivo avente un solo attributo, `count.index`, indicante il numero di istanza corrispondente. Infine, per riferirsi ad un oggetto specifico della risorsa, bisogna indicare l'indice dell'istanza, usando la nomenclatura `<TIPO_RISORSA>.<NOME_LOCALE>[<INDICE>]`. For_each funziona similmente a count, ma accetta o un set di stringhe o una mappa key-value; in questo caso all'interno dei blocchi dov'è impostato for_each è disponibile un oggetto `each` avente due attributi. Tramite `each.key` viene indicata la chiave della mappa che corrisponde all'istanza, mentre con `each.value` si fa riferimento al suo valore (in caso di set di stringhe, each.key e each.value coincidono). In maniera analoga a count, per riferirsi ad un oggetto specifico della risorsa, bisogna indicare il nome dell'istanza, che in questo caso corrisponderà al nome della chiave; quindi usando la nomenclatura `<TIPO_RISORSA>.<NOME_LOCALE>[<KEY>]`.

- Con `provider` è possibile, se è stato impostato, specificare una configurazione diversa del provider a cui si sta facendo riferimento. Infatti, come spiegato in precedenza, nel blocco `provider` è possibile assegnare all'argomento `alias` un nome. Questo nome viene poi utilizzato come valore per il meta-argomento provider nelle risorse; in questo modo si associa una particolare configurazione provider a quella risorsa.

- Per finire `lifecycle` è utilizzato per personalizzare dei dettagli del ciclo di vita delle risorse. Non è semplicemente un argomento, bensì si tratta di un blocco nidificato che al suo interno può contenere meta-argomenti o

blocchi. I meta-argomenti supportati sono: `create_before_destroy`, `prevent_destroy`, `ignore_changese` e `replace_triggered_by`.

I primi due accettano valori booleani; `create_before_destroy` serve per indicare di creare la risorsa prima che questa venga distrutta durante un rimpiazzamento (e non dopo) mentre `prevent_destroy` per generare un errore se viene avviata la distruzione della risorsa. Gli ultimi due invece accettano un elenco di attributi; `ignore_changese` fa si che se all'esterno della configurazione dell'infrastruttura, come ad esempio all'interno del cloud provider stesso, vengono effettuate delle modifiche agli attributi specificati, Terraform non effettuerà il re-provisioning cercando di sovrascrivere queste modifiche. `replace_triggered_by` invece accetta anche i riferimenti alle risorse e serve per indicare a Terraform che se è stato effettuato un cambiamento agli elementi specificati, deve sostituire la risorsa del meta-argomento lifecycle con una nuova.

I blocchi supportati da lifecycle sono `precondition` e `postcondition`, entrambi servono per effettuare verifiche sulla validità di determinati valori ed entrambi accettano gli argomenti `condition` e `error_message`; se l'espressione che viene passata a condition è falsa, Terraform interromperà il provisioning e restituirà il messaggio indicato in error_message. Con precondition la verifica della condizione viene effettuata prima della creazione/modifica della risorsa, viceversa con postcondition questa verifica viene effettuata dopo.

Blocco `data`: questo blocco consente di recuperare le informazioni di risorse definite al di fuori della configurazione dell'infrastruttura, come ad esempio risorse già istanziate all'interno del cloud provider. Come per il blocco resource, è a completa discrezione del plugin provider definire quali risorse è possibile leggere e quali dati poter caricare. Anche in questo caso sono presenti due etichette, le stesse per il blocco resource; in aggiunta per riferirsi univocamente ad un istanza di questo tipo, bisognerà aggiungere `data.` davanti alla nomenclatura. All'interno del blocco è importante definire un argomento che sia in grado di riferirsi univocamente alla risorsa che si desidera leggere.

Nel listato 9.5 è stata importata la risorsa `openstack_identity_role_v3` avente nome `member`; questa risorsa viene creata in automatico durante le fasi di installazione di OpenStack e corrisponde al ruolo che gli utenti base avranno. Per richiamare questa risorsa in altri punti dell'infrastruttura, bisognerà scrivere `data.openstack_identity_role_v3.role_member`. Questo blocco è stato inserito all'interno del file `users.tf`.

```
data "openstack_identity_role_v3" "role_member" {
  name = "member"
}
```

Listato 9.5: Sintassi del blocco `data` nel file `users.tf`.

Blocco `variable`: è il blocco che identifica la prima tipologia di variabile (vedi punto Variabili nella sezione 9.2.2), ovvero quella di tipo input. Possiede soltanto un etichetta, la quale indica il nome univoco della variabile all'interno del progetto. Ad ogni variabile è necessario indicare la tipologia, che sia questa una stringa, un numero, un valore booleano, un dato complesso o altro, ed è possibile associarle una descrizione. Inoltre è possibile aggiungere argomenti in grado di arricchire e potenziare l'uso delle variabili. Ad esempio è possibile aggiungere dei costrutti per permettere di verificare se il valore che assumeranno rispetti determinati vincoli, è possibile dichiarare che conterranno un dato sensibile, definire valori di default o renderle opzionali.

La loro dichiarazione viene effettuata all'interno dei file `*.tf` mentre la loro definizione all'interno dei file `*.tfvars`. Nell'esempio in esame sono state dichiarate nel file `variables.tf` e definite nel file `admin_login.tfvars` e `openstack.auto.tfvars`. Come già spiegato nel punto Variabili nella sezione 9.2.2, durante il plan e l'apply `openstack.auto.tfvars` viene caricata automaticamente, mentre `admin_login.tfvars` deve essere caricata esplicitamente, ad esempio per l'apply si deve indicare al comando il nome del file con `terraform apply -var-file"admin_login.tfvars"`.

Nel listato 9.6 viene creata una variabile di tipo `string` che conterrà il nome utente per il login, e nel listato 9.7 viene definito il suo valore.

Per utilizzare una variabile basta richiamarla con `var.<NOME_VAR>`. Nel listato 9.3 è possibile vedere come vengono richiamate; ad esempio la variabile `login_username` è stata richiamata scrivendo `var.login_username`.

```
variable "login_username" {
  description = "Il nome utente per il login"
  type        = string
}
```

Listato 9.6: Sintassi del blocco `variable` nel file `variables.tf`.

```
login_username     = "admin"
```

Listato 9.7: Esempio di definizione di variabili di input nel file `admin_login.tfvars`.

Blocco `output`: il blocco output è utilizzato per definire delle variabili di tipo output, le quali servono per rendere disponibile nella riga di comando informazioni relative all'infrastruttura. I valori delle variabili output vengono visualizzati a provisioning terminato e successivamente è anche possibile ricavarlo inserendo il comando `terraform output <nome variabile output>`. Questo tipo di variabili vengono spesso utilizzati per poter conoscere i risultati di alcune operazioni che a priori non è possibile sapere, come ad esempio indici id, indirizzi IP, etc. Anche in questo caso possiede soltanto un etichetta, rappresentante il nome univoco della variabile all'interno del progetto. All'interno del blocco bisogna definire l'argomento `value`, che indica il valore che assumerà la variabile.

Nell'esempio sono state codificate nel file `output.tf` e nel listato 9.8 è possibile vedere la sintassi di una di queste variabili utilizzate. In questo caso la variabile d'output `alice` rappresenterà il valore dell'id della risorsa `openstack_identity_user_v3.user_alice` (precedentemente creata nel listato 9.4).

```
output "alice" {
  description = "L'id dell'utente alice"
  value       = openstack_identity_user_v3.user_alice.id
}
```

Listato 9.8: Sintassi del blocco `output` nel file `output.tf`.

Blocco `locals`: questo è l'ultimo blocco per definire le variabili, ed è utilizzato per poter creare delle variabili d'appoggio; infatti queste non verranno richieste in input e non appariranno in output. All'interno di questo blocco vengono elencate direttamente tutte le variabili che si voglio utilizzare e li si associa immediatamente un valore; è comunque possibile creare più blocchi locals anche all'interno dello stesso file `*.tf`. Il blocco locals non richiede alcuna etichetta per essere creato. Per accedere al valore di queste variabili bisogna usare la notazione `local.<nome variabile local>`.

Nel listato 9.9 viene mostrato un semplice esempio di creazione di una variabile locale, utilizzata poi nel listato 9.4.

```
locals {
  desc = "Utente creato con Terraform"
}
```

Listato 9.9: Sintassi del blocco `locals` nel file `users.tf`.

Blocco `module`: il blocco module è utilizzato per richiamare un modulo specifico e quindi per includere le sue risorse all'interno della configurazione.

Ripetendo il blocco module è possibile creare più copie delle risorse del modulo, senza la necessità di dover definire ogni volta tutte le risorse il quale modulo definisce. Grazie ai moduli è possibile riutilizzare le definizioni di svariate risorse in modo molto semplice.

Questo blocco possiede un etichetta che definisce il nome del modulo, ed è usata per identificare univocamente le variabili output che il modulo definisce internamente. Queste saranno richiamate utilizzando la nomenclatura `module.<nome modulo>.<nome variabile output>`.

Al sui interno è necessario indicare attraverso l'argomento `source` il percorso locale o remoto del modulo che si vuole utilizzare. Inoltre, ogni variabile di tipo input che il modulo utilizza va elencata come argomento di questo blocco.

Nel listato 9.10 è mostrato un esempio d'uso del modulo `create_user` presente nel progetto in figura 9.4. Il modulo crea delle risorse user (similmente come avviene nel listato 9.4) e per farlo richiede in input le variabili `name` e `password`.

```
module "user_bob" {
  source     = "./modules/create_user"
  name       = "bob"
  password   = "123"
  project_id = data.openstack_identity_project_v3.project_esempio.id
  role_id    = data.openstack_identity_role_v3.role_member.id
}
```

Listato 9.10: Sintassi del blocco `module` nel file `users.tf`.

Blocco `provisioner`: questo è un blocco particolare, e va inserito all'interno di un blocco resource. Tramite l'uso di questo blocco si ha la possibilità di eseguire delle azioni in locale o in remoto in determinate fasi di provisioning della risorsa. Per impostazione predefinita, i provisioner vengono eseguiti quando la risorsa viene creata, ma è possibile cambiare questo comportamento attraverso l'argomento `when = destroy`, che permette l'esecuzione durante l'eliminazione della risorsa.

Ha solo un etichetta, la quale può assumere solamente tre valori (e identifica il tipo di provisioner): `file`, `local-exe` e `remote-exe`.

Tramite il provisioner di tipo `file` è possibile copiare file o cartelle dalla macchina che sta eseguendo Terraform alla risorsa su cui si è effettuato il provisioning.

Con `local-exec` è possibile richiamare un eseguibile locale dopo il provisioning della risorsa, avviando un processo esterno a Terraform.

Infine con `remote-exec` è possibile eseguire uno script su una risorsa remota, una volta che è stato effettuato il provisioning su questa. Sono supportate sia connessioni *ssh* che *winrm*.

In questo caso è stato creato un esempio ad hoc, in quanto i blocchi provisioning aggiungono un alto livello di complessità ed entropia e sia nel progetto d'esempio che nel progetto di tesi non sono stati utilizzati. Nel listato 9.11 è mostrata la sintassi del blocco provisioning di tipo `local-exec`, integrata al blocco resource descritto nel listato 9.4. Durante l'eliminazione della risorsa, questo provisioner stamperà la scritta `Goodbye!` a riga di comando.

```
resource "openstack_identity_user_v3" "user_alice" {
  ...
  provisioner "local-exec" {
    when    = destroy
    command = "echo 'Goodbye!'"
  }
}
```

Listato 9.11: Sintassi del blocco innestato `provisioner`.

Tipi di valore degli argomenti Il risultato di ogni espressione è un valore e ha un tipo ben definito. Vengono definiti pochi ma più che sufficienti tipi di valore, e questi sono:

- `string`, una sequenza di caratteri Unicode per rappresentare del testo.
- `number`, rappresentano un valore numerico, sia interi che decimali.
- `bool`, servono per indicare i canonici valori booleani *true* o *false* e per poter creare espressioni condizionali.

Inoltre, sono presenti anche qualche tipo complesso come le collezioni:

- `list(<TYPE>)` rappresenta un elenco non ordinato di valori potenzialmente duplicati.
- `set(<TYPE>)` rappresenta una sequenza di valori ordinati e non duplicati.
- `tuple([<TYPE>, ...])` similmente alle list, con la differenza che il numero di elementi è definito a priori ed ognuno assume un tipo ben preciso.

Gli elementi di una list, set o tuple sono identificati da numeri interi consecutivi, a partire da zero, e si prelevano con la consueta nomenclatura a parentesi quadrate `COLLEZIONE[INDICE]`.

- `map(<TYPE>)` rappresenta un elenco di coppie di elementi, caratterizzate da chiave valore; le coppie sono identificate attraverso la chiave, e sono ordinate e non duplicate. Se il tipo non viene specificato, ogni valore sarà di tipo string.

- `object({<NOME_ATTR>=<TYPE>, ...})` definisce un tipo di dato complesso e viene gestito come se fosse una map con tanti tipi diversi. Come ci si può aspettare, gli attributi non sono duplicati e assumono un unico valore e anche se del tutto irrilevante, gli attributi vengono ordinati.

Gli elementi di una map sono identificati attraverso la chiave, e ci si riferisce a loro con con la nomenclatura a parentesi quadrate `MAPPA["CHIAVE"]`. Per quanto riguarda gli object invece, per accedere ai singoli attributi si usa la nomenclatura `OGGETTO.ATTRIBUTO`.

Infine esistono due tipi speciali di valore:

- `any` è utilizzato per rappresentare qualsiasi tipo di valore.

- `null` rappresenta l'assenza o l'omissione di valore; Terraform si comporta come se il valore fosse stato completamente omesso.

Funzionalità del linguaggio Come già anticipato, il linguaggio HCL non impone solamente delle regole sulla sintassi e, oltre a fornire dei costrutti base per descrivere l'infrastruttura, offre anche una serie di funzionalità tipiche della normale programmazione procedurale. Sono presenti i seguenti operatori.

- **Operatori aritmetici**, come la somma `+`, la sottrazione `-`, la moltiplicazione `*`, la divisione `/`, il modulo `%`.

- **Operatori di comparazione aritmetica**, come minore `<`, minore e uguale `<=`, maggiore `>`, maggiore uguale `>=`.

- **Operatori logici** (anche aritmetici), come l'uguaglianza a `==`, diseguaglianza a `!=`, la disgiunzione OR inclusiva `||`, la congiunzione AND `&&`, negazione `!`.

- **Operatore ternario** `?:`, utile per la restituzione dei valori a seconda del verificarsi o meno una determinata condizione.

Il linguaggio offre anche una vastità di funzioni integrate per la manipolazione dei dati, che si possono richiamare all'interno delle espressioni. Le funzioni vengono chiamate tramite la sintassi `<FUNZIONE>(<1°ARG>, <2°ARG>, ...)`.

Di seguito verranno raccolte alcune categorie con alcuni esempi di funzioni che si possono trovare.

- **Funzioni per i numeri.** Qui sono presenti le più classiche funzioni numeriche, tra cui `abs`, `ceil`, `floor`, `max`, `min`, etc.
- **Funzioni per le stringhe.** Varie funzioni per la manipolazione agevolata delle stringhe. Sono presenti `lower`, `regex`, `replace`, `split`, `substr`, `trimspace`, `upper` e molte altre.
- **Funzioni per le collezioni.** Tutte quelle funzioni utili per la gestione delle varie collezioni. Ad esempio `alltrue`, `coalesce`, `contain`, `flatten`, `length`, `merge`, `sum`, `zipmap`, e tante altre.
- **Funzioni per le operazioni sul filesystem.** Funzioni molto utili per le operazioni quali lettura e scrittura di file, come ad esempio `dirname`, `file`, `fileexist`, etc
- **Funzioni per la conversione di tipo.** Varie funzioni designate per la conversione del tipo di vari valori, tra qui `tolist`, `tomap`, `tonumber`, `tostring`, etc.

Finita questa breve ma indispensabile introduzione al linguaggio usato in Terraform, verrà mostrata la configurazione d'esempio post-installazione di OpenStack della sezione 7.3 completamente creata con Terraform.

9.3 Utilizzo di Terraform per il provisioning di risorse su OpenStack

In questo progetto di tesi si è provato ad effettuare con Terraform varie configurazioni di infrastrutture sul cloud OpenStack precedentemente messo in funzione. Si è cercato dunque di gestire tutti quegli aspetti che in generale un amministratore di sistema del cloud dovrà affrontare e parallelamente tutte quelle risorse che gli utenti base vorranno creare, a seconda dei privilegi che disporranno all'interno del cloud; tutto questo su configurazioni di infrastrutture ben separate e distinte. Per esempio l'amministratore di sistema dovrà verosimilmente gestire aspetti amministrativi e gerarchicamente più ad "alto

livello", come le reti per la comunicazione esterna, le immagini che si vogliono mettere a disposizione e i relativi flavor, gli utenti associati a progetti e domini, etc. Ognuno di questi aspetti potrà essere organizzato in una o più infrastrutture a seconda della complessità del sistema che si vorrà gestire e comunque a discrezione dell'amministratore D'altro canto un utente, a seconda dei privilegi che gli vengono messi a disposizione, vorrà creare macchine virtuali, reti interne e router che permettono la comunicazione tra la rete interna e l'eventuale rete esterna.

Un aspetto molto importante è che le configurazione dell'infrastruttura Terraform che l'amministratore e i vari utenti creeranno non saranno necessariamente collocate sulla stessa macchina. Per esempio l'amministratore di sistema potrebbe creare le infrastrutture su una macchina collegata alla stessa rete del cloud, mentre i vari utenti potranno creare le proprie infrastrutture sui propri computer personali, collegate magari su reti completamente differenti.

In questa sede, sono stati sviluppate dunque due configurazioni principali: la configurazione d'amministratore e la configurazione di un utente, rispecchianti rispettivamente gli esempi di configurazione trattati a post-installazione del cloud OpenStack nella sezione 7.3.

Come spiegato nella sezione 9.1.2, per poter interagire con OpenStack è necessario l'utilizzo di un plugin provider. Fortunatamente è presente nel Terraform Registry il plugin *openstack* cui i dettagli verranno trattati in seguito.

Versione di Terraform. Al momento della creazione delle due configurazioni, l'ultima versione di Terraform era la `v1.3.4`. Essendo un progetto molto attivo e in continuo sviluppo, non si assicura la completa compatibilità nelle configurazioni trattate in questa sede. Tuttavia, HashiCorp garantisce [44] la piena compatibilità in tutte le versioni `v1.x`.

Plugin provider openstack. Il plugin provider *openstack* [41] è necessario per poter interagire con le moltitudini di risorse che il cloud OpenStack offre attraverso le API. Il progetto di questo plugin è completamente open source ed è gestito e sviluppato dalla community, e non dunque da HashiCorp o da Canonical. Lo scopo del progetto è quello di riuscire in una completa interazione tra Terraform e il cloud OpenStack, quest'ultimo alla versione "vanilla", ovvero la versione che Canonical offre.

Il plugin non è esente da problemi, tuttavia il progetto è in continuo sviluppo e l'ultima versione, la `1.49.0` è stata rilasciata nell'Ottobre 2022; comunque nel corso di questo progetto di tesi non sono state rilevate problematiche nel suo utilizzo. Anche la documentazione risulta essere un po' scarna nel descrivere i dettagli dell'utilizzo di alcune risorse, malgrado ciò con un po' di tentativi si riesce ad utilizzarle con successo.

9.3.1 Provisioning di risorse da parte dell'amministratore

In questo scenario si presuppone che l'amministratore di sistema stia configurando l'infrastruttura da una macchina all'interno della rete del cloud e che abbia accesso diretto ad essa. Nel caso specifico di questa tesi, tale configurazione è stata effettuata all'interno del Raspberry Pi utilizzato per il sistema maas sezione 3.3.

Premessa: per cercare di compattare il codice in questo documento, le configurazioni riportate sono prive di alcuni costrutti quali variabili e moduli, e i valori degli argomenti vengono inseriti "nudi e crudi". Nella progettazione reale invece sono stati usati questi blocchi, in modo da avere una configurazione finale che sia modulare e facilmente manutenibile per il futuro.

main.tf e init del progetto. Come prima cosa, nel listato 9.12 viene creato il file `main.tf` con le configurazioni di Terraform e del provider *openstack*. Una volta salvato, è possibile inizializzare il progetto tramite il comando `terraform init`. A questo punto, Terraform scaricherà il plugin provider all'interno del workspace e restituirà l'esito dell'operazione a riga di comando.

```
terraform {
  required_version = ">= 1.3.4"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.49.0"
    }
  }
}

provider "openstack" {
  region      = "RegionOne"
  auth_url    = "https://10.0.0.7:5000/v3"
  cacert_file = "/home/pi/snap/openstackclients/common/root-ca.crt"

  user_name   = "admin"
  password    = "ejco3C6Wgxj2je9B"
  domain_name = "admin_domain"
  tenant_name = "admin"
}
```

Listato 9.12: File `main.tf` nella configurazione dell'amministratore di sistema.

- In `auth_url` va indicato l'indirizzo del servizio di autenticazione di OpenStack utilizzato, ovvero Keystone.
- In `cacert_file` va indicato il percorso del file certificato CA; se non si possiede, è possibile omettere quest'argomento ma in questo caso bisogna settare con il valore `true` l'argomento `insecure` come nel listato 9.3. Qui è stato utilizzato il certificato fornito da Vault.

Image e flavor. A questo punto, vengono configurati gli aspetti per dare la possibilità agli utenti nel creare le varie macchine virtuali. Quindi bisogna importare l'immagine e impostare un flavor rappresentate la macchina fisica da emulare. Per far ciò, viene utilizzata la risorsa `openstack_images_image_v2` per l'immagine, mentre per il flavor `openstack_compute_flavor_v2`. Nei seguenti listati 9.13 e 9.14 vengono mostrati le relative configurazioni nei file `images.tf` e `flavors.tf`.

```
resource "openstack_images_image_v2" "image" {
  name             = "jammy-amd64"
  visibility       = "public"
  local_file_path  = "/home/pi/client/cloud-images/jammy-amd64.img"
  disk_format      = "qcow2"
  container_format = "bare"
}
```

Listato 9.13: File `images.tf` nella configurazione dell'amministratore di sistema.

- Invece di indicare il percorso locale dell'immagine da utilizzare, è possibile indicare un URL da dove andare a scaricare l'immagine, tramite l'argomento `image_source_url`.

```
resource "openstack_compute_flavor_v2" "flavor" {
  name        = "m1.small"
  description = "Flavor creato tramite Terraform"
  is_public   = true
  vcpus       = 1
  ram         = 2048
  disk        = 20
  ephemeral   = 20
}
```

Listato 9.14: File `flavors.tf` nella configurazione dell'amministratore di sistema.

Networking. In questo paragrafo vengono mostrate le configurazioni per poter creare una rete esterna e la relativa subnet. A differenza delle configurazioni per i flavor e per le image, si è deciso di accorpare net e subnet in un unico file chiamato `networks+subnets.tf` ed è possibile vederne il codice nel listato 9.15; questo nell'ottica di creare una sola sottorete associata ad una rete. L'alta flessibilità di Terraform non vieta comunque lo scorporamento in futuro.

Per la creazione della rete esterna è stata utilizzata la risorsa di networking `openstack_networking_network_v2`, mentre per la sua sottorete è stata utilizzata `openstack_networking_subnet_v2`.

```
resource "openstack_networking_network_v2" "network" {
  name        = "ext_net"
  description = "Rete esterna creata tramite Terraform"

  external = true
  shared   = true
  segments {
    network_type     = "flat"
    physical_network = "physnet1"
  }
}

resource "openstack_networking_subnet_v2" "subnet" {
  name        = "ext_subnet"
  description = "Subnet esterna creata tramite Terraform"

  network_id  = openstack_networking_network_v2.network.id
  enable_dhcp = false

  cidr       = "10.0.0.0/24"
  gateway_ip = "10.0.0.1"
  allocation_pool {
    start = "10.0.0.140"
    end   = "10.0.0.239"
  }
}
```

Listato 9.15: File `networks+subnets.tf` nella configurazione dell'amministratore di sistema.

- In `network_id` deve essere indicato l'id della risorsa adibita per la rete; la nomenclatura utilizzata per il riferimento a quella risorsa è spiegato nei dettagli nel paragrafo Blocco resource della sezione 9.2.3.

Domain e project. Il provider utilizzato fornisce un'unica risorsa per la creazione di domini e progetti, ed è `openstack_identity_project_v3`; la tipologia viene comunque distinta attraverso l'argomento booleano `is_domain`.

Sia per il domain che per il projetto è stato dedicato un file a parte, rispettivamente `domains.tf` e `projects.tf`, con l'ottica di poter creare più domini e più progetti per gli utenti. Il contenuto dei due file è stato raggruppato nel listato 9.16.

```
# file domains.tf
resource "openstack_identity_project_v3" "domain" {
  name        = "domain1"
  description = "Domain creato tramite Terraform"
  enabled     = true
  is_domain   = true
}

# file projects.tf
resource "openstack_identity_project_v3" "project" {
  name        = "project1"
  description = "Project creato tramite Terraform"
  enabled     = true
  domain_id   = openstack_identity_project_v3.domain.id
}
```

Listato 9.16: File `domains.tf` e `projects.tf` nella configurazione dell'amministratore di sistema.

- Con l'argomento `enable` è possibile abilitare o disabilitare un domain o un progetto; questo perché, se si volesse cancellare un domain, OpenStack richiede che prima venga disabilitato. Inoltre risulta essere molto utile poter disabilitare un domain o un progetto senza doverlo cancellare, potendo mantenere così le eventuali risorse ad essi associate.

User e role. Per finire, l'amministratore dovrà certamente creare i vari utenti, associando ad ognuno di essi un ruolo. Per far ciò si utilizzano le risorse `openstack_identity_user_v3` per l'utente e per l'associazione *utente-ruolo-progetto* si usa `openstack_identity_role_assignment_v3`. Tramite il blocco `data` e la risorsa `openstack_identity_role_v3`, si importa il ruolo *Member*.

Tutto ciò è stato inserito all'interno di un unico file `users.tf`, ed è possibile visionarne il codine nel listato 9.17.

```
data "openstack_identity_role_v3" "role_member" {
  name = "Member"
}

resource "openstack_identity_user_v3" "user" {
  name        = "user1"
  password    = "ubuntu"
  description = "User creato tramite Terraform"

  domain_id = openstack_identity_project_v3.domain.id
  enabled   = true

  lifecycle {
    ignore_changes = [
      password,
      description,
      extra,
    ]
  }
}

resource "openstack_identity_role_assignment_v3" "user1_role_project1" {
  user_id    = openstack_identity_user_v3.user.id
  project_id = openstack_identity_project_v3.project.id
  role_id    = data.openstack_identity_role_v3.role_member.id
}
```

Listato 9.17: File `users.tf` nella configurazione dell'amministratore di sistema.

- Con il meta-argomento `ignore_change` del meta-blocco `lifecycle` vengono specificati quali argomenti Terraform non deve considerare durante i possibili futuri re-provisioning. Questo perché l'utente potrebbe cambiare per esempio la propria descrizione o indirizzo e-mail al di fuori ovviamente di questa configurazione. Quindi in un futuro apply, Terraform si accorgerebbe di questa discrepanza ed effettuerà un reprovisioning della risorsa, andando a sovrascrivere quelle impostazioni che l'utente si è configurato da se.

9.3.2 Provisioning di risorse da parte dell'utente

In uno scenario realistico, gli utenti (o clienti) possono creare le configurazioni di infrastrutture nelle più disparate locazioni possibili, che siano queste macchine di un ufficio collegato direttamente alla stessa rete del cloud oppure computer personali in abitazioni private, e quindi collegate su reti diverse.

Nello scenario di questa tesi, le configurazioni dell'utente sono state effettuate su un computer personale, situato su una rete differente da quella del cloud. Ciò nonostante, grazie all'accesso diretto in ssh con la macchina maas, è stato possibile interagire con il cloud anche se collocato su una rete privata.

Premessa: per cercare di compattare il codice in questo documento, le configurazioni riportate sono prive di alcuni costrutti quali variabili e moduli, e i valori degli argomenti vengono inseriti "nudi e crudi". Nella progettazione reale invece sono stati usati questi blocchi, in modo da avere una configurazione finale che sia modulare e facilmente manutenibile per il futuro. In appendice è possibile visionare il codice completo.

main.tf e init del progetto. Anche in questo caso, come prima cosa viene creato il file `main.tf` con le configurazioni di Terraform e del provider *openstack*. In questo caso però, le configurazioni del provider differiscono leggermente; bisogna specificare i vari endpoint dei servizi del cloud che si desiderano utilizzare, in quanto non situati all'interno della stessa rete.

Nel listato 9.18 è presente la configurazione utilizzata nel file `main.tf`; come di consueto una volta salvato, è possibile inizializzare il progetto tramite il comando `terraform init`.

```
terraform {
  required_version = ">= 1.3.4"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.49.0"
}}}
provider "openstack" {
  region   = "RegionOne"
  auth_url = "https://localhost:5000/v3"
  insecure = true
  user_name   = "user1"
  password    = "ubuntu"
  domain_name = "domain1"
  tenant_name = "project1"
  endpoint_overrides = {
    compute  = "https://localhost:8774/v2.1/"
    identity = "https://localhost:5000/v3/"
    image    = "https://localhost:9292/v2/"
    network  = "https://localhost:9696/v2.0/"
  }
}
```

Listato 9.18: File `main.tf` nella configurazione dell'utente.

- Nell'esempio sono riportati indirizzi *localhost*, in quanto sono state inseriti i vari re-indirizzamenti nel file config di ssh nella configurazione della connessione con maas. Per esempio, per l'argomento `auth_url` è stata aggiunta la riga `LocalForward 5000 10.0.0.7:5000`, mentre per il servizio compute la riga `LocalForward 8774 10.0.0.21:8774`.

- L'argomento `endpoint_overrides` è una mappa di stringhe, e serve per indicare gli endpoint dei servizi del cloud OpenStack che si andranno ad utilizzare. Ogni indirizzo inserito deve terminare con il carattere `/`, spesso preceduto dalla versione.

  Nella documentazione [42] è possibile verificare quali servizi il provider supporta, mentre l'elenco dei service endpoint con i relativi indirizzi è possibile visionarlo collegandosi alla dashboard di OpenStack.

Networking. In questo paragrafo vengono mostrate le configurazioni per poter creare una rete privata, la relativa subnet e un router virtuale in grado di collegare la rete privata con la rete esterna creata dall'amministratore.

La configurazione riguardante la rete è stata creata all'interno del file `internal_networking.tf` e come si può vedere nel listato 9.19 non differisce molto dalla configurazione della rete esterna creata dall'amministratore del listato 9.15.

```
resource "openstack_networking_network_v2" "network" {
  name        = "user1_net"
  description = "Rete interna dello user1 creata tramite Terraform"
  external = false
  shared   = false
}

resource "openstack_networking_subnet_v2" "subnet" {
  name        = "user1_subnet"
  description = "Subnet interna dello user1 creata tramite Terraform"
  network_id  = openstack_networking_network_v2.network.id
  enable_dhcp = true
  cidr       = "192.168.0.0/24"
  gateway_ip = "192.168.0.1"
  allocation_pool {
    start = "192.168.0.10"
    end   = "192.168.0.20"
  }
  dns_nameservers = ["10.0.0.2"]
}
```

Listato 9.19: File `internal_networking.tf` nella configurazione dell'utente.

Invece la configurazione per il router è stata inserita nel file `router.tf`. Come mostrato nel listato 9.20, per il router è stata utilizzata la risorsa `openstack_networking_router_v2"` mentre è stata utilizzata la risorsa `openstack_networking_router_interface_v2` per poter creare un'interfaccia di rete nel router.

```
data "openstack_networking_network_v2" "ext_net" {
  name = "ext_net"
}

resource "openstack_networking_router_v2" "router" {
  name                = "user1_router"
  description         = "Router dello user1 creato tramite Terraform"
  external_network_id = data.openstack_networking_network_v2.ext_net.id
}

resource "openstack_networking_router_interface_v2" "name" {
  router_id = openstack_networking_router_v2.router.id
  subnet_id = openstack_networking_subnet_v2.subnet.id
}
```

Listato 9.20: File `router.tf` nella configurazione dell'utente.

Import delle chiavi pubbliche ssh. Per importare le chiavi pubbliche ssh, il provider mette a disposizione la risorsa `openstack_compute_keypair_v2` e il suo uso è mostrato nel listato 9.21. Per questo scopo è stato riservato il file `keypair.tf`, così se l'utente volesse aggiungere altre chiavi, può elencarle all'interno del file dedicato.

```
data "openstack_identity_auth_scope_v3" "user" {
  name = "user1"
}

resource "openstack_compute_keypair_v2" "user_keypairs" {
  name       = "user1-PC"
  public_key = file("../public_keys_ssh/id_rsa.pub")

  user_id = data.openstack_identity_auth_scope_v3.user.user_id
}
```

Listato 9.21: File `keypair.tf` nella configurazione dell'utente.

- L'argomento `public_key` è di tipo string e accetta la chiave per esteso. Nell'esempio è stata utilizzata la funzione `file("path")`, che restituisce l'intero contenuto del file nel percorso passato come argomento.

Regole per la connessione ssh. Per poter accedere all'istanza, l'utente deve anche creare il security group con i valori di connessione del protocollo ssh. Quindi per poterli creare, vengono messe a disposizione le risorse `openstack_networking_secgroup_v2` per creare il gruppo e per creare la regola `openstack_networking_secgroup_rule_v2`. Nel listato 9.22 è mostrato il contenuto del file `security_group.tf` usato per questo scopo.

```
resource "openstack_networking_secgroup_v2" "security_group" {
  name        = "Allow_SSH"
  description = "Allow SSH creato da Terraform"
}

resource "openstack_networking_secgroup_rule_v2" "secgroup_rule" {
  direction        = "ingress"
  ethertype        = "IPv4"
  protocol         = "tcp"
  port_range_min   = 22
  port_range_max   = 22
  remote_ip_prefix = "0.0.0.0/0"

  security_group_id = openstack_networking_secgroup_v2.security_group.id
}
```

Listato 9.22: File `security_group.tf` nella configurazione dell'utente.

Creazione di un'istanza di macchina virtuale. Infine l'utente creerà delle macchine virtuali. Anche in questo caso è stato creato un unico file, `instance.tf`, rappresentante l'istanza della macchina virtuale. Per migliorarne la consultazione, il codice è stato suddiviso i due listati. Nel listato 9.23 viene mostrata la configurazione per della macchina virtuale, ed è stata utilizzata la risorsa `openstack_compute_instance_v2`. Nel listato 9.24 invece, viene mostrata la configurazione per poter associare un indirizzo IP esterno alla vm creata, utilizzando le risorse `openstack_networking_floatingip_v2` e `openstack_compute_floatingip_associate_v2`. Sono state utilizzate anche due variabili output per poter visionare a provisioning terminato gli indirizzi IP pubblici e privati della macchia virtuale.

```
data "openstack_images_image_v2" "jammy-amd64" {
  name = "jammy-amd64"
}

data "openstack_compute_flavor_v2" "m1_small" {
  name = "m1.small"
}
```

```
8
9  resource "openstack_compute_instance_v2" "instance" {
10   name = "jammy-user1-terraform"
11
12   image_id  = data.openstack_images_image_v2.jammy-amd64.id
13   flavor_id = data.openstack_compute_flavor_v2.m1_small.id
14
15   key_pair        = openstack_compute_keypair_v2.user_keypairs.name
16   security_groups = [openstack_networking_secgroup_v2.security_group.name]
17
18   network {
19     name = openstack_networking_network_v2.network.name
20   }
21 }
```

Listato 9.23: Prima parte del file `instance.tf` nella configurazione dell'utente.

- L'argomento `security_groups` accetta un elenco di stringhe, quindi è stato necessario inserire i nomi dei gruppi di sicurezza all'interno delle parentesi quadre `[ ]`.

```
1  resource "openstack_networking_floatingip_v2" "fip" {
2    pool = data.openstack_networking_network_v2.ext_net.name
3  }
4
5  resource "openstack_compute_floatingip_associate_v2" "fip_associate" {
6    floating_ip = openstack_networking_floatingip_v2.fip.address
7    instance_id = openstack_compute_instance_v2.instance.id
8  }
9
10 output "private_IP" {
11   value = openstack_compute_instance_v2.instance.access_ip_v4
12 }
13
14 output "public_IP" {
15   value = openstack_networking_floatingip_v2.fip.address
16 }
```

Listato 9.24: Seconda parte del file `instance.tf` nella configurazione dell'utente.

9.3.3 Considerazioni finali sulla sicurezza dei dati.

Aspetti riguardanti la sicurezza dei dati, ad esempio gli username e le password, non sono stati trattati in questo progetto. Bisogna tenere in considerazione che Terraform salva in chiaro questi dati, sia nei singoli file di configurazione sia nel file rappresentante lo stato dell'infrastruttura. Se si utilizzano variabili, il solo contrassegnare il loro valore sensibile con l'argomento booleano `sensitive` non è sufficiente, in quanto questa opzione disabilita l'output di questi valori a riga di comando, ma il loro valore viene comunque salvato nei suddetti file.

Per quanto riguarda la password utente tuttavia, un'accortezza è stata comunque presa, ovvero è stata abilitata l'opzione che al primo accesso effettuato l'utente è obbligato a cambiare la propria password prima di poter proseguire. Per fare ciò, è stato sufficiente aggiungere un'impostazione nella configurazione di Keyston. Siccome tutte le applicazioni del cloud sono gestite da Juju, anche in questo caso si è utilizzato questo strumento per aggiungere tale modifica.

```
1 juju config keystone password-security-compliance="
    change_password_upon_first_use: true"
```

Listato 9.25: Abilitazione del cambio password al primo accesso.

Capitolo 10

Conclusioni e sviluppi futuri

In definitiva, si può affermare di aver superato tutti gli obiettivi pianificati, riuscendo nell'installazione di un cloud privato OpenStack perfettamente funzionante e nell'interazione con esso tramite Terraform.

In primo luogo è stato progettato e pianificato il cluster con tutte le sue componenti, sia software che hardware. Partendo dai requisiti che i software necessitavano, si è potuto dimensionare tutti gli apparati hardware in maniera adeguata. Quindi per il cluster si sono utilizzati sei computer desktop del 2014, di fascia media per quegli anni , con architettura AMD64. La scelta del dispositivo per la gestione e il coordinamento del cluster, ovvero quello che ospita il sistema MAAS, si è ripiegata sul SBC Raspberry Pi 4, in quanto anche se possiede un'architettura ARM (e quindi differente rispetto le macchine del cluster) ha caratteristiche hardware più che sufficienti per il suo scopo. Infine, si è utilizzato un altro Raspberry Pi 4 con l'aggiunta di un interfaccia di rete USB, a cui si è dato il compito esclusivo di comportarsi da router tra la rete esterna e la rete del cluster. Tutte le otto macchine sono state collegate tra loro attraverso un comune switch da otto porte.

Per quanto riguarda il deploy del cluster e del cloud, sia MAAS che Juju offrono un ottimo livello di astrazione e di automatismo. MAAS si è rilevato un potente strumento per la gestione delle macchine e per il deploy del sistema operativo in esse; il tutto limitando notevolmente l'intervento umano. Anche Juju si è rilevato un valido strumento per il deploy automatizzato delle applicazioni che compongono il cloud. Tuttavia, a differenza di MAAS, sono sorti alcuni aspetti negativi da tenere in considerazione.

- La comprensione dei comandi e del suo utilizzo ha richiesto non poco tempo, specialmente per i non "addetti ai lavori".

- La personalizzazione di ogni singolo componente software non è semplice e immediata, complice anche il fatto che si tratti di uno strumento di

automatizzazione e che quindi tende a nascondere alcuni aspetti.

- L'individuazione e la soluzione dei problemi durante il deploy delle applicazioni si è rilevata alquanto ostica; sebbene offra strumenti di lettura dei log, come `juju debug-log --include <NOME_UNIT>`, i messaggi d'errore che riporta sono spesso fuorvianti o non di facile comprensione e per quanto possa sembrare paradossale è capitato molto spesso di risolvere i vari problemi riavviando semplicemente i singoli container o le intere macchine.

Infine, una volta installato OpenStack, si è studiato lo strumento Terraform per la creazione di IaC, infrastrutture attraverso il codice. Questo ha permesso in primo luogo di poter inizializzare e creare l'ambiente cloud da parte dell'amministratore di sistema, gestendo aspetti come la creazione degli utenti, dei progetti delle immagini delle future VM, e in secondo luogo la creazione di macchine virtuali da parte degli utenti. Terraform si è dunque rilevato un ottimo strumento in questo aspetto, e grazie ad esso è possibile automatizzare le parti tipicamente manuali di creazione e configurazione di risorse, offrendo al tempo stesso una rapida riusabilità delle infrastrutture create.

10.1 Sviluppi futuri

Un aspetto molto importante da tenere in considerazione in un' ipotetica implementazione futura di OpenStack sarà quella di verificare che tutta la procedura descritta in questo documento sia valida anche per release di OpenStack successive a Yoga; infatti, come descritto nella sezione 7.1.2, OpenStack è in continuo sviluppo e dall'inizio dei lavori sono uscite e sono state programmate diverse versioni. Specificando un esempio concreto, per la release successiva a Yoga, ovvero Zed, inizialmente non era stata rilasciata la guida per la tipologia di deployment affrontata in questo documento, lasciando intendere una cessazione del supporto a tale metodologia. Solo svariati mesi dopo, a pochi giorni dal completamente della stesura di questo documento, è stata rilasciata la suddetta guida sul deployment in charm [28], mantenendo così il supporto per questa tipologia.

Uno dei punti deboli del sistema di deployment affrontato, è quello dell'essere dipendenti dalle versioni dei charm che accompagnano le versioni delle applicazioni, e questo in un futuro potrebbe compromettere il parziale uso di alcune funzionalità a causa di possibili mancanze di release aggiornate degli applicativi.

Un altro aspetto che è stato notato nell'implementazione e utilizzo di OpenStack è relativa a un'anomala lentezza durante sia l'installazione dei charm

nelle macchine, sia l'uso della dashboard di OpenStack, in particolar modo nelle fasi di log-in. Non si è ritenuto fosse un problema ne di rete, ne di capacità computazionale delle CPU o della quantità di RAM, poiché sono state effettuate dei monitoraggi durante le fasi di installazione e queste componenti non venivano saturate. Non essendo parte del core del progetto, le indagini a tale anomalia si sono fermate qui. Tuttavia, si è speculato sul fatto che si sono utilizzati dei vecchi HDD come dispositivi di storage e che quindi questi potrebbero avere settori danneggiati che ritardano l'I/O delle varie operazioni, o più semplicemente essendo di fascia medio/bassa possano avere una ridotta velocità di lettura/scrittura, diminuendo così le varie prestazioni dell'intero sistema. Si consiglia dunque per una prossima implementazione, di verificare l'integrità e la velocità dei vari dispositivi di storage, optando eventualmente la scelta su SSD piuttosto che su HDD.

Per finire, un aspetto che non è stato affrontato riguarda la gestione e la sicurezza dei dati sensibili degli utenti da parte dell'amministratore di sistema, in particolar modo su Terraform. Come accennato nella sezione 9.3.3, su Terraform ogni configurazione viene salvata in chiaro e l'amministratore conosce questi valori; Ad esempio, l'amministratore crea gli utenti assegnando un nome ed una password che conoscerà a priori, e quest'ultima viene archiviata e salvata in chiaro. L'unico accorgimento che è stato preso, è stato quello di impostare a Keystone che quando l'utente effettua il log-in la prima volta, è obbligato prima di procedere al cambiamento della password. In questo modo, l'amministratore non sarà più in grado di poter accedere con quell'utente. Un aspetto interessante da approfondire è dunque verificare che tipo di supporto possa offrire sia OpenStack, partendo sicuramente da una miglior configurazione di Keystone [18], che Terraform o qualche suo plugin provider annesso, riguardanti la crittazione dei dati o generazione casuale di valori come appunto le password.

Ringraziamenti

In questo capitolo per ovvie ragioni, abbandono la forma impersonale e passo alla prima persona, per poter parlare in modo più colloquiale e scherzoso. Tengo a precisare che i ringraziamenti non sono ordinati per importanza o per altri bizzarri parametri, ma semplicemente partono dalla mia carriera universitaria, ai miei famigliari e parenti, per poi passare ai miei amici e conoscenti.

Parto col ringraziare il mio relatore, Vittorio Ghini, che in questi mesi mi ha aiutato durante lo svolgimento del progetto, fornendomi conoscenze e supporto fondamentali per la sua riuscita, oltre che un continuo incoraggiamento.
Ringrazio il mio compagno di tesi Matteo Bambini, senza le sue conoscenze in materia di reti, server e cloud avrei sicuramente incontrato non poche difficoltà nello svolgimento di questo progetto.
Un ringraziamento va anche a tutte le altre professoresse e professori che mi han sostenuto e aiutato durante questo percorso (mentre a quelli che mi hanno bellamente ignorato faccio a loro una bella pernacchia).

Ringrazio la mia famiglia, a partire dai miei genitori che, nonostante tutte le difficoltà, mi sostengono e sopportano i miei continui sbalzi d'umore.
Un ringraziamento speciale va a mio fratello, compagno di giochi, che non solo mi ha sostenuto moralmente, ma ha anche contribuito al mio sostentamento economico.
Ringrazio tutti i miei parenti, anche quelli estesi, che sono stati al mio fianco durante questo periodo, compresi quelli che purtroppo in questi anni ci hanno lasciato.

Ringrazio tutte le mie amiche e i miei amici, passati e presenti, che fino ad oggi hanno condiviso con me esperienze, risate, abbracci e dolori, in particolare coloro che, nonostante il mio caratteraccio, continuano a starmi vicino.

Infine, per cercare di non dimenticare nessuno, ringrazio tutte le altre persone che mi hanno aiutato in qualunque modo, compresi coloro che mi hanno regalato una risata o una semplice carezza, anche quelle che purtroppo non leggeranno mai queste parole.

Grazie di cuore <3

Bibliografia

[1] Documentazione Ceph, https://docs.ceph.com/en/quincy/, ultimo accesso 23 Gennaio 2023.

[2] Charmhub - The Open Operator Collection, https://charmhub.io/, ultimo accesso 26 Gennaio 2023.

[3] Che cos'è il bursting nel cloud, https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-is-cloud-bursting, ultimo accesso 31 Gennaion 2023.

[4] Fully Qualified Domain Name, FQDN, https://www.rfc-editor.org/rfc/rfc4703, ultimo accesso 13 Gennaio 2023.

[5] Juju, the Operator Lifecycle Manager, https://juju.is/, ultimo accesso 24 Gennaio 2023.

[6] Juju: Bundle, https://juju.is/docs/sdk/charm-bundles, ultimo accesso 01 Febbraio 2023.

[7] Juju: Charmed operator, https://juju.is/docs/olm/charmed-operators, ultimo accesso 26 Gennaio 2023.

[8] Juju: Client, https://juju.is/docs/olm/the-juju-client, ultimo accesso 25 Gennaio 2023.

[9] Juju: Cloud(substrate), https://juju.is/docs/olm/cloud, ultimo accesso 25 Gennaio 2023.

[10] Juju: Controller, https://juju.is/docs/olm/controller, ultimo accesso 26 Gennaio 2023.

[11] How to install Juju client, https://juju.is/docs/olm/install-juju, ultimo accesso 27 Gennaio 2023.

[12] Deploy OpenStack: install Juju, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/install-juju.html, ultimo accesso 27 Gennaio 2023.

[13] Juju: Model, https://juju.is/docs/olm/model, ultimo accesso 25 Gennaio 2023.

[14] Juju: Charmed Operator Lifecycle Manager, https://juju.is/docs/olm, ultimo accesso 26 Gennaio 2023.

[15] Juju: Integration, https://juju.is/docs/olm/integration, ultimo accesso 01 Febbraio 2023.

[16] Juju: Unit, https://juju.is/docs/olm/unit, ultimo accesso 01 Febbraio 2023.

[17] How to use MAAS with Juju, https://juju.is/docs/olm/maas, ultimo accesso 27 Gennaio 2023.

[18] Keystone: Security compliance, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/keystone.html, ultimo accesso 01 Marzo 2023.

[19] Errore nell'inizializzazione automatica di LXD, https://discuss.linuxcontainers.org/t/2148, ultimo accesso 31 Gennaion 2023.

[20] Metal as a Service, MAAS, https://maas.io/, ultimo accesso 20 Dicembre 2022.

[21] MAAS glossary, https://maas.io/docs/maas-glossary, ultimo accesso 23 Gennaio 2023.

[22] How MAAS works, https://maas.io/how-it-works, ultimo accesso 23 Gennaio 2023.

[23] How to Install MAAS, https://maas.io/docs/how-to-install-maas, ultimo accesso 07 Gennaio 2023.

[24] Deploy OpenStack: install MAAS, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/install-maas.html, ultimo accesso 07 Gennaio 2023.

[25] Power management reference on MAAS, https://maas.io/docs/power-management-reference, ultimo accesso 16 Gennaio 2023.

[26] MAAS installation requirements, https://maas.io/docs/maas-installation-requirements, ultimo accesso 04 Gennaio 2023.

[27] Deployment di OpenStack Yoga con i charm, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/, ultimo accesso 28 Febbraio 2023.

[28] Deployment di OpenStack Zed con i charm, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/zed/, ultimo accesso 28 Febbraio 2023.

[29] Tipi di charm di OpenStack, https://docs.openstack.org/charm-guide/latest/concepts/index.html, ultimo accesso 25 Febbraio 2023.

[30] Guida di installazione di OpenStack con Juju, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/configure-openstack.html, ultimo accesso 23 Febbraio 2023.

[31] Guide al deployment di OpenStack, https://docs.openstack.org/yoga/deploy/index.html, ultimo accesso 21 Gennaio 2023.

[32] Guida di installazione di OpenStack con Juju, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/install-openstack.html, ultimo accesso 22 Gennaio 2023.

[33] Open Virtual Network, https://www.ovn.org/en/, ultimo accesso 23 Gennaio 2023.

[34] Open vSwitch, https://www.openvswitch.org, ultimo accesso 23 Gennaio 2023.

[35] Confronto tra object storage, file storage e block storage, https://www.purestorage.com/it/knowledge/what-is-object-storage.html, ultimo accesso 23 Gennaio 2023.

[36] Raspberry Pi OS, https://www.raspberrypi.com/software/, ultimo accesso 06 Gennaio 2023.

[37] Raspberry Pi 4 Model B Specifications, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, ultimo accesso 06 Gennaio 2023.

[38] Terraform: configurazione backend, https://developer.hashicorp.com/terraform/language/settings/backends/configuration, ultimo accesso 22 Febbraio 2023.

[39] Perform CRUD Operations with Providers, https://developer.hashicorp.com/terraform/tutorials/configuration-language/provider-use#terraform-plugins, ultimo accesso 23 Febbraio 2023.

[40] What Is Terraform, https://developer.hashicorp.com/terraform/intro, ultimo accesso 23 Febbraio 2023.

[41] Terraform Provider Openstack, https://registry.terraform.io/providers/terraform-provider-openstack/openstack/latest, ultimo accesso 15 Febbraio 2023.

[42] Terraform Provider Openstack - Documentation, https://registry.terraform.io/providers/terraform-provider-openstack/openstack/latest/docs, ultimo accesso 15 Febbraio 2023.

[43] Terraform: comando validate, https://developer.hashicorp.com/terraform/cli/commands/validate, ultimo accesso 22 Febbraio 2023.

[44] Terraform v1.x Compatibility Promises, https://developer.hashicorp.com/terraform/language/v1-compatibility-promises, ultimo accesso 22 Febbraio 2023.

[45] Documentazione Vault, https://developer.hashicorp.com/vault/docs, ultimo accesso 23 Gennaio 2023.

[46] Vault, https://developer.hashicorp.com/vault/docs/what-is-vault, ultimo accesso 23 Gennaio 2023.

[47] Understanding VLAN Trunking Protocol (VTP), https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/vtp.html#wp1049555, ultimo accesso 18 Gennaio 2023.

Appendice A

Listati dei comandi

A.1 Installazione di MAAS

Di seguito sono riportati i comandi per l'installazione completa di MAAS.

```
sudo snap install maas --channel=3.1/stable
sudo snap install maas-test-db
sudo maas init region+rack --maas-url http://10.0.0.2:5240/MAAS --database-
    uri maas-test-db:///
sudo maas createadmin --username admin --password ubuntu --email
    admin@example.com --ssh-import lp:<usernameLaunchpad>
sudo maas apikey --username admin > ~/admin-api-key-file
```

Listato A.1: Installazione di MAAS.

A.2 Installazione di Juju

Di seguito sono riportati i comandi per la corretta installazione di Juju e dell'inizializzazione del cloud.

```
sudo snap install juju --classic
juju add-cloud --client -f maas-cloud.yaml maas-one
juju add-credential --client -f maas-creds.yaml maas-one
juju bootstrap --bootstrap-series=focal --constraints "tags=juju arch=amd64
    " maas-one maas-controller
juju add-model --config default-series=jammy openstack
```

Listato A.2: Installazione di Juju e inizializzazione del cloud.

A.3 Installazione semi-automatizzata di OpenStack

Di seguito è riportato l'elenco completo dei comandi eseguiti per il deploy di OpenStack attraverso Juju. In rosso sono evidenziati quei valori che necessitano di essere modificati.

```
# posizionamento sul controller "maas-controller" e sul model "openstack"
juju switch maas-controller:openstack

# ceph-osd
juju deploy -n 4 --series jammy --channel quincy/stable --config ceph-osd.
    yaml --constraints tags=compute ceph-osd

# nova-compute
juju deploy -n 3 --to 1,2,3 --series jammy --channel yoga/stable --config
    nova-compute.yaml nova-compute

# mysql-innodb-cluster
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --series jammy --channel 8.0/stable
     mysql-innodb-cluster

# vault
juju deploy --to lxd:3 --series jammy --channel 1.7/stable vault
juju deploy --channel 8.0/stable mysql-router vault-mysql-router
juju add-relation vault-mysql-router:db-router mysql-innodb-cluster:db-
    router
juju add-relation vault-mysql-router:shared-db vault:shared-db

####
# configurazione di vault
sudo snap install vault

# IP vault
IP_VAULT=$(juju status --format=yaml vault | grep public-address | awk '{
    print $2}' | head -1)
export VAULT_ADDR="http://${IP_VAULT}:8200"

# generazione 5 chiavi
vault operator init -key-shares=5 -key-threshold=3 > vault-keys

# apertura vault
vault operator unseal <key1>
vault operator unseal <key2>
vault operator unseal <key3>

# autorizzazione charm
export VAULT_TOKEN=<Initial Root Token>
```

```
vault token create -ttl=10m
juju run-action --wait vault/leader authorize-charm token=<token>

# generazione certificato autofirmato
juju run-action --wait vault/leader generate-root-ca > vault-ca.crt

# aggiunta al DB del certificato autofirmato
juju add-relation mysql-innodb-cluster:certificates vault:certificates

# neutron-api + neutron-api-plugin-ovn (subordinate) --- ovn-central + ovn-
    chassis (subordinate)
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --series jammy --channel 22.03/
    stable --config neutron.yaml ovn-central
juju deploy --to lxd:1 --series jammy --channel yoga/stable --config
    neutron.yaml neutron-api
juju deploy --channel yoga/stable neutron-api-plugin-ovn
juju deploy --channel 22.03/stable --config neutron.yaml ovn-chassis
juju add-relation neutron-api-plugin-ovn:neutron-plugin neutron-api:neutron
    -plugin-api-subordinate
juju add-relation neutron-api-plugin-ovn:ovsdb-cms ovn-central:ovsdb-cms
juju add-relation ovn-chassis:ovsdb ovn-central:ovsdb
juju add-relation ovn-chassis:nova-compute nova-compute:neutron-plugin
juju add-relation neutron-api:certificates vault:certificates
juju add-relation neutron-api-plugin-ovn:certificates vault:certificates
juju add-relation ovn-central:certificates vault:certificates
juju add-relation ovn-chassis:certificates vault:certificates
juju deploy --channel 8.0/stable mysql-router neutron-api-mysql-router
juju add-relation neutron-api-mysql-router:db-router mysql-innodb-cluster:
    db-router
juju add-relation neutron-api-mysql-router:shared-db neutron-api:shared-db

# keystone
juju deploy --to lxd:0 --series jammy --channel yoga/stable keystone
juju deploy --channel 8.0/stable mysql-router keystone-mysql-router
juju add-relation keystone-mysql-router:db-router mysql-innodb-cluster:db-
    router
juju add-relation keystone-mysql-router:shared-db keystone:shared-db
juju add-relation keystone:identity-service neutron-api:identity-service
juju add-relation keystone:certificates vault:certificates

# rabbitmq-server
juju deploy --to lxd:2 --series jammy --channel 3.9/stable rabbitmq-server
juju add-relation rabbitmq-server:amqp neutron-api:amqp
juju add-relation rabbitmq-server:amqp nova-compute:amqp

# nova-cloud-controller
juju deploy --to lxd:3 --series jammy --channel yoga/stable --config ncc.
    yaml nova-cloud-controller
juju deploy --channel 8.0/stable mysql-router ncc-mysql-router
```

```
juju add-relation ncc-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation ncc-mysql-router:shared-db nova-cloud-controller:shared-
    db
juju add-relation nova-cloud-controller:identity-service keystone:identity-
    service
juju add-relation nova-cloud-controller:amqp rabbitmq-server:amqp
juju add-relation nova-cloud-controller:neutron-api neutron-api:neutron-api
juju add-relation nova-cloud-controller:cloud-compute nova-compute:cloud-
    compute
juju add-relation nova-cloud-controller:certificates vault:certificates

# placement
juju deploy --to lxd:3 --series jammy --channel yoga/stable placement
juju deploy --channel 8.0/stable mysql-router placement-mysql-router
juju add-relation placement-mysql-router:db-router mysql-innodb-cluster:db-
    router
juju add-relation placement-mysql-router:shared-db placement:shared-db
juju add-relation placement:identity-service keystone:identity-service
juju add-relation placement:placement nova-cloud-controller:placement
juju add-relation placement:certificates vault:certificates

# openstack-dashboard
juju deploy --to lxd:2 --series jammy --channel yoga/stable openstack-
    dashboard
juju deploy --channel 8.0/stable mysql-router dashboard-mysql-router
juju add-relation dashboard-mysql-router:db-router mysql-innodb-cluster:db-
    router
juju add-relation dashboard-mysql-router:shared-db openstack-dashboard:
    shared-db
juju add-relation openstack-dashboard:identity-service keystone:identity-
    service
juju add-relation openstack-dashboard:certificates vault:certificates

# glance
juju deploy --to lxd:3 --series jammy --channel yoga/stable glance
juju deploy --channel 8.0/stable mysql-router glance-mysql-router
juju add-relation glance-mysql-router:db-router mysql-innodb-cluster:db-
    router
juju add-relation glance-mysql-router:shared-db glance:shared-db
juju add-relation glance:image-service nova-cloud-controller:image-service
juju add-relation glance:image-service nova-compute:image-service
juju add-relation glance:identity-service keystone:identity-service
juju add-relation glance:certificates vault:certificates

# ceph-mon
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --series jammy --channel quincy/
    stable --config ceph-mon.yaml ceph-mon
juju add-relation ceph-mon:osd ceph-osd:mon
juju add-relation ceph-mon:client nova-compute:ceph
```

```
juju add-relation ceph-mon:client glance:ceph

# cinder
juju deploy --to lxd:1 --series jammy --channel yoga/stable --config cinder
    .yaml cinder
juju deploy --channel 8.0/stable mysql-router cinder-mysql-router
juju add-relation cinder-mysql-router:db-router mysql-innodb-cluster:db-
    router
juju add-relation cinder-mysql-router:shared-db cinder:shared-db
juju add-relation cinder:cinder-volume-service nova-cloud-controller:cinder
    -volume-service
juju add-relation cinder:identity-service keystone:identity-service
juju add-relation cinder:amqp rabbitmq-server:amqp
juju add-relation cinder:image-service glance:image-service
juju add-relation cinder:certificates vault:certificates
juju deploy --channel yoga/stable cinder-ceph
juju add-relation cinder-ceph:storage-backend cinder:storage-backend
juju add-relation cinder-ceph:ceph ceph-mon:client
juju add-relation cinder-ceph:ceph-access nova-compute:ceph-access

# ceph-radosgw
juju deploy --to lxd:0 --series jammy --channel quincy/stable ceph-radosgw
juju add-relation ceph-radosgw:mon ceph-mon:radosgw
```

Listato A.3: Installazione di OpenStack attraverso Juju.