ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

Deployment di un cloud privato basato su OpenStack

Tesi di laurea in
VIRTUALIZZAZIONE E INTEGRAZIONE DI SISTEMI

Relatore
Prof. Vittorio Ghini

Candidato
Matteo Bambini

Prima Sessione di Laurea
Anno Accademico 2022-2023

Indice

1 Introduzione **1**
1.1 Il progetto 2
1.2 Struttura del documento 2

2 Obiettivi **5**
2.1 Progettazione del cluster 5
2.2 Installazione e configurazione dei sistemi di gestione del cloud 5
2.3 Installazione manuale del cloud OpenStack 6
2.4 Installazione in bundle del cluster OpenStack 6
2.5 Integrazione tra Terraform e OpenStack 6

3 Progettazione del cloud **7**
3.1 Hardware 7
3.2 Metodologia di deployment 8
3.3 Architettura del cloud 9
3.3.1 Networking 11

4 Prerequisiti **13**
4.1 MAAS 13
4.1.1 Funzionamento: i Controller 13
4.1.2 Risorse: i Nodi 15
4.1.3 Gestione della rete 16
4.1.4 Requisiti del sistema MAAS 17
4.1.5 Installazione 19
4.1.6 Installazione del region e rack controller 20
4.1.7 Configurazione 22
4.1.8 Aggiunta dei nodi 26
4.2 Juju 30
4.2.1 Funzionamento: Controller e Charmed Operator 30
4.2.2 Concetti chiave 32
4.2.3 Installazione 34

4.2.4 Installazione e configurazione del client Juju 35
4.2.5 Deploy del controller Juju e creazione del model 38
4.2.6 Accesso alla dashboard 40

5 OpenStack 43
5.1 Componenti di OpenStack 43
5.1.1 Cinder 43
5.1.2 Glance 44
5.1.3 Horizon 44
5.1.4 Keystone 44
5.1.5 Neutron 44
5.1.6 Nova 45
5.1.7 Placement 45
5.2 Componenti esterni a OpenStack 45
5.2.1 Vault 45
5.2.2 Open vSwitch e Open Virtual Network 46
5.2.3 Ceph 47

6 Installazione di OpenStack 49
6.1 Concetti preinstallazione 49
6.1.1 Tipologie di charm su OpenStack. 49
6.1.2 Versione di OpenStack. 50
6.1.3 Distribuzione dei charm all'interno delle macchine del cluster. 50
6.1.4 Monitoraggio del deploy. 51
6.2 Installazione manuale 53
6.2.1 Accesso alla dashboard Horizon. 63
6.3 Installazione in bundle 67
6.4 Configurazione iniziale 69
6.4.1 Configurazioni da parte dell'amministratore 69
6.4.2 Configurazioni da parte dell'utente 72

7 Utilizzo di OpenStack 77
7.1 Identity 77
7.1.1 Domains 77
7.1.2 Groups 78
7.1.3 Roles 78
7.1.4 Projects 78
7.1.5 Users 79
7.2 Network 79
7.2.1 Networks e Subnets 79

7.2.2 Routers 81
7.2.3 Security Groups 81
7.2.4 Floating IPs 83
7.3 Storage 84
7.3.1 Volumes 84
7.3.2 Snapshots 84
7.4 Compute 84
7.4.1 Flavors 84
7.4.2 Key Pairs 85
7.4.3 Images 86
7.4.4 Instances 86

8 Load Balancer 89
8.1 Componenti 91
8.1.1 Octavia 91
8.1.2 Barbican 91
8.2 Installazione 91
8.2.1 Installazione manuale 91
8.2.2 Installazione in bundle 93
8.3 Configurazione iniziale 94
8.4 Utilizzo 95
8.4.1 Creazione di un load balancer 96
8.4.2 Gestione del load balancer 97

9 Terraform 101
9.1 Funzionamento 101
9.2 Utilizzo 102
9.2.1 Workflow 102
9.2.2 Progetti con Terraform 103
9.2.3 Configurazioni di Terraform: HCL 104
9.3 Terraform e OpenStack 106
9.3.1 Introduzione 106
9.3.2 Progetto 1: istanza 106
9.3.3 Progetto 2: infrastruttura completa 109
9.3.4 Progetto 3: Proof of Concept 112

10 Conclusioni e sviluppi futuri 117
10.1 Sviluppi futuri 118

A Bundle di installazione 123
A.1 OpenStack 123
A.2 Octavia e Brabican 131

B Progetti Terraform **135**
B.1 Progetto 1: istanza 135
B.1.1 main.tf 135
B.2 Progetto 2: infrastruttura completa 136
B.2.1 example.tfvars 136
B.2.2 instances.tf 136
B.2.3 main.tf 137
B.2.4 networking.tf 138
B.3 Progetto 3: Proof of Concept 139
B.3.1 example.tfvars 139
B.3.2 init-instance.sh 139
B.3.3 instances.tf 140
B.3.4 load_balancer.tf 140
B.3.5 main.tf 142
B.3.6 networking.tf 143
B.3.7 outputs.tf 144

Capitolo 1

Introduzione

Il cloud computing consiste in una serie di servizi che permettono di creare e gestire risorse IT on-demand con una tariffa basata sul consumo. I principali vantaggi sono la flessibilità dell'infrastruttura e la notevole riduzione dei costi di manutenzione, considerando anche che rispetto ad un'infrastruttura on-premises non c'è nessun tipo di hardware da gestire. Per questi motivi negli ultimi anni la richiesta di servizi di cloud computing è aumentata notevolmente e ad oggi di tratta di un mercato da centinaia di miliardi di dollari all'anno che vede come player principali Amazon Web Services, Google Cloud Platform e Microsoft Azure.

Purtroppo però non è tutto oro quel che luccica; esistono infatti diversi svantaggi nell'utilizzare questi tipi di servizi tra i quali il più rilevante è sicuramente il costo. Nonostante i benefici citati prima può essere poco conveniente utilizzare servizi di cloud compunting per casi d'uso in cui si ha bisogno di ingenti risorse di calcolo o di elevati volumi di storage. Per questo motivo molte realtà aziendali preferiscono creare un'infrastruttura cloud privata. Ci sono numerose piattaforme che permettono di realizzare questo tipo di infrastrutture e una di queste è OpenStack.

OpenStack è una piattaforma di cloud computing open source che permette di realizzare infrastrutture cloud sia private che pubbliche. Esistono infatti cloud provider che utilizzano OpenStack come base per erogare i loro servizi; un esempio è OVH, uno dei più grandi provider d'Europa. OpenStack è anche scelto da molte aziende come base per il loro cloud privato per via della sua architettura modulare e grazie al fatto che le funzionalità offerte sono molto simili a quelle dei cloud provider.

1.1 Il progetto

L'obiettivo di questo progetto di tesi è prima di tutto installare e configurare un cloud privato basato su OpenStack per poi scoprirne le caratteristiche e studiarne le funzionalità nel dettaglio.

È inoltre previsto lo studio di Terraform, un Infrastructure as Code (IaC) tool, ovvero uno strumento che permette di definire infrastrutture cloud utilizzando solamente codice scritto all'interno di file di configurazione. In particolare si vuole studiare come questo strumento si integra con OpenStack e di come è possibile creare risorse all'interno di OpenStack utilizzandolo.

1.2 Struttura del documento

In questa sezione verrà esplicitata la struttura del documento descrivendo in maniera molto sintetica il contenuto di ciascun capitolo.

Obiettivi - Descrizione approfondita degli obbiettivi posti per lo svolgimento del progetto.

Progettazione del cluster - Descrizione dell'architettura del cloud e approfondimento sulle scelte di progettazione e sulle motivazione di tali scelte.

Prerequisiti - Descrizione dei componenti software necessari come supporto all'installazione di OpenStack (MAAS e Juju) e spiegazione delle procedure di installazione e configurazione di tali componenti.

OpenStack - Descrizione della piattaforma OpenStack, dei suo componenti principali che verranno installati e dei software esterni necessari per far funzionare la piattaforma.

Installazione di OpenStack - Descrizione delle procedure di installazione manuale e in bundle di OpenStack e spiegazione delle procedure di configurazione.

Utilizzo di OpenStack - Istruzioni su come utilizzare al meglio un cloud OpenStack e su come sfruttarne a pieno le risorse.

Load Balancer - Descrizione del servizio di load balancing di OpenStack, dei componenti che ne fanno parte, spiegazione delle procedure di installazione e di configurazione di tali componenti e istruzioni per l'utilizzo del load balancer.

Terraform - Descrizione del tool Terraform, di come questo tool si integra con OpenStack e dei progetti Terraform realizzati durante lo svolgimento della tesi.

Capitolo 2

Obiettivi

In questo capitolo verranno descritti i principali obiettivi di questo progetto di tesi. Alcuni termini e concetti verranno dati per scontati ma saranno poi approfonditi e spiegati nel dettaglio nei capitoli successivi.

Lo scopo principale è creare un cloud privato OpenStack che si avvicini il più possibile come funzionalità ad uno che potrebbe essere usato in una realtà aziendale e successivamente studiare in che modo Terraform si integra con OpenStack e quali funzionalità mette a disposizione.

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 a nostra disposizione. Dato il numero limitato di macchine con architettura amd64 disponibili per questo progetto, alcuni dei servizi che lo supportano verranno installati su hardware a basso costo e con architettura diversa (verosimilmente Raspberry Pi), in modo da avere il numero maggiore di nodi dedicati esclusivamente ai servizi di OpenStack. Questo ovviamente verrà fatto dopo aver verificato che non ci siano incompatibilità per via delle diverse architetture.

2.2 Installazione e configurazione dei sistemi di gestione del cloud

Il tipo di deployment che è stato scelto comporta l'utilizzo di sistemi avanzati di gestione del cloud, il cui scopo è semplificare il provisioning delle macchine fisiche e l'installazione del cloud stesso; nello specifico, si tratta di MAAS e

Juju (approfonditi nei capitoli 4.1 e 4.2). Questi sistemi devono essere installati e configurati prima di procedere con l'installazione di OpenStack.

2.3 Installazione manuale del cloud OpenStack

Come scritto nella guida all'installazione di OpenStack [35], è consigliato fare la prima installazione manualmente, in modo da comprendere quali componenti formano il cloud e in che modo interagiscono tra loro. Per questo motivo è stato deciso di procedere in questa maniera prima utilizzare l'installazione in bundle.

2.4 Installazione in bundle del cluster OpenStack

La metodologia di deployment scelta supporta anche l'installazione in bundle: tutti i componenti con le relative relazioni vengono definiti all'interno di un file in formato YAML e Juju, leggendo questo file, si occupa di installare tutti i Charms necessari. Questo tipo di installazione è sicuramente molto efficiente in ambienti reali in cui si possono avere anche centinaia di macchine da gestire.

2.5 Integrazione tra Terraform e OpenStack

L'ultimo passo, dopo aver ottenuto un cloud OpenStack perfettamente funzionante, sarà quello di studiare il supporto di Terraform a OpenStack e di verificare quali funzionalità sono disponibili e quali non lo sono.

Capitolo 3

Progettazione del cloud

All'interno di questo capitolo verranno descritte quelle che sono state le scelte di progettazione del cloud 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 [33]:

- 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. 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 [30].

Ansible in Docker Containers Questa metodologia di deployment consiste nell'utilizzare Kolla, un servizio di OpenStack, per installare ciascuno dei componenti necessari al cloud 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 cloud 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 cloud 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 cloud

In figura 3.1 è descritta l'architettura del cloud. 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.

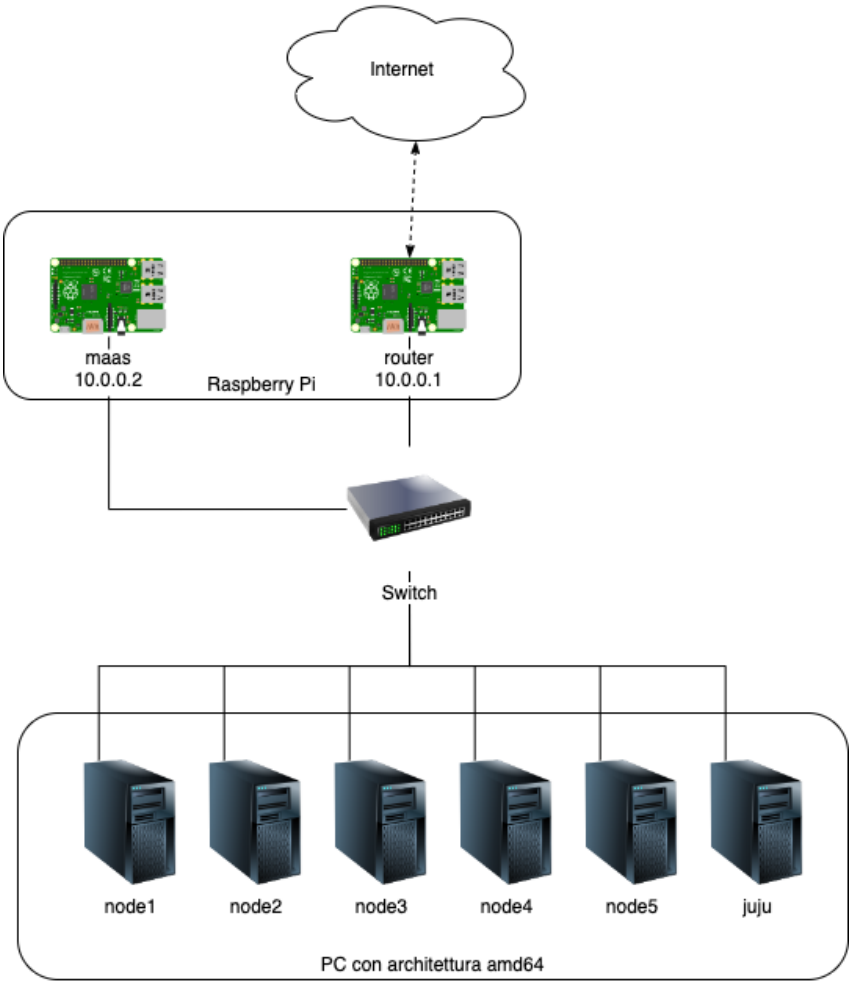

Figura 3.1: Architettura di rete del cloud.

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 cloud 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 cloud 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.

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 [35]. 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 [20]. 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

Prerequisiti

In questo capitolo, verranno affrontate tutte le tecnologie che si rendono necessarie per la messa in opera del cloud OpenStack. In particolare, come visto nella sezione 3.2, per poter effettuare il deploy del cloud tramite *Charms Deployment*, metodologia affrontata in questa sede, è necessaria l'installazione del sistema MAAS in primis e poi del sistema Juju. Di seguito verranno affrontati nei dettagli, partendo da cosa sono e dal loro funzionamento fino ad arrivare alla loro installazione e configurazione.

4.1 MAAS

MAAS (Metal-as-a-Service [21]) è uno strumento open source e gratuito di provisioning di server e di host bare-metal creato e mantenuto 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.

4.1.1 Funzionamento: i Controller

Il funzionamento di MAAS è suddiviso 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'interfaccia 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).

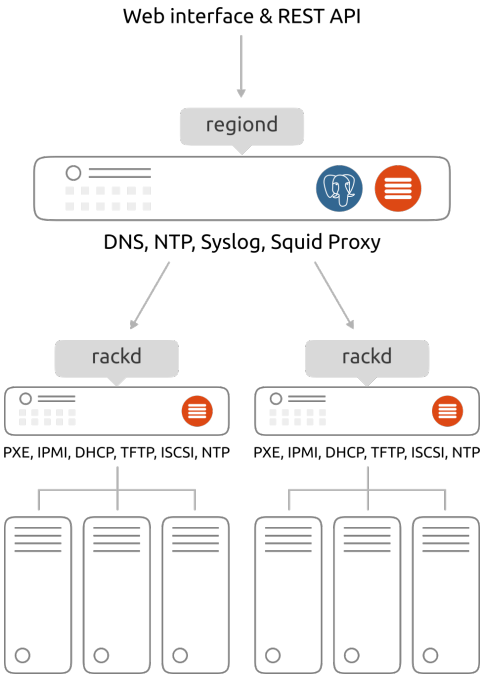

Figura 4.1: Configurazione con un region controller (regiond) e due rack controller (rackd) [23].

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 [43]) 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.

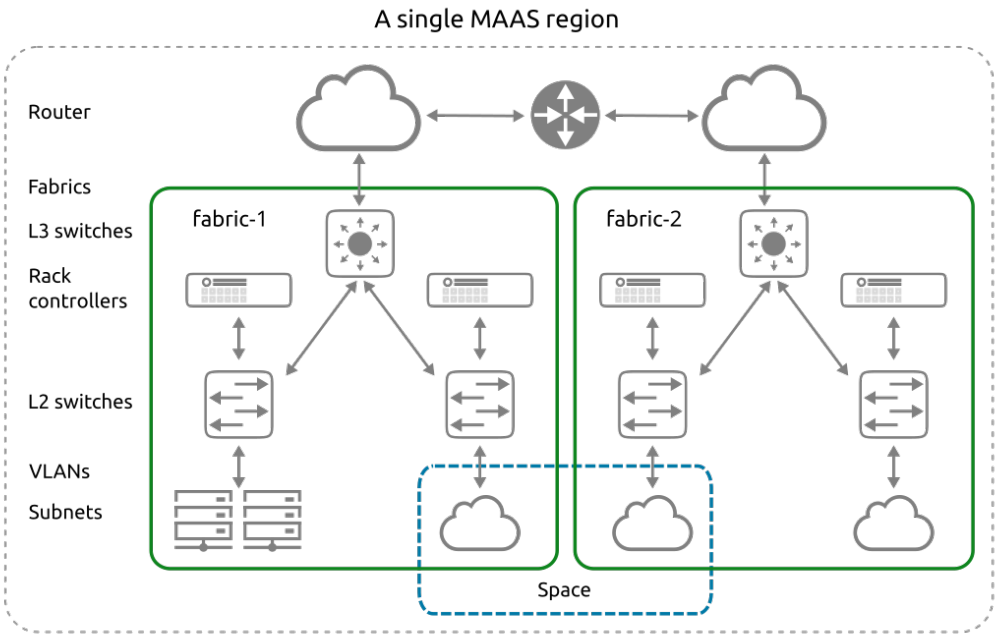

Figura 4.2: Esempio generico di configurazione di rete di un data center [22].

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 [27].

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.

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

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.

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.

4.1.5 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 4.2.3).
- 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 [40]).

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* [39], 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 [25] e alla documentazione MAAS [24].

4.1.6 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.

```
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.

```
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.

```
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 4.1.7).

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 4.9 nella sezione 4.2.3). 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 4.1.7).

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

Listato 4.5: Salvataggio della chiave API.

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

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`.

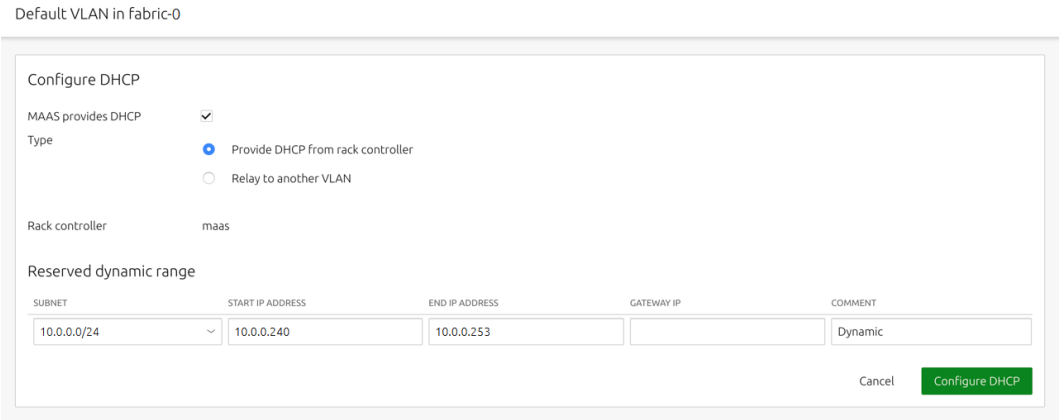

Figura 4.3: Finestra di configurazione del DHPC.

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.

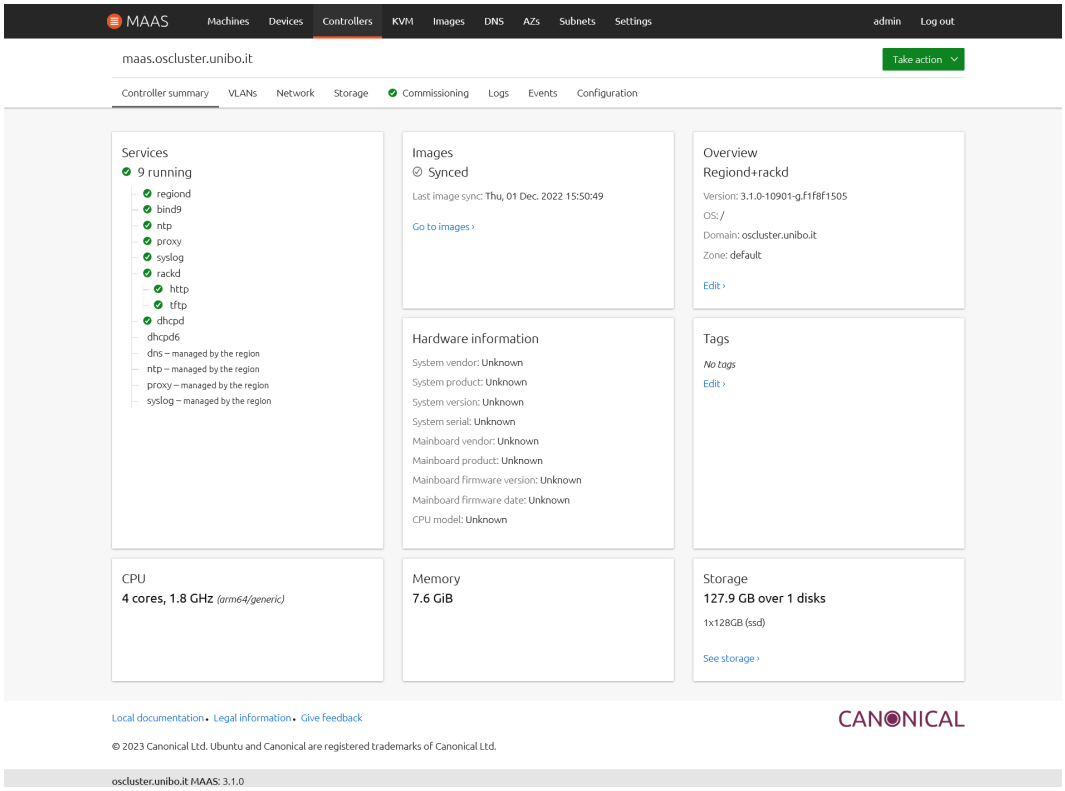

Figura 4.4: Schermata di sintesi del region+rack controller.

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*".

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.

4.1.8 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 (e.g. 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 [26].

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 5.2.2.

   (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.

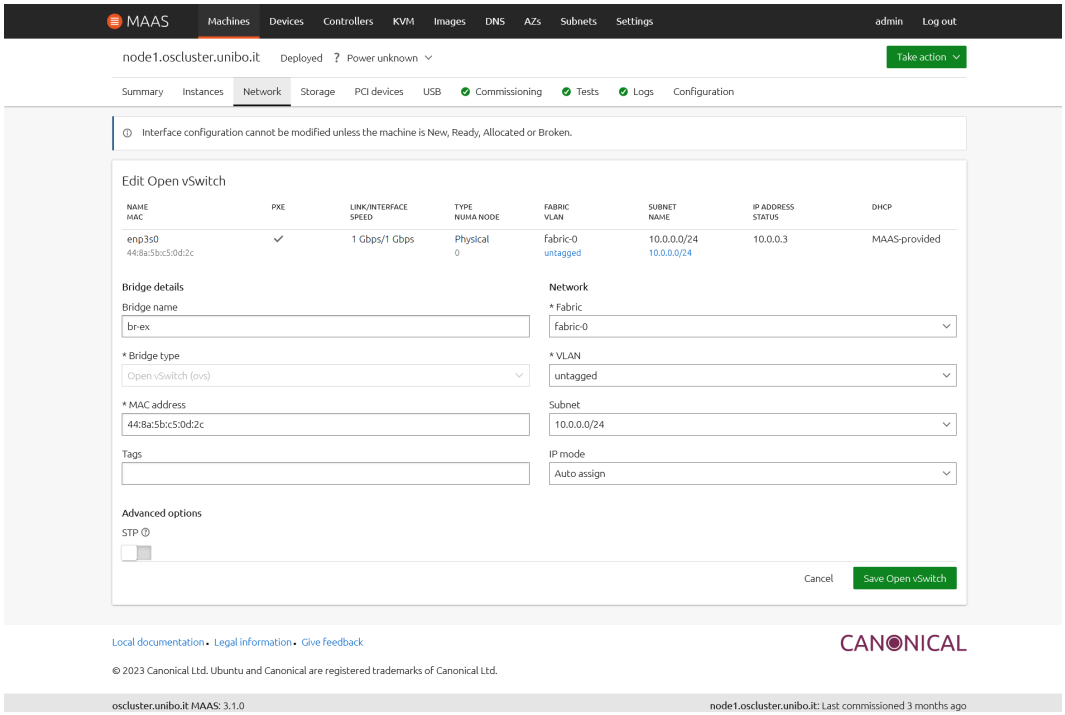

Figura 4.5: schermata di editing della configurazione Open vSwitch su un nodo.

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.

4.2 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].

4.2.1 Funzionamento: Controller e Charmed Operator

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 4.6), quest'ultimi sono descritti attraverso i modelli.

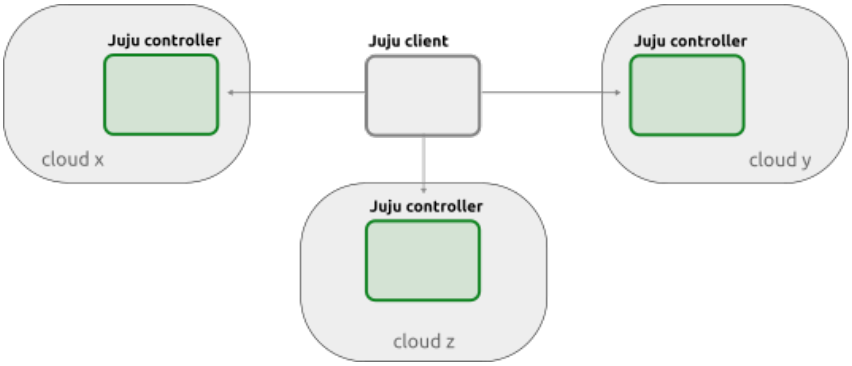

Figura 4.6: Schema del sistema Juju. Un Juju client può comunicare con più Juju controller [8].

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.

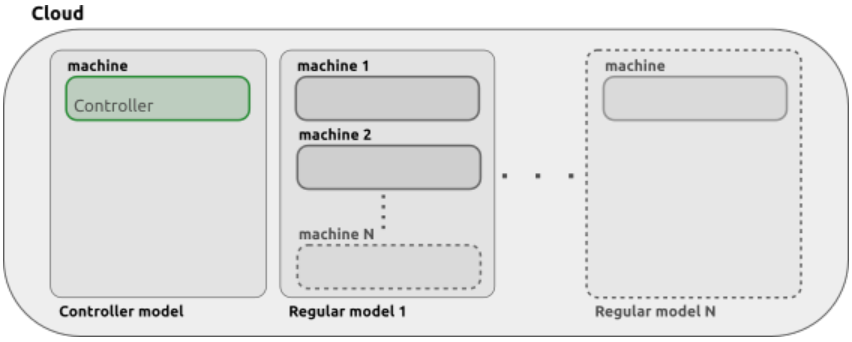

Figura 4.7: Rappresentazione del cloud in modelli. Un singolo Controller model e diversi Regular model su diverse macchine [13].

In sintesi, un cloud è composto da un controller che gestisce vari modelli e ogni modello distribuisce applicazioni su diverse macchine (figura 4.7).

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* [3], il luogo dove è possibile sviluppare e condividere gratuitamente i propri charm.

4.2.2 Concetti chiave

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* [3].

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.

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 4.8 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.

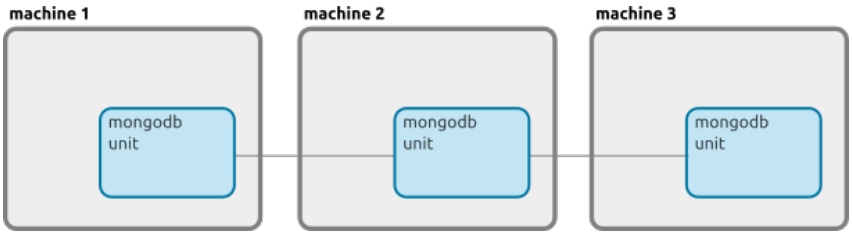

Figura 4.8: Suddivisione dell'applicazione MongoDB su tre unit [16].

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 4.9, 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.

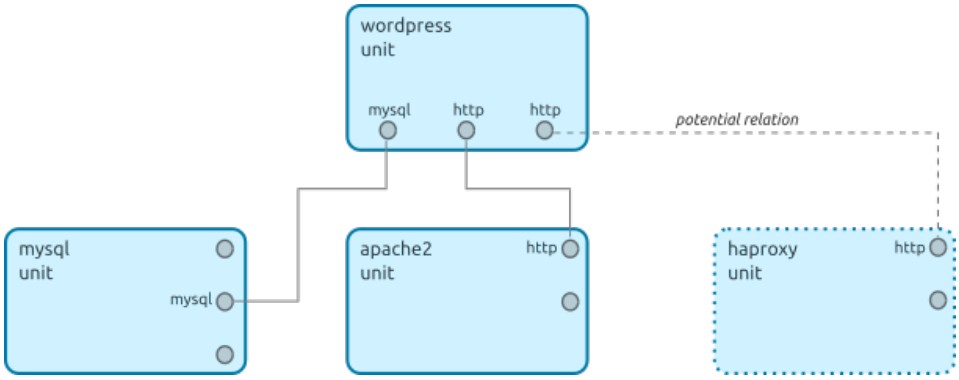

Figura 4.9: WordPress in relation con MySQL e Apache (eventualmente anche con HAProxy) [15].

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

Va evidenziato che le relation non sono connessioni dirette tra i vari charm, bensì 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.

4.2.3 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 4.1.5), 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 (sezione 4.1). 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 4.1.5, 4.1.7 e 4.1.8 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 4.6, 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].

4.2.4 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.

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

Listato 4.6: 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]
```

```
5     endpoint: http://10.0.0.2:5240/MAAS
```

Listato 4.7: 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 4.9 e 4.10).
- `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.

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

Listato 4.8: 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 4.7
- Il valore *mass-one* indica il nome del cloud corrispondente a quello scelto sempre nel listato 4.7.

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 4.10.

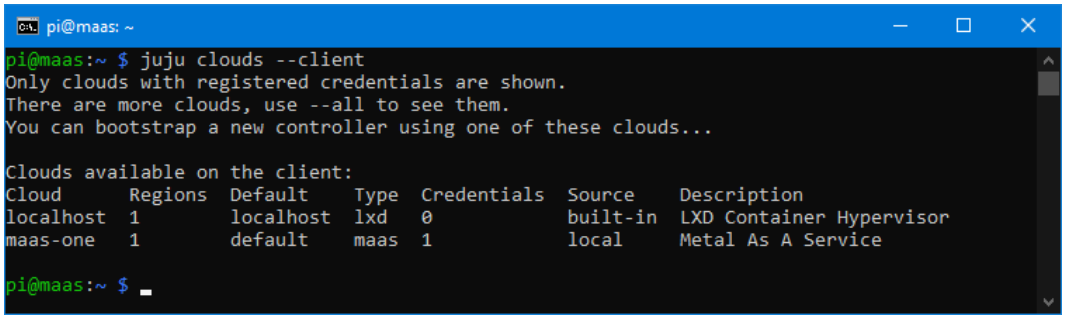

Figura 4.10: Elenco dei cloud presenti nel sistema Juju.

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.

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

Listato 4.9: File yaml contenente le credenziali del cloud MAAS.

- `mass-one` è il nome del cloud scelto nel listato 4.7.
- `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 4.10: Aggiunta delle credenziali del cloud MAAS in Juju.

(Vedasi il listato 4.8 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 4.11.

```
pi@maas: ~
pi@maas:~ $ juju credentials --client --show-secrets --format yaml
client-credentials:
  maas-one:
    anyuser:
      auth-type: oauth1
      maas-oauth: zAgKqRKwhdqk2jMjwc:WWFWEuuDSdD4CRFJWG:QrQfs5L7h8bdkUsswhkQSPsXnG7Ejkwx
pi@maas:~ $
```

Figura 4.11: Elenco delle credenziali presenti nel sistema Juju.

4.2.5 Deploy del controller Juju e creazione del model

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.

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

Listato 4.11: 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 4.1.7.

  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 4.11.

  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 4.7, 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 4.1.7) 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 4.1.8) 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 4.12.

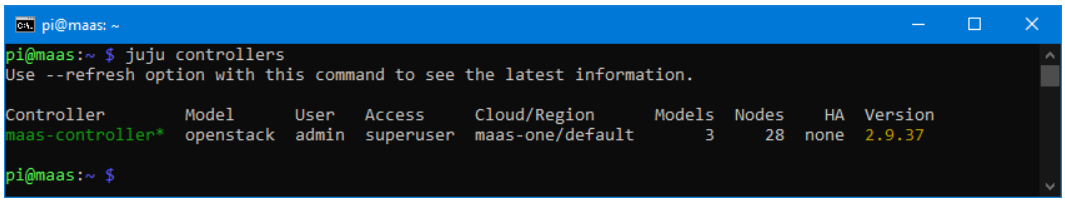

Figura 4.12: Elenco dei controller registrati nel client Juju.

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.

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

Listato 4.12: 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 4.13 l'output riassume il cloud appena creato.

```
pi@maas: ~
pi@maas:~ $ juju status
Model      Controller       Cloud/Region      Version  SLA          Timestamp
openstack  maas-controller  maas-one/default  2.9.37   unsupported  21:18:53+01:00

Model "admin/openstack" is empty.
pi@maas:~ $
```

Figura 4.13: Status del cloud vuoto mass-one.

4.2.6 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 4.14 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 4.12.

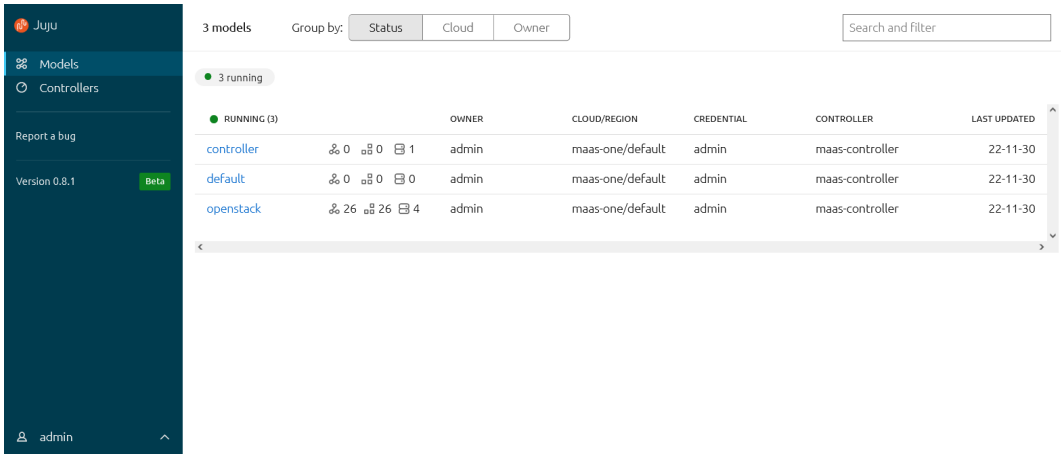

Figura 4.14: Esempio di schermata d'accesso della Juju dashboard.

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 5

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 5.2.

5.1 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.

5.1.1 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).

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.5 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.

5.1.6 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.

5.1.7 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).

5.2 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.

5.2.1 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 5.1 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

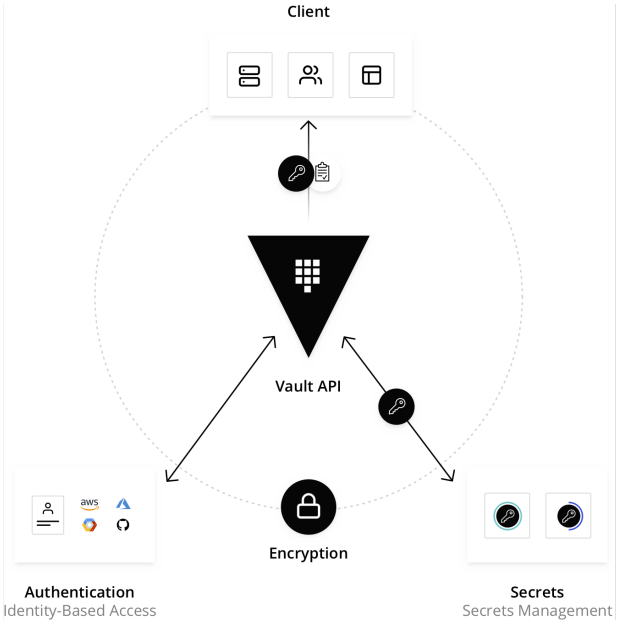

Figura 5.1: Schema di funzionamento di Vault [42].

vasta flessibilità. Il charm di Vault utilizza di default il provider basato sui Token [41].

5.2.2 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 [37].

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 [36].

5.2.3 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 [2].

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 3 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 [38].

Capitolo 6

Installazione di OpenStack

Una volta creato il model con Juju (ref), che rappresenta il contenuto software del cloud, arriva il momento di popolarlo con le applicazioni che costituiranno i servizi del cloud. 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.

6.1 Concetti preinstallazione

6.1.1 Tipologie di charm su OpenStack.

Su OpenStack esistono due tipologie di charm [31]: 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.

6.1.2 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.

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

Come spiegato nella sezione 4.2.1, 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 per il deployment dei servizi dedicati al load balancer).

- 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`).

6.1.4 Monitoraggio del deploy.

Indipendentemente dalla tipologia di deployment scelta (un charm alla volta o in bundle rif), 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.

```
pi@maas: ~
Every 2.0s: juju status --color                          maas.oscluster.unibo.it: Sat Aug  6 01:46:50 2022

Model      Controller       Cloud/Region      Version  SLA          Timestamp
openstack  maas-controller  maas-one/default  2.9.32   unsupported  01:46:50+02:00

App           Version  Status   Scale  Charm         Channel        Rev  Exposed  Message
ceph-osd      17.2.0   blocked      4  ceph-osd      quincy/stable  534  no       Missing relation: monitor
nova-compute           waiting    1/3  nova-compute  yoga/stable    594  no       agent initializing

Unit             Workload     Agent       Machine  Public address  Ports  Message
ceph-osd/0       blocked      idle        0        10.0.0.3               Missing relation: monitor
ceph-osd/1*      blocked      idle        1        10.0.0.4               Missing relation: monitor
ceph-osd/2       blocked      idle        2        10.0.0.6               Missing relation: monitor
ceph-osd/3       blocked      idle        3        10.0.0.5               Missing relation: monitor
nova-compute/0*  maintenance  executing   1        10.0.0.4               (install) Installing apt packages
nova-compute/1   waiting      allocating  2        10.0.0.6               agent initializing
nova-compute/2   waiting      allocating  3        10.0.0.5               agent initializing

Machine  State    DNS       Inst id  Series  AZ       Message
0        started  10.0.0.3  node1    jammy   default  Deployed
1        started  10.0.0.4  node2    jammy   default  Deployed
2        started  10.0.0.6  node4    jammy   default  Deployed
3        started  10.0.0.5  node3    jammy   default  Deployed
```

Figura 6.1: Esempio d'output di `juju status` durante le prime fasi di deploy dei charm.

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 6.1.

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.

6.2 Installazione manuale

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 [35].

Come prima cosa, bisogna assicurarsi che i comandi verranno impartiti al Juju controller e al model corretto (sezione 4.2.5). Per farlo, baserà eseguire il comando mostrato nel listato 6.1.

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

Listato 6.1: 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 5.2.3). 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 6.2.

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

Listato 6.2: 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 6.3 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 6.3: 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 4.1.8).

Nova Compute. Il prossimo charm da installare è nova-compute (sezione 5.1.6), 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 6.4.

```
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 6.4: 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 6.5: 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 6.6: Deploy del charm *mysq-innodb-cluster*.

Vault. In questa fase verrà installato il charm vault (sezione 5.2.1) 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 6.7: 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 6.8.

```
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 6.8: 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 6.9: 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 6.10: 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 6.11: Generazione delle chiavi d'apertua del vault.

Ora è possibile aprire Vault, utilizzando tre delle cinque chiavi generate nel listato 6.11.

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

Listato 6.12: 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 6.13: Generazione del token temporaneo.

- Il `<Initial Root Token>` è possibile trovarlo nell'output (nel file se è stato salvato) del listato 6.11.

Infine, è possibile autorizzare il charm vault con il token appena generato nel listato 6.13. 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 6.14: Autorizzazione al charm *vault*.

- Al posto di `<token>` bisogna inserire il token temporaneo generato nel listato 6.13.

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 6.15: Creazione del certificato autofirmato.

Come ultimo passaggio, verrà collegato il certificato di vault, creato nel listato 6.15, al DB del cloud attraverso l'aggiunta della relativa relazione.

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

Listato 6.16: Aggiunta della relazione per collegare il certificato a DB.

Neutron. Per implementare la rete con Neutron (sezione 5.1.5), verranno installati quattro charms:

- ovn-central per il controllo delle OVN (sezione 5.2.2), 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.

```yaml
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 6.17: Configurazione di *Neutron* nel file *neutron.yaml*.

- `bridge-interface-mappings` indica la mappatura del *OvS bridge* creato nel voce 6 nella sezione 4.1.8 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 6.18: 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 6.8)

```
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
```

```
9
10 juju deploy --channel 8.0/stable mysql-router neutron-api-mysql-router
11 juju add-relation neutron-api-mysql-router:db-router mysql-innodb-cluster:
       db-router
12 juju add-relation neutron-api-mysql-router:shared-db neutron-api:shared-db
```

Listato 6.19: Aggiunta delle varie relazioni per *Neutron*.

Keystone. Il charm keystone (sezione 5.1.4) è il componente che si occuperà di fornire le API per l'autenticazione dei client. Verrà installato in un'unica unit.

```
1 juju deploy --to lxd:0 --series jammy --channel yoga/stable keystone
2
3 juju deploy --channel 8.0/stable mysql-router keystone-mysql-router
4 juju add-relation keystone-mysql-router:db-router mysql-innodb-cluster:
       db-router
5 juju add-relation keystone-mysql-router:shared-db keystone:shared-db
6
7 juju add-relation keystone:identity-service neutron-api:identity-service
8 juju add-relation keystone:certificates vault:certificates
```

Listato 6.20: 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.

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

Listato 6.21: 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`.

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

Listato 6.22: 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 6.23: Deploy del charm *nova-cloud-controller*.

Placement. Placement (sezione 5.1.7), 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 6.24: 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 5.1.3), 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 6.25: Deploy del charm *openstack-dashboard*.

Glance. Glance (sezione 5.1.2) 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 6.26: 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 6.27: 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 6.28: Deploy del charm *ceph-mon*.

Cinder. Cinder (sezione 5.1.1) 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 6.29: 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 6.30: 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 6.29 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 6.31: Deploy del charm *cinder-ceph*.

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

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 6.32: 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 6.2 a 6.4. Tutte le applicazioni e le unit avranno lo status *active* con i relativi agent in *idle*, e come message mostreranno che sono in *ready*.

6.2.1 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`

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 6.3: 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 6.4: Status finale del cloud OpenStack post deployment con i charm elenco delle Machine.

(come mostrato in figura 6.3) sia estrapolarlo utilizzando la combinazione di comandi mostrati nel listato 6.33.

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

Listato 6.33: 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`.

Dopodiché, è necessario conoscere le credenziali dell'amministratore di sistema. La password può essere richiesta a Keystone come mostrato nel seguente listato listato 6.34.

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

Listato 6.34: 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 6.5.

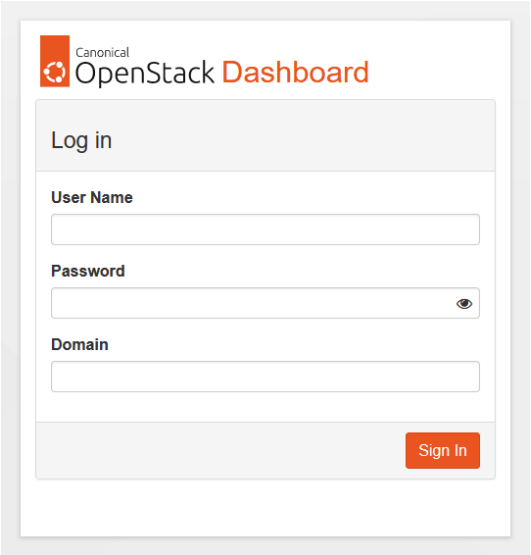

Figura 6.5: Schermata di login della dashboard di OpenStack.

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 6.6).

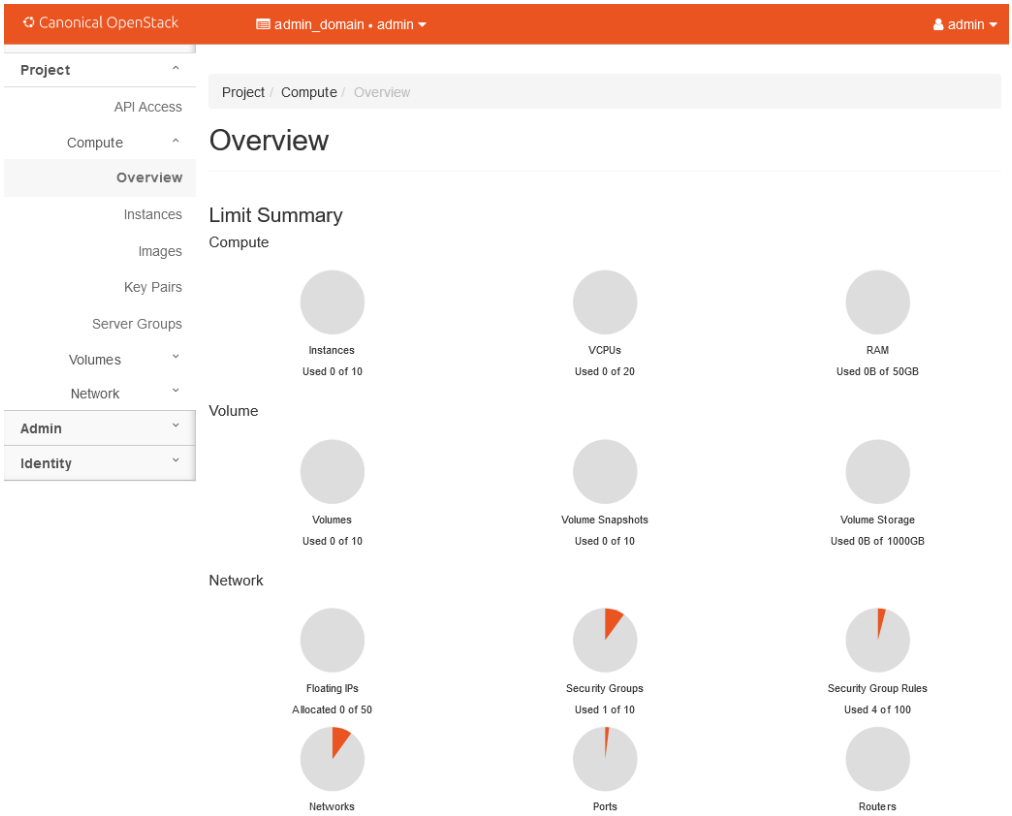

Figura 6.6: Schermata post log-in della dashboard di OpenStack con qualche risorsa in uso.

6.3 Installazione in bundle

Come accennato in precedenza all'interno del capitolo riguardante Juju, un bundle è un file yaml contenente la lista di tutti i charm che si vogliono installare con le relative configurazioni; è possibile inoltre definire delle variabili all'interno del file bundle che poi potranno essere utilizzate come parametri di configurazione di uno o più charm.

All'interno del listato 6.35 è riportato un esempio estratto dal bundle realizzato durante lo svolgimento di questo progetto, che mostra in che modo deve essere strutturata la configurazione di un charm (l'intero bundle è consultabile in appendice A.1).

```yaml
variables:
    osd-devices: &osd-devices /dev/sdb /dev/sdc
    openstack-origin: &openstack-origin cloud:focal-yoga
    ceph-channel: &ceph-channel quincy/stable
applications:
    ceph-osd:
        charm: ch:ceph-osd
        channel: *ceph-channel
        num_units: 4
        options:
          osd-devices: *osd-devices
          source: *openstack-origin
        to:
        - '0'
        - '1'
        - '2'
        - '3'
```

Listato 6.35: Esempio di configurazione di un charm all'interno del bundle.

Come si può notare dall'esempio sopra la variabili devono essere definite all'intenro della chiave *variables* e devono avere il seguente formato: `nome-variabile: &nome-variabile valore`. Possono essere richiamate utilizzando la sintassi seguente: `*nome-variabile`

Per quanto riguarda i charm, ciascuno di essi deve essere definito all'interno della chiave *applications*. Come si può vedere dall'esempio nel listato 6.35, il charm (in questo caso *ceph-osd*) ha diversi parametri di configurazione: alcuni sono comuni a tutti i charm mentre altri sono specifici per il charm che si sta configurando. I parametri in comune tra tutti i charm sono i seguenti:

- **charm**: indica il charm da installare (il prefisso *ch:* indica che deve essere scaricato da Charmhub)

- **channel**: indica la versione del charm
- **num_units**: indica il numero di istanze che devono essere eseguite
- **options**: al suo interno devono essere specificate tutti le configurazioni specifiche del charm
- **to**: indica le macchine sulle quali deve essere installato il charm; se viene inserito solamente il numero della macchina il charm verrà installato direttamente sul sistema operativo host mentre se viene specificato nel formato `lxd:<id_macchina>` verrà istanziato un container sulla macchine selezionata all'interno del quale verrà installato il charm

Generazione del bundle. La generazione del bundle è un'operazione abbastanza complessa che può richiedere molto tempo e durante la quale è facile fare piccoli errori che causerebbero il malfunzionamento di tutto il sistema. Per questo motivo esiste una funzionalità di Juju che permette di generare un bundle partendo da un deployment già esistente. Questa operazione può essere eseguita semplicemente lanciando il comando `juju export-bundle`.

Durante lo svolgimento di questo progetto è stato tentato questo approccio però sono stati riscontrati diversi problemi a causa dei quali non è stato possibile installare OpenStack utilizzando solamente il bundle generato da Juju. La soluzione adottata è stata quella di fare un'unione manuale tra il bundle generato e quello di esempio messo a disposizione da OpenStack [34].

Deploy del bundle Una volta generato il bundle è possibile eseguire il deploy semplicemente lanciando il comando `juju deploy ./bundle.yaml`. È importante che il nome del file bundle sia inserito come percorso anche se si trova nella cartella corrente perché in caso contrario Juju lo tratterebbe come se fosse il nome di un charm e ovviamente l'installazione fallirebbe.

Una volta conclusa l'installazione dei charm sarà comunque necessario eseguire la configurazione iniziale di *Vault* manualmente seguendo la procedura utilizzata durante l'installazione manuale di OpenStack e descritta nella sezione 6.2.

6.4 Configurazione iniziale

In questo capitolo verrà trattata la configurazione iniziale di un cloud OpenStack. Tutta la configurazione verrà fatta seguendo la guida di installazione [32] 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 6.2.1.

Per comprendere meglio le configurazioni utilizzate in questa sezione, sia lato amministratore che utente, si consiglia di visionare la capitolo 7 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 6.36

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

Listato 6.36: 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 6.37 per impostare le variabili d'ambiente in modo da avere accesso al cloud con privilegi di amministrazione.

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

Listato 6.37: Configurazione dell'accesso al cloud.

6.4.1 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 6.38 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 6.38: 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 6.39.

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

Listato 6.39: 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 6.40 (per maggiori informazioni sui flavor si veda la sezione 7.4.1).

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

Listato 6.40: 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 6.41.

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

Listato 6.41: 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 6.42.

```
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 6.42: 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 6.43.

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

Listato 6.43: 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 6.44; in questo modo che potrà effettuare tutte quelle operazioni che non richiedono permessi di amministrazione.

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

Listato 6.44: Assegnazione del ruolo al nuovo utente.

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

6.4.2 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 6.45. 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 6.45: File per l'ambiente utente *user1*.

- I valori delle variabili sono i medesimi creati per l'utente nel listato 6.43.
- 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 6.46 per poterla creare.

```
openstack network create --internal user1_net

openstack subnet create --network user1_net --dns-nameserver 10.0.0.2 --
    subnet-range 192.168.0/24 --allocation-pool start=192.168.0.10,end
    =192.168.0.199 user1_subnet
```

Listato 6.46: 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 6.47: Creazione di una router virtuale.

- `--external-gateway`: indica il nome della rete esterna creata nel listato 6.41
- 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 6.48 è mostrato come viene importata la chiave pubblica SSH all'interno di OpenStack.

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

Listato 6.48: 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 6.49, 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 6.49: 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 6.50

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

Listato 6.50: 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 6.51 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 6.51: 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 7 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 7

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.

7.1 Identity

7.1.1 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.

7.1.2 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.

7.1.3 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.

7.1.4 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 7.1) e successivamente selezionare dal menu a tendina il progetto che si vuole utilizzare.

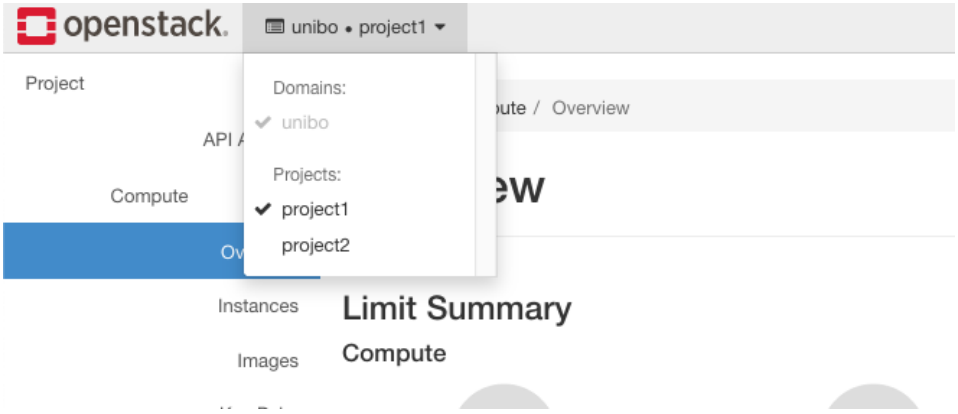

Figura 7.1: Selezione del progetto tramite l'interfaccia Web.

7.1.5 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.

7.2 Network

7.2.1 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`)

7.2.2 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.

7.2.3 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

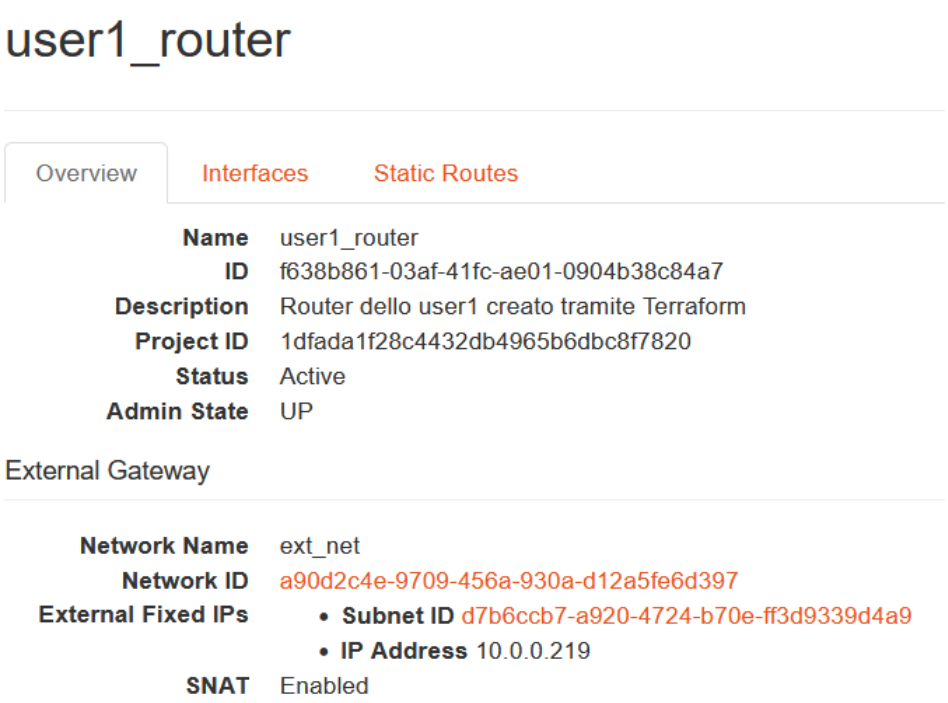

Figura 7.2: Pagina con le informazioni di un router.

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

7.2.4 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.

7.3 Storage

7.3.1 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.

7.3.2 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.

7.4 Compute

7.4.1 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.

7.4.2 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*.

Per gestire le coppie di chiavi 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 bottone *Create Key Pair*, mentre per importare una chiave pubblic già esistente si deve cliccare su *Import Public Key*.

7.4.3 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

7.4.4 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 8

Load Balancer

Il load balancing è una tecnica che consiste nella distribuzione del carico di lavoro tra molteplici server aumentando l'affidabilità a la scalabilità di un sistema. Nello specifico quello che verrà trattato in questo capitolo è il servizio LBaaS (Load Balancer as a Service) di OpenStack, ovvero il servizio di OpenStack che permette agli utenti di creare e utilizzare i load balancer.

I servizi di networking di OpenStack mettono a disposizione due diverse implementazioni di LBaaS attraverso il plugin *neutron-lbaas*: LBaaS v1 (attualmente deprecato) e LBaaS v2. Octavia, il componente che è stato installato durante lo svolgimento di questo progetto, è una delle implementazioni di LBaaS v2.

Ci sono alcuni concetti e termini chiave riguardanti il funzionamento di Octavia e, più in generale, di LBaaS v2 che è necessario comprendere prima di proseguire con la descrizione dei componenti e dell'installazione; in figura 8.1 è presente il diagramma che rappresenta il funzionamento logico di LBaaS v2.

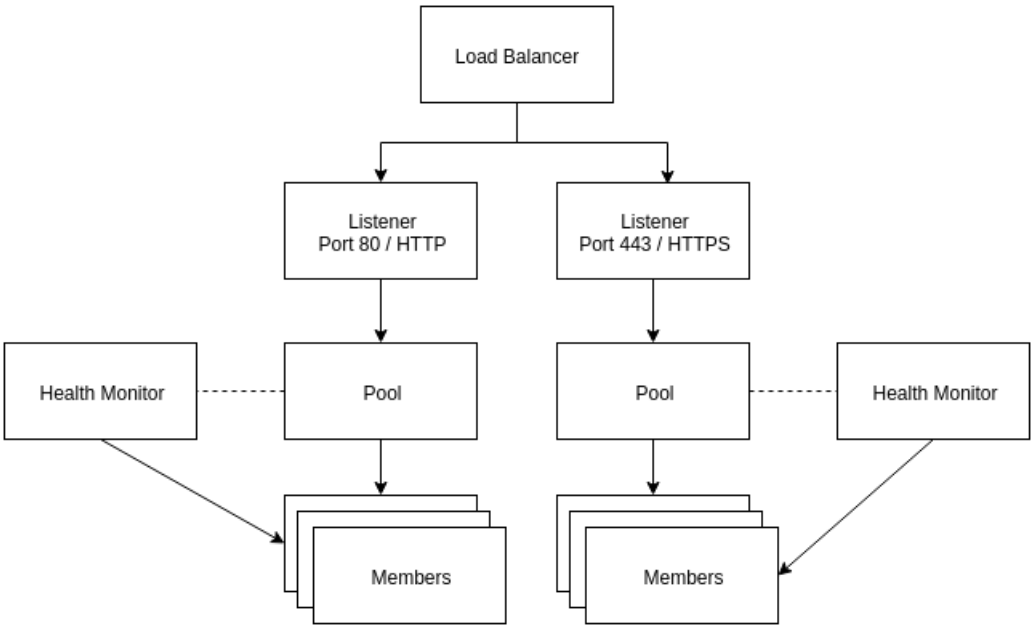

Figura 8.1: Diagramma concettuale di LBaaS v2 [18]

Load Balancer. Il load balancer occupa una porta di rete e ha un indirizzo IP assegnato da una subnet.

Listener. I load balancer possono accettare richieste su porte multiple e supportano diversi tipi di protocolli. Il listener è l'elemento del load balancer che resta in ascolto su una determinata porta utilizzando un protocollo prestabilito e inoltra la richiesta al pool selezionato in fase di configurazione.

Pool. Il pool è una lista di *membri* che contiene tutte le informazioni riguardante l'algoritmo da utilizzare per il load balancing. All'interno del pool sono specificati anche il protocollo e la porta da utilizzare per accedere alle risorse esposte dai membri.

Members. I membri sono server che forniscono le risorse richieste attraverso il load balancer. Ciascun membro è specificato dall'indirizzo IP e dalla porta che usa per fornire le risorse richieste; di conseguenza i membri possono essere macchine virtuali, container o server bare metal.

Health Monitor. Gli health monitor servono per monitorare lo stato dei membri di un pool. Se ad esempio un macchina non è più raggiungibile per

un qualsiasi motivo, l'health monitor se ne accorge e devia il traffico verso gli altri membri del pool. Gli health monitor sono associati ai pool.

8.1 Componenti

Nei capitoli seguenti verranno descritti i componenti di OpenStack che permettono la creazione di load balancer.

8.1.1 Octavia

Octavia è una soluzione di load balancing open source progettata per lavorare insieme a OpenStack.

Octavia esegue i suoi compiti gestendo una flotta di macchine, container o server bare metal, noti comunemente come *amphorae*, che vengono avviati su richiesta. Ciascuno di questi *amphorae* contiene al suo interno un'istanza di load balancer [29].

8.1.2 Barbican

Barbican è il Key Manager di OpenStack, ovvero il servizio che si occupa di memorizzare e gestire i dati segreti, come ad esempio chiavi simmetriche, chiavi asimmetriche, certificati, password e dati binari [1]. Come si può intuire, le sue funzionalità sono molto simili a quelle di Vault (descritto nel capitolo 5.2.1), però è necessario installare anche Barbican perché Octavia non ha un supporto diretto per Vault. È possibile però configurare Barbican in modo che utilizzi Vault come backend, in modo da avere una gestione unificata di tutti i *secrets*.

8.2 Installazione

8.2.1 Installazione manuale

Le istruzioni per l'installazione riportate in questo capitolo danno come presupposto che si disponga di un cloud OpenStack base già installato e funzionante.

Barbican. Il primo passo è installare i charm barbican e barbican-vault, che sono rispettivamente Barbican e il software che permette l'integrazione tra Barbican e Vault con le rispettive relazioni:

```
1 juju deploy barbican --channel yoga/stable --series focal --to lxd:3
2 juju deploy barbican-vault --channel yoga/stable --series focal
```

```
juju add-relation barbican rabbitmq-server
juju add-relation barbican keystone
juju add-relation barbican barbican-vault
juju add-relation barbican-vault:secrets barbican:secrets
juju add-relation vault:secrets barbican-vault:secrets-storage
```

Il secondo passo è installare un router MySQL per permettere a Barbican di collegarsi al cluster InnoDB e aggiungere le relazioni necessarie:

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

Octavia. Dopo l'installazione di Barbican si può procedere con quella di Octavia. Prima di procedere con l'installazione vera e propria è necessario però riconfigurare Neutron per abilitare il driver per l'estensione ML2. È necessario fare questo perché i gruppi di sicurezza di OpenStack hanno delle policy anti-spoofing che non permettono alle macchine virtuali di instradare il traffico attraverso loro stesse e, grazie a questa estensione, è possibile specificare dei flag che permettono di abilitare o disabilitare i filtri sulle porte [28]. Per abilitare l'estensione ML2 è necessario eseguire il seguente comando:

```
juju config neutron-api enable-ml2-port-security=True
```

Fatto questo si può procedere con l'installazione di Octavia:

```
juju deploy octavia --channel yoga/stable --series focal --to lxd:3
juju add-relation octavia rabbitmq-server
juju add-relation octavia keystone
juju add-relation octavia:neutron-openvswitch ovn-chassis:nova-compute
juju add-relation octavia neutron-api
```

Successivamente si può installare il router MySQL che permette la connessione al cluster InnoDB con le relative relazioni:

```
juju deploy mysql-router octavia-mysql-router --channel 8.0/stable --series
     focal
juju add-relation octavia-mysql-router:db-router mysql-innodb-cluster:
    db-router
```

```
juju add-relation octavia-mysql-router:shared-db octavia:shared-db
```

Infine è necessario installare Octavia Dashboard, il plugin per OpenStack Dashboard che mette a disposizione l'interfaccia grafica per configurare i load balancer:

```
juju deploy octavia-dashboard --channel yoga/stable --series focal
juju add-relation octavia-dashboard openstack-dashboard
```

8.2.2 Installazione in bundle

L'installazione in bundle prevede l'utilizzo di un file bundle analogo a quello visto nella sezione 6.3 riguardante l'installazione di OpenStack, con l'unica differenza che in questo caso il bundle è stato strutturato come overlay (per ulteriori dettagli vedere la sezione 4.2.2). È stata presa questa decisione per 2 motivi:

1. il load balancer non è un componente necessario per far funzionare OpenStack, quindi creando un overlay su può decidere se installarlo o meno
2. durante lo svolgimento del progetto l'installazione del load balancer è stata fatta successivamente alla prima installazione di OpenStack, quindi con un overlay è stato possibile eseguire un'installazione tramite bundle senza dover eliminare e ricreare il cloud già esistente

Utilizzando un overlay per il deployment ci si può trovare in due situazioni:

1. deve essere installato tutto il sistema (compreso il bundle principale)
2. si deve installare solo l'overlay su un sistema già esistente

Nel caso in cui si debba installare tutto il sistema sarà sufficiente eseguire il deployment del bundle aggiungendo l'opzione `--overlay` e specificando il nome del bundle overlay come nell'esempio seguente:

```
juju deploy ./bundle.yaml --overlay ./overlays/octavia.yaml
```

Nel caso in cui si voglia aggiungere l'overlay ad un'installazione già esistente sarà necessario esportare il modello di Juju in un file bundle e poi eseguire il deploy del modello aggiungendo l'overlay. Di seguito è riportato un esempio:

```
juju export-bundle --filename exported-bundle.yaml
juju deploy ./exported-bundle.yaml --overlay ./overlays/octavia.yaml
```

8.3 Configurazione iniziale

Il primo passo per configurare il load balancer di OpenStack è generare i certificate necessari per l'autenticazione e la comunicazione sicura tra gli *Amphorae* (le istanze del load balancer) e il sistema di controllo di Octavia [19]. Per fare questo è necessario eseguire i seguenti comandi:

```
mkdir -p demoCA/newcerts
touch demoCA/index.txt
touch demoCA/index.txt.attr

openssl genpkey -algorithm RSA -aes256 -pass pass:foobar -out
    issuing_ca_key.pem
openssl req -x509 -passin pass:foobar -new -nodes -key issuing_ca_key.pem \
    -config /etc/ssl/openssl.cnf \
    -subj "/C=US/ST=Somestate/O=Org/CN=www.example.com" \
    -days 365 \
    -out issuing_ca.pem

openssl genpkey -algorithm RSA -aes256 -pass pass:foobar -out
    controller_ca_key.pem
openssl req -x509 -passin pass:foobar -new -nodes \
        -key controller_ca_key.pem \
    -config /etc/ssl/openssl.cnf \
    -subj "/C=US/ST=Somestate/O=Org/CN=www.example.com" \
    -days 365 \
    -out controller_ca.pem
openssl req \
    -newkey rsa:2048 -nodes -keyout controller_key.pem \
    -subj "/C=US/ST=Somestate/O=Org/CN=www.example.com" \
    -out controller.csr
openssl ca -passin pass:foobar -config /etc/ssl/openssl.cnf \
    -cert controller_ca.pem -keyfile controller_ca_key.pem \
    -create_serial -batch \
    -in controller.csr -days 365 -out controller_cert.pem
cat controller_cert.pem controller_key.pem > controller_cert_bundle.pem
```

Successivamente si deve configurare il charm con i certificati appena generati. Per fare questo è sufficiente eseguire il seguente comando:

```
juju config octavia \
    lb-mgmt-issuing-cacert="$(base64 issuing_ca.pem)" \
    lb-mgmt-issuing-ca-private-key="$(base64 issuing_ca_key.pem)" \
    lb-mgmt-issuing-ca-key-passphrase=foobar \
    lb-mgmt-controller-cacert="$(base64 controller_ca.pem)" \
    lb-mgmt-controller-cert="$(base64 controller_cert_bundle.pem)"
```

Per completare la configurazione e l'attivazione di Octavia è necessario configurare le risorse (reti e macchine virtuali) che dovrà utilizzare. Dato che la gestione delle risorse di Octavia è automatizzata, per svolgere quest'ultima operazione è sufficiente eseguire il seguente comando:

```
juju run-action --wait octavia/0 configure-resources
```

Come detto in precedenza Octavia utilizza una serie di macchine virtuali gestite in maniera automatica per istanziare i load balancer; di conseguenza è necessario anche fornire delle immagini di sistemi operativi che possano essere utilizzate per queste macchine virtuali. È possibile scaricare e importare queste immagini automaticamente eseguendo i seguenti comandi:

```
juju run-action --wait glance-simplestreams-sync/0 sync-images
juju run-action --wait octavia-diskimage-retrofit/leader retrofit-image
```

8.4 Utilizzo

Prima di poter utilizzare le funzionalità messe a disposizione dal load balancer è necessario assicurarsi che l'utente che si sta utilizzando abbia i permessi per accedere a queste funzionalità; Octavia infatti crea in maniera automatica alcuni ruoli che possono essere utilizzati per garantire diversi tipi di accesso e di privilegi agli utenti. I ruoli sono i seguenti:

- **load-balancer_observer**: garantisce privilegi di sola lettura
- **load-balancer_global_observer**: garantisce privilegi di sola lettura estesi anche a risorse appartenenti ad altri utenti
- **load-balancer_member**: garantisce privilegi di lettura e scrittura sul load balancer

- **load-balancer_quota_admin**: garantisce privilegi di amministrazione solamente sulle quota API, ovvero sulle API che permettono di impostare il massimo numero di risorse utilizzabili da un utente o da un progetto

- **load-balancer_admin**: garantisce privilegi di amministrazione su tutte le risorse di load balancing, incluse quelle appartenenti ad altri utenti

Per poter aggiungere uno di questi ruoli ad un utente è sufficiente eseguire il seguente comando:

```
openstack role add --user-domain domain1 --user user1 \
   --project-domain domain1 --project project1 \
   load-balancer_member
```

8.4.1 Creazione di un load balancer

Tutte le operazioni riguardanti i load balancer possono essere eseguite tramite la pagina raggiungibile dal menu a tendina *Project* > *Network* selezionando la voce *Load Balancers*.

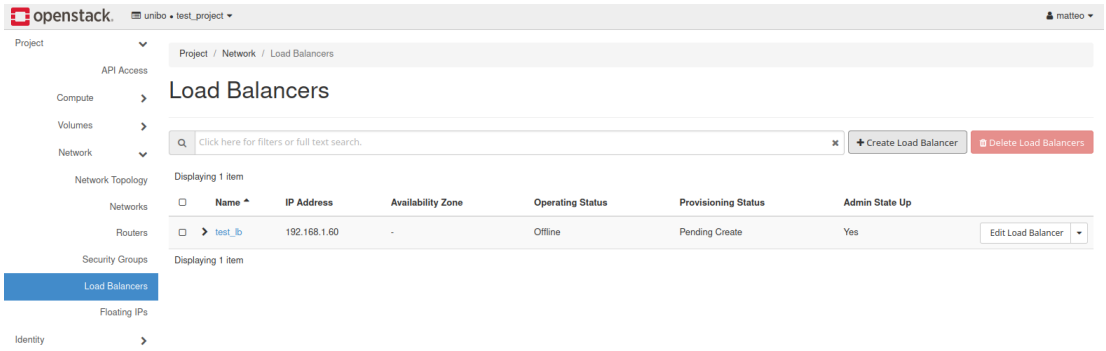

Figura 8.2: Interfaccia di gestione dei load balancers

Per creare un nuovo load balancer si deve cliccare il bottone *Create load balancer*; in questo modo verrà aperta una finestra di dialogo che consentirà all'utente di inserire tutti i parametri di configurazione. Gli unici parametri obbligatori sono il nome e la subnet alla quale collegare l'interfaccia di rete. È possibile inoltre creare il listener e il pool direttamente da questa finestra di dialogo, però questi elementi verranno trattati nella sezione 8.4.2.

8.4.2 Gestione del load balancer

Cliccando sul nome di uno dei load balancer è possibile accedere alla pagina di gestione, dalla quale si possono visualizzare tutte le informazioni riguardanti il load balancer selezionato e gestire i listener e i pool.

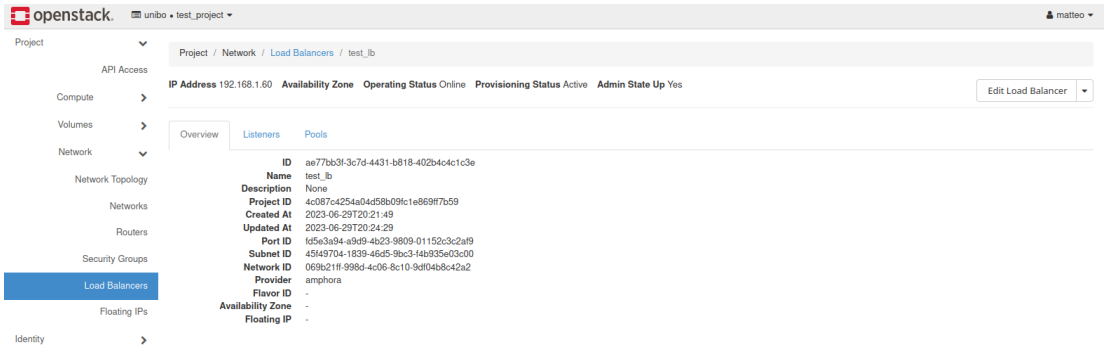

Figura 8.3: Interfaccia configurazione di un load balancer

Listener. È possibile aggiungere e modificare i listener dall'interfaccia di configurazione del load balancer andando nella scheda *Listeners*. Per creare un nuovo listener si vede cliccare sul bottone *Create Listener* e inserire tutti i parametri di configurazione richiesti. Nello specifico i parametri sono i seguenti:

- **Name**: nome del listener
- **Protocol**: protocollo su cui il listener resta in ascolto
- **Port**: porta sulla quale il listener resta in ascolto
- **Client Data Timeout**: timeout del client (espresso in millisecondi)
- **TCP Inspect Timeout**: tempo massimo di attesa di ulteriori pacchetti TCP per l'ispezione del contenuto
- **Member Connect Timeout**: timeout di connessione per i membri dei pool
- **Member Data Timeout**: timout di inattività per i membri dei pool
- **Connection Limit**: numero massimo di connessioni simultanee (si può disabilitare inserendo -1)

- **Allowed Cidrs**: permette di restringere le connessioni solamente alle subnet specificate

È possibile inoltre creare un pool contemporaneamente al listener oppure associarene uno in seguito.

Pool. Per creare e gestire i pool si deve aprire la scheda *Pools* all'interno dell'interfaccia di gestione del load balancer. Da qui è possibile midificare o pool esistenti oppure crearne uno nuovo cliccando sul bottone *Create Pool*. I parametri richiesti per la creazione di un nuovo pool sono:

- **Name**: nome del pool
- **Algorithm**: algoritmo da utilizzare per il load balancing (tutti gli algoritmi sono descritti nel paragrafo seguente)
- **Protocol**: protocollo utilizzato dai membri del pool
- **Session Persistance**: permette di indicare il tipo di persistenza della sessione da utilizzare
- **TLS Enabled**: se attivato le connessioni tra il load balancer e i membri vengono cifrate

Algoritmi di load balancing. Il load balancer utilizzato durante lo svolgimento di questo progetto implementa 3 diversi algoritmi di load balancing:

- Least Connections
- Round Robin
- Source IP

L'algoritmo *Least Connections* inoltra la richiesta del client al membro del pool con il minor numero di connessioni attive al momento della richiesta. Questo può essere molto utile nel caso in cui le richieste all'applicazione ospitata dietro il load balancer abbiano tempi di risposta molto diversi (ad esempio il download di file) e nel caso in cui tutti i membri del pool abbiano risorse hardware molto simili.

L'algoritmo *Round Robin* inoltra le richieste dei client a rotazione a tutti i membri del pool. Questo algoritmo può essere molto utile in casi in cui tutte le richieste sono molto simili come tempi di risposta e complessità delle operazioni da eseguire, ma potrebbe essere inefficiente nel caso in cui le richieste

hanno tempi di risposta molto diversi o richiedono l'esecuzione di calcoli molto complessi.

L'algoritmo *Source IP* consiste nell'indirizzare tutte le richieste provenienti dallo stesso indirizzo IP al medesimo membro del pool. Questo algoritmo è utile specialmente quando è imperativo che il client si connetta sempre allo stesso server per ciascuna richiesta.

Tipi di persistenza della sessione. Octavia implementa 3 tecniche di persistenza della sessione:

- Source IP
- HTTP Cookie
- APP Cookie

Come si può intuire, se si utilizza la tecnica *Source IP* la sessione viene mantenuta in base all'indirizzo IP del client. Utilizzando invece la tecnica *HTTP Cookie* viene generato in automatico un cookie HTTP aggiuntivo che permette al load balancer di capire a quale membro del pool il client si è connesso in precedenza. La tecnica *APP Cookie* è molto simile come funzionamento alla precedente, con l'unica differenza che il cookie non viene generato in automatico ma viene impostato dall'applicazione ospitata dietro il load balancer e, durante la configurazione del pool, deve essere specificato qual è il nome di questo cookie.

Capitolo 9

Terraform

Terraform è un Infrastructure as Code (IaC) tool, ovvero uno strumento che permette di definire infrastrutture cloud o on-premises tramite file di configurazione semplici da scrivere e da comprendere e che possono essere versionati, modificati e condivisi in base alle esigenze dell'utente. Tramite Terraform è possibile gestire sia componenti di basso livello (ad esempio server, reti, storage, ecc.) sia componenti di livello più alto, come ad esempio record DNS.

9.1 Funzionamento

Terraform crea e gestisce le risorse sulle piattaforme cloud o su altre piattaforme tramite le API fornite da tali servizi; purtroppo queste API non seguono uno standard e ogni provider implementa le proprie in maniera totalmente arbitraria, rendendo tendenzialmente difficile e dispendioso sviluppare tool che possano automatizzare le operazioni su più provider. Per questo motivo all'interno di Terraform esistono dei plugin, chiamati *Terraform Provider* che hanno il compito di astrarre la connessione al provider (come mostrato in figura 9.1 e di fornire all'utente una serie di API di configurazione più o meno standard. Esistono migliaia di provider sviluppati sia dalla community che da HashiCorp (l'azienda madre di Terraform); tra questi sono presenti anche quelli che permettono l'interfacciamento con le piattaforme cloud più utilizzate (AWS, GCP, Azure, ...). È possibile cercare e consultare la documentazione di tutti i provider sul *Terraform Registry*[1].

[1] Terraform Registry: `https://registry.terraform.io`

Figura 9.1: Schema di funzionamento di Terraform

9.2 Utilizzo

9.2.1 Workflow

L'utilizzo di Terraform modifica notevolmente il workflow di chi deve progettare e realizzare un'infrastruttura cloud riducendo sensibilmente i tempi richiesti per la configurazione dell'infrastruttura stessa e per la manutenzione. Il workflow con Terraform è composto infatti da 3 step (come mostrato in figura 9.2):

- **Write**: si definisce l'infrastruttura desiderata tramite i file di configurazione di Terraform; questa infrastruttura può includere anche provider multipli

- **Plan**: Terraform individua quali elementi della configurazione fornita devono essere creati o distrutti; questa decisione viene presa in base alla configurazione e alle risorse dell'infrastruttura già esistente

- **Apply**: una volta che l'utente ha approvato le modifiche proposte, Terraform esegue le operazioni di aggiornamento creando e/o distruggendo risorse

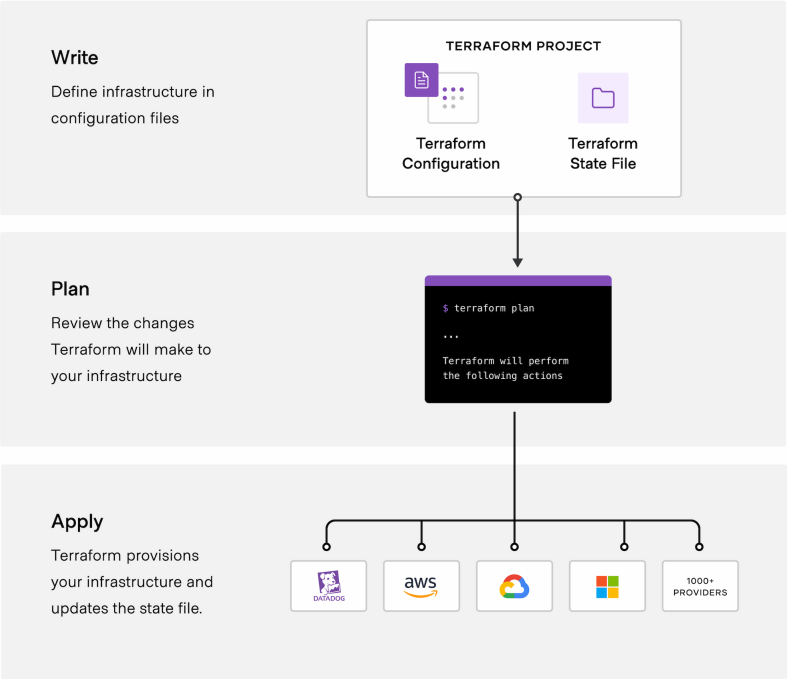

Figura 9.2: Workflow con Terraform

9.2.2 Progetti con Terraform

Un progetto Terraform è la configurazione di un'infrastruttura che si vuole modellare; questa infrastruttura può appoggiarsi su più provider e quindi essere distribuita su più cloud. I progetti Terraform possono assumere diverse strutture più o meno complesse e generalmente tale struttura varia in base alla complessità dell'infrastruttura da definire.

Per un progetto molto semplice è sufficiente creare una cartella all'interno della quale si scrive il file `main.tf` che contiene la configurazione dell'infrastruttura che volgiamo realizzare. Ogni volta che si crea un nuovo progetto o si aggiungono provider ad uno esistente è necessario eseguire il comando `terraform init` in modo che vengano scaricati i provider necessari al progetto.

Una volta terminata la configurazione si possono verificare le modifiche eseguite lanciando il comando `terraform plan`; l'output mostrerà tutte le operazioni che Terraform intende eseguire per applicare la configurazione definita partendo dalle risorse già esistenti. Per applicare le modifiche si deve lanciare il comando `terraform apply` che, dopo aver mostrato nuovamente le operazioni che intende eseguire, chiederà una conferma all'utente e poi procederà con l'aggiornamento dell'infrastruttura.

Se si vuole cancellare completamente tutta l'infrastruttura creata è sufficiente lanciare il comando `terraform destroy` che, dopo aver chiesto conferma all'utente, cancellerà tutte le risorse associate al progetto.

9.2.3 Configurazioni di Terraform: HCL

HCL (HashiCorp Configuration Language) è un toolkit sviluppato direttamente da HashiCorp che permette di creare linguaggi di configurazione strutturati che siano facilmente comprensibili dagli essere umani e al contempo facilmente interpretabili dalle macchine. HCL supporta sia una sintassi nativa che il formato JSON (tendenzialmente utilizzato quando i file di configurazione vengono generati automaticamente). Tutti i file di configurazione di Terraform utilizzano una sintassi basata su HCL.

Argomenti e Blocchi. Il linguaggio di Terraform si basa su due elementi principali: argomenti e blocchi. Gli argomenti sono l'equivalente delle variabili in un linguaggio di programmazione tradizionale, ovvero permettono di associare un valore ad un nome definito dall'utente, come nell'esempio seguente:

```
image_id = "abc123"
```

I blocchi sono invece contenitori per altro contenuto; ciascun blocco è definito da un tipo e, in base ad esso, cambia il numero di etichette da inserire dopo la definizione del tipo del blocco. Nell'esempio che segue è definito un blocco di tipo `resource` che accetta 2 etichette (in questo caso `"aws_instance"` e `"example"`):

```
resource "aws_instance" "example" {
    ...
}
```

Espressioni condizionali. Le espressioni condizionali permettono di selezionare uno tra due valori utilizzando un'espressione booleana; la sintassi di queste espressioni è la seguente:

```
condition ? true_val : false_val
```

Gli operatori logici e di comparazione utilizzabili all'interno della condizione sono analoghi a quelli utilizzabili nel linguaggio C (`==`, `!=`, `<=`, `&&`, `!`, ...)

Espressioni iterative. All'interno del linguaggio di Terraform è integrata l'espressione `for` che, analogamente all'omonimo costrutto dei linguaggi di programmazione, serve per eseguire delle iterazioni. Tramite l'utilizzo di questa espressione è possibile iterare sia le liste che le mappe; di seguito sono riportati due esempi:

```
# iterazione di una lista
[for s in var.list : upper(s)]
# iterazione di una mappa
[for k, v in var.map : length(k) + length(v)]
```

Blocchi dinamici. I blocchi dinamici agiscono in maniera simile alle espressioni `for`, con la differenza che l'output è un insieme di blocchi. Questo tipo di blocco può essere usato solamente all'interno di altri blocchi e, nello specifico, all'interno di blocchi di tipo `resource`, `data`, `provider` o `provisioner`. Di seguito è riportato un esempio di configurazione contenente un blocco dinamico:

```
resource "aws_elastic_beanstalk_environment" "tfenvtest" {
  name                = "tf-test-name"
  application         = "${aws_elastic_beanstalk_application.tftest.name}"
  solution_stack_name = "64bit Amazon Linux 2018.03 v2.11.4 running Go
    1.12.6"

  dynamic "setting" {
    for_each = var.settings
    content {
      namespace = setting.value["namespace"]
      name = setting.value["name"]
      value = setting.value["value"]
    }
  }
}
```

In questo caso l'etichetta `"setting"` indica il tipo di blocco, l'argomento `for_each` indica i valori che devono essere iterati per la generazione dei blocchi e il blocco `content` definisce quale deve essere la struttura dei blocchi generati.

9.3 Terraform e OpenStack

9.3.1 Introduzione

Durante lo svolgimento di questo progetto io e il mio collega Sauro abbiamo deciso di prendere due direzioni diverse per quanto riguarda lo studio dell'integrazione tra Terraform e OpenStack: io ho approcciato il provisioning delle risorse tramite Terraform dal punto di vista di un utente che utilizza OpenStack come cloud provider per ospitare le proprie applicazioni, mentre il mio collega si è concentrato maggiormente sul provisioning delle risorse dal punto di vista dell'amministratore del cloud.

L'approccio che ho utilizzato per questo studio è stato iterativo: ho iniziato con un progetto molto semplice che consisteva nel creare una sola istanza per poi arrivare alla terza iterazione ad un progetto molto più complesso che utilizza tutte le possibili risorse offerte dal nostro cloud OpenStack e simula in maniera abbastanza realistica quello che potrebbe essere un caso d'uso reale. Nelle sezioni seguenti saranno descritti nel dettaglio tutti i progetti Terraform che ho realizzato.

9.3.2 Progetto 1: istanza

Come accennato in precedenza il primo progetto consiste solamente nel deployment di un'istanza tramite Terraform. L'unico file presente è `main.tf` che al suo interno contiene 3 blocchi che verranno riportati e descritti di seguito.

Blocco terraform. Il primo blocco è di tipo `terraform`, ovvero quello che contiene le impostazioni di Terraform e la lista dei provider necessari per il deployment dell'infrastruttura.

```
terraform {
  required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}
```

Listato 9.1: Blocco di tipo `terraform`

Blocco provider. Il secondo blocco è di tipo `provider` e permette di definire le informazioni di connessione ai provider scelti (in questo caso OpenStack).

```
provider "openstack" {
  user_name   = "user"
  password    = "password"
  auth_url    = "https://10.0.0.7:5000/v3" # keystone endpoint URL
  domain_name = "domain"
  tenant_name = "project1" # project name
  insecure    = true       # allow self-signed SSL cert
}
```

Listato 9.2: Blocco di tipo `provider`

Come si può vedere dal listato 9.2 il provider OpenStack accetta i seguenti parametri:

- `user_name`: nome dell'utente con il quale accedere alle API di OpenStack
- `password`: password dell'utente
- `auth_url`: endpoint di Keystone, il servizio che gestisce le identità su OpenStack (per altri dettagli vedere sezione 5.1.4)
- `domain_name`: nome del dominio all'interno del quale è definito l'utente
- `tenant_name`: nome del progetto all'interno del quale devono essere create le risorse definite tramite Terraform
- `insecure`: permette di specificare se autorizzare o meno le connessioni HTTPS se i certificati SSL non sono validi

Blocco resource. Il terzo e ultimo blocco è di tipo `resource` ed è stato utilizzato per definire i dettagli dell'istanza da creare.

```
resource "openstack_compute_instance_v2" "test-server" {
  name            = "test-server"
  image_id        = "f2aa2ed8-cb58-4152-b881-5e3a3c24fe30"
  flavor_id       = "cf492d92-2496-4f62-9f3f-d160758c812c"
  key_pair        = "terraform"
  security_groups = ["default"]

  network {
    name = "internal"
  }
}
```

Listato 9.3: Blocco di tipo `resource`

Come si può vedere dal listato 9.3 il tipo di risorsa da utilizzare per creare un'istanza è `openstack_compute_instance_v2` e, in questo caso, il nome assegnato alla risorsa è `test-server`. Gli argomenti per configurare l'istanza sono gli stessi trattati nella sezione 7.4.4 quando si parla dell'interfaccia che permette di creare una nuova istanza.

Come si può notare sempre dal listato 9.3 la configurazione della rete è definita tramite blocchi all'interno della risorsa; in questo caso il blocco è molto semplice perché viene specificato solo il nome della rete a cui l'istanza deve essere collegata.

Deployment dell'infratruttura Lanciando il comando il comando `terraform apply` partendo da un progetto senza nessuna risorsa già creata si ottiene l'output mostrato in figura 9.3 (ritagliato per motivi di spazio). Come si può notare Terraform mostra tutte le modifiche che intende fare all'infrastruttura e chiede conferma all'utente prima di eseguire qualsiasi azione. Una volta data la conferma Terraform procederà con la creazione delle risorse mostrate e attenderà fino a quando ciascuna di esse sarà attiva.

```
# openstack_compute_instance_v2.test-server will be created
+ resource "openstack_compute_instance_v2" "test-server" {
    + access_ip_v4        = (known after apply)
    + access_ip_v6        = (known after apply)
    + all_metadata        = (known after apply)
    + all_tags            = (known after apply)
    + availability_zone   = (known after apply)
    + flavor_id           = "f5f660da-d1bb-4497-9db2-7265f0c9919d"
    + flavor_name         = (known after apply)
    + force_delete        = false
    + id                  = (known after apply)
    + image_id            = "4dc32112-ec68-4f92-90fe-44f4a0acf404"
    + image_name          = (known after apply)
    + key_pair            = "Matteo-PC"
    + name                = "test-server"
    + power_state         = "active"
    + region              = (known after apply)
    + security_groups     = [
        + "default",
      ]
    + stop_before_destroy = false

    + network {
        + access_network = false
        + fixed_ip_v4    = (known after apply)
        + fixed_ip_v6    = (known after apply)
        + floating_ip    = (known after apply)
        + mac            = (known after apply)
        + name           = "internal"
        + port           = (known after apply)
        + uuid           = (known after apply)
      }
  }
```

Figura 9.3: Pianificazione delle modifiche all'infrastruttura

9.3.3 Progetto 2: infrastruttura completa

Il secondo progetto sviluppato è ovviamente un'evoluzione del primo e prevede la configurazione tramite Terraform di un'infrastruttura completa. Per infrastruttura completa si intende che la configurazione deve includere sia le istanze che tutta la parte di networking.

In questo caso la struttura del progetto rimane abbastanza semplice ma sono stati aggiunti alcuni file in modo mantenere tutto il codice più organizzato. Di seguito è riportato uno schema della struttura del progetto:

```
02_full_infrastructure
├── example.tfvars
├── instances.tf
├── main.tf
└── networking.tf
```

In appendice B.2 è possibile vedere tutti i file di questo progetto compresi quelli che non vengono mostrati qui per motivi di spazio.

Variabili di input. All'interno di questo progetto sono state aggiunte alcune variabili di input. Queste variabili sono dichiarate all'interno del file `main.tf` nel seguente modo:

```
provider "openstack" {
  user_name   = var.auth.user
  password    = var.auth.password
  auth_url    = var.auth.keystone_url
  domain_name = var.auth.domain
  tenant_name = var.auth.project # project name
  insecure    = true             # allow self-signed SSL cert
}

variable "auth" {
  type = object({
    user         = string
    password     = string
    keystone_url = string
    project      = string
    domain       = string
  })
}

variable "network" {
  type = object({
    external_network_id   = string
    external_network_name = string
    dns_nameservers       = list(string)
  })
```

```
}

variable "compute" {
  type = object({
    image_id  = string
    flavor_id = string
    ssh_key   = string
  })
}
```

Come si può notare ciascuna variabile dichiarata è di tipo `object` e al suo interno contiene altre variabili di tipi differenti. Quando si esegue il deployment tramite Terraform è necessario specificare un file tramite l'argomento `-varfile` che contiene le assegnazioni dei valori a queste variabili. All'interno del progetto è presente infatti il file `example.tfvars` che contiene appunto un esempio di come deve essere strutturato un file con tutte le variabili di questo progetto assegnate.

```
auth = {
  user         = "matteo"                   # openstack user
  password     = "ubuntu"                   # user's password
  keystone_url = "https://10.0.0.7:5000/v3" # openstack identity service
    url
  project      = "project2"                 # openstack project
  domain       = "domain"                   # openstack domain
}

network = {
  external_network_id   = "68d5f950-bd27-43b7-8ca6-4fc96341a6dd" #public
    network id
  external_network_name = "ext_net"                              #public
    network name
  dns_nameservers       = ["8.8.8.8", "8.8.4.4"]                 # default
    dns servers
}

compute = {
  image_id  = "76c6c0f5-91cc-4446-bc3f-00bd7ee3fb08" # default compute
    image id
  flavor_id = "51c1c7b0-1811-4a61-bc55-fb82c86d24cb" # flavor id
  ssh_key   = "terraform"                            # ssh key name
}
```

La sintassi generica per poter accedere alle variabili definite all'interno del progetto è la seguente: `var.<nome_variabile>`; dato che le variabili definite qui contengono a loro volta altre variabili per accedere ad esempio al nome dell'utente la sintassi è la seguente: `var.auth.user`.

Networking. Tutte le configurazioni riguardanti la rete dell'infrastruttura si trovano nel file `networking.tf`. Al suo interno ci sono 5 blocchi di tipo `resource` utilizzati per configurare le seguenti risorse:

- network
- subnet
- security group da assegnare all'istanza
- router
- interfaccia che collega il router alla subnet definita in precedenza

Nel listato 9.4 è riportato il blocco che configura il collegamento del router alla subnet tramite una nuova interfaccia.

```
resource "openstack_networking_router_interface_v2" "tf_router_interface" {
  router_id = openstack_networking_router_v2.tf_router.id
  subnet_id = openstack_networking_subnet_v2.tf_subnet.id
}
```

Listato 9.4: Configurazione dell'interfaccia del router che lo collega alla subnet

Come si può notare gli unici due argomenti sono l'ID del router e l'ID della subnet a cui l'interfaccia si deve collegare. Questo esempio è interessante perché mostra come è possibile utilizzare gli argomenti di determinate risorse per configurarne altre. Nello specifico la sintassi utilizzata per accedere agli argomenti di altre risorse è la seguente: `<tipo_risorsa>.<etichetta_risorsa>.<parametro>`. Se ad esempio si volesse accedere all'argomento `router_id` dell'interfaccia creata dall'esempio sopra la sintassi sarebbe la seguente: `openstack_networking_router_interface_v2.tf_router_interface.router_id`.

Istanza Tutte le configurazioni riguardanti l'istanza si trovano all'interno del file `instances.tf`. Il blocco riguardante l'istanza stessa è molto simile a quello del progetto precedente con una sola eccezione: in questo caso è stato aggiunto un altro blocco al suo interno chiamato `block_device` (riportato nel listato 9.5) che permette di collegare un volume all'istanza.

```
  block_device {
    uuid                  = var.compute.image_id
    source_type           = "image"
    volume_size           = 30
    boot_index            = 0
    destination_type      = "volume"
    delete_on_termination = true
  }
```

Listato 9.5: Configurazione di un volume collegato all'istanza

All'interno del file `instances.tf` è anche presente la configurazione del floating IP e l'assegnazione di quest'ultimo all'istanza creata.

9.3.4 Progetto 3: Proof of Concept

Il terzo progetto è l'ultima iterazione e sfrutta tutte le funzionalità offerte dal cloud OpenStack installato. Lo scopo è quello di simulare il deployment di un'applicazione web reale creando più istanze e configurando un load balancer in modo che divida le richieste tra di esse. In figura 9.4 è raffigurata la topologia di rete (escluso il load balancer); come si può vedere sono presenti 3 istanze collegate ad una rete privata alla quale è collegato anche un router che permette alle suddette istanze di accedere a internet. Il load balancer è configurato in modo da essere collegato sia alla rete pubblica che a quella privata così da poter essere raggiungibile dai client e potersi collegare alle istanze tramite la rete interna.

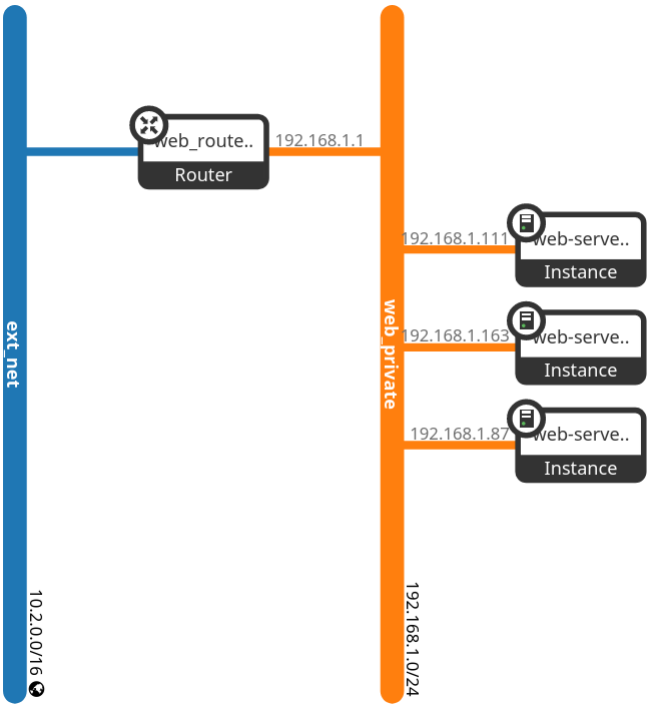

Figura 9.4: Topologia di rete del progetto

Il progetto ha la seguente struttura:

```
03_proof_of_concept
├── example.tfvars
├── init-instance.sh
├── instances.tf
├── load_balancer.tf
├── main.tf
├── networking.tf
└── output.tf
```

Tutti i file del progetto compresi quelli non riportati in questa sezione possono essere consultati in appendice B.3

Istanze. La configurazione delle istanze si trova all'interno del file `instances.tf` che, rispetto al progetto precedente, presente 2 aggiunte: l'argomento `count` che permette di specificare il numero di istanze da creare e il parametro `user_data` che permette di specificare uno script bash che sarà poi eseguito dalle macchine all'avvio. La porzione di configurazione dell'argomento `user_data` è la seguente:

```
user_data = templatefile("${path.module}/init-instance.sh", {
  instance_idx = count.index
})
```

In questo caso viene utilizzata la funzione `templatefile` per caricare uno script da file e nello specifico carica il file chiamato `init-instance.sh` contenuto nella directory principale del modulo; il secondo argomento di tale funzione è un oggetto che permette di inoltrare variabili allo script (in questo caso `instance_idx`. Lo script `init-instance.sh` è molto semplice ed è riportato di seguito:

```
#!/bin/bash
apt-get update
apt-get install -y apache2
echo "<DOCTYPE html>
<html lang=\"en\">
    <body>
        <h1>Hello, this is instance #${instance_idx}</h1>
    </body>
</html>" > /var/www/html/index.html
```

Il suo compito è quello di installare Apache Web Server e di creare un file `index.html` all'interno del quale viene scritto l'indice della macchina appena creata. In questo modo ciascuna macchina creata avrà un indice diverso e, accedendo attraverso il load balancer, sarà possibile capire a quale macchina ci si sta connettendo e verificare che l'algoritmo di load balancing stia funzionando come ci si aspetta.

Load Balancer. Tutte le configurazione del load balancer si trovano all'interno del file `load_balancer.tf`. Di seguito è riportata la configurazione del pool che potrebbe essere interessante perché contiene un blocco dinamico:

```
# targets pool
resource "openstack_lb_pool_v2" "web_lb_pool" {
  name            = "web_lb_pool"
  protocol        = "HTTP"
  lb_method       = "ROUND_ROBIN"
  loadbalancer_id = openstack_lb_loadbalancer_v2.web_lb.id
}

# pool members
resource "openstack_lb_members_v2" "web_lb_pool_members" {
  pool_id = openstack_lb_pool_v2.web_lb_pool.id

  # iterate all instances and add their IP address to the pool
  dynamic "member" {
    for_each = openstack_compute_instance_v2.web-server
    iterator = instance
    content {
      address       = instance.value.network[0].fixed_ip_v4
      protocol_port = 80
      weight        = 1
    }
  }
}
```

Il primo blocco di questa configurazione è quello che definisce il pool; si può notare come l'algoritmo utilizzato per il load balancing è round robin e che il protocollo specificato per la connessione agli host è HTTP. Il secondo blocco invece è la configurazione dei membri del pool; al suo interno si trova un altro blocco dinamico chiamato `"member"` che itera le istanze creando per ciascuna un blocco contenente come argomenti l'indirizzo IP, la porta su cui è esposto il servizio e il peso (utilizzato durante il load balancing per dare priorità di connessione ad una macchine rispetto alle altre).

Sono inoltre configurati un listener sulla porta 80 con protocollo HTTP e un health monitor che esegue periodicamente richieste HTTP alle istanze per verificare che siano ancora online e che il web server stia funzionando correttamente.

Output. Terraform permette di definire un output che viene generato una volta completate le modifiche e permette di visualizzare alcune informazioni potenzialmente molto utili sull'infrastruttura appena creata. All'interno del file `output.tf` sono presenti due blocchi di tipo output:

```
output "loadbalancer_ip" {
  value = openstack_networking_floatingip_v2.web_lb_floating_ip.address
}

output "instances_info" {
  description = "Instances Informations"
  value = [for instance in openstack_compute_instance_v2.web-server : {
    id   = instance.id
    name = instance.name
    ip   = instance.network[0].fixed_ip_v4
  }]
}
```

Il primo blocco mostra il floating IP assegnato al load balancer mentre il secondo mostra i dettagli di tutte le istanze create. In figura 9.5 è mostrato un esempio di quello che viene scritto in output dichiarando questi due blocchi.

```
Outputs:

instances_info = [
  {
    "id" = "2db1bb5d-31ee-4d37-af0a-ad21f76b3e84"
    "ip" = "192.168.1.111"
    "name" = "web-server-0"
  },
  {
    "id" = "b3545fe9-5c98-4e11-a0ef-63fdbbb35d3f"
    "ip" = "192.168.1.87"
    "name" = "web-server-1"
  },
  {
    "id" = "9634ba9c-1ea0-4de7-beff-76119ad2f593"
    "ip" = "192.168.1.163"
    "name" = "web-server-2"
  },
]
loadbalancer_ip = "10.2.1.160"
```

Figura 9.5: Output di terraform

Capitolo 10

Conclusioni e sviluppi futuri

Si può affermare di aver ampiamente raggiunto gli obbiettivi posti all'inizio del progetto, riuscendo a realizzare un cloud privato funzionante basato su OpenStack e a realizzare infrastrutture appoggiate a tale cloud utilizzando esclusivamente Terraform.

OpenStack. OpenStack si è rivelato essere una piattaforma estremamente versatile e personalizzabile, adatta a molteplici esigenze e casi d'uso grazie alla sua modularità e alle funzionalità che offre. OpenStack offre infatti un set di funzionalità paragonabile ad un qualsiasi cloud provider e, in base alla mia esperienza, addirittura superiore ad altri sistemi esistenti che permettono di realizzare cloud privati, sia a pagamento che open source (ad esempio VMware vSphere o Proxmox).

L'unica criticità che ho riscontrato in OpenStack è legata al numero di servizi richiesti per far funzionare il cloud; sono infatti presenti numerosi servizi che definirei di "supporto", ovvero che servono per far funzionare la piattaforma ma non aggiungono alcun tipo di funzionalità per l'utente finale. Questi servizi causano un overhead di risorse che, nel caso di deployment di piccole dimensioni e con poche risorse hardware disponibili, potrebbe essere critico o addirittura determinate l'esclusione di OpenStack dalla lista delle possibili piattaforme su cui basare il cloud.

Deployment del cloud. I sistemi di supporto al deployment utilizzati (MAAS e Juju) hanno notevolmente facilitato l'installazione e la configurazione, creando un livello di astrazione aggiuntivo tra l'utente e i singoli servizi. È bastato infatti scrivere poche linee di configurazione per installare tutto il cloud senza la necessità intervenire sui singoli file di configurazione di ciascun servizio.

Questa ulteriore astrazione porta però con sé una criticità: è molto complicato e addirittura sconsigliato da Juju andare a modificare manualmente i

file di configurazione dei singoli servizi; questo può essere un problema specialmente nei casi in cui si tratta di servizi distribuiti su più macchine, come ad esempio Ceph o Nova.

Nel caso di Ceph per esempio, è necessario che tutte le macchine abbiano esattamente lo stesso numero di dischi configurati nello stesso modo perché, come visto nella sezione 6.2, i device assegnati a Ceph vengono definiti globalmente e non possono essere modificati per una singola macchina.

Questa limitazione nella configurazione porta ad avere la necessità di installare tutti i servizi dello stesso tipo su macchine molto simili se non identiche.

Terraform. Personalmente prima dell'inizio di questo progetto di tesi non conoscevo tool come Terraform o simili; nonostante questo è stato relativamente semplice utilizzarlo grazie anche alla documentazione molto esaustiva fornita. Terraform si è rivelato essere uno strumento estremamente potente e utile, che permette realizzare e modificare infrastrutture molto complesse in maniera abbastanza semplice e sicuramente più veloce rispetto ad eseguire tutte le operazioni manualmente.

10.1 Sviluppi futuri

La prima attività da menzionare quando si parla dei possibili sviluppi futuri è sicuramente verificare che la metodologia di deployment utilizzata durante questo progetto sia ancora valida per le versioni successive di OpenStack. Dato il livello di astrazione fornito dai sistemi di deployment utilizzati mi sento abbastanza sicuro nel dire che dovrebbe bastare la modifica di alcuni parametri all'interno dei file bundle di Juju per poterli riutilizzare per il deployment di una versione successiva.

Un'altra attività fondamentale sarebbe quella di verificare lo stato dei dischi all'interno delle macchine e, nel caso siano danneggiati, sostituirli e rifare il deployment del cloud. Questo perché durante lo svolgimento del progetto sono sorti alcuni problemi di responsività del sistema e, dato il fallimento del test S.M.A.R.T. su alcune macchine, si presume che alcuni dischi abbiano dei settori danneggiati e che ciò influisca in maniera negativa sulle performance.

Altre possibili attività potrebbero riguardare l'installazione e la configurazione di servizi aggiuntivi di OpenStack, come ad esempio Zun che permette di gestire container direttamente da OpenStack senza dover creare istanze, oppure Trove che implementa un servizio di database as a service. Mi sembra chiaro ormai che se si vogliono esplorare altri componenti e funzionalità di OpenStack le possibilità sono moltissime e c'è l'imbarazzo della scelta.

Bibliografia

[1] Documentazione Barbican, https://docs.openstack.org/barbican/latest/

[2] Documentazione Ceph, https://docs.ceph.com/en/quincy/, ultimo accesso 23 Gennaio 2023.

[3] Charmhub - The Open Operator Collection, https://charmhub.io/, ultimo accesso 26 Gennaio 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] Documentazione LBaaS, https://docs.openstack.org/mitaka/networking-guide/config-lbaas.html

[19] Guida all'installazione del load balancer, https://docs.openstack.org/charm-guide/latest/admin/networking/load-balancing.html, ultimo accesso 26 Giugno 2023

[20] Errore nell'inizializzazione automatica di LXD, https://discuss.linuxcontainers.org/t/2148, ultimo accesso 31 Gennaion 2023.

[21] Metal as a Service, MAAS, https://maas.io/, ultimo accesso 20 Dicembre 2022.

[22] MAAS glossary, https://maas.io/docs/maas-glossary, ultimo accesso 23 Gennaio 2023.

[23] How MAAS works, https://maas.io/how-it-works, ultimo accesso 23 Gennaio 2023.

[24] How to Install MAAS, https://maas.io/docs/how-to-install-maas, ultimo accesso 07 Gennaio 2023.

[25] Deploy OpenStack: install MAAS, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/install-maas.html, ultimo accesso 07 Gennaio 2023.

[26] Power management reference on MAAS, https://maas.io/docs/power-management-reference, ultimo accesso 16 Gennaio 2023.

[27] MAAS installation requirements, https://maas.io/docs/maas-installation-requirements, ultimo accesso 04 Gennaio 2023.

[28] Documentazione Neutron ML2, https://wiki.openstack.org/wiki/Neutron/ML2PortSecurityExtensionDriver, ultimo accesso 23 Gennaio 2023.

[29] Documentazione Octavia, https://docs.openstack.org/octavia/latest/reference/introduction.html

[30] Deployment di OpenStack Yoga con i charm, https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/yoga/, ultimo accesso 28 Febbraio 2023.

[31] Tipi di charm di OpenStack, https://docs.openstack.org/charm-guide/latest/concepts/index.html, ultimo accesso 25 Febbraio 2023.

[32] 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.

[33] Guide al deployment di OpenStack, https://docs.openstack.org/yoga/deploy/index.html, ultimo accesso 21 Gennaio 2023.

[34] Bundle di installazione di esempio, https://github.com/openstack-charmers/openstack-bundles/blob/master/stable/openstack-base/bundle.yaml, ultimo accesso 22 Giugno 2023

[35] 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.

[36] Open Virtual Network, https://www.ovn.org/en/, ultimo accesso 23 Gennaio 2023.

[37] Open vSwitch, https://www.openvswitch.org, ultimo accesso 23 Gennaio 2023.

[38] 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.

[39] Raspberry Pi OS, https://www.raspberrypi.com/software/, ultimo accesso 06 Gennaio 2023.

[40] Raspberry Pi 4 Model B Specifications, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, ultimo accesso 06 Gennaio 2023.

[41] Documentazione Vault, https://developer.hashicorp.com/vault/docs, ultimo accesso 23 Gennaio 2023.

[42] Vault, https://developer.hashicorp.com/vault/docs/what-is-vault, ultimo accesso 23 Gennaio 2023.

[43] 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

Bundle di installazione

In questo capitolo verranno riportate le versioni finali del bundle di installazione di OpenStack e del bundle overlay che permette di installare Octavia e Barbican.

A.1 OpenStack

```yaml
# Please refer to the OpenStack Charms Deployment Guide for more information.
# https://docs.openstack.org/project-deploy-guide/charm-deployment-guide
#
# NOTE: Please review the value for the configuration option
#       ‘bridge-interface-mappings‘ for the ‘ovn-chassis‘ charm (see ‘data-port‘ variable).
#       Refer to the [Open Virtual Network (OVN)](https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/latest/app-ovn.html)
#       section of the [OpenStack Charms Deployment Guide](https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/latest/)
#       for more information.
name: openstack-base
series: focal
variables:
  data-port: &data-port br-ex:enp3s0
  osd-devices: &osd-devices /dev/sdb
  expected-osd-count: &expected-osd-count 3
  expected-mon-count: &expected-mon-count 3
  openstack-origin: &openstack-origin cloud:focal-yoga
```

```yaml
  # custom variables
  openstack-channel: &openstack-channel yoga/stable
  ceph-channel: &ceph-channel quincy/stable
  mysql-channel: &mysql-channel 8.0/stable
  rabbitmq-channel: &rabbitmq-channel 3.9/stable
  ovn-channel: &ovn-channel 22.03/stable
  vault-channel: &vault-channel 1.7/stable

machines:
  '0':
    constraints: arch=amd64 tags=compute
  '1':
    constraints: arch=amd64 tags=compute
  '2':
    constraints: arch=amd64 tags=compute
  '3':
    constraints: arch=amd64 tags=compute
applications:
  ceph-mon:
    charm: ch:ceph-mon
    channel: *ceph-channel
    num_units: 3
    options:
      expected-osd-count: *expected-osd-count
      monitor-count: *expected-mon-count
      source: *openstack-origin
    to:
    - lxd:0
    - lxd:1
    - lxd:2
  ceph-osd:
    charm: ch:ceph-osd
    channel: *ceph-channel
    num_units: 4
    options:
      osd-devices: *osd-devices
      source: *openstack-origin
    to:
    - '0'
    - '1'
    - '2'
    - '3'
  ceph-radosgw:
    charm: ch:ceph-radosgw
```

```
61      channel: *ceph-channel
62      num_units: 1
63      options:
64        source: *openstack-origin
65      to:
66      - lxd:0
67    cinder-mysql-router:
68      charm: ch:mysql-router
69      channel: *mysql-channel
70    cinder:
71      charm: ch:cinder
72      channel: *openstack-channel
73      num_units: 1
74      options:
75        block-device: None
76        glance-api-version: 2
77        openstack-origin: *openstack-origin
78      to:
79      - lxd:1
80    cinder-ceph:
81      charm: ch:cinder-ceph
82      channel: *openstack-channel
83      num_units: 0
84    glance-mysql-router:
85      charm: ch:mysql-router
86      channel: *mysql-channel
87    glance:
88      charm: ch:glance
89      channel: *openstack-channel
90      num_units: 1
91      options:
92        openstack-origin: *openstack-origin
93      to:
94      - lxd:3
95    keystone-mysql-router:
96      charm: ch:mysql-router
97      channel: *mysql-channel
98    keystone:
99      charm: ch:keystone
100     channel: *openstack-channel
101     num_units: 1
102     options:
103       openstack-origin: *openstack-origin
104     to:
```

```
105       - lxd:0
106     ncc-mysql-router:
107       charm: ch:mysql-router
108       channel: *mysql-channel
109     neutron-api-plugin-ovn:
110       charm: ch:neutron-api-plugin-ovn
111       channel: *openstack-channel
112     neutron-api:
113       charm: ch:neutron-api
114       channel: *openstack-channel
115       num_units: 1
116       options:
117         neutron-security-groups: true
118         flat-network-providers: physnet1
119         openstack-origin: *openstack-origin
120         enable-ml2-port-security: true
121       to:
122       - lxd:1
123     neutron-api-mysql-router:
124       charm: ch:mysql-router
125       channel: *mysql-channel
126     placement-mysql-router:
127       charm: ch:mysql-router
128       channel: *mysql-channel
129     placement:
130       charm: ch:placement
131       channel: *openstack-channel
132       num_units: 1
133       options:
134         openstack-origin: *openstack-origin
135       to:
136       - lxd:3
137     nova-cloud-controller:
138       charm: ch:nova-cloud-controller
139       channel: *openstack-channel
140       num_units: 1
141       options:
142         console-access-protocol: novnc
143         network-manager: Neutron
144         openstack-origin: *openstack-origin
145       to:
146       - lxd:3
147     nova-compute:
148       charm: ch:nova-compute
```

```
149       channel: *openstack-channel
150       num_units: 3
151       options:
152         config-flags: default_ephemeral_format=ext4
153         enable-live-migration: true
154         enable-resize: true
155         migration-auth-type: ssh
156         openstack-origin: *openstack-origin
157       to:
158       - '1'
159       - '2'
160       - '3'
161     dashboard-mysql-router:
162       charm: ch:mysql-router
163       channel: *mysql-channel
164     openstack-dashboard:
165       charm: ch:openstack-dashboard
166       channel: *openstack-channel
167       num_units: 1
168       options:
169         openstack-origin: *openstack-origin
170       to:
171       - lxd:2
172     rabbitmq-server:
173       charm: ch:rabbitmq-server
174       channel: *rabbitmq-channel
175       num_units: 1
176       to:
177       - lxd:2
178     mysql-innodb-cluster:
179       charm: ch:mysql-innodb-cluster
180       channel: *mysql-channel
181       num_units: 3
182       to:
183       - lxd:0
184       - lxd:1
185       - lxd:2
186     ovn-central:
187       charm: ch:ovn-central
188       channel: *ovn-channel
189       num_units: 3
190       options:
191         source: *openstack-origin
192       to:
```

```
193     - lxd:0
194     - lxd:1
195     - lxd:2
196   ovn-chassis:
197     charm: ch:ovn-chassis
198     channel: *ovn-channel
199     options:
200       ovn-bridge-mappings: physnet1:br-ex
201       bridge-interface-mappings: *data-port
202   vault-mysql-router:
203     charm: ch:mysql-router
204     channel: *mysql-channel
205   vault:
206     charm: ch:vault
207     channel: *vault-channel
208     num_units: 1
209     to:
210     - lxd:3
211 relations:
212 - - vault-mysql-router:db-router
213   - mysql-innodb-cluster:db-router
214 - - vault-mysql-router:shared-db
215   - vault:shared-db
216 - - mysql-innodb-cluster:certificates
217   - vault:certificates
218 - - neutron-api-plugin-ovn:neutron-plugin
219   - neutron-api:neutron-plugin-api-subordinate
220 - - neutron-api-plugin-ovn:ovsdb-cms
221   - ovn-central:ovsdb-cms
222 - - ovn-chassis:ovsdb
223   - ovn-central:ovsdb
224 - - ovn-chassis:nova-compute
225   - nova-compute:neutron-plugin
226 - - neutron-api:certificates
227   - vault:certificates
228 - - neutron-api-plugin-ovn:certificates
229   - vault:certificates
230 - - ovn-central:certificates
231   - vault:certificates
232 - - ovn-chassis:certificates
233   - vault:certificates
234 - - neutron-api-mysql-router:db-router
235   - mysql-innodb-cluster:db-router
236 - - neutron-api-mysql-router:shared-db
```

```
237     - neutron-api:shared-db
238   - - keystone-mysql-router:db-router
239     - mysql-innodb-cluster:db-router
240   - - keystone-mysql-router:shared-db
241     - keystone:shared-db
242   - - keystone:identity-service
243     - neutron-api:identity-service
244   - - keystone:certificates
245     - vault:certificates
246   - - rabbitmq-server:amqp
247     - neutron-api:amqp
248   - - rabbitmq-server:amqp
249     - nova-compute:amqp
250   - - ncc-mysql-router:db-router
251     - mysql-innodb-cluster:db-router
252   - - ncc-mysql-router:shared-db
253     - nova-cloud-controller:shared-db
254   - - nova-cloud-controller:identity-service
255     - keystone:identity-service
256   - - nova-cloud-controller:amqp
257     - rabbitmq-server:amqp
258   - - nova-cloud-controller:neutron-api
259     - neutron-api:neutron-api
260   - - nova-cloud-controller:cloud-compute
261     - nova-compute:cloud-compute
262   - - nova-cloud-controller:certificates
263     - vault:certificates
264   - - placement-mysql-router:db-router
265     - mysql-innodb-cluster:db-router
266   - - placement-mysql-router:shared-db
267     - placement:shared-db
268   - - placement:identity-service
269     - keystone:identity-service
270   - - placement:placement
271     - nova-cloud-controller:placement
272   - - placement:certificates
273     - vault:certificates
274   - - dashboard-mysql-router:db-router
275     - mysql-innodb-cluster:db-router
276   - - dashboard-mysql-router:shared-db
277     - openstack-dashboard:shared-db
278   - - openstack-dashboard:identity-service
279     - keystone:identity-service
280   - - openstack-dashboard:certificates
```

```
281   - vault:certificates
282 - - glance-mysql-router:db-router
283   - mysql-innodb-cluster:db-router
284 - - glance-mysql-router:shared-db
285   - glance:shared-db
286 - - glance:image-service
287   - nova-cloud-controller:image-service
288 - - glance:image-service
289   - nova-compute:image-service
290 - - glance:identity-service
291   - keystone:identity-service
292 - - glance:certificates
293   - vault:certificates
294 - - ceph-mon:osd
295   - ceph-osd:mon
296 - - ceph-mon:client
297   - nova-compute:ceph
298 - - ceph-mon:client
299   - glance:ceph
300 - - cinder-mysql-router:db-router
301   - mysql-innodb-cluster:db-router
302 - - cinder-mysql-router:shared-db
303   - cinder:shared-db
304 - - cinder:cinder-volume-service
305   - nova-cloud-controller:cinder-volume-service
306 - - cinder:identity-service
307   - keystone:identity-service
308 - - cinder:amqp
309   - rabbitmq-server:amqp
310 - - cinder:image-service
311   - glance:image-service
312 - - cinder:certificates
313   - vault:certificates
314 - - cinder-ceph:storage-backend
315   - cinder:storage-backend
316 - - cinder-ceph:ceph
317   - ceph-mon:client
318 - - cinder-ceph:ceph-access
319   - nova-compute:ceph-access
320 - - ceph-radosgw:mon
321   - ceph-mon:radosgw
```

A.2 Octavia e Brabican

```yaml
variables:
  openstack-origin: &openstack-origin cloud:focal-yoga
  openstack-channel: &openstack-channel yoga/stable
  mysql-channel: &mysql-channel 8.0/stable
  ovn-channel: &ovn-channel 22.03/stable
applications:
  barbican-mysql-router:
    charm: ch:mysql-router
    channel: *mysql-channel
  barbican:
    charm: ch:barbican
    channel: *openstack-channel
    num_units: 1
    options:
      openstack-origin: *openstack-origin
    to:
    - lxd:4
  barbican-vault:
    charm: ch:barbican-vault
    channel: *openstack-channel
  octavia-ovn-chassis:
    charm: ch:ovn-chassis
    channel: *openstack-channel
  octavia-mysql-router:
    charm: ch:mysql-router
    channel: *mysql-channel
  octavia:
    charm: ch:octavia
    channel: *openstack-channel
    num_units: 1
    options:
      openstack-origin: *openstack-origin
    to:
    - lxd:4
  octavia-dashboard:
    charm: ch:octavia-dashboard
    channel: *openstack-channel
  glance-simplestreams-sync:
    charm: ch:glance-simplestreams-sync
    channel: *openstack-channel
    num_units: 1
```

```
42      options:
43        source: ppa:simplestreams-dev/trunk
44        use_swift: false
45      to:
46      - lxd:4
47    octavia-diskimage-retrofit:
48      charm: ch:octavia-diskimage-retrofit
49      channel: *openstack-channel
50      options:
51        amp-image-tag: 'octavia-amphora'
52  relations:
53  - - octavia-mysql-router:db-router
54    - mysql-innodb-cluster:db-router
55  - - octavia-mysql-router:shared-db
56    - octavia:shared-db
57  - - octavia:certificates
58    - vault:certificates
59  - - barbican-mysql-router:db-router
60    - mysql-innodb-cluster:db-router
61  - - barbican-mysql-router:shared-db
62    - barbican:shared-db
63  - - barbican:certificates
64    - vault:certificates
65  - - keystone:identity-service
66    - octavia:identity-service
67  - - keystone:identity-service
68    - barbican:identity-service
69  - - rabbitmq-server:amqp
70    - octavia:amqp
71  - - rabbitmq-server:amqp
72    - barbican:amqp
73  - - neutron-api:neutron-load-balancer
74    - octavia:neutron-api
75  - - octavia-ovn-chassis:ovsdb-subordinate
76    - octavia:ovsdb-subordinate
77  - - octavia-ovn-chassis:certificates
78    - vault:certificates
79  - - octavia-ovn-chassis:ovsdb
80    - ovn-central:ovsdb
81  - - octavia:ovsdb-cms
82    - ovn-central:ovsdb-cms
83  - - openstack-dashboard:dashboard-plugin
84    - octavia-dashboard:dashboard
85  - - barbican-vault:secrets
```

```
86    - barbican:secrets
87  - - vault:secrets
88    - barbican-vault:secrets-storage
89  - - glance-simplestreams-sync:juju-info
90    - octavia-diskimage-retrofit:juju-info
91  - - keystone:identity-service
92    - glance-simplestreams-sync:identity-service
93  - - glance-simplestreams-sync:certificates
94    - vault:certificates
95  - - keystone:identity-credentials
96    - octavia-diskimage-retrofit:identity-credentials
```

Appendice B

Progetti Terraform

B.1 Progetto 1: istanza

B.1.1 main.tf

```
terraform {
  required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

provider "openstack" {
  user_name   = "user"
  password    = "password"
  auth_url    = "https://10.0.0.7:5000/v3" # keystone endpoint URL
  domain_name = "domain"
  tenant_name = "project1" # project name
  insecure    = true       # allow self-signed SSL cert
}

resource "openstack_compute_instance_v2" "test-server" {
  name            = "test-server"
  image_id        = "f2aa2ed8-cb58-4152-b881-5e3a3c24fe30"
  flavor_id       = "cf492d92-2496-4f62-9f3f-d160758c812c"
  key_pair        = "terraform"
  security_groups = ["default"]

  network {
    name = "internal"
  }
```

```
}
```

B.2 Progetto 2: infrastruttura completa

B.2.1 example.tfvars

```
auth = {
  user          = "matteo"                    # openstack user
  password      = "ubuntu"                    # user's password
  keystone_url = "https://10.0.0.7:5000/v3" # openstack identity service
    url
  project       = "project2"                  # openstack project
  domain        = "domain"                    # openstack domain
}

network = {
  external_network_id   = "68d5f950-bd27-43b7-8ca6-4fc96341a6dd" #public
    network id
  external_network_name = "ext_net"                              #public
    network name
  dns_nameservers       = ["8.8.8.8", "8.8.4.4"]                 # default
    dns servers
}

compute = {
  image_id  = "76c6c0f5-91cc-4446-bc3f-00bd7ee3fb08" # default compute
    image id
  flavor_id = "51c1c7b0-1811-4a61-bc55-fb82c86d24cb" # flavor id
  ssh_key   = "terraform"                            # ssh key name
}
```

B.2.2 instances.tf

```
resource "openstack_compute_instance_v2" "test-server" {
  name            = "test-server"
  flavor_id       = var.compute.flavor_id
  key_pair        = var.compute.ssh_key
  security_groups = ["${openstack_compute_secgroup_v2.tf_secgroup.name}"]

  block_device {
    uuid                  = var.compute.image_id
    source_type           = "image"
    volume_size           = 30
    boot_index            = 0
    destination_type      = "volume"
```

```
    delete_on_termination = true
  }
  //image_id = var.compute.image_id

  network {
    name = openstack_networking_network_v2.tf_private.name
  }
}


# floating ip configuration
resource "openstack_networking_floatingip_v2" "test-server_ip" {
  pool = var.network.external_network_name
}

resource "openstack_compute_floatingip_associate_v2" "test-server_ip" {
  floating_ip = openstack_networking_floatingip_v2.test-server_ip.address
  instance_id = openstack_compute_instance_v2.test-server.id
}
```

B.2.3 main.tf

```
terraform {
  required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}


provider "openstack" {
  user_name   = var.auth.user
  password    = var.auth.password
  auth_url    = var.auth.keystone_url
  domain_name = var.auth.domain
  tenant_name = var.auth.project # project name
  insecure    = true             # allow self-signed SSL cert
}

variable "auth" {
  type = object({
    user         = string
    password     = string
    keystone_url = string
    project      = string
```

```
    domain       = string
  })
}

variable "network" {
  type = object({
    external_network_id   = string
    external_network_name = string
    dns_nameservers       = list(string)
  })
}

variable "compute" {
  type = object({
    image_id  = string
    flavor_id = string
    ssh_key   = string
  })
}
```

B.2.4 networking.tf

```
resource "openstack_networking_network_v2" "tf_private" {
  name           = "tf_private"
  admin_state_up = "true"
}

resource "openstack_networking_subnet_v2" "tf_subnet" {
  name            = "tf_subnet"
  network_id      = openstack_networking_network_v2.tf_private.id
  ip_version      = 4
  cidr            = "192.168.1.0/24"
  gateway_ip      = "192.168.1.1"
  dns_nameservers = var.network.dns_nameservers
}

resource "openstack_compute_secgroup_v2" "tf_secgroup" {
  name        = "tf_secgroup"
  description = "Terraform security group"

  rule {
    from_port   = 22
    to_port     = 22
    ip_protocol = "tcp"
    cidr        = "0.0.0.0/0"
  }
}

```

```
resource "openstack_networking_router_v2" "tf_router" {
  name                = "tf_router"
  admin_state_up      = true
  external_network_id = var.network.external_network_id
}

resource "openstack_networking_router_interface_v2" "tf_router_interface" {
  router_id = openstack_networking_router_v2.tf_router.id
  subnet_id = openstack_networking_subnet_v2.tf_subnet.id
}
```

B.3 Progetto 3: Proof of Concept

B.3.1 example.tfvars

```
auth = {
  user         = "matteo"                    # openstack user
  password     = "ubuntu"                    # user's password
  keystone_url = "https://10.0.0.7:5000/v3" # openstack identity service
    url
  project      = "project3"                  # openstack project
  domain       = "domain"                    # openstack domain
}

network = {
  external_network_id   = "68d5f950-bd27-43b7-8ca6-4fc96341a6dd" #public
    network id
  external_network_name = "ext_net"                               #public
    network name
  dns_nameservers       = ["8.8.8.8", "8.8.4.4"]                  # default
    dns servers
}

compute = {
  image_id       = "76c6c0f5-91cc-4446-bc3f-00bd7ee3fb08" # default compute
     image id
  flavor_id      = "51c1c7b0-1811-4a61-bc55-fb82c86d24cb" # flavor id
  ssh_key        = "terraform"                            # ssh key name
  instance_count = 3                                      # instances count
}
```

B.3.2 init-instance.sh

```
#!/bin/bash
apt-get update
```

```
apt-get install -y apache2
echo "<DOCTYPE html>
<html lang=\"en\">
    <body>
        <h1>Hello, this is instance #${instance_idx}</h1>
    </body>
</html>" > /var/www/html/index.html
```

B.3.3 instances.tf

```
resource "openstack_compute_instance_v2" "web-server" {
  name            = "web-server-${count.index}"
  flavor_id       = var.compute.flavor_id
  image_id        = var.compute.image_id
  key_pair        = var.compute.ssh_key
  security_groups = ["${openstack_compute_secgroup_v2.web_secgroup.name}"]
  count           = var.compute.instance_count
  #user_data = replace("${file("init-instance.sh")}", "#x", count.index)
  #user_data = "${file("init-instance.sh")}"
  user_data = templatefile("${path.module}/init-instance.sh", {
    instance_idx = count.index
  })

  network {
    name = openstack_networking_network_v2.web_private.name
  }

}
```

B.3.4 load_balancer.tf

```
# load balancer
resource "openstack_lb_loadbalancer_v2" "web_lb" {
  name          = "web_lb"
  vip_subnet_id = openstack_networking_subnet_v2.web_private_subnet.id
}

# targets pool
resource "openstack_lb_pool_v2" "web_lb_pool" {
  name            = "web_lb_pool"
  protocol        = "HTTP"
  lb_method       = "ROUND_ROBIN"
  loadbalancer_id = openstack_lb_loadbalancer_v2.web_lb.id
}

# pool members
```

```
resource "openstack_lb_members_v2" "web_lb_pool_members" {
  pool_id = openstack_lb_pool_v2.web_lb_pool.id

  # iterate all instances and add their IP address to the pool
  dynamic "member" {
    for_each = openstack_compute_instance_v2.web-server
    iterator = instance
    content {
      address       = instance.value.network[0].fixed_ip_v4
      protocol_port = 80
      weight        = 1
    }
  }
}

# HTTP listener
resource "openstack_lb_listener_v2" "web_lb_listener" {
  name            = "web_lb_listener_http"
  protocol        = "HTTP"
  protocol_port   = 80
  loadbalancer_id = openstack_lb_loadbalancer_v2.web_lb.id
  default_pool_id = openstack_lb_pool_v2.web_lb_pool.id
}

resource "openstack_lb_monitor_v2" "web_lb_monitor" {
  name           = "web_lb_monitor"
  pool_id        = openstack_lb_pool_v2.web_lb_pool.id
  type           = "HTTP"
  delay          = 30
  timeout        = 10
  max_retries    = 10
  url_path       = "/"
  http_method    = "GET"
  expected_codes = "200"
}

# floating IP
resource "openstack_networking_floatingip_v2" "web_lb_floating_ip" {
  pool = var.network.external_network_name
}

resource "openstack_networking_floatingip_associate_v2" "
    web_lb_floating_ip_associate" {
  floating_ip = openstack_networking_floatingip_v2.web_lb_floating_ip.
    address
  port_id     = openstack_lb_loadbalancer_v2.web_lb.vip_port_id
}
```

B.3.5 main.tf

```
terraform {
  required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}


provider "openstack" {
  user_name   = var.auth.user
  password    = var.auth.password
  auth_url    = var.auth.keystone_url
  domain_name = var.auth.domain
  tenant_name = var.auth.project # project name
  insecure    = true             # allow self-signed SSL cert
}

variable "auth" {
  type = object({
    user         = string
    password     = string
    keystone_url = string
    project      = string
    domain       = string
  })
}

variable "network" {
  type = object({
    external_network_id   = string
    external_network_name = string
    dns_nameservers       = list(string)
  })
}

variable "compute" {
  type = object({
    image_id       = string
    flavor_id      = string
    ssh_key        = string
    instance_count = number
  })
}
```

B.3.6 networking.tf

```
resource "openstack_networking_network_v2" "web_private" {
  name           = "web_private"
  admin_state_up = "true"
}

resource "openstack_networking_subnet_v2" "web_private_subnet" {
  name            = "web_private_subnet"
  network_id      = openstack_networking_network_v2.web_private.id
  ip_version      = 4
  cidr            = "192.168.1.0/24"
  gateway_ip      = "192.168.1.1"
  dns_nameservers = var.network.dns_nameservers
}

resource "openstack_compute_secgroup_v2" "web_secgroup" {
  name        = "web_secgroup"
  description = "Terraform security group"

  rule {
    from_port   = 22
    to_port     = 22
    ip_protocol = "tcp"
    cidr        = "0.0.0.0/0"
  }

  rule {
    from_port   = 80
    to_port     = 80
    ip_protocol = "tcp"
    cidr        = "0.0.0.0/0"
  }
}

resource "openstack_networking_router_v2" "web_router" {
  name                = "web_router"
  admin_state_up      = true
  external_network_id = var.network.external_network_id
}

resource "openstack_networking_router_interface_v2" "web_router_interface"
    {
  router_id = openstack_networking_router_v2.web_router.id
  subnet_id = openstack_networking_subnet_v2.web_private_subnet.id
}
```

B.3.7 outputs.tf

```
output "loadbalancer_ip" {
  value = openstack_networking_floatingip_v2.web_lb_floating_ip.address
}

output "instances_info" {
  description = "Instances Informations"
  value = [for instance in openstack_compute_instance_v2.web-server : {
    id   = instance.id
    name = instance.name
    ip   = instance.network[0].fixed_ip_v4
  }]
}
```