

ALMA MATER STUDIORUM – UNIVERSITÀ DI
BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA,
INFORMATICA E TELECOMUNICAZIONI

**ALTERNATIVE PER LA
CONFIGURAZIONE
DELL'INFRASTRUTTURA DI RETE
IN PIATTAFORME OPENSTACK**

elaborato in
APPLICAZIONE E TECNICHE DI
TELECOMUNICAZIONI

Relatore
Prof. Ing. Walter Cerroni

Presentata da
Nikolas Subrani

Sessione II
Anno Accademico 2015-2016

*“alla mia famiglia che sempre mi sostiene
e alla nonna, supporto morale
in tante giornate di studio”*

INDICE

SOMMARIO	(7)
1. INTRODUZIONE AI PARADIGMI	(9)
1.1.CLOUD COMPUTING	(9)
1.2.SOFTWARE DEFINED NETWORKING	(11)
1.3.OPENSTACK	(12)
2. INSTALLAZIONE DEL CLUSTER OPENSTACK	(15)
2.1. TOPOLOGIA DEL LABORATORIO	(15)
2.2. OPERAZIONI PRELIMINARI	(17)
2.3. CONFIGURAZIONE KEYSTONE, GLANCE, NOVA	(17)
2.4. NEUTRON CONCETTI DI BASE	(19)
2.5. SCRIPT DI AVVIO DEL SISTEMA	(20)
3. SCENARIO PROVIDER NETWORK CON LINUXBRIDGE	(21)
3.1. CONFIGURAZIONE FILE DI NEUTRON E LAYOUT DI RETE	(21)
3.2.ANALISI PACKET FLOW	(25)
4. SCENARIO CLASSIC CON LINUXBRIDGE	(29)
4.1. CONFIGURAZIONE FILE DI NEUTRON E LAYOUT DI RETE	(30)
4.2. ANALISI PACKET FLOW	(33)
5. SCENARIO CLASSIC CON OPENVSWITCH	(36)
5.1.CONFIGURAZIONE FILE DI NEUTRON E LAYOUT DI RETE	(37)
5.2.ANALISI PACKET FLOW	(41)
6. APPROFONDIMENTO SULLE INTESAZIONI E DIMENSIONE DEI PACCHETTI NEGLI SCENARI PROPOSTI	(48)
7. CONCLUSIONI	(50)
BIBLIOGRAFIA	(52)

SOMMARIO

In questo scritto si analizzeranno alcune alternative nella configurazione di rete della piattaforma di cloud computing open source OpenStack. Verrà mostrata un'installazione in ambiente di laboratorio di un cluster completo basato sulla release Liberty di Openstack, per poi modificarne la componente dedicata al Networking in modo da sfruttare diversi plugin e diversi protocolli. Si osserverà il traffico generato all'interno e verso l'esterno del sistema Openstack in modo da avere un quadro generale del comportamento dell'infrastruttura.

CAPITOLO 1.

INTRODUZIONE AI PARADIGMI

Con il progredire delle prestazioni della rete Internet e la nascita di una nuova generazione di dispositivi, soprattutto mobili, sono sorte nuove esigenze sia per le aziende sia per gli utenti privati. È sotto questi impulsi che sono stati sviluppati due importanti paradigmi che stanno rivoluzionando il mondo delle Reti di Telecomunicazioni: il Cloud Computing e il Software-defined Networking.

1.1 CLOUD COMPUTING

Il Cloud Computing è un modello in grado di permettere l'accesso a risorse di calcolo in maniera diffusa e on-demand, indipendentemente dall'hardware a disposizione dell'utente, attraverso internet. Nato per dare una risposta alla sempre crescente domanda di capacità di calcolo, spazio di memorizzazione e flessibilità delle infrastrutture IT aziendali, ha conosciuto un notevole sviluppo, anche consumer, a partire dagli anni 2010. Il punto di forza di questo paradigma sono le sue peculiarità ben riassunte nelle “cinque caratteristiche essenziali” dal National Institute of Standard and Technology's (NIST) statunitense:

- ***On-demand self-service***, la capacità di svincolare l'utilizzatore dall'interazione con i singoli provider di servizi, come ad esempio quelli di storage
- ***Broad network access***, i servizi sono offerti e utilizzabili attraverso la rete e accessibili tramite meccanismi standard e da terminali eterogenei
- ***Resource pooling***, le risorse, fisiche e virtuali, disponibili sono raggruppate per servire più utenti secondo un modello multi-tenant e assegnate dinamicamente in base alla domanda
- ***Rapid elasticity***, la capacità di calcolo può essere fornita o redistribuita in maniera tale da scalare rapidamente il sistema a seconda del bisogno
- ***measured service***, i cloud sono in grado di controllare e ottimizzare, anche automaticamente, l'utilizzo della potenza di calcolo monitorando le capacità del sistema in base alla tipologia di servizio da offrire.

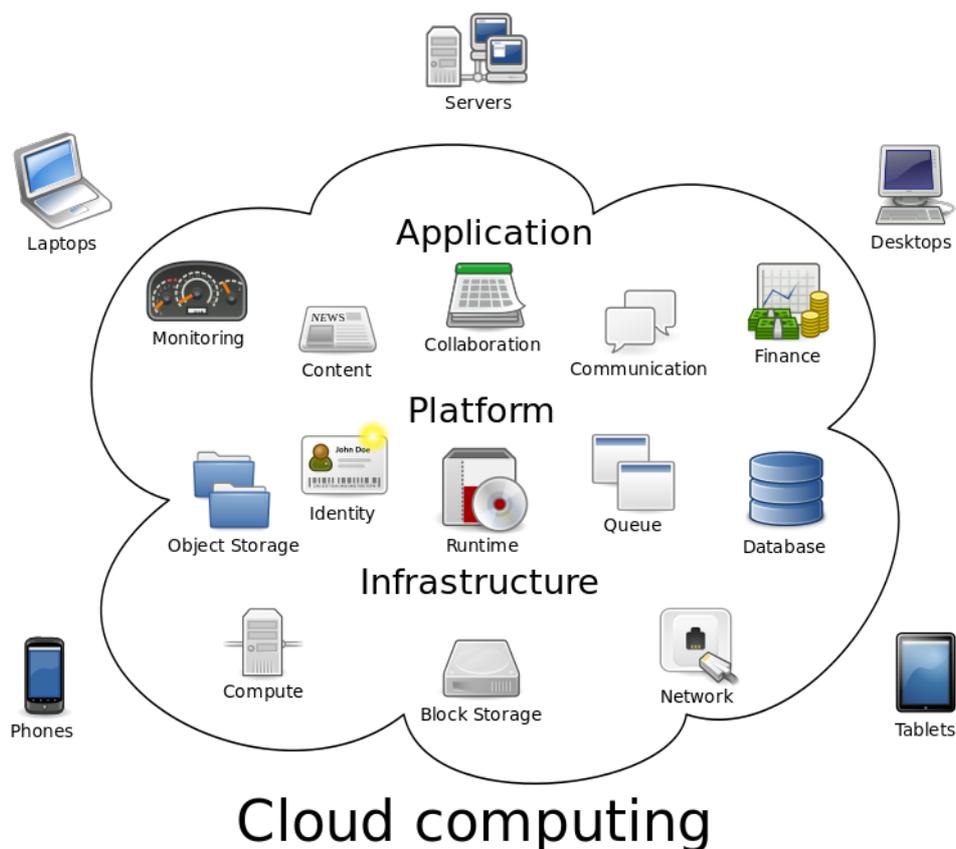


Figura 1: schema cloud computing

Secondo i fautori, il successo del cloud computing è dovuto all'elevata compatibilità con le richieste delle imprese di tutti i settori. Infatti consente di evitare costi per il mantenimento delle infrastrutture, e quindi di investire la quota di budget ad essi dedicata su nuovi progetti o differenziare il proprio business. Da un punto di vista più concreto, grazie al modello *pay as you go*, l'utilizzatore si trova ad avere applicazioni più snelle, miglior gestione delle risorse, flessibilità nei confronti del mercato e, come già accennato, minor manutenzione. Non sono trascurabili neanche fattori come le prestazioni, la produttività e la sicurezza. Aggregando le capacità di calcolo e affidandole ad un provider dedicato risulta più semplice mantenere aggiornato l'hardware e il software a disposizione, in quanto non vi è la necessità di intervenire sugli strumenti dei singoli utenti. L'aumento della produttività è intrinseco nel modello di cloud computing, poiché supporta utenti multipli in un singolo progetto anche simultaneamente, consentendo di risparmiare tempo e risorse. Infine la sicurezza: su questo tema ci sono molte controversie in quanto è comune pensiero il fatto che ci possa essere una perdita di controllo sui dati, in alcuni casi sensibili, ma è innegabile che l'elevato grado di

complessità dei sistemi e la distribuzione dei dati su una grande area o un notevole numero di dispositivi accrescono il grado di sicurezza rispetto ad attacchi dall'esterno.

1.2 SOFTWARE-DEFINED NETWORKING

Strettamente collegato al Cloud Computing, anche il paradigma del Software-defined Networking ha visto un notevole sviluppo negli ultimi anni, prevalentemente grazie all'adozione del protocollo OpenFlow sul finire del 2011.

Il concetto alla base del Software-defined Networking (SDN) è l'introduzione di opportuni livelli di astrazione in maniera tale da rendere programmabili i nodi di rete. Tramite questo approccio si può disaccoppiare il controllo dei flussi dall'infrastruttura fisica separando il piano di controllo da quello di trasporto del traffico. Elemento cardine di questo metodo è proprio il protocollo OpenFlow in quanto rende l'architettura SDN direttamente programmabile, permettendo agli sviluppatori di regolare dinamicamente il flusso del traffico su tutta la rete.

Si possono individuare alcune caratteristiche principali che definiscono l'architettura SDN che risulta:

- ***direttamente programmabile***, il controllo della rete è diviso dalle funzioni di forwarding, quindi liberamente impostabile
- ***flessibile***, è possibile regolare il flusso del traffico per far fronte ai bisogni del sistema
- ***gestita in maniera centralizzata***, la gestione è affidata ad un controller software che, al livello delle applicazioni, appare come uno switch
- ***orientata alla programmazione***, il SDN abilita i network manager alla configurazione, gestione e ottimizzazione attraverso programmi scritti autonomamente, poichè non esiste software proprietario
- ***basata su standard aperti e vendor-neutral***.

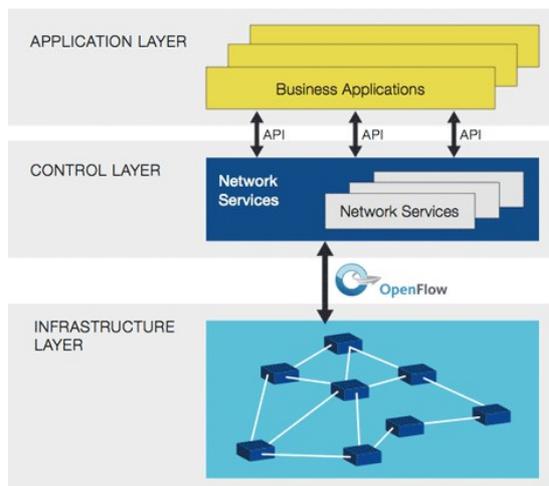


Figura 2: SDN con OpenFlow

Questa elasticità e flessibilità consente un notevole adattamento alle esigenze del mercato, tanto che si stima una crescita del settore intorno ai 35 miliardi di dollari nel 2018. A riprova di ciò, nel 2013 erano già sorte 225 imprese nell'ambito SDN, un vera e propria esplosione, contando che nel 2009 non se ne contava nemmeno una.

1.3 OPENSTACK

Nel 2010 Rackspace Hosting e NASA hanno unito le forze per sfruttare i modelli del SDN e del Cloud Computing per creare una piattaforma software open-source denominata OpenStack. Ad oggi oltre 500 compagnie del settore IT stanno collaborando a questo progetto, diventato ormai il punto di riferimento nel settore Cloud.

A testimonianza dell'adattabilità e della varietà di impieghi di un tale tipo di infrastruttura, tra gli utilizzatori si trovano società non appartenenti al settore ICT, come BMW, eBay, BBVA, e anche agenzie governative come la National Security Agency degli Stati Uniti.

Sul piano tecnico OpenStack si può classificare come una Infrastructure-as-a-Service (IaaS), ovvero il livello più basso di astrazione disponibile nel cloud computing. Sempre la NIST definisce una IaaS come il servizio che fornisce all'utente la capacità di elaborazione, lo storage, le reti e altre risorse di calcolo fondamentali. Per l'utente, quindi, questo corrisponde all'utilizzo di una o più macchine virtuali, implementate tramite l'hypervisor KVM, liberamente configurabili per poter eseguire software arbitrario, sistemi operativi o altri tipi di

applicazioni. Riassumendo, si può dire che si esegue una virtualizzazione di hardware tradizionale.

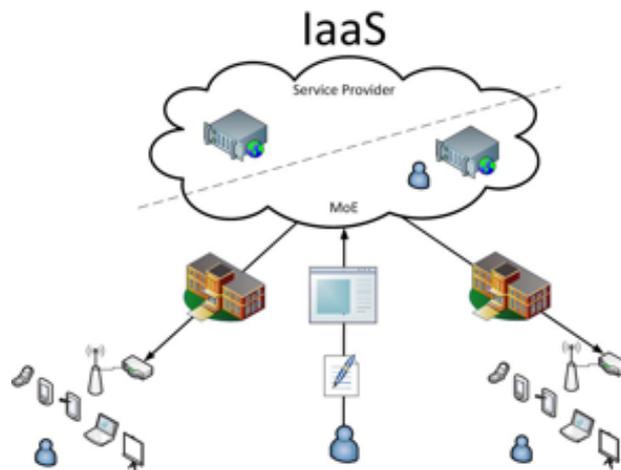


Figura 3: schema IaaS

Per poter ottenere questo risultato gli sviluppatori hanno suddiviso il sistema in quelli che sono chiamati OpenStack projects, ovvero 3 grandi *blocchi* di servizi: Compute, Storage e Compute+Storage. In base alla destinazione d'uso del cloud da approntare si possono scegliere i core services necessari che, interfacciandosi tra loro, costituiscono il cuore pulsante del sistema. Questi sono l'identity service (codename Keystone), il compute service (Nova), il networking service (Neutron), l'object storage (Swift) e l'object storage (Cinder). Per il progetto installato di cui si discute in questo documento si sono utilizzati i servizi Keystone, Glance, Nova, Neutron e il servizio opzionale Horizon, cioè l'utile dashboard via web.

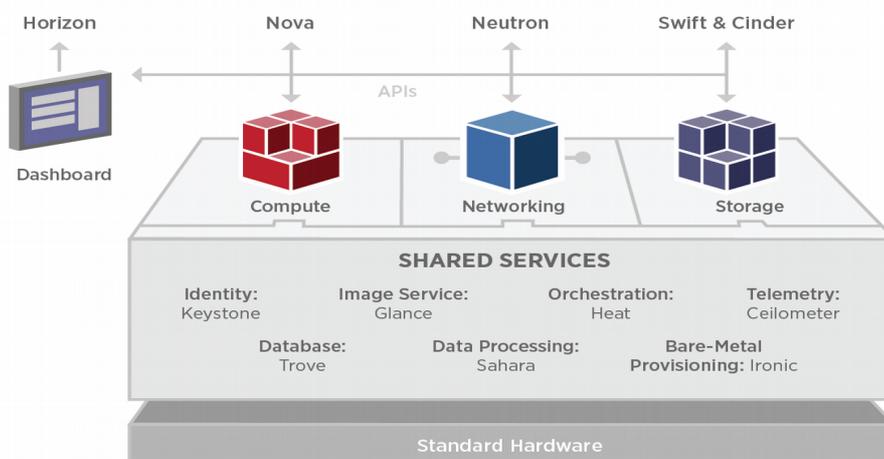


Figura 4: componenti di Openstack

Il software Openstack viene aggiornato ogni sei mesi attraverso una nuova release resa disponibile sul sito ufficiale della società. Attualmente, l'ultima versione è quella denominata *Mitaka*, rilasciata il 7 Aprile 2016, ma per il caso di studio in oggetto si è scelto di utilizzare la versione precedente *Liberty*, risalente all'Ottobre 2015, in modo tale da prevenire eventuali incompatibilità con il sistema operativo a disposizione.

CAPITOLO 2.

INSTALLAZIONE DEL CLUSTER OPENSTACK

Il servizio di networking Neutron di Openstack ha come scopo principale quello di fornire una struttura Networking-as-a-service attraverso il paradigma SDN, in modo tale da permettere al cloud manager di creare e collegare tra loro le eventuali interfacce create dagli altri componenti di OpenStack, ma in generale il compito principale di Neutron è quello di provvedere alla connettività e alle reti delle macchine virtuali, dette *istanze*, del servizio Nova compute. In base alle esigenze di chi disloca il sistema, il software concede la possibilità di scegliere la soluzione di rete più conforme alla destinazione d'uso. Lo scopo di questo scritto è proprio quello di analizzare le differenze tra i vari scenari proposti dagli sviluppatori, implementandoli nel cluster OpenStack installato presso il laboratorio LeLe sito a Cesena in via Venezia 260.

2.1 TOPOLOGIA DEL LABORATORIO

L'hardware a disposizione è composto da 4 pc dotati di due schede di rete, una delle quali

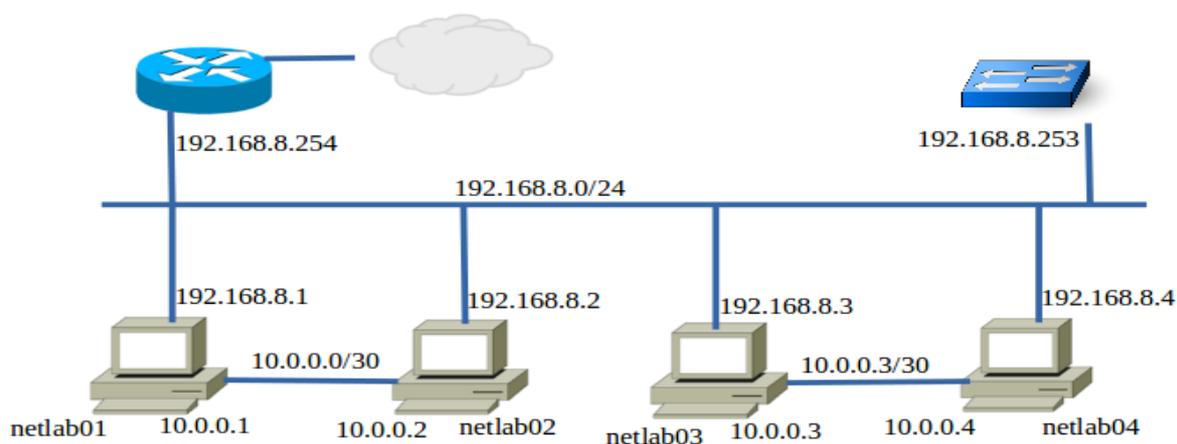


Figura 5: Topologia di laboratorio

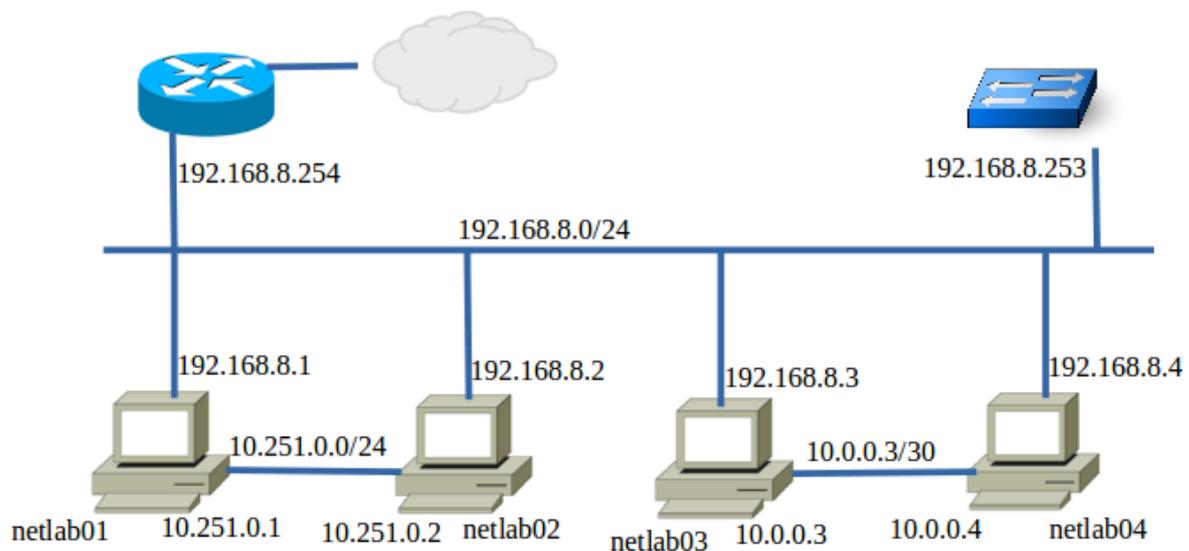
facente parte di una rete punto-punto (come indicato in figura 5).

La rete *principale* è la rete 192.168.8.0/24, tramite la quale le macchine possono connettersi alla rete esterna attraverso il router all'indirizzo 192.168.8.254. È presente anche uno switch ethernet managed HP procure 2524 dotato di 24 porte e situato all'indirizzo 192.168.8.253, in grado di gestire eventuali VLAN . In seguito, per quanto riguarda la rete punto-punto tra netlab01 e netlab02 si è decisi di modificarla in una /24 tramite uno script per poter, all'occorrenza, simulare la rete esterna sulla macchina netlab02.

```
#!/bin/sh
ifconfig eth0 down
ifconfig eth0 10.251.0.1/24
ifconfig eth0 up

sysctl net.ipv4.ip_forward=1 net.ipv4.conf.all.rp_filter=0 net.ipv4.conf.default.rp_filter=0
```

La dotazione di rete dei pc del laboratorio non è quella consigliata dagli sviluppatori del software di Openstack, in quanto sono richieste almeno due o tre NIC in base alla destinazione d'uso del dispositivo. Questa circostanza potrebbe portare ad alcune limitazioni, ma un' adeguata progettazione del layout di rete può sopperire a queste mancanze, se ne vedranno alcuni esempi nei prossimi capitoli.



Sulle quattro macchine netlab01-02-03-04 è installato come Sistema Operativo CentOS 7, basato sulla versione del kernel Linux 3.10.

2.2 OPERAZIONI PRELIMINARI

Prima di procedere con l'installazione vera e propria dei vari *blocchi* del sistema OpenStack, sono stati assegnati i ruoli alle diverse macchine a disposizione. Nello scenario default di Liberty, per funzionare correttamente Openstack richiede la presenza di almeno 2 dispositivi, ovvero un nodo che funga da controller e un nodo dedicato al computing. Nel caso in oggetto a netlab01 è stata attribuita la funzione di controller node e a netlab03 e netlab04 il compito di fornire le risorse fisiche per il servizio Nova compute. In altri scenari, che verranno presentati più avanti, sarà necessario un quarto nodo che dovrà supervisionare al networking. Questa mansione sarà affidata a netlab02 o a netlab01, in base dello scenario.

In seguito, utilizzando la guida d'installazione, si è provveduto a scaricare e ad aggiornare i pacchetti richiesti da OpenStack, per prevenire qualsivoglia malfunzionamento legato a conflitti dovuti ai nuovi package. Tra questi, nel controller, sono presenti l' SQL database dedicato all'archiviazione dei dati riguardanti i vari servizi, un message queue come RabbitMQ, per gestire la comunicazione tra il controller e gli altri nodi e, su tutte le macchine del cluster, un server Chrony, per gestire la sincronizzazione per mezzo di un Network Time Protocol.

2.3 CONFIGURAZIONE KEYSTONE, GLANCE, NOVA

Keystone gestisce le autenticazioni degli utenti e le autorizzazioni all'interno del cluster. Perciò, dopo aver configurato le service entity e gli API endpoint, si prosegue con la creazione dei progetti, cioè l'insieme di capacità dedicate agli utenti abilitati ad operare sul sistema. Per prima cosa si crea l'utente Admin, in grado di poter gestire e modificare le istanze e le reti per tutti gli utilizzatori del cluster, poi si procede con la realizzazione di Demo, esemplificazione di un possibile utilizzatore del cluster, in grado di gestire un determinato pool di risorse, in completa autonomia e, se la configurazione di rete lo permette, anche di creare subnet dedicate per il suo progetto.

Per poter acquisire le credenziali necessarie a utilizzare i comandi di Openstack tramite linea di comando si compilano script creati appositamente e richiamabili sempre tramite la CLI.

```
#!/bin/bash

export OS_PROJECT_DOMAIN_ID=default
export OS_USER_DOMAIN_ID=default
export OS_PROJECT_NAME=admin
export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_AUTH_URL=http://controller:35357/v3
export OS_IDENTITY_API_VERSION=3
```

Eseguendo le verifiche di test del servizio Keystone e digitando quindi il comando *openstack token issue* nel terminale, si è rilevata una criticità in quanto il sistema richiede la versione 2 dell' `OS_IDENTITY_API_VERSION` di Keystone, se sono specificati `OS_PROJECT_DOMAIN_ID` e `OS_USER_DOMAIN_ID`. Ricercando online sul servizio review.openstack.org è stata trovata una proposta di soluzione del problema che implementa ulteriori righe di codice nel file `/openstackclient/common/clientmanager.py` in modo tale da bypassare la richiesta di una versione di Keystone diversa da quella indicata nello script.

```
198 # NOTE(hieulq): If USER_DOMAIN_NAME, USER_DOMAIN_ID, PROJECT_DOMAIN_ID
199 # or PROJECT_DOMAIN_NAME is present and API_VERSION is 2.0, then
200 # ignore all domain related configs.
201 if (self._api_version.get('identity') == '2.0' and
202     self.auth_plugin_name.endswith('password')):
203     LOG.warning("Ignoring domain related configs "
204                "because identity API version is 2.0")
205     domain_props = ['project_domain_name', 'project_domain_id',
206                   'user_domain_name', 'user_domain_id']
207     for prop in domain_props:
208         self._auth_params.pop(prop, None)
209
```

Richard Theis	This message is always issued even if I don't have domain configs	May 6 3:39 PM
Hieu LE	Done	May 9 6:43 AM

Figura 6: righe di codice aggiunte in `clientmanager.py`

In seguito si prosegue con l'impostazione dell'immagine service Glance, che rende disponibili agli utenti le immagini disco per le virtual machine e con l'installazione di Nova Compute. Per rendere la gestione e il monitoraggio del cluster più immediato si è, come descritto poc'anzi, scelto di installare Horizon, la web dashboard. Da un qualunque browser Internet sul controller è possibile accedere all'applicazione mediante il percorso `controller/dashboard`, da cui si possono lanciare le istanze, controllare e modificare le reti, monitorare e gestire le risorse hardware e altri compiti di interesse minore, per l'argomento trattato in questo scritto.

Giunti a questo punto, resta da approntare il networking service e, nei prossimi capitoli, verranno mostrate le varie configurazioni possibili, compatibilmente con le capacità dell'hardware a disposizione.

2.4 NEUTRON, CONCETTI DI BASE

Openstack Networking gestisce tutti gli aspetti riguardanti la Virtual Networking Infrastructure e l'accesso alla rete fisica del cluster. Neutron provvede a fornire le astrazioni necessarie per simulare gli strumenti fisici: si dispone, quindi, di reti con possibilità di effettuare il subnetting e di router in grado di indirizzare i pacchetti e implementare firewall.

In ogni set-up deve essere presente una rete fisica connessa con l'esterno e raggiungibile dal di fuori dell'installazione Openstack. Questa network è sfruttata per connettere le VM ad Internet e per garantire l'accesso a queste da remoto. Tra loro, invece, le macchine virtuali sono collegate attraverso software-defined network interne al cluster. Queste reti virtuali richiedono una rete fisica su cui agire. Il tramite tra l'esterno e l'interno è operato dai virtual router, il cui gateway è, appunto, sulla rete pubblica. Al VR possono essere connesse più sottoreti, per mezzo di interfacce virtuali. Come nel caso fisico, virtual machine su subnet diverse possono dialogare tra loro a patto che siano collegate ad uno stesso router. Il meccanismo che rende possibile l'accesso da remoto alle istanze è quello dei floating IP, qualora non si utilizzino configurazioni *provider*. Indipendentemente su quale rete opera, alla macchina virtuale viene assegnato un indirizzo della network esterna, accessibile da tutti. L'IP può essere pubblico se si dispongono di reti con tale caratteristica, dato che il DHCP sulla external network deve essere disabilitato.

Nella versione *vanilla* Liberty di Openstack Neutron sono richieste almeno due network fisiche. La prima è quella definita come provider network e fornisce gli indirizzi pubblici o comunque l'accesso alla rete Internet. La seconda è la rete di management utilizzata per gestire i nodi del cluster e, solo a partire da questa release, anche per il tunneling tra le varie VM. In altri scenari sono necessarie diverse configurazioni come, ad esempio, una rete dedicata per il tunneling separata da quella di management.

L'installazione Openstack effettuata per questo elaborato ha dovuto tenere conto delle limitazioni imposte dalla configurazione di laboratorio, per cui si è utilizzata la rete

192.168.8.0/24 con funzioni sia di rete esterna sia di management-tunneling. Si vedrà in seguito che questo vincolo ha portato ad adottare alcuni accorgimenti.

2.5 SCRIPT DI AVVIO DEL SISTEMA

Le macchine utilizzate per l'analisi descritta in questo documento non erano riservate solo al cluster di Openstack, per non pesare su altri processi sono stati creati script da eseguire all'accensione con il comando “*sudo ./start_script.sh*” per avviare il sistema.

Su netlab01, quindi sul controller, lo script deve avviare Keystone, Glance, Nova, Neutron e Horizon oltre ad altri servizi utili al funzionamento di Openstack, mentre sui compute node netlab03 e netlab04 solo Nova e Neutron.

```
#!/bin/sh

systemctl start chronyd.service mariadb.service mongod.service
systemctl start memcached.service httpd.service
systemctl start openstack-glance-api.service openstack-glance-registry.service
systemctl start openstack-nova-api.service openstack-nova-cert.service openstack-nova-consoleauth.service openstack-nova-scheduler.service
openstack-nova-conductor.service openstack-nova-novncproxy.service
systemctl start neutron-server.service neutron-linuxbridge-agent.service neutron-dhcp-agent.service neutron-metadata-agent.service neutron-
l3-agent.service
```

Figura 7: script di avvio su netlab01

```
#!/bin/sh

systemctl start libvirtd.service openstack-nova-compute.service
systemctl start neutron-linuxbridge-agent.service
```

Figura 8: script sul compute node

Si precisa che per quanto riguarda lo script dei compute i servizi di Neutron possono richiedere l'avvio dell' Openvswitch service invece del Linuxbridge.

CAPITOLO 3.

SCENARIO PROVIDER NETWORK CON LINUXBRIDGE

Questa configurazione di Openstack Networking è la più semplice dal punto di vista dell'implementazione, in quanto non necessita di un nodo dedicato per la gestione del traffico e tutta la struttura del networking poggia una o più generiche reti. Tale specifica consente di ottenere migliori prestazioni e più affidabilità. Le prestazioni sono garantite dal fatto che le operazioni di rete sono amministrare dall'infrastruttura fisica che sottosta all'installazione Openstack, mentre l'affidabilità e anche la semplicità sono dovute all'utilizzo come plugin Modular layer 2 del Linuxbridge, completamente nativo sul sistema operativo in uso. Essenzialmente in questo scenario si effettua il bridging tra le reti virtuali, se sono configurate, e il livello fisico. Tipicamente per la *provider* network si utilizzano VLAN, ma sono permesse anche flat network. Nel caso di studio sono state utilizzate proprio queste ultime.

Gli sviluppatori hanno poi inserito la possibilità di modificare lo scenario per poter utilizzare quelle che sono definite self-service networks, cioè la capacità degli utenti di gestire la propria infrastruttura di rete virtuale, senza coinvolgere gli amministratori del cloud. Questa condizione sfrutta servizi di livello 3 e tecnologie di network virtualization come le VXLAN. Nella analisi effettuata e descritta nei prossimi paragrafi è stata utilizzata la feature delle self-service networks.

3.1 CONFIGURAZIONE FILE DI NEUTRON E LAYOUT DI RETE

Al fine di definire i meccanismi utilizzati bisogna modificare opportunamente i file di Openstack, inserendo le righe di codice adeguate.

Per prima cosa si modificano i file sul controller, quindi sulla macchina netla01. In */etc/neutron/neutron.conf* nella sezione [DEFAULT] si abilitano il plugin ML2, i router service e gli indirizzi overlapping IP.

```
[DEFAULT]
...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

In */etc/neutron/plugins/ml2/ml2.conf* nella sezione [ml2] si abilitano le flat, VLAN, VXLAN network, le VXLAN per le reti private, i meccanismi Linuxbridge e l2population e port security come extension driver. Nelle sezioni [ml2_type_flat] e [ml2_type_vxlan] vengono settati la rete public e il range del VXLAN network identifier. In [security_group] si abilita l'ipset, per aumentare l'efficienza delle regole dei security-group.

```
[ml2]
...
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = linuxbridge,l2population
extension_drivers = port_security

[ml2_type_flat]
...
flat_networks = public

[ml2_type_vxlan]
...
vni_ranges = 1:1000

[securitygroup]
...
enable_ipset = True
```

Si prosegue configurando il Linuxbridge agent in */etc/neutron/plugins/ml2/linuxbridge_agent.ini*. Nella sezione [linuxbridge] si mappa la rete virtuale public sull'interfaccia fisica scelta, nel nostro caso la eth1 di netla01 con indirizzo 192.168.8.1. In [vxlan] si abilitano le VXLAN come protocollo di incapsulamento dei pacchetti, il protocollo l2population e si configura l'interfaccia che gestisce la rete di tunneling, per noi sempre la eth1. In [agent] e [securitygroup] si attiva la protezione dall'ARP spoofing e i security group, mentre sempre in [securitygroup] si definisce il driver per il firewall.

```

[linux_bridge]
physical_interface_mappings = public:eth1

[vxlan]
enable_vxlan = True
local_ip = 192.168.8.1
l2_population = True

[agent]
...
prevent_arp_spoofing = True

[securitygroup]
...
enable_security_group = True
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver

```

Infine si modificano */etc/neutron/l3_agent.ini* e */etc/neutron/dhcp_agent.ini*, abilitando i driver richiesti e lasciando vuota la specifica external network bridge per consentire più reti esterne per un singolo controller.

```

[DEFAULT]
...
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
external_network_bridge =

```

```

[DEFAULT]
...
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = True
|

```

A questo punto si passa a configurare i compute node netlab03 e netlab04, modificando il file */etc/neutron/plugins/ml2/linuxbridge_agent.ini*. Anche in questo caso si mappa la public virtual network sull'interfaccia eth1 con indirizzo 192.168.8.3, utilizzando quest'ultima anche come interfaccia dei tunnel. Si abilitano, inoltre, la prevenzione dell' ARP spoofing, i security group e i driver necessari.

```

[linux_bridge]
physical_interface_mappings = public:eth1

[vxlan]
enable_vxlan = True
local_ip = 192.168.8.3
l2_population = True

[agent]
...
prevent_arp_spoofing = True

[securitygroup]
...
enable_security_group = True
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver

```

Per verificare la corretta installazione, dopo aver ottenuto le credenziali di admin con il comando `source admin-openrc.sh`, si eseguono `neutron ext-list` al fine di appurare il corretto lancio del neutron-server e `neutron agent-list` in modo da constatare l'avvio di tutti gli agent sul controller e sui compute. Su netlab01 sono attivi il linuxbridge agent, il dhcp agent, il metadata agent e il L3 agent, nel tempo in cui su netlab03 e netlab04 sono funzionanti i Linuxbridge agent.

Con l'infrastruttura di rete pronta si creano le reti. Si parte con la public network tramite il comando `neutron net-create public --shared --provider:physical_network public --provider:network_type flat` e la rispettiva subnet `neutron subnet-create public 192.168.8.0/24 --name public --allocation-pool start=192.168.8.50,end=192.168.8.200 --dns-nameserver 137.204.78.17 --gateway 192.168.8.254`. Si prosegue con la private project network: `neutron net-create private`, `neutron subnet-create private 172.16.2.0/24 --name private --dns-nameserver 137.204.78.17 --gateway 172.16.2.1`. Per connettere le due reti si crea un virtual router contenente un interfaccia connessa ad ognuna delle due reti. Comandi: `neutron net-update public --router:externali`, `neutron router-create router`, `neutron router-interface-add-add router private`, `neutron router-gateway-set router public`.

In questo momento si è pronti a lanciare le instance, una sulla rete public e una sulla rete private. Tramite la web dashboard si seleziona l'immagine disco (si è scelto Ubuntu 14.04 cloud), la rete su cui lanciare la macchina virtuale e le regole dei security group. All'instance sulla public network viene assegnato un indirizzo per mezzo del DHCP, mentre all'istanza sulla private network occorre assegnare un floating IP sulla rete 192.168.8.0/24 per potervi accedere da remoto. `neutron floatingip-create public` e `nova floatingip-associate [nome_istanza] [indirizzo_floatingip]` i comandi da eseguire.

3.2 ANALISI PACKET FLOW

Per analizzare il traffico si eseguono dei ping tra le instance e verso Internet, prima però è utile soffermarsi a osservare la topologia del cluster dopo aver lanciato le virtual machine.

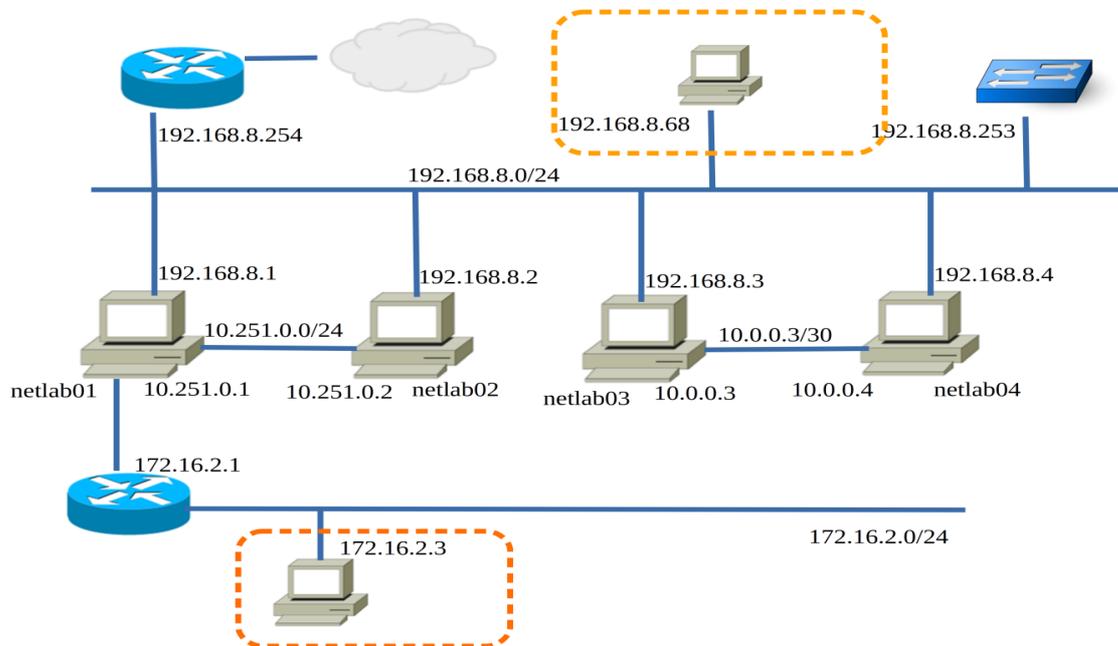


Figura 9: cluster con instance attive

Nella figura 9 abbiamo le macchine virtuali racchiuse dal tratteggio. Grazie al fatto che le VXLAN operano a livello 4 UDP, possiamo pensare la topologia di livello 2 come in figura, tenendo conto delle differenze tra elementi virtuali e fisici.

Gli elementi virtuali risiedono sul controller e sui compute node e sono i bridge creati dal Linuxbridge agent. Sul controller e sui compute vengono creati tanti bridge virtuali quante sono le istanze lanciate. Nei compute node il bridge connesso alla rete public crea una tap interface in modo da connettere la macchina virtuale con l'interfaccia fisica, al contrario il bridge che opera sulla rete private, oltre alla tap, ha un'ulteriore virtual interface dedicata alla VXLAN.

```
[nikolas.subrani@netlab03 ~]$ brctl show
bridge name      bridge id          STP enabled      interfaces
brq71a4cfc-57    8000.def140132885  no               tap921e0a27-57
                  vxlan-47
brqffc85a87-9f   8000.901b0e5cdebb  no               eth1
                  tape513cb9b-5e
virbr0           8000.525400aed1cf  yes              virbr0-nic
```

Sul controller viene ripetuto lo stesso schema, ma, come è logico attendersi, nei bridge sono presenti tante tap quante sono le istanze nel cluster.

```

[nikolas.subrani@netlab01 ~]$ brctl show
bridge name      bridge id          STP enabled      interfaces
brq71a4cfcc-57   8000.36dc2eb32d4c  no               tapc50db707-cb
                  tapfc2db601-21
                  vxlan-47
brqffc85a87-9f   8000.422baf0eca8c  no               eth1
                  tap7a903452-bc
                  tapd59f319c-7c
virbr0           8000.525400472bcc  yes              virbr0-nic

```

Il layout è quello mostrato nella figura 10, tenendo presente che il VXLAN tunnel agisce su quella che dovrebbe essere la rete di management, nella nostra installazione questo compito è, come già specificato, svolto sempre dalla 192.168.8.0/24. Quindi tutto il traffico generato da Openstack attraversa una sola rete e una sola interfaccia fisica su ogni nodo.

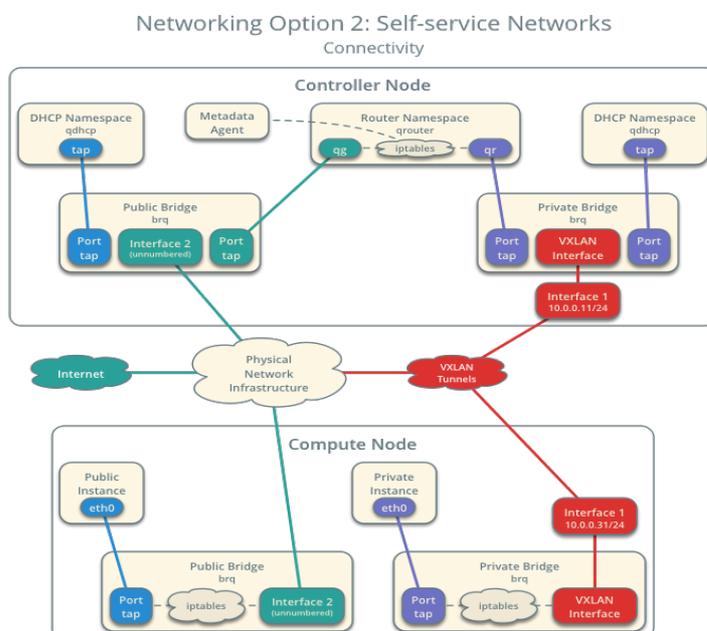


Figura 10: Figura 10 layout di rete

Effettuando il ping dalla VM sulla rete private con indirizzo 172.16.2.4 alla VM sulla rete public con indirizzo 192.168.8.68, utilizzando il tool *tcpdump* sono state effettuate le catture di pacchetti sulle interfacce dei vari nodi per analizzarne il percorso. I pacchetti generati dall'istanza entra nel bridge public, il quale sulla macchina netlab04 (che ospita fisicamente la macchina virtuale) è nominato brq71a4cfcc-57, tramite la tap per poi uscire sulla rete attraverso l'interfaccia vxlan-47.

```
[nikolas.subrani@netlab04 ~]$ sudo tcpdump -i vxlan-47 proto ICMP -n -vv
tcpdump: WARNING: vxlan-47: no IPv4 address assigned
tcpdump: listening on vxlan-47, link-type EN10MB (Ethernet), capture size 65535 bytes
11:50:20.336187 IP (tos 0x0, ttl 64, id 19961, offset 0, flags [DF], proto ICMP (1), length 84)
    172.16.2.4 > 192.168.8.68: ICMP echo request, id 1177, seq 1557, length 64
11:50:20.337159 IP (tos 0x0, ttl 63, id 40592, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.8.68 > 172.16.2.4: ICMP echo reply, id 1177, seq 1557, length 64
11:50:21.337706 IP (tos 0x0, ttl 64, id 20033, offset 0, flags [DF], proto ICMP (1), length 84)
    172.16.2.4 > 192.168.8.68: ICMP echo request, id 1177, seq 1558, length 64
11:50:21.338544 IP (tos 0x0, ttl 63, id 40686, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.8.68 > 172.16.2.4: ICMP echo reply, id 1177, seq 1558, length 64
```

I packets diretti all'indirizzo 192.168.8.68 devono attraversare per forza il controller, eseguendo infatti la cattura sull'interfaccia vxlan-47 su netlab01 si nota questo transito. Dato che le virtual machine sono due reti distinte, è obbligatorio il passaggio attraverso il router virtuale, il quale opererà il routing. Le porte del router sono la qr-fc2db601-21 avente indirizzo 172.16.2.1 e la qg-d59f319c-7c sulla rete 192.168.8.0/24. Per eseguire tale sniffing è necessario entrare nel namespace del router con il comando “*sudo ip netns [identificativo del router] bash*”.

```
[root@netlab01 nikolas.subrani]# tcpdump -i qr-fc2db601-21 proto ICMP -n -vv
tcpdump: listening on qr-fc2db601-21, link-type EN10MB (Ethernet), capture size 65535 bytes
11:55:53.857788 IP (tos 0x0, ttl 64, id 62634, offset 0, flags [DF], proto ICMP (1), length 84)
    172.16.2.4 > 192.168.8.68: ICMP echo request, id 1177, seq 1890, length 64
11:55:53.858369 IP (tos 0x0, ttl 63, id 17086, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.8.68 > 172.16.2.4: ICMP echo reply, id 1177, seq 1890, length 64
11:55:54.859434 IP (tos 0x0, ttl 64, id 62735, offset 0, flags [DF], proto ICMP (1), length 84)
    172.16.2.4 > 192.168.8.68: ICMP echo request, id 1177, seq 1891, length 64
11:55:54.860021 IP (tos 0x0, ttl 63, id 17156, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.8.68 > 172.16.2.4: ICMP echo reply, id 1177, seq 1891, length 64
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

```
#cattura nel router sul controller sulla porta qg... della rete 192.168.8.0/24
[root@netlab01 nikolas.subrani]# tcpdump -i qg-d59f319c-7c proto ICMP -n -vv
tcpdump: listening on qg-d59f319c-7c, link-type EN10MB (Ethernet), capture size 65535 bytes
11:57:17.993083 IP (tos 0x0, ttl 63, id 7502, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.8.73 > 192.168.8.68: ICMP echo request, id 1177, seq 1974, length 64
11:57:17.993564 IP (tos 0x0, ttl 64, id 27693, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.8.68 > 192.168.8.73: ICMP echo reply, id 1177, seq 1974, length 64
11:57:18.994774 IP (tos 0x0, ttl 63, id 7640, offset 0, flags [DF], proto ICMP (1), length 84)
    192.168.8.73 > 192.168.8.68: ICMP echo request, id 1177, seq 1975, length 64
11:57:18.995269 IP (tos 0x0, ttl 64, id 27890, offset 0, flags [none], proto ICMP (1), length 84)
    192.168.8.68 > 192.168.8.73: ICMP echo reply, id 1177, seq 1975, length 64
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Il ping viene indirizzato dal router al bridge public brqffc85a87-9f e quindi giunge alla rete 192.168.8.0/24. Da notare che il router applica il NAT, sostituendo l'indirizzo sorgente 172.16.2.4 con il floating ip 192.168.8.73 assegnato all'istanza. A questo punto, la mansione del corretto smistamento del pacchetto è affidata all'infrastruttura fisica di rete, più precisamente al gateway all'indirizzo 192.168.8.254. Dopo di che il pacchetto ICMP giunge alla macchina netlab03, dove risiede l'altra VM, percorre il bridge brqffc85a87-9f e arriva a destinazione. La virtual machine di destinazione vede i pacchetti echo request pervenire con un indirizzo della rete 192.168.8.0/24 e dunque anche gli echo reply avranno un indirizzo

destinazione di medesima fattura, sarà sempre compito del router virtuale applicare il DNAT per far sì che i pacchetti giungano correttamente sulla rete 172.16.2.0/24.

Se invece il ping viene effettuato dall'istanza sulla rete private verso Internet, i pacchetti eseguiranno lo stesso percorso sino al bridge brqffc85a87-9f su netlab01 e poi verranno indirizzati all'esterno dal gateway fisico, con la sola differenza che sul controller i packets ICMP attraversano l'interfaccia fisica eth1 per uscire sulla rete 192.168.8.0/24.

```
[nikolas.subrani@netlab01 ~]$ sudo tcpdump -c 6 -i eth1 proto ICMP -n
tcpdump: WARNING: eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
12:35:32.377340 IP 192.168.8.72 > 162.242.140.107: ICMP echo request, id 1195, seq 1249, length 64
12:35:32.515960 IP 162.242.140.107 > 192.168.8.72: ICMP echo reply, id 1195, seq 1249, length 64
12:35:33.379113 IP 192.168.8.72 > 162.242.140.107: ICMP echo request, id 1195, seq 1250, length 64
12:35:33.515272 IP 162.242.140.107 > 192.168.8.72: ICMP echo reply, id 1195, seq 1250, length 64
4 packets captured
5 packets received by filter
0 packets dropped by kernel
```

I pacchetti indirizzati verso l'esterno dalla VM public si comportano come normale traffico della rete 102.168.8.0/24.

CAPITOLO 4.

SCENARIO CLASSIC CON LINUXBRDIGE

Questa configurazione di Neutron è lo scenario più consono a una implementazione classica del cluster Openstack. Sono previste almeno due reti: una chiamata esterna che provvede alla connessione del cluster con Internet e la rete project dove vengono lanciate le istanze. La dotazione hardware standard richiesta è mostrata nella figura 9, ma coerentemente con le risorse a disposizione si sono utilizzate la rete 192.168.8.0/24 come rete di management e dei tunnel e come rete esterna la punto-punto 10.251.0.0/24 tra netlab01 e netlab02.

Hardware Requirements

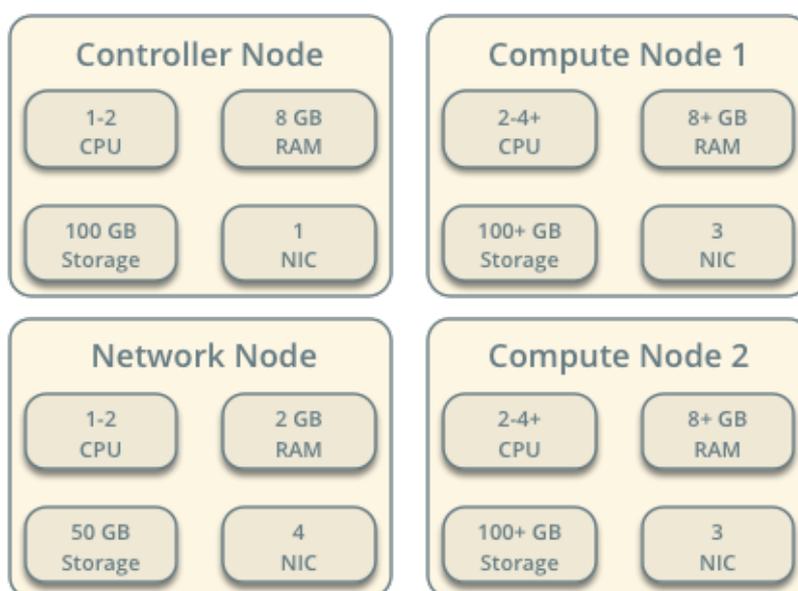


Figura 11: requisiti consigliati per un'installazione openstack

In questa configurazione è necessario un network node che si occupi della gestione della rete. In una prima installazione si è testato il funzionamento con netlab02, facente funzione di nodo di rete, in seguito si è affidata la competenza sulle network a netlab01 unificando, quindi, i ruoli di network e controller node. Questo per riprodurre una struttura più realistica, simulando una situazione in cui netlab01 si affacci su una rete pubblica e rendendo, di fatto, netlab02 un nodo in grado di effettuare il routing. La prima installazione non verrà analizzata in quanto di minor interesse.

Si evidenzia che, come verrà illustrato nei paragrafi successivi, per poter far sì che i pacchetti provenienti dalla rete 10.251.0.0/24 possano dialogare con internet, sul gateway del laboratorio all'indirizzo 192.168.8.0/24 sono state opportunamente modificate le regole di IPTABLES.

4.1 CONFIGURAZIONE FILE DI NEUTRON E LAYOUT DI RETE

Su netlab01 si modificano:

/etc/neutron/neutron.conf

```
[[DEFAULT]]
verbose = True
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

/etc/neutron/plugins/ml2/ml2_conf.ini

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vlan,vxlan
mechanism_drivers = linuxbridge,l2population
extension_drivers = port_security

[ml2_type_flat]
flat_networks = external

[ml2_type_vlan]
network_vlan_ranges = external,vlan:1:1000

[ml2_type_vxlan]
vni_ranges = 1:10000

[securitygroup]
enable_ipset = True
```

/etc/neutron/plugins/ml2/linuxbridge_agent.ini

```
[linux_bridge]
physical_interface_mappings = vlan:eth1,external:eth0

[vxlan]
enable_vxlan = True
local_ip = 192.168.8.1|
l2_population = True

[agent]
prevent_arp_spoofing = True

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver
enable_security_group = True
```

/etc/neutron/l3_agent.ini

```
[DEFAULT]
verbose = True
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
use_namespaces = True
external_network_bridge =

|
```

/etc/neutron/dhcp_agent.ini

```
[DEFAULT]
verbose = True
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = True|
```

Su netlab03 e netlab04:

/etc/neutron/plugins/ml2/linuxbridge_agent.ini

```
[linux_bridge]
physical_interface_mappings = vlan:eth1

[vxlan]
enable_vxlan = True
local_ip = 192.168.8.3 [su netlab04: 192.168.8.4]|
l2_population = True

[agent]
prevent_arp_spoofing = True

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver
enable_security_group = True
```

Per verificare l'effettiva operatività si sfrutta il comando *“neutron agent-list”* e osservando che siano presenti i Linuxbridge agent sul network e sui compute oltre al metadata agent, dhcp agent e L3 agent.

Poi, in primis, si crea la rete con la rispettiva subnet esterna con *“neutron net-create ext-net –router:external True –provider:physical_network external –provider:network_type flat”* e *“neutron subnet-create ext-net –name ext-subnet –allocation-pool start=10.251.0.100,end=10.251.0.200 –disable-dhcp –gateway 10.251.0.2 10.251.0.0/24”*. L'indirizzo del gateway è l'interfaccia eth0 su netlab02, in accordo con il progetto di rete.

Per quanto riguarda le project network si sono dovute utilizzare le VLAN, poiché il routing delle VXLAN non è supportato dalla versione del kernel del sistema operativo in uso. Viene richiesta il kernel 3.14, mentre si dispone del kernel 3.10. I comandi per la creazione delle project network sono: *“neutron net create admin-net –provider:network_type vlan”*, *“neutron subnet-create admin-net –name admin-subnet1 –gateway 172.16.1.1 172.16.1.0/24”* e *“neutron subnet-create admin-net –name admin-subnet2 –gateway 172.16.1.1 172.16.2.0/24”*. Per quanto riguarda il router, invece, *“neutron router-create admin-router1”*, *“neutron router-interface-add admin-subnet1”*, *“neutron router-interface-add admin-subnet2”* e *“neutron router-gateway-set ext-net”*. Infine si creano e assegnano i floating ip sulla rete 10.251.0.0/24 alle virtual machine lanciate via dashboard con *“neutron floatingip-create ext-net”* e *“nova floatingip-create [ID_istanza] [ip_floating]”*.

Perchè la configurazione con le VLAN funzioni è necessario configurare lo switch HP procurve 2524 all'indirizzo 192.168.8.253. Per fare ciò, prima si consulta la tabella relativa alle porte cui sono collegate i vari host, poi si accede allo switch via telnet eseguendo

Host	IP	HWaddress	Switch Port
netlab01	192.168.8.1	90:1b:0e:5c:3d:54	10
netlab02	192.168.8.2	90:1b:0e:5c:3d:10	3
netlab03	192.168.8.3	90:1b:0e:5c:de:bb	9
netlab04	192.168.8.4	90:1b:0e:5c:0c:36	12
netlab05	192.168.8.5	90:1b:0e:5c:3d:0b	22
netlab06	192.168.8.6	90:1b:0e:5c:0c:07	14
netlab07	192.168.8.7	90:1b:0e:5c:c4:13	4
netlab08	192.168.8.8	90:1b:0e:5c:de:b8	20
netlab09	192.168.8.9	90:1b:0e:5c:c4:17	19
netlab10	192.168.8.10	90:1b:0e:5c:c4:15	5
netlab11	192.168.8.11	90:1b:0e:5c:0c:3d	7
netlab12	192.168.8.12	90:1b:0e:5c:c4:3d	1
netlab13	192.168.8.13	90:1b:0e:5c:de:b9	6
netlab14	192.168.8.14	90:1b:0e:5c:c4:5e	21
netlab15	192.168.8.15	90:1b:0e:5c:c4:12	23
netlab16	192.168.8.16	90:1b:0e:5c:de:b7	2

Figura 12: tabella porte dello switch

seguenti comandi:

1. *telenet 192.168.8.253*
2. *show vlan*
3. *configure terminal*
4. *vlan 89*
5. *tagged 10*
6. *tagged 9*
7. *tagged 12*
8. *vlan 92*
9. *tagged 10*
10. *tagged 9*
11. *tagged 12*
12. *exit*

4.2 ANALISI PACKET FLOW

Si è analizzato il flusso dei pacchetti effettuando un ping tra due istanze su reti e host diversi.

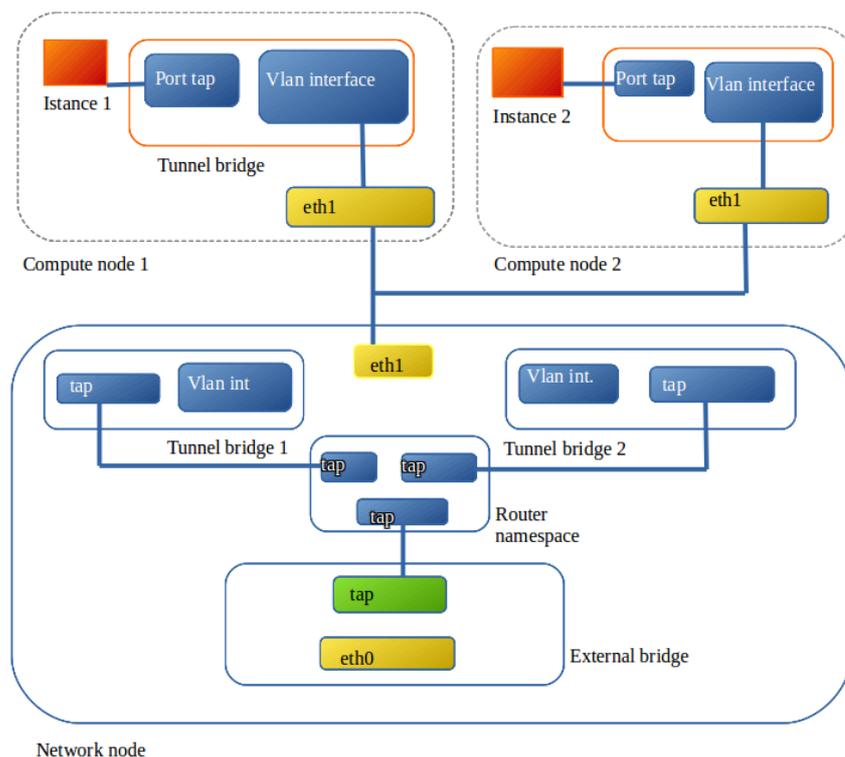


Figura 13: layout di rete

Le due macchine virtuali hanno indirizzi 172.16.1.3 appartenente alla VLAN con ID 89 e 172.16.2.4 della VLAN ID 92 e risiedono rispettivamente su netlab03 e netlab04. Il pacchetto ICMP, in uscita dalla virtual machine con indirizzo 172.16.1.3, attraversa il bridge brqaf5dcff1-b0 dedicato al tunneling su netlab03 uscendo sulla sub-interfaccia creata per la VLAN 89 eth1.89 dell'interfaccia fisica eth1.

```
[nikolas.subrani@netlab03 ~]$ sudo tcpdump -i eth1.89 -n proto ICMP -c 4|
tcpdump: WARNING: eth1.89: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1.89, link-type EN10MB (Ethernet), capture size 65535 bytes
15:42:09.527312 IP 172.16.1.3 > 172.16.2.4: ICMP echo request, id 1531, seq 165, length 64
15:42:09.527834 IP 172.16.2.4 > 172.16.1.3: ICMP echo reply, id 1531, seq 165, length 64
15:42:10.526345 IP 172.16.1.3 > 172.16.2.4: ICMP echo request, id 1531, seq 166, length 64
15:42:10.526934 IP 172.16.2.4 > 172.16.1.3: ICMP echo reply, id 1531, seq 166, length 64
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Tramite lo switch il ping viene indirizzato al network/controller netlab01, qui viene ricevuto sulla eth1.89, porta del tunnel bridge brq58bf34fa-d1, e poi passato al router virtuale, il quale invia il pacchetto all'interfaccia eth1.92 della VLAN 92.

```
[nikolas.subrani@netlab01 ~]$ sudo tcpdump -n -c 6 proto ICMP -i eth1.92
tcpdump: WARNING: eth1.92: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1.92, link-type EN10MB (Ethernet), capture size 65535 bytes
15:49:44.939287 IP 172.16.1.3 > 172.16.2.4: ICMP echo request, id 1531, seq 620, length 64
15:49:44.939609 IP 172.16.2.4 > 172.16.1.3: ICMP echo reply, id 1531, seq 620, length 64
15:49:45.939295 IP 172.16.1.3 > 172.16.2.4: ICMP echo request, id 1531, seq 621, length 64
15:49:45.939633 IP 172.16.2.4 > 172.16.1.3: ICMP echo reply, id 1531, seq 621, length 64
15:49:46.939241 IP 172.16.1.3 > 172.16.2.4: ICMP echo request, id 1531, seq 622, length 64
15:49:46.939563 IP 172.16.2.4 > 172.16.1.3: ICMP echo reply, id 1531, seq 622, length 64
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

Sempre lo switch fisico indirizza il pacchetto a netlab04, dove passa per l'interfaccia eth1.92, integrata nel tunnel bridge, e poi alla macchina virtuale.

```
[nikolas.subrani@netlab04 ~]$ sudo tcpdump -c 4 -n proto ICMP -i eth1.92
tcpdump: WARNING: eth1.92: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1.92, link-type EN10MB (Ethernet), capture size 65535 bytes
15:51:46.059520 IP 172.16.1.3 > 172.16.2.4: ICMP echo request, id 1531, seq 741, length 64
15:51:46.059672 IP 172.16.2.4 > 172.16.1.3: ICMP echo reply, id 1531, seq 741, length 64
15:51:47.059414 IP 172.16.1.3 > 172.16.2.4: ICMP echo request, id 1531, seq 742, length 64
15:51:47.059556 IP 172.16.2.4 > 172.16.1.3: ICMP echo reply, id 1531, seq 742, length 64
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

In un ping tra VM sulla stessa rete, invece, non è necessario che i packets attraversino il network node, poiché il tunnel consente una consegna diretta, naturalmente sempre passando dallo switch che implementa le VLAN.

Al contrario, un ping verso Internet deve passare dal controller e dal router virtuale. Quest'ultimo applica il DNAT sostituendo l'indirizzo della rete 172.16.1.0/24 con il floating ip assegnato sulla ext-net, prima di eseguire il forwarding verso l'external bridge brqaa913549-97.

```
[nikolas.subrani@netlab01 ~]$ sudo tcpdump -i brqaa913549-97 -n proto ICMP -c 6
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on brqaa913549-97, link-type EN10MB (Ethernet), capture size 65535 bytes
14:47:02.034652 IP 10.251.0.101 > 8.8.8.8: ICMP echo request, id 1429, seq 973, length 64
14:47:02.045513 IP 8.8.8.8 > 10.251.0.101: ICMP echo reply, id 1429, seq 973, length 64
14:47:03.036068 IP 10.251.0.101 > 8.8.8.8: ICMP echo request, id 1429, seq 974, length 64
14:47:03.045752 IP 8.8.8.8 > 10.251.0.101: ICMP echo reply, id 1429, seq 974, length 64
14:47:04.037262 IP 10.251.0.101 > 8.8.8.8: ICMP echo request, id 1429, seq 975, length 64
14:47:04.048255 IP 8.8.8.8 > 10.251.0.101: ICMP echo reply, id 1429, seq 975, length 64
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

CAPITOLO 5.

SCENARIO CLASSIC CON OPENVSWITCH

In questa configurazione si utilizza come plugin ML2 Open vSwitch, un virtual switch open source. Sostanzialmente il funzionamento non si discosta molto da quello analizzato nel Capitolo 4, tranne che viene creato un bridge in più: l'integration bridge. Un linuxbridge è sempre presente sia sul network node, sia sui compute node, per poter implementare le IPTABLES in quanto non presenti in Open vSwitch.

L'integration bridge al suo interno aggrega porte tap e porte patch. Le patch provvedono a mettere in comunicazione il br-int con il tunnel bridge, il bridge vlan (se utilizzato) e l'external bridge sul network node.

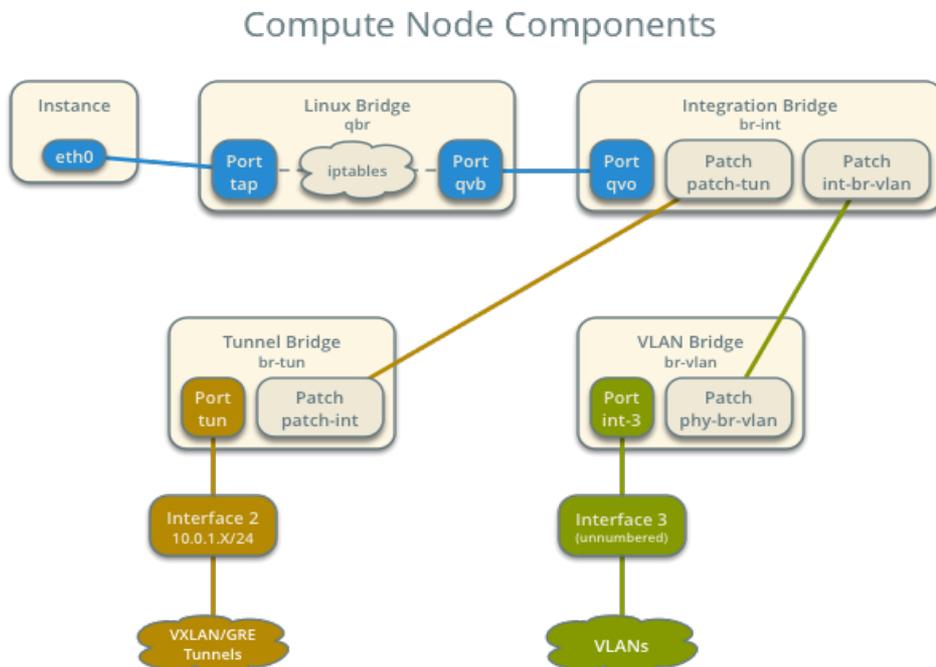


Figura 14: componenti dei compute node

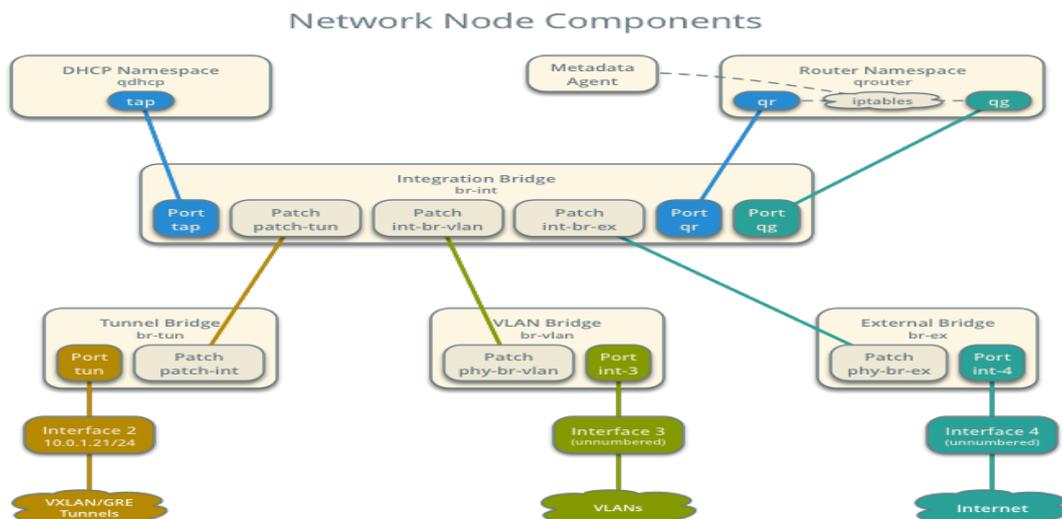


Figura 15: componenti del network node

Anche in questo scenario i requisiti fisici raccomandati dagli sviluppatori sono almeno 3 reti con 2 NIC sui compute node e 3 NIC sul network node. Le reti raccomandate sono quella di management, quella dei tunnel e la rete esterna. In base alle risorse a disposizione e come già testato negli scenari visti nei capitoli precedenti, la rete 192.168.8.0/24 è stata scelta come rete tunnel e di management e la rete 10.251.0.0/24 come rete esterna. Alla stessa maniera della configurazione Classic con Linuxbridge, il network node è stato prima installato su netlab02, salvo poi trasferirlo su netlab01.

Il protocollo di tunneling scelto è il Generic Routing Encapsulation in quanto era il protocollo standard per le release di Openstack precedenti a Liberty e perché il routing delle VXLAN non è supportato dal kernel 3.10.

5.1 CONFIGURAZIONE FILE DI NEUTRON E LAYOUT DI RETE

Open vSwitch (OVS) non è un componente nativo su Linux, dunque è indispensabile scaricarlo i file con “*sudo yum install openstack-neutron openstack-neutron-ml2 openstack-neutron-openvswitch python-neutronclient which*” su netlab01 e “*yum install openstack-neutron openstack-neutron-ml2 openstack-neutron-penvswitch*” su netlab03 e netlab04. Si prosegue con la modifica dei file.

Su netlab01:

/etc/neutron/neutron.conf

```
[[DEFAULT]]
verbose = True
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

/etc/neutron/plugins/ml2/openvswitch_agent.ini

```
[ml2]
type_drivers = flat,vlan,gre,vxlan
tenant_network_types = vlan,gre,vxlan
mechanism_drivers = openvswitch,l2population
extension_drivers = port_security

[ml2_type_flat]
flat_networks = external

[ml2_type_vlan]
network_vlan_ranges = external,vlan:1:1000

[ml2_type_gre]
tunnel_id_ranges = 1:1000

[ml2_type_vxlan]
vni_ranges = 1:1000

[securitygroup]
enable_ipset = True

[ovs]
local_ip = 192.168.8.1
bridge_mappings = vlan:br-vlan,external:br-ex

[agent]
tunnel_types = gre,vxlan
l2_population = True
prevent_arp_spoofing = True

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
enable_security_group = True
```

/etc/neutron/l3_agent.ini

```
[DEFAULT]
verbose = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
external_network_bridge =|
```

/etc/neutron/dhcp_agent.ini

```
[DEFAULT]
verbose = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = True
```

su netlab03 e netlab04:

/etc/neutron/plugins/ml2/openvswitch_agent.ini

```
[ovs]
local_ip = 192.168.8.3 [192.168.8.4 su netlab04]
bridge_mappings = vlan:br-vlan

[agent]
tunnel_types = gre,vxlan
l2_population = True
prevent_arp_spoofing = True

[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
enable_security_group = True
```

All'atto della verifica con “*neutron agent-list*”, dopo aver modificato lo script mostrato nel paragrafo 2.5 in modo da avviare Open vSwitch, si è notato come l' Open vSwitch agent non fosse attivato né sul network/controller né sui compute. Controllando nel file di log */var/log/neutron/openvswitch_agent.log* si è appurato che la mancata creazione di alcuni bridge, specificati nel file *openvswitch_agent.ini*, impediva l'avvio del componente. Su netlab01, utilizzando i comandi *ovs-vsctl*, si sono dovuti aggiungere manualmente il br-ex e br-vlan con “*sudo ovs-vsctl add-br br-ex*” e “*sudo add-br br-vlan*”; mentre su netlab02 si è creato soltanto il br-vlan. Con “*sudo ovs-vsctl show*” si è verificata la corretta creazione dei bridge, ma si è notato come mancasse qualunque connessione con le interfacce fisiche eth1 e eth0 nel br-provider e nel br-ex su netlab01.

```

[nikolas.subrani@netlab01 ~]$ sudo ovs-vsctl show
9b6ec525-f1a4-49d0-ac9c-faaee877235c
  Bridge br-provider
    Port br-provider
      Interface br-provider
        type: internal
    Port phy-br-provider
      Interface phy-br-provider
        type: patch
        options: {peer=int-br-provider}
  Bridge br-ex
    Port phy-br-ex
      Interface phy-br-ex
        type: patch
        options: {peer=int-br-ex}
    Port br-ex
      Interface br-ex
        type: internal

```

Figura 16: OVS privo di alcuni bridge

Tale mancanza avrebbe reso lo switch virtuale OVS un' entità scollegata dalle interfacce fisiche, perciò, sempre utilizzando i comandi `ovs-vsctl`, si sono aggiunte come porte `eth1` al `br-provider` e `eth0` al `br-ex`: “`sudo ovs-vsctl add-port br-provider eth1`” e “`sudo add-port br-ex eth0`”.

Inoltre, spostando le interfacce fisiche all'interno dello switch OVS, si genera una perdita di connettività. Per ovviare a questa problematica serve assegnare gli indirizzi IP e MAC di `eth1` e `eth0` ai bridge che le contengono, tramite uno script. Bisogna anche aver cura di modificare la tabella di routing, inserendo come default gateway il `netlab02` in modo da farlo combaciare con quello da specificare al momento della creazione delle reti di Openstack con i comandi di Neutron.

```
#!/bin/sh
```

```

ifconfig br-ex 10.251.0.1/24
ifconfig br-provider 192.168.8.1/24
ip addr del 192.168.8.1/24 dev eth1
ip addr del 10.251.0.1/24 dev eth0
ovs-vsctl set bridge br-ex other-config:hwaddr=78:44:76:c3:bb:17
ovs-vsctl set bridge br-provider other-config:hwaddr=90:1b:0e:5c:3d:54
route add default gw 10.251.0.2

```

Accertata la connettività con dei ping, si passa alla creazione delle reti. La rete esterna si stabilisce con “`neutron net-create ext-net --router:external True --provider:physical_network external --provider:network_type flat`” e “`neutron subnet-create ext-net --name ext-subnet --allocation-pool start=10.251.0.100,end=10.251.0.200 --disable-dhcp --gateway 10.251.0.2 10.251.0.0/24`”. Per le reti private: “`neutron net-create admin-net --provider:network_type gre`”, “`neutron subnet-create admin-net --name admin-subnet1 --gateway 172.16.1.1`”.

172.16.1.0/24” e “*neutron subnet-create admin-net --name admin-subnet2 --gateway 172.16.2.1 172.16.2.0/24*”. Infine si origina il virtual router “*neutron router create admin-router*”, “*neutron router-interface-add admin-routeradmin-subnet1*”, “*neutron router-interface-add admin-routeradmin-subnet2*” e “*neutron router-gateway-set admin-router ext-net*”.

Dopo aver lanciato le virtual machine, si è tentato l’accesso da remoto (via SSH) come da prassi. La connessione SSH, però, non riusciva ad aprirsi a causa del Path Maximum Transfer Unit Discovery (PMTUD). L’utilizzo del GRE come protocollo di tunneling implica l’inserimento di una intestazione nei pacchetti. All’interno del cluster di Openstack il MTU di ogni interfaccia è settato al valore standard di 1500 byte e questo causa il fallimento dell’apertura della connessione SSH, in quanto il PMTUD si attende un valore MTU più alto, in grado contenere anche l’overhead. Per ovviare a questo problema, sul network node nel file */etc/neutron/dhcp_agent* nella sezione [DEFAULT] si inserisce questa riga

- `dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf`

e si crea il file */etc/neutron/dnsmasq-neutron.conf* con una sola riga di codice

- `dhcp-option-force=26,1450`

Il compito di queste righe di codice è impostare il MTU delle interfacce delle virtual machine a 1450 byte. I 50 di byte di scarto sono sufficienti a contenere l’intestazione del protocollo GRE.

5.2 ANALISI PACKET FLOW

Avendo scelto il protocollo GRE, si può notare come venga stabilito un tunnel tra tutti gli host fisici del cluster, con porte nei rispettivi br-tun, per separare il traffico generato dalle macchine virtuali, lanciate nel cluster. Ad esempio, su netlab03 vengono stabiliti due tunnel: uno tra sé stesso e netlab01 e uno verso netlab04. Così facendo, se due VM sono sulla stessa rete virtuale, viene effettuata una consegna diretta senza dover attraversare il controller. Di seguito si riporta la struttura dello switch OVS sul controller/network e sul compute netlab03 (su netlab04 è speculare):

```
[nikolas.subrani@netlab01 ~]$ sudo ovs-vsctl show
9b6ec525-f1a4-49d0-ac9c-faaee877235c
Bridge br-provider
```

```
Port br-provider
  Interface br-provider
  type: internal
Port "eth1"
  Interface "eth1"
Port phy-br-provider
  Interface phy-br-provider
  type: patch
  options: {peer=int-br-provider}
```

Bridge br-ex

```
Port phy-br-ex
  Interface phy-br-ex
  type: patch
  options: {peer=int-br-ex}
```

```
Port "eth0"
  Interface "eth0"
```

```
Port br-ex
  Interface br-ex
  type: internal
```

Bridge br-vlan

```
Port br-vlan
  Interface br-vlan
  type: internal
Port phy-br-vlan
  Interface phy-br-vlan
  type: patch
  options: {peer=int-br-vlan}
```

Bridge br-tun

```
fail_mode: secure
```

```
Port "gre-c0a80804"
Interface "gre-c0a80804"
type: gre
options: {df_default="true", in_key=flow, local_ip="192.168.8.1", out_key=flow,
remote_ip="192.168.8.4"}
```

Port "gre-c0a80803"

Interface "gre-c0a80803"

type: gre

options: {df_default="true", in_key=flow, local_ip="192.168.8.1",
out_key=flow, remote_ip="192.168.8.3"}

Port br-tun

Interface br-tun

type: internal

Port patch-int

Interface patch-int

type: patch

options: {peer=patch-tun}

Bridge br-int

fail_mode: secure

Port patch-tun

Interface patch-tun

type: patch

options: {peer=patch-int}

Port int-br-ex

Interface int-br-ex

type: patch

options: {peer=phy-br-ex}

Port "tap55b48351-25"

tag: 1

Interface "tap55b48351-25"

type: internal

Port "tapf5ebf44b-db"

Interface "tapf5ebf44b-db"

type: internal

Port int-br-provider

Interface int-br-provider

type: patch

options: {peer=phy-br-provider}

Port br-int

```
Interface br-int
  type: internal
Port "qg-3101e72e-63"
  tag: 2
    Interface "qg-3101e72e-63"
      type: internal
Port int-br-vlan
  Interface int-br-vlan
    type: patch
    options: {peer=phy-br-vlan}
Port "qr-4c1d3267-83"
  tag: 1
    Interface "qr-4c1d3267-83"
      type: internal
      ovs_version: "2.4.0"
```

```
[nikolas.subrani@netlab03 ~]$ sudo ovs-vsctl show
```

```
[sudo] password for nikolas.subrani:
```

```
83f3240e-5665-4142-a952-f5e1db633286
```

```
Bridge br-int
```

```
fail_mode: secure
```

```
Port int-br-vlan
  Interface int-br-vlan
    type: patch
    options: {peer=phy-br-vlan}
Port br-int
  Interface br-int
    type: internal
Port "qvo075c437c-df"
  tag: 1
    Interface "qvo075c437c-df"
Port patch-tun
  Interface patch-tun
    type: patch
```

```

        options: {peer=patch-int}
    Port int-br-provider
        Interface int-br-provider
        type: patch
        options: {peer=phy-br-provider}
Bridge br-vlan
    Port phy-br-vlan
        Interface phy-br-vlan
        type: patch
        options: {peer=int-br-vlan}
    Port br-vlan
        Interface br-vlan
        type: internal
Bridge br-tun
fail_mode: secure
    Port br-tun
        Interface br-tun
        type: internal
    Port "gre-c0a80801"
        Interface "gre-c0a80801"
        type: gre
        options: {df_default="true", in_key=flow, local_ip="192.168.8.3",
out_key=flow, remote_ip="192.168.8.1"}
    Port "gre-c0a80804"
        Interface "gre-c0a80804"
        type: gre
        options: {df_default="true", in_key=flow, local_ip="192.168.8.3",
out_key=flow, remote_ip="192.168.8.4"}
    Port patch-int
        Interface patch-int
        type: patch
        options: {peer=patch-tun}
ovs_version: "2.4.0"

```

Se si effettua il ping da una instance verso un indirizzo esterno, catturando i pacchetti sull'interfaccia fisica eth1 del compute si può osservare l'incapsulamento GRE applicato per dialogare con il network node, dove risiede il router virtuale che sottende alle reti private.

```
[nikolas.subrani@netlab03 ~]$ sudo tcpdump proto GRE -n -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
11:59:38.572224 IP 192.168.8.3 > 192.168.8.1: GREv0, key=0xe, length 106: IP 172.16.1.3 > 8.8.8.8: ICMP echo request, id 2191, seq 87, length 64
11:59:38.580934 IP 192.168.8.1 > 192.168.8.3: GREv0, key=0xe, length 106: IP 8.8.8.8 > 172.16.1.3: ICMP echo reply, id 2191, seq 87, length 64
```

```
[nikolas.subrani@netlab01 ~]$ sudo tcpdump -n proto GRE -i eth1
tcpdump: WARNING: eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
12:06:32.216360 IP 192.168.8.3 > 192.168.8.1: GREv0, key=0xe, length 106: IP 172.16.1.3 > 8.8.8.8: ICMP echo request, id 2191, seq 500, length 64
12:06:32.224597 IP 192.168.8.1 > 192.168.8.3: GREv0, key=0xe, length 106: IP 8.8.8.8 > 172.16.1.3: ICMP echo reply, id 2191, seq 500, length 64
```

Similmente avviene la stessa cosa per gli echo reply uscenti dal netlab01.

È utile notare come il GRE si applichi a collegamenti punto-punto e operi a livello 3. Infatti eseguendo uno sniff sulla porta patch qvb075c437c-df del br-int collegata all'interfaccia della macchina virtuale, si nota che l'incapsulamento non è presente, in accordo con le ipotesi appena fatte. Si ricorda che il router applica il NAT, sostituendo l'indirizzo sorgente con il floating ip assegnato.

```
[nikolas.subrani@netlab03 ~]$ sudo tcpdump -c 4 -n proto ICMP -i qvb075c437c-df
tcpdump: WARNING: qvb075c437c-df: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on qvb075c437c-df, link-type EN10MB (Ethernet), capture size 65535 bytes
11:53:12.371094 IP 172.16.1.3 > 8.8.8.8: ICMP echo request, id 2188, seq 223, length 64
11:53:12.379974 IP 8.8.8.8 > 172.16.1.3: ICMP echo reply, id 2188, seq 223, length 64
11:53:13.372746 IP 172.16.1.3 > 8.8.8.8: ICMP echo request, id 2188, seq 224, length 64
11:53:13.380951 IP 8.8.8.8 > 172.16.1.3: ICMP echo reply, id 2188, seq 224, length 64
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Per quanto riguarda il ping tra instance su una stessa rete, si osserva come i pacchetti attraversino solamente il tunnel tra i due compute per una consegna diretta.

```
[nikolas.subrani@netlab04 ~]$ sudo tcpdump -n -i eth1 proto GRE -c 4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
12:14:34.565281 IP 192.168.8.3 > 192.168.8.4: GREv0, key=0xe, length 106: IP 172.16.1.3 > 172.16.1.6: ICMP echo request, id 2192, seq 43, length 64
12:14:34.565458 IP 192.168.8.4 > 192.168.8.3: GREv0, key=0xe, length 106: IP 172.16.1.6 > 172.16.1.3: ICMP echo reply, id 2192, seq 43, length 64

[nikolas.subrani@netlab03 ~]$ sudo tcpdump -n proto gre -i eth1 -c 4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
12:17:36.696318 IP 192.168.8.3 > 192.168.8.4: GREv0, key=0xe, length 106: IP 172.16.1.3 > 172.16.1.6: ICMP echo request, id 2192, seq 225, length 64
12:17:36.696856 IP 192.168.8.4 > 192.168.8.3: GREv0, key=0xe, length 106: IP 172.16.1.6 > 172.16.1.3: ICMP echo reply, id 2192, seq 225, length 64
```

Al contrario se si esegue il ping tra istanze su reti diverse, i tunnel attraversati sono due e tramite NAT l'indirizzo sorgente della rete 172.16.x.0/24 viene sostituito con il floating ip. I tunnel sono quelli netlab04-netlab01 e netlab01-netlab03.

```
[nikolas.subrani@netlab01 ~]$ sudo tcpdump -i eth1 -n proto GRE
tcpdump: WARNING: eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
12:44:36.109023 IP 192.168.8.3 > 192.168.8.1: GREv0, key=0x16, length 106: IP 172.16.2.3 > 10.251.0.108: ICMP echo request, id 1387, seq 38, length 64
12:44:36.109076 IP 192.168.8.1 > 192.168.8.3: GREv0, key=0xe, length 106: IP 10.251.0.113 > 172.16.1.3: ICMP echo request, id 1387, seq 38, length 64
```

```
[nikolas.subrani@netlab01 ~]$ sudo tcpdump -i eth1 -n proto GRE
[sudo] password for nikolas.subrani:
tcpdump: WARNING: eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
12:47:16.490581 IP 192.168.8.4 > 192.168.8.1: GREv0, key=0x16, length 106: IP 172.16.2.4 > 10.251.0.108: ICMP echo request, id 1386, seq 23, length 64
12:47:16.490627 IP 192.168.8.1 > 192.168.8.3: GREv0, key=0xe, length 106: IP 10.251.0.114 > 172.16.1.3: ICMP echo request, id 1386, seq 23, length 64
```

Il tragitto dei pacchetti ricevuto in compute node all'interno dello switch OVS segue questi passi:

1. interfaccia fisica eth1
2. br-tun
3. br-int
4. Linuxbridge (in modo da applicare le Iptables)
5. interfaccia della virtual machine

Sul network node, i packets, dopo esser stati ricevuti sul br-tun passano al br-int dove vengono inviati al router virtuale dove viene eseguito il forwarding. I pacchetti processati dal router vengono nuovamente trasmessi al br-int, da quale possono uscire nuovamente sul br-tun, se la destinazione è interna a Openstack o attraversare il br-ex per andare verso l'esterno.

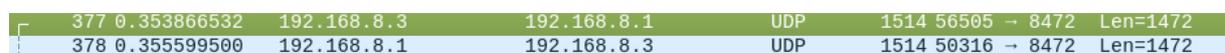
CAPITOLO 6.

APPROFONDIMENTO SULLE INTESTAZIONI E DIMENSIONE DEI PACCHETTI NEGLI SCENARI PROPOSTI

Le configurazioni analizzate, come visto, implementano diversi metodi di tunneling per separare il traffico nelle reti delle istanze.. Si sono effettuati dei ping, che attraversassero i tunnel generati, con dimensione del pacchetto variabile al fine di scoprire il massimo possibile per il datagramma, tenendo presente come valore MTU 1450 in modo da avere un confronto equo tra tutte le configurazioni.

La sintassi del comando ping sfruttato è “*ping -s [dimensione pacchetto] [indirizzo destinazione]*”, si ricorda che l’ICMP introduce un header di 8 byte il quale è escluso dalla dimensione del pacchetto specificato nel comando, ma andrà ugualmente conteggiato.

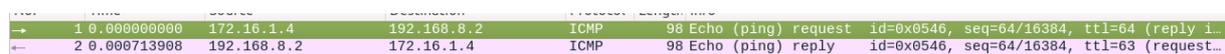
Nel primo scenario, quello di default affrontato nel Capitolo 3, si utilizzano le VXLAN. Questo tipo di protocollo lavora a livello 4 con intestazioni UDP, infatti eseguendo una cattura su eth1 con l’analizzatore Wireshark si osserva un dialogo tra il compute e il controller (tra gli indirizzi 192.168.8.3 e 192.168.8.1).



377	0.353866532	192.168.8.3	192.168.8.1	UDP	1514	56505	→	8472	Len=1472
378	0.355599500	192.168.8.1	192.168.8.3	UDP	1514	50316	→	8472	Len=1472

Figura 17: cattura wireshark dei pacchetti UDP

Catturando sull’interfaccia del bridge, invece, si notano i pacchetti ICMP con l’indirizzo sorgente correttamente appartenente alla rete private.



1	0.000000000	172.16.1.4	192.168.8.2	ICMP	98	Echo (ping) request	id=0x0546, seq=64/16384, ttl=64 (reply i...
2	0.000713908	192.168.8.2	172.16.1.4	ICMP	98	Echo (ping) reply	id=0x0546, seq=64/16384, ttl=63 (request...

Figura 18: cattura wireshark sul bridge dei pacchetti ICMP

La dimensione massima dei pacchetti è 1430 byte (1422 di dati e 8 di intestazione ICM), a cui va sommata l’intestazione IP di 20 byte. Il totale è 1450 byte, come ci si attendeva.

```
11:13:29.781394 IP (tos 0x0, ttl 64, id 47245, offset 0, flags [DF], proto ICMP (1), length 1450)
 172.16.1.4 > 192.168.8.2: ICMP echo request, id 1345, seq 4, length 1430
```

Figura 19: cattura pacchetti di dimensione massima con VXLAN

Nello scenario Classic con Linux Bridge, presentato nel capitolo 4, si fa uso delle VLAN tagged che sfruttano lo standard IEEE 802.1Q. Non vi è quindi un incapsulamento dei pacchetti. Si è ricercato, quindi, il massimo ottenibile prima che il pacchetto venga frammentato. La frammentazione avviene quando il pacchetto supera i 1500 byte totali.

```
10:36:32.577024 IP (tos 0x0, ttl 63, id 8187, offset 0, flags [none], proto ICMP (1), length 1500)
 10.251.0.181 > 10.251.0.2: ICMP echo request, id 1399, seq 9, length 1480
10:37:47.693617 IP (tos 0x0, ttl 64, id 17527, offset 0, flags [none], proto ICMP (1), length 1500)
 172.16.1.4 > 10.251.0.2: ICMP echo request, id 1399, seq 84, length 1480
```

Figura 20: cattura pacchetti di dimensione massima VLAN

Il ping quindi ha dimensione di 1480 byte (1472 di dati più 8 di ICMP).

Nel terzo scenario all'interno del cluster vengono creati dei tunnel GRE. Il GRE opera su connessioni punto-punto. Al pacchetto ICMP viene aggiunta una intestazione di 42 byte, se il ping è di dimensione standard, il datagramma che attraversa il tunnel è di 106 byte e la dimensione massima è di 1438 byte (1430 di dati e 8 di ICMP header).

```
12:47:16.490581 IP 192.168.8.4 > 192.168.8.1: GREv0, key=0x16, length 106: IP 172.16.2.4 > 10.251.0.108: ICMP echo request, id 1386, seq 23, length 64
12:47:16.490627 IP 192.168.8.1 > 192.168.8.3: GREv0, key=0xe, length 106: IP 10.251.0.114 > 172.16.1.3: ICMP echo request, id 1386, seq 23, length 64
```

Figura 21: incapsulamento GRE con pacchetti standard

```
12:03:28.635112 IP 192.168.8.1 > 192.168.8.3: GREv0, key=0x27, length 1480: IP 10.251.0.2 > 172.16.4.3: ICMP echo request, id 13293, seq 9, length 1438
12:03:28.635397 IP 192.168.8.3 > 192.168.8.1: GREv0, key=0x27, length 1480: IP 172.16.4.3 > 10.251.0.2: ICMP echo reply, id 13293, seq 9, length 1438
```

Figura 22: incapsulamento GRE con pacchetti di dimensione massima

In tabella 1 vengono riassunte le dimensioni massime dei pacchetti negli scenari analizzati per l'installazione di Openstack discussa in questo documento.

Protocollo	Dimensione massima del pacchetto (byte)
VXLAN	1430
VLAN	1480
GRE	1438

Tabella 1: dimensioni massime consentite dai diversi protocolli

CAPITOLO 7.

CONCLUSIONI

In questo documento si sono analizzate le varie possibilità di configurazione del servizio Openstack Networking e si è testato il funzionamento di un cluster con determinate limitazioni hardware.

Innanzitutto si è provato come con solo una rete fisica che interconnette i 4 nodi sia possibile implementare Neutron correttamente per tutti gli scenari proposti nei capitoli precedenti. Non è stato, invece, possibile implementare la configurazione *provider con Open vSwitch*, analoga a quella del Capitolo 3, dove al posto del Linuxbridge viene utilizzato OVS poiché quando si annette l'interfaccia eth1 allo switch virtuale si perde connettività. Questa perdita di connettività, per come è strutturata l'infrastruttura fisica e virtuale dello scenario, senza una rete di management separata dalla rete di tunneling non è ripristinabile, rendendo, dunque, inutilizzabile tutta l'infrastruttura..

Inoltre, si è appurato come la configurazione di default sia più semplice sia per risorse sia come utilizzo pratico. Rispetto a quelle presentate nei capitoli 4 e 5, vi è la possibilità di avere un sistema “pronto all'uso” (lanciando le istanze sulla rete chiamata public nel Capitolo 3) senza essere obbligati a progettare e creare reti e router virtuali, a prezzo però di minor flessibilità e scalabilità per l'utente senza privilegi di amministrazione.

Gli scenari classici dei Capitoli 4 e 5 sono più complessi dal punto di vista dell'implementazione e meno immediati all'utilizzo, in quanto vi è una divisione logica tra il piano dei componenti di rete virtuali di Openstack e il piano delle risorse fisiche. Questo si traduce in maggior libertà per l'utente nella progettazione delle reti e più sicurezza data dalla separazione del traffico generato dalle macchine virtuali da quello degli host fisici.

Tutta la sperimentazione è stata effettuata, come descritto all'inizio del documento, usufruendo della release Liberty del software Openstack. Le considerazioni fatte sono valide anche per l'ultima versione, Mitaka, in quanto le modifiche apportate sono solo piccole migliorie e aggiunte di ulteriori feature. Tra funzionalità inserite nell'ultima release si segnalano la dimensione minima della memoria RAM impostabile a 64 MB per il flavour

dell'immagine disco tipo *.nano* (in Liberty il minimo è 512 MB con il tipo *.tiny*) e la creazione di un nuovo scenario, Classic con MacVtap.

BIBLIOGRAFIA

- <http://www.openstack.org/>
- <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublicatio>
[n800-145.pdf](#)
- <http://docs.openstack.org/liberty/networking-guide/deploy.html>
- <http://docs.openstack.org/liberty/install-guide-rdo/>
- <https://review.openstack.org/#/c/301014/3>
- https://en.wikipedia.org/wiki/Generic_Routing_Encapsulation
- https://en.wikipedia.org/wiki/Virtual_Extensible_LAN
- https://en.wikipedia.org/wiki/Virtual_LAN
- <https://tools.ietf.org/html/rfc1191>
- https://en.wikipedia.org/wiki/IEEE_802.1Q
- <https://tools.ietf.org/html/rfc2784>
- https://en.wikipedia.org/wiki/Cloud_computing
- https://en.wikipedia.org/wiki/Software-defined_networking
- <http://docs.openstack.org/mitaka/install-guide-rdo/>
- <http://docs.openstack.org/mitaka/networking-guide/>