



UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Enrutamiento en el Datacenter

Matias Daniel Capucho Cirlinas

Santiago Elizondo Sosa

Programa de Grado en Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Marzo de 2020



UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Enrutamiento en el Datacenter

Matias Daniel Capucho Cirlinas

Santiago Elizondo Sosa

Tesis de Grado de Ingeniería en Computación, presentada al tribunal evaluador de la Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Ingeniero en Computación.

Director:

Dr. Eduardo Grampín

Codirectores:

Msc. Martín Giachino

Dr. Alberto Castro

Montevideo – Uruguay

Marzo de 2020

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

Dr. Santiago Iturriaga

Ing. Javier Barreiro

Ing. Mariela De León

Montevideo – Uruguay

Marzo de 2020

RESUMEN

El presente documento estudia los aspectos fundamentales de la plataforma OpenStack para instalación y gestión de Datacenters. A su vez, relata el proceso de diseño, instalación, y configuración básica de un ambiente de pruebas mediante la utilización de Ansible como herramienta de automatización. Se especifican los pasos necesarios a seguir y se detallan puntos sumamente relevantes a tener en cuenta. Una vez alcanzado un ambiente funcional, se detallan procesos de gestión tales como administración de la plataforma a través del gestor web, procedimientos de actualización de versiones y tareas de escalamiento horizontal del Datacenter. Por otro lado, se realiza un análisis del módulo de red Neutron enfocado en resolver la comunicación entre máquinas virtuales dentro de la plataforma, particularmente buscando responder cómo realiza la comunicación entre las redes físicas de la infraestructura y las redes virtuales creadas por el usuario final.

Palabras claves:

datacenter, redes de computadoras, openstack, virtualización, computación en la nube.

ABSTRACT

This document studies the fundamental aspects of OpenStack, the Datacenters installation and management platform. At the same time, reports the design, installation and basic configuration process of a testing environment through Ansible as an automation engine. The required steps to follow are specified, detailing the most relevant points that should be taken into account. Once a functional environment has been reached, management processes such as platform administration through the web dashborad, version updates proceedings and horizontal scaling tasks are detailed. On the other hand, it presents an analysis of Neutron, the network module in charge of solving the communication between virtual machines inside the platform, specifically trying to undestand the way that infrastructure physical networks and virtual tenant networks are communicated.

Keywords:

datacenter, computer networks, openstack, virtualization, cloud computing.

Lista de figuras

2.1	Hipervisores. Extraída de [17].	4
2.2	Virtualización vs Contenedores. Extraída de [18].	6
2.3	Proceso de replicación en OSDs de Ceph. Extraída de [6].	10
2.4	Proceso de almacenamiento en pools de Ceph. Extraída de [6].	11
2.5	Proceso de almacenamiento en PGs de Ceph. Extraída de [6].	11
2.6	Relacionamiento entre módulos core. Extraída de [19]	13
2.7	Arquitectura simplificada de Neutron. Extraída de [10]	19
2.8	Diagrama de componentes de Open vSwitch	24
2.9	Routers virtuales previo a una falla. Extraída de [65].	28
2.10	Routers virtuales luego de una falla. Extraída de [65].	28
2.11	Componentes de red en OpenStack. [29].	29
2.12	Despliegue de servicios OpenStack en contenedores. Extraída de [25].	31
3.1	Arquitectura diseñada para instalación Queens	38
3.2	Arquitectura diseñada para instalación Stein	40
3.3	Diagrama con la distribución de los servicios	41
5.1	Diagrama de arquitectura para el escenario 1 de Linux Bridge	52
5.2	Paquete ARP request capturado en la interfaz eth0 de la instancia 1 .	55
5.3	Paquete ARP request encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1	56
5.4	Paquete ARP reply encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1	57
5.5	Paquete ARP reply capturado en la interfaz eth0 de la instancia 1 . .	58
5.6	Paquete ICMP request capturado en la interfaz eth0 de la instancia 1	59
5.7	Paquete ICMP request encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1	59
5.8	Diagrama de arquitectura para el escenario 2 de Linux Bridge	60
5.9	Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1	66

5.10	Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 2	66
5.11	Diagrama de arquitectura para el escenario 3 de Linux Bridge	67
5.12	Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1	71
5.13	Paquete ARP request taggeado con el VLAN ID 100 capturado en la interfaz br-vlan en el nodo de red	72
5.14	Paquete ICMP echo request capturado en la interfaz br-vlan del nodo de red	73
5.15	Diagrama de arquitectura para el escenario 4 de Linux Bridge	74
5.16	Paquete ICMP echo request etiquetado con el VLAN ID 100 capturado en la interfaz eth3 del router físico	76
5.17	Paquete ICMP echo request capturado en la interfaz qg del router de Neutron	77
5.18	Paquete ICMP echo request capturado en la interfaz qr del router de Neutron	77
5.19	Diagrama de arquitectura para el escenario 1 de Open vSwitch	79
5.20	Paquete ARP request capturado en la interfaz eth0 de la instancia 1	85
5.21	ARP request encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1	87
5.22	ARP reply encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1	91
5.23	Paquete ARP reply capturado en la interfaz eth0 de la instancia 1	91
5.24	Paquete ICMP request capturado en la interfaz eth0 de la instancia 1	92
5.25	Paquete ICMP request encapsulado en VXLAN 19 capturado en el bridge br-vxlan en el nodo de cómputo 1	93
5.26	Diagrama de arquitectura para el escenario 2 de Open vSwitch	94
5.27	Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1	99
5.28	Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 2	99
5.29	Diagrama de arquitectura para el escenario 3 de Open vSwitch	100
5.30	Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1	103
5.31	Paquete ARP request taggeado con el VLAN ID 100 capturado en la interfaz br-vlan en el nodo de red	105
5.32	Paquete ICMP echo request capturado en la interfaz br-vlan del nodo de red	106

5.33	Diagrama de arquitectura para el escenario 4 de Open vSwitch	107
5.34	Paquete ICMP echo request taggeado con el VLAN ID 100 capturado en la interfaz eth3 del router físico	109
5.35	Paquete ICMP echo request capturado en la interfaz qg del router de Neutron	110
5.36	Paquete ICMP echo request capturado en la interfaz qr del router de Neutron	110
6.1	Cantidad de usuarios por despliegue. Extraída de [51].	114
6.2	Cantidad de instancias por despliegue. Extraída de [51].	115
A.1	Distribución del tiempo por tarea	127
B.1	Nueva conexión en virt-manager.	131
B.2	Configuración de redes virtuales en virt-manager.	132
C.1	Acceso remoto al servidor renata.	139
C.2	Túnel reverso y esquema de servidores.	140
C.3	Salida a Internet en los nodos de Openstack.	141
D.1	Vista del login de Horizon.	178
D.2	Creación de un proyecto (1/2).	179
D.3	Creación de un proyecto (2/2).	180
D.4	Creación de un usuario.	181
D.5	Creación de un flavor (1/2).	182
D.6	Creación de un flavor (2/2).	182
D.7	Creación de una red provider (1/2).	183
D.8	Creación de una red provider (2/2).	183
D.9	Creación de una imagen (1/2).	184
D.10	Creación de una imagen (2/2).	185
D.11	Creación de una red (1/3).	185
D.12	Creación de una red (2/3).	186
D.13	Creación de una red (3/3).	186
D.14	Creación de un router.	187
D.15	Creación de una interfaz en un router.	187
D.16	Creación de una key pair.	187
D.17	Lanzar una nueva instancia (1/5).	188
D.18	Lanzar una nueva instancia (2/5).	189
D.19	Lanzar una nueva instancia (3/5).	190
D.20	Lanzar una nueva instancia (4/5).	190

D.21 Lanzar una nueva instancia (5/5).	191
D.22 Asignación de floating IP.	192
D.23 Asociación de floating IP.	193
D.24 Reglas security group por defecto.	193
D.25 Agregar regla para tráfico ICMP.	194
D.26 Agregar regla para tráfico SSH.	194
F.1 Arquitectura diseñada para desarrollo	208
F.2 Arquitectura diseñada para producción	209

Tabla de contenidos

Lista de figuras	VI
1 Introducción	1
2 Estado del arte	3
2.1 Fundamentos teóricos	3
2.1.1 Cloud computing	3
2.1.2 Virtualización	4
2.1.3 Contenedores	5
2.1.4 Datacenters	7
2.1.5 Redes	7
2.1.6 Interfaces y bridges	8
2.1.7 Backends de almacenamiento	9
2.2 OpenStack	12
2.2.1 Origen y definición	12
2.2.2 Módulos Core	12
2.2.3 Tipos de nodos	14
2.2.4 Servicios de infraestructura	15
2.2.5 Métodos de instalación	16
2.3 Neutron	18
2.3.1 Tipos de redes en OpenStack	19
2.3.2 Plugin ml2	20
2.3.3 Funcionalidades avanzadas	27
2.4 OpenStack-Ansible	28
2.4.1 Arquitectura	29
2.4.2 Configuración OSA	31
2.4.3 Proceso de instalación	33
3 Diseño	36
3.1 Distribución del tráfico	36

3.2	Arquitectura desarrollo	37
3.3	Arquitectura producción	38
3.4	Distribución de los servicios	39
4	Gestión del Datacenter	42
4.1	Recuperación ante fallas	42
4.1.1	Verificación general	42
4.1.2	Tareas de mantenimiento	42
4.1.3	Solución de problemas	43
4.1.4	Problemas con Ceph	43
4.2	Escalamiento horizontal	43
4.2.1	Agregar nodo de Cómputo	43
4.2.2	Eliminar un nodo de cómputo	43
4.2.3	Agregar nodo de Infraestructura	44
4.3	Actualización de versión	44
5	Pruebas de concepto de Neutron	46
5.1	Escenarios de prueba	46
5.1.1	Escenario 1: tráfico este-oeste (misma red tenant)	47
5.1.2	Escenario 2: tráfico este-oeste (distintas redes tenant)	47
5.1.3	Escenario 3: tráfico norte-sur (acceso hacia el exterior)	48
5.1.4	Escenario 4: tráfico norte-sur (acceso desde el exterior)	50
5.2	Linux Bridge	52
5.2.1	Ventajas y desventajas	77
5.3	Open vSwitch	79
5.3.1	Ventajas y desventajas	111
6	Conclusiones	113
6.1	Trabajo a futuro	115
	Referencias	117
	Anexos	123
	Anexo A Bitácora del proyecto	124
	A.1 Inconvenientes	127
	Anexo B Virtualización con KVM.	131
	B.1 Utilización virt-manager	131
	B.1.1 Conexión remota	131
	B.1.2 Creación de una red	132
	B.1.3 Crear nodo	134

Anexo C	Instalación	138
C.1	Ambiente de trabajo	138
C.1.1	Hardware utilizado	138
C.1.2	Conexión remota hacia el servidor renata	139
C.1.3	Especificaciones servidor renata	140
C.1.4	Acceso al exterior desde nodos	141
C.2	Instalación versión Queens	141
C.2.1	Preparación de nodos	142
C.2.2	Configuración	150
C.3	Instalación versión Stein	156
C.3.1	Preparación de nodos	156
C.3.2	Configuración archivos OSA	168
C.3.3	Cambios para driver OVS	174
C.4	Verificación	176
Anexo D	Interacción	178
D.1	Configuraciones de administrador	179
D.2	Interacción de un usuario	183
D.3	Acceso a una instancia	191
D.3.1	Por SPICE	191
D.3.2	Por SSH	192
D.3.3	Por virsh	195
Anexo E	Gestión	196
E.1	Recuperación ante fallas	196
E.1.1	Tareas de mantenimiento	196
E.1.2	Problemas con Ceph	197
E.2	Escalamiento horizontal	197
E.2.1	Agregar nodo de Cómputo	197
E.2.2	Eliminar un nodo de cómputo	199
E.2.3	Agregar nodo de Infraestructura	201
E.3	Actualización Queens a Rocky	204
Anexo F	Imágenes	208

Capítulo 1

Introducción

En la actualidad hay un fuerte potencial de crecimiento en los Datacenters, principalmente dado por la demanda de sus servicios provenientes de nuevas tendencias como la digitalización de la información, el crecimiento del comercio electrónico y la adopción del cloud computing como una alternativa a los servidores locales. En función de esto resulta sumamente interesante investigar procedimientos que deben ser llevados a cabo para desplegar y operar un Datacenter. En la actualidad existen múltiples plataformas que permiten alcanzar estos objetivos como por ejemplo OpenStack, CloudStack y OpenShift.

Una tendencia que se encuentra en constante aumento en los últimos años es la utilización de OpenStack como plataforma de despliegue y gestión de código abierto. Se trata de una de las comunidades de código abierto con mayor crecimiento en el mundo, formada por más de 100.000 miembros, distribuidos por 185 países, con el soporte de aproximadamente 700 compañías. Año tras año se realizan dos conferencias en donde se presentan nuevas versiones, funcionalidades y experiencias de una amplia comunidad, manteniendo el apoyo de grandes organizaciones como Ubuntu, T-Mobile, Huawei, Intel, RedHat y Cisco.

Con esto en mente, este es un proyecto de ingeniería enfocado en la gestión de la configuración de un Datacenter, con el requerimiento de utilizar OpenStack como plataforma de despliegue. Se realiza un especial hincapié en entender cómo se da el enrutamiento dentro de un Datacenter. Cuando se mantiene un ambiente de gran escala, configurar las comunicaciones entre servidores en forma manual (p. ej. comunicaciones cableadas entre switches físicos) resulta impracticable y es una limitante para escalar horizontalmente. El módulo de red Neutron utilizado por OpenStack simplifica estas tareas permitiendo que un usuario final configure de manera sencilla la comunicación entre sus instancias y redes sin que la intervención de un administrador sea indispensable.

Concretamente, los objetivos del estudio son:

- Llevar a cabo una primera experiencia de despliegue de la plataforma OpenStack en un ambiente de prueba en el Instituto de Computación de la Facultad de Ingeniería.
- Dado que la administración de un Datacenter operativo suele ser un proceso tedioso y de vital importancia, otro de los objetivos será estudiar la gestión en operación de la plataforma. Esto último va desde una guía básica de utilización del dashboard hasta procesos de actualización de la herramienta y modificaciones de la infraestructura sin detener su ejecución.
- Una vez alcanzado un ambiente estable y adquirir mayor experiencia en la plataforma, es de particular interés analizar cómo se resuelve la comunicación a nivel de red entre los planos virtuales y físicos del Datacenter. En función de esto se estudiarán alternativas que brinda Neutron para resolver la interrogante.

En el anexo [A](#) se detallan cronológicamente las tareas realizadas durante todo el proyecto, indicando los objetivos e inconvenientes ocurridos en cada etapa.

Capítulo 2

Estado del arte

2.1. Fundamentos teóricos

A continuación se presentan conceptos introductorios relevantes para comprender el estudio que se desarrollará más adelante en el informe. Estas definiciones intentan poner sobre la mesa terminología típica del mundo de Datacenters, principalmente orientadas a OpenStack, tales como estilos de virtualización, elementos de redes y backends de almacenamiento.

2.1.1. Cloud computing

Cloud computing resulta ser un modelo que involucra tanto a los servicios computacionales provistos a los clientes a través de Internet, como a la implementación de hardware y software que logra proveerlos. Esta última se aloja en los denominados Datacenters, que integran adecuadamente diversos recursos tales como redes, servidores, almacenamiento y software necesarios para ofrecer los mencionados servicios en función de la demanda. Este modelo computacional, impulsado durante el siglo 21, ha evolucionado velozmente y ha migrado el mundo de TI de las antiguas PCs locales de usuarios y de los cuartos de servidores empresariales a la llamada “nube” alojada en lejanos Datacenters.

Existen tres modelos de cloud computing que se distinguen según el tipo de servicio provisto [69]:

- **Software as a Service (SaaS):** refiere al servicio que provee aplicaciones a través de Internet que se encuentran corriendo en la infraestructura de la nube (Datacenter). Los consumidores del servicio no manipulan la infraestructura subyacente ni las configuraciones de aplicación.
- **Platform as a Service (PaaS):** consiste en proveer recursos a nivel de

plataforma, como por ejemplo soporte de sistemas operativos, que permiten al cliente desplegar sus propias aplicaciones. El consumidor no puede manipular la infraestructura pero sí es capaz de administrar lo que se despliega sobre ella.

- **Infrastructure as a Service (IaaS):** se asocia al suministro de recursos de infraestructura a demanda. Se suele proveer al cliente de capacidad de procesamiento, almacenamiento y conectividad de red. Típicamente se ofrecen máquinas virtuales con la posibilidad de que el cliente manipule los sistemas operativos y las aplicaciones. Nuevamente el consumidor no es capaz de influir en la infraestructura que implementa la nube. Esta última clasificación es en la que se encuentra el despliegue que es realizado mediante OpenStack.

2.1.2. Virtualización

Uno de los principales conceptos que se encuentra involucrado en la implementación de cloud computing es el de virtualización [17][66]. Esta tecnología permite simular diversos ambientes con recursos dedicados a partir de un solo sistema físico. Es posible crear aplicaciones, servidores, almacenamiento y redes, utilizando al máximo todos los recursos del sistema subyacente aumentando el rendimiento general. Los usuarios finales interactúan directamente con las virtualizaciones, llamadas máquinas virtuales. Una máquina virtual puede ser transferida de un sistema host a otro y funcionar de igual forma.

La implementación de esta tecnología se da mediante los denominados **hipervisores** [17][62], los cuales se encargan de conectar los recursos físicos de la máquina host con las máquinas virtuales. Estos trabajan sobre el sistema operativo o directamente en el hardware, creando una plataforma virtual que divide los recursos físicos, sobre la cual se ejecutan las diferentes virtualizaciones. Por otro lado también son responsables de crear ambientes aislados que brinden seguridad entre las máquinas virtuales.

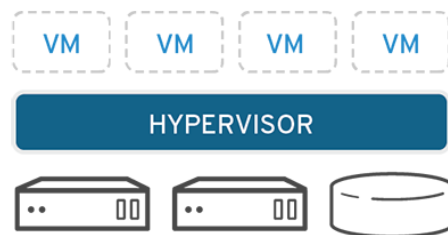


Figura 2.1: Hipervisores. Extraída de [17].

Los hipervisores se suelen clasificar en dos tipos:

- **Nativos o bare metal:** se trata de software que corre directamente sobre el hardware del sistema host, controlando el hardware y monitoreando el sistema operativo invitado. Algunos ejemplos son Oracle VM, Microsoft Hyper-V, VMware ESXi y Xen.
- **Alojados o hosted:** consisten en hipervisores que corren dentro de un sistema operativo tradicional. Se encuentran en una capa de aplicación por encima del sistema operativo del host pero por debajo del sistema operativo guest. Algunos ejemplos son Oracle VM VirtualBox, VMware Server y Workstation, Microsoft Virtual PC, KVM, QEMU y Parallels. Las máquinas virtuales creadas mediante la plataforma OpenStack, son instanciadas por un hipervisor de esta categoría.

KVM Kernel-based Virtual Machine (KVM) [16] se trata de una tecnología open source de virtualización sobre Linux. Se encuentra formada por un módulo del kernel denominado KVM.ko y permite transformar un sistema operativo Linux en un hipervisor. Como módulo del kernel se encuentra incluido en Linux a partir de la versión 2.6.20. Como hipervisor se clasifica como de tipo alojado, o hosted, utilizando los componentes del sistema Linux para implementar cada máquina virtual como un proceso de Linux. Este es el hipervisor utilizado por OpenStack mencionado anteriormente.

2.1.3. Contenedores

Un contenedor [13] es una unidad de software liviana diseñada para ejecutar una aplicación. Está formado únicamente por el código de la aplicación y las dependencias necesarias para que esta ejecute, creando un paquete portable e independiente. De esta forma múltiples contenedores pueden correr como procesos diferentes en una misma máquina host compartiendo el kernel del sistema operativo. Si bien tanto contenedores como máquinas virtuales tienen como fin aislar y compartir recursos de la máquina host, lo hacen en diferentes niveles. Los primeros virtualizan únicamente el sistema operativo guest, utilizando el kernel del host subyacente, mientras que las VMs virtualizan a nivel de hardware.

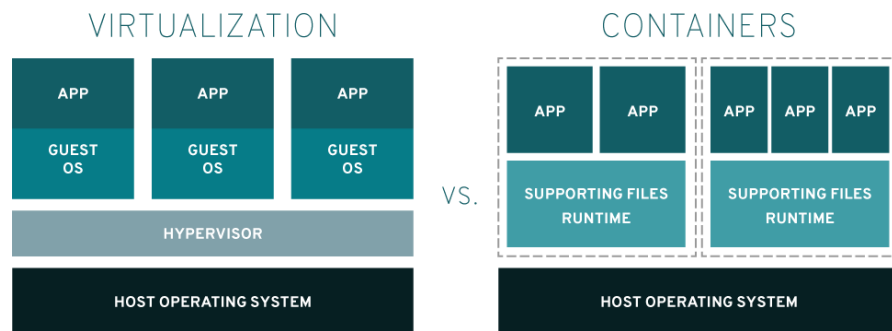


Figura 2.2: Virtualización vs Contenedores. Extraída de [18].

La combinación de estas tecnologías provee gran flexibilidad al realizar el despliegue de aplicaciones sobre varios servidores, complementando lo ligero de los contenedores con el aislamiento de recursos físicos y la seguridad obtenidos mediante VMs.

Citando la ‘Application Container Security Guide’ (SP 800-190 de NIST [64]): “A pesar de que a veces se considera que los contenedores son la siguiente fase de virtualización, sobrepasando la virtualización de hardware, la realidad de la mayoría de las organizaciones apunta menos a la revolución que a la evolución. Los contenedores y la virtualización de hardware no solo pueden, sino que frecuentemente lo hacen, coexistir y heredar las capacidades del otro. Las VMs brindan muchos beneficios, tales como fuerte aislamiento, automatización de SO, y un amplio y profundo ecosistema de soluciones. Las organizaciones no necesitan tomar una decisión entre contenedores y máquinas virtuales. Por el contrario pueden continuar utilizando VMs para desplegar, particionar y administrar su hardware, mientras que utilizan contenedores para empaquetar sus aplicaciones, aprovechando cada VM más eficientemente.” Es por esto que ambas tecnologías serán utilizadas más adelante en la puesta en marcha de un Datacenter de prueba mediante OpenStack.

LXC Un Linux Container (LXC) [3][18] es un contenedor formado por un conjunto de procesos que se encuentran aislados del resto del sistema host. Como tal, brinda un entorno virtual con su propia CPU, memoria, red, etc, implementado mediante el uso de los namespaces y cgroups del kernel Linux corriendo en la máquina host. Este tipo de contenedores es utilizado por OpenStack durante su despliegue para ejecutar los servicios en cuya configuración se haya indicado que utilicen contenedores en lugar de correr directo sobre el servidor.

2.1.4. Datacenters

La infraestructura que se encuentra por debajo de la mencionada nube se conoce con el nombre de Datacenter. Un Datacenter [9] es un espacio físico que aloja múltiples componentes de hardware interconectados tales como servidores, racks, switches, routers, sistemas de almacenamiento, etc. Estos proveen un conjunto de recursos de red, cómputo y almacenamiento, necesarios para alojar diversas aplicaciones o grandes cantidades de datos. A su vez, los nuevos Datacenters escalan implementando infraestructuras virtualizadas, utilizando los mecanismos mencionados anteriormente, por encima de la física ya existente llegando a interconectar múltiples espacios físicos ubicados en diversas partes del mundo. Debido al gran potencial computacional existente en este tipo de infraestructuras, su mayor explotación se encuentra en la ejecución de tecnologías tales como big data, inteligencia artificial, aprendizaje automático, entre otras. Por su gran importancia en la tecnología de la información actual, los Datacenters no solo requieren un diseño de infraestructura de recursos computacionales sino también un diseño de componentes físicos externos que garanticen la seguridad física y una tasa de resistencia a fallas prácticamente perfecta.

2.1.5. Redes

En la etapa de configuración e instalación y posteriormente en la utilización de OpenStack, se hace referencia a diversos conceptos de red, por ejemplo, protocolos, tipos de redes y componentes virtualizados. A continuación se introducirán brevemente algunos de estos conceptos.

Flat Una red flat hace referencia a una red en la cual no se utiliza ningún tipo de tag. Las interfaces físicas o virtuales se asocian directamente al switch o bridge por lo tanto solamente una red flat puede existir por cada interfaz física.

VLAN Una Virtual Local Area Network (VLAN) permite segmentar de manera virtual un dominio de difusión de capa 2 en múltiples dominios de difusión. Esto permite utilizar un switch como si fueran múltiples switches. A modo de ejemplo, dos hosts conectados a un mismo switch pero en distintas VLANs no podrán ver el tráfico generado por el otro. Para identificar a qué VLAN corresponde una trama se utiliza un nuevo campo en el cual se introduce el ID de la VLAN, estos cambios que se realizan a la trama Ethernet se establecen en el estándar IEEE 802.1Q [1].

VXLAN El protocolo Virtual eXtensible Local Area Network (VXLAN) se ubica dentro de los protocolos de superposición (overlay protocols) que utilizan el mecanismo de tunelización para el transporte de datos. VXLAN encapsula una trama Ethernet dentro de paquetes UDP los cuales pueden ser ruteados. Esto permite extender una red local sobre múltiples redes de capa 3 en forma transparente para los host finales. El funcionamiento del protocolo se encuentra en el RFC 7348 [21]. Para diferenciar las distintas redes virtuales en lugar de utilizar un VLAN ID se utiliza un VXLAN Network Identifier (VNI), el cual puede tomar aproximadamente 16 millones de valores siendo una de las principales diferencias con las VLANs que pueden tomar 4096 valores únicos. Esta diferencia para Datacenters de gran porte es vital para poder aislar el tráfico de los clientes del mismo. Un componente necesario para encapsular y desencapsular son los VXLAN Tunnel Endpoints (VTEPs) los cuales residen en los nodos físicos.

A la hora de decidir el tipo de red a utilizar para los clientes de un Datacenter es importante evaluar las pros y contras de cada una de ellas. En [47] se presenta un listado en el que se comparan los beneficios y perjuicios de utilizar redes de capa 2 y capa 3.

Network namespaces El concepto de namespace es utilizado para brindar aislamiento entre dos conjuntos de recursos. Precisamente en Linux los namespaces de red aíslan diversos recursos de red permitiendo mantener disjuntos conjuntos de interfaces e incluso tablas de ruteo. Esto último es utilizado por el módulo de red de OpenStack a la hora de instanciar servicios de red virtuales tales como routing y DHCP. De esta forma es posible que coexistan diferentes redes virtuales con el mismo direccionamiento IP en un mismo servidor de red sin solaparse [48].

2.1.6. Interfaces y bridges

Desde el diseño de la arquitectura física hasta la implementación de instancias virtuales por parte de OpenStack, se utilizan una serie de interfaces y bridges de diferentes tipos para proveer conectividad de red. Estos tipos son detallados brevemente a continuación.

Interfaces tap Estas interfaces son creadas y utilizadas por un hipervisor como QEMU/KVM para conectar una instancia de una VM con el host subyacente. Las interfaces en el host se corresponden a una interfaz de red dentro de la instancia.

Linux bridge Es una interfaz virtual que conecta múltiples interfaces de red. Se comporta como un switch virtual al cual se asocian uno o más segmentos de red. Particularmente, en OpenStack se utilizan este tipo de bridges para brindar conexión a los servicios que se encuentran corriendo en contenedores. También son aplicados por el módulo de red a la hora de interconectar las instancias con las redes virtuales a nivel del host físico [4].

Veth cables Son interfaces virtuales que imitan el comportamiento de un cable de red. Una trama enviada en un extremo del veth cable será recibida por el otro. El módulo de red utiliza estos dispositivos cuando realiza conexiones entre network namespaces y Linux bridges, así como cuando conecta Linux bridges con OpenvSwitch switches.

Interfaces físicas Una interfaz física representa una interfaz alojada en un host, asociada a una infraestructura de red física.

Interfaces VLAN En los sistemas Linux el soporte para el manejo de tags 802.1q de VLAN se realiza a través del uso de interfaces VLAN virtuales. El modulo del kernel utilizado es 8021q.

Interfaces VXLAN Es una interfaz virtual utilizada para encapsular y reenviar tráfico basándose en parámetros configurados durante la creación de la interfaz, entre los que se incluyen el VNI (VXLAN Network Identifier) y el VTEP (VXLAN Tunnel End Point).

2.1.7. Backends de almacenamiento

2.1.7.1. LVM

Logical Volume Manager (LVM) es una herramienta para la administración de volúmenes lógicos en sistemas operativos Linux. A nivel del sistema es capaz de asignar discos, realizar copias y ajustar el tamaño de los mencionados volúmenes [15]. A nivel de OpenStack es relevante debido a que se trata del backend de almacenamiento utilizado por defecto por el módulo Cinder.

2.1.7.2. Ceph

Ceph se trata de un sistema de almacenamiento de datos distribuido y libre. Unifica el almacenamiento de objetos, bloques y archivos en un sistema altamente confiable y escalable basado en el servicio RADOS [68]. Este utiliza el algoritmo

CRUSH [67] para procesar la ubicación de los datos en forma eficiente dentro del cluster que puede alcanzar grandes cantidades de nodos. Cada uno de los nodos del cluster ejecuta un demonio que es capaz de comunicarse por si solo con los demás con el fin de replicar y redistribuir la información en forma dinámica. Estos demonios se clasifican en dos tipos:

- **Monitors:** mantienen la copia maestra del mapa del cluster. Los clientes obtienen una copia del estado del cluster desde un nodo monitor. A su vez suelen haber múltiples nodos de este tipo en el cluster con el fin de garantizar la alta disponibilidad.
- **OSDs:** su abreviatura viene de Object Storage Device, este tipo de demonios corre en los nodos de almacenamiento y son capaces de comunicarse entre sí sin tener que depender de una unidad central.

Ceph modela la información que recibe del cliente como objetos y la almacena en los OSDs como un archivo dentro de un namespace sin jerarquía de directorios. Cada uno de estos objetos también mantiene un identificador y posible metadata interpretada por el cliente. Durante el proceso de escritura, el cliente primero debe obtener una copia actualizada del cluster desde un nodo monitor para luego escribir directamente sobre un OSD primario. Este último se encarga de crear réplicas en otros nodos para garantizar la seguridad y alta disponibilidad de los datos. Los monitores se encargan de controlar el estado de la información para brindar las garantías mencionadas [6].

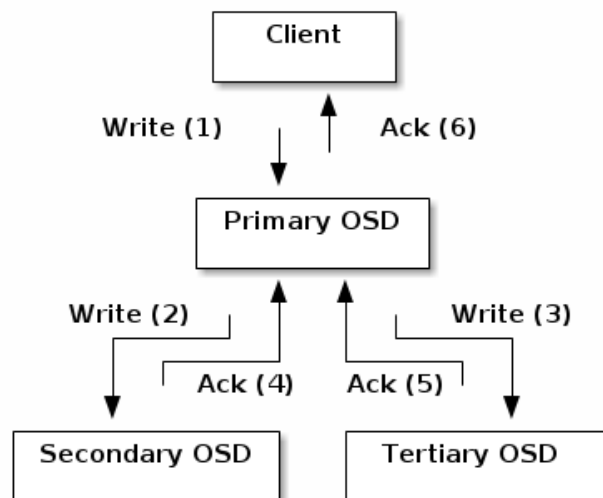


Figura 2.3: Proceso de replicación en OSDs de Ceph. Extraída de [6].

El sistema maneja particiones lógicas para el almacenamiento denominadas 'Pools'. Ceph determina dónde almacenar la información en función del tamaño del pool, la regla del algoritmo CRUSH y la cantidad de PGs (placement groups).

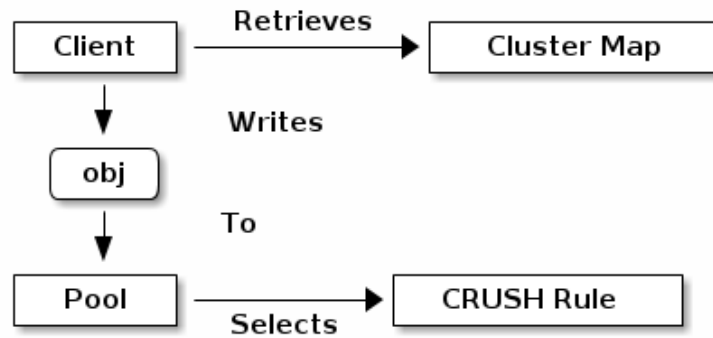


Figura 2.4: Proceso de almacenamiento en pools de Ceph. Extraída de [6].

Dentro de cada pool existen una cantidad de PGs, sobre los que el algoritmo de CRUSH determinará en cuál posiciona cada objeto. A su vez cada PG es asignado en forma dinámica con uno o varios OSDs. Esta estructura permite generar una abstracción al cliente evitando que deba conocer en qué OSD se encuentra su objeto. Aunque el cluster se rebalancee debido a la existencia de un nuevo nodo o a la falla de otro, el cliente podrá acceder siempre a su información si tiene una copia del mapa del cluster y el algoritmo de CRUSH.

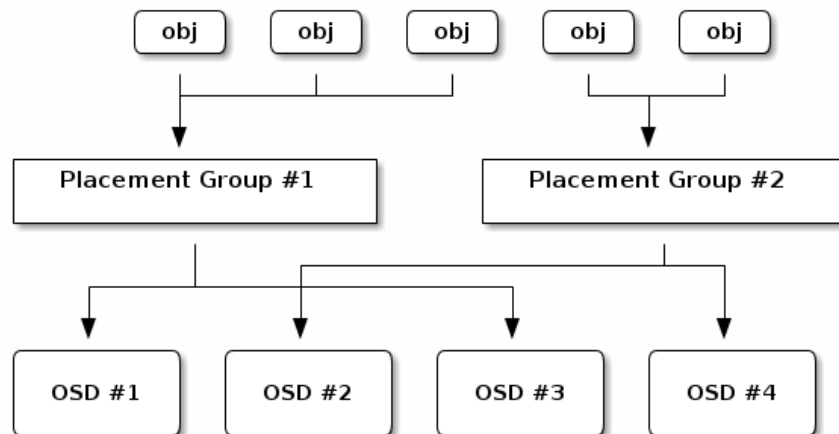


Figura 2.5: Proceso de almacenamiento en PGs de Ceph. Extraída de [6].

2.2. OpenStack

En esta sección se presenta la plataforma OpenStack utilizada para desplegar y gestionar Datacenters. Se comienza con una breve definición e introducción histórica para luego detallar cada uno de los módulos que componen el core de la plataforma junto a una clasificación de los nodos de la infraestructura. Sobre el final se mencionan métodos de instalación existentes, poniendo especial énfasis en la herramienta Ansible utilizada durante el proyecto.

2.2.1. Origen y definición

OpenStack fue creado en los primeros meses del 2010, momento durante el cual la empresa Rackspace pretendía reescribir el código de infraestructura que corría en sus servidores cloud, considerando hacer open source el código existente. En ese entonces, Anso Labs, quienes trabajaban para la NASA, publicaron el código beta para Nova, un proyecto basado en Python descrito como “cloud computing fabric controller”. Dadas las semejanzas en las necesidades de ambas empresas, decidieron unir fuerzas dando como resultado una base de OpenStack.

En septiembre de 2012, fue lanzada la fundación de OpenStack como una institución independiente proporcionando recursos para proteger, potenciar y promover el software OpenStack y la comunidad que lo rodea [35]. Como es definido en el sitio de OpenStack, *“es un sistema operativo en la nube que controla grandes grupos de recursos de computación, almacenamiento y redes a través de un centro de datos, administrados y provistos a través de APIs”*[57].

OpenStack es un software open source gobernado por una fundación. Ser parte de la misma es gratis y está abierta a todo el mundo. El proyecto tiene una arquitectura modular, en donde cada instalación de OpenStack tendrá instalados y configurados los módulos que se ajusten a las necesidades del caso. Dichos módulos están implementados en Python. Seis de estos proyectos se denominan como módulos core dado que se encargan de las funciones principales del cloud como son las conexiones de red, el almacenamiento, el servicio de identidad, servicios de imágenes y de cómputo. Permite construir nubes públicas y privadas ofreciendo principalmente servicios de infraestructura (IaaS) y en un grado menor, servicios de plataforma (PaaS).

2.2.2. Módulos Core

En esta sección se describirán brevemente los módulos Nova, Neutron, Glance, Cinder, Keystone y Swift, llamados core. Los mismos se componen por un conjunto

de servicios internos cuyas funcionalidades se exponen mediante una o varios endpoints.

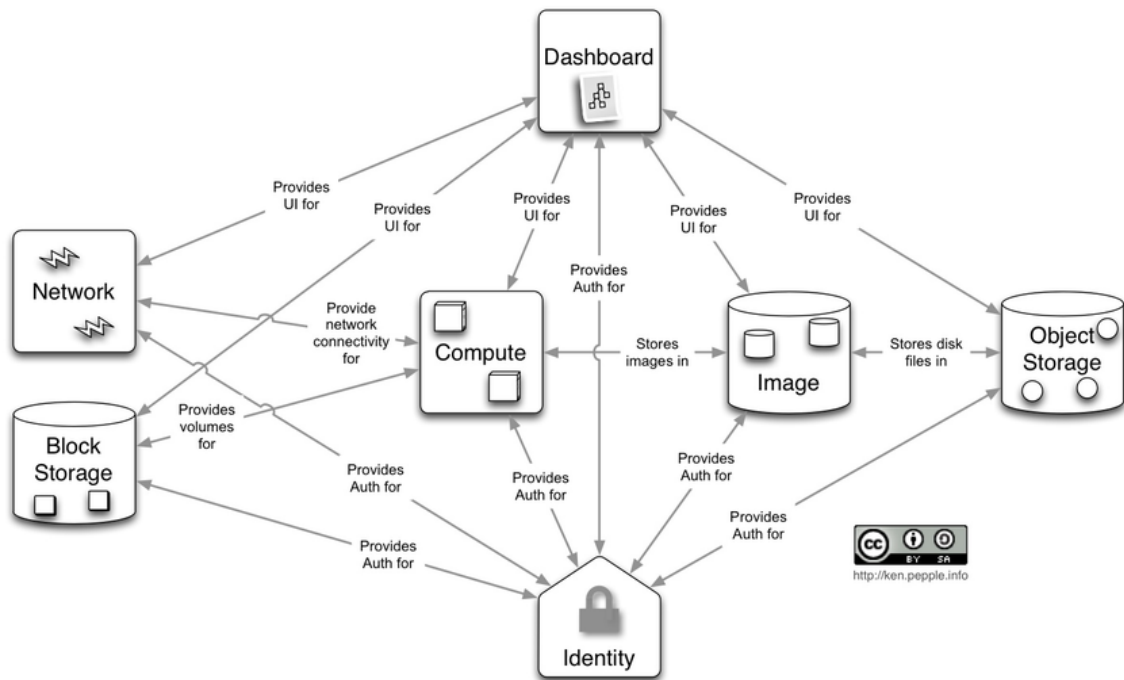


Figura 2.6: Relacionamiento entre módulos core. Extraída de [19]

2.2.2.1. Keystone

Keystone es el servicio de identidad que utiliza OpenStack para la autenticación y autorización [36][22]. Está compuesto por los servicios de identidad, recursos, asignaciones, tokens y catálogo.

2.2.2.2. Nova

Nova es el proyecto que se encarga de aprovisionar instancias o servidores virtuales. El proyecto soporta la creación de imágenes, servidores baremetal con la ayuda del proyecto Ironic y un soporte limitado para el manejo de containers. Adicionalmente para poder desarrollar las funciones básicas, este módulo requiere de los siguientes servicios core de OpenStack: Keystone, Glance y Neutron. Internamente los componentes de Nova se comunican mediante RPC mientras que la interfaz de cara al usuario es una API REST. Dentro de sus principales componentes se encuentran compute, scheduler, conductor y placement.

2.2.2.3. Neutron

Neutron es el proyecto encargado de proveer y administrar los recursos de red virtualizados en una nube creada con OpenStack. Es un sistema escalable

horizontalmente y diseñado para añadirle diversos plugins con el fin de proporcionar nuevas funcionalidades. Como otros servicios de OpenStack, Neutron requiere una base de datos para persistir las configuraciones de los elementos de red. Debido a que uno de los objetivos del proyecto es el estudio de este módulo, el mismo se profundiza en la sección [2.3](#).

2.2.2.4. Glance

Glance es el nombre del proyecto utilizado por OpenStack para la administración de imágenes. Este servicio permite a los usuarios gestionar tanto las imágenes de las máquinas virtuales como la metadata asociada a estas. Típicamente esta gestión involucra el descubrimiento, registro y obtención de los recursos mencionados. La implementación del servicio Glance se basa en una arquitectura cliente-servidor.

2.2.2.5. Cinder

El servicio de almacenamiento de bloques proporciona administración de almacenamiento de bloques persistente para discos duros virtuales. Las operaciones básicas que permite realizar son:

- Crear, listar y eliminar volúmenes.
- Crear, listar y eliminar snapshots.
- Asociar y desasociar volúmenes a máquinas virtuales.

Los volúmenes son utilizados como una solución para persistir los datos de una instancia incluso luego de destruir la misma, estos pueden ser asignados a una instancia a la vez.

2.2.2.6. Swift

Se trata del componente de almacenamiento de objetos de OpenStack, implementado mediante un sistema distribuido de alta disponibilidad. Gestiona el almacenamiento a largo plazo de grandes cantidades de información utilizando redundancia de datos en clusters bajo una arquitectura que evita tener un único punto de control, permitiendo escalar fácilmente.

2.2.3. Tipos de nodos

En general los sistemas de OpenStack están contruidos con servidores o nodos físicos, aunque también es posible utilizarlos virtualizados como es el caso de este trabajo, en donde se pueden agrupar en 4 categorías [\[11\]](#):

Nodo de control Estos nodos en general corren las APIs de todos los servicios de los componentes de OpenStack. Además estos nodos alojan la base de datos de los módulos, los servidores de mensajes y los componentes de caché en memoria como Memcached. Para escalar horizontalmente, las APIs pueden ser instaladas en varios nodos de control pudiendo adicionalmente balancear la carga.

Nodo de red Estos nodos en general corren los servicios de metadatos y DHCP. Además permiten crear routers virtuales cuando el agente de Neutron L3 es instalado. Al igual que los nodos de control, para mejorar el rendimiento y escalar horizontalmente se pueden separar los servicios de red entre los distintos nodos de red. El uso de nodos de red dedicados mejora la seguridad y resistencia, dado que los nodos de control tendrán menor riesgo de que se sature la red y sus recursos.

Nodo de cómputo Estos nodos en general corren un hipervisor como puede ser KVM. Hyper-V o Xen; o por el lado de los contenedores LXC o Docker.

Nodo de almacenamiento Estos nodos en general se limitan a correr software que esté relacionado con los proyectos como Cinder, Ceph o Swift. En estos nodos no es común alojar servicios de otro tipo como de red o cómputo.

Nodo de balanceo de carga Estos nodos son fundamentales para el funcionamiento óptimo de una instalación de OpenStack, dado que se espera que los servicios que se ofrecen tengan una alta disponibilidad. Como la forma de mejorar el rendimiento es escalar horizontalmente, resulta vital tener nodos que se encarguen de distribuir la carga entre los mismos.

2.2.4. Servicios de infraestructura

Estos servicios son transversales a todos los módulos del sistema y son obligatorios para el funcionamiento de OpenStack. En general estos servicios son instalados en los nodos de control como fue mencionado en la sección anterior.

Galera - MariaDB La base de datos MariaDB se utiliza para almacenar el estado de todos los servicios de OpenStack, utilizando usualmente una base de datos MySQL. El servidor de base de datos en general se despliega con una configuración activo/pasivo en donde solamente el servidor principal (activo) podrá ser utilizado por los servicios. Para poder utilizar un esquema activo/activo se puede utilizar Galera, el cual se define como un software de clusterización para MySQL, que utiliza un mecanismo sincrónico de replicación para lograr una alta disponibilidad [63].

Message queue OpenStack utiliza una cola de mensajes para llevar a cabo la comunicación entre procesos. Los servicios de cola de mensaje que OpenStack soporta son: RabbitMQ y Qpid. Ambos servicios son Advanced Message Queuing Protocol (AMQP) frameworks, los cuales proveen colas de mensajes para comunicaciones punto a punto. En general las colas de mensajes son desplegadas como pools de servidores centralizados o descentralizados. En la instalación realizada se utilizó RabbitMQ [44].

Memcached Es un sistema de caché de objetos en memoria distribuido, apuntado a mejorar el rendimiento de los sistemas mediante la reducción de carga a la base de datos. En OpenStack este software es utilizado por el mecanismo de autenticación de Keystone para cachear los tokens del sistema [43].

2.2.5. Métodos de instalación

Realizar la instalación básica de OpenStack (módulos core) es una tarea sumamente compleja. Esto se debe a la gran cantidad de configuraciones y diversos tópicos en los cuales hay que tener un grado de entendimiento no menor, como en bases de datos, sistemas operativos Linux, redes y backends de almacenamiento. Las guías de instalación que se pueden encontrar en el sitio oficial de OpenStack consisten de cientos de configuraciones y comandos a ejecutar en donde es muy probable equivocarse y en consecuencia instalar incorrectamente los módulos de la herramienta.

Debido a la relevancia que ha tomado en los últimos años OpenStack, la amplia comunidad formada por decenas de compañías y personas buscaron caminos alternativos a realizar la instalación manualmente. Estas formas se basan en la automatización de las tareas, para las cuales existen varias tecnologías como Puppet [61] y Ansible [49]. Además existen diversas distribuciones de OpenStack, como DevStack que permite armar un ambiente rápidamente para realizar pruebas, Packstack-RDO o TripleO. Finalmente existe una gran oferta de distribuciones comerciales donde podemos encontrar grandes compañías como IBM, Debian, DELL, Red Hat, VMware, Huawei, etc. Un listado más extenso se puede ver en [50].

Otra de las razones principales para utilizar herramientas de automatización además de facilitar el despliegue es para poder mantener el sistema luego de su instalación. En el caso de realizar las tareas manualmente no se podrá escalar la nube a cientos o miles de servidores dado que sería prácticamente imposible de mantener o actualizar.

En el presente trabajo se emplea la herramienta de automatización Ansible

dentro del proyecto OpenStack-Ansible (OSA) descrito en la sección 2.4. Antes de alcanzar dicha sección, se introduce brevemente el concepto de Ansible.

2.2.5.1. Ansible

Ansible es una herramienta para la automatización de tareas. Permite configurar sistemas, aplicaciones o dispositivos, desplegar o mantener software, entre otras tareas de IT. Algunos puntos a destacar son: su diseño simple orientado a que sea fácil de utilizar, el uso de OpenSSH como transporte y el diseño del lenguaje, que no requiere de conocimientos de programación. Estas características junto a que el software no tiene muchas dependencias son lo que potencian a utilizar OSA. A continuación se describirán los principales conceptos, necesarios para la utilización de Ansible.

Nodo de control El nodo de control será quien ejecute comandos o playbooks de Ansible. Los requisitos serán tener Python instalado y Ansible. En la arquitectura utilizada, como se menciona luego en el informe, el nodo de control es el de deploy, el cual contiene los diversos scripts de Ansible y se comunica con el resto de los nodos para instalar y configurar los módulos y componentes de OpenStack. Este nodo puede ser uno de los utilizados como parte del Datacenter o de uso exclusivo para la instalación.

Inventario Mantiene una lista de los nodos administrados. El inventario es un archivo en el cual se puede especificar la IP de cada nodo administrado, se pueden organizar los nodos en grupos para una mejor escalabilidad. En la sección 2.4.2.2 se profundiza en este punto.

Módulos Todas las acciones que se pueden realizar con Ansible se llevan a cabo con un módulo. Estos son las unidades de código que se ejecutan. En cada tarea se puede ejecutar uno o más módulos. Una lista completa de los módulos se encuentra en [2].

Tarea Es la unidad de acción en Ansible.

Playbook Contienen una lista ordenada de tareas. Las playbooks se escriben en YAML lo cual beneficia la lectura, escritura y la comprensión de las mismas. Son una forma de organizar las tareas bajo un criterio determinado. En el caso de OSA utiliza varias playbooks para preparar el ambiente inicial, instalar los componentes de infraestructura, entre otros que se especificarán en las próximas secciones.

Conclusión

Luego de haber investigado la composición de OpenStack, se observa que se trata de una plataforma de gran porte cuyo funcionamiento y configuración no parece ser sencillo. Sin embargo resulta relevante utilizar alguna de las herramientas de instalación que fueron mencionadas con el fin de tener un primer acercamiento a un despliegue completo. Por otro lado, se destaca como eficaz el hecho de que esté compuesto por módulos descentralizados, brindando la posibilidad de investigar y desarrollar cada uno de ellos en forma aislada. Otro aspecto sumamente positivo es que en su diseño se cuente con un conjunto de módulos core básicos lo que permite realizar un despliegue con lo mínimo indispensable. En caso de buscar funcionalidades más avanzadas, es posible agregar módulos adicionales más específicos.

2.3. Neutron

Como se mencionó anteriormente, Neutron está encargado de administrar los recursos de redes virtuales dentro de OpenStack. Permite crear desde redes y subredes hasta topologías de red avanzadas las cuales pueden contener firewalls, balanceadores, VPNs entre otros.

Las instalaciones del módulo de red deben tener al menos una red externa, la cual no es una SDN definida dentro de OpenStack sino que es una red accesible por fuera del mismo. Estas redes permiten que las redes virtuales construidas con Neutron tengan conectividad con el mundo exterior. Por otro lado, Neutron permite crear redes internas a las cuales se conectarán directamente las VMs. Para lograr que estas últimas se conecten con las redes externas son necesarios routers que relacionen ambos tipos de redes.

Una arquitectura simplificada de Neutron se puede ver en la figura 2.7. En esta se muestra cómo el servidor de Neutron se conecta con la base de datos en la que se persisten los recursos de red creados. Por otro lado el servidor acepta solicitudes de usuarios y servicios a través de la API que expone.

neutron-server El servicio neutron-server es en donde reside la API recibiendo y respondiendo las solicitudes de recursos de red. El servidor se comunica con la base de datos para almacenar las configuraciones existentes como se mencionó previamente.

Plugins y agentes Por otro lado se encuentran una serie de agentes que se distribuyen típicamente en los nodos de red y cómputo. Estos agentes se agregan a

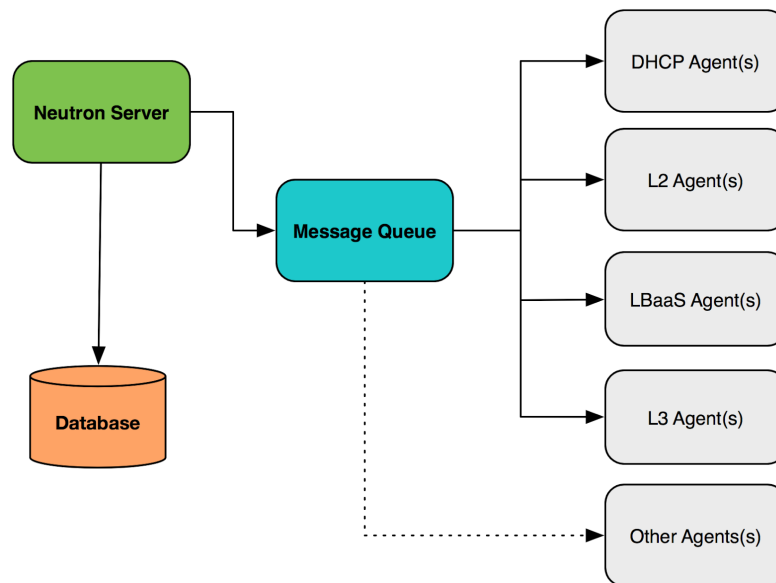


Figura 2.7: Arquitectura simplificada de Neutron. Extraída de [10]

demanda según las funcionalidades y rendimiento requeridos, siendo los encargados de crear los recursos virtuales de red del cloud.

Cola de mensajes En general se utiliza en las instalaciones del módulo de red para llevar a cabo la comunicación entre el neutron-server y los distintos agentes.

2.3.1. Tipos de redes en OpenStack

En OpenStack los usuarios tienen la posibilidad de crear su propio esquema de red, decidiendo libremente el direccionamiento IP a utilizar. Los recursos de red creados dentro de un proyecto son privados al mismo. En OpenStack se pueden crear dos tipos de redes:

- **Project/tenant network:** es una red virtual creada por un proyecto o por un administrador en nombre del mismo. Este tipo de red se encarga de proveer conectividad a los recursos de un proyecto. Los usuarios pueden crear, modificar y eliminar redes tenants. En general este tipo de redes está aislado del resto de las redes de proyecto por VLANs o algún otro mecanismo de segmentación como VXLAN.
- **Provider network:** es una red virtual creada para asociarse a una red física de los servidores que alojan OpenStack. Este tipo de redes son creadas para habilitar el acceso a recursos de red externos a la nube de OpenStack. Las mismas son creadas y administradas por los administradores ya que deben tener conocimiento de la infraestructura subyacente.

Cuando se crea una red de proyecto, los aspectos físicos de cómo es implementada son transparentes al usuario, por lo contrario en la red de proveedor se puede especificar el tipo de red, la interfaz física y el identificador de segmentación utilizado.

2.3.2. Plugin ml2

ml2 es el plugin central de Neutron introducido en OpenStack en la versión Havana. El plugin se creó para reemplazar los modelos de plugins monolíticos, permitiendo con el diseño modular la operación de varias tecnologías de forma concurrente. Los principales plugins monolíticos, OVS y LB, quedaron obsoletos desde la liberación mencionada y reimplementados como mechanism driver de ml2.

El principal motivo para realizar este nuevo diseño se debe a que en los despliegues de OpenStack se debía decidir en tiempo de implementación qué plugin se iba a utilizar sin tener la opción en un futuro de cambiarlo o utilizar otro en paralelo. Esto era una gran limitante para las organizaciones con diversos requerimientos con la necesidad de utilizar distintas tecnologías.

Los drivers dentro de ml2 implementan por un lado un conjunto de tipos de redes y por otro mecanismos para acceder a redes de esos tipos definidos.

Type drivers Definen cómo una red de OpenStack es realizada técnicamente, por ejemplo las redes VXLAN. Cada tipo de red disponible es administrado por un type driver de ml2.

Mechanism drivers Definen el mecanismo para acceder a una red de OpenStack de un determinado tipo, por ejemplo Open vSwitch mechanism driver. Son responsables de tomar la información establecida por un Type driver y aplicarla correctamente de acuerdo a los mecanismos de red que estén disponibles.

2.3.2.1. Linux Bridge

El mechanism driver Linux Bridge tiene soporte para los siguientes tipos de drivers: local, flat, VLAN y VXLAN. El agente de Neutron utiliza los módulos del kernel bridge, 8021q y vxlan para llevar a cabo la conectividad entre las instancias y los distintos recursos de red. Algunas de las principales características a destacar son su robustez, la madurez de la tecnología y la facilidad para diagnosticar y solucionar problemas.

Para llevar a cabo la implementación de las funciones de red este driver utiliza las siguientes cinco tipos de interfaces: taps, físicas, VLANs, VXLANs y Linux bridges, los cuales fueron descritos brevemente en la sección 2.1.6. Por cada red

creada en OpenStack el driver Linux Bridge crea un bridge en los nodos de cómputo que alberguen instancias en dicha red y en los nodos de red. Estos bridges se conectan a los distintos tipos de interfaces mencionadas de acuerdo al tipo de red que implementen.

Archivos de configuración

Se detallan brevemente las principales configuraciones de los archivos ubicados en el directorio `/etc/neutron/plugins/ml2`.

ml2_conf.ini

```
[ml2]
type_drivers = flat,vlan,vxlan,local
tenant_network_types = vxlan,vlan,flat
mechanism_drivers = linuxbridge
extension_drivers = port_security
# ML2 flat networks

[ml2_type_flat]
flat_networks = flat
# ML2 VLAN networks

[ml2_type_vlan]
network_vlan_ranges = vlan:150:200,vlan:300:400
# ML2 VXLAN networks

[ml2_type_vxlan]
vxlan_group = 239.1.1.1
vni_ranges = 1:1000

[ml2_type_geneve]
vni_ranges =
max_header_size = 38
# Security groups

[securitygroup]
enable_security_group = True
enable_ipset = True
```

- **[ml2]**: esta sección detalla los tipos de redes soportados (vxlan, vlan y flat) y los tipos de redes soportados para las redes tenants (vxlan y vlan), en donde

el orden indica sobre qué tipo de red tendrá preferencia Neutron al crear una nueva red tenant. Luego se indica el driver utilizado.

- **[ml2_type_flat]**: en esta sección se encuentran las configuraciones correspondientes a las redes flat, en donde se indica la etiqueta asociada a la interfaz de la infraestructura subyacente para dar soporte a este tipo de red.
- **[ml2_type_vlan]**: en esta sección se encuentran las configuraciones correspondientes a las redes VLAN, en donde se indican las redes físicas que pueden ser utilizadas para redes provider y tenant, así como los rangos de VLAN ID disponibles.
- **[ml2_type_vxlan]**: en esta sección se encuentran las configuraciones correspondientes a las redes VXLAN, en donde se indica la dirección del grupo de multicast y el rango de VNI para ser utilizados por las redes tenant.
- **[securitygroup]**: en esta sección se indica si los grupos de seguridad están habilitados para su uso en OpenStack.

linuxbridge_agent.ini

```
[linux_bridge]
physical_interface_mappings = flat:eth12,vlan:br-vlan
# Linux bridge agent VXLAN networks

[vxlan]
enable_vxlan = True
vxlan_group = 239.1.1.1
# VXLAN local tunnel endpoint
local_ip = 10.0.31.11
l2_population = False
ttl = 32
# Agent

[agent]
# Security groups

[securitygroup]
firewall_driver = iptables
enable_security_group = True
```

En este archivo se encuentran las configuraciones correspondientes al agente Linux Bridge.

- **[linux_bridge]**: la configuración `physical_interface_mappings` describe la correspondencia entre una etiqueta artificial como es “vlan” con un switch virtual creado con LB como es `br-vlan`. En OpenStack al crear una red se asocia con alguna de las etiquetas indicadas en esta opción.
- **[vxlan]**: en esta sección se indica si las redes vxlan estarán disponibles, se determina el `vxlan_group` utilizado para realizar el multicast, la dirección IP del nodo que se utilizará para construir la red de overlay entre los distintos hosts y si se habilita o no el `l2_population`. En la arquitectura presentada, el tráfico de las redes de overlay llega a la infraestructura física mediante una subinterfaz de la `eth2` asociada al bridge `br-vxlan` con la subred `10.0.31.0/24`.
- **[securitygroup]**: se indica nuevamente si los grupos de seguridad se habilitan y el driver utilizado para implementar el firewall.

2.3.2.2. Open vSwitch

El mechanism driver Open vSwitch tiene soporte para los siguientes tipos de drivers: local, flat, VLAN, VXLAN y GRE. Dentro de Neutron, OVS opera como un switch implementado por software el cual utiliza bridges virtuales y reglas de flujos (flow rules) para hacer el forwarding entre distintos host. A continuación se describen brevemente los componentes de OVS:

- **Kernel module**: el módulo `OpenvSwitch` es el equivalente a un ASIC en un switch por hardware. Dicho módulo se encarga de procesar todos los paquetes.
- **vSwitch daemon**: el demonio `ovs-vswitchd` es un proceso de Linux que corre en el espacio del usuario en cada nodo físico y se encarga de indicar como estará programado el kernel module.
- **Database server**: es una base de datos local utilizada en cada nodo físico llamada Open vSwitch Database Server (OVSDB), la cual mantiene la configuración de los switches virtuales.

En la figura 2.8 se muestra un diagrama en alto nivel de estos componentes.

El agente que utiliza OVS en Neutron se llama `neutron-openvswitch-agent`. Es un servicio configurado en los hosts utilizando el mechanism driver Open vSwitch y es responsable de manejar la implementación de redes y sus interfaces relacionadas. El agente conecta interfaces tap con Open vSwitch switches o Linux Bridges y programa flujos utilizando herramientas de OVS como `ovs-vsctl` y `ovs-ofctl` basándose en datos que provee el servicio `neutron-server`. En una implementación basada en OVS existen 5 tipos de dispositivos virtuales de red:

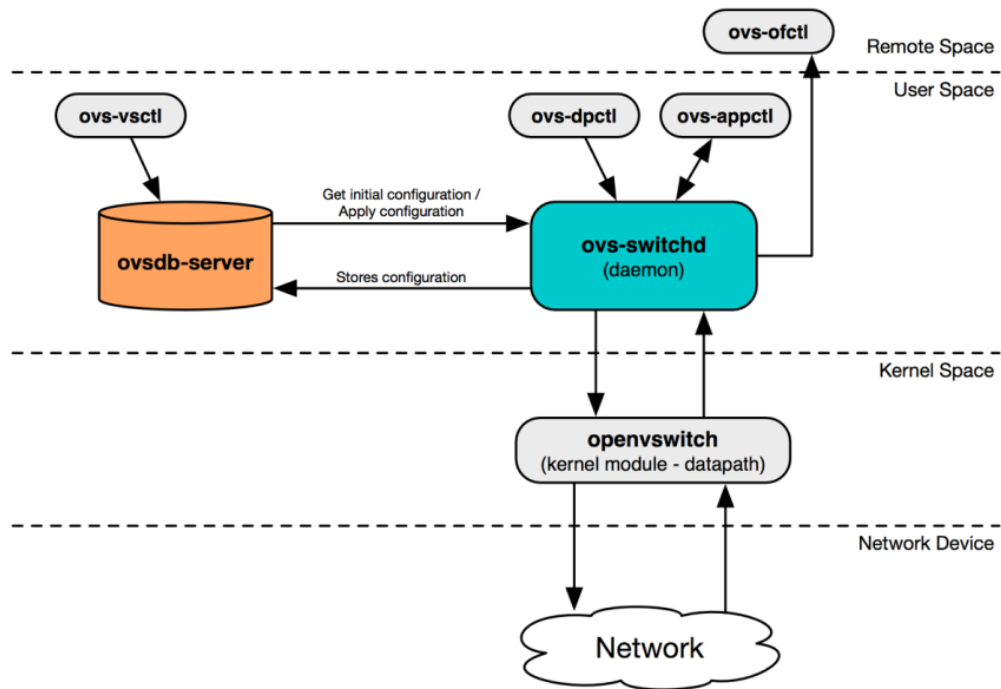


Figura 2.8: Diagrama de componentes de Open vSwitch

- Tap interfaces
- Linux bridges
- Virtual ethernet cables (veth)
- OVS bridges
- OVS patch ports

Open vSwitch tiene un tipo de puerto propio que imita el comportamiento de los veth cables pero optimizado para el uso con OVS bridges. Cuando se conectan dos OVS bridges, un puerto es reservado en cada switch como un patch port. Los patch ports son configurados con un peer name que corresponde al patch port en el otro switch.

El driver OVS utiliza una serie de bridges para implementar las distintas funcionalidades que ofrece Neutron. A continuación se mencionan los mismos junto a una breve descripción del rol que cumplen:

- **Integration bridge:** el bridge de integración, comúnmente llamado br-int es el switch virtual central al cual se conectan la mayor cantidad de dispositivos, como son las instancias, servidores de DHCP y routers, entre otros. Dependiendo del driver del firewall utilizado las instancias se conectan

directamente al bridge en cuestión o utilizan un dispositivo adicional (un Linux bridge) para llegar al br-int.

- **Provider bridge:** el nombre de este bridge dependerá de la configuración del ambiente, en la instalación utilizada se llama br-provider. Se encarga de proveer conectividad con la red física del Datacenter mediante una interfaz física del nodo. Además se conecta con el bridge de integración mediante patch ports.
- **Tunnel bridge:** se trata de un provider bridge particular que se utiliza para conectar los endpoints de las redes de overlay. El nombre por defecto para este bridge es br-tun.

Cada nodo de control, cómputo y red del ambiente de OpenStack tendrá un bridge de integración, proveedor y túnel. A diferencia de Linux Bridge, el driver OVS no crea nuevos dispositivos virtuales para implementar las redes tenant o provider. En este caso las redes creadas utilizan como infraestructura base los bridges mencionados y reglas de OpenFlow, mediante las cuales se indica, por ejemplo, la etiqueta que se tiene que agregar o quitar a un paquete en función de la interfaz que lo recibe.

Archivos de configuración

Al igual que en el estudio del driver Linux Bridge se detallan brevemente las principales configuraciones de los archivos ubicados en el directorio `/etc/neutron/plugins/ml2`.

`ml2_conf.ini`

```
[ml2]
type_drivers = vxlan,vlan,flat
tenant_network_types = vxlan,vlan
mechanism_drivers = openvswitch
extension_drivers = port_security
# ML2 flat networks

[ml2_type_flat]
flat_networks =
# ML2 VLAN networks

[ml2_type_vlan]
network_vlan_ranges = vlan:150:200,vlan:300:400
# ML2 VXLAN networks
```

```
[ml2_type_vxlan]
vxlan_group = 239.1.1.1
vni_ranges = 1:1000
```

```
[ml2_type_geneve]
vni_ranges =
max_header_size = 38
# Security groups
```

```
[securitygroup]
firewall_driver = iptables_hybrid
enable_security_group = True
enable_ipset = True
```

- **[ml2]**: análogo a LB.
- **[ml2_type_flat]**: en este caso esta sección se encuentra vacía debido a que en la instalación utilizada para OpenvSwitch no se soportan redes flat.
- **[ml2_type_vlan]**: análogo a LB.
- **[ml2_type_vxlan]**: análogo a LB.
- **[securitygroup]**: en esta sección se indica si los grupos de seguridad están habilitados para su uso en OpenStack y en caso positivo el driver para implementar el firewall en las instancias. En particular el driver iptables_hybrid implementa las reglas de firewall usando iptables lo que implica la existencia de un Linux bridge adicional como intermediario entre la interfaz tap de las instancias y el bridge de integración.

openvswitch_agent.ini

```
[ovs]
local_ip = 10.0.31.11
bridge_mappings = vlan:br-provider
```

```
[agent]
l2_population = False
tunnel_types = vxlan
enable_distributed_routing = False
extensions =
# Security groups
```

```
[securitygroup]
firewall_driver = iptables_hybrid
enable_security_group = True
enable_ipset = True
```

En este archivo se encuentran las configuraciones correspondientes al agente Open vSwitch.

- **[ovs]**: en esta sección se indica en primer lugar la dirección IP del nodo que se utilizará para construir la red de overlay entre los distintos hosts. En la arquitectura presentada, el tráfico de las redes de overlay llega a la infraestructura física mediante una subinterfaz de la eth2 asociada al bridge br-vxlan con la subred 10.0.31.0/24. La configuración bridge_mappings es análoga a la physical_interface_mappings de LB.
- **[agent]**: en esta sección se indica el tipo de las redes de overlay y si se habilita o no l2_population.
- **[securitygroup]**: es idéntica a la mencionada en el archivo ml2.conf.ini.

2.3.3. Funcionalidades avanzadas

2.3.3.1. Layer 3 High Availability

Una instalación de Neutron sin ningún mecanismo de alta disponibilidad será vulnerable a posibles fallos que surjan en los nodos físicos del Datacenter. Si bien los ambientes productivos de OpenStack contienen varios nodos físicos dedicados a las funciones de red, esto no garantiza la alta disponibilidad de conexión en las instancias. Esto se da debido a que al crear un nuevo router virtual, este es planificado para correr en un único agente L3 físico, generando que su funcionamiento dependa exclusivamente del mismo. Si este único nodo presenta algún problema, provocará que las instancias pierdan conectividad con redes externas y que las IPs flotantes no se encuentren disponibles.

Para solucionar este problema se utiliza una configuración de alta disponibilidad basada en routers con dos estados: pasivo y activo, empleando el estándar VRRP (Virtual Routing Redundancy Protocol) definido en el RFC 5798 [23]. En una explicación a alto nivel, esta implementación planifica la creación de un router virtual en múltiples nodos de red de OpenStack, en donde únicamente se designa uno de estos como activo mientras que el resto queda en un estado de espera o pasivo. Un ejemplo de esto se puede ver en las figuras 2.9 y 2.10, en donde en la primera se aprecia como los routers 1 y 2 tienen su par de backup en el otro nodo físico. Por otro lado en la segunda figura se observa como el L3 HA replanifica el/los dispositivos

afectados por una falla, en este caso el Router1, a un nodo operativo en donde existe una copia en estado de espera del mismo.

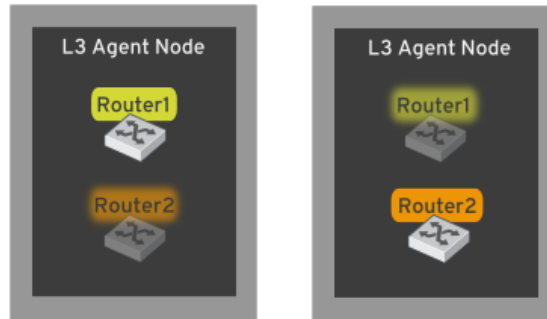


Figura 2.9: Routers virtuales previo a una falla. Extraída de [65].

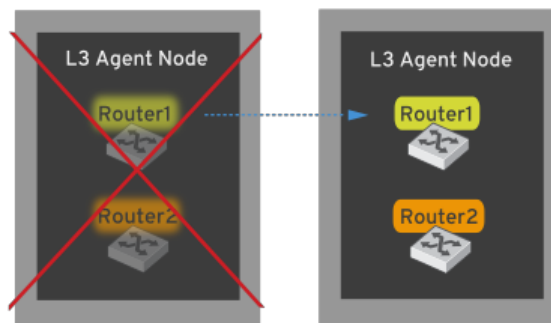


Figura 2.10: Routers virtuales luego de una falla. Extraída de [65].

2.4. OpenStack-Ansible

OpenStack-Ansible se trata de un proyecto basado en la herramienta de automatización Ansible para realizar el despliegue de la plataforma OpenStack. Dicho proyecto se encarga de proveer playbooks y roles para desplegar y configurar un ambiente basado en el concepto de Infrastructure as Code (IaC). OSA no es un proyecto que funcione simplemente con los archivos y configuraciones por defecto sino que modificaciones por parte del administrador del cloud serán necesarias. El resultado final que se obtiene con OSA es un cloud de OpenStack probado para ambientes de cualquier tamaño, desde Datacenters de testing hasta producción. Como fue mencionado en la sección 2.2.5, la utilización de una herramienta de esta índole no podía cuestionarse. Sin embargo, era relevante determinar cuál de todas las

existentes utilizar. Para tomar la decisión se consultó con Edgar Magana ¹, allegado a los tutores, quién argumentó que OSA era la opción indicada para comenzar. En función de esto, en este capítulo se analiza la arquitectura, configuraciones y funcionamiento de OSA.

2.4.1. Arquitectura

El método de instalación OSA emplea LXC para desplegar los servicios de OpenStack. Además utiliza Linux bridges entre los contenedores y las interfaces físicas o lógicas del host con el fin de proveer conectividad directa a nivel de capa 2. El aislamiento de cada contenedor se logra mediante la utilización de namespaces, esto genera que se deban utilizar pares de interfaces virtuales (veth pairs) para tener conectividad entre los mismos. Esta implementación se puede ver en la imagen 2.11.

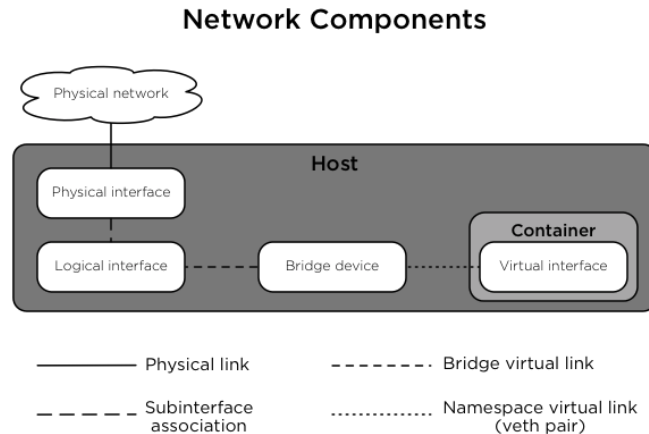


Figura 2.11: Componentes de red en OpenStack. [29].

2.4.1.1. Arquitectura de red

OSA soporta múltiples arquitecturas de red, las cuales se diferencian según sea para un ambiente de producción o testing, cantidad de interfaces físicas de los servidores o módulos de OpenStack utilizados. Para los ambientes de producción es común utilizar interfaces emparejadas mejorando la disponibilidad de los servicios.

¹Edgard es un arquitecto de operaciones en la nube con un amplio conocimiento de la plataforma OpenStack. Tiene un PhD en sistemas distribuidos y se especializa en Cloud Computing, Network Virtualization, Software-defined Networking (SDN), Network Functions Virtualization (NFV), OpenFlow y OpenStack. Actualmente es un miembro central en la comunidad de desarrollo de Neutron y durante los últimos tres años ha dirigido tanto el desarrollo de OpenStack como equipos de despliegue. Ha alcanzado despliegues exitosos de la plataforma sobre la infraestructura de grandes compañías, generando una vasta experiencia en despliegues automatizados de infraestructuras de Cloud Computing mediante herramientas como puppet y chef.

En general para segmentar el tráfico, tanto en los casos donde se realiza bonding o en donde hay interfaces simples, se utilizan VLANs asignando un ID para cada subred utilizada dentro de OpenStack. A continuación se describen las subredes empleadas por OSA para su funcionamiento:

Management Network La red de administración o también container network se encarga de proveer la administración y comunicación entre la infraestructura y los servicios de OpenStack en containers o en servidores físicos. Para que todas las instancias tengan acceso a esta red, es necesario que todos los nodos host del Datacenter cuenten con el bridge br-mgmt. A este último se le asocian las interfaces virtuales de cada contenedor y la lógica o física del host en cuestión, asignadas a dicha red. Esta interfaz suele ser la primaria del nodo mediante la cual se accede por SSH.

Overlay Network La red de superposición o también tunnel network, provee conectividad entre los hosts virtualizados dentro de OpenStack encapsulando el tráfico con algún protocolo de tunelización como VXLAN. La VLAN o interfaz utilizada para esta subred se asocia al bridge br-vxlan. Este bridge debe ser instanciado en todos los nodos que manejen agentes del módulo Neutron, típicamente involucra los nodos de cómputo y/o red.

Storage Network La red de almacenamiento provee acceso entre los backends de almacenamiento, tales como Block Storage, y los servicios de OpenStack, como Cinder o Glance. En este caso las interfaces o VLANs se asocian al bridge br-storage, que debe ser instanciado en todos los nodos o contenedores que alojan servicios de cómputo o almacenamiento.

En la figura [2.12](#) se presenta un diseño que ilustra servicios desplegados en contenedores y cómo varían las conexiones de red para proveer conectividad.

Como se menciona anteriormente los contenedores que corren servicios de OpenStack requieren de interfaces virtuales para conectarse con los bridges del nodo físico. Además se puede observar cómo varían las conexiones necesarias a los distintos tipos de red, en función del tipo de agente que corre en el contenedor. Al utilizar contenedores, OSA crea automáticamente una nueva red que se utilizará para la administración de los mismos (por ejemplo para descargar paquetes). Un punto a tener en cuenta es que Ansible para crear esta subred utiliza el rango 10.0.3.0/24.

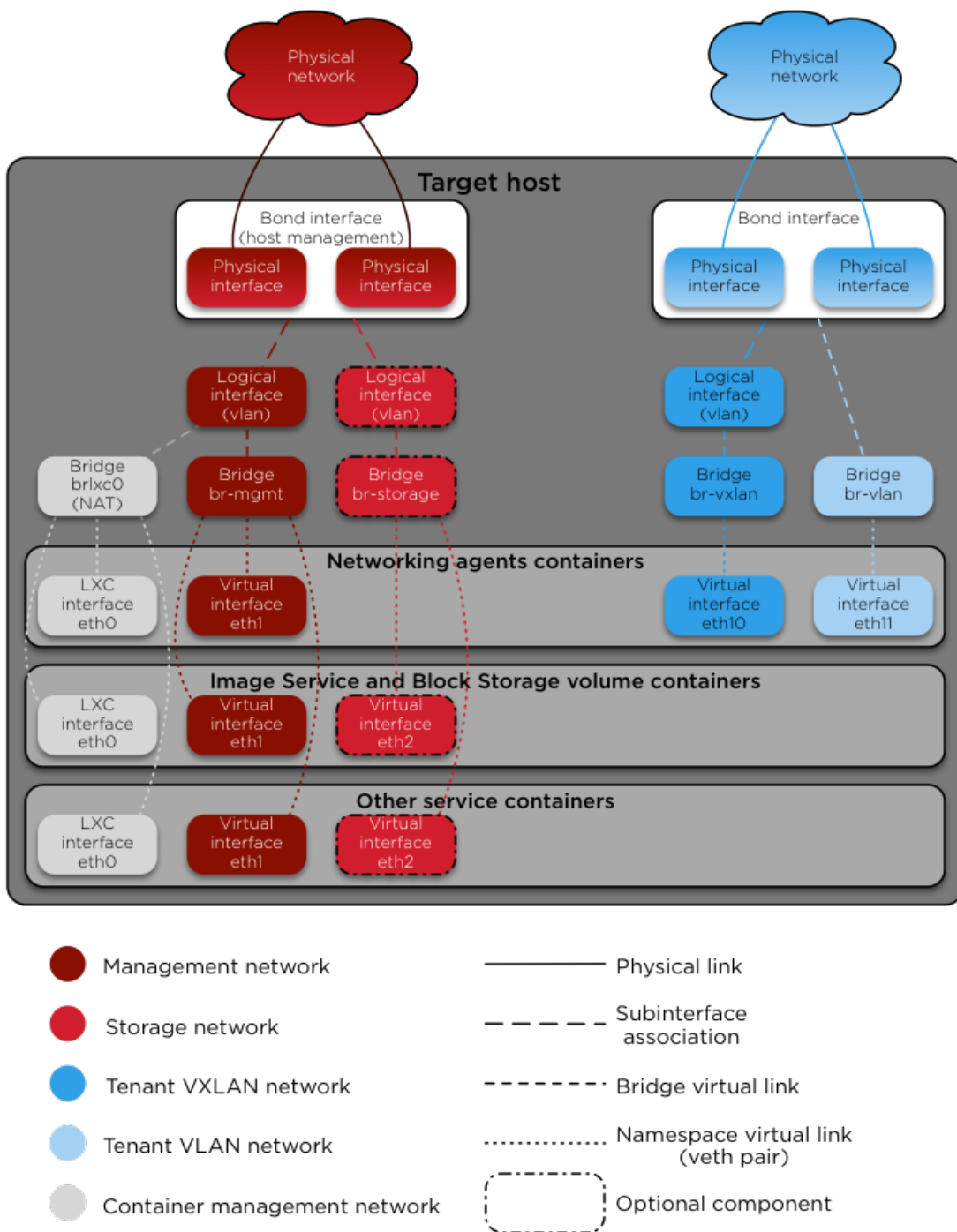


Figura 2.12: Despliegue de servicios OpenStack en contenedores. Extraída de [25].

2.4.2. Configuración OSA

En esta sección se presentan las configuraciones y conceptos más relevantes del nodo Deploy a tener en cuenta durante un despliegue de OpenStack utilizando Ansible. Estas incluyen las convenciones de directorios empleadas, la configuración estándar y el significado del contenido de los archivos que deben ser modificados.

2.4.2.1. Convenciones

- El repositorio de OSA se clona generalmente en el directorio `/opt/openstack-ansible`.
- Los roles de Ansible utilizados por defecto se encuentran en el directorio `/etc/ansible/roles` los cuales son generados a partir del archivo `/opt/openstack-ansible/ansible-role-requirements.yml` mediante la ejecución del `scriptbootstrap-ansible.sh`.
- Las configuraciones realizadas por el administrador son indicadas en el directorio `/etc/openstack_deploy`.

2.4.2.2. Inventario

Define las especificaciones de los hosts y contenedores dentro del ambiente actual de OpenStack. Esta información se encuentra en el archivo `/etc/openstack-ansible/openstack_inventory.json`, generado a partir de los host groups, containers groups y components indicados en:

- La estructura por defecto almacenada en `/opt/openstack-ansible/inventory/env.d`
- Lo configurado por el administrador dentro de `/etc/openstack_deploy/` en:
 - El archivo `openstack_user_config.yml`
 - El archivo `user_variables.yml`
 - El directorio `conf.d/`
 - El directorio `env.d/`

Estos archivos son considerados como referencia en cualquiera de los comandos asociados al despliegue de OSA por lo tanto nunca debe ser eliminado o modificado en un ambiente de producción.

Los components hacen referencia a los diferentes servicios que serán instalados durante el despliegue de OpenStack, tanto en contenedores virtuales como directamente en los target host. Los containers groups agrupan estos components, determinando los potenciales contenedores a ser creados junto con sus especificaciones. En las configuraciones realizadas en ambos directorios `env.d/` se asocian los containers groups anteriores con los hosts groups, los cuales agrupan diversos target hosts. De esta forma se determina qué servicio debe ser instalado en qué target host.

2.4.2.3. `openstack_user_config.yml`

Es el principal archivo de configuración creado por el operador de OpenStack. A continuación se realiza un breve resumen del rol que cumple cada sección del mismo. Como referencia se utilizó el siguiente documento [52].

- En la sección `cidr_networks`, se describen las subredes utilizadas en la instalación de OpenStack.
- En la sección `used_ips` se configuran las direcciones IP que están siendo utilizadas por los hosts físicos de la infraestructura en las redes definidas para el funcionamiento de OSA, con el fin de reservarlas y que la instalación no utilice ninguna de ellas para la estructura de OpenStack.
- En la sección `global_overrides` se especifican valores que deben ser ajustados de acuerdo a la arquitectura física establecida. Las principales opciones requeridas son: `internal_lb_vip_address` y `external_lb_vip_address` indicando las IPs de los balanceadores de carga tanto internos (utilizadas por los servicios de OpenStack) como externos (puerta de acceso para los usuarios), los nombres de los bridges utilizados para las redes de management, storage y tunnel, y una lista de las redes provistas por los nodos físicos subyacentes en la subsección `provider_networks`, describiendo adicionalmente como se conectan con los contenedores.
- La última sección describe en qué servidor o grupo de servidores corre cada servicio de OpenStack e infraestructura.

2.4.2.4. `user_variables.yml`

Los archivos llamados con el formato `user*.yml` ubicados en `/etc/openstack_deploy` son considerados automáticamente para cada comando de `openstack-ansible`. En particular las variables definidas en el archivo `user_variables.yml` dependen fuertemente del ambiente a desplegar.

2.4.3. Proceso de instalación

Para instalar OpenStack con Ansible es necesario correr las playbooks principales del proyecto, las cuales se encuentran en el directorio `/opt/openstack-ansible/playbooks`. En primer lugar se ejecutan tres scripts para realizar un chequeo de sintaxis de la configuración preparada y los scripts a utilizar. Esto se realiza de la siguiente forma:

```
# openstack-ansible setup-hosts.yml --syntax-check
# openstack-ansible setup-infrastructure.yml --syntax-check
# openstack-ansible setup-openstack.yml --syntax-check
```

En caso de no tener errores, se comienza la ejecución de las playbooks en el orden que se describen:

2.4.3.1. setup-hosts.yml

Esta playbook se encarga de configurar todos los hosts descritos en el archivo `openstack_user_config.yml`. Con el siguiente comando se ejecuta la playbook:

```
# openstack-ansible -vvv setup-hosts.yml 2>&1 | tee /var/log/openstack/
  hostsXX.log
```

La opción `-vvv` es para que la salida sea más detallada y el `final` es para mostrar la salida del comando en la consola y almacenarla en un archivo de log.

2.4.3.2. setup-infrastructure.yml

En este paso demora un poco más que el primer setup y se encargará de construir todos los contenedores donde luego se instalarán los servicios de OpenStack. Este script además se encarga de instalar los servicios de infraestructura como son RabbitMQ y Galera DB para luego ser configurados en la playbook final. Además se lleva a cabo la configuración de los balanceadores de carga implementados con HAProxy. Con el siguiente comando se ejecuta la playbook:

```
# openstack-ansible -vvv setup-infrastructure.yml 2>&1 | tee /var/log/
  openstack/infrastructureXX.log
```

2.4.3.3. setup-openstack.yml

En este paso final es cuando se configuran todos los servicios, indicados en los archivos de configuración de OSA, de OpenStack. Esta playbook es la que demora más tiempo en su ejecución, en el orden de las horas. Con el siguiente comando se ejecuta la playbook:

```
# openstack-ansible -vvv setup-openstack.yml 2>&1 | tee /var/log/openstack
  /openstackXX.log
```

Conclusiones

El proyecto OpenStack-Ansible es una herramienta fundamental para instalar una plataforma de gran porte como OpenStack. La alta complejidad para obtener

un ambiente funcional surge de la gran cantidad de configuraciones y tecnologías necesarias, en donde a su vez estas últimas son complejas por si mismas. Las herramientas de automatización como Ansible, permiten describir la arquitectura que se pretende desplegar mediante un archivo de configuración y abstraerse de los detalles para llevarlo a cabo. Otro de los factores positivos para utilizar OSA es que permite tener consistencia en la repetición de las instalaciones. Además cabe destacar la celeridad de OpenStack y OpenStack-Ansible para resolver los problemas que surgen y agregar nuevas funcionalidades en las liberaciones principales realizadas cada 6 meses y en los parches menores anunciados durante dicho período.

Capítulo 3

Diseño

En el siguiente capítulo se describen las diversas decisiones tomadas y las estructuras y diseños utilizados para realizar la instalación de OpenStack-Ansible. Debido a la flexibilidad que presenta OpenStack al momento de desplegar sus componentes entre nodos y contenedores, es de suma importancia evaluar cuáles serán los requerimientos, analizando de qué forma se utilizará y la magnitud del Datacenter.

Los diseños que se presentan en este capítulo están orientados a ser utilizados con servidores y componentes de red físicos. Debido a que no se cuenta con los suficientes recursos, todo el ambiente físico será virtualizado dentro de un único servidor. De aquí en más cada uno de los nodos virtualizados del Datacenter será considerado como un nodo físico, además los switches físicos necesarios serán emulados mediante Linux Bridges. Por último en el caso en que es necesario un router TOR, el mismo se implementa mediante una VM que mantiene configurados los forwarding necesarios.

3.1. Distribución del tráfico

El tráfico en OpenStack se puede dividir en dos categorías: plano de control y plano de datos. El plano de control está relacionado con el tráfico utilizado para administración de los nodos físicos o contenedores, para las API de los servicios de OpenStack y todo el tráfico que no esté relacionado con las instancias virtuales de OpenStack. El plano de datos está relacionado con el tráfico generado o dirigido hacia instancias virtuales.

En ambientes de producción es recomendable utilizar interfaces físicas diferentes para los distintos tipos de tráfico [12]. La decisión de colapsar todo el tráfico en una sola interfaz y segmentarlo con VLANs o utilizar múltiples interfaces depende de las necesidades de cada caso. Al utilizar una sola interfaz física, aumenta el riesgo de fallas de red.

3.2. Arquitectura desarrollo

El diagrama de la figura 3.1 (para mayor definición, ver en F.1) presenta la arquitectura diseñada para la instalación de OpenStack Ansible para un ambiente de desarrollo. Al tratarse de un escenario meramente de prueba, sólo se cuenta con un nodo de cada tipo definido previamente 2.2.3. En el mismo se puede apreciar que no se han utilizado VLANs ni bonds en las interfaces de los servidores, sino que se han agregado tantas interfaces físicas como fueran necesarias. Se detallan también las numeraciones IP a ser utilizadas en cada una de las redes necesarias y las asignaciones a los diferentes bridges de cada uno de los nodos. Cabe mencionar que la IP pública del balanceador de carga pertenece a la subred en la que se encuentra el servidor físico sobre el cual se virtualizan todos los nodos, con el fin de tener acceso externo a la plataforma. Por otro lado el backend de almacenamiento utilizado es LVM dado que no tiene relevancia utilizar una tecnología más robusta.

En este diseño se utilizan 4 redes necesarias para intercomunicar los planos de control y datos de los nodos del Datacenter y una red pública.

- Red de management en la subred 10.0.1.0/24.
- Red de storage en la subred 10.0.10.0/24.
- Red de VXLAN en la subred 10.0.2.0/24.
- Red de VLAN en la subred 10.0.4.0/24.
- Red pública en la subred 192.168.60.0/24.

En el diseño se utilizan 5 nodos, cuyas conexiones de red se detallan en la tabla 3.1.

Nodo	management	storage	VXLAN	VLAN	public
deploy	X				
haproxy1	X				X
infra1	X	X	X	X	
compute1	X	X	X	X	
storage1	X	X			

Tabla 3.1: Conexiones de red en ambiente de desarrollo

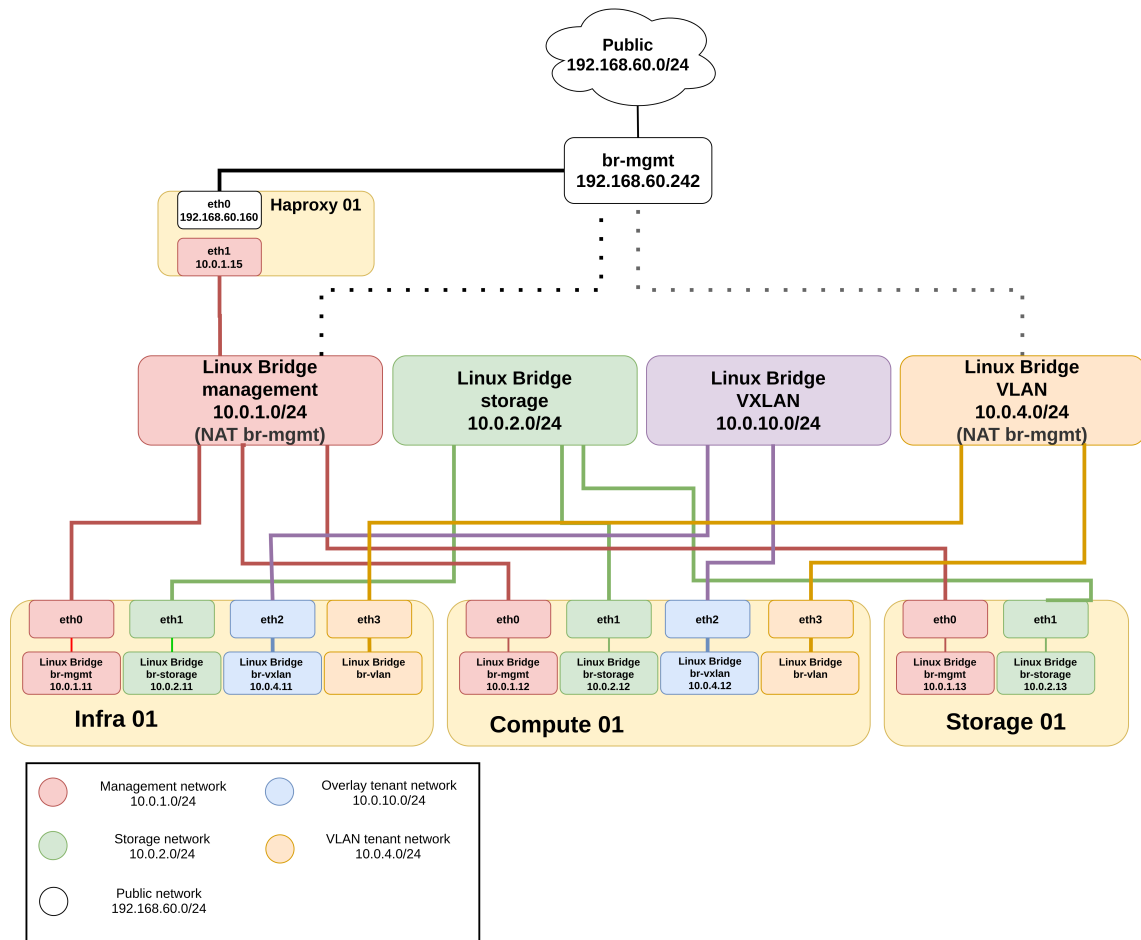


Figura 3.1: Arquitectura diseñada para instalación Queens

3.3. Arquitectura producción

La arquitectura planteada para instalar OpenStack en un ambiente de producción se muestra en la figura 3.2 (para mayor definición, ver en F.2). Con esta arquitectura se intenta aproximar una instalación para utilizar en un ambiente de producción, contando con varios nodos de almacenamiento y cómputo. A diferencia de la arquitectura empleada para desarrollo se utilizan VLANs para aislar el tráfico de datos de las redes tenant de tipo VLAN o VXLAN sobre la interfaz eth2 de los nodos de infraestructura y cómputo. Para el tráfico de management y storage se dedican interfaces exclusivas en cada uno de los nodos del Datacenter. El router agregado pretende simular un TOR configurado para realizar tareas como: reenviar el tráfico originado por las instancias en OpenStack hacia redes externas, administrar los distintos rangos de VLANs utilizables para redes provider o dar soporte a redes de tipo flat. Nuevamente para cada subred utilizada por OSA se crea un Linux Bridge que emula el comportamiento de un switch que conecta distintas interfaces de los nodos. Por último, cabe destacar que se modifica el backend utilizado para el

módulo de almacenamiento Cinder. En este caso se hace uso de Ceph, mencionado en el marco teórico, el cual presenta muchas ventajas frente al utilizado por defecto LVM. En este diseño se utilizan 6 redes necesarias para intercomunicar los planos de control y datos de los nodos del Datacenter y una red pública.

- Red de management en la subred 10.0.10.0/24.
- Red de storage en la subred 10.0.20.0/24.
- Red de tenant en la subred 10.0.30.0/24.
- Red de overlay tenant en la subred 10.0.31.0/24.
- Red de VLAN en la subred 10.0.100.0/24 con el VLAN ID 100.
- Red pública en la subred 192.168.60.0/24.

En el diseño se utilizan 8 nodos, cuyas conexiones de red se detallan en la tabla 3.2.

Nodo	management	storage	tenant	overlay	VLAN	public
deploy	X					
haproxy1	X					X
infra1	X	X	X	X	X	
compute1	X	X	X	X	X	
compute2	X	X	X	X	X	
storage1	X	X				
storage2	X	X				
router	X	X	X			X

Tabla 3.2: Conexiones de red en ambiente de desarrollo

En una arquitectura estándar de producción el router TOR físico podría actuar adicionalmente como un firewall y los nodos deberían tener 4 interfaces físicas en donde utilizando bonding y VLANs se ganaría mayor disponibilidad de los servicios.

3.4. Distribución de los servicios

Una de las principales características de OpenStack-Ansible es el despliegue de servicios en contenedores. Esto permite escalar horizontalmente los mismos de una forma sencilla evitando la complejidad de utilizar servidores físicos o máquinas virtuales y proporcionan una alta disponibilidad. Mas allá de la facilidad de utilizar contenedores, ciertos servicios continúan teniendo un mejor rendimiento cuando corren directamente sobre el host físico, como por ejemplo Nova, en donde

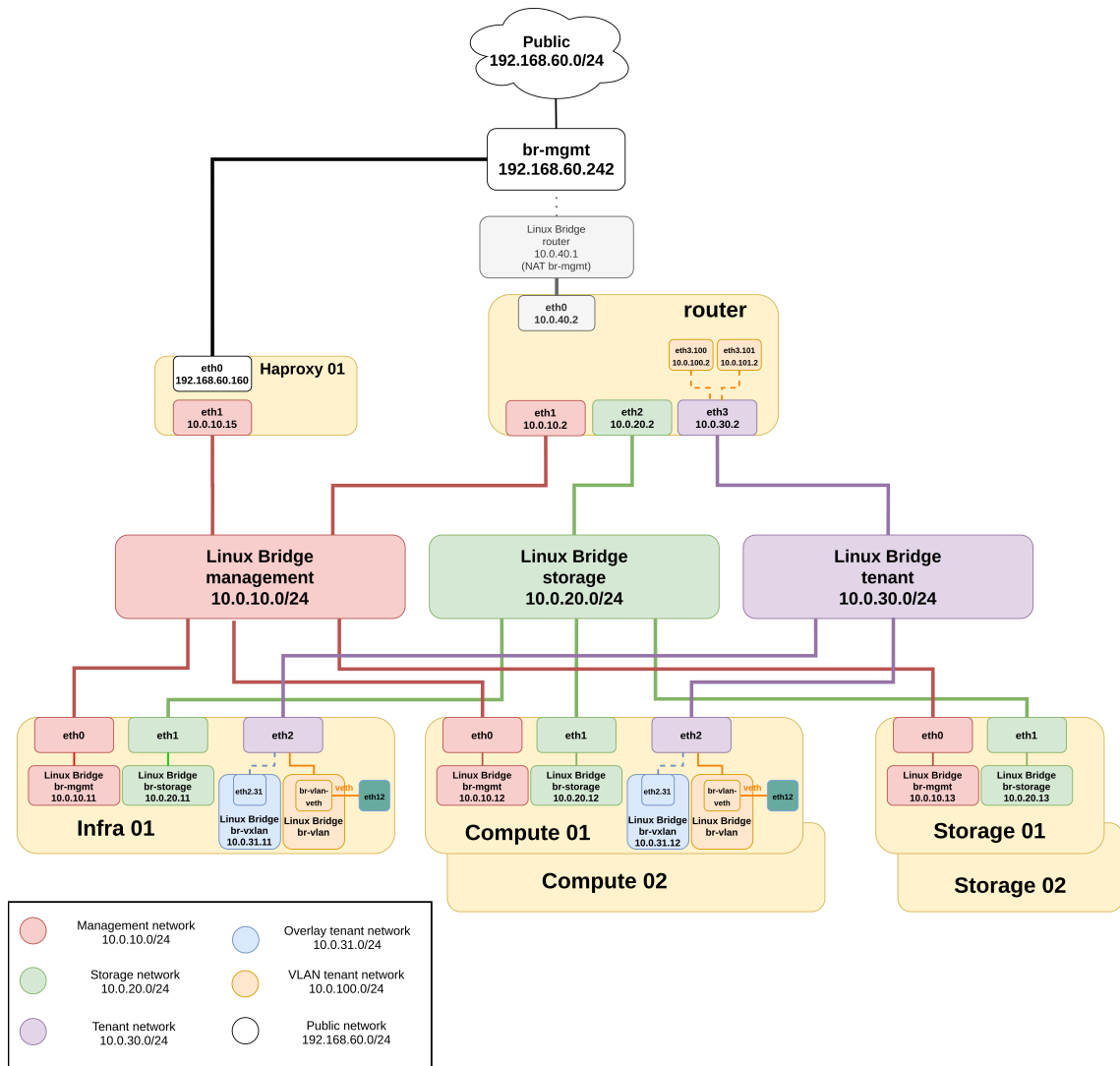


Figura 3.2: Arquitectura diseñada para instalación Stein

los servicios asociados a dicho módulo utilizan un hipervisor para crear nuevas instancias.

En cuanto a los servicios a desplegar, los administradores pueden optar por cuáles instalar dentro de los opcionales y de qué forma con la finalidad de crear un ambiente que se adapte a las necesidades de cada caso. Dicho esto, un aspecto fundamental en todos los servicios y en especial sobre los que interactúan directamente los usuarios finales es la alta disponibilidad.

En este trabajo los servicios que se desplegaron directamente sobre los hosts físicos son los relacionados con Neutron y Nova debido a que agregar una capa de abstracción como son los contenedores no es favorable en ninguno de los casos ya sea por los hipervisores o los recursos de red, en donde ambos requieren tener una comunicación directa con el kernel de Linux. Finalmente los servicios se encuentran distribuidos de la siguiente forma:

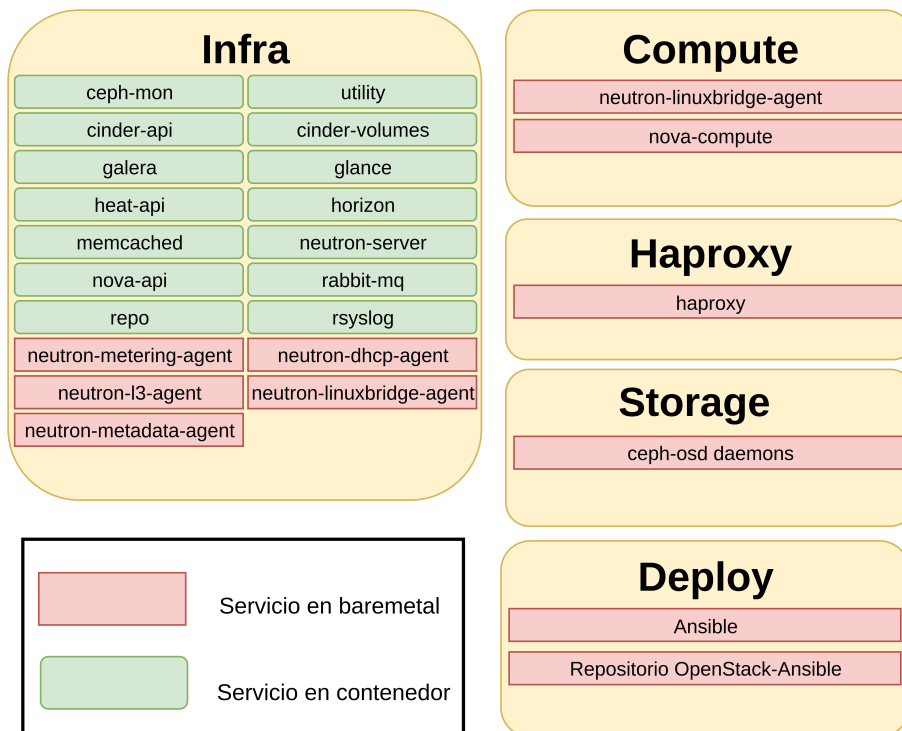


Figura 3.3: Diagrama con la distribución de los servicios

Capítulo 4

Gestión del Datacenter

En este capítulo se detallan un conjunto de tareas de gestión del Datacenter implementado con OpenStack. Estas tareas deben ser realizadas por los usuarios con rol de administrador dentro de la plataforma. En las siguientes secciones se analiza cómo agregar o remover nodos físicos al Datacenter, cómo recuperarse ante posibles fallas y cómo realizar una actualización de versión de OpenStack.

4.1. Recuperación ante fallas

Ante la presencia de problemas en los nodos del Datacenter, como por ejemplo un corte de energía o un mal funcionamiento de algún servicio de OpenStack, será necesaria la participación del administrador del sistema para detectar y solucionar el problema. A continuación se presentan una serie de secciones a modo de guía para que el administrador responda ante los inconvenientes mencionados.

4.1.1. Verificación general

En la instalación de OSA se despliega el container utility en los nodos de infraestructura, el cual provee de una CLI para realizar cambios, instanciar recursos o simplemente verificar las funcionalidades. Para esto se accede a cualquier nodo del tipo mencionado y se siguen los pasos indicados en [60]. En resumen lo que intenta realizar la guía referenciada es ejecutar comandos que involucren a todos los servicios desplegados de OpenStack, haciendo especial hincapié en el estado de los endpoints de los distintos módulos.

4.1.2. Tareas de mantenimiento

Los servicios subyacentes de infraestructura que permiten el funcionamiento de OpenStack se deben verificar y mantener regularmente. La guía [38] muestra en

detalle cómo realizar estas tareas para el cluster de Galera, el cluster de RabbitMQ y el firewall de los nodos físicos. Un resumen de los comandos más relevantes se puede encontrar en [E.1.1](#).

4.1.3. Solución de problemas

En este caso se hace referencia a la guía [\[59\]](#) orientada a resolver problemas operacionales, como pueden ser inconvenientes en la comunicación entre hosts en alguna de las subredes del ambiente de OSA, problemas de conectividad de alguna instancia o problemas con algún servicio de OpenStack. La sección para esto último resulta útil ya que lista todos los servicios existentes para cada módulo de OpenStack y en qué nodo físico se ubican.

4.1.4. Problemas con Ceph

La ventaja de tener un backend como Ceph para Cinder es que está diseñado para soportar la falla de alguno de sus nodos, dependiendo de cuantos OSDs se tengan configurados. Por lo tanto si algún nodo sufre una falla de hardware o tiene otro problema externo como un corte de energía, los usuarios del Datacenter no percibirán problema alguno. Además de los problemas críticos mencionados, se pueden presentar inconvenientes en el funcionamiento del cluster realizado con Ceph. Para diagnosticar y solucionar estos problemas referirse a [E.1.2](#).

4.2. Escalamiento horizontal

4.2.1. Agregar nodo de Cómputo

En función de la demanda requerida por los clientes, es posible que sea necesario aumentar la capacidad de cómputo del Datacenter agregando un nuevo nodo. El paso a paso de cómo realizar esta tarea mediante OSA se describe en [E.2.1](#).

4.2.2. Eliminar un nodo de cómputo

Ante la necesidad de eliminar un nodo de cómputo, por ejemplo por una actualización de hardware, se deben seguir los pasos indicados en [E.2.2](#). Durante este proceso se destaca la importancia de migrar las instancias que se encuentren corriendo en el nodo en cuestión previo a su eliminación.

4.2.3. Agregar nodo de Infraestructura

Agregar nodos de infraestructura es una tarea compleja y sensible debido a todas las tareas que involucra, como por ejemplo modificación de estructuras, creación de nuevos contenedores y la posterior sincronización con los existentes, actualización de los balanceadores de carga, entre otros. En este proceso se obtuvieron diferentes resultados para las versiones Queens y Stein, los procedimientos realizados junto con sus respectivos inconvenientes se detallan en [E.2.3](#).

4.3. Actualización de versión

Las actualizaciones de versiones mayores solamente pueden ser realizadas entre liberaciones secuenciales. Es sumamente importante resaltar que estos procesos de actualización se encuentran en constante desarrollo y por lo tanto se recomienda ensayar el procedimiento en un ambiente de pruebas. Por otro lado, las correcciones que se realizan dentro de una versión se refieren como actualizaciones menores. Estas últimas no serán presentadas pero el proceso para llevarlas a cabo es detallado en [\[46\]](#). Dado que en el trabajo fueron utilizadas las versiones Queens y Stein, se intentó realizar el procedimiento de actualización mayor con cada una de ellas. En particular, la actualización de Queens a Rocky se presenta en detalle en [E.3](#), mencionando los problemas que surgieron en el proceso y sus respectivas soluciones.

Por otro lado, en el caso de la actualización de la versión Stein a Train (siguiendo la guía [\[39\]](#)), no se detallan los pasos específicos sino que se utiliza el escenario para resaltar la importancia de revisar las notas de liberación de la versión antes de actualizar la plataforma para prever posibles inconvenientes [\[58\]](#).

En la liberación Train se actualiza la versión de Ansible utilizada, pasando de 2.7 a 2.8. Esta modificación genera cambios en la forma en que se procesan las configuraciones de Ansible que son correctamente manejados dentro de OSA. Sin embargo, esta situación también lleva a que se deba actualizar el proyecto ceph-ansible de la versión Mimic (3.x) a Nautilus (4.x) en los casos en que se haya desplegado el cluster de Ceph directamente desde OSA. La actualización de versión del cluster de Ceph no es manejada en forma directa por OpenStack-Ansible por lo tanto es necesario que sea realizada por el administrador previo al cambio de Stein a Train, o de lo contrario el proceso fallará con el siguiente mensaje:

```
TASK [ceph-mon : waiting for the monitor(s) to form the quorum...]
fatal: [infra1_ceph-mon_container-83fc4980]: FAILED! => {"msg": "The
  conditional check '(ceph_health_raw.stdout | length > 0) and (
  ceph_health_raw.stdout | default('{}') | from_json)['state'] in ['
```



```
leader', 'peon']\n' failed. The error was: No JSON object could be decoded"}
```

La actualización de un cluster de Ceph mediante Ansible requiere la existencia de al menos 3 nodos monitores, lo cual implica agregar un nuevo nodo de infraestructura a la arquitectura utilizada para la instalación de la versión Stein. Si bien este proceso escapa del alcance del proyecto, se intentó realizar el procedimiento utilizando un ambiente con las condiciones requeridas pero no se obtuvo un resultado exitoso. Esta situación frenó el proceso de actualización de OSA a la versión Train.

Conclusiones

Repasando todas las operaciones descritas en este capítulo es posible obtener varias conclusiones:

- La presencia de fallas es algo que siempre hay que contemplar y más aún en el caso de un Datacenter donde la resiliencia y la conservación de los datos son aspectos primordiales. Para lograr esto se deben tener políticas de respaldo tanto para el cluster de Galera como los almacenamientos de bloque. Por otro lado, el restablecimiento del servicio como se describió en el capítulo puede requerir de una secuencia considerable de comandos o verificaciones, por lo tanto es recomendable tener scripts que automaticen estas tareas ganando tiempo y consistencia.
- En cuanto a realizar modificaciones a la plataforma, tanto para escalarlo horizontalmente como para actualizar la versión utilizada, queda evidenciada a lo largo del capítulo la complejidad y los riesgos que conlleva realizar este tipo de tareas. Por esto siempre es recomendable probar los procedimientos en un ambiente de desarrollo para mitigar la mayor cantidad de inconvenientes que puedan surgir.
- OSA es una plataforma compleja en sí misma, si además se utilizan otras tecnologías para mejorar su funcionamiento, como por ejemplo Ceph en el backend de almacenamiento, es necesario considerar que se introducirán complejidades propias de estas tecnologías. Esto puede implicar modificaciones en OSA para que exista compatibilidad o incluso requerimientos extras en procesos estándares como ocurre en la segunda actualización de versión mencionada.

Capítulo 5

Pruebas de concepto de Neutron

Una vez alcanzada una instalación funcional del Datacenter, el foco de la investigación se centra en el módulo de red Neutron y en cómo funcionan los diversos mechanism drivers que resuelven la conectividad entre instancias virtuales. Particularmente se analiza el comportamiento de Linux Bridge y Open vSwitch, brindando en principio una descripción inicial de su funcionamiento. Luego se muestran escenarios concretos de comunicación entre instancias, diferenciados por el sentido del tráfico y la distribución de red junto con una descripción de los componentes virtuales instanciados y un análisis de tráfico detallado. Para analizar el tráfico se hace uso de la herramienta ping, enviando un único paquete ICMP desde el origen al destino especificados.

5.1. Escenarios de prueba

A continuación se detallan las principales configuraciones de OpenStack que se utilizan para llevar a cabo los escenarios de prueba y la composición de cada uno de ellos.

Flavor	CPUs	RAM	Root Disk
small	1	512	10

Tabla 5.1: Sabores creados para análisis de red

Image	Tipo	Disco min GB	RAM min MB
cirros	qcow2	10	512

Tabla 5.2: Imágenes creadas para análisis de red

Provider network	Network type	Physical network	Segmentation ID	External network
provider-vlan	VLAN	vlan	100	True

Tabla 5.3: Redes provider creadas para análisis de red

Provider network	Subnet provider	CIDR	Gateway	DHCP	Allocation pools	DNS
provider-flat	subnet-provider-flat	10.0.30.0/24	10.0.30.2	True	10.0.30.50, 10.0.30.150	192.168.60.230
provider-vlan	subnet-provider-vlan	10.0.100.0/24	10.0.100.2	True	10.0.100.50, 10.0.100.150	192.168.60.230

Tabla 5.4: Subredes provider creadas para análisis de red

Los escenarios de prueba que se plantean pretenden cubrir las distintas combinaciones de tráfico entre instancias, ya sea dentro (de forma horizontal) o fuera (de forma vertical) del Datacenter.

5.1.1. Escenario 1: tráfico este-oeste (misma red tenant)

Ejemplifica la comunicación de red entre dos instancias asociadas a la misma red tenant de tipo VXLAN alojadas en diferentes nodos de cómputo. Esto permite apreciar cómo Neutron resuelve el pasaje de la red virtual a la red física subyacente. En caso de que las instancias estuvieran en el mismo nodo de cómputo, el tráfico se resolvería por completo mediante los componentes virtuales dentro de un único nodo.

5.1.1.1. Composición del escenario

- Red tenant 1 de tipo VXLAN (5.5).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (5.6).
- Instancia 2 alojada en el nodo de cómputo 2 asociada a la red tenant 1 (5.6).

5.1.2. Escenario 2: tráfico este-oeste (distintas redes tenant)

Ejemplifica la comunicación de red entre dos instancias asociadas a distintas redes tenant de tipo VXLAN. El nodo de cómputo en el que residan las instancias no tiene relevancia para este escenario, sin embargo para uniformizar las pruebas con los distintos mechanism drivers las instancias se alojan en distintos nodos de cómputo. Este escenario permite analizar cómo Neutron resuelve el routing mediante

Network	tenant-network-1
Subnet	tenant-subnet-1
CIDR	192.168.1.0/24
Allocation Pools	192.168.1.50, 192.168.1.150
DNS	192.168.60.230
Gateway	192.168.1.1
DHCP	True

Tabla 5.5: Escenario 1: detalles de la subred 1

Instance	Image	Flavor	Network	IP
instance-1	cirros	small	tenant-network-1	192.168.1.101
instance-2	cirros	small	tenant-network-1	192.168.1.102

Tabla 5.6: Escenario 1: detalles de las instancias

el módulo L3 entre redes internas del Datacenter. A diferencia del primer ambiente, el nodo de red albergará varios componentes para la comunicación.

5.1.2.1. Composición del escenario

- Red tenant 1 de tipo VXLAN (5.7).
- Red tenant 2 de tipo VXLAN (5.8).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (5.9).
- Instancia 2 alojada en el nodo de cómputo 2 asociada a la red tenant 2 (5.9).
- Router X asociado a las redes tenant 1 y 2 (5.10).

Network	tenant-network-1
Subnet	tenant-subnet-1
CIDR	192.168.1.0/24
Allocation Pools	192.168.1.50, 192.168.1.150
DNS	192.168.60.230
Gateway	192.168.1.1
DHCP	True

Tabla 5.7: Escenario 2: detalles de la subred 1

5.1.3. Escenario 3: tráfico norte-sur (acceso hacia el exterior)

Ejemplifica la comunicación de una instancia asociada a una red tenant de tipo VXLAN, con un host externo a OpenStack a través de una red provider

Network	tenant-network-2
Subnet	tenant-subnet-2
CIDR	192.168.2.0/24
Allocation Pools	192.168.2.50, 192.168.2.150
DNS	192.168.60.230
Gateway	192.168.2.1
DHCP	True

Tabla 5.8: Escenario 2: detalles de la subred 2

Instance	Image	Flavor	Network	IP
instance-1	cirros	small	tenant-network-1	192.168.1.101
instance-2	cirros	small	tenant-network-2	192.168.2.102

Tabla 5.9: Escenario 2: detalles de las instancias

Router	External network	Port 1 IP	Port 2 IP
router-X	-	192.168.1.1	192.168.2.1

Tabla 5.10: Escenario 2: detalles del router

de tipo VLAN. Este escenario permite analizar cómo Neutron resuelve el routing mediante el módulo L3 entre una red interna del Datacenter y una red externa. Adicionalmente a los componentes analizados en el escenario anterior, también será relevante inspeccionar los componentes físicos que dan soporte a la infraestructura de OpenStack.

5.1.3.1. Composición del escenario

- Red tenant 1 de tipo VXLAN (5.11).
- Red provider vlan de tipo VLAN (5.12).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (5.13).
- Router Y asociado a las redes tenant 1 y provider vlan (5.14).

Network	tenant-network-1
Subnet	tenant-subnet-1
CIDR	192.168.1.0/24
Allocation Pools	192.168.1.50, 192.168.1.150
DNS	192.168.60.230
Gateway	192.168.1.1
DHCP	True

Tabla 5.11: Escenario 3: detalles de la subred 1

Network	provider-vlan
Subnet	subnet-provider-vlan
CIDR	10.0.100.0/24
Allocation Pools	10.0.100.50, 10.0.100.150
DNS	192.168.60.230
Gateway	10.0.100.2
DHCP	True
VLAN ID	100

Tabla 5.12: Escenario 3: detalles de la subred provider vlan

Instance	Image	Flavor	Network	IP
instance-1	cirros	small	tenant-network-1	192.168.1.101

Tabla 5.13: Escenario 3: detalles de las instancias

Router	External network	External fixed IP	Port 1 IP
router-Y	provider-vlan	DHCP	192.168.1.1

Tabla 5.14: Escenario 3: detalles del router

5.1.4. Escenario 4: tráfico norte-sur (acceso desde el exterior)

Ejemplifica la comunicación hacia una instancia asociada a una red tenant de tipo VXLAN, desde un host externo a OpenStack a través de una red provider de tipo VLAN. Este escenario permite analizar cómo Neutron resuelve el routing mediante el módulo L3 desde una red externa al Datacenter hacia una red interna. En este caso los componentes a inspeccionar coinciden con los del escenario anterior, con la salvedad de que se agrega una propiedad de la instancia denominada floating IP. Esta última es requerida para tener acceso directo desde el exterior.

5.1.4.1. Composición del escenario

- Red tenant 1 de tipo VXLAN (5.15).
- Red provider vlan de tipo VLAN (5.16).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (5.17).
- Router Z asociado a las redes tenant 1 y provider vlan (5.18).

Network	tenant-network-1
Subnet	tenant-subnet-1
CIDR	192.168.1.0/24
Allocation Pools	192.168.1.50, 192.168.1.150
DNS	192.168.60.230
Gateway	192.168.1.1
DHCP	True

Tabla 5.15: Escenario 4: detalles de la subred 1

Network	provider-vlan
Subnet	subnet-provider-vlan
CIDR	10.0.100.0/24
Allocation Pools	10.0.100.50, 10.0.100.150
DNS	192.168.60.230
Gateway	10.0.100.2
DHCP	True
VLAN ID	100

Tabla 5.16: Escenario 4: detalles de la subred provider vlan

Instance	Image	Flavor	Network	IP	Floating IP
instance-1	cirros	small	tenant-network-1	192.168.1.101	DHCP

Tabla 5.17: Escenario 4: detalles de las instancias

Router	External network	External fixed IP	Port 1 IP
router-Z	provider-vlan	DHCP	192.168.1.1

Tabla 5.18: Escenario 4: detalles del router

5.2. Linux Bridge

Escenario 1

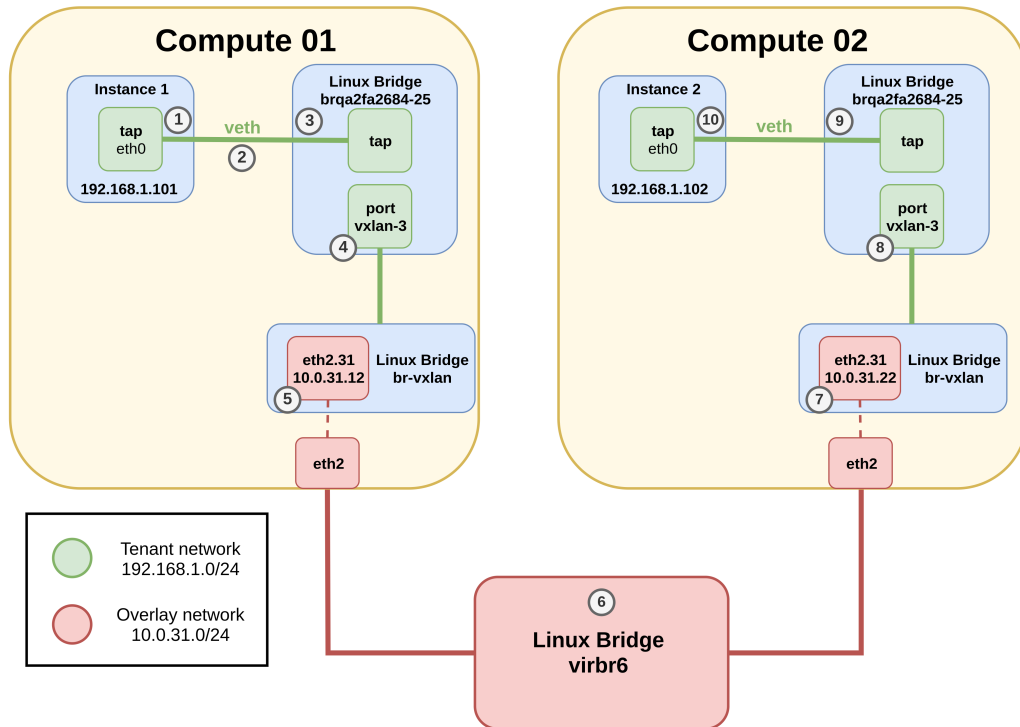


Figura 5.1: Diagrama de arquitectura para el escenario 1 de Linux Bridge

5.2.0.1. Análisis de componentes

A continuación se detallan los componentes virtuales requeridos para instanciar el escenario propuesto.

Nodos de cómputo

El hipervisor utilizado (KVM) se encarga de crear una nueva máquina virtual para cada nueva instancia definida. Cada VM debe estar conectada al menos a una red, en este caso se trata de la red tenant 192.168.1.0/24. Esta última es instanciada mediante un Linux Bridge que actúa como switch a nivel local dentro de cada nodo de cómputo y provee conexión hacia afuera. Este bridge es creado por Neutron utilizando un identificador con el formato `<brq+id_net>`, siendo `id_net` los primeros 10 caracteres del UUID de la red en cuestión. En forma práctica se puede obtener información de estos componentes mediante los comandos mostrados a continuación.

Listar la red del Escenario 1 para obtener el identificador, cuyos primeros dígitos serán utilizados para formar el nombre del bridge `brqa2fa2684-25`.


```
[root@infra1-utility-container-eebf40f8 ~]# openstack network list --
  project "Escenario 1" -f json
[
  {
    "Subnets": "3c7ee681-e670-4f53-bfe1-cd96f15575e2",
    "ID": "a2fa2684-2573-4e0d-9db2-d555fbdecee4",
    "Name": "tenant-network-1"
  }
]
```

A este bridge se le asocian dos puertos de red necesarios para llevar a cabo la comunicación entre la instancia y la red. Estas interfaces se pueden listar mediante:

```
[root@compute1 ~]# brctl show brqa2fa2684-25
bridge name      bridge id          STP enabled      interfaces
brqa2fa2684-25   8000.6e2b1cee7e5e no                 tapf0dd10ee-cb
                                                          vxlan-3
```

La interfaz vxlan-X es la encargada de encapsular el tráfico VXLAN con VNI X (en este caso el 3) que será enviado a la red de overlay. Esto último se logra gracias a que también se encuentra conectada al br-vxlan. La interfaz `tapf0dd10ee-cb` es un veth creado para que la instancia tenga conectividad con la red tenant. Un extremo de esta interfaz está asociado al bridge de la red, mientras que el otro se asocia con la interfaz `eth0` de la instancia creada. Con los siguientes comandos se puede corroborar la correspondencia de la interfaz virtual con la máquina instanciada por KVM.

Listar las instancias del escenario 1:

```
[root@infra1-utility-container-eebf40f8 ~]# openstack server list --
  project "Escenario 1"
[
  {
    "Status": "ACTIVE",
    "Name": "instance-1",
    "Image": "",
    "ID": "0cbc782a-9ebb-479e-b066-37ee72865909",
    "Flavor": "small",
    "Networks": "tenant-network-1=192.168.1.101"
  },
  {
    "Status": "ACTIVE",
    "Name": "instance-2",
    "Image": "",
```

```

        "ID": "2fcd5a2e-18aa-4903-ae96-6c89d801a08b",
        "Flavor": "small",
        "Networks": "tenant-network-1=192.168.1.102"
    }
]

```

Obtener detalles de la instancia 1:

```

[root@infra1-utility-container-eebf40f8 ~]# openstack server show 0cbc782a
-9ebb-479e-b066-37ee72865909 -f json

```

```

{
    "OS-EXT-STS:task_state": null,
    "addresses": "tenant-network-1=192.168.1.101",
    "image": "",
    "OS-EXT-STS:vm_state": "active",
    "OS-EXT-SRV-ATTR:instance_name": "instance-0000000f",
    "OS-SRV-USG:launched_at": "2019-12-14T15:49:25.000000",
    "flavor": "small (f302f726-10a9-46a7-9a6d-23cc14b324db)",
    "id": "0cbc782a-9ebb-479e-b066-37ee72865909",
    "security_groups": "name='default'",
    "volumes_attached": "id='38b9d1de-670f-487c-b4c7-48d680ac0945'",
    "user_id": "7d5f584230924048bbf9b406a1656855",
    "OS-DCF:diskConfig": "AUTO",
    "accessIPv4": "",
    "accessIPv6": "",
    "progress": 0,
    "OS-EXT-STS:power_state": "Running",
    "OS-EXT-AZ:availability_zone": "nova",
    "config_drive": "",
    "status": "ACTIVE",
    "updated": "2019-12-14T15:49:25Z",
    "hostId": "267cd9c5264c2ad32926c7d63ffeb95e0afe0e2c9bceccb9b8a3c43d",
    "OS-EXT-SRV-ATTR:host": "compute1",
    "OS-SRV-USG:terminated_at": null,
    "key_name": null,
    "properties": "",
    "project_id": "b7b6e99be1e843339d0969341f9b5a13",
    "OS-EXT-SRV-ATTR:hypervisor_hostname": "compute1.openstack.local",
    "name": "instance-1",
    "created": "2019-12-14T15:49:09Z"
}

```

Listar detalles de las interfaces creadas por KVM en el nodo de cómputo 1 asociadas a la instancia 1:

```
[root@compute1 ~]# virsh domiflist instance-0000000f
Interface      Type      Source      Model  MAC
-----
tapf0dd10ee-cb bridge    brqa2fa2684-25  virtio fa:16:3e:d2:5b:cc
```

5.2.0.2. Análisis de tráfico

En este primer ejemplo se analiza el tráfico generado cuando la instancia 1 intenta comunicarse con la instancia 2, asumiendo que las mismas aún no conocen las direcciones MAC destino.

Paso 1 La instancia 1 envía el paquete ICMP echo request hacia la instancia 2. Para esto determina que se encuentra directamente conectada con el host destino y por lo tanto no necesita ningún salto intermedio.

Paso 2 La instancia 1 debe obtener la MAC de la instancia 2, como esta no es conocida se dispara el protocolo ARP. A continuación se detalla cómo se lleva a cabo el protocolo de resolución de direcciones entre las instancias virtuales.

1. La instancia 1 envía una trama ARP request preguntando por la IP 192.168.1.102 (1). Dicha trama es enviada hacia el Linux Bridge asociado a la red tenant a través del veth pair (2).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fa:16:3e:d2:5b:cc	Broadcast	ARP	42	Who has 192.168.1.102? Tell 192.168.1.101
2	0.011413	fa:16:3e:b1:8f:68	fa:16:3e:d2:5b:cc	ARP	42	192.168.1.102 is at fa:16:3e:b1:8f:68

```

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  Source: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
  Type: ARP (0x0806)
Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
  Sender IP address: 192.168.1.101
  Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.1.102

```

Figura 5.2: Paquete ARP request capturado en la interfaz eth0 de la instancia 1

2. El bridge recibe el pedido ARP (3) y lo reenvía a través del puerto vxlan-3 (4). Este último se encarga de envolver la trama original, generando el paquete IP del protocolo VXLAN teniendo en cuenta que se trata de un broadcast de capa 2. El paquete contendrá el identificador VXLAN de la red tenant en cuestión (VNI 3), la IP del VTEP de origen (10.0.31.12) y la dirección del grupo multicast sobre el cual se lleva a cabo la red de overlay (239.1.1.1).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fa:16:3e:d2:5b:cc	Broadcast	ARP	92	Who has 192.168.1.102? Tell 192.168.1.101
2	0.011146	fa:16:3e:b1:8f:68	fa:16:3e:d2:5b:cc	ARP	92	192.168.1.102 is at fa:16:3e:b1:8f:68

```

▶ Frame 1: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)
▶ Ethernet II, Src: RealtekU_be:9c:39 (52:54:00:be:9c:39), Dst: IPv4mcast_01:01:01 (01:00:5e:01:01:01)
▶ Internet Protocol Version 4, Src: 10.0.31.12, Dst: 239.1.1.1
▶ User Datagram Protocol, Src Port: 44739, Dst Port: 8472
▼ Virtual eXtensible Local Area Network
  ▶ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 3
    Reserved: 0
  ▼ Ethernet II, Src: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
    ▶ Source: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
    Type: ARP (0x0806)
  ▼ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
    Sender IP address: 192.168.1.101
    Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.1.102

```

Figura 5.3: Paquete ARP request encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1

- El paquete es recibido por el Linux Bridge br-vxlan y reenviado a través de la sub-interfaz eth2.31 al grupo de multicast mencionado (5). El mismo es transportado mediante la infraestructura física subyacente (6), alcanzando a todos los VTEPs del grupo.
- Luego en cada VTEP se comprueba que el VNI recibido coincida con el de algún segmento VXLAN que manejado por el mismo (7). En caso negativo, el paquete es descartado. En este paso todos los VTEP del grupo de multicast guardan en sus tablas de forwarding, la correspondencia de la MAC de la instancia 1 con la IP del VTEP origen, en este caso el nodo compute01.

Tabla de forwarding antes:

```
[root@compute2 ~]# bridge fdb show dev vxlan-3
2e:a2:c8:44:94:2c vlan 1 master brqa2fa2684-25 permanent
2e:a2:c8:44:94:2c master brqa2fa2684-25 permanent
00:00:00:00:00:00 dst 239.1.1.1 via br-vxlan self permanent
```

Tabla de forwarding después

```
[root@compute2 ~]# bridge fdb show dev vxlan-3
2e:a2:c8:44:94:2c vlan 1 master brqa2fa2684-25 permanent
2e:a2:c8:44:94:2c master brqa2fa2684-25 permanent
fa:16:3e:0e:1d:fd master brqa2fa2684-25
fa:16:3e:d2:5b:cc master brqa2fa2684-25
00:00:00:00:00:00 dst 239.1.1.1 via br-vxlan self permanent
fa:16:3e:d2:5b:cc dst 10.0.31.12 self
fa:16:3e:0e:1d:fd dst 10.0.31.11 self
```

5. En caso positivo se reenvía el paquete al puerto vxlan-3 en donde se eliminan los cabezales del protocolo VXLAN obteniendo la trama original (8). La petición ARP es enviada al Linux Bridge asociado a la red tenant.
6. Debido a que se trata de un ARP request, el bridge reenvía la trama a todas las instancias asociadas al mismo utilizando los veth pairs correspondientes (9).
7. Todas las instancias recibirán la trama, pero solo aquella por cuya IP se está consultando responderá a la instancia 1 con un ARP reply (10) directo a su MAC. En este paso todas las instancias aprenden la MAC de la instancia 1 (tabla ARP).
8. El Linux Bridge recibe la trama ARP (9) y la reenvía a través del puerto vxlan-3, el cual encapsula la trama manteniendo el mismo VNI. A diferencia de la consulta y gracias a que en el paso 4 se actualizó la tabla de redireccionamiento, la IP destino del paquete VXLAN será la del nodo de cómputo 1 (10.0.31.12).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fa:16:3e:d2:5b:cc	Broadcast	ARP	92	Who has 192.168.1.102? Tell 192.168.1.101
2	0.011146	fa:16:3e:b1:8f:68	fa:16:3e:d2:5b:cc	ARP	92	192.168.1.102 is at fa:16:3e:b1:8f:68

```

<
> Frame 2: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)
> Ethernet II, Src: RealtekU aa:e4:85 (52:54:00:aa:e4:85), Dst: RealtekU be:9c:39 (52:54:00:be:9c:39)
> Internet Protocol Version 4, Src: 10.0.31.22, Dst: 10.0.31.12
> User Datagram Protocol, Src Port: 52349, Dst Port: 8472
< Virtual extensible Local Area Network
  > Flags: 0x0800, VXLAN Network ID (VNI)
    > Group Policy ID: 0
    > VXLAN Network Identifier (VNI): 3
    > Reserved: 0
< Ethernet II, Src: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68), Dst: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
  > Destination: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
  > Source: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68)
  > Type: ARP (0x0806)
< Address Resolution Protocol (reply)
  > Hardware type: Ethernet (1)
  > Protocol type: IPv4 (0x0800)
  > Hardware size: 6
  > Protocol size: 4
  > Opcode: reply (2)
  > Sender MAC address: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68)
  > Sender IP address: 192.168.1.102
  > Target MAC address: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
  > Target IP address: 192.168.1.101

```

Figura 5.4: Paquete ARP reply encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1

9. El paquete respuesta es enviado a través de la sub-interfaz eth2.31 (7) hacia el nodo de cómputo 1 mediante la infraestructura física (6).
10. Cuando al VTEP del nodo de cómputo 1 recibe el paquete (5), comprueba que el VNI pertenece a alguno de sus segmentos VXLAN y lo reenvía al puerto vxlan-3 asociado. Adicionalmente actualiza su tabla de forwarding con la correspondencia entre la MAC de la instancia 2 y la IP del VTEP del nodo de cómputo 2.

Tabla de forwarding antes:

```
[root@compute1 ~]# bridge fdb show dev vxlan-3
6e:2b:1c:ee:7e:5e  vlan 1 master brqa2fa2684-25 permanent
6e:2b:1c:ee:7e:5e master brqa2fa2684-25 permanent
00:00:00:00:00:00 dst 239.1.1.1 via br-vxlan self permanent
```

Tabla de forwarding después

```
[root@compute1 ~]# bridge fdb show dev vxlan-3
6e:2b:1c:ee:7e:5e  vlan 1 master brqa2fa2684-25 permanent
6e:2b:1c:ee:7e:5e master brqa2fa2684-25 permanent
fa:16:3e:0e:1d:fd master brqa2fa2684-25
fa:16:3e:b1:8f:68 master brqa2fa2684-25
00:00:00:00:00:00 dst 239.1.1.1 via br-vxlan self permanent
fa:16:3e:b1:8f:68 dst 10.0.31.22 self
fa:16:3e:0e:1d:fd dst 10.0.31.11 self
```

- En (4), se desencapsula el paquete y se reenvía la trama de respuesta a la instancia 1 a través del Linux Bridge y el correspondiente veth pair.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fa:16:3e:d2:5b:cc	Broadcast	ARP	42	Who has 192.168.1.102? Tell 192.168.1.101
2	0.011413	fa:16:3e:b1:8f:68	fa:16:3e:d2:5b:cc	ARP	42	192.168.1.102 is at fa:16:3e:b1:8f:68


```

Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68), Dst: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
  Destination: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
  Source: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68)
  Type: ARP (0x0806)
  Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68)
    Sender IP address: 192.168.1.102
    Target MAC address: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)
    Target IP address: 192.168.1.101

```

Figura 5.5: Paquete ARP reply capturado en la interfaz eth0 de la instancia 1

De ahora en más, mientras se mantengan actualizadas las tablas de resolución ARP de las instancias y las tablas de forwarding de los VTEPs, la comunicación entre ambas máquinas virtuales será siempre en formato unicast de forma análoga a lo detallado para el ARP reply. Cuando alguna de las tablas mencionadas deba refrescar sus datos, la comunicación a nivel de la red de overlay será mediante un multicast, al igual que lo analizado en el ARP request.

Paso 3 Ahora que se conoce la dirección MAC, la instancia 1 retoma el envío del paquete ICMP hacia la instancia 2.

- El mensaje es enviado por el veth pair (2) hacia el bridge de la red tenant-network-1 (3).

```

3 0.014151 192.168.1.101 192.168.1.102 ICMP 98 Echo (ping) request id=0x8c01, seq=0/0, t
4 0.019718 192.168.1.102 192.168.1.101 ICMP 98 Echo (ping) reply id=0x8c01, seq=0/0, t
▶ Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▶ Ethernet II, Src: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc), Dst: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68)
▼ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.102
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 84
  Identification: 0x6d7d (28029)
  ▶ Flags: 0x4000, Don't fragment
  Time to live: 64
  Protocol: ICMP (1)
  Header checksum: 0x4910 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.1.101
  Destination: 192.168.1.102
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xa80b [correct]
  [Checksum Status: Good]
  Identifier (BE): 35841 (0x8c01)
  Identifier (LE): 396 (0x018c)
  Sequence number (BE): 0 (0x0000)
  Sequence number (LE): 0 (0x0000)
  [Response frame: 4]
▶ Data (56 bytes)

```

Figura 5.6: Paquete ICMP request capturado en la interfaz eth0 de la instancia 1

2. En el bridge se aplican las reglas del grupo de seguridad. Luego la trama es reenviada hacia el Linux Bridge br-vxlan a través de la subinterfaz vxlan-3 (4). En este paso se encapsula la trama original en un paquete IP del protocolo VXLAN. En este caso como en la tabla de forwarding existe una entrada que relaciona la IP del destinatario con un VTEP, la comunicación será unicast.
3. El paquete es recibido por el Linux Bridge br-vxlan y reenviado a través de la sub-interfaz eth2.31 hacia el nodo de cómputo 2 (5). El mismo es transportado mediante la infraestructura física subyacente (6), alcanzando el VTEP destino.

```

3 0.014148 192.168.1.101 192.168.1.102 ICMP 148 Echo (ping) request id=0x8c01, seq=0/0, t
4 0.019493 192.168.1.102 192.168.1.101 ICMP 148 Echo (ping) reply id=0x8c01, seq=0/0, t
▶ Frame 3: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
▶ Ethernet II, Src: RealtekU_be:9c:39 (52:54:00:be:9c:39), Dst: RealtekU_aa:e4:85 (52:54:00:aa:e4:85)
▼ Internet Protocol Version 4, Src: 10.0.31.12, Dst: 10.0.31.22
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 134
  Identification: 0x3608 (13832)
  ▶ Flags: 0x0000
  Time to live: 32
  Protocol: UDP (17)
  Header checksum: 0x123e [validation disabled]
  [Header checksum status: Unverified]
  Source: 10.0.31.12
  Destination: 10.0.31.22
▶ User Datagram Protocol, Src Port: 36814, Dst Port: 8472
▼ Virtual eXtensible Local Area Network
  ▶ Flags: 0x0800, VXLAN Network ID (VNI)
  Group Policy ID: 0
  VXLAN Network Identifier (VNI): 3
  Reserved: 0
▶ Ethernet II, Src: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc), Dst: fa:16:3e:b1:8f:68 (fa:16:3e:b1:8f:68)
▶ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.102
▶ Internet Control Message Protocol

```

Figura 5.7: Paquete ICMP request encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1

4. El paquete es recibido por el Linux Bridge br-vxlan del nodo de cómputo 2 a través de la subinterfaz eth2.3 (7). Al verificar que el VNI es válido se reenvía el paquete al puerto vxlan-3 en donde se eliminan los cabezales del protocolo VXLAN obteniendo la trama original (8). El paquete ICMP request es enviado

al bridge de la red tenant.

- En el bridge se aplican las reglas del grupo de seguridad y en caso de no filtrar, el paquete se envía a la instancia 2 a través de la tap asociada a la misma (9).

Paso 4 El procedimiento para la respuesta con el paquete ICMP reply es similar al descrito pero en sentido contrario.

Escenario 2

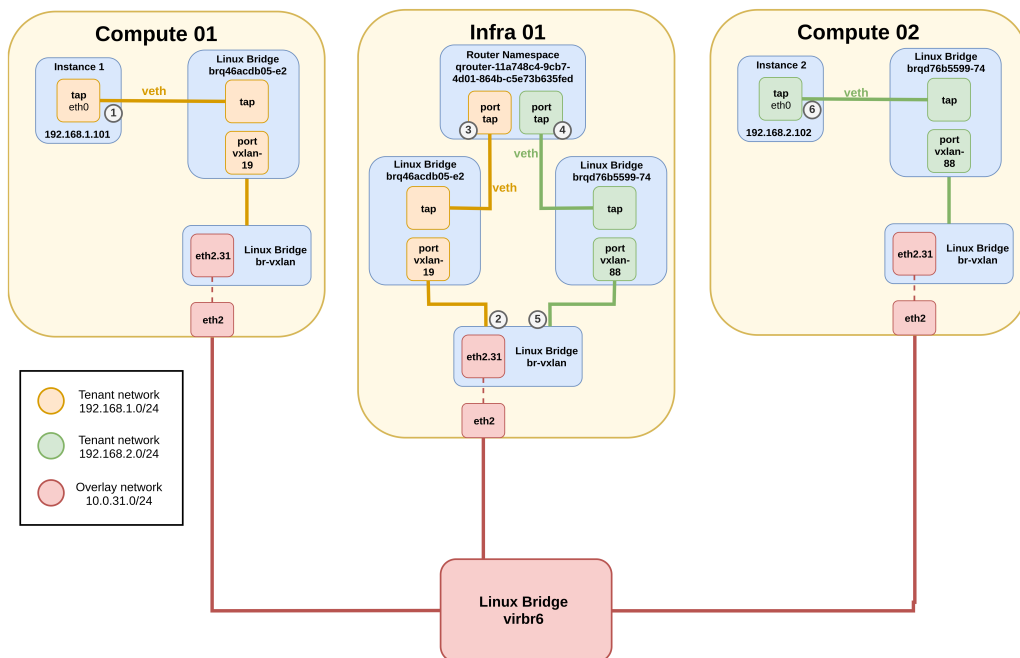


Figura 5.8: Diagrama de arquitectura para el escenario 2 de Linux Bridge

5.2.0.3. Análisis de componentes

A continuación se detallan los componentes virtuales requeridos para instanciar el escenario propuesto. Salvo diferencias en los identificadores, los componentes que son creados en los nodos de cómputo son iguales a los del escenario 1, por lo cual no es relevante volver a detallarlos. Por el contrario, sí se detallan los nuevos componentes en el nodo de infraestructura encargados de interconectar las redes tenant.

Nodos de infraestructura/red

Las redes de Neutron en este escenario son:

```
[root@infra1-utility-container-eebf40f8 ~]# openstack network list --
project "Escenario 2" -f json
[
```



```

    {
      "Subnets": "2665f57b-656b-41ae-b354-853568d48576",
      "ID": "46acdb05-e28c-4399-ae59-73b9b3a4af0b",
      "Name": "tenant-network-1"
    },
    {
      "Subnets": "31d3b88c-c2b3-4115-a3b4-0f3eba06a7c2",
      "ID": "d76b5599-74df-40a7-81f4-64a0b7320e85",
      "Name": "tenant-network-2"
    }
  ]

```

La comunicación entre dos subredes diferentes requiere de un router como intermediario. Este router es instanciado por Neutron en el nodo de red utilizando un network namespace identificado por el nombre `qrouter-<id_router>`. Los siguientes comandos ofrecen información acerca de los routers del escenario.

Obtener un listado de los routers:

```

[root@infra1-utility-container-eebf40f8 ~]# openstack router list --
  project "Escenario 2" -f json
[
  {
    "Status": "ACTIVE",
    "Name": "router-X",
    "Project": "bf19b626dfbc4b6f90763e2600a788b2",
    "State": "UP",
    "HA": false,
    "ID": "11a748c4-9cb7-4d01-864b-c5e73b635fed"
  }
]

```

Acceder a la información del router X:

```

[root@infra1-utility-container-eebf40f8 ~]# openstack router show 11a748c4
  -9cb7-4d01-864b-c5e73b635fed -f json
{
  "external_gateway_info": null,
  "status": "ACTIVE",
  "availability_zone_hints": "",
  "availability_zones": "nova",
  "description": "",
  "admin_state_up": "UP",
  "created_at": "2019-11-25T20:27:54Z",

```

```

"tags": "",
"updated_at": "2019-12-14T15:35:14Z",
"name": "router-X",
"interfaces_info": "[{\"subnet_id\": \"2665f57b-656b-41ae-b354-853568d48576\", \"ip_address\": \"192.168.1.1\", \"port_id\": \"b1d7f4c9-f6fa-4d0b-bf18-cc7df44a8777\"}, {\"subnet_id\": \"31d3b88c-c2b3-4115-a3b4-0f3eba06a7c2\", \"ip_address\": \"192.168.2.1\", \"port_id\": \"c2b59f24-f871-48ac-91e1-531b42952adc\"}]",
"project_id": "bf19b626dfbc4b6f90763e2600a788b2",
"flavor_id": null,
"revision_number": 4,
"routes": "",
"ha": false,
"id": "11a748c4-9cb7-4d01-864b-c5e73b635fed",
"location": {
  "project": {
    "domain_id": null,
    "id": "bf19b626dfbc4b6f90763e2600a788b2",
    "name": null,
    "domain_name": null
  },
  "zone": null,
  "region_name": "RegionOne",
  "cloud": ""
}
}

```

Sabiendo el identificador del router es posible acceder al namespace, siguiendo el formato mencionado, mediante el siguiente comando:

```
[root@infra1 ~]# ip netns exec qrouter-11a748c4-9cb7-4d01-864b-c5e73b635fed <command>
```

El namespace asociado al router tiene tantas interfaces como puertos tenga configurados, en este caso son dos debido a que se están conectando dos redes tenant. Con los siguientes comandos se analizan las interfaces mencionadas:

Listar interfaces del namespace:

```
[root@infra1 ~]# ip netns exec qrouter-11a748c4-9cb7-4d01-864b-c5e73b635fed ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
```

```

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: qr-b1d7f4c9-f6@if88: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UP group default qlen 1000
    link/ether fa:16:3e:99:f9:9f brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.1/24 brd 192.168.1.255 scope global qr-b1d7f4c9-f6
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe99:f99f/64 scope link
        valid_lft forever preferred_lft forever
3: qr-c2b59f24-f8@if94: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UP group default qlen 1000
    link/ether fa:16:3e:4b:35:4b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.2.1/24 brd 192.168.2.255 scope global qr-c2b59f24-f8
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe4b:354b/64 scope link
        valid_lft forever preferred_lft forever

```

Los nombres de las interfaces de los routers tienen el siguiente formato: qr-`<id_port>`, siendo el `id_port` los primeros diez caracteres del identificador del puerto, los cuales se pueden listar con el siguiente comando en donde el `device.id` es el identificador del router.

```
[root@infra1-utility-container-eebf40f8 ~]# openstack port list --device-id <device_id>
```

Para que el router se encuentre asociado a ambas redes, es necesario que las mismas se encuentren instanciadas en el nodo de red. Esto se logra de la misma forma que en los nodos de cómputo, mediante un Linux Bridge que actúa como switch local y provee conectividad hacia el resto de la red. El nombre de los bridges vinculados a una misma red es el mismo en los distintos nodos de OpenStack dado que este depende del identificador de la red de Neutron.

La explicación de los componentes de la red tenant se realizará solamente para la red tenant-network-1 dado que para la tenant-network-2 es análoga. A este bridge se le asocian tres puertos de red necesarios para llevar a cabo la comunicación entre la red tenant con el servicio de DHCP y el router. Los siguientes comandos permiten obtener información acerca de estas interfaces.

Listar las interfaces del bridge:

```
[root@infra1 ~]# brctl show brq46acdb05-e2
```

bridge name	bridge id	STP enabled interfaces
-------------	-----------	------------------------

```
brq46acdb05-e2 8000.023308eec5d0 no          tapb1d7f4c9-f6
                                              tapb43bdd81-01
                                              vxlan-19
```

La interfaz vxlan-X es la encargada de encapsular el tráfico VXLAN con VNI X (en este caso el 19) que será enviado a la red de overlay. Esto último se logra gracias a que también se encuentra conectada al br-vxlan.

Para analizar la primer tap listada se utilizan los siguientes comandos:

```
[root@infra1 ~]# ip link show tapb1d7f4c9-f6
88: tapb1d7f4c9-f6@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
    noqueue master brq46acdb05-e2 state UP mode DEFAULT group default qlen
    1000
link/ether 02:33:08:ee:c5:d0 brd ff:ff:ff:ff:ff:ff link-netnsid 26
```

Esto indica que la tap es la interfaz con índice 88 y que esta linkeada con la interfaz con índice 2 (@if2) del namespace 26 (link-netnsid 26). Buscando dicho namespace se obtiene el siguiente resultado:

```
[root@infra1 ~]# ip netns | grep "id: 26"
qrouter-11a748c4-9cb7-4d01-864b-c5e73b635fed (id: 26)
```

Luego ejecutando el comando “ip a” en el namespace se obtiene la interfaz del router linkeada con la tap.

Para la siguiente tap se realiza el mismo procedimiento:

```
[root@infra1 ~]# ip link show tapb43bdd81-01
77: tapb43bdd81-01@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
    noqueue master brq46acdb05-e2 state UP mode DEFAULT group default qlen
    1000
link/ether 36:c3:c4:50:87:1e brd ff:ff:ff:ff:ff:ff link-netnsid 15
```

En este caso la tap es la interfaz con índice 77 y está linkeada con la interfaz con índice 2 del namespace 15. Luego buscando dicho namespace se obtiene que el mismo es utilizado para brindar el servicio de DHCP para la red tenant.

```
[root@infra1 ~]# ip netns | grep "id: 15"
qdhcp-46acdb05-e28c-4399-ae59-73b9b3a4af0b (id: 15)
```

Por último, ejecutando el siguiente comando se obtienen las interfaces del namespace:

```
[root@infra1 ~]# ip netns exec qdhcp-46acdb05-e28c-4399-ae59-73b9b3a4af0b
    ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
    default qlen 1000
```

```

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: ns-b43bdd81-01@if77: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UP group default qlen 1000
    link/ether fa:16:3e:8b:ca:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.50/24 brd 192.168.1.255 scope global ns-b43bdd81-01
        valid_lft forever preferred_lft forever
    inet 169.254.169.254/16 brd 169.254.255.255 scope global ns-
        b43bdd81-01
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe8b:caee/64 scope link
        valid_lft forever preferred_lft forever

```

5.2.0.4. Análisis de tráfico

Como se describe en el escenario, se analiza el tráfico generado en la comunicación de la instancia 1 ubicada en el nodo de cómputo 1 con la instancia 2 alojada en el nodo de cómputo 2. Se supone que las instancias no tienen cargadas las tablas ARP y los VTEPs no tienen datos en las tablas de forwarding.

Paso 1 La instancia 1 envía el paquete ICMP echo request hacia la instancia 2. Para esto determina que se encuentran en subredes diferentes y por lo tanto busca el siguiente salto en la tabla de ruteo.

```

$ ip r
default via 192.168.1.1 dev eth0
169.254.169.254 via 192.168.1.50 dev eth0
192.168.1.0/24 dev eth0 src 192.168.1.101

```

En la salida se observa que la IP del default gateway es la 192.168.1.1.

Paso 2 La instancia 1 debe obtener la MAC asociada a la IP del router X, como esta no es conocida se dispara el protocolo ARP. Este proceso no se explica dado que es análogo al paso 2 del escenario 1.

Paso 3 Ahora que se conoce la dirección MAC, la instancia 1 retoma el envío del paquete ICMP hacia la instancia 2 a través del router X. El proceso desde que el paquete parte de la instancia 1 hasta el router es análogo al explicado en el paso 3 del escenario 1.

```

→ 5 6.716799 192.168.1.101 192.168.2.102 ICMP 148 Echo (ping) request id=0x7d01, seq=0/0,
← 6 6.718899 192.168.2.102 192.168.1.101 ICMP 148 Echo (ping) reply id=0x7d01, seq=0/0,
▶ Frame 5: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
▶ Ethernet II, Src: RealtekU_be:9c:39 (52:54:00:be:9c:39), Dst: RealtekU_c1:1e:48 (52:54:00:c1:1e:48)
▶ Internet Protocol Version 4, Src: 10.0.31.12, Dst: 10.0.31.11
▶ User Datagram Protocol, Src Port: 51923, Dst Port: 8472
▼ Virtual eXtensible Local Area Network
  ▶ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 19
    Reserved: 0
  ▶ Ethernet II, Src: fa:16:3e:b6:d0:fd (fa:16:3e:b6:d0:fd), Dst: fa:16:3e:99:f9:9f (fa:16:3e:99:f9:9f)
  ▶ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.2.102
  ▶ Internet Control Message Protocol

```

Figura 5.9: Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1

Paso 4 Cuando el paquete llega al router se envía hacia la red tenant-network-2 de acuerdo a la tabla de ruteo del namespace:

```

[root@infra1 ~]# ip netns exec qrouter-11a748c4-9cb7-4d01-864b-
c5e73b635fed ip r
192.168.1.0/24 dev qr-b1d7f4c9-f6 proto kernel scope link src 192.168.1.1
192.168.2.0/24 dev qr-c2b59f24-f8 proto kernel scope link src 192.168.2.1

```

Como se encuentra directamente conectado con el host destino, no necesita ningún salto intermedio.

Paso 5 El router X debe obtener la MAC de la instancia 2, como hasta el momento esta no es conocida se dispara el protocolo ARP. Este proceso no se explica dado que es análogo a los descubrimientos detallados anteriormente.

Paso 6 Ahora que se conoce la dirección MAC, el router X retoma el envío del paquete ICMP hacia la instancia 2. El proceso desde que el paquete parte del router hasta su destino es análogo al explicado en el paso 3 del escenario 1.

```

→ 1 0.000000 192.168.1.101 192.168.2.102 ICMP 148 Echo (ping) request id=0x7d01, seq=0/0,
← 2 0.001359 192.168.2.102 192.168.1.101 ICMP 148 Echo (ping) reply id=0x7d01, seq=0/0,
▶ Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
▶ Ethernet II, Src: RealtekU_c1:1e:48 (52:54:00:c1:1e:48), Dst: RealtekU_aa:e4:85 (52:54:00:aa:e4:85)
▶ Internet Protocol Version 4, Src: 10.0.31.11, Dst: 10.0.31.22
▶ User Datagram Protocol, Src Port: 50692, Dst Port: 8472
▼ Virtual eXtensible Local Area Network
  ▶ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 88
    Reserved: 0
  ▶ Ethernet II, Src: fa:16:3e:4b:35:4b (fa:16:3e:4b:35:4b), Dst: fa:16:3e:d5:38:83 (fa:16:3e:d5:38:83)
  ▶ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.2.102
  ▶ Internet Control Message Protocol

```

Figura 5.10: Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 2

Paso 7 El procedimiento para la respuesta con el paquete ICMP reply es similar al descrito pero en sentido contrario.

Escenario 3

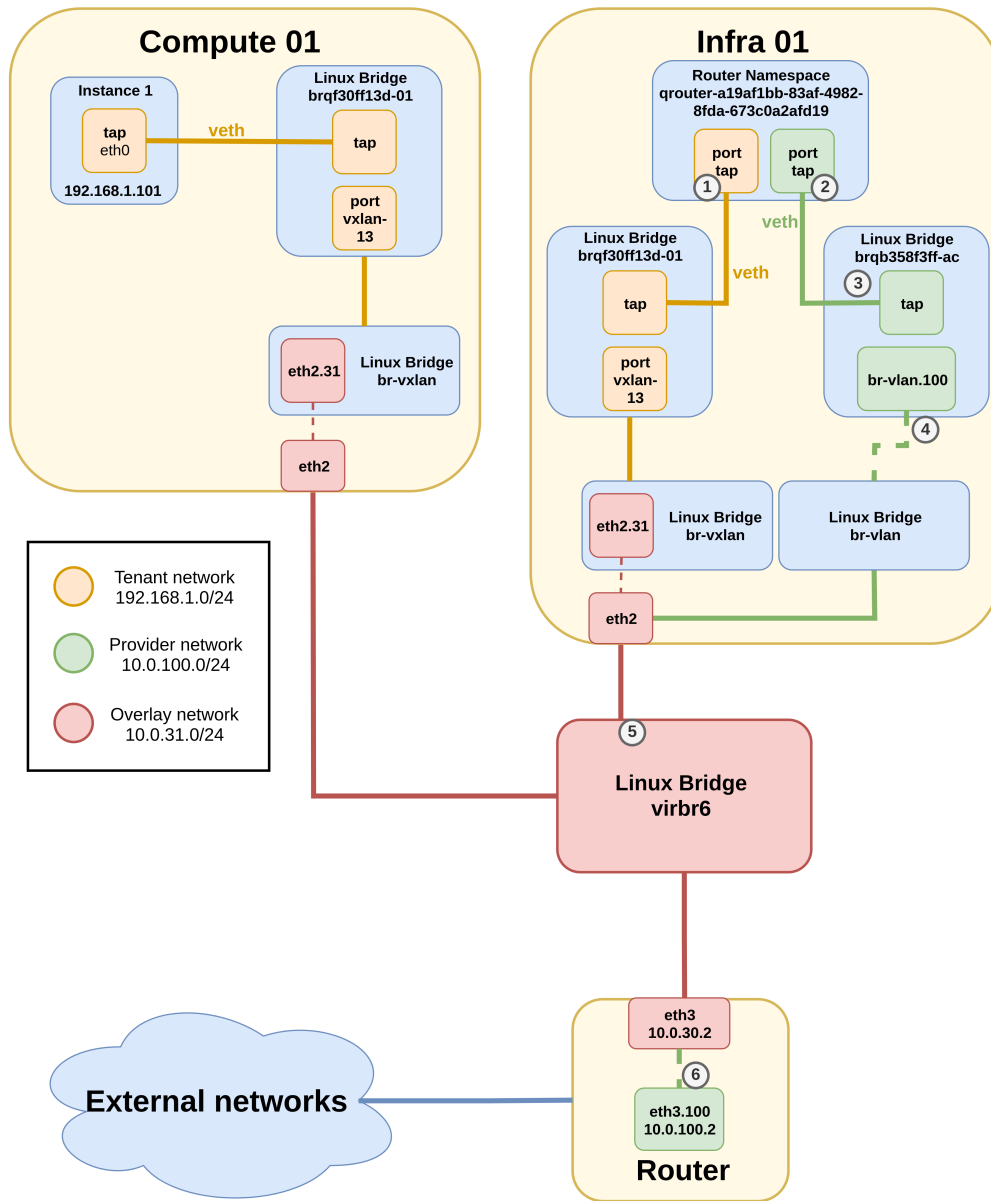


Figura 5.11: Diagrama de arquitectura para el escenario 3 de Linux Bridge

5.2.0.5. Análisis de componentes

A continuación se detallan los componentes virtuales requeridos para instanciar el escenario propuesto. Los componentes relacionados a la red tenant en el nodo de cómputo y en el nodo de infraestructura son análogos a los escenarios 1 y 2. Los componentes a analizar son los empleados para dar soporte a la red provider.

Nodos de infraestructura/red

Las red tenant de Neutron en este escenario es:

```
[root@infra1-utility-container-eebf40f8 ~]# openstack network list --
  project "Escenario 3" -f json
[
  {
    "Subnets": "c624efd4-a8e2-4991-8459-c3a636d04edc",
    "ID": "6773801c-682d-4362-b429-d987b0ecee01",
    "Name": "tenant-network-1"
  }
]
```

La red provider es:

```
[root@infra1-utility-container-eebf40f8 ~]# openstack network list --
  external -f json
[
  {
    "Subnets": "16427b7a-1418-4c9b-bfbb-f3972e022f17",
    "ID": "b358f3ff-ac16-481c-9a93-db6000181ad0",
    "Name": "provider-vlan"
  }
]
```

La comunicación entre redes tenant y provider requiere de un router como intermediario. El siguiente comando permite obtener un listado de los routers.

```
[root@infra1-utility-container-eebf40f8 ~]# openstack router list --
  project "Escenario 3" -f json
[
  {
    "Status": "ACTIVE",
    "Name": "router-Y",
    "Project": "0229ec0838de40f18d03531dca0aa690",
    "State": "UP",
    "HA": false,
    "ID": "9fabdbe0-a667-4d73-b7d0-cbf1aae49709"
  }
]
```

En este escenario el namespace asociado al router tiene dos interfaces, una asociada a la red tenant y otra asociada a la red provider. Como se analizó en el escenario 2, el nombre de las interfaces asociadas a una red tenant tienen el formato `qr-<id_port>`. Por otro lado, el nombre de las interfaces asociadas a una red provider tienen el formato `qg-<id_tap>` siendo el `id_tap` el identificador de la

tap en el nodo de infraestructura asociada a dicha interfaz. El siguiente comando retorna las interfaces del router en cuestión.

```
[root@infra1 ~]# ip netns exec qrouter-9fabdbe0-a667-4d73-b7d0-
cbf1aae49709 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: qr-20a088ee-90@if90: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UP group default qlen 1000
    link/ether fa:16:3e:ec:c6:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.1/24 brd 192.168.1.255 scope global qr-20a088ee-90
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:feec:c6ee/64 scope link
        valid_lft forever preferred_lft forever
4: qg-57e066bc-f9@if133: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP group default qlen 1000
    link/ether fa:16:3e:17:f7:9a brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.100.83/24 brd 10.0.100.255 scope global qg-57e066bc-f9
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe17:f79a/64 scope link
        valid_lft forever preferred_lft forever
```

Como se describió previamente, para instanciar la red tenant se utiliza un Linux Bridge al cual se le asocian dos interfaces: la subinterfaz vxlan correspondiente y una tap vinculada al router.

Por su parte, para instanciar la red provider con el vlan id 100 se requiere de dos estructuras. La primera es la arquitectura subyacente creada por el administrador, la cual Neutron supone que está previamente configurada y se compone de:

- Un Linux Bridge br-vlan dedicado al tráfico tenant.
- Una interfaz física eth2 asociada al bridge anterior.
- Un Linux Bridge tenant que interrelaciona a todos los nodos de cómputo e infraestructura con el router externo.
- Las subredes correspondiente a cada VLAN ID.

La segunda estructura es la creada por Neutron en el nodo de red para comunicar la estructura anterior con el router mencionado previamente. Esta está compuesta por:

- Un Linux Bridge que actúa como switch local y provee conectividad hacia la red provider. A este bridge se le asocian tres puertos de red necesarios para llevar a cabo la comunicación entre la red provider con el servicio de DHCP y el router. Estas interfaces se pueden listar mediante:

```
[root@infra1 ~]# brctl show brqb358f3ff-ac
bridge name      bridge id          STP enabled  interfaces
brqb358f3ff-ac  8000.1ac7c8b21544  no          br-vlan.100
                                          tap264e728a-2b
                                          tap57e066bc-f9
```

- La interfaz `tap57e066bc-f9` se encarga de conectar el router con el bridge.
- Al igual que en las redes tenant se puede configurar el servicio de DHCP que es accesible desde la interfaz `tap264e728a-2b`.
- Dado que la red provider es de tipo VLAN con id 100, Neutron instancia la subinterfaz `br-vlan.100` para poder establecer la comunicación de capa 2 con el resto de la red.

5.2.0.6. Análisis de tráfico

El objetivo de este escenario es analizar el tráfico generado en la comunicación entre la instancia 1 y un host externo al manejo de OpenStack. En este caso se considera al router de la infraestructura física como host externo debido a que si la instancia logra comunicarse con el mismo, entonces tendrá acceso a cualquier host que este alcance.

Paso 1 La instancia 1 envía el paquete ICMP echo request hacia el router físico. Para esto determina que se encuentran en subredes diferentes y por lo tanto busca el siguiente salto en la tabla de ruteo.

```
$ ip r
default via 192.168.1.1 dev eth0
169.254.169.254 via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0 src 192.168.1.101
```

En la salida se observa que la IP del default gateway es la 192.168.1.1.

Paso 2 La instancia 1 debe obtener la MAC asociada a la IP del router Y, como esta no es conocida se dispara el protocolo ARP. Este proceso no se explica dado que es análogo al paso 2 del escenario 1.

Paso 3 Ahora que se conoce la dirección MAC, la instancia 1 retoma el envío del paquete ICMP hacia el router físico a través del router Y. El proceso desde que el paquete parte de la instancia 1 hasta el router Y es análogo al explicado en el paso 3 del escenario 1.

No.	Time	Source	Destination	Protocol	Length	Info
→	1 0.000000	192.168.1.101	10.0.100.2	ICMP	148	Echo (ping) request id=0xb301, seq=0/0,
←	2 0.001023	10.0.100.2	192.168.1.101	ICMP	148	Echo (ping) reply id=0xb301, seq=0/0,


```

▶ Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
▶ Ethernet II, Src: RealtekU_be:9c:39 (52:54:00:be:9c:39), Dst: IPv4mcast_01:01:01 (01:00:5e:01:01:01)
▶ Internet Protocol Version 4, Src: 10.0.31.12, Dst: 239.1.1.1
▶ User Datagram Protocol, Src Port: 60893, Dst Port: 8472
▼ Virtual eXtensible Local Area Network
  ▶ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 69
    Reserved: 0
  ▶ Ethernet II, Src: fa:16:3e:f5:b3:0e (fa:16:3e:f5:b3:0e), Dst: fa:16:3e:ec:c6:ee (fa:16:3e:ec:c6:ee)
  ▶ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 10.0.100.2
  ▶ Internet Control Message Protocol
  
```

Figura 5.12: Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1

Paso 4 Cuando el paquete llega al router Y se envía hacia la red provider-vlan de acuerdo a la tabla de ruteo del namespace:

```

[root@infra1 ~]# ip netns exec qrouter-9fabdbe0-a667-4d73-b7d0-
    cbf1aae49709 ip r
default via 10.0.100.2 dev qg-57e066bc-f9
10.0.100.0/24 dev qg-57e066bc-f9 proto kernel scope link src 10.0.100.83
192.168.1.0/24 dev qr-20a088ee-90 proto kernel scope link src 192.168.1.1
  
```

Como se encuentra directamente conectado con el host destino, no necesita ningún salto intermedio.

Paso 5 El router Y debe obtener la MAC del router físico, como hasta el momento esta no es conocida se dispara el protocolo ARP. A continuación se detalla cómo se lleva a cabo el protocolo de resolución de direcciones entre el router virtual y el físico.

1. El router Y envía una trama ARP request preguntando por la IP 10.0.100.2 (5). Dicha trama es enviada hacia el Linux Bridge asociado a la red provider a través del veth pair (1).
2. El bridge recibe el pedido ARP (2) y lo reenvía a través del puerto br-vlan.100 (3). Este último se encarga de taggear la trama original con el VLAN ID 100.

3. El paquete es recibido por el Linux Bridge br-vlan y reenviado a través de la interfaz eth2. El mismo es transportado mediante la infraestructura física subyacente (4), alcanzando todos los nodos dentro de la VLAN 100.

```

9 10.881119 fa:16:3e:17:f7:9a RealtekU_cc:b7:63 ARP 46 who has 10.0.100.2? Tell 10.0.100.83
10 10.881316 RealtekU_cc:b7:63 fa:16:3e:17:f7:9a ARP 46 10.0.100.2 is at 52:54:00:cc:b7:63
▶ Frame 9: 46 bytes on wire (368 bits), 46 bytes captured (368 bits)
▶ Ethernet II, Src: fa:16:3e:17:f7:9a (fa:16:3e:17:f7:9a), Dst: RealtekU_cc:b7:63 (52:54:00:cc:b7:63)
▼ 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 100
  000. .... = Priority: Best Effort (default) (0)
  ...0 .... = DEI: Ineligible
  .... 0000 0110 0100 = ID: 100
  Type: ARP (0x0806)
▼ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: fa:16:3e:17:f7:9a (fa:16:3e:17:f7:9a)
  Sender IP address: 10.0.100.83
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 10.0.100.2

```

Figura 5.13: Paquete ARP request taggeado con el VLAN ID 100 capturado en la interfaz br-vlan en el nodo de red

4. Cuando el router físico recibe la consulta por su interfaz eth3.100 (5), quita el tag de VLAN y procesa el paquete. Esto genera una respuesta hacia al router Y con un ARP reply directo a su MAC.
5. El procedimiento para la respuesta es similar al descrito pero en sentido contrario.

Paso 6 Ahora que se conoce la dirección MAC, el router Y retoma el envío del paquete ICMP hacia el router físico. Previo al mismo, debe encargarse de realizar un source NAT para permitir que la respuesta alcance la instancia. Esto se realiza utilizando la interfaz qg (10.0.100.100) como la dirección IP de origen.

1. El mensaje es enviado por el veth pair (1) hacia el bridge de la red provider-vlan (2).
2. Luego la trama es reenviada hacia el Linux Bridge br-vlan a través de la subinterfaz br-vlan.100 (3). Este último se encarga de taggear la trama original con el VLAN ID 100.
3. El paquete es recibido por el Linux Bridge br-vlan y reenviado a través de la interfaz eth2 hacia el router físico. El mismo es transportado mediante la infraestructura física subyacente (4), alcanzando el destino.
4. El paquete es recibido por la interfaz eth3.100 del router físico, la cual quita el tag de VLAN y procesa el ICMP echo request.

```

+-----+-----+-----+-----+-----+-----+
| 13 15.428233 | 10.0.100.83 | 10.0.100.2 | ICMP | 102 Echo (ping) request | id=0xb901, seq=0/0,
+-----+-----+-----+-----+-----+-----+
| 14 15.428542 | 10.0.100.2 | 10.0.100.83 | ICMP | 102 Echo (ping) reply | id=0xb901, seq=0/0,
+-----+-----+-----+-----+-----+-----+
|> Frame 13: 102 bytes on wire (816 bits), 102 bytes captured (816 bits)
|> Ethernet II, Src: fa:16:3e:17:f7:9a (fa:16:3e:17:f7:9a), Dst: RealtekU_cc:b7:63 (52:54:00:cc:b7:63)
|> 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 100
|> 000. .... = Priority: Best Effort (default) (0)
|> .... = DEI: Ineligible
|> .... 0000 0110 0100 = ID: 100
|> Type: IPv4 (0x0800)
|> Internet Protocol Version 4, Src: 10.0.100.83, Dst: 10.0.100.2
|> Internet Control Message Protocol

```

Figura 5.14: Paquete ICMP echo request capturado en la interfaz br-vlan del nodo de red

Paso 7 El procedimiento para la respuesta ICMP echo reply es similar al descrito pero en sentido contrario.

Escenario 4

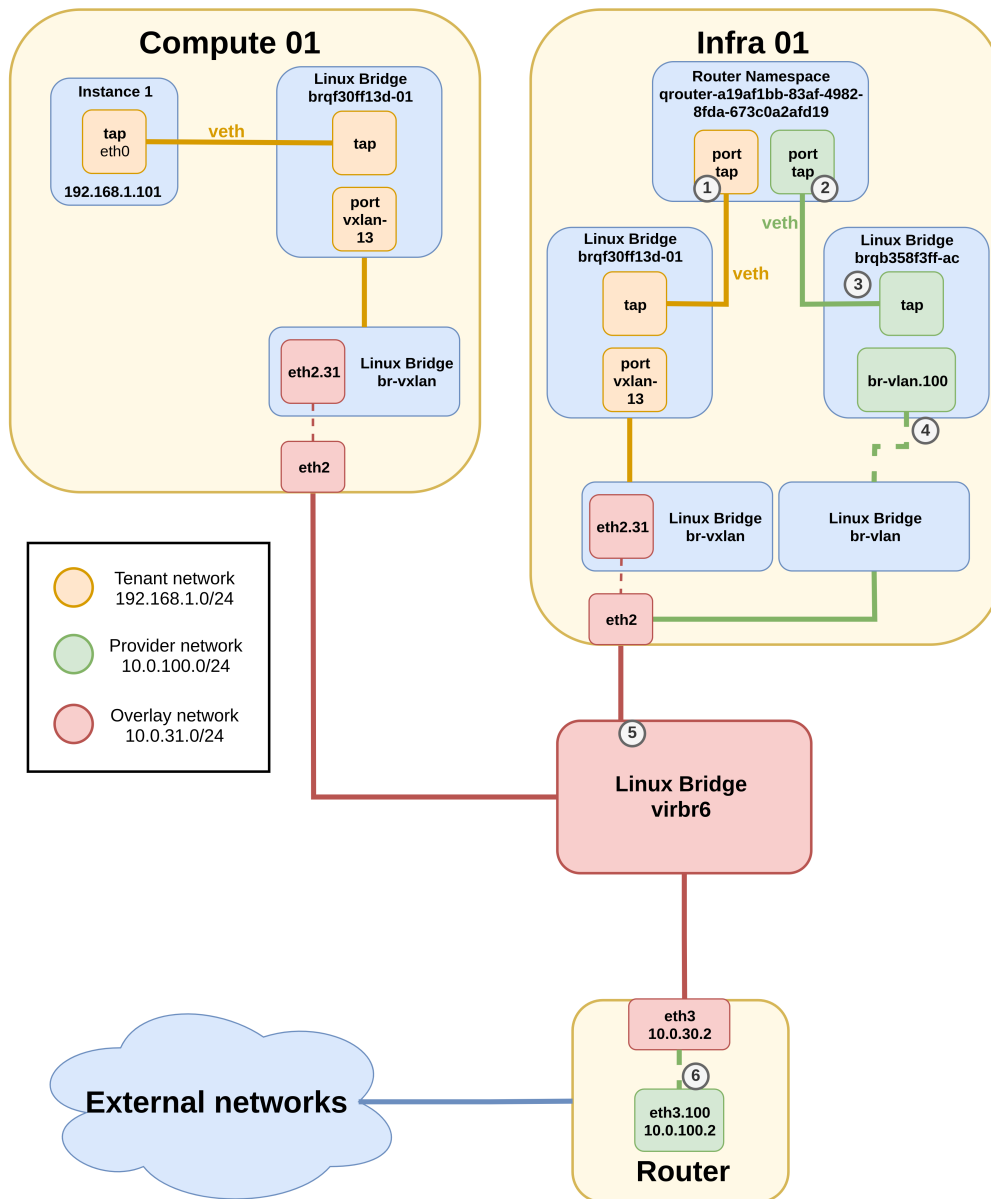


Figura 5.15: Diagrama de arquitectura para el escenario 4 de Linux Bridge

5.2.0.7. Análisis de componentes

Los componentes virtuales requeridos para instanciar el escenario 4 son exactamente los mismos que los definidos en el escenario anterior. La única diferencia existente se da en las interfaces del router Z en el nodo de infraestructura, debido a que debe realizar la correspondencia entre la IP flotante y la fija de la instancia. Al igual que fue mencionado en el componente anterior, se detallan las interfaces del router accediendo al namespace del mismo mediante el siguiente comando:

```

[root@infra1 ~]# ip netns exec qrouter-a19af1bb-83af-4982-8fda-673
c0a2afd19 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: qg-7226b8ea-32@if131: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP group default qlen 1000
    link/ether fa:16:3e:73:85:77 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.100.68/24 brd 10.0.100.255 scope global qg-7226b8ea-32
        valid_lft forever preferred_lft forever
    inet 10.0.100.64/32 brd 10.0.100.64 scope global qg-7226b8ea-32
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe73:8577/64 scope link
        valid_lft forever preferred_lft forever
3: qr-83fa3614-ce@if132: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UP group default qlen 1000
    link/ether fa:16:3e:b0:de:17 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.1/24 brd 192.168.1.255 scope global qr-83fa3614-ce
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:feb0:de17/64 scope link
        valid_lft forever preferred_lft forever

```

En la salida de este comando se observa que la interfaz `qg-7226b8ea-32` dedicada para la red provider, tiene asignadas la IP del router (10.0.100.68) y la flotante de la instancia (10.0.100.64) necesaria para implementar el NAT estático uno a uno con la IP fija de la instancia en la red tenant (192.168.1.101).

5.2.0.8. Análisis de tráfico

El objetivo de este escenario es analizar el tráfico generado en la comunicación desde un host externo al manejo de Openstack hacia la instancia 1. En este caso se considera al router de la infraestructura física como host externo debido a que si este logra comunicarse con la instancia, entonces también podrá hacerlo cualquier host externo que alcance al router.

Paso 1 El router físico envía el paquete ICMP echo request hacia la instancia 1 utilizando la IP flotante, es decir que el destino será el router Z. Para esto determina

que se encuentra directamente conectado con el host destino y por lo tanto no necesita ningún salto intermedio. Con el siguiente comando se observa la tabla de ruteo de dicho router:

```
[root@router ~]# ip r
default via 10.0.40.1 dev eth0 proto static metric 100
10.0.10.0/24 dev eth1 proto kernel scope link src 10.0.10.2 metric 101
10.0.20.0/24 dev eth2 proto kernel scope link src 10.0.20.2 metric 102
10.0.30.0/24 dev eth3 proto kernel scope link src 10.0.30.2 metric 103
10.0.40.0/24 dev eth0 proto kernel scope link src 10.0.40.2 metric 100
10.0.100.0/24 dev eth3.100 proto kernel scope link src 10.0.100.2 metric
400
10.0.101.0/24 dev eth3.101 proto kernel scope link src 10.0.101.2 metric
401
```

Paso 2 El router físico debe obtener la MAC del router Z, como hasta el momento esta no es conocida se dispara el protocolo ARP. Este procedimiento no se explica dado a que es análogo, pero en sentido inverso, al detallado en el paso 5 del escenario 3.

Paso 3 Ahora que se conoce la dirección MAC, el router físico retoma el envío del paquete ICMP hacia el router Z.

1. El mensaje es enviado por la subinterfaz eth3.100 (6) en donde se etiqueta el mismo con el VLAN ID 100 hacia la infraestructura física subyacente (5).

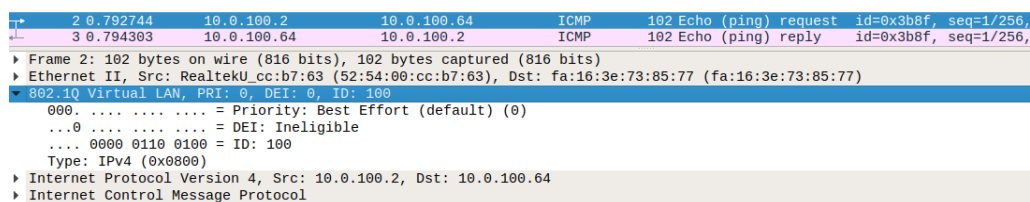


Figura 5.16: Paquete ICMP echo request etiquetado con el VLAN ID 100 capturado en la interfaz eth3 del router físico

2. El paquete es recibido por la interfaz eth2 del nodo de red y se reenvía hacia el Linux Bridge br-vlan llegando a la subinterfaz br-vlan.100 (4) asociada a la red provider. Esta interfaz se encarga de remover la etiqueta con el VLAN ID 100.
3. A través de dicha subinterfaz se alcanza el Linux Bridge asociado a la red provider. Luego la trama es enviada hacia la interfaz qg del namespace del router de Neutron mediante el veth pair (3), alcanzando el destino (2).

1	0.000000	10.0.100.2	10.0.100.64	ICMP	98 Echo (ping) request	id=0x38d8, seq=1/256,
2	0.006371	10.0.100.64	10.0.100.2	ICMP	98 Echo (ping) reply	id=0x38d8, seq=1/256,

```

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▶ Ethernet II, Src: RealtekU_cc:b7:63 (52:54:00:cc:b7:63), Dst: fa:16:3e:73:85:77 (fa:16:3e:73:85:77)
▶ Internet Protocol Version 4, Src: 10.0.100.2, Dst: 10.0.100.64
▶ Internet Control Message Protocol

```

Figura 5.17: Paquete ICMP echo request capturado en la interfaz qg del router de Neutron

Paso 4 En el router el servicio de iptables realiza el DNAT sobre el paquete utilizando la IP de la instancia como dirección destino. Estos mapeos se pueden ver en detalle con el comando:

```
qrouter-a19af1bb-83af-4982-8fda-673c0a2afd19 iptables -t nat -L
```

Paso 5 El router envía el nuevo paquete ICMP hacia la instancia 1. El proceso desde que el paquete parte del router hasta su destino es análogo al explicado en el paso 3 del escenario 1.

1	0.000000	10.0.100.2	192.168.1.101	ICMP	98 Echo (ping) request	id=0x38d8, seq=1/256,
2	0.005970	192.168.1.101	10.0.100.2	ICMP	98 Echo (ping) reply	id=0x38d8, seq=1/256,

```

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▶ Ethernet II, Src: fa:16:3e:b0:de:17 (fa:16:3e:b0:de:17), Dst: fa:16:3e:a4:b3:c3 (fa:16:3e:a4:b3:c3)
▶ Internet Protocol Version 4, Src: 10.0.100.2, Dst: 192.168.1.101
▶ Internet Control Message Protocol

```

Figura 5.18: Paquete ICMP echo request capturado en la interfaz qr del router de Neutron

Paso 6 El procedimiento para la respuesta ICMP echo reply es similar al descrito pero en sentido contrario, con la salvedad de que en el momento en que el router Z reenvía la respuesta hacia el router físico, deberá realizar un source NAT cambiando la IP de origen de la instancia 1 por la de la interfaz qg.

5.2.1. Ventajas y desventajas

A continuación se presenta un listado de aspectos positivos y negativos del plugin Linux Bridge que son relevantes a la hora de determinar que implementación utilizar.

Ventajas:

- Gran estabilidad y fiabilidad en su implementación.
- Facilidad para diagnosticar problemas utilizando herramientas estándar (p. ej. tcpdump, ifconfig) gracias a que se implementa mediante interfaces Linux.
- Gran soporte por parte de la comunidad junto con amplia documentación de su funcionamiento.

- Amplia madurez del producto garantizada por sus años de existencia.
- Mantiene soporte para redes de tunelización de tipo VXLAN.
- Sus funcionalidades se encuentran implementadas en base a módulos de Linux, los cuales tienen soporte a largo plazo de toda la comunidad proporcionando estabilidad.
- Es compatible con el servicio de QoS de Neutron.
- No requiere herramientas ni paquetes adicionales a los existentes en un SO Linux.
- Soporta el protocolo VRRP para brindar una alta disponibilidad de los routers.

Desventajas:

- No tiene soporte para DVR.
- No es compatible con redes de tipo GRE.

5.3. Open vSwitch

Escenario 1

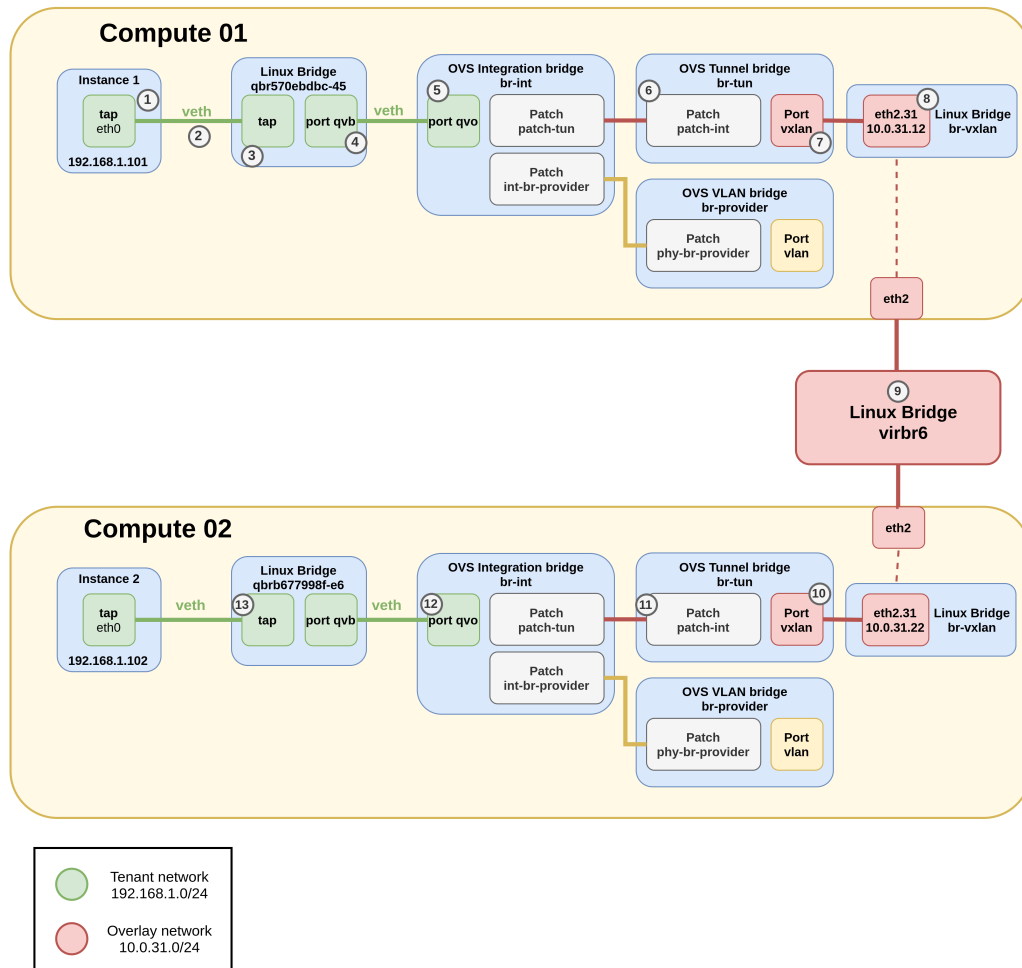


Figura 5.19: Diagrama de arquitectura para el escenario 1 de Open vSwitch

5.3.0.1. Análisis de componentes

A continuación se detallan los componentes ubicados en los nodos de cómputo para dar soporte a este escenario. Dado que los componentes son análogos en ambos nodos se describen solamente los del nodo de cómputo 1.

Nodos de cómputo

Al igual que con el driver Linux Bridge, el hipervisor KVM se encarga de crear una nueva máquina virtual para cada nueva instancia definida.

Debido a la utilización de los security group en forma híbrida, las instancias no se conectan directamente al br-int sino que tienen como intermediario un Linux Bridge llamado brq-xx. Este último se conecta al integration bridge mediante un veth pair, donde la interfaz asociada al brq-xx se llama qvb-xx y la asociada al br-int se llama

qvo-xx. A continuación se muestra la salida de algunos comandos para apreciar lo mencionado: Listar las instancias del escenario 1:

```
[root@infra1-utility-container-161eebae ~]# openstack server list --
  project "Escenario 1" -f json
[
  {
    "Status": "ACTIVE",
    "Name": "instance-2",
    "Image": "",
    "ID": "045e7faf-852b-449d-9cec-4bf1f462fa6d",
    "Flavor": "small",
    "Networks": "tenant-network-1=192.168.1.102"
  },
  {
    "Status": "ACTIVE",
    "Name": "instance-1",
    "Image": "",
    "ID": "dbda5442-4fb7-4cf6-93a5-be85ede7d8d7",
    "Flavor": "small",
    "Networks": "tenant-network-1=192.168.1.101"
  }
]
```

Con los siguientes comandos se listan y detallan las redes del Escenario 1:

```
[root@infra1-utility-container-161eebae ~]# openstack network list --
  project "Escenario 1" -f json
[
  {
    "Subnets": "afc6b545-0507-4392-9618-290af027f1cc",
    "ID": "4974712b-c7ce-4a89-a4f1-145bc64d4d2b",
    "Name": "tenant-network-1"
  }
]
```

```
[root@infra1-utility-container-161eebae ~]# openstack network show 4974712
  b-c7ce-4a89-a4f1-145bc64d4d2b -f json
{
  "provider:physical_network": null,
  "ipv6_address_scope": null,
  "dns_domain": null,
  "is_vlan_transparent": null,
```

```

"revision_number": 2,
"port_security_enabled": true,
"provider:network_type": "vxlan",
"id": "4974712b-c7ce-4a89-a4f1-145bc64d4d2b",
"router:external": "Internal",
"availability_zone_hints": "",
"availability_zones": "nova",
"segments": null,
"name": "tenant-network-1",
"location": {
  "project": {
    "domain_id": null,
    "id": "65fc6646a9e943d491b24a71b8d8f19f",
    "name": null,
    "domain_name": null
  },
  "zone": null,
  "region_name": "RegionOne",
  "cloud": ""
},
"ipv4_address_scope": null,
"shared": false,
"project_id": "65fc6646a9e943d491b24a71b8d8f19f",
"status": "ACTIVE",
"subnets": "afc6b545-0507-4392-9618-290af027f1cc",
"description": "",
"tags": "",
"updated_at": "2020-01-04T12:15:01Z",
"is_default": null,
"provider:segmentation_id": 19,
"qos_policy_id": null,
"admin_state_up": "UP",
"created_at": "2020-01-04T12:15:00Z",
"mtu": 1450
}

```

Entre los detalles de la red tenant-network-1 podemos destacar el tipo de red que es VXLAN y el VNI el cual es 19. Con el ID de la instance-1 se obtienen más detalles sobre la misma:

```

[root@infra1-utility-container-161eebae ~]# openstack server show dbda5442
-4fb7-4cf6-93a5-be85ede7d8d7 -f json

```

```

{
  "OS-EXT-STS:task_state": null,
  "addresses": "tenant-network-1=192.168.1.101",
  "image": "",
  "OS-EXT-STS:vm_state": "active",
  "OS-EXT-SRV-ATTR:instance_name": "instance-0000000b",
  "OS-SRV-USG:launched_at": "2020-01-04T18:10:08.000000",
  "flavor": "small (ba063391-5431-456e-ad92-a7e9fe1e8c82)",
  "id": "dbda5442-4fb7-4cf6-93a5-be85ede7d8d7",
  "security_groups": "name='default'",
  "volumes_attached": "id='74748f6c-ca5a-4f86-a683-f02a61533b53'",
  "user_id": "d844634b4f8942cc85e2cf88c8d1f096",
  "OS-DCF:diskConfig": "AUTO",
  "accessIPv4": "",
  "accessIPv6": "",
  "progress": 0,
  "OS-EXT-STS:power_state": "Running",
  "OS-EXT-AZ:availability_zone": "nova",
  "config_drive": "",
  "status": "ACTIVE",
  "updated": "2020-01-04T18:10:07Z",
  "hostId": "b91cc2c2ec44a5b4231f42a8de608b0c26430838aa0a7f49ce32aa62",
  "OS-EXT-SRV-ATTR:host": "compute1",
  "OS-SRV-USG:terminated_at": null,
  "key_name": null,
  "properties": "",
  "project_id": "65fc6646a9e943d491b24a71b8d8f19f",
  "OS-EXT-SRV-ATTR:hypervisor_hostname": "compute1.openstack.local",
  "name": "instance-1",
  "created": "2020-01-04T18:09:54Z"
}

```

A partir de esta salida se puede ver que la instancia está alojada en el nodo de cómputo 1 y que el nombre de la misma en el hipervisor es `instance-0000000b`. Con este último dato es posible obtener el nombre de la interfaz tap creada por KVM para dicha instancia:

```

[root@compute1 ~]# virsh domiflist instance-0000000b
Interface      Type      Source          Model  MAC
-----
tap570ebdbc-45 bridge qbr570ebdbc-45 virtio fa:16:3e:be:b9:2e

```

Con esta salida se obtiene el identificador “xx” mencionado previamente, que se repite en los veth pairs y el bridge creados para la instancia. En este caso es qbr570ebdbc-45.

Con el siguiente comando se puede ver que efectivamente el bridge tiene asociadas dos interfaces, la tap encargada de conectarse con la instancia y la qvb encargada de conectarse con el br-int.

```
[root@compute1 ~]# brctl show qbr570ebdbc-45
bridge name      bridge id          STP enabled  interfaces
qbr570ebdbc-45  8000.e6e8fe347d4a  no          qvb570ebdbc-45
                                     tap570ebdbc-45
```

Con el siguiente comando se pueden ver las interfaces conectadas a todos los switches virtuales de OVS.

```
[root@compute1 ~]# ovs-vsctl show
eabf27d2-60fe-4b0c-9fb0-1adfab96ffa3
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port int-br-provider
            Interface int-br-provider
                type: patch
                options: {peer=phy-br-provider}
        Port "qvo570ebdbc-45"
            tag: 6
            Interface "qvo570ebdbc-45"
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port br-int
            Interface br-int
                type: internal
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port patch-int
```

```

        Interface patch-int
            type: patch
            options: {peer=patch-tun}
    Port "vxlan-0a001f16"
        Interface "vxlan-0a001f16"
            type: vxlan
            options: {df_default="true", egress_pkt_mark
                ="0", in_key=flow, local_ip="10.0.31.12",
                out_key=flow, remote_ip="10.0.31.22"}
    Port "vxlan-0a001f0b"
        Interface "vxlan-0a001f0b"
            type: vxlan
            options: {df_default="true", egress_pkt_mark
                ="0", in_key=flow, local_ip="10.0.31.12",
                out_key=flow, remote_ip="10.0.31.11"}
    Port br-tun
        Interface br-tun
            type: internal
    Bridge br-provider
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
    Port phy-br-provider
        Interface phy-br-provider
            type: patch
            options: {peer=int-br-provider}
    Port br-provider
        Interface br-provider
            type: internal
    Port br-vlan
        Interface br-vlan
    ovs_version: "2.11.0"

```

A las interfaces qvo se les asigna un tag interno que luego en el bridge provider o túnel será utilizado para realizar el mapeo que corresponda a el tipo de red seleccionado. La interfaz que brinda conectividad a la instancia 1 es qvo570ebdbc-45 con el tag interno 6.

Como la red tenant creada es de tipo VXLAN este escenario utiliza el bridge dedicado para las redes de overlay llamado br-tun, conectado al integration bridge mediante el patch port "patch-int". En la entrada de options se indica el nombre del patch port par en el otro bridge. Adicionalmente este bridge tiene dos puertos

utilizados para crear la red mesh de VXLAN entre el nodo de cómputo en cuestión, el otro nodo de cómputo y el nodo de red. Estos son `vxlan-0a001f16` y `vxlan-0a001f0b`.

5.3.0.2. Análisis de tráfico

En este primer ejemplo se analiza el tráfico generado cuando la instancia 1 intenta comunicarse con la instancia 2, asumiendo que las mismas aún no conocen las direcciones MAC destino.

Paso 1 La instancia 1 envía el paquete ICMP echo request hacia la instancia 2. Para esto determina que se encuentra directamente conectada con el host destino y por lo tanto no necesita ningún salto intermedio.

Paso 2 La instancia 1 debe obtener la MAC de la instancia 2, como esta no es conocida se dispara el protocolo ARP. A continuación se detalla cómo se lleva a cabo el protocolo de resolución de direcciones entre las instancias virtuales.

1. La instancia 1 envía una trama ARP request preguntando por la IP 192.168.1.102 (1). Dicha trama es enviada hacia el Linux Bridge asociado a través del veth pair (2).

```

1 0.000000 fa:16:3e:be:b9:2e Broadcast ARP 42 Who has 192.168.1.102? Tell 192.168.1.101
2 0.004320 fa:16:3e:92:57:e4 fa:16:3e:be:b9:2e ARP 42 192.168.1.102 is at fa:16:3e:92:57:e4

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  Source: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Type: ARP (0x0806)
Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Sender IP address: 192.168.1.101
  Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.1.102

```

Figura 5.20: Paquete ARP request capturado en la interfaz eth0 de la instancia 1

2. El bridge recibe el pedido ARP (3) y lo reenvía a través del puerto qvb (4) hacia el integration bridge de OVS.
3. Este último recibe el pedido por el puerto qvo (5), el cual etiqueta el paquete con el VLAN ID 6.
4. Luego el bridge lo procesa siguiendo las reglas de OpenFlow, mostradas a continuación. A efectos de mejorar la legibilidad de las mismas se eliminan los campos: `duration`, `cookies`, `n_packets` y `n_bytes` (en las siguientes salidas del mismo comando se realizará el mismo filtro).

```

[root@compute1 ~]# ovs-ofctl dump-flows br-int --rsort
1 table=0,priority=65535,vlan_tci=0x0fff/0x1fff actions=drop
2 table=0,priority=10,icmp6,in_port="qvo570ebdbc-45",icmp_type=136
  actions=resubmit(,24)
3 table=0,priority=10,arp,in_port="qvo570ebdbc-45" actions=resubmit
  (,24)
4 table=0,priority=9,in_port="qvo570ebdbc-45" actions=resubmit(,25)
5 table=60,priority=3 actions=NORMAL
6 table=0,priority=2,in_port="int-br-provider" actions=drop
7 table=24,priority=2,icmp6,in_port="qvo570ebdbc-45",icmp_type=136,
  nd_target=fe80::f816:3eff:febe:b92e actions=resubmit(,60)
8 table=24,priority=2,arp,in_port="qvo570ebdbc-45",arp_spa
  =192.168.1.101 actions=resubmit(,25)
9 table=25,priority=2,in_port="qvo570ebdbc-45",dl_src=fa:16:3e:be:b9:2e
  actions=resubmit(,60)
10 table=0,priority=0 actions=resubmit(,60)
11 table=23,priority=0 actions=drop
12 table=24,priority=0 actions=drop

```

La primer regla que coincide con el paquete recibido en el `br-int` es la número 3 la cual establece que los paquetes ARP recibidos en el puerto `qvo570ebdbc-45` se envíen a la tabla 24. Luego las reglas 8 y 9 son utilizadas para evitar un spoofing de ARP, en donde se verifica que la dirección IP y MAC de origen del paquete ARP, coincidan con las de la instancia 1. Finalmente el paquete es procesado por la tabla 60 en la regla 5 donde se toma la acción NORMAL indicando a Open vSwitch que debe actual como un switch en modo de aprendizaje generando que el tráfico sea enviado a todos los puertos con la excepción del entrante.

5. Luego del procesamiento el paquete ARP request llega al bridge `br-tun` mediante el patch port `patch-int` (6).
6. En dicho bridge se aplican nuevamente reglas de OpenFlow, listadas a continuación.

```

[root@compute1 ~]# ovs-ofctl dump-flows br-tun --rsort
1 table=0,priority=1,in_port="patch-int" actions=resubmit(,2)
2 table=0,priority=1,in_port="vxlan-0a001f0b" actions=resubmit(,4)
3 table=0,priority=1,in_port="vxlan-0a001f16" actions=resubmit(,4)
4 table=4,priority=1,tun_id=0x13 actions=mod_vlan_vid:6,resubmit(,10)
5 table=10,priority=1 actions=learn(table=20,hard_timeout=300,priority
  =1,cookie=0xb8003f26cd0d7430,NXM_OF_VLAN_TCI[0..11]),

```

```

NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:
NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:OXM_OF_IN_PORT()),output
:"patch-int"
6 table=22,priority=1,dl_vlan=6 actions=strip_vlan,load:0x13->
NXM_NX_TUN_ID[],output:"vxlan-0a001f0b",output:"vxlan-0a001f16"
7 table=0,priority=0 actions=drop
8 table=2,priority=0,dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions
=resubmit(,20)
9 table=2,priority=0,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions
=resubmit(,22)
10 table=3,priority=0 actions=drop
11 table=4,priority=0 actions=drop
12 table=6,priority=0 actions=drop
13 table=20,priority=0 actions=resubmit(,22)
14 table=22,priority=0 actions=drop

```

La primer regla que coincide es la 1 en donde se envía el procesamiento a la tabla 2. La regla 8 no coincide debido a que no se conoce la MAC destino, entrando en la regla 9 donde se envía a la tabla 22. Luego en la regla 6 se verifica que coincida el VLAN ID del paquete con el asignado al manejo interno (`dl_vlan=6`). Luego se elimina el VLAN ID (`strip_vlan`), y se agrega el VNI 19 en este caso (`load:0x13->NXM_NX_TUN_ID[]`), para finalmente enviar el paquete por todos los puertos vxlan (7) dado que no conoce que VTEP alberga a la instancia por la que se está consultando.

- De esta forma el paquete ARP alcanza la infraestructura subyacente que brinda soporte al tráfico VXLAN. En primer lugar pasa por el Linux Bridge `br-vxlan`, el cual envía el paquete a la subinterfaz `eth2.31` (8).

```

1 0.000000 fa:16:3e:be:b9:2e Broadcast ARP 92 Who has 192.168.1.102? Tell 192.168.1.101
3 0.003627 fa:16:3e:92:57:e4 fa:16:3e:be:b9:2e ARP 92 192.168.1.102 is at fa:16:3e:92:57:e4

Frame 1: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)
Ethernet II, Src: RealtekU_1b:ff:38 (52:54:00:1b:ff:38), Dst: RealtekU_6f:96:fe (52:54:00:6f:96:fe)
Internet Protocol Version 4, Src: 10.0.31.12, Dst: 10.0.31.11
User Datagram Protocol, Src Port: 36660, Dst Port: 4789
Virtual eXtensible Local Area Network
  Flags: 0x0800, VXLAN Network ID (VNI)
  Group Policy ID: 0
  VXLAN Network Identifier (VNI): 19
  Reserved: 0
Ethernet II, Src: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  Source: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Type: ARP (0x0806)
Address Resolution Protocol (request)

```

Figura 5.21: ARP request encapsulado en VXLAN capturado en la interfaz `br-vxlan` del nodo de cómputo 1

- En la subinterfaz el paquete es etiquetado con el tag 31 y enviado hacia el bridge `virbr6` (9) (infraestructura física) a través de `eth2`.

9. Al llegar al nodo de cómputo 2 se realiza el proceso inverso de los puntos 7 y 8. Alcanzando así la solicitud ARP al bridge br-tun (10) del nodo de cómputo 2.
10. En dicho bridge se aplican nuevamente reglas de Open Flow, listadas a continuación.

```
[root@compute2 ~]# ovs-ofctl dump-flows br-tun --rsort
1 table=0,priority=1,in_port="patch-int" actions=resubmit(,2)
2 table=0,priority=1,in_port="vxlan-0a001f0b" actions=resubmit(,4)
3 table=0,priority=1,in_port="vxlan-0a001f0c" actions=resubmit(,4)
4 table=4,priority=1,tun_id=0x13 actions=mod_vlan_vid:3,resubmit(,10)
5 table=10,priority=1 actions=learn(table=20,hard_timeout=300,priority
  =1,cookie=0x3f05eaa5b010cd3f,NXM_OF_VLAN_TCI[0..11],
  NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:
  NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:OXM_OF_IN_PORT[]),output
  : "patch-int"
6 table=22,priority=1,dl_vlan=3 actions=strip_vlan,load:0x13->
  NXM_NX_TUN_ID[],output:"vxlan-0a001f0b",output:"vxlan-0a001f0c"
7 table=0,priority=0 actions=drop
8 table=2,priority=0,dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions
  =resubmit(,20)
9 table=2,priority=0,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions
  =resubmit(,22)
10 table=3,priority=0 actions=drop
11 table=4,priority=0 actions=drop
12 table=6,priority=0 actions=drop
13 table=20,priority=0 actions=resubmit(,22)
14 table=22,priority=0 actions=drop
```

Con el comando `ovs-vsctl show` se puede observar que el puerto “vxlan-0a001f0c” se corresponde con el nodo de cómputo 1, por lo tanto la primer regla que coincide es la 3, en donde se envía a la tabla 4. En la regla 4 se verifica que el VNI del paquete sea 19, se modifica el VLAN ID al asignado por OVS para esta red tenant en el nodo de cómputo 2 y se envía a la tabla 10. El tag interno asignado a una red virtual puede ser diferente en cada nodo físico, en este caso el tag es 3. La regla 5 a partir del paquete recibido crea una nueva regla, en donde las precondiciones serán que el VLAN ID coincida con el tag interno asignado (`NXM_OF_VLAN_TCI[0..11]`) y la MAC destino sea igual a la MAC del paquete recibido (`NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[]`) y las acciones serán cargar el VLAN ID con 0 (`load:0->NXM_OF_VLAN_TCI[]`) y el VNI de la red virtual (`load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[]`) además

de enviar el paquete por el puerto correspondiente (`output:OXM_OF_IN_PORT[]`). Luego de crear la regla se envía el paquete hacia el br-int por el puerto “patch-int”

11. Nuevamente en el bridge br-int se procesa el paquete de acuerdo a las reglas de OpenFlow listadas a continuación.

```
[root@compute2 ~]# ovs-ofctl dump-flows br-int --rsort
1 table=0,priority=65535,vlan_tci=0x0fff/0x1fff actions=drop
2 table=0,priority=10,icmp6,in_port="qvob677998f-e6",icmp_type=136
  actions=resubmit(,24)
3 table=0,priority=10,arp,in_port="qvob677998f-e6" actions=resubmit
  (,24)
4 table=0,priority=9,in_port="qvob677998f-e6" actions=resubmit(,25)
5 table=60,priority=3 actions=NORMAL
6 table=0,priority=2,in_port="int-br-provider" actions=drop
7 table=24,priority=2,icmp6,in_port="qvob677998f-e6",icmp_type=136,
  nd_target=fe80::f816:3eff:fe92:57e4 actions=resubmit(,60)
8 table=24,priority=2,arp,in_port="qvob677998f-e6",arp_spa
  =192.168.1.102 actions=resubmit(,25)
9 table=25,priority=2,in_port="qvob677998f-e6",dl_src=fa:16:3e:92:57:e4
  actions=resubmit(,60)
10 table=0,priority=0 actions=resubmit(,60)
11 table=23,priority=0 actions=drop
12 table=24,priority=0 actions=drop
```

La primera regla que coincide es la 10 en donde envía el procesamiento a la tabla 60. En la regla 5 como no tiene ninguna precondition se procesa ejecutando la acción NORMAL. De esta forma se envía por todos los puertos qvo, en particular por el puerto “qvob677998f-e6” (12) hacia el Linux Bridge asociado a la instancia 2. Previo al reenvío, el puerto qvo se encarga de quitar el tag de VLAN 3.

12. En el Linux Bridge se aplican las reglas del grupo de seguridad y se envía hacia la instancia 2 mediante el veth pair (13).
13. La instancia 2 recibirá la trama, y al verificar que se está consultando por su IP responderá a la instancia 1 con un ARP reply directo a su MAC. Esta respuesta es enviada hacia el Linux Bridge asociado a través del veth pair (13).
14. El bridge recibe el pedido ARP reply y lo reenvía a través del puerto qvb hacia el integration bridge de OVS.

15. Este último recibe el pedido por el puerto qvo (12), el cual etiqueta el paquete con el VLAN ID 3.
16. Luego el bridge lo procesa siguiendo las mismas reglas de OpenFlow del paso 11. El procesamiento es análogo al descrito en el paso 4, realizando una verificación anti spoofing y reenviando el paquete hacia el `br-tun`.
17. El `br-tun` recibe el paquete por el `patch-int` y lo procesa con las siguientes reglas de OpenFlow, incluyendo la sexta regla aprendida en el paso 10.

```
[root@compute2 ~]# ovs-ofctl dump-flows br-tun --rsort
1 table=0,priority=1,in_port="patch-int" actions=resubmit(,2)
2 table=0,priority=1,in_port="vxlan-0a001f0b" actions=resubmit(,4)
3 table=0,priority=1,in_port="vxlan-0a001f0c" actions=resubmit(,4)
4 table=4,priority=1,tun_id=0x13 actions=mod_vlan_vid:3,resubmit(,10)
5 table=10,priority=1 actions=learn(table=20,hard_timeout=300,priority
  =1,cookie=0x3f05eaa5b010cd3f,NXM_OF_VLAN_TCI[0..11],
  NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:
  NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:OXM_OF_IN_PORT[]),output
  : "patch-int"
6 table=20,hard_timeout=300, priority=1,vlan_tci=0x0003/0x0fff,dl_dst=
  fa:16:3e:be:b9:2e actions=load:0->NXM_OF_VLAN_TCI[],load:0x13->
  NXM_NX_TUN_ID[],output:"vxlan-0a001f0c"
7 table=22,priority=1,dl_vlan=3 actions=strip_vlan,load:0x13->
  NXM_NX_TUN_ID[],output:"vxlan-0a001f0b",output:"vxlan-0a001f0c"
8 table=0,priority=0 actions=drop
9 table=2,priority=0,dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions
  =resubmit(,20)
10 table=2,priority=0,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions
  =resubmit(,22)
11 table=3,priority=0 actions=drop
12 table=4,priority=0 actions=drop
13 table=6,priority=0 actions=drop
14 table=20,priority=0 actions=resubmit(,22)
15 table=22,priority=0 actions=drop
```

De esta forma la regla 1 es la primera que coincide y se envía a la tabla 2. En este caso como la MAC destino no es de broadcast la regla 9 procesa y envía a la tabla 20. Finalmente la nueva regla se encarga en primer lugar de verificar que el VLAN ID coincida con el tag interno (en este caso 3) y la MAC destino se corresponda con la asignada a la instancia 1, para luego modificar

el VLAN ID, el VNI y enviar el paquete por el puerto vxlan asociado al nodo de cómputo 1.

- El envío del paquete a través de la infraestructura física hasta el `br-tun` del nodo de cómputo 1 es análogo a lo detallado en los pasos 7, 8 y 9.

```

1 0.000000 fa:16:3e:be:b9:2e Broadcast ARP 92 Who has 192.168.1.102? Tell 192.168.1.101
3 0.003627 fa:16:3e:92:57:e4 fa:16:3e:be:b9:2e ARP 92 192.168.1.102 is at fa:16:3e:92:57:e4

Frame 3: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)
Ethernet II, Src: RealtekU_5b:6a:66 (52:54:00:5b:6a:66), Dst: RealtekU_1b:ff:38 (52:54:00:1b:ff:38)
Internet Protocol Version 4, Src: 10.0.31.22, Dst: 10.0.31.12
User Datagram Protocol, Src Port: 44565, Dst Port: 4789
Virtual eXtensible Local Area Network
  Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 19
    Reserved: 0
Ethernet II, Src: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4), Dst: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Destination: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Source: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4)
  Type: ARP (0x0806)
Address Resolution Protocol (reply)

```

Figura 5.22: ARP reply encapsulado en VXLAN capturado en la interfaz `br-vxlan` del nodo de cómputo 1

- Al alcanzar al bridge `br-tun` el paquete es procesado por las reglas de OpenFlow de forma análoga al paso 10. Es aquí donde el bridge aprende la siguiente regla y se reenvía el paquete al bridge de integración. `table=20,hard_timeout=300,priority=1,vlan_tci=0x0006/0x0fff,dl_dst=fa:16:3e:92:57:e4actions=load:0->NXM_OF_VLAN_TCI[],load:0x13->NXM_NX_TUN_ID[],output:"vxlan-0a001f16"`
- Ya en el `br-int` el recorrido para alcanzar la instancia 1 es análogo a los pasos 11 y 12.

```

1 0.000000 fa:16:3e:be:b9:2e Broadcast ARP 42 Who has 192.168.1.102? Tell 192.168.1.101
2 0.004320 fa:16:3e:92:57:e4 fa:16:3e:be:b9:2e ARP 42 192.168.1.102 is at fa:16:3e:92:57:e4

Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
Ethernet II, Src: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4), Dst: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Destination: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Source: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4)
  Type: ARP (0x0806)
Address Resolution Protocol (reply)
Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: reply (2)
Sender MAC address: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4)
Sender IP address: 192.168.1.102
Target MAC address: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
Target IP address: 192.168.1.101

```

Figura 5.23: Paquete ARP reply capturado en la interfaz `eth0` de la instancia 1

- De ahora en más, mientras se mantengan actualizadas las tablas de resolución ARP de las instancias y las reglas OpenFlow, la comunicación entre ambas máquinas virtuales será siempre en forma directa, análogamente a lo detallado para el ARP reply.

Paso 3 Ahora que se conoce la dirección MAC, la instancia 1 retoma el envío del paquete ICMP hacia la instancia 2.

1. El mensaje es enviado por el veth pair (2) hacia el Linux Bridge.

```

3 0.005733      192.168.1.101      192.168.1.102      ICMP      98 Echo (ping) request  id=0xde01, seq=0/0, ttl=64 (reply in 4)
4 0.009934      192.168.1.102      192.168.1.101      ICMP      98 Echo (ping) reply   id=0xde01, seq=0/0, ttl=64 (request in 3)

Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0
Ethernet II, Src: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e), Dst: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4)
  Destination: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4)
  Source: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.102
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 64
  Identification: 0xd976 (55670)
  Flags: 0x4000, Don't fragment
  Time to live: 64
  Protocol: ICMP (1)
  Header checksum: 0xdd16 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.1.101
  Destination: 192.168.1.102
Internet Control Message Protocol

```

Figura 5.24: Paquete ICMP request capturado en la interfaz eth0 de la instancia 1

2. En el bridge se aplican las reglas del grupo de seguridad reenvía el paquete a través del puerto qvb (4) hacia el integration bridge de OVS.
3. Este último recibe el pedido por el puerto qvo (5), el cual etiqueta el paquete con el VLAN ID 6.
4. Luego el bridge lo procesa siguiendo las mismas reglas de OpenFlow del paso 2.4. La primer regla que coincide con el paquete recibido en el `br-int` es la número 4 recibidos por el puerto `qvo570ebdbc-45` se envían a la tabla 25. Luego la regla 9 es utilizada para evitar spoofing, en donde se verifica que la dirección MAC de origen del paquete coincida con la de la instancia 1. Finalmente el paquete es procesado por la tabla 60 en la regla 5 donde se toma la acción `NORMAL`, enviando el paquete al `br-tun`.
5. Una vez en el bridge de tunelización, se aplican las mismas reglas de OpenFlow del paso 2.6 incluyendo la aprendida en el paso 2.19.

```

[root@compute1 ~]# ovs-ofctl dump-flows br-tun --rsort
1 table=0,priority=1,in_port="patch-int" actions=resubmit(,2)
2 table=0,priority=1,in_port="vxlan-0a001f0b" actions=resubmit(,4)
3 table=0,priority=1,in_port="vxlan-0a001f16" actions=resubmit(,4)
4 table=4,priority=1,tun_id=0x13 actions=mod_vlan_vid:6,resubmit(,10)
5 table=10,priority=1 actions=learn(table=20,hard_timeout=300,priority
    =1,cookie=0xb8003f26cd0d7430,NXM_OF_VLAN_TCI[0..11],
    NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:
    NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:OXM_OF_IN_PORT[]),output
    : "patch-int"

```



```

6 table=20,hard_timeout=300, priority=1,vlan_tci=0x0006/0x0fff,dl_dst=
  fa:16:3e:92:57:e4 actions=load:0->NXM_OF_VLAN_TCI[],load:0x13->
  NXM_NX_TUN_ID[],output:"vxlان-0a001f16"
7 table=22,priority=1,dl_vlan=6 actions=strip_vlan,load:0x13->
  NXM_NX_TUN_ID[],output:"vxlان-0a001f0b",output:"vxlان-0a001f16"
8 table=0,priority=0 actions=drop
9 table=2,priority=0,dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions
  =resubmit(,20)
10 table=2,priority=0,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions
  =resubmit(,22)
11 table=3,priority=0 actions=drop
12 table=4,priority=0 actions=drop
13 table=6,priority=0 actions=drop
14 table=20,priority=0 actions=resubmit(,22)
15 table=22,priority=0 actions=drop

```

El procesamiento de las reglas es análogo al detallado en el paso 2.17 pero con sentido opuesto. De esta forma mediante el port “vxlان-0a001f16” se alcanza al bridge br-vxlان (8).

```

4 0.005594 192.168.1.101 192.168.1.102 ICMP 148 Echo (ping) request id=0xde01
5 0.008447 192.168.1.102 192.168.1.101 ICMP 148 Echo (ping) reply id=0xde01

```

```

Frame 4: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
Ethernet II, Src: RealtekU_1b:ff:38 (52:54:00:1b:ff:38), Dst: RealtekU_5b:6a:66 (52:54:00:5b:6a:66)
Internet Protocol Version 4, Src: 10.0.31.12, Dst: 10.0.31.22
User Datagram Protocol, Src Port: 36635, Dst Port: 4789
Virtual eXtensible Local Area Network
  Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 19
    Reserved: 0
Ethernet II, Src: fa:16:3e:be:b9:2e (fa:16:3e:be:b9:2e), Dst: fa:16:3e:92:57:e4 (fa:16:3e:92:57:e4)
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.102
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xda69 [correct]
  [Checksum Status: Good]
  Identifier (BE): 56833 (0xde01)
  Identifier (LE): 478 (0x01de)
  Sequence number (BE): 0 (0x0000)
  Sequence number (LE): 0 (0x0000)
  [Response frame: 5]
  Data (56 bytes)

```

Figura 5.25: Paquete ICMP request encapsulado en VXLAN 19 capturado en el bridge br-vxlان en el nodo de cómputo 1

- El procesamiento desde este bridge hasta la instancia 2 es análogo a los pasos 2.7 - 2.12 con la salvedad que es un paquete ICMP request en lugar de un ARP request.

Paso 4 El procedimiento para la respuesta con el paquete ICMP reply es similar al descrito pero en sentido contrario.

Escenario 2

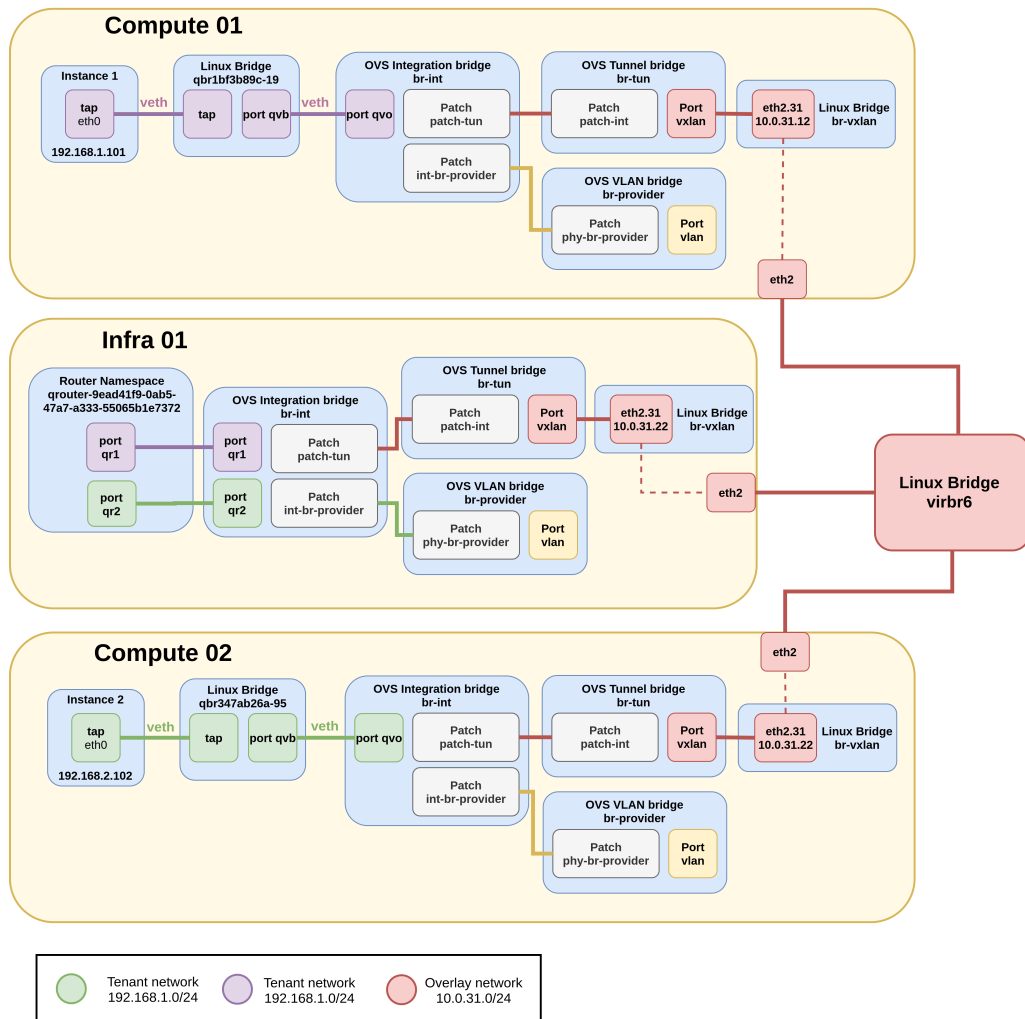


Figura 5.26: Diagrama de arquitectura para el escenario 2 de Open vSwitch

5.3.0.3. Análisis de componentes

A continuación se detallan los componentes virtuales requeridos para instanciar el escenario propuesto. Salvo diferencias en los identificadores, los componentes que son creados en los nodos de cómputo son iguales a los del escenario 1, con lo cual no es relevante volver a detallarlos. Por el contrario, sí se detallan los nuevos componentes en el nodo de infraestructura encargados de interconectar las redes tenant.

Nodos de infraestructura / red

En este escenario las redes instancias por Neutron son:

```
[root@infra1-utility-container-161eebae ~]# openstack network list --
project "Escenario 2" -f json
[
  {
```

```

        "Subnets": "c9e2ffc8-3802-4881-8367-a234483bca33",
        "ID": "c2f9aefe-fde9-452b-9cce-9071268129c6",
        "Name": "tenant-network-1"
    },
    {
        "Subnets": "13c29613-fa50-4f97-8751-5dad3b7703cb",
        "ID": "ed653eb8-9e23-47df-a73e-62e15263c5cf",
        "Name": "tenant-network-2"
    }
]

```

Al igual que en el plugin Linux Bridge, la comunicación entre dos subredes diferentes requiere de un router como intermediario, implementado como un network namespace. El siguiente comando brinda una lista de los routers del escenario.

```

[root@infra1-utility-container-161eebae ~]# openstack router list --
  project "Escenario 2" -f json
[
  {
    "Status": "ACTIVE",
    "Name": "router-X",
    "Project": "e7e830fdcaeb49e3a947104a39440261",
    "State": "UP",
    "HA": false,
    "ID": "9ead41f9-0ab5-47a7-a333-55065b1e7372"
  }
]

```

El namespace asociado al router tiene tantas interfaces como puertos tenga configurados, en este caso son dos debido a que se están conectando dos redes tenant. Con el siguiente comando se listan las interfaces mencionadas:

```

[root@infra1 ~]# ip netns exec qrouter-9ead41f9-0ab5-47a7-a333-55065
  b1e7372 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
  default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
150: qr-b39964a8-f9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
  noqueue state UNKNOWN group default qlen 1000

```

```

link/ether fa:16:3e:5d:84:93 brd ff:ff:ff:ff:ff:ff
inet 192.168.1.1/24 brd 192.168.1.255 scope global qr-b39964a8-f9
    valid_lft forever preferred_lft forever
inet6 fe80::f816:3eff:fe5d:8493/64 scope link
    valid_lft forever preferred_lft forever
151: qr-b6abf816-c0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UNKNOWN group default qlen 1000
    link/ether fa:16:3e:50:de:59 brd ff:ff:ff:ff:ff:ff
inet 192.168.2.1/24 brd 192.168.2.255 scope global qr-b6abf816-c0
    valid_lft forever preferred_lft forever
inet6 fe80::f816:3eff:fe50:de59/64 scope link
    valid_lft forever preferred_lft forever

```

Para que el router se encuentre asociado a ambas redes, es necesario que las mismas se encuentren instanciadas en el nodo de red. Esto se logra de la misma forma que en los nodos de cómputo, instanciando los tres bridges de OVS. A su vez, las interfaces qr del router pertenecen al integration bridge con el fin de comunicar el namespace con Open vSwitch. Esto se puede observar con la salida del siguiente comando, donde dentro de las interfaces del `br-int` también se encuentran las tap asociadas a los servicios de DHCP.

```

[root@infra1 ~]# ovs-vsctl show
952599e3-c14d-4e8f-a02a-1c9ecc3db874
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-provider
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port phy-br-provider
            Interface phy-br-provider
                type: patch
                options: {peer=int-br-provider}
        Port br-provider
            Interface br-provider
                type: internal
        Port br-vlan
            Interface br-vlan
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure

```

```

Port "vxlan-0a001f0c"
    Interface "vxlan-0a001f0c"
        type: vxlan
        options: {df_default="true", egress_pkt_mark
            ="0", in_key=flow, local_ip="10.0.31.11",
            out_key=flow, remote_ip="10.0.31.12"}
Port patch-int
    Interface patch-int
        type: patch
        options: {peer=patch-tun}
Port "vxlan-0a001f16"
    Interface "vxlan-0a001f16"
        type: vxlan
        options: {df_default="true", egress_pkt_mark
            ="0", in_key=flow, local_ip="10.0.31.11",
            out_key=flow, remote_ip="10.0.31.22"}
Port br-tun
    Interface br-tun
        type: internal
Bridge br-int
Controller "tcp:127.0.0.1:6633"
    is_connected: true
fail_mode: secure
Port "tape888d441-79"
    tag: 2
    Interface "tape888d441-79"
        type: internal
Port "tapa62a5790-6b"
    tag: 3
    Interface "tapa62a5790-6b"
        type: internal
Port patch-tun
    Interface patch-tun
        type: patch
        options: {peer=patch-int}
Port "qr-b39964a8-f9"
    tag: 2
    Interface "qr-b39964a8-f9"
        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 "qr-b6abf816-c0"
    tag: 3
    Interface "qr-b6abf816-c0"
        type: internal
Port "tap6f197381-b4"
    tag: 5
    Interface "tap6f197381-b4"
        type: internal
ovs_version: "2.11.0"

```

5.3.0.4. Análisis de tráfico

Como se describe en el escenario, se analiza el tráfico generado en la comunicación de la instancia 1 ubicada en el nodo de cómputo 1 con la instancia 2 alojada en el nodo de cómputo 2. Se supone que las instancias no tienen cargadas las tablas ARP y las reglas de OpenFlow para los destinos específicos no se encuentran cargadas.

Paso 1 La instancia 1 envía el paquete ICMP echo request hacia la instancia 2. Para esto determina que se encuentran en subredes diferentes y por lo tanto busca el siguiente salto en la tabla de ruteo.

```

$ ip r
default via 192.168.1.1 dev eth0
169.254.169.254 via 192.168.1.50 dev eth0
192.168.1.0/24 dev eth0 src 192.168.1.101

```

En la salida se observa que la IP del default gateway es la 192.168.1.1.

Paso 2 La instancia 1 debe obtener la MAC asociada a la IP del router X, como esta no es conocida se dispara el protocolo ARP. Este proceso no se explica dado que es análogo al paso 2 del escenario 1.

Paso 3 Ahora que se conoce la dirección MAC, la instancia 1 retoma el envío del paquete ICMP hacia la instancia 2 a través del router X. El proceso desde que el paquete parte de la instancia 1 hasta el router es análogo al explicado en el paso 3 del escenario 1.

1	0.000000	192.168.1.101	192.168.2.102	ICMP	148 Echo (ping) request	id=0xc001,
4	0.008566	192.168.2.102	192.168.1.101	ICMP	148 Echo (ping) reply	id=0xc001,
5	1.003836	192.168.1.101	192.168.2.102	ICMP	148 Echo (ping) request	id=0xc001,

```

Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
Ethernet II, Src: RealtekU_1b:ff:38 (52:54:00:1b:ff:38), Dst: RealtekU_6f:96:fe (52:54:00:6f:96:fe)
Internet Protocol Version 4, Src: 10.0.31.12, Dst: 10.0.31.11
User Datagram Protocol, Src Port: 39574, Dst Port: 4789
Virtual eXtensible Local Area Network
  Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 30
    Reserved: 0
Ethernet II, Src: fa:16:3e:d9:5b:4d (fa:16:3e:d9:5b:4d), Dst: fa:16:3e:5d:84:93 (fa:16:3e:5d:84:93)
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.2.102
Internet Control Message Protocol

```

Figura 5.27: Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1

Paso 4 Cuando el paquete llega al router se envía hacia la red tenant-network-2 de acuerdo a la tabla de ruteo del namespace:

```

[root@infra1 ~]# ip netns exec qrouter-9ead41f9-0ab5-47a7-a333-55065
b1e7372 ip r
192.168.1.0/24 dev qr-b39964a8-f9 proto kernel scope link src 192.168.1.1
192.168.2.0/24 dev qr-b6abf816-c0 proto kernel scope link src 192.168.2.1

```

Como se encuentra directamente conectado con el host destino, no necesita ningún salto intermedio.

Paso 5 El router X debe obtener la MAC de la instancia 2, como hasta el momento esta no es conocida se dispara el protocolo ARP. Este proceso no se explica dado que es análogo a los descubrimientos detallados anteriormente.

Paso 6 Ahora que se conoce la dirección MAC, el router X retoma el envío del paquete ICMP hacia la instancia 2. El proceso desde que el paquete parte del router hasta su destino es análogo al explicado en el paso 3 del escenario 1.

1	0.000000	192.168.1.101	192.168.2.102	ICMP	148 Echo (ping) request	id=0xc001,
2	0.003929	192.168.2.102	192.168.1.101	ICMP	148 Echo (ping) reply	id=0xc001,
4	1.003201	192.168.1.101	192.168.2.102	ICMP	148 Echo (ping) request	id=0xc001,

```

Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
Ethernet II, Src: RealtekU_6f:96:fe (52:54:00:6f:96:fe), Dst: RealtekU_5b:6a:66 (52:54:00:5b:6a:66)
Internet Protocol Version 4, Src: 10.0.31.11, Dst: 10.0.31.22
User Datagram Protocol, Src Port: 49601, Dst Port: 4789
Virtual eXtensible Local Area Network
  Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 82
    Reserved: 0
Ethernet II, Src: fa:16:3e:50:de:59 (fa:16:3e:50:de:59), Dst: fa:16:3e:bf:0c:85 (fa:16:3e:bf:0c:85)
Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.2.102
Internet Control Message Protocol

```

Figura 5.28: Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 2

Paso 7 El procedimiento para la respuesta con el paquete ICMP reply es similar al descrito pero en sentido contrario.

Escenario 3

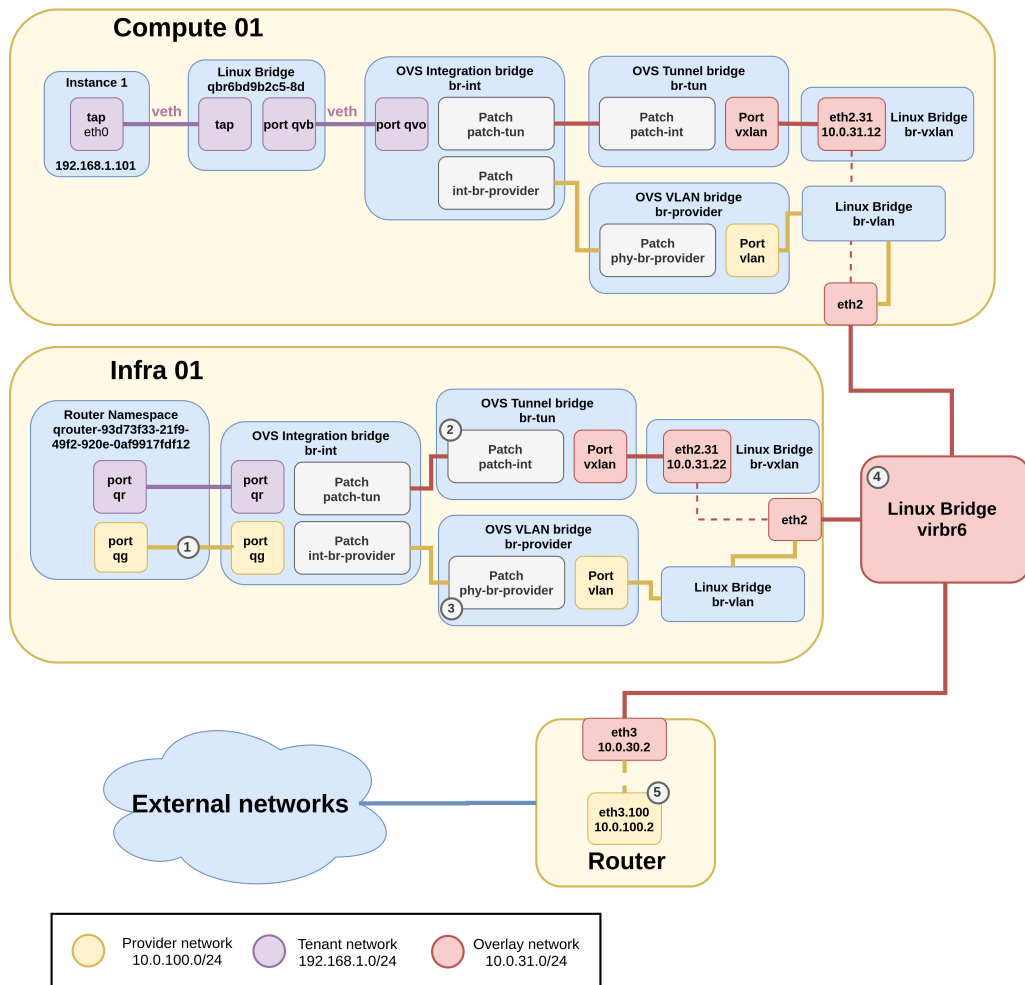


Figura 5.29: Diagrama de arquitectura para el escenario 3 de Open vSwitch

5.3.0.5. Análisis de componentes

A continuación se detallan los componentes virtuales requeridos para instanciar el escenario propuesto. Los componentes relacionados a la red tenant en el nodo de cómputo y en el nodo de infraestructura son análogos a las escenas 1 y 2. Los componentes a analizar son los empleados para dar soporte a la red provider.

Nodos de infraestructura / red

En este escenario las redes tenant instancias por Neutron son:

```
[root@infra1-utility-container-161eebae ~]# openstack network list --
project "Escenario 3" -f json
[
  {
    "Subnets": "af7d8f67-8d00-40f9-ba9e-b38224e7d7ee",
    "ID": "94b2baa9-ed72-409b-a74d-fd47c0b50d19",
```



```

        "Name": "tenant-network-1"
    }
]

```

Mientras que la red externa utilizada es:

```

[root@infra1-utility-container-161eebae ~]# openstack network list --
external -f json
[
  {
    "Subnets": "905e5c2f-14b9-40d8-9647-4fb0680aca22",
    "ID": "0610e256-5b34-4479-95a4-452519c10234",
    "Name": "provider-vlan"
  }
]

```

Al igual que en el escenario anterior, la comunicación entre dos subredes diferentes requiere de un router como intermediario, implementado como un network namespace. Con el siguiente comando se obtiene el router del escenario.

```

[root@infra1-utility-container-161eebae ~]# openstack router list --
project "Escenario 3" -f json
[
  {
    "Status": "ACTIVE",
    "Name": "router-Y",
    "Project": "4909f8000a82406b9dd34d647e23f03c",
    "State": "UP",
    "HA": false,
    "ID": "93d73f33-21f9-49f2-920e-0af9917fdf12"
  }
]

```

En este escenario el namespace asociado al router tiene dos interfaces, una asociada a la red tenant y otra asociada a la red provider. Como ya se analizó con el plugin Linux Bridge, el nombre de las interfaces asociadas a una red tenant tienen el formato qr-<id_port> mientras que las interfaces asociadas a una red provider tienen el formato qg-<id_port>.

```

[root@infra1 ~]# ip netns exec qrouter-93d73f33-21f9-49f2-920e-0
af9917fdf12 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

```

```

inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
164: qg-1559b31f-5d: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UNKNOWN group default qlen 1000
    link/ether fa:16:3e:31:8a:5a brd ff:ff:ff:ff:ff:ff
    inet 10.0.100.114/24 brd 10.0.100.255 scope global qg-1559b31f-5d
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe31:8a5a/64 scope link
        valid_lft forever preferred_lft forever
165: qr-e85900b1-10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UNKNOWN group default qlen 1000
    link/ether fa:16:3e:60:4f:63 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global qr-e85900b1-10
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe60:4f63/64 scope link
        valid_lft forever preferred_lft forever

```

La red tenant sigue siendo implementada mediante OVS por los bridges `br-int` y `br-tun` al igual que en los escenarios anteriores. Por otro lado, para instanciar la red provider con el vlan id 100 se requiere de dos estructuras:

- La primera es la arquitectura subyacente creada por el administrador, la cual es idéntica a la detallada en el escenario 3 del plugin Linux Bridge.
- La segunda estructura consiste en un nuevo bridge provider de OVS que brinda conectividad con la red VLAN de la infraestructura física. En este caso concreto se nombra `br-provider` y tiene asociadas las siguientes interfaces:

```

Bridge br-provider
    Controller "tcp:127.0.0.1:6633"
        is_connected: true
    fail_mode: secure
    Port phy-br-provider
        Interface phy-br-provider
            type: patch
            options: {peer=int-br-provider}
    Port br-provider
        Interface br-provider
            type: internal
    Port br-vlan
        Interface br-vlan

```

Dentro de estas interfaces cabe destacar el patch port compuesto por los pares (phy-br-provider y int-br-provider) el cual brinda conectividad con el br-int y la interfaz br-vlan que es la interfaz física del nodo de red dedicada a las redes provider VLAN.

5.3.0.6. Análisis de tráfico

El objetivo de este escenario es analizar el tráfico generado en la comunicación entre la instancia 1 y un host externo al manejo de OpenStack. En este caso se considera al router de la infraestructura física como host externo debido a que si la instancia logra comunicarse con el mismo, entonces tendrá acceso a cualquier host que este alcance.

Paso 1 La instancia 1 envía el paquete ICMP echo request hacia el router físico. Para esto determina que se encuentran en subredes diferentes y por lo tanto busca el siguiente salto en la tabla de ruteo

```
$ ip r
default via 192.168.1.1 dev eth0
169.254.169.254 via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0 src 192.168.1.101
```

En la salida se observa que la IP del default gateway es la 192.168.1.1.

Paso 2 La instancia 1 debe obtener la MAC asociada a la IP del router Y, como esta no es conocida se dispara el protocolo ARP. Este proceso no se explica dado que es análogo al paso 2 del escenario 1.

Paso 3 Ahora que se conoce la dirección MAC, la instancia 1 retoma el envío del paquete ICMP hacia el router físico a través del router Y. El proceso desde que el paquete parte de la instancia 1 hasta el router Y es análogo al explicado en el paso 3 del escenario 1.

No.	Time	Source	Destination	Protocol	Length	Info
→	4 0.002488	192.168.1.101	10.0.100.2	ICMP	148	Echo (ping) request id=0xe801, seq=0/0,
←	5 0.004207	10.0.100.2	192.168.1.101	ICMP	148	Echo (ping) reply id=0xe801, seq=0/0,
<ul style="list-style-type: none"> ▶ Frame 4: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) ▶ Ethernet II, Src: RealtekU_5b:6a:66 (52:54:00:5b:6a:66), Dst: RealtekU_6f:96:fe (52:54:00:6f:96:fe) ▶ Internet Protocol Version 4, Src: 10.0.31.22, Dst: 10.0.31.11 ▶ User Datagram Protocol, Src Port: 59082, Dst Port: 4789 ▶ Virtual eXtensible Local Area Network <ul style="list-style-type: none"> ▶ Flags: 0x0800, VXLAN Network ID (VNI) ▶ Group Policy ID: 0 ▶ VXLAN Network Identifier (VNI): 24 ▶ Reserved: 0 ▶ Ethernet II, Src: fa:16:3e:f0:c6:fc (fa:16:3e:f0:c6:fc), Dst: fa:16:3e:60:4f:63 (fa:16:3e:60:4f:63) ▶ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 10.0.100.2 ▶ Internet Control Message Protocol 						

Figura 5.30: Paquete ICMP echo request encapsulado en VXLAN capturado en la interfaz br-vxlan en el nodo de cómputo 1

Paso 4 Cuando el paquete llega al router Y se envía hacia la red provider-vlan de acuerdo a la tabla de ruteo del namespace:

```
[root@infra1 ~]# ip netns exec qrouter-93d73f33-21f9-49f2-920e-0
af9917fdf12 ip r
default via 10.0.100.2 dev qg-1559b31f-5d
10.0.100.0/24 dev qg-1559b31f-5d proto kernel scope link src 10.0.100.114
192.168.1.0/24 dev qr-e85900b1-10 proto kernel scope link src 192.168.1.1
```

Como se encuentra directamente conectado con el host destino, no necesita ningún salto intermedio.

Paso 5 El router Y debe obtener la MAC del router físico, como hasta el momento esta no es conocida se dispara el protocolo ARP. A continuación se detalla cómo se lleva a cabo el protocolo de resolución de direcciones entre el router virtual y el físico.

1. El router Y envía una trama ARP request preguntando por la IP 10.0.100.2 hacia el integration bridge a través del puerto qg (1).
2. El br-int recibe el pedido ARP, lo etiqueta con el VLAN ID interno 8 y lo reenvía siguiendo las siguientes reglas de OpenFlow:

```
[root@infra1 ~]# ovs-ofctl dump-flows br-int --rsort
1 table=0,priority=65535,vlan_tci=0x0fff/0x1fff actions=drop
2 table=0,priority=3,in_port="int-br-provider",dl_vlan=100 actions=
  mod_vlan_vid:8,resubmit(,60)
3 table=60,priority=3 actions=NORMAL
4 table=0,priority=2,in_port="int-br-provider" actions=drop
5 table=0,priority=0 actions=resubmit(,60)
6 table=23,priority=0 actions=drop
7 table=24,priority=0 actions=drop
```

La primer regla que coincide es la número 5 en donde se envía el procesamiento a la tabla 60. La regla 3 ejecuta la acción NORMAL enviando el paquete a todas las interfaces del br-int menos la qg por la cual llego el mensaje.

3. Las reglas del bridge br-tun descartaran el paquete debido a que no manejan la etiqueta interna 8 (2). Por otro lado, el br-provider (3) modificará el VLAN interno por la etiqueta 100 y reenviará el paquete a todas las interfaces (en particular la br-vlan).

```
root@infra1 ~]# ovs-ofctl dump-flows br-provider --rsort
```

- 1 table=0,priority=4,in_port="phy-br-provider",dl_vlan=8 actions=mod_vlan_vid:100,NORMAL
 - 2 table=0,priority=2,in_port="phy-br-provider" actions=drop
 - 3 table=0,priority=0 actions=NORMAL
4. El paquete es recibido por el Linux Bridge br-vlan y reenviado a través de la interfaz eth2. El mismo es transportado mediante la infraestructura física subyacente (4), alcanzando todos los nodos dentro de la VLAN 100.

No.	Time	Source	Destination	Protocol	Length	Info
2	0.440248	fa:16:3e:31:8a:5a	Broadcast	ARP	46	Who has 10.0.100.2? Tell 10.0.100.114
3	0.440512	RealtekU_cc:b7:63	fa:16:3e:31:8a:5a	ARP	46	10.0.100.2 is at 52:54:00:cc:b7:63

```

Frame 2: 46 bytes on wire (368 bits), 46 bytes captured (368 bits)
Ethernet II, Src: fa:16:3e:31:8a:5a (fa:16:3e:31:8a:5a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 100
    000. .... = Priority: Best Effort (default) (0)
    ...0 .... = DEI: Ineligible
    .... 0000 0110 0100 = ID: 100
    Type: ARP (0x0806)
  Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: fa:16:3e:31:8a:5a (fa:16:3e:31:8a:5a)
    Sender IP address: 10.0.100.114
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.100.2

```

Figura 5.31: Paquete ARP request taggeado con el VLAN ID 100 capturado en la interfaz br-vlan en el nodo de red

5. Cuando el router físico recibe la consulta por su interfaz eth3.100 (5), quita el tag de VLAN y procesa el paquete. Esto genera una respuesta hacia el router Y con un ARP reply directo a su MAC.
6. El procedimiento para la respuesta es similar al descrito pero en sentido contrario.

Paso 6 Ahora que se conoce la dirección MAC, el router Y retoma el envío del paquete ICMP hacia el router físico. Previo al mismo, debe encargarse de realizar un source NAT para permitir que la respuesta alcance la instancia. Esto se realiza utilizando la interfaz qg (10.0.100.114) como la dirección IP de origen.

1. El mensaje es enviado hacia el integration bridge por el puerto qg (1).
2. El br-int recibe el paquete y realiza la misma acción que el punto 2 del paso 5.
3. Nuevamente el br-provider realiza la misma acción que el punto 3 del paso 5.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.440796	10.0.100.114	10.0.100.2	ICMP	102	Echo (ping) request id=0xe801, seq=0/0
5	0.441034	10.0.100.2	10.0.100.114	ICMP	102	Echo (ping) reply id=0xe801, seq=0/0,

```

▶ Frame 4: 102 bytes on wire (816 bits), 102 bytes captured (816 bits)
▶ Ethernet II, Src: fa:16:3e:31:8a:5a (fa:16:3e:31:8a:5a), Dst: RealtekU_cc:b7:63 (52:54:00:cc:b7:63)
▼ 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 100
  000. .... = Priority: Best Effort (default) (0)
  ...0 .... = DEI: Ineligible
  ... 0000 0110 0100 = ID: 100
  Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 10.0.100.114, Dst: 10.0.100.2
▶ Internet Control Message Protocol

```

Figura 5.32: Paquete ICMP echo request capturado en la interfaz br-vlan del nodo de red

4. El paquete es recibido por el Linux Bridge `br-vlan` y reenviado a través de la interfaz `eth2` hacia el router físico. El mismo es transportado mediante la infraestructura física subyacente (4), alcanzando el destino.
5. El paquete es recibido por la interfaz `eth3.100` del router físico, la cual quita el tag de VLAN y procesa el ICMP echo request.

Paso 7 El procedimiento para la respuesta `ICMPEchoreply` es similar al descrito pero en sentido contrario.

Escenario 4

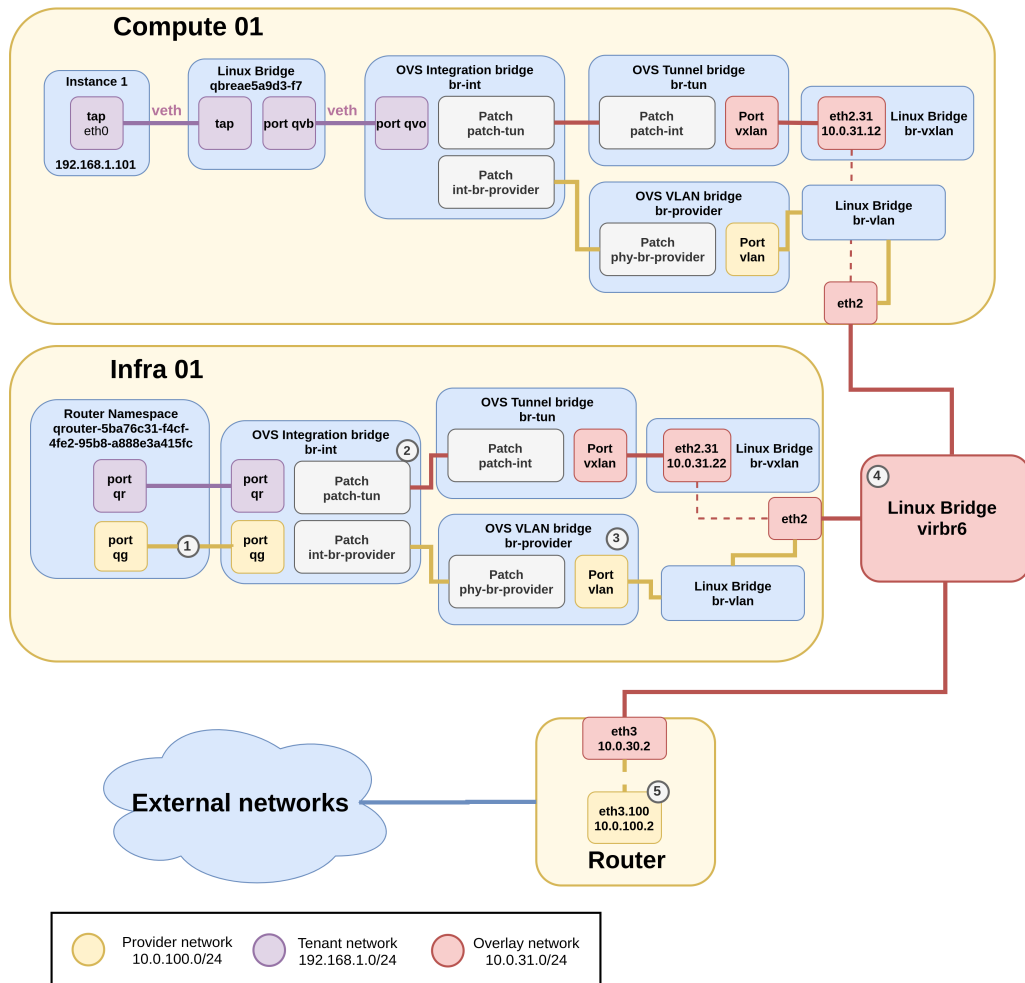


Figura 5.33: Diagrama de arquitectura para el escenario 4 de Open vSwitch

5.3.0.7. Análisis de componentes

Los componentes virtuales requeridos para instanciar el escenario 4 son exactamente los mismos que los definidos en el escenario anterior. La única diferencia existente se da en las interfaces del router Z en el nodo de infraestructura, debido a que debe realizar la correspondencia entre la IP flotante y la IP fija de la instancia. Al igual que fue mencionado en el análisis de componentes anterior, se detallan las interfaces del router accediendo al namespace del mismo mediante el siguiente comando:

```
[root@infra1 ~]# ip netns exec grouter-5ba76c31-f4cf-4fe2-95b8-a888e3a415fc ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```

    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
163: qg-3f5f5f87-bb: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UNKNOWN group default qlen 1000
    link/ether fa:16:3e:df:7d:3e brd ff:ff:ff:ff:ff:ff
    inet 10.0.100.136/24 brd 10.0.100.255 scope global qg-3f5f5f87-bb
        valid_lft forever preferred_lft forever
    inet 10.0.100.71/32 brd 10.0.100.71 scope global qg-3f5f5f87-bb
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fedf:7d3e/64 scope link
        valid_lft forever preferred_lft forever
167: qr-6c9d0914-5e: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UNKNOWN group default qlen 1000
    link/ether fa:16:3e:49:3f:f8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global qr-6c9d0914-5e
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe49:3ff8/64 scope link
        valid_lft forever preferred_lft forever

```

En la salida de este comando se observa que la interfaz `qg-3f5f5f87-bb` dedicada para la red provider, tiene asignadas la IP del router (10.0.100.136) y la flotante de la instancia (10.0.100.71) necesaria para implementar el NAT estático uno a uno con la IP fija de la instancia en la red tenant (192.168.1.101).

5.3.0.8. Análisis de tráfico

El objetivo de este escenario es analizar el tráfico generado en la comunicación desde un host externo al manejo de OpenStack hacia la instancia 1. En este caso se considera al router de la infraestructura física como host externo debido a que si este logra comunicarse con la instancia, entonces también podrá hacerlo cualquier host externo que alcance al router.

Paso 1 El router físico envía el paquete ICMP echo request hacia la instancia 1 utilizando la IP flotante, es decir que el destino será el router Z. Para esto determina que se encuentra directamente conectado con el host destino y por lo tanto no necesita ningún salto intermedio. Con el siguiente comando se observa la tabla de ruteo de dicho router:

```

[root@router ~]# ip r
default via 10.0.40.1 dev eth0 proto static metric 100

```



```

10.0.10.0/24 dev eth1 proto kernel scope link src 10.0.10.2 metric 101
10.0.20.0/24 dev eth2 proto kernel scope link src 10.0.20.2 metric 102
10.0.30.0/24 dev eth3 proto kernel scope link src 10.0.30.2 metric 103
10.0.40.0/24 dev eth0 proto kernel scope link src 10.0.40.2 metric 100
10.0.100.0/24 dev eth3.100 proto kernel scope link src 10.0.100.2 metric
400
10.0.101.0/24 dev eth3.101 proto kernel scope link src 10.0.101.2 metric
401

```

Paso 2 El router físico debe obtener la MAC del router Z, como hasta el momento esta no es conocida se dispara el protocolo ARP. Este procedimiento no se explica dado a que es análogo, pero en sentido inverso, al detallado en el paso 5 del escenario 3.

Paso 3 Ahora que se conoce la dirección MAC, el router físico retoma el envío del paquete ICMP hacia el router Z.

1. El mensaje es enviado por la subinterfaz `eth3.100` (5) en donde se etiqueta el mismo con el VLAN ID 100 hacia la infraestructura física subyacente (4).

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000993	10.0.100.2	10.0.100.71	ICMP	102	Echo (ping) request id=0x7c9c, seq=1/256,
4	0.012261	10.0.100.71	10.0.100.2	ICMP	102	Echo (ping) reply id=0x7c9c, seq=1/256,

▶ Frame 3: 102 bytes on wire (816 bits), 102 bytes captured (816 bits)
 ▶ Ethernet II, Src: RealtekU_cc:b7:63 (52:54:00:cc:b7:63), Dst: fa:16:3e:df:7d:3e (fa:16:3e:df:7d:3e)
 ▼ 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 100
 000. = Priority: Best Effort (default) (0)
 ...0 = DEI: Ineligible
 ... 0000 0110 0100 = ID: 100
 Type: IPv4 (0x0800)
 ▶ Internet Protocol Version 4, Src: 10.0.100.2, Dst: 10.0.100.71
 ▶ Internet Control Message Protocol

Figura 5.34: Paquete ICMP echo request taggeado con el VLAN ID 100 capturado en la interfaz `eth3` del router físico

2. El paquete es recibido por la interfaz `eth2` del nodo de red y se reenvía hacia el Linux Bridge `br-vlan` llegando luego al bridge `br-provider` (3).
3. En este punto el paquete es procesado por las reglas OpenFlow que determinan el reenvío el paquete por todas las interfaces, particularmente por el patch port `phy-br-provider`.

```

root@infra1 ~]# ovs-ofctl dump-flows br-provider --rsort
1 table=0,priority=4,in_port="phy-br-provider",dl_vlan=8 actions=
  mod_vlan_vid:100,NORMAL
2 table=0,priority=2,in_port="phy-br-provider" actions=drop
3 table=0,priority=0 actions=NORMAL

```

- El paquete llega al bridge de integración (2) de OVS donde nuevamente es procesado por las reglas OpenFlow.

```
[root@infra1 ~]# ovs-ofctl dump-flows br-int --rsort
1 table=0,priority=65535,vlan_tci=0x0fff/0x1fff actions=drop
2 table=0,priority=3,in_port="int-br-provider",dl_vlan=100 actions=
  mod_vlan_vid:8,resubmit(,60)
3 table=60,priority=3 actions=NORMAL
4 table=0,priority=2,in_port="int-br-provider" actions=drop
5 table=0,priority=0 actions=resubmit(,60)
6 table=23,priority=0 actions=drop
7 table=24,priority=0 actions=drop
```

En esta oportunidad, se ejecuta la regla 2 que cambia el tag 100 por el interno 8 y lo envía a todas las interfaces mediante la regla 3.

- En particular el paquete alcanza la interfaz qg (1) del namespace del router de Neutron.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000550	10.0.100.2	10.0.100.71	ICMP	98	Echo (ping) request id=0x7c9c, seq=1/256,
4	0.011435	10.0.100.71	10.0.100.2	ICMP	98	Echo (ping) reply id=0x7c9c, seq=1/256,
▶ Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) ▶ Ethernet II, Src: RealtekU_cc:b7:63 (52:54:00:cc:b7:63), Dst: fa:16:3e:df:7d:3e (fa:16:3e:df:7d:3e) ▶ Internet Protocol Version 4, Src: 10.0.100.2, Dst: 10.0.100.71 ▶ Internet Control Message Protocol						

Figura 5.35: Paquete ICMP echo request capturado en la interfaz qg del router de Neutron

Paso 4 En el router el servicio de iptables realiza el DNAT sobre el paquete utilizando la IP de la instancia como dirección destino. Estos mapeos se pueden ver en detalle con el comando: `ip netns exec qrouter-5ba76c31-f4cf-4fe2-95b8-a888e3a415fc iptables -t nat -L`.

Paso 5 El router envía el nuevo paquete ICMP hacia la instancia 1. El proceso desde que el paquete parte del router hasta su destino es análogo al explicado en el paso 3 del escenario 1.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.100.2	192.168.1.101	ICMP	98	Echo (ping) request id=0x7c9c, seq=1/256,
2	0.010788	192.168.1.101	10.0.100.2	ICMP	98	Echo (ping) reply id=0x7c9c, seq=1/256,
▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) ▶ Ethernet II, Src: fa:16:3e:49:3f:f8 (fa:16:3e:49:3f:f8), Dst: fa:16:3e:c5:eb:f5 (fa:16:3e:c5:eb:f5) ▶ Internet Protocol Version 4, Src: 10.0.100.2, Dst: 192.168.1.101 ▶ Internet Control Message Protocol						

Figura 5.36: Paquete ICMP echo request capturado en la interfaz qr del router de Neutron

Paso 6 El procedimiento para la respuesta ICMP echo reply es similar al descrito pero en sentido contrario, con la salvedad de que en el momento en que el router Z reenvía la respuesta hacia el router físico, deberá realizar un source NAT cambiando la IP de origen de la instancia 1 por la de la interfaz qg.

5.3.1. Ventajas y desventajas

Al igual que en la sección anterior, se presenta un listado de aspectos positivos y negativos relevantes del plugin Open vSwitch.

Ventajas:

- Gran soporte por parte de la comunidad junto con amplia documentación de su funcionamiento.
- Mantiene soporte para redes de tunelización de tipo VXLAN y GRE.
- Es compatible con el servicio de QoS de Neutron.
- Soporta la mayoría de las funcionalidades avanzadas de Neutron, como por ejemplo VRRP y DVR.
- Tiene un manejo más eficiente de las reglas de firewall debido a que utiliza OpenFlow en lugar de iptables.
- Trae consigo todas las funcionalidades propias de Open vSwitch.

Desventajas:

- El diagnóstico de problemas puede resultar dificultoso debido a que las interfaces creadas pueden no ser accesibles por herramientas estándares.
- Para su administración es necesario utilizar un conjunto de comandos de mayor complejidad que Linux Bridge.

Conclusiones

Luego de haber finalizado el análisis del módulo de red Neutron, se logró dimensionar la importancia del mismo dentro de una plataforma de Datacenters. El hecho de que le brinde al usuario final una abstracción de la capa de red virtualizada por encima de la infraestructura subyacente es sumamente relevante debido a que le permite administrar sus recursos de red de forma independiente. El único caso en el

que se requiere una mínima participación de un administrador de la plataforma, es cuando el usuario necesita una conexión hacia el exterior mediante una red provider.

Se resalta la existencia del plugin ml2, el cual permite independizar las funcionalidades centrales del módulo de red de la implementación a bajo nivel realizada por los diversos mechanism drivers. En particular, este comportamiento se confirmó al estudiar los drivers Linux Bridge y OVS en donde si bien difieren las implementaciones para resolver la comunicación, desde el punto de vista del usuario final el comportamiento de Neutron es indiferente.

De los drivers analizados se destaca su valor dentro de la comunidad, siendo los dos más utilizados e incluso documentados oficialmente por el equipo de OpenStack. Este prestigio se debe a la madurez y robustez que presentan ambas tecnologías. A la hora de determinar cuál de las dos emplear, se deben tener claras los requerimientos que se pretenden alcanzar. En función de esto, no solo se deben tener en cuenta las funcionalidades innovadoras provistas sino también la simplicidad de las implementaciones. Se debe prestar suma atención a este último punto ya que cuanto más simple sea el diseño de OpenStack, más sencillo será su despliegue y posterior mantenimiento.

Durante el análisis realizado, naturalmente se planteó estudiar un escenario formado por varias subredes tenant distribuidas de tal forma que el camino entre dos instancias se componga por varios saltos de routers. Este escenario tenía por objetivo estudiar el routing dinámico dentro de OpenStack. Lamentablemente, luego de haber investigado, se observó que Neutron no soporta ruteo dinámico, por lo tanto en caso de requerir rutas específicas para alcanzar una red o crear el camino más eficiente se debe configurar manualmente.

Capítulo 6

Conclusiones

Luego de haber realizado todo el proceso de despliegue de un Datacenter de pruebas mediante OpenStack, se tomó conciencia de todos los aspectos y conceptos que involucra la instalación y mantenimiento de una plataforma de esta magnitud. Es por esto que para llevar a cabo el trabajo fue necesario adquirir y aplicar conocimientos en diversas áreas computacionales tales como virtualización, manejo de contenedores, gestión de redes, almacenamiento de datos y administración de sistemas. OpenStack como herramienta resulta ser muy valorada gracias a que posee una gran flexibilidad para adecuar el despliegue de un Datacenter a las necesidades de cada caso, permitiendo instalar módulos con funcionalidades específicas y prescindir de aquellos que no sean necesarios. En esta línea es que se debe ser muy cuidadoso durante la etapa de diseño, al momento de tomar decisiones sobre qué tecnologías utilizar. Es fundamental no perder el foco sobre qué requerimientos se deben alcanzar y evitar agregar complejidad con tecnologías de mayores prestaciones que las necesarias para el problema que se busca resolver.

El proceso de instalación resulta sumamente complejo debido a la cantidad de herramientas y configuraciones a tener en cuenta, es por esto que la utilización de un mecanismo de automatización de tareas resulta inevitable. Como se menciona en el documento, se utilizó Ansible para facilitar el proceso ya que permite ejecutar reiteradas instalaciones de forma idéntica y sobre múltiples servidores sin una carga operativa adicional. Aún así, la experiencia no fue sencilla debido a los diversos inconvenientes que se encontraron durante el camino, lo que implicó que lograr todo el trabajo llevará un tiempo mayor al estimado. Una vez alcanzado un ambiente funcional, las tareas de mantenimiento y expansión del Datacenter tampoco resultaron triviales. Se enfatiza la importancia de mantener entornos de producción y desarrollo por separado con el fin de poder simular la tarea que se pretende realizar y reducir el riesgo de generar errores en producción.

Aunque se puede encontrar una vasta documentación acerca del despliegue y

mantenimiento de la plataforma, la experiencia demuestra que muchas veces existe una amplia brecha entre lo teórico y lo práctico. Interactuar con OpenStack de primera mano permite adquirir destrezas que resultan fundamentales para manipular ambientes productivos. A pesar de su dificultad, se resalta el gran apoyo e inversión a nivel internacional en esta plataforma, lo que ha generado altos niveles de crecimiento durante la última década. Esto último sumado a la rapidez con la que se generan nuevas versiones y funcionalidades son indicadores de que OpenStack es, sin duda alguna, algo por lo que apostar en el ambiente de infraestructuras.

En cuanto al módulo de red Neutron, si bien no se alcanzó a realizar un extenso análisis como fue planificado inicialmente, sí se comprendió la importancia que tiene dentro de la plataforma. Una vez que se encuentra configurado correctamente, comunicar instancias, redes y routers virtuales resulta sumamente sencillo para un usuario final y no requiere intervención constante por parte de un administrador. Esta capacidad brinda una capa de abstracción vital para un Datacenter comercial ya que disminuye la necesidad de mantener una mano remota que conecte switches físicos para cumplir con los requerimientos de comunicación del cliente.

En base a lo mencionado hasta el momento, vale la pena cuestionarse en qué situaciones amerita desplegar OpenStack, teniendo en cuenta el tamaño de la infraestructura a utilizar y el dinamismo de los ambientes de los usuarios. Debido a que existe una carga adicional necesaria para adquirir habilidades con la herramienta, tanto de despliegue como de gestión, se tiende a creer que la utilización de OpenStack sólo se justifica en ambientes de mediano o gran porte. Sin embargo, en las encuestas oficiales realizadas en 2019 por el equipo de OpenStack (tomando una muestra de aprox. 400 despliegues) se observa que también es empleado para pequeños ambientes (figuras 6.1 y 6.2).

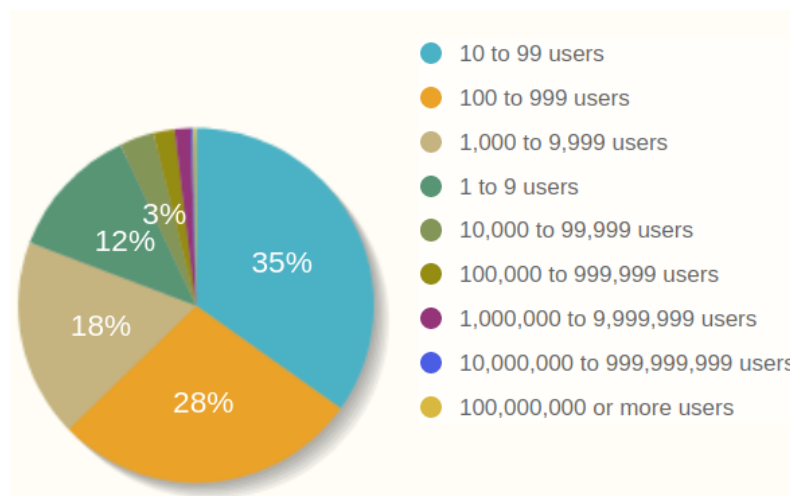


Figura 6.1: Cantidad de usuarios por despliegue. Extraída de [51].

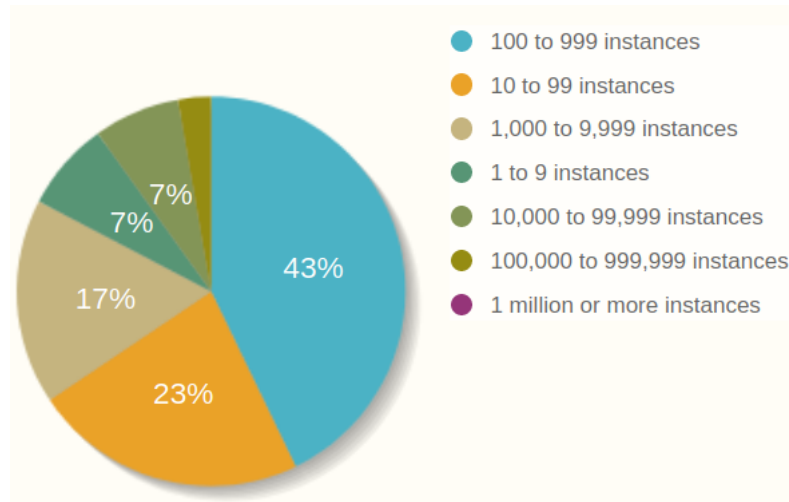


Figura 6.2: Cantidad de instancias por despliegue. Extraída de [51].

6.1. Trabajo a futuro

En esta sección se presentan diferentes ideas y propuestas a realizar a futuro que fueron consideradas por el equipo pero que no fueron alcanzadas principalmente por falta de tiempo o porque no se encontraban en el alcance original del proyecto. Además algunos puntos, como el acceso directo a Internet, escapan del control de los autores.

Firewall

Deshabilitar por completo la denegación de paquetes por defecto en el firewall de CentOS deja el servidor expuesto ante vulnerabilidades, lo cual para un ambiente de producción es inadmisibile. Como aspecto a mejorar se propone analizar y documentar todas las conexiones HTTP realizadas por OpenStack Ansible durante su instalación y en su posterior ejecución entre cada uno de los nodos involucrados. Una vez identificadas, se debe agregar en cada nodo las reglas iptables correspondientes que habiliten dichas conexiones en particular.

Arquitectura segura

Se propone evaluar diversos diseños de arquitectura que involucren la utilización de un firewall que separe adecuadamente la red interna de la pública utilizando opcionalmente una DMZ. Esta evaluación debe tener en cuenta tanto accesos SSH a los nodos del Datacenter por parte de administradores, como accesos directos a las instancias utilizadas por los usuarios finales. Además, debe permitir el tráfico HTTP para la correcta interacción con el dashboard de la plataforma. A la hora

de tomar una decisión se debe considerar que de momento no se cuenta con una conexión directa hacia el exterior, por lo que en caso de que esto ocurra se debe reforzar adecuadamente la seguridad del Datacenter.

Brindar conexión directa a Internet

Con el fin de simplificar el acceso hacia el exterior tanto durante la instalación como en su posterior ejecución, se debe evaluar una solución que permita mantener una conexión directa con el proxy de la FING, evitando así la necesidad de realizar un túnel SSH reverso como fue mencionado en la descripción del ambiente de trabajo.

Ambiente de producción

Con el fin de alcanzar un ambiente de producción real en el Instituto de Computación, se debe ajustar el diseño de arquitectura propuesto aumentando la cantidad de nodos y los recursos de los mismos. Esto asegurará un aumento de performance, capacidad de cómputo, almacenamiento y mayor resiliencia. A modo de guía se sugiere utilizar un mínimo de tres nodos de infraestructura, tres nodos de almacenamiento con amplia capacidad, dos balanceadores de carga y nodos de cómputo con la mayor cantidad de recursos de CPU y RAM posibles. Por otro lado, se recomienda la utilización de cuatro interfaces físicas por nodo emparejadas en dos bonds que separen el plano de datos del de control.

Analizar plugins de red

Durante el proyecto se logró investigar el comportamiento de los dos métodos más utilizados para resolver la comunicación por parte del módulo de red. Con fines de investigación, resulta interesante estudiar drivers menos convencionales como pueden ser Calico, Open Virtual Network, OpenContrail y ONOS. Posteriormente sería relevante realizar un estudio comparativo entre estos, incluyendo pruebas de performance, que permita hacer un balance entre los beneficios y perjuicios de cada uno.

Contenedores

Dado el creciente aumento en la utilización de contenedores en el mundo de TI, es pertinente buscar y estudiar la forma de instanciar contenedores dentro de un cloud OpenStack.

Referencias

- [1] 802.1q-2014 - bridges and bridged networks. <http://www.ieee802.org/1/pages/802.1Q-2014.html>. Accedido: 2019-06-15.
- [2] Ansible. Module index. https://docs.ansible.com/ansible/latest/modules/modules_by_category.html. Accedido: 2019-07-05.
- [3] ArchLinux. Linux containers. https://wiki.archlinux.org/index.php/Linux_Containers. Accedido: 2019-06-20.
- [4] ArchLinux. Network bridge. https://wiki.archlinux.org/index.php/Network_bridge. Accedido: 2020-01-08.
- [5] Red Hat Bugzilla. Bug 1704413. https://bugzilla.redhat.com/show_bug.cgi?id=1704413. Accedido: 2020-01-13.
- [6] Ceph. Architecture. <https://docs.ceph.com/docs/master/architecture/>. Accedido: 2020-01-12.
- [7] Ceph. Cluster operations. <https://docs.ceph.com/docs/jewel/rados/operations/>. Accedido: 2020-01-12.
- [8] Ceph. Osd scenario. <https://docs.ceph.com/ceph-ansible/master/osds/scenarios.html>. Accedido: 2020-01-22.
- [9] Cisco. What is a data center. <https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html>. Accedido: 2019-06-20.
- [10] James Denton. *Learning Openstack Networking*, chapter 1 Introduction to Openstack Networking, page 15. Packt Publishing, 3rd edition, 2018.
- [11] James Denton. *Learning Openstack Networking*, chapter 1 Introduction to Openstack Networking, pages 21–23. Packt Publishing, 3rd edition, 2018.
- [12] James Denton. *Learning Openstack Networking*, chapter 1 Introduction to Openstack Networking. Packt Publishing, 3rd edition, 2018.

- [13] Docker. What is a container? <https://www.docker.com/resources/what-container>. Accedido: 2019-06-20.
- [14] GitHub. Redhat: stop updating system unnecessarily. https://github.com/openstack/openstack-ansible-lxc_container_create/commit/b15b97fa0ab73579e4939ae767a2810196b33df2#diff-0dc9e326c64f4a15341a43e16b94ee60. Accedido: 2020-01-22.
- [15] Red Hat. Lvm (logical volume manager). https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch-lvm. Accedido: 2020-01-12.
- [16] Red Hat. What is kvm? <https://www.redhat.com/en/topics/virtualization/what-is-KVM>. Accedido: 2019-06-20.
- [17] Red Hat. What is virtualization? <https://www.redhat.com/en/topics/virtualization/what-is-virtualization>. Accedido: 2019-06-20.
- [18] Red Hat. What's a linux container. <https://www.redhat.com/en/topics/containers/whats-a-linux-container>. Accedido: 2019-06-20.
- [19] IBM. A quick overview of openstack technology. <https://www.ibm.com/blogs/cloud-computing/2014/08/06/quick-overview-openstack-technology/>. Accedido: 2020-01-22.
- [20] libvirt. virsh. <https://libvirt.org/manpages/virsh.html>. Accedido: 2020-01-13.
- [21] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Larry Kreeger, T. Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. RFC 7348.
- [22] Steve Martinelli, Henry Nash, and Brad Topol. *Identity, Authentication, and Access Management in OpenStack: Implementing and Deploying Keystone*. O'Reilly Media, Inc., 1st edition, 2015.
- [23] Stephen Nadas. Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6. RFC 5798, March 2010.
- [24] OpenStack. Aio build fails on selinux file context tasks. <https://bugs.launchpad.net/openstack-ansible/+bug/1782798>. Accedido: 2019-07-05.

- [25] OpenStack. Appendix e: Container networking. <https://docs.openstack.org/project-deploy-guide/openstack-ansible/ocata/app-networking.html#network-diagrams>. Accedido: 2019-07-05.
- [26] OpenStack. Ceph production example. <https://docs.openstack.org/openstack-ansible/latest/user/ceph/full-deploy.html>. Accedido: 2020-01-22.
- [27] OpenStack. Configure access and security for instances. <https://docs.openstack.org/horizon/latest/user/configure-access-and-security-for-instances.html>. Accedido: 2019-08-03.
- [28] OpenStack. Configure ssh between compute nodes. <https://docs.openstack.org/nova/stein/admin/ssh-configuration.html>. Accedido: 2020-01-22.
- [29] OpenStack. Container networking. <https://docs.openstack.org/openstack-ansible/rocky/reference/architecture/container-networking.html>. Accedido: 2019-07-05.
- [30] OpenStack. Create and manage networks. <https://docs.openstack.org/horizon/latest/user/create-networks.html>. Accedido: 2019-08-03.
- [31] OpenStack. Deployment guide stein. <https://docs.openstack.org/project-deploy-guide/openstack-ansible/stein/index.html>. Accedido: 2020-01-22.
- [32] OpenStack. Drop selinux support for centos 7. <https://review.opendev.org/#/c/603860/>. Accedido: 2019-07-05.
- [33] OpenStack. Flush all of the cache in memcached issue. <https://bugs.launchpad.net/openstack-ansible/+bug/1783423>. Accedido: 2020-01-25.
- [34] OpenStack. Get images. <https://docs.openstack.org/image-guide/obtain-images.html>. Accedido: 2019-08-04.
- [35] OpenStack. Introduction. <https://docs.openstack.org/project-team-guide/introduction.html>. Accedido: 2019-06-15.
- [36] OpenStack. Keystone architecture. <https://docs.openstack.org/keystone/latest/getting-started/architecture.html>. Accedido: 2019-06-20.
- [37] OpenStack. Launch and manage instances. <https://docs.openstack.org/horizon/latest/user/launch-instances.html>. Accedido: 2019-08-03.

- [38] OpenStack. Maintenance tasks. <https://docs.openstack.org/openstack-ansible/latest/admin/maintenance-tasks.html>. Accedido: 2020-01-12.
- [39] OpenStack. Major upgrades. <https://docs.openstack.org/openstack-ansible/train/admin/upgrades/major-upgrades.html>. Accedido: 2020-01-25.
- [40] OpenStack. Major upgrades. <https://docs.openstack.org/openstack-ansible/rocky/admin/upgrades/major-upgrades.html>. Accedido: 2020-01-25.
- [41] OpenStack. Manage flavors. <https://docs.openstack.org/horizon/latest/admin/manage-flavors.html>. Accedido: 2019-08-03.
- [42] OpenStack. Manage projects and users. <https://docs.openstack.org/horizon/latest/admin/manage-projects-and-users.html>. Accedido: 2019-08-03.
- [43] OpenStack. Memcached. <https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-memcached.html>. Accedido: 2019-06-20.
- [44] OpenStack. Message queuing. <https://docs.openstack.org/security-guide/messaging.html>. Accedido: 2019-06-20.
- [45] OpenStack. Migrate instances. <https://docs.openstack.org/nova/stein/admin/migration.html>. Accedido: 2020-01-22.
- [46] OpenStack. Minor version upgrade. <https://docs.openstack.org/openstack-ansible/stein/admin/upgrades/minor-upgrades.html>. Accedido: 2020-01-24.
- [47] OpenStack. Networking concepts. <https://docs.openstack.org/arch-design/design-networking/design-networking-concepts.html>. Accedido: 2019-06-15.
- [48] OpenStack. Network namespaces. <https://docs.openstack.org/neutron/pike/admin/intro-network-namespaces.html>. Accedido: 2020-01-08.
- [49] OpenStack. Openstack-ansible documentation. <https://docs.openstack.org/openstack-ansible/latest/>. Accedido: 2019-07-05.

- [50] OpenStack. The openstack marketplace. <https://www.openstack.org/marketplace/distros/>. Accedido: 2019-07-05.
- [51] OpenStack. Openstack survey report. <https://www.openstack.org/analytics>. Accedido: 2020-02-09.
- [52] OpenStack. openstack_user_config settings reference. <https://docs.openstack.org/openstack-ansible/queens/reference/inventory/openstack-user-config-reference.html>. Accedido: 2019-06-20.
- [53] OpenStack. Scaling your environment. <https://docs.openstack.org/openstack-ansible/queens/admin/scale-environment.html>. Accedido: 2020-01-24.
- [54] OpenStack. Scaling your environment. <https://docs.openstack.org/openstack-ansible/stein/admin/scale-environment.html>. Accedido: 2020-01-24.
- [55] OpenStack. Scenario - using open vswitch. https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-openvswitch.html. Accedido: 2020-01-12.
- [56] OpenStack. Settings reference. <https://docs.openstack.org/horizon/queens/configuration/settings.html>. Accedido: 2019-08-04.
- [57] OpenStack. Software. <https://www.openstack.org/software/>. Accedido: 2019-06-15.
- [58] OpenStack. Train series release notes. <https://docs.openstack.org/releasenotes/openstack-ansible/train.html>. Accedido: 2020-01-29.
- [59] OpenStack. Troubleshooting. <https://docs.openstack.org/openstack-ansible/latest/admin/troubleshooting.html>. Accedido: 2020-01-12.
- [60] OpenStack. Verify openstack-ansible cloud. <https://docs.openstack.org/openstack-ansible/latest/admin/openstack-firststrun.html>. Accedido: 2020-01-12.
- [61] OpenStack. Welcome to the puppet openstack guide! <https://docs.openstack.org/puppet-openstack-guide/latest/>. Accedido: 2019-07-05.
- [62] Oracle. What are hypervisors? https://docs.oracle.com/cd/E50245_01/E50249/html/vmcon-hypervisor.html. Accedido: 2019-06-20.

- [63] Ben Silverman and Michael Solberg. *Openstack for Architects*, chapter 3 Planning for Failure and Success, page 37. Packt Publishing, 2nd edition, 2018.
- [64] Murugiah Souppaya, John Morello, and Karen Scarfone. SP 800-190, Application Container Security Guide. Special publication, National Institute of Standards & Technology, 2017.
- [65] OpenStack Team. Red Hat OpenStack Platform 9 Networking Guide. Networking guide, Red Hat, 2019.
- [66] VMware. Virtualization. <https://www.vmware.com/solutions/virtualization.html>. Accedido: 2019-06-20.
- [67] Sage Weil, Scott Brandt, Ethan Miller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. 11 2006.
- [68] Sage Weil, Andrew Leung, Scott Brandt, and Carlos Maltzahn. Rados: A scalable, reliable storage service for petabyte-scale storage clusters. 01 2007.
- [69] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1), May 2010.

ANEXOS

Anexo A

Bitácora del proyecto

El trabajo tuvo una duración de 18 meses, iniciando en Agosto de 2018 y extendiéndose hasta Febrero de 2020. Originalmente el principal objetivo consistía en investigar los distintos modos de enrutamiento dentro de un Datacenter basado en OpenStack. Esto implicaba desplegar la plataforma para tener un ambiente de pruebas, lo cual se estimaba como alcanzable en 5 meses. En la práctica, obtener un primer ambiente funcional llevó mucho más tiempo debido a dificultades no anticipadas. Por esta razón se decidió cambiar el enfoque del proyecto reduciendo el tiempo de análisis del módulo de red para prestar mayor detalle al diseño de arquitectura y aspectos de gestión de OpenStack. A continuación se realiza un breve recorrido sobre el desarrollo del proyecto, mencionando los objetivos que se alcanzaron cronológicamente y referenciando las secciones o capítulos del trabajo que profundizan sobre cada punto.

Agosto - Setiembre 2018: en el inicio del proyecto se comenzó a relevar la tecnología teniendo como único respaldo la documentación oficial. Al tratarse de una primera experiencia de despliegue con OpenStack a nivel académico, no existía ningún precedente dentro de la Udelar para tener como referencia. Planteado esto, se comenzó por entender el funcionamiento de la plataforma a nivel macro (sección 2.2) utilizando ambientes locales meramente de aprendizaje desplegados con DevStack (sección 2.2.5). Además se debió decidir qué versión y con qué mecanismo instalar OpenStack (sección 2.4).

Octubre - Diciembre 2018: se planteó realizar una primera instalación de OSA en un ambiente local. Siguiendo la guía de despliegue de OSA se decidió una arquitectura simplificada de 4 nodos y virtualizada con VirtualBox. En este punto se esperaba tener un instalación funcional local en un periodo de 1 mes de pruebas. La realidad fue que surgieron múltiples problemas al comenzar la ejecución de las playbooks con Ansible, en donde por cada problema se investigaba su origen y se realizaban modificaciones a nivel de los archivos de configuración, de la arquitectura

virtualizada o sobre los recursos creados por la propia instalación, para reintentar el proceso. Estos ciclos de prueba y error se extendieron durante dos meses, ayudando a comprender mejor diferentes aspectos de OpenStack Ansible. Sobre el final de esta etapa se llegó a la conclusión de que era necesario un ambiente de desarrollo compartido y con mejores prestaciones debido a que cada ciclo de ejecución consumía prácticamente la totalidad de los recursos durante varias horas. Además permitiría mejorar la metodología de trabajo uniformizando las pruebas gracias a la eliminación de las diferencias entre los ambientes locales.

Enero 2019: el ambiente de desarrollo mencionado fue implementado por un conjunto de 4 máquinas virtuales alojadas en un servidor del INCO con el fin de replicar el ambiente local. A la hora de realizar las configuraciones requeridas, surgió la necesidad de modificar los recursos de red y almacenamiento de las VMs. Esta tarea se vio imposibilitada por no tener permisos directamente sobre el servidor físico. Como consecuencia se intentó virtualizar todo el ambiente dentro de una sola máquina virtual, lo que implicaba el uso de VirtualBox dentro de una instancia de KVM. Esta virtualización anidada generó severos inconvenientes al intentar acceder a una instancia para los cuales no se encontró una solución. Esto apuntaba a que para trabajar en un ambiente de desarrollo era necesario contar con ciertos permisos en el servidor físico.

Febrero - Marzo 2019: mientras se buscaba una solución para el ambiente remoto, se decidió intentar avanzar con las pruebas en ambientes locales. En todo este periodo de 2 meses se lograron grandes avances en la ejecución de las playbooks dado que 2 de las 3 se completaron correctamente (sección 2.4.3). Finalmente en esta etapa de pruebas a nivel local surgió un problema de falta de recursos que impedía finalizar la instalación. La arquitectura utilizada por más simple que fuera, quedaba demasiado grande para las computadoras personales que se estaban utilizando, generando por ejemplo que algunos servicios se vieran interrumpidos abruptamente por falta de memoria RAM.

Abril - Junio 2019: debido a la falta de recursos, la idea de utilizar un servidor físico dedicado cobró relevancia nuevamente. En esta etapa se obtuvieron los permisos de administrador en el servidor renata del INCO (sección C.1). Como consecuencia de este cambio surgieron nuevos puntos a resolver, como: el acceso al servidor, la salida a Internet desde el mismo, de qué forma utilizar los recursos para la instalación de OpenStack y la necesidad de adquirir conocimiento con el hipervisor KVM (capítulo 3). Gracias a la gran cantidad de recursos del servidor y a los conocimientos adquiridos hasta el momento, completar una instalación funcional llevó aproximadamente dos meses. Finalmente el proceso completo para alcanzar el primer objetivo llevó el doble del tiempo estimado (10 meses), dejando en claro que

instalar y configurar una plataforma de Datacenter no es para nada trivial.

Junio - Julio 2019: esta etapa consistió en documentar todo lo aprendido y experimentado hasta el momento.

Agosto - Setiembre 2019: durante este período se realizó una evaluación de la arquitectura, con el fin de establecer posibles mejoras y acercarse a la estructura de un ambiente de producción (sección 3.3). Para esto se llevó a cabo un relevamiento en la documentación de OpenStack Ansible y una videoconferencia con un experto en la plataforma. Como resultado se decidió agregar un nuevo nodo de cómputo y se evaluó la posibilidad de utilizar Ceph como tecnología para el almacenamiento en lugar de LVM (sección 2.1.7). De acuerdo a distintas fuentes y por el funcionamiento de dicha tecnología se concluyó que era más adecuada para un ambiente de producción. Estos cambios planteados requirieron modificaciones tanto en la arquitectura base como en los archivos de configuración utilizados. Finalmente se obtuvo una instalación funcional más robusta.

Setiembre - Octubre 2019: el principal objetivo en esta etapa fue la gestión de OpenStack. Inicialmente se planteó investigar y realizar actualizaciones a versiones mayores y agregar y remover nodos físicos (cómputo, almacenamiento e infraestructura) del Datacenter. Dentro de la documentación oficial existen guías para realizar estas acciones en donde se presenta como un proceso medianamente sencillo. En la ejecución de las mismas surgieron múltiples problemas comentados en el capítulo 4.

Noviembre 2019: con el objetivo de encaminar el análisis del módulo de red, se decidió ajustar la arquitectura diseñada hasta el momento para asemejarse aún más a los ambientes de producción descritos en la documentación oficial. Para esto fueron segmentadas mediante VLANs algunas de las subredes de comunicación entre los nodos físicos y se agregó un nuevo nodo virtual que emula el comportamiento de un router TOR. Debido a inconvenientes inesperados que provocaron la falla durante la instalación de la versión Queens, este nuevo ambiente fue desplegado utilizando la última versión estable disponible (Stein).

Diciembre 2019 - Enero 2020: finalmente se realizó el estudio del módulo de red, analizando cómo resuelven el routing los drivers Linux Bridge y OVS del plugin ml2. Para cada uno de estos se diseñaron cuatro escenarios que buscan cubrir las posibles combinaciones de tráfico entre instancias, ya sea dentro (horizontalmente) o fuera (verticalmente) del Datacenter (capítulo 5).

Enero - Febrero 2020: durante estos meses finales se formalizó y detalló toda la documentación creada hasta el momento.

Con el fin de ilustrar gráficamente el esfuerzo realizado en cada una de las tareas mencionadas anteriormente, en la figura A.1 se presenta una distribución del tiempo

dedicado a cada una de estas. En la misma se aprecia que alcanzar un ambiente funcional consumió aproximadamente el 50 % de la duración del proyecto.

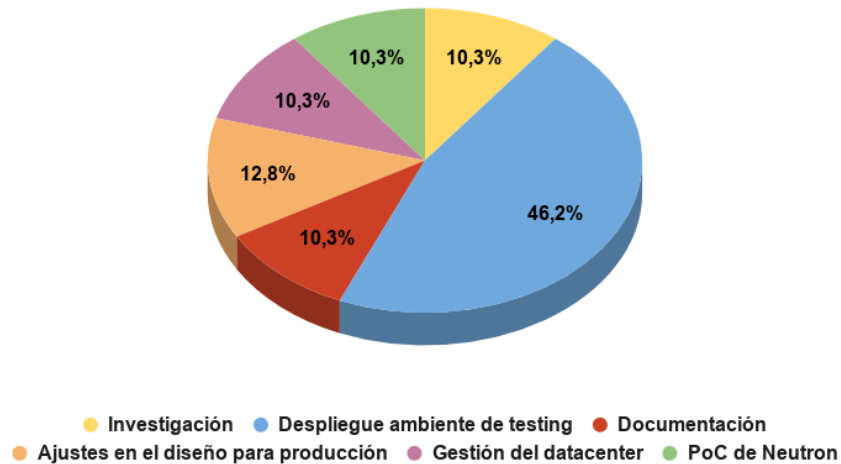


Figura A.1: Distribución del tiempo por tarea

A.1. Inconvenientes

Durante los procesos de instalación de OSA en las versiones utilizadas y en las distintas operaciones de gestión surgieron varios problemas para los cuales se tuvieron que aplicar algunos cambios con el objetivo de encontrar una solución. Algunos de los inconvenientes presentados ocurrieron en la instalación de Queens (anexo C.2) mientras que otros son problemas generales.

Bloqueo de paquetes

En los servidores virtuales y el servidor físico las reglas por defecto del firewall de CentOS bloquean tanto el tráfico utilizado para interconectar los servicios de OpenStack como el empleado para las conexiones con redes externas. Para solucionar esto de forma momentánea se eliminaron estas reglas con los comandos:

```
# iptables -D INPUT -j REJECT --reject-with icmp-host-prohibited  
# iptables -D FORWARD -j REJECT --reject-with icmp-host-prohibited
```

Módulo de seguridad SELinux

Como se mencionó en la sección C.2.2.4 del anexo C.2, OSA ha perdido el mantenimiento de este módulo de seguridad, por lo cual fue necesario aplicar los

parches realizados para la siguiente versión (Rocky) a la utilizada en la instalación (Queens) para discontinuar el uso de dicho módulo.

Percona-release

Durante la instalación de la playbook setup-infrastructure se detecta el siguiente error a la hora de instalar los componentes del contenedor de galera :

```
warning: /var/cache/yum/x86_64/7/percona-release-x86_64/packages/qpress-11-1.e17.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID 8507efa5: NOKEY
```

```
The GPG keys listed for the \"Percona-Release YUM repository - x86_64\" repository are already installed but they are not correct for this package.
```

```
Check that the correct key URLs are configured for this repository.
```

```
Failing package is: qpress-11-1.e17.x86_64
```

```
GPG Keys are configured as: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-Percona
```

La solución para este problema es actualizar el paquete percona-release, dentro del contenedor de galera siguiendo los siguientes pasos:

1. Acceder por ssh al nodo infra1

```
# ssh root@10.0.1.11
```

2. Listar los contenedores LXC existentes hasta el momento y obtener el nombre del contenedor pertinente:

```
# lxc-ls galera
```

3. Acceder a dicho contenedor:

```
# lxc-attach -n infra1_galera_container-15357d7d
```

4. Actualizar el paquete mencionado

```
# yum upgrade percona-release -y
```

5. Volver a ejecutar la playbook setup-infrastructure.

Subred reservada

Debido a que no se encuentra especificado en la documentación oficial de OpenStack-Ansible, en las primeras instalaciones realizadas la subred definida para la red de tunelización (VXLAN) fue la 10.0.3.0/24. Esto generaba grandes

inconsistencias de red que no tenían una causa identificada claramente. Luego de un extenso análisis de la situación se logró determinar que la red creada internamente por OpenStack para la comunicación entre contenedores LXC utiliza dicho CIDR. En función de esta observación es que se especificó el rango 10.0.10.0/24 para la red VXLAN.

Versiones de librerías y SO

Al tratarse de una herramienta de gran magnitud, OpenStack depende de una extensa cantidad de librerías externas. Esto genera que sea sumamente relevante tener una correcta gestión de versiones de las mismas. Como fue analizado en el proceso de instalación obtenido de la documentación oficial, uno de los pasos requeridos consiste en obtener las últimas versiones de los repositorios del SO base. Esto implica que al momento de realizar nuevas instalaciones de OpenStack, si las playbooks utilizadas no especifican la versión de los paquetes, estos podrían diferir impactando en la estabilidad del proceso. A modo de ejemplo se detallan dos casos concretos que generaron problemáticas durante el trabajo.

- En abril de 2019 el proceso de instalación se vio perjudicado por un bug introducido en la versión aria2-1.34.0-4.el7 de la herramienta de descarga Aria2 [5]. Esto implicó general un parche local de forma temporal hasta que una solución oficial fue liberada semanas después.
- Las nuevas instalaciones realizadas luego de la liberación de la versión 7.7 de CentOS (6 de agosto de 2019), presentaron problemas con el servidor de repositorios del Datacenter. Particularmente el servidor mantenía en cache los repositorios de la nueva versión mientras que el resto de los servicios solicitaba librerías de la versión instalada en los nodos base (7.6). Esto generaba fallas en el proceso de instalación de todos los contenedores. La única solución encontrada fue actualizar todos los nodos a la última versión del SO.

Luego de un tiempo de alcanzar un ambiente estable basado en la versión Queens junto con Ceph como backend de almacenamiento, donde realizar una nueva instalación era un proceso rutinario, surgieron problemas inesperados en la comunicación del módulo Cinder. Tras varios intentos buscando mitigar estos problemas, se decidió pasar a una nueva versión de OpenStack con el fin de poder avanzar con el proyecto. Es por esto que la versión utilizada para instalar la arquitectura diseñada para producción es Stein (anexo C.3)

Soporte para CentOS

Si bien en la documentación oficial se menciona el soporte de los sistemas operativos Ubuntu, CentOS y openSUSE, la experiencia adquirida demuestra que varios de los bugs y problemas enfrentados no se encuentran reportados para Ubuntu. A su vez la mayor parte de las guías, ejemplos de testing y producción, soporte de plugins, entre otros, sólo se encuentran detallados para este último. Esto aumenta la dificultad durante todo el proceso debido al requerimiento de utilizar un SO basado en RedHat.

Problema con container-setup.sh

En la ejecución del script `container-setup.sh` ubicado en `/opt` de los contenedores del nodo de infraestructura ocurrió un error reportado en [14]. Este error se da debido a que el código está orientado a sistemas Ubuntu y su solución fue seguir el commit de la referencia mencionada.

Repositorio

Durante los procesos de instalación surgió en reiteradas ocasiones un inconveniente con el repositorio del paquete `epel-lxc_hosts` en donde la URL base que se configura (`http://download.fedoraproject.org/pub/epel/7/x86_64/repodata/repomd.xml`) generaba una redirección a un cache de la Facultad de Derecho (`https://espejito.fder.edu.uy/fedora/epel/7/x86_64/repodata/repomd.xml`). OSA utiliza la herramienta `apt-cacher-ng` como servidor cache de repositorios la cual no maneja correctamente la redirección originando un mensaje 500. Para solucionar el problema se obtuvo la URL original (`http://mirror.nextlayer.at/epel/7/x86_64/repodata/repomd.xml`) y se reemplazó en el archivo `/etc/yum.repos.d/epel-lxc_hosts.repo` en el nodo de infraestructura.

Anexo B

Virtualización con KVM

Para crear la arquitectura de nodos se utilizó el virtualizador KVM, debido a que es la tecnología de virtualización utilizada normalmente en los servidores del INCO. Con el fin de facilitar la interacción con KVM a través de una interfaz gráfica, se utilizó el programa virt-manager.

B.1. Utilización virt-manager

B.1.1. Conexión remota

Dentro del virt-manager lo primero a realizar es configurar una nueva conexión desde el menú Archivo ->Añadir conexión... de la siguiente forma:

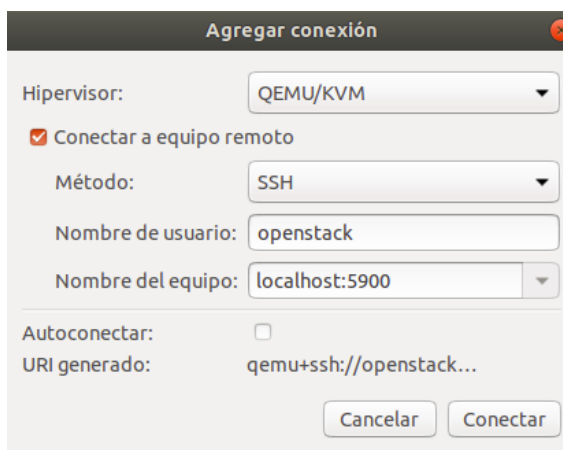


Figura B.1: Nueva conexión en virt-manager.

Como un host remoto está a dos conexiones SSH del servidor Renata, la configuración que se muestra en la figura B.1 no será suficiente. Para el correcto funcionamiento se debe crear una regla de forwarding que envíe todas las acciones

realizadas por el virt-manager hacia el host lulu.fing.edu.uy el cual tiene acceso al servidor. Para lograr esto se debe ejecutar:

```
ssh -L 5900:192.168.60.242:22 <usuario_fing>@lulu.fing.edu.uy
```

El número de puerto utilizado puede ser cualquiera que no esté siendo utilizado y no sea privilegiado.

El orden adecuado para conectarse al servidor mediante virt-manager es:

1. Crear la conexión ssh indicada.
2. Iniciar virt-manager.
3. Inicializar la conexión. En este paso se puede llegar a requerir la contraseña del usuario del servidor renata desde la consola que esté ejecutando el manager.

B.1.2. Creación de una red

A continuación se presenta un paso a paso de como crear una red con la herramienta virt-manager. Se debe ir al menú Editar ->Detalles de la conexión. Luego como se muestra en la imagen en la pestaña de redes virtuales seleccionar el icono (+).

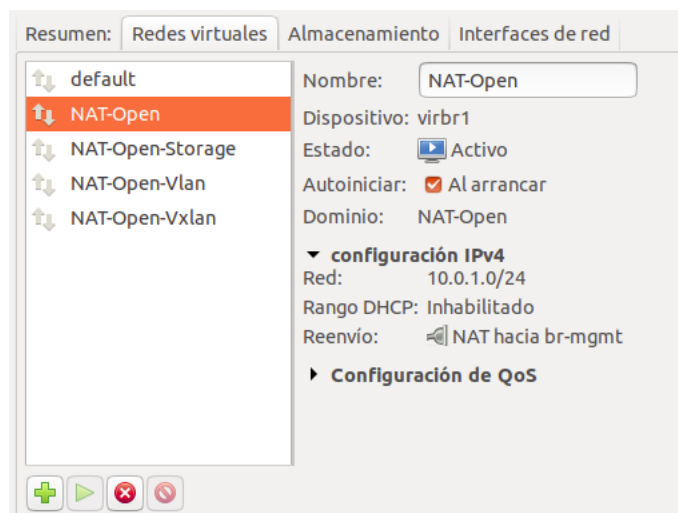


Figura B.2: Configuración de redes virtuales en virt-manager.

Al presionar el botón para agregar un nueva red, desplegará en pantalla un wizard. Mediante el mismo se muestra la creación de una red NAT-Open a modo de ejemplo:

1. Seleccionar el nombre de la red

Crear una red virtual
Etapa 1 of 4

Elegir un nombre para su máquina virtual:

Nombre de la Red:

Ejemplo: red1

2. Seleccionar la subred y el rango para el DHCP

Crear una red virtual
Etapa 2 of 4

Elegir el espacio de dirección **IPv4** para la red virtual:

Habilitar definición de un espacio de dirección de red IPv4^o

Red:

Consejo: La red debería ser seleccionada desde alguno de los rangos de dirección privada IPv4 . Por ejemplo: 10.0.0.0/8 o 192.168.0.0/16

Puerta de enlace: 10.0.1.1
Tipo: Privado

Habilitar DHCPv4

Inicio:

Fin:

Habilitar la definición de ruta estática

3. No habilitar direcciones IPv6

Crear una red virtual
Etapa 3 of 4

Elegir **IPv6** espacio de dirección para red virtual:

Habilitar definición de espacio de dirección de red IPv6

4. Seleccionar el tipo de red virtual

Crear una red virtual
Etapa 4 of 4

Conectado a una **red física**:

Red virtual aislada

Reenvío a la red física

Destino:

Modo:

Reserva con los VF de un dispositivo SR-IOV.

Lista de dispositivos:

Activar encaminamiento o red interna IPv6
Si la dirección de red IPv6 **no** se especifica, se habilitará el encaminamiento interno IPv6 entre las máquinas virtuales. El encaminamiento IPv4 interno se habilita de forma predeterminada.

Nombre de dominio DNS:

B.1.3. Crear nodo

Se detalla el paso a paso de cómo crear una instancia virtual.

1. En el presente ejemplo se crea una VM suponiendo que se instalará el SO.

Creación de una máquina virtual nueva
Etapa 1 of 5

Conexión: KVM localhost:5900

Elija cómo le gustaría instalar el sistema operativo

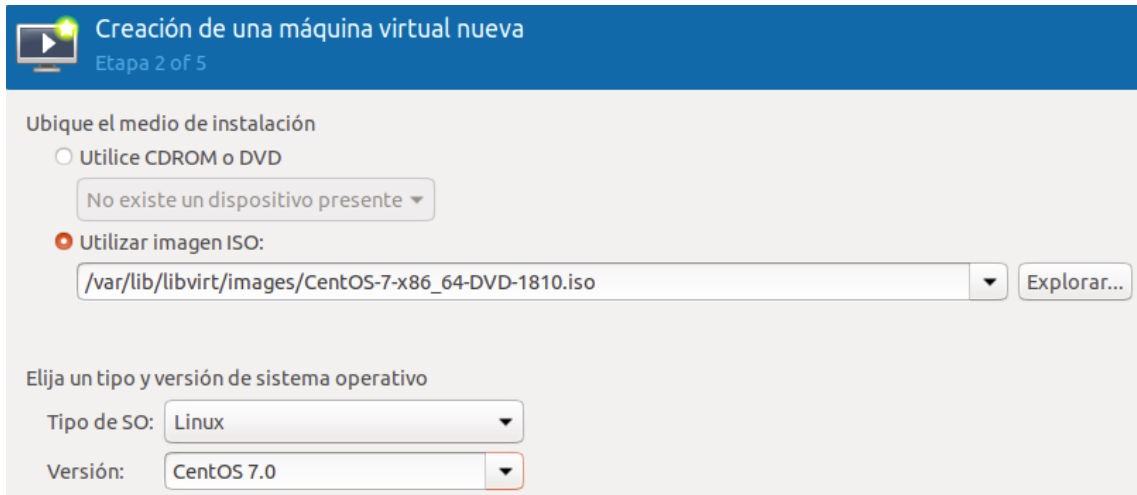
Medio de instalación local (Imagen ISO ó CDROM)

Instalación por Red (HTTP, FTP, ó NFS)

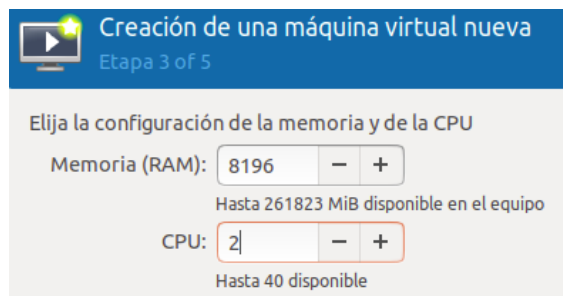
Arranque por Red (PXE)

Importar imagen de disco existente

2. Seleccionar la imagen a utilizar y el tipo de SO. La misma deberá estar en un directorio del servidor físico.

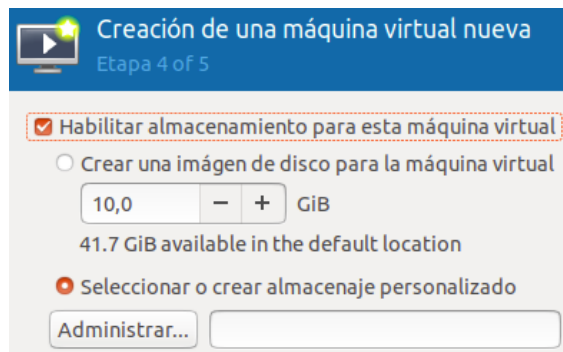


3. Seleccionar RAM y CPUs.



4. Para el almacenamiento se deberá crear un nuevo volumen de la siguiente forma:

4.1. Elegir la segunda opción y presionar Administrar.



4.2. Esto desplegará una ventana mostrando directorios del sistema de archivos de renata. Se puede elegir un volumen existente o crear uno nuevo con el botón (+).

Elija un volumen de Almacenamiento

default
Directorio del Sistema de Archivos

imagenes
Directorio del Sistema de Archivos

VMs
Directorio del Sistema de Archivos

Tamaño: 27820.79 GiB Free / 116.89 GiB In Use

Ubicación: /datos/VMs

Volúmenes + ↻ -

Volúmenes	Tamaño	Formato
centos7.0-clone-1-clone-clone-1.qcow2	40.00 GiB	qcow2

4.3. Al crearlo se debe especificar el nombre, el tipo y la capacidad.

+ **Crear volumen de almacenamiento**

Crear una unidad de almacenamiento para ser utilizada directamente por la máquina virtual.

Nombre:

Formato:

▶ Almacén de respaldo

Cuota de volumen de almacenamiento
VMs's espacio disponible: 27820.79 GiB

Máxima capacidad: - + GiB

5. Luego se debe ingresar el nombre de la máquina y seleccionar la red que conecta el host directamente al bridge del servidor físico en lugar de la red NAT-Open para que el host tenga conexión a Internet durante la instalación y poder realizar las primeras configuraciones en el mismo.

▶ **Creación de una máquina virtual nueva**
Etapa 5 of 5

Listo para iniciar la instalación

Nombre:

SO: CentOS 7.0

Instalar: CDROM/ISO local

Memoria: 8196 MiB

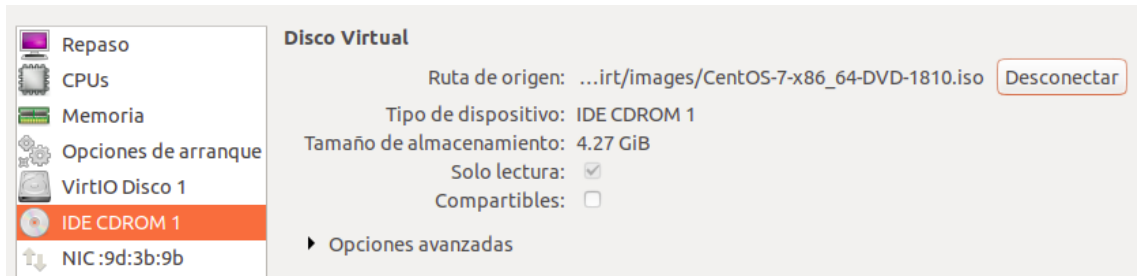
CPUs: 2

Almacenamiento: /datos/VMs/centos7.0Deploy.qcow2

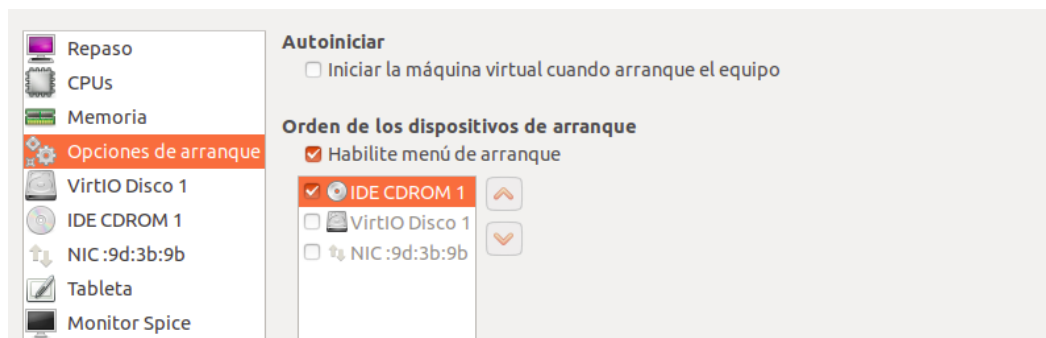
Personalizar configuración antes de instalar

▼ Selección de Red

6. Se debe verificar en la opción IDE CDROM 1 que esté correctamente asignada la imagen a utilizar para instalar el SO.



7. Verificar en las opciones de arranque que el IDE CDROM 1 esté en primer lugar para que la máquina inicie con la imagen seleccionada.



8. Finalmente al confirmar todos los cambios se lanzará la VM creada y se instalará el SO.

Anexo C

Instalación

C.1. Ambiente de trabajo

C.1.1. Hardware utilizado

Para realizar la instalación de OpenStack se utilizó un servidor físico (denominado renata) alojado en el Instituto de Computación de la Facultad de Ingeniería (INCO). El mismo cuenta con una amplia cantidad de recursos destacando sus 40 procesadores virtuales, 128 GB de RAM y 40 TB de disco duro. Se aloja en una red privada del INCO en donde para salir a Internet se debe pasar por un proxy, provocando algunas limitaciones que luego se mencionan.

En el servidor fueron necesarias las siguientes configuraciones de red:

1. Creación de un bridge el cual será utilizado por la interfaz física del servidor y por los distintos NAT que se deben crear con KVM para la arquitectura que se virtualiza. Para esto se creó la interfaz br-mgmt con la siguiente configuración:

```
DEVICE="br-mgmt"  
BOOTPROTO="none"  
IPADDR="192.168.60.242"  
PREFIX="24"  
GATEWAY="192.168.60.1"  
DNS1="192.168.60.230"  
ONBOOT="yes"  
TYPE="Bridge"  
NM_CONTROLLED="no"
```

2. En la interfaz eno2 la cual tenía configurada la IP del bridge quedó de la siguiente forma:

```
BOOTPROTO=none
```

```

NAME=eno2
UUID=824cd835-662a-4d47-a148-512aec3dd237
DEVICE=eno2
ONBOOT=yes
BRIDGE="br-mgmt"
NM_CONTROLLED="no"

```

C.1.2. Conexión remota hacia el servidor renata

Dado que el servidor se encuentra en una red privada del INCO, para conectarse al mismo de forma remota se deben establecer algunas conexiones SSH. A continuación se detallan las conexiones y comandos utilizados.

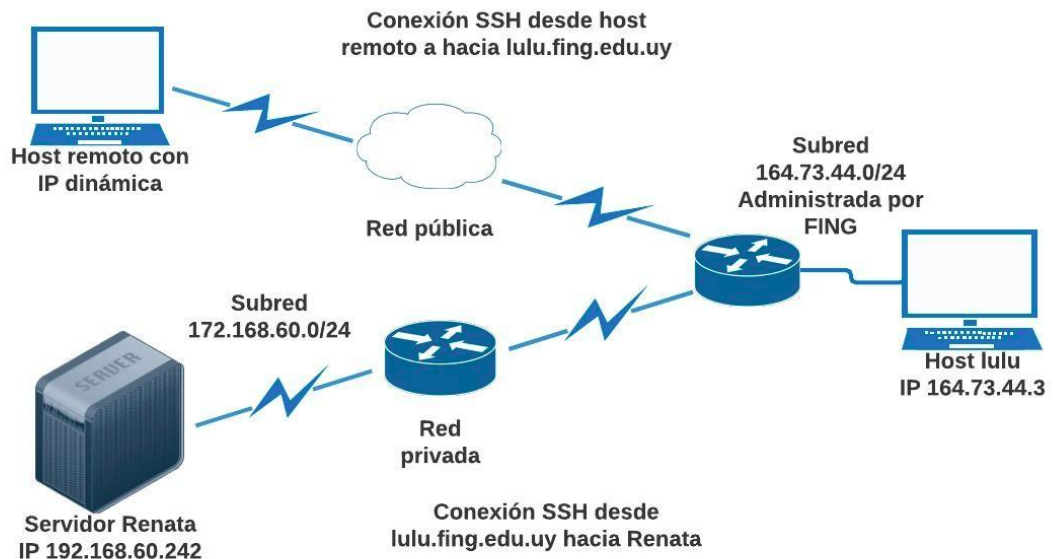


Figura C.1: Acceso remoto al servidor renata.

1. Como el servidor se encuentra en una red privada del INCO solo se puede acceder desde un host que se encuentre en una red interna de la FING, por ej: lulu.fing.edu.uy. Para esto ejecutar:

```
$ ssh usuario_fing@lulu.fing.edu.uy
```

2. Desde el host lulu para conectarse al servidor renata se debe ejecutar:

```
$ ssh openstack@192.168.60.242
```

C.1.3. Especificaciones servidor renata

La red en la que se encuentra el servidor Renata no cuenta con acceso a Internet. Para solucionar este problema se realizó un túnel ssh reverse desde el host `lulu.fing.edu.uy` al servidor Renata para establecer como proxy del mismo el host `proxy.fing.edu.uy`, siendo este último el proxy de la FING.

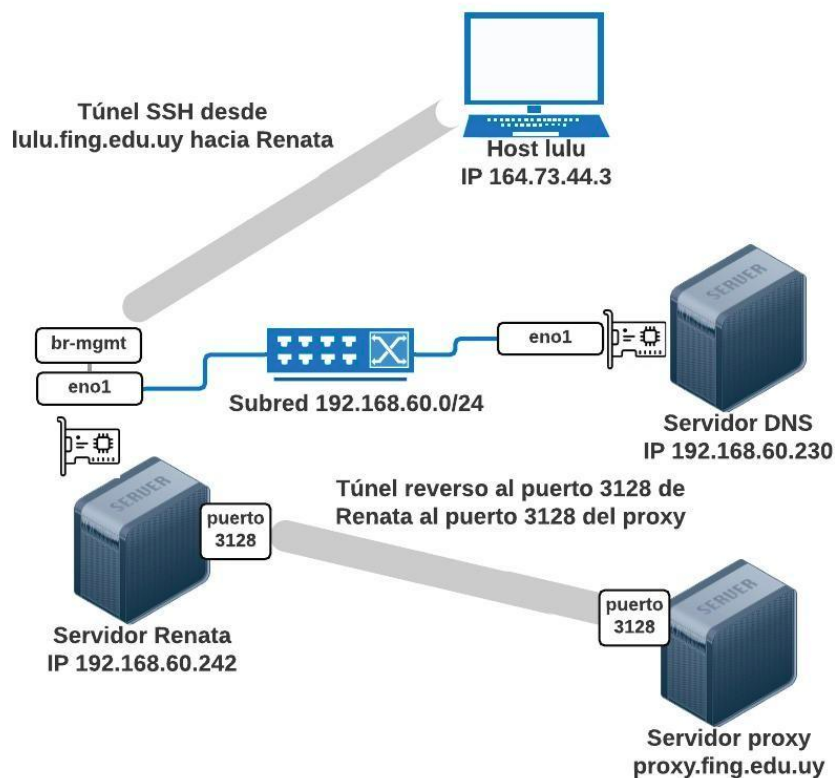


Figura C.2: Túnel reverse y esquema de servidores.

El funcionamiento de esta configuración se detalla a continuación:

- El host `lulu.fing.edu.uy` inicia una conexión SSH con renata en donde indica que cree una conexión inversa con el proxy de la FING.
- Al establecer la conexión, el servidor renata crea el túnel inverso en donde envía todos los paquetes destinados al puerto 3128 al servidor proxy en el puerto 3128.
- Finalmente para que renata tenga acceso a Internet se asignan las variables de entorno `http_proxy` y `https_proxy` con el valor `http://localhost:3128`. De esta forma todo el tráfico http será enviado al puerto 3128 local y gracias al túnel reverse todos los pedidos http serán enviados al proxy logrando tener acceso a Internet.

Para lograr esto se debe ejecutar el siguiente comando al iniciar la conexión SSH desde lulu hacia renata:

```
$ ssh -R 3128:proxy.fing.edu.uy:3128 openstack@192.168.60.242
```

Como se muestra en la figura C.2 el servidor DNS utilizado se encuentra en la misma red local.

C.1.4. Acceso al exterior desde nodos

Los nodos virtualizados en el servidor renata con la configuración por defecto no tendrán acceso a Internet. Esto sucede por que al realizar un pedido http o https será enviado al gateway de estos nodos virtuales que es la IP 10.0.1.1 de la interfaz virtual de renata, en donde solamente forwardea los paquetes recibidos en el puerto 3128 al proxy de la FING. Para solucionar esto basta con configurar las variables de entorno `http_proxy` y `https_proxy` en los nodos virtualizados con el siguiente valor: `http://10.0.1.1:3128`. Esto aplica para los 5 nodos utilizados en la instalación.

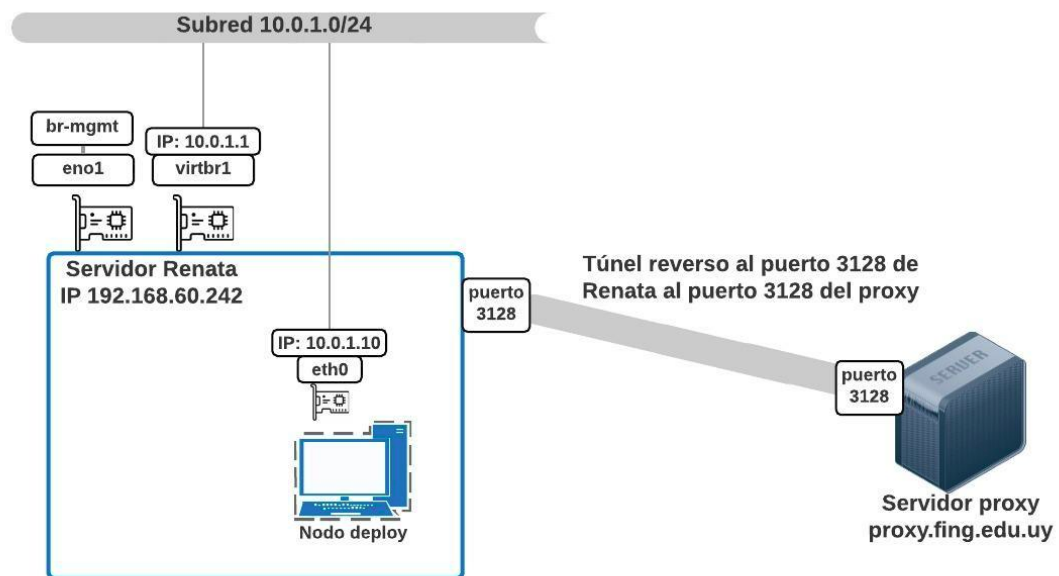


Figura C.3: Salida a Internet en los nodos de Openstack.

C.2. Instalación versión Queens

Queens es la décimo séptima versión liberada por OpenStack. La misma fue publicada el 28 de febrero del 2018. Junto a Pike y Ocata (ambas anteriores) son las versiones actuales en estado de mantenimiento extendido para utilizar en ambientes

de producción. La arquitectura utilizada para este escenario es la detallada en la sección 3.2 orientada a un ambiente de desarrollo.

C.2.1. Preparación de nodos

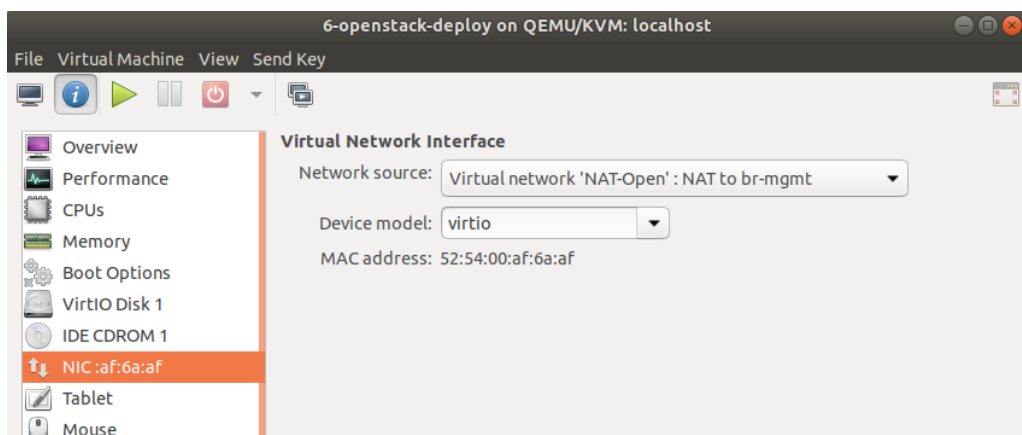
En la siguiente sección se detallan los pasos necesarios a seguir en cada uno de los nodos para iniciar con la instalación de OpenStack. Para realizar dicha configuración inicial será necesario que los nodos cuenten con conexión a Internet. En el ambiente de trabajo actual, esto es equivalente a verificar que las variables del proxy estén bien configuradas.

Deploy

1. Configurar la interfaz de red eth0 de la siguiente forma:

```
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.0.1.10
PREFIX=24
GATEWAY=10.0.1.1
DNS1=192.168.60.230
```

2. Asociar, en el virt-manager, la interfaz de red eth0 a la red virtual de management 10.0.1.0/24.



3. Por simplificación , cambiar el hostname de la máquina a 'deploy' en el archivo /etc/hostname y con el comando:

```
# hostname deploy
```

4. Configurar las variables de proxy en `/etc/environment` con los siguientes comandos:

```
# echo "http_proxy=http://10.0.1.1:3128" >> /etc/environment
# echo "https_proxy=http://10.0.1.1:3128" >> /etc/environment
# echo "HTTP_PROXY=http://10.0.1.1:3128" >> /etc/environment
# echo "HTTPS_PROXY=http://10.0.1.1:3128" >> /etc/environment
# source /etc/environment
```

Verificar el acceso a Internet mediante el comando:

```
# curl www.google.com.
```

5. Instalaciones necesarias:

- 5.1. Actualizar repositorios y reiniciar:

```
# yum upgrade -y
# reboot
```

- 5.2. Instalar repositorio OpenStack Queens:

```
# yum install -y https://rdoproject.org/repos/openstack-queens/
rdo-release-queens.rpm
```

- 5.3. Instalar herramientas auxiliares:

```
# yum install -y git ntp nano net-tools ntpdate openssh-server
python-devel sudo '@Development Tools'
```

6. Deshabilitar firewall de CentOS:

```
# systemctl stop firewalld
# systemctl mask firewalld
```

7. Configurar el servicio de NTP mediante Chrony, sustituyendo en el archivo `/etc/chrony.conf` las siguientes líneas:

```
server 0.south-america.pool.ntp.org
server 1.south-america.pool.ntp.org
server 2.south-america.pool.ntp.org
server 3.south-america.pool.ntp.org
```

Reiniciar el servicio mediante:

```
# systemctl restart chronyd
```

8. Clonar el repositorio de OpenStack-Ansible Queens

```
# git clone -b 17.1.10 https://git.openstack.org/openstack/openstack-ansible /opt/openstack-ansible
```

9. Preparar openstack-ansible:

```
# /opt/openstack-ansible/scripts/bootstrap-ansible.sh
```

Este script se utiliza para asegurar que Ansible y todas las dependencias necesarias para la instalación de OSA están instaladas en el nodo de deploy.

10. Copiar el contenido de configuración al /etc:

```
# cp -r /opt/openstack-ansible/etc/openstack_deploy /etc/
```

11. Crear la carpeta para logs de la instalación:

```
# mkdir /var/log/openstack
```

Infra1

1. Configuraciones de red:

- 1.1. Configurar los bridges br-mgmt, br-storage y br-vxlan.

```
DEVICE=br-mgmt                DEVICE=br-storage
BOOTPROTO=none                BOOTPROTO=none
IPADDR=10.0.1.11              IPADDR=10.0.2.11
PREFIX=24                      PREFIX=24
GATEWAY=10.0.1.1              ONBOOT=yes
DNS1=192.168.60.230           TYPE=Bridge
ONBOOT=yes                     NM_CONTROLLED=no
TYPE=Bridge
NM_CONTROLLED=no
```

```
DEVICE=br-vxlan
BOOTPROTO=none
IPADDR=10.0.10.11
PREFIX=24
ONBOOT=yes
TYPE=Bridge
NM_CONTROLLED=no
```

- 1.2. Configurar las interfaces eth0, eth1, eth2 y eth3.

```

TYPE="Ethernet"
BOOTPROTO=none
DEVICE=eth0
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br-mgmt

TYPE="Ethernet"
BOOTPROTO=none
DEVICE=eth1
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br-storage

TYPE="Ethernet"
BOOTPROTO=none
DEVICE=eth2
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br-vxlan

TYPE="Ethernet"
BOOTPROTO=none
DEVICE=eth3
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br-storage

```

2. Asociar, en el virt-manager, las interfaces de red:

2.1. eth0 a la red virtual de management 10.0.1.0/24.

2.2. eth1 a la red virtual de storage 10.0.2.0/24.

2.3. eth2 a la red virtual de VXLAN 10.0.10.0/24.

2.4. eth3 a la red virtual de VLAN 10.0.4.0/24.

3. Cambiar el hostname de la máquina a 'infra1' en el archivo `/etc/hostname` y con el comando:

```
# hostname infra1
```

4. Configurar las variables de proxy en la sesión pero no en el sistema, porque luego el nodo deploy configura el `/etc/environment` en todos los nodos.

```
# export http_proxy=http://10.0.1.1:3128
# export https_proxy=http://10.0.1.1:3128
# export HTTP_PROXY=http://10.0.1.1:3128
# export HTTPS_PROXY=http://10.0.1.1:3128
```

Verificar el acceso a Internet mediante el comando:

```
# curl www.google.com
```

5. Actualizar repositorios y reiniciar:

```
# yum upgrade -y
# reboot
```

6. Instalar herramientas auxiliares:

```
# yum install bridge-utils iputils lsof lvm2 ntp ntpdate openssh-  
server sudo tcpdump python net-tools nano
```

7. Deshabilitar el Network Manager:

```
# chkconfig NetworkManager off  
# chkconfig network on  
# service NetworkManager stop  
# service network start
```

8. Deshabilitar el SELinux, cambiando en `/etc/sysconfig/selinux`, "SELINUX=enforcing" por "SELINUX=disabled".

9. Habilitar bonding de interfaces y VLANs:

```
# echo 'bonding' >> /etc/modules-load.d/openstack-ansible.conf  
# echo '8021q' >> /etc/modules-load.d/openstack-ansible.conf
```

10. Configurar el servicio de NTP mediante Chrony, sustituyendo en el archivo `/etc/chrony.conf` las siguientes líneas:

```
server 0.south-america.pool.ntp.org  
server 1.south-america.pool.ntp.org  
server 2.south-america.pool.ntp.org  
server 3.south-america.pool.ntp.org
```

Reiniciar el servicio mediante:

```
# systemctl restart chronyd
```

11. Eliminar reglas de firewall que bloquean el tráfico:

11.1. Eliminar las reglas manualmente:

```
# iptables -D INPUT -j REJECT --reject-with icmp-host-prohibited  
# iptables -D FORWARD -j REJECT --reject-with icmp-host-  
prohibited
```

11.2. Exportar las reglas a un archivo:

```
# iptables-save > /etc/sysconfig/iptables
```

11.3. En cada reboot de la máquina, importar las reglas del archivo anterior:

```
# iptables-restore < /etc/sysconfig/iptables
```

Compute1

1. Configuraciones de red:

1.1. Configurar los bridges br-mgmt, br-storage y br-vxlan.

```
DEVICE=br-mgmt                DEVICE=br-storage
BOOTPROTO=none                BOOTPROTO=none
IPADDR=10.0.1.12              IPADDR=10.0.2.12
PREFIX=24                      PREFIX=24
GATEWAY=10.0.1.1              ONBOOT=yes
DNS1=192.168.60.230           TYPE=Bridge
ONBOOT=yes                     NM_CONTROLLED=no
TYPE=Bridge
```

```
DEVICE=br-vxlan
BOOTPROTO=none
IPADDR=10.0.10.12
PREFIX=24
ONBOOT=yes
TYPE=Bridge
NM_CONTROLLED=no
```

1.2. Configurar las interfaces eth0, eth1, eth2 y eth3.

```
TYPE="Ethernet"                TYPE="Ethernet"
BOOTPROTO=none                  BOOTPROTO=none
DEVICE=eth0                      DEVICE=eth1
ONBOOT=yes                       ONBOOT=yes
NM_CONTROLLED=no                 NM_CONTROLLED=no
BRIDGE=br-mgmt                    BRIDGE=br-storage
```

```
TYPE="Ethernet"                TYPE="Ethernet"
BOOTPROTO=none                  BOOTPROTO=none
DEVICE=eth2                      DEVICE=eth3
ONBOOT=yes                       ONBOOT=yes
NM_CONTROLLED=no                 NM_CONTROLLED=no
BRIDGE=br-vxlan
```

2. Asociar, en el virt-manager, las interfaces de red:

2.1. eth0 a la red virtual de management 10.0.1.0/24.

- 2.2. eth1 a la red virtual de storage 10.0.2.0./24.
 - 2.3. eth2 a la red virtual de VXLAN 10.0.10.0./24.
 - 2.4. eth3 a la red virtual de VLAN 10.0.4.0./24.
3. Cambiar el hostname de la máquina a 'compute1' en el archivo `/etc/hostname` y con el comando:

```
# hostname compute1
```

A partir de este punto, el procedimiento es idéntico al nodo infra1. Se deben realizar los puntos 4 al 11.

Storage1

1. Configuraciones de red:
 - 1.1. Configurar los bridges br-mgmt y br-storage.

DEVICE=br-mgmt	DEVICE=br-storage
BOOTPROTO=none	BOOTPROTO=none
IPADDR=10.0.1.13	IPADDR=10.0.2.13
PREFIX=24	PREFIX=24
GATEWAY=10.0.1.1	ONBOOT=yes
DNS1=192.168.60.230	TYPE=Bridge
ONBOOT=yes	NM_CONTROLLED=no
TYPE=Bridge	
NM_CONTROLLED=no	

- 1.2. Configurar las interfaces eth0 y eth1.

TYPE="Ethernet"	TYPE="Ethernet"
BOOTPROTO=none	BOOTPROTO=none
DEVICE=eth0	DEVICE=eth1
ONBOOT=yes	ONBOOT=yes
NM_CONTROLLED=no	NM_CONTROLLED=no
BRIDGE=br-mgmt	BRIDGE=br-storage

2. Asociar, en el virt-manager, las interfaces de red:
 - 2.1. eth0 a la red virtual de management 10.0.1.0/24.
 - 2.2. eth1 a la red virtual de storage 10.0.2.0./24.

3. Cambiar el hostname de la máquina a 'storage1' en el archivo `/etc/hostname` y con el comando:

```
# hostname storage1
```

A partir de este punto, el procedimiento es idéntico al nodo `infra1`. Se deben realizar los puntos 4 al 11.

12. Crear el volumen de LVM para utilizar Cinder:

- 12.1. Listar los devices en la máquina:

```
# lvmdiskscan
```

- 12.2. Formatear la pieza física de almacenamiento de 200 GB:

```
# pvcreate --metadatasize 2048 <physical_volume_device_path>
```

- 12.3. Crear el nuevo grupo de almacenamiento que será utilizado por OpenStack:

```
# vgcreate cinder-volumes <physical_volume_device_path>
```

- 12.4. Verificar que el grupo quedó creado correctamente:

```
# vgdisplay
```

HProxy1

1. Configurar las interfaces `eth0` y `eth1`.

```
TYPE=Ethernet
```

```
BOOTPROTO=none
```

```
DEFROUTE=yes
```

```
DEVICE=eth0
```

```
ONBOOT=yes
```

```
IPADDR=10.0.1.15
```

```
PREFIX=24
```

```
GATEWAY=10.0.1.1
```

```
DNS1=192.168.60.230
```

```
TYPE=Ethernet
```

```
BOOTPROTO=none
```

```
DEFROUTE=yes
```

```
DEVICE=eth1
```

```
ONBOOT=yes
```

```
IPADDR=192.168.60.160
```

```
PREFIX=24
```

2. Asociar, en el `virt-manager`, las interfaces de red:

- 2.1. `eth0` a la red virtual de management `10.0.1.0/24`.

- 2.2. `eth1` al bridge alojado en el servidor `renata` en la red `192.168.60.0/24`.

3. Cambiar el hostname de la máquina a 'haproxy1' en el archivo `/etc/hostname` y con el comando:

```
# hostname haproxy1
```

A partir de este punto, el procedimiento es similar al nodo `infra1`. Se deben realizar los puntos 4 al 11 con la excepción del punto 7 en donde se deshabilita el `NetworkManager`.

C.2.2. Configuración

Luego de completar la preparación de los nodos y verificar la conectividad entre los mismos, los últimos pasos antes de iniciar con la instalación de OpenStack son configurar los archivos que OSA utiliza y los requerimientos extras de la herramienta utilizada para la instalación.

C.2.2.1. Configuración claves SSH

Como se menciona en la sección de Ansible, el nodo de deploy requiere de una conexión SSH con cada uno de los servidores que componen el Datacenter, para poder configurar y operar directamente sobre cada uno de ellos. Para esto se utilizan un par de claves SSH público-privada con el fin de brindarle al nodo de deploy mayor flexibilidad al momento de acceder a los servidores. Se deberá propagar la clave del usuario `root` dado que es este usuario el que llevará a cabo la instalación. El comando que se debe ejecutar para crear las claves es el siguiente:

```
# ssh-keygen -t rsa -f ~/.ssh/id_rsa -N ""
```

Donde:

- `-t`: especifica el tipo de encriptación.
- `-f`: determina el archivo en donde quedará la clave privada (por defecto, la pública será igual con la extensión `.pub`).
- `-N ""` indica que no se utilizará ninguna passphrase en la clave.

Esto genera dos archivos (`id_rsa` y `id_rsa.pub`) donde el primero es la clave privada (que no se deberá compartir) y el segundo la clave pública que se copiará al resto de los servidores con los siguientes comandos:

```
# ssh-copy-id root@10.0.1.11
# ssh-copy-id root@10.0.1.12
# ssh-copy-id root@10.0.1.13
# ssh-copy-id root@10.0.1.15
```

Para verificar la configuración bastará con realizar un ssh a cada servidor desde el deploy con el usuario root, accediendo en forma directa al prompt de dicho servidor.

C.2.2.2. Archivos de configuración OSA

openstack_user_config.yml

A continuación se muestra las configuraciones utilizadas en el archivo, el significado de cada sección fue explicado en [2.4.2.3](#).

```
cidr_networks:
  container: 10.0.1.0/24
  tunnel: 10.0.10.0/24
  storage: 10.0.2.0/24

used_ips:
  - "10.0.1.1,10.0.1.20" # red de management
  - "10.0.2.1,10.0.2.20" # red de storage
  - "10.0.10.1,10.0.10.20" # red de vxlan

global_overrides:
  internal_lb_vip_address: 10.0.1.15
  external_lb_vip_address: 192.168.60.160

  tunnel_bridge: "br-vxlan"
  management_bridge: "br-mgmt"
  storage_bridge: "br-storage"

provider_networks:
  - network:
      group_binds:
        - all_containers
        - hosts
      type: "raw"
      container_bridge: "br-mgmt"
      container_interface: "eth1"
      container_type: "veth"
      ip_from_q: "container"
      is_container_address: true
      is_ssh_address: true
  - network:
      group_binds:
```

```

        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute
    type: "raw"
    container_bridge: "br-storage"
    container_type: "veth"
    container_interface: "eth2"
    container_mtu: "9000"
    ip_from_q: "storage"
- network:
    group_binds:
        - neutron_linuxbridge_agent
    container_bridge: "br-vxlan"
    container_type: "veth"
    container_interface: "eth10"
    container_mtu: "9000"
    ip_from_q: "tunnel"
    type: "vxlan"
    range: "1:1000"
    net_name: "vxlan"
- network:
    group_binds:
        - neutron_linuxbridge_agent
    container_bridge: "br-vlan"
    container_type: "veth"
    container_interface: "eth12"
    host_bind_override: "eth3"
    type: "flat"
    net_name: "flat"

### Infrastructure
# galera, memcache, rabbitmq, utility
shared-infra_hosts:
    infra1:
        ip: 10.0.1.11
# repository (apt cache, python packages, etc)
repo-infra_hosts:
    infra1:
        ip: 10.0.1.11
# load balancer

```

```
# Dedicated hardware is best for improved performance and security.
haproxy_hosts:
    balancer1:
        ip: 10.0.1.15
# rsyslog server
log_hosts:
    infra1:
        ip: 10.0.1.11

### OpenStack
# keystone
identity_hosts:
    infra1:
        ip: 10.0.1.11
# cinder api services
storage-infra_hosts:
    infra1:
        ip: 10.0.1.11
# glance
image_hosts:
    infra1:
        ip: 10.0.1.11
# nova api, conductor, etc services
compute-infra_hosts:
    infra1:
        ip: 10.0.1.11
# heat
orchestration_hosts:
    infra1:
        ip: 10.0.1.11
# horizon
dashboard_hosts:
    infra1:
        ip: 10.0.1.11
# neutron server, agents (L3, etc)
network_hosts:
    infra1:
        ip: 10.0.1.11
# nova hypervisors
compute_hosts:
    compute1:
```

```

        ip: 10.0.1.12
# cinder volume hosts
storage_hosts:
    storage1:
        ip: 10.0.1.13
        container_vars:
            cinder_backends:
                lvm:
                    volume_backend_name: LVM
                    volume_driver: cinder.volume.drivers.
                        lvm.LVMVolumeDriver
                    volume_group: cinder-volumes
                    iscsi_ip_address: "10.0.2.13"

```

Particularmente se resalta, en el último punto, la configuración de cinder necesaria para desplegar un backend de almacenamiento basado en LVM.

user_variables.yml

Debido a las limitaciones de red mencionadas anteriormente, es necesario configurar un proxy de salida que será propagado por ansible hacia todos los contenedores y hosts durante la instalación. Para esto se deben configurar las siguientes variables:

```

## Example environment variable setup:
## This is used by apt-cacher-ng to download apt packages:
proxy_env_url: http://10.0.1.1:3128/

## (1) This sets up a permanent environment, used during and after
    deployment:
no_proxy_env: "localhost,127.0.0.1,{{ internal_lb_vip_address }},{{
    external_lb_vip_address }},{% for host in groups['all_containers']
    %}{{ hostvars[host]['container_address'] }}{% if not loop.last %},{%
    endif %}{% endfor %}"
global_environment_variables:
    HTTP_PROXY: "{{ proxy_env_url }}"
    HTTPS_PROXY: "{{ proxy_env_url }}"
    NO_PROXY: "{{ no_proxy_env }}"
    http_proxy: "{{ proxy_env_url }}"
    https_proxy: "{{ proxy_env_url }}"
    no_proxy: "{{ no_proxy_env }}"
#

```

```
## (2) This is applied only during deployment, nothing is left after
deployment is complete:
```

```
deployment_environment_variables:
```

```
  http_proxy: "{{ proxy_env_url }}"
  https_proxy: "{{ proxy_env_url }}"
  no_proxy: "localhost,127.0.0.1,{{ internal_lb_vip_address }},{{
    external_lb_vip_address }},{% for host in groups['keystone_all
    '] %}{{ hostvars[host]['container_address'] }}{% if not loop.
    last %},{% endif %}{% endfor %}"
```

Además, para poder crear una imagen desde el Horizon remotamente se debe agregar sobre el final del archivo la siguiente directiva:

```
horizon_images_upload_mode: "legacy"
```

Si bien en [56] se menciona que el valor por defecto de esta variable es *legacy*, durante el proceso de instalación se detectó que es configurada como *direct*. La diferencia entre estos modos es que *legacy* permite subir archivos locales desde la máquina del usuario al servidor web de Horizon y luego de este hacia el módulo Glance. Por su parte *direct* evita esta sobrecarga de red y almacenamiento en el servidor web, conectado a través de una API al usuario con el módulo Glance. Sin embargo, esto último requiere de configuraciones extras como tener acceso al puerto 9292 (API de Glance) y un correcto uso de CORS.

cinder-volume.yml

En caso de utilizar un backend de storage LVM se debe indicar que este debe ser desplegado en metal, para esto se debe configurar el archivo `/etc/openstack_deploy/env.d/cinder-volume.yml` con lo siguiente:

```
container_skel:
  cinder_volumes_container:
    properties:
      is_metal: true
```

C.2.2.3. Generación de claves

Se deben configurar las passphrases requeridas por OpenStack durante su instalación y posterior uso. Esto se alcanza mediante:

```
# cd /opt/openstack-ansible
# ./scripts/pw-token-gen.py --file /etc/openstack_deploy/user_secrets.yml
```

C.2.2.4. Correcciones

SELinux

Durante el proceso de instalación se detectó un bug asociado a SELinux, el cual fue verificado en [24]. Debido a que en el proyecto OSA se dejó de mantener el uso de SELinux por falta de personal, se debió aplicar el commit indicado en [32] de la versión Rocky de OSA que desactiva por completo la utilización de este módulo. Concretamente el commit consiste en:

- Eliminar los siguientes archivos:

```
# rm /etc/ansible/roles/os_nova/files/osa-nova.te
# rm /etc/ansible/roles/os_nova/tasks/nova_selinux.yml
```

- Modificar el archivo `/etc/ansible/roles/os_nova/tasks/nova_post_install.yml` eliminando las siguientes líneas:

```
include_tasks: nova_selinux.yml
    when:
        - ansible_selinux.status == "enabled"
```

C.3. Instalación versión Stein

Stein es la décimo novena versión liberada por OpenStack. La misma fue publicada el 10 de abril del 2019. Junto a Train, liberada el 16 de octubre del 2019, son las versiones más recientes en estado estable para utilizar en ambientes de producción. La arquitectura utilizada para este escenario es la detallada en la sección 3.3 orientada a un ambiente de producción.

C.3.1. Preparación de nodos

En esta sección se detallan los pasos necesarios a seguir en cada uno de los nodos para iniciar con la instalación de OpenStack. Al igual que en la instalación previa es necesario contar con conexión a Internet en todos los nodos. La guía de referencia utilizada para preparar el ambiente se encuentra en [31].

Deploy

1. Configurar la interfaz de red `eth0` de la siguiente forma:

```
TYPE=Ethernet
BOOTPROTO=none
```



```
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.0.1.10
PREFIX=24
GATEWAY=10.0.1.1
DNS1=192.168.60.230
```

2. Asociar en el virt-manager, la interfaz de red eth0 a la red virtual de management 10.0.10.0/24.

3. Cambiar el nombre del host a 'deploy' en el archivo /etc/hostname y con el comando:

```
# hostname deploy
```

4. Configurar las variables de proxy en el archivo /etc/environment con los siguientes comandos:

```
# echo "http_proxy=http://192.168.60.242:3128" >> /etc/environment
# echo "https_proxy=http://192.168.60.242:3128" >> /etc/environment
# echo "HTTP_PROXY=http://192.168.60.242:3128" >> /etc/environment
# echo "HTTPS_PROXY=http://192.168.60.242:3128" >> /etc/environment
# source /etc/environment
```

Verificar el acceso a Internet mediante el comando:

```
# curl www.google.com
```

5. Instalaciones necesarias:

- 5.1. Actualizar repositorios y reiniciar:

```
# yum upgrade -y
# reboot
```

- 5.2. Instalar repositorio OpenStack Stein:

```
# yum install -y https://rdoproject.org/repos/openstack-stein/
rdo-release-stein.rpm
```

- 5.3. Instalar herramientas auxiliares:

```
# yum install git ntp ntpdate openssh-server python-devel sudo '
@Development Tools'
```

6. Deshabilitar firewall de CentOS:

```
# systemctl stop firewalld
# systemctl mask firewalld
```

7. Configurar el servicio de NTP mediante Chrony, sustituyendo en el archivo `/etc/chrony.conf` las siguientes líneas:

```
server 0.south-america.pool.ntp.org
server 1.south-america.pool.ntp.org
server 2.south-america.pool.ntp.org
server 3.south-america.pool.ntp.org
```

Reiniciar el servicio mediante:

```
# systemctl restart chronyd
```

8. Clonar el repositorio de OpenStack-Ansible Stein:

```
# git clone -b 19.0.6 https://opendev.org/openstack/openstack-ansible
/opt/openstack-ansible
```

9. Preparar openstack-ansible:

```
# cd /opt/openstack-ansible
# ./scripts/bootstrap-ansible.sh
```

10. Copiar el contenido de configuración al `/etc`:

```
# cp -r /opt/openstack-ansible/etc/openstack_deploy /etc/
```

11. Crear la carpeta para logs de la instalación:

```
# mkdir /var/log/openstack
```

Infra1

1. Configuraciones de red:

- 1.1. Configurar los bridges `br-mgmt`, `br-storage`, `br-vxlan` y `br-vlan`:

```
DEVICE=br-mgmt                DNS1=192.168.60.230
BOOTPROTO=none                ONBOOT=yes
IPADDR=10.0.10.11             TYPE=Bridge
PREFIX=24                     NM_CONTROLLED=no
GATEWAY=10.0.10.2
```

```

DEVICE=br-storage                ONBOOT=yes
BOOTPROTO=none                   TYPE=Bridge
IPADDR=10.0.20.11                NM_CONTROLLED=no
PREFIX=24

DEVICE=br-vlan                   DEVICE=br-vxlan
BOOTPROTO=none                   BOOTPROTO=none
ONBOOT=yes                       IPADDR=10.0.31.11
TYPE=Bridge                      PREFIX=24
NM_CONTROLLED=no                 ONBOOT=yes
                                 TYPE=Bridge
                                 NM_CONTROLLED=no

```

1.2. Configurar las interfaces eth0, eth1, eth2 y eth2.31:

```

TYPE="Ethernet"                  TYPE="Ethernet"
BOOTPROTO=none                   BOOTPROTO=none
DEVICE=eth0                      DEVICE=eth1
ONBOOT=yes                       ONBOOT=yes
NM_CONTROLLED=no                 NM_CONTROLLED=no
BRIDGE=br-mgmt                   BRIDGE=br-storage

TYPE="Ethernet"                  BOOTPROTO=none
BOOTPROTO=none                   DEVICE="eth2.31"
DEVICE=eth2                      NAME="eth2.31"
ONBOOT=yes                       ONBOOT=yes
NM_CONTROLLED=no                 BRIDGE=br-vxlan
BRIDGE=br-vlan                   NM_CONTROLLED=no
                                 VLAN=yes

```

1.3. Asociar, en el virt-manager, las interfaces de red:

- 1.3.1. eth0 a la red virtual de management 10.0.10.0/24.
- 1.3.2. eth1 a la red virtual de storage 10.0.20.0/24.
- 1.3.3. eth2 a la red virtual de tenant 10.0.30.0/24.

2. Cambiar el hostname de la máquina a 'infra1' en el archivo `/etc/hostname` y con el comando:

```
# hostname infra1
```

3. Configurar las variables de proxy en la sesión pero no en el sistema, porque luego el nodo deploy configura el `/etc/environment` en todos los nodos.

```
# export http_proxy=http://10.0.1.1:3128
# export https_proxy=http://10.0.1.1:3128
# export HTTP_PROXY=http://10.0.1.1:3128
# export HTTPS_PROXY=http://10.0.1.1:3128
```

Verificar el acceso a Internet mediante el comando:

```
# curl www.google.com
```

4. Actualizar repositorios y reiniciar:

```
# yum upgrade -y
# reboot
```

5. Instalar herramientas auxiliares:

```
# yum install bridge-utils iputils lsof lvm2 chrony openssh-server
sudo tcpdump python
```

6. Deshabilitar el Network Manager:

```
# chkconfig NetworkManager off
# chkconfig network on
# service NetworkManager stop
# service network start
```

7. Deshabilitar el SELinux, cambiando en `/etc/sysconfig/selinux`, "SELINUX=enforcing" por "SELINUX=disabled".

8. Habilitar bonding de interfaces y vlans:

```
# echo 'bonding' >> /etc/modules-load.d/openstack-ansible.conf
# echo '8021q' >> /etc/modules-load.d/openstack-ansible.conf
```

9. Configurar el servicio de NTP mediante Chrony, sustituyendo en el archivo `/etc/chrony.conf` las siguientes líneas:

```
server 0.south-america.pool.ntp.org
server 1.south-america.pool.ntp.org
server 2.south-america.pool.ntp.org
server 3.south-america.pool.ntp.org
```

Reiniciar el servicio mediante:

```
# systemctl restart chronyd
```

10. Eliminar reglas de firewall que bloquean el tráfico:

10.1. Eliminar las reglas manualmente:

```
# iptables -D INPUT -j REJECT --reject-with icmp-host-prohibited
# iptables -D FORWARD -j REJECT --reject-with icmp-host-
  prohibited
```

10.2. Exportar las reglas a un archivo:

```
# iptables-save > /etc/sysconfig/iptables
```

10.3. En cada reboot de la máquina, importar las reglas del archivo anterior:

```
# iptables-restore < /etc/sysconfig/iptables
```

Compute1

1. Configuraciones de red:

1.1. Configurar los bridges br-mgmt, br-storage, br-vxlan y br-vlan:

```
DEVICE=br-mgmt                DEVICE=br-storage
BOOTPROTO=none                BOOTPROTO=none
IPADDR=10.0.10.12             IPADDR=10.0.20.12
PREFIX=24                     PREFIX=24
GATEWAY=10.0.10.2            ONBOOT=yes
DNS1=192.168.60.230          TYPE=Bridge
ONBOOT=yes                    NM_CONTROLLED=no
TYPE=Bridge
NM_CONTROLLED=no

DEVICE=br-vlan                DEVICE=br-vxlan
BOOTPROTO=none                BOOTPROTO=none
ONBOOT=yes                    IPADDR=10.0.31.12
TYPE=Bridge                   PREFIX=24
NM_CONTROLLED=no              ONBOOT=yes
                               TYPE=Bridge
                               NM_CONTROLLED=no
```

1.2. Configurar las interfaces eth0, eth1, eth2 y eth2.31:

TYPE="Ethernet"	TYPE="Ethernet"
BOOTPROTO=none	BOOTPROTO=none
DEVICE=eth0	DEVICE=eth1
ONBOOT=yes	ONBOOT=yes
NM_CONTROLLED=no	NM_CONTROLLED=no
BRIDGE=br-mgmt	BRIDGE=br-storage
TYPE="Ethernet"	BOOTPROTO=none
BOOTPROTO=none	DEVICE="eth2.31"
DEVICE=eth2	NAME="eth2.31"
ONBOOT=yes	ONBOOT=yes
NM_CONTROLLED=no	VLAN=yes
BRIDGE=br-vlan	NM_CONTROLLED=no
	BRIDGE=br-vxlan

2. Asociar, en el virt-manager, las interfaces de red:
 - 2.1. eth0 a la red virtual de management 10.0.10.0/24.
 - 2.2. eth1 a la red virtual de storage 10.0.20.0/24.
 - 2.3. eth2 a la red virtual de tenant 10.0.30.0/24.
3. Cambiar el hostname de la máquina a 'compute1' en el archivo /etc/hostname y con el comando:

```
# hostname compute1
```

A partir de este punto, el procedimiento es idéntico al nodo infra1. Se deben realizar los puntos 4 al 11.

Compute2

La configuración de este nodo es análoga a la del compute1 salvando las diferencias de IPs.

Storage1

1. Configuraciones de red:
 - 1.1. Configurar los bridges br-mgmt y br-storage.

DEVICE=br-mgmt	DEVICE=br-storage
BOOTPROTO=none	BOOTPROTO=none
IPADDR=10.0.10.13	IPADDR=10.0.20.13
PREFIX=24	PREFIX=24
GATEWAY=10.0.10.2	ONBOOT=yes
DNS1=192.168.60.230	TYPE=Bridge
ONBOOT=yes	NM_CONTROLLED=no
TYPE=Bridge	
NM_CONTROLLED=no	

1.2. Configurar las interfaces eth0 y eth1.

TYPE="Ethernet"	TYPE="Ethernet"
BOOTPROTO=none	BOOTPROTO=none
DEVICE=eth0	DEVICE=eth1
ONBOOT=yes	ONBOOT=yes
NM_CONTROLLED=no	NM_CONTROLLED=no
BRIDGE=br-mgmt	BRIDGE=br-storage

2. Asociar, en el virt-manager, las interfaces de red:

2.1. eth0 a la red virtual de management 10.0.10.0/24.

2.2. eth1 a la red virtual de storage 10.0.20.0./24.

3. Cambiar el hostname de la máquina a 'storage1' en el archivo /etc/hostname y con el comando:

```
# hostname storage1
```

A partir de este punto, el procedimiento es idéntico al nodo infra1. Se deben realizar los puntos 4 al 11.

12. Formatear la pieza física de almacenamiento de 200 GB.

```
# pvcreate --metadatasize 2048 physical_volume_device_path
```

Storage2

La configuración de este nodo es análoga a la del storage1 salvando las diferencias de IPs.

HAproxy1

1. Configurar las interfaces eth0 y eth1.

```
TYPE=Ethernet
BOOTPROTO=none
IPV4_FAILURE_FATAL=no
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=192.168.60.160
PREFIX=24
```

```
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=10.0.10.15
PREFIX=24
GATEWAY=10.0.10.2
DNS1=192.168.60.230
```

2. Asociar, en el virt-manager, las interfaces de red:
 - 2.1. eth0 al bridge alojado en el servidor renata en la red 192.168.60.0/24.
 - 2.2. eth1 a la red virtual de management 10.0.10.0/24.
3. Cambiar el hostname de la máquina a 'haproxy1' en el archivo `/etc/hostname` y con el comando:

```
# hostname haproxy1
```

A partir de este punto, el procedimiento es similar al nodo infra1. Se deben realizar los puntos 4 al 11 con la excepción del punto 7 en donde se deshabilita el NetworkManager.

Router

1. Configuraciones de red:
 - 1.1. Configurar las interfaces eth0, eth1, eth2 y eth3:

```
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
NAME=eth0
DEVICE=eth0
ONBOOT=yes
IPADDR=10.0.40.2
PREFIX=24
NETWORK=10.0.40.0
```

```
GATEWAY=10.0.40.1
DNS1=192.168.60.230
ZONE=public
PROXY_METHOD=none
BROWSER_ONLY=no
IPV4_FAILURE_FATAL=no
IPV6INIT=no
```



```
TYPE=Ethernet
BOOTPROTO=none
NAME=eth1
DEVICE=eth1
ONBOOT=yes
IPADDR=10.0.10.2
PREFIX=24
```

```
NETWORK=10.0.10.0
PROXY_METHOD=none
BROWSER_ONLY=no
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=no
```

```
TYPE=Ethernet
BOOTPROTO=none
NAME=eth2
DEVICE=eth2
ONBOOT=yes
IPADDR=10.0.20.2
PREFIX=24
NETWORK=10.0.20.0
PROXY_METHOD=none
BROWSER_ONLY=no
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=no
```

```
TYPE=Ethernet
BOOTPROTO=none
NAME=eth3
DEVICE=eth3
ONBOOT=yes
IPADDR=10.0.30.2
PREFIX=24
NETWORK=10.0.30.0
PROXY_METHOD=none
BROWSER_ONLY=no
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=no
```

1.2. Configurar las VLANs:

```
BOOTPROTO=none
NAME=eth3.100
DEVICE=eth3.100
ONBOOT=yes
IPADDR=10.0.100.2
PREFIX=24
NETWORK=10.0.100.0
VLAN=yes
TYPE=Vlan
PHYSDEV=eth3
VLAN_ID=100
REORDER_HDR=yes
GVRP=no
MVRP=no
PROXY_METHOD=none
BROWSER_ONLY=no
```

```
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=no
```

```

BOOTPROTO=none          VLAN_ID=101
NAME=eth3.101          REORDER_HDR=yes
DEVICE=eth3.101        GVRP=no
ONBOOT=yes             MVRP=no
IPADDR=10.0.101.2      PROXY_METHOD=none
PREFIX=24              BROWSER_ONLY=no
NETWORK=10.0.101.0     DEFROUTE=yes
VLAN=yes               IPV4_FAILURE_FATAL=no
TYPE=Vlan              IPV6INIT=no
PHYSDEV=eth3

```

1.3. Asociar, en el virt-manager, las interfaces de red:

1.3.1. eth0 a la red virtual de router-nat 10.0.40.0/24.

1.3.2. eth1 a la red virtual de management 10.0.10.0/24.

1.3.3. eth2 a la red virtual de storage 10.0.20.0/24.

1.3.4. eth3 a la red virtual de tenant 10.0.30.0/24.

2. Cambiar el hostname de la máquina a 'router' en el archivo `/etc/hostname` y con el comando:

```
# hostname router
```

3. Habilitar el módulo para vlans y verificar que quede habilitado con los comandos:

```
# modprobe 8021q
# lsmod |grep 8021q
```

4. Reiniciar el servicio de red:

```
# systemctl restart network
```

5. Configurar el forwarding entre redes:

5.1. Revisar el estado de las interfaces:

```

# nmcli d
DEVICE    TYPE        STATE        CONNECTION
eth0      ethernet    connected    eth0
eth1      ethernet    connected    eth1
eth2      ethernet    connected    eth2
eth3      ethernet    connected    eth3
eth3.100  vlan        connected    eth3.100

```

```
eth3.101 vlan      connected   eth3.101
lo         loopback unmanaged  --
```

5.2. Excluir los servicios basados en iptables:

```
# systemctl mask iptables ip6tables ebtables
```

5.3. En principio todas las interfaces en la zona pública y debería ser la zona por default:

```
# firewall-cmd --get-active-zones
# firewall-cmd --get-default-zone
```

5.4. Resetear la default zone a public para asegurarse:

```
# firewall-cmd --set-default-zone=public
```

5.5. Agregar las zonas management, storage y tenant:

```
# firewall-cmd --permanent --new-zone=management
# firewall-cmd --permanent --new-zone=storage
# firewall-cmd --permanent --new-zone=tenant
# firewall-cmd --reload
```

5.6. Remover la interfaz eth1 de la zona public y asignarla a la zona management:

```
# firewall-cmd --remove-interface=eth1 --zone=public
# firewall-cmd --permanent --add-interface=eth1 --zone=
management
```

5.7. Remover la interfaz eth2 de la zona public y asignarla a la zona storage:

```
# firewall-cmd --remove-interface=eth2 --zone=public
# firewall-cmd --permanent --add-interface=eth2 --zone=storage
```

5.8. Remover las interfaces eth3, eth3.100 y eth3.101 de la zona public y asignarla a la zona tenant:

```
# firewall-cmd --remove-interface=eth3 --zone=public
# firewall-cmd --permanent --add-interface=eth3 --zone=tenant
# firewall-cmd --remove-interface=eth3.100 --zone=public
# firewall-cmd --permanent --add-interface=eth3.100 --zone=
tenant
# firewall-cmd --remove-interface=eth3.101 --zone=public
# firewall-cmd --permanent --add-interface=eth3.101 --zone=
tenant
```

5.9. Recargar la configuración:

```
# firewall-cmd --reload
```

5.10. Revisar la configuración de las zonas:

```
# firewall-cmd --get-active-zones
```

5.11. Asegurarse de que las zonas están bien configuradas en las interfaces:

```
# nmcli con mod eth0 connection.zone public
# nmcli con mod eth1 connection.zone management
# nmcli con mod eth2 connection.zone storage
# nmcli con mod eth3 connection.zone tenant
# nmcli con mod eth3.100 connection.zone tenant
# nmcli con mod eth3.101 connection.zone tenant
# nmcli c reload
```

5.12. Habilitar el forwarding de ipv4:

5.12.1. Primero en forma permanente:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/ip_forward.
conf
```

5.12.2. Luego en tiempo de ejecución:

```
# sysctl -w net.ipv4.ip_forward=1
```

5.13. Habilitar el masquerade para generar el NAT entre las redes:

```
# firewall-cmd --permanent --zone=public --add-masquerade
# firewall-cmd --reload
```

La configuración de las claves SSH se realiza de forma análoga a la realizada para la instalación de Queens en la sección [C.2.2.1](#).

C.3.2. Configuración archivos OSA

openstack_user_config.yml

A continuación se muestran las configuraciones utilizadas en el archivo, el significado de cada sección fue explicado en [2.4.2.3](#). Adicionalmente se explican las modificaciones realizadas para adaptar la playbook a la nueva arquitectura planteada, en donde se utiliza Ceph como el backend de Cinder.

```
cidr_networks:
  container: 10.0.10.0/24
  tunnel: 10.0.31.0/24
  storage: 10.0.20.0/24
```

```

used_ips:
  - "10.0.10.1,10.0.10.30" # red de management
  - "10.0.20.1,10.0.20.30" # red de storage
  - "10.0.31.1,10.0.31.30" # red de vxlan

global_overrides:
  internal_lb_vip_address: 10.0.10.15
  external_lb_vip_address: 192.168.60.160

  tunnel_bridge: "br-vxlan"
  management_bridge: "br-mgmt"
  storage_bridge: "br-storage"

  provider_networks:
    - network:
        group_binds:
          - all_containers
          - hosts
        type: "raw"
        container_bridge: "br-mgmt"
        container_interface: "eth1"
        container_type: "veth"
        ip_from_q: "container"
        is_container_address: true
        is_ssh_address: true
    - network:
        group_binds:
          - glance_api
          - cinder_api
          - cinder_volume
          - nova_compute
          - ceph-osd
        type: "raw"
        container_bridge: "br-storage"
        container_type: "veth"
        container_interface: "eth2"
        container_mtu: "9000"
        ip_from_q: "storage"
    - network:
        group_binds:
          - neutron_linuxbridge_agent
        container_bridge: "br-vxlan"

```

```

        container_type: "veth"
        container_interface: "eth10"
        container_mtu: "9000"
        ip_from_q: "tunnel"
        type: "vxlan"
        range: "1:1000"
        net_name: "vxlan"
- network:
    group_binds:
        - neutron_linuxbridge_agent
    container_bridge: "br-vlan"
    container_type: "veth"
    container_interface: "eth11"
    type: "vlan"
    range: "150:200,300:400"
    net_name: "vlan"
- network:
    group_binds:
        - neutron_linuxbridge_agent
    container_bridge: "br-vlan"
    container_type: "veth"
    container_interface: "eth12"
    host_bind_override: "eth12"
    type: "flat"
    net_name: "flat"

```

La red asociada al bridge de storage se relaciona con un nuevo grupo de hosts llamado ceph-osd. En dicho grupo se encuentran las IPs de los nodos que se utilizan como OSDs en la instalación de Ceph, siendo en este caso los nodos de storage.

El bridge br-vlan es utilizado por las redes de tipo flat y VLAN. En cada una de estas secciones una de las configuraciones más relevantes es la que define de qué interfaz física dependen las redes que se disponibilizan en OpenStack. Por lo tanto las redes de tipo VXLAN dependen del bridge br-vxlan, las VLANs del bridge br-vlan y en el caso de las redes de tipo flat se agrega la configuración `host_bind_override: "eth12"` porque Neutron no soporta tener redes definidas sobre el mismo recurso de red. Con esta opción se indica que la red flat definida en Neutron utilizará la interfaz eth12 en los nodos de red y cómputo. Es por esta razón que en la preparación de los nodos se agrega el veth pair eth12.

En la última sección en donde se indica en qué servidor o grupo de servidores corre cada servicio, se agregan los nuevos nodos utilizados con sus respectivas IPs.

Además se modifica la sintaxis utilizada con el fin de evitar escribir múltiples veces una misma IP, reduciendo la probabilidad de un error de tipeo.

```
###
### Infrastructure
###

_infrastructure_hosts: &infrastructure_hosts
    infra1:
        ip: 10.0.10.11

# nova hypervisors
compute_hosts:
    compute1:
        ip: 10.0.10.12
    compute2:
        ip: 10.0.10.22

# ceph Object Storage Deamons
ceph-osd_hosts:
    osd1:
        ip: 10.0.10.13
    osd2:
        ip: 10.0.10.23

# galera, memcache, rabbitmq, utility
shared-infra_hosts: *infrastructure_hosts

# ceph monitor containers
ceph-mon_hosts: *infrastructure_hosts

# repository (apt cache, python packages, etc)
repo-infra_hosts: *infrastructure_hosts

# load balancer
# Ideally the load balancer should not use the Infrastructure hosts.
# Dedicated hardware is best for improved performance and security.
haproxy_hosts:
    balancer1:
        ip: 10.0.10.15
```

```

# rsyslog server
log_hosts: *infrastructure_hosts

###
### OpenStack
###

# keystone
identity_hosts: *infrastructure_hosts

# cinder api services
storage-infra_hosts: *infrastructure_hosts

# glance
image_hosts: *infrastructure_hosts

# nova api, conductor, etc services
compute-infra_hosts: *infrastructure_hosts

# heat
orchestration_hosts: *infrastructure_hosts

# horizon
dashboard_hosts: *infrastructure_hosts

# neutron server, agents (L3, etc)
network_hosts: *infrastructure_hosts

# cinder volume hosts (Ceph RBD-backed)
storage_hosts: *infrastructure_hosts

```

Los parámetros para desplegar Ceph, a diferencia de LVM, se realizan en el archivo `user_variables.yml` que se presenta a continuación.

user_variables.yml

En primer lugar se configura el proxy de salida a ser utilizado por los contenedores y hosts:

```

## Example environment variable setup:
## This is used by apt-cacher-ng to download apt packages:
proxy_env_url: http://192.168.60.242:3128/

```



```

## (1) This sets up a permanent environment, used during and after
    deployment:
no_proxy_env: "localhost,127.0.0.1,{{ internal_lb_vip_address }},{{
    external_lb_vip_address }},{% for host in groups['all_containers']
    %}{{ hostvars[host]['container_address'] }}{% if not loop.last %},{%
    endif %}{{% endfor %}"
global_environment_variables:
    HTTP_PROXY: "{{ proxy_env_url }}"
    HTTPS_PROXY: "{{ proxy_env_url }}"
    NO_PROXY: "{{ no_proxy_env }}"
    http_proxy: "{{ proxy_env_url }}"
    https_proxy: "{{ proxy_env_url }}"
    no_proxy: "{{ no_proxy_env }}"

#
## (2) This is applied only during deployment, nothing is left after
    deployment is complete:
deployment_environment_variables:
    http_proxy: "{{ proxy_env_url }}"
    https_proxy: "{{ proxy_env_url }}"
    no_proxy: "localhost,127.0.0.1,{{ internal_lb_vip_address }},{{
        external_lb_vip_address }},{% for host in groups['keystone_all
        '] %}{{ hostvars[host]['container_address'] }}{% if not loop.
        last %},{% endif %}{{% endfor %}"

```

Se mantiene la variable `horizon_images_upload_mode: "legacy"` para poder crear imágenes desde el Horizon remotamente.

Por último se configuran las variables requeridas por las playbooks de Ceph siguiendo los documentos [8] y [26] como referencia.

```

## ceph-ansible settings
## See https://github.com/ceph/ceph-ansible/tree/master/group_vars for
## additional configuration options available.
monitor_address_block: "{{ cidr_networks.container }}"
public_network: "{{ cidr_networks.container }}"
cluster_network: "{{ cidr_networks.storage }}"
osd_scenario: lvm
osd_objectstore: bluestore
osd_auto_discovery: false
lvm_volumes:
    - data: /dev/vdb

```

```

journal_size: 10240 # size in MB
# ceph-ansible automatically creates pools & keys for OpenStack services
openstack_config: true
cinder_ceph_client: cinder
glance_ceph_client: glance
glance_default_store: rbd
glance_rbd_store_pool: images
nova_libvirt_images_rbd_pool: vms

cinder_backends:
  RBD:
    volume_driver: cinder.volume.drivers.rbd.RBDDriver
    rbd_pool: volumes
    rbd_ceph_conf: /etc/ceph/ceph.conf
    rbd_store_chunk_size: 8
    volume_backend_name: rbd
    rbd_user: "{{ cinder_ceph_client }}"
    rbd_secret_uuid: "{{ cinder_ceph_client_uuid }}"
    report_discard_supported: true

```

cinder-volume.yml

En este caso el despliegue de Ceph se debe indicar que no será en metal dado que se crearán los OSDs en los nodos de storage.

```

container_skel:
  cinder_volumes_container:
    properties:
      is_metal: false

```

Generación de claves

Este paso es análogo al descrito para la instalación de la versión Queens en la sección [C.2.2.3](#).

C.3.3. Cambios para driver OVS

En [55] se encuentra una guía para realizar una instalación de OSA Stein junto con el driver OVS del módulo de red. Utilizando la configuración para el driver Linux Bridge como base, es necesario realizar las siguientes modificaciones.

1. No se utiliza el veth pair eth12 en los nodos de red y cómputo debido a que no es necesario para que plugin de Open vSwitch soporte redes flat. Además en la instalación planteada para el driver OVS no se configuran redes de tipo flat.
2. Se deben realizar ciertas modificaciones en la sección de redes provider del archivo `openstack_user_config.yml`:

```

provider_networks:
  - network:
      group_binds:
        - all_containers
        - hosts
      type: "raw"
      container_bridge: "br-mgmt"
      container_interface: "eth1"
      container_type: "veth"
      ip_from_q: "container"
      is_container_address: true
      is_ssh_address: true
  - network:
      group_binds:
        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute
        - ceph-osd
      type: "raw"
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
      container_mtu: "9000"
      ip_from_q: "storage"
  - network:
      group_binds:
        - neutron_openvswitch_agent
      container_bridge: "br-vxlan"
      container_type: "veth"
      container_interface: "eth10"
      container_mtu: "9000"
      ip_from_q: "tunnel"
      type: "vxlan"

```

```

        range: "1:1000"
        net_name: "vxlan"
- network:
    group_binds:
        - neutron_openvswitch_agent
    container_bridge: "br-provider"
    container_type: "veth"
    container_interface: "eth11"
    type: "vlan"
    range: "150:200,300:400"
    net_name: "vlan"
    network_interface: "br-vlan"

```

2.1. Se elimina la red provider definida para las redes de tipo flat.

2.2. Se modifica la red provider definida para las redes de tipo vlan. En primer lugar se cambia la opción `container_bridge` por el valor `br-provider`, siendo el nombre del bridge de OVS que será creado automáticamente. Además, se utiliza la opción `network_interface` para indicar con cuál interfaz del nodo se deberá comunicar el bridge de OVS.

3. En el archivo `user_variables.yml` se agregan las siguientes variables:

```

# Configuración para el plugin de OpenVSwitch
neutron_plugin_type: ml2.ovs
neutron_ml2_drivers_type: "vxlan,vlan"

```

Indicando el driver que se debe configurar en la instalación de OSA y los tipos de redes que podrá soportar.

4. Es requerido crear el archivo de grupo de variables `/etc/openstack_deploy/group_vars/network_hosts`, con el siguiente contenido:

```

# Ensure the openvswitch kernel module is loaded
openstack_host_specific_kernel_modules:
  - name: "openvswitch"
    pattern: "CONFIG_OPENVSWITCH"

```

C.4. Verificación

Luego de que la última playbook haya terminado su ejecución sin error, se debe verificar que la instalación fue exitosa. Esto se realiza de forma manual siguiendo los pasos que se indican a continuación:

1. Acceder al nodo de infra1 como usuario root.
2. La instalación que realiza OSA crea contenedores de utilidad los cuales proveen de todas las herramientas desde la consola para utilizar OpenStack. En primer lugar se deben listar todos los containers del nodo físico ejecutando:

```
# lxc-ls -f
```

3. El contenedor que se utiliza es el que tiene en su nombre la palabra utility, para acceder al mismo es necesario ejecutar el siguiente comando de lxc:

```
# lxc-attach -n <nombre_contenedor>
```

4. Para utilizar los servicios de OpenStack es necesario enviar las credenciales del usuario que invocará a las APIs de los servicios. Esto se debe al funcionamiento de OpenStack en donde cada llamada a una API debe ser validada por el módulo de Keystone. Para evitar escribir las credenciales en cada comando, OSA genera un archivo llamado openrc para cargar la información del usuario como variables de entorno. El archivo se carga con el siguiente comando:

```
# source openrc
```

5. Algunos comandos que se pueden ejecutar son:

- Para listar usuarios:

```
# openstack user list --os-cloud=default
```

- Para listar servidores:

```
# openstack server list
```

- Para listar redes:

```
# openstack network list
```

- Para listar los agentes de red:

```
# openstack network agent list
```

6. Por otro lado se puede verificar el dashboard de Horizon accediendo a la ip definida en `external_lb_vip_address` en el archivo `/etc/openstack_deploy/openstack_user_config.yml` en el puerto 443 dado que utiliza HTTPS. Para autenticarse como admin es necesaria la password que se encuentra definida en la opción `keystone_auth_admin_password` del archivo `/etc/openstack_deploy/user_secrets.yml`.

Anexo D

Interacción

Esta sección se realizó utilizando el dashboard de OpenStack, brindado por el módulo Horizon. Debido a las limitaciones de red ya mencionadas, es necesario realizar un reenvío de puertos mediante SSH para acceder al mismo. Esto se logra en dos pasos mediante:

1. Primero se realiza un forwarding desde la máquina local hasta lulu:

```
$ ssh -L 2443:localhost:2443 <usuario_fing>@lulu.fing.edu.uy
```
2. Una vez dentro de lulu, se realiza un nuevo forwarding desde la misma hasta la IP pública de el balanceador, a través del servidor renata.

```
$ ssh -L 2443:192.168.60.160:443 openstack@192.168.60.242 -4
```

Luego, accediendo a través de un navegador a la dirección `https://localhost:2443` se llega a la vista de login de OpenStack.



Figura D.1: Vista del login de Horizon.

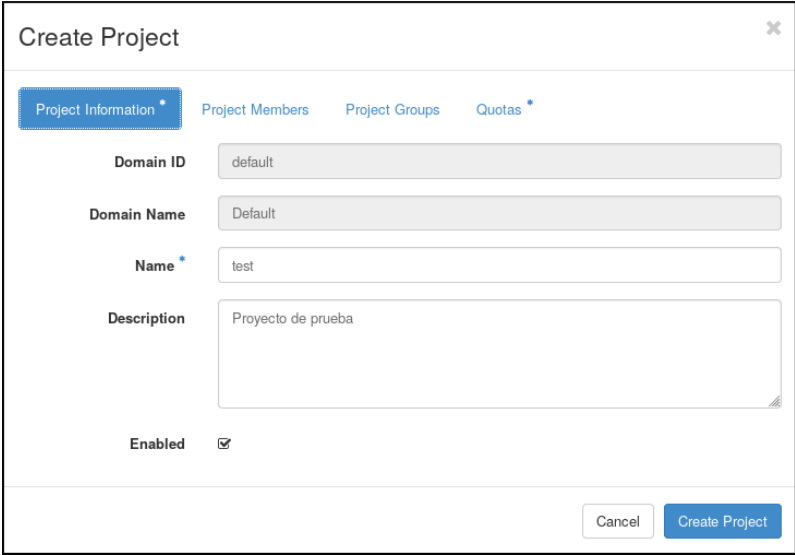
Una vez allí, se debe acceder con la cuenta de administrador para realizar las configuraciones iniciales utilizando las credenciales indicadas en el último paso de la sección de verificación.

D.1. Configuraciones de administrador

Para que los usuarios finales puedan operar sobre la plataforma OpenStack, se deben configurar ciertos aspectos como lo son proyectos, usuarios, flavors y redes provider, entre otros. A continuación se presenta un instructivo básico para ello.

Crear proyecto

Lo primero a configurar es crear un proyecto sobre el cual se realizarán las pruebas siguientes. Para ello en el menú lateral se accede a Identity >Projects >Create Project, completando los campos como se muestra en la figura [D.2](#).



The screenshot shows a 'Create Project' dialog box with a close button (X) in the top right corner. It features four tabs: 'Project Information' (active), 'Project Members', 'Project Groups', and 'Quotas'. The 'Project Information' tab contains the following fields:

- Domain ID:** default
- Domain Name:** Default
- Name:** test
- Description:** Proyecto de prueba
- Enabled:**

At the bottom right, there are two buttons: 'Cancel' and 'Create Project'.

Figura D.2: Creación de un proyecto (1/2).

Antes de confirmar la creación, en la pestaña Project Members agregamos al admin como miembro del proyecto.

Create Project ✕

Project Information * **Project Members** Project Groups Quotas *

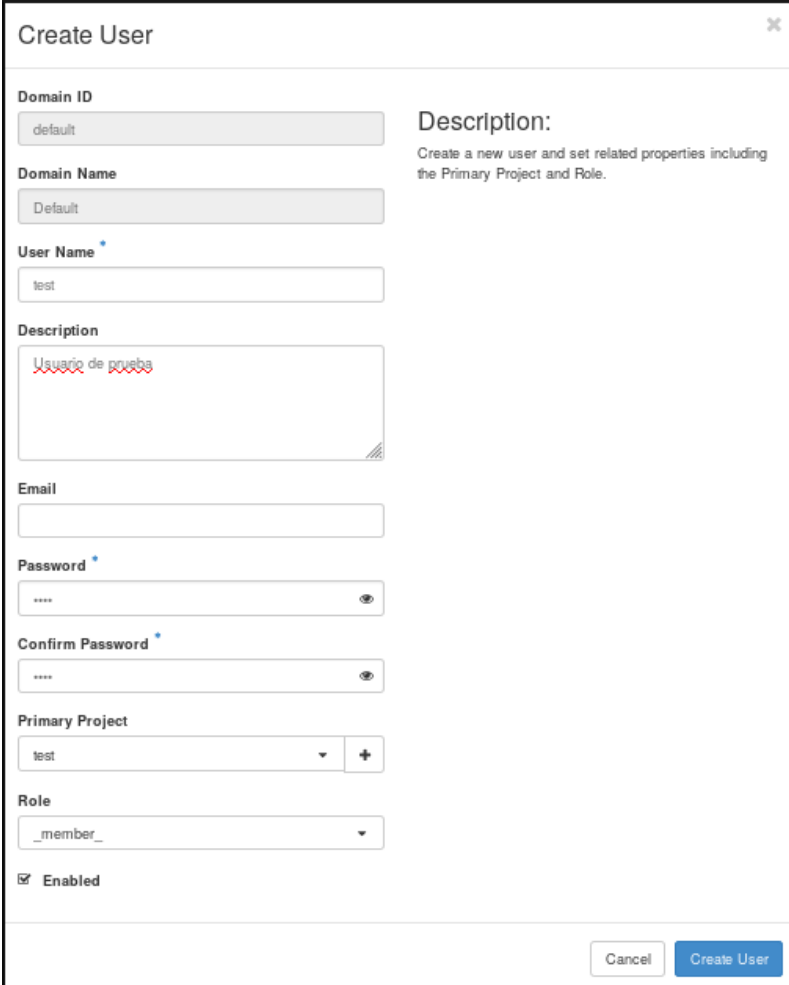
All Users	Filter	Q
cinder		+
placement		+
nova		+
keystone		+
neutron		+
heat		+
glance		+

Project Members	Filter	Q
admin	member ▾	-

Figura D.3: Creación de un proyecto (2/2).

Crear usuario

Luego se crea un usuario de pruebas accediendo nuevamente por la pestaña lateral a Identity >Users >Create User, rellenando los campos solicitados como se presenta en la figura D.4.



The screenshot shows a 'Create User' form with the following fields and values:

- Domain ID:** default
- Domain Name:** Default
- User Name:** test
- Description:** Usuario de prueba
- Email:** (empty)
- Password:** (masked with ****)
- Confirm Password:** (masked with ****)
- Primary Project:** test
- Role:** _member_
- Enabled:**

At the bottom right, there are two buttons: 'Cancel' and 'Create User'.

Figura D.4: Creación de un usuario.

Por más detalles se puede visitar [42].

Crear flavor

El siguiente paso será crear un flavor de pruebas con algunas características básicas y permitirle acceso al proyecto de test creado. Se accede desde Admin >Compute >Flavors. Esto se ilustra en las figuras D.5 y D.6. Más detalles en [41].

Create Flavor

Flavor Information * Flavor Access

Name *
test-1

ID ⓘ
auto

VCPUs *
2

RAM (MB) *
4

Root Disk (GB) *
50

Ephemeral Disk (GB)
0

Swap Disk (MB)
0

RX/TX Factor
1

Flavors define the sizes for RAM, disk, number of cores, and other resources and can be selected when users deploy instances.

Cancel Create Flavor

Figura D.5: Creación de un flavor (1/2).

Create Flavor

Flavor Information * Flavor Access

Select the projects where the flavors will be used. If no projects are selected, then the flavor will be available in all projects.

All Projects	Filter	Selected Projects	Filter
admin		test	
service			

Cancel Create Flavor

Figura D.6: Creación de un flavor (2/2).

Crear provider network

Este tipo de red es manejado por los administradores y para crearlas se debe acceder a Admin >Network >Networks >Create Network, completando el formulario como se muestra en las imágenes [D.7](#) y [D.8](#).

Create Network

Network * Subnet Subnet Details

Name

Project *

Provider Network Type * ⓘ

Physical Network * ⓘ

Enable Admin State

Shared

External Network

Create Subnet

Availability Zone Hints ⓘ

Create a new network. In addition, a subnet associated with the network can be created in the following steps of this wizard.

Figura D.7: Creación de una red provider (1/2).

El valor del campo Physical Network es el especificado en la red de tipo “flat” en la sección de provider networks del archivo `openstack_user_config.yml`.

Create Network

Network * Subnet Subnet Details

Subnet Name

Network Address ⓘ

IP Version

Gateway IP ⓘ

Disable Gateway

Creates a subnet associated with the network. You need to enter a valid "Network Address" and "Gateway IP". If you did not enter the "Gateway IP", the first value of a network will be assigned by default. If you do not want gateway please check the "Disable Gateway" checkbox. Advanced configuration is available by clicking on the "Subnet Details" tab.

Figura D.8: Creación de una red provider (2/2).

D.2. Interacción de un usuario

A partir de ahora, el ambiente se encuentra con las configuraciones mínimas necesarias para que el usuario test pueda acceder y hacer uso de la plataforma. Por

lo tanto, los pasos a continuación se realizan sobre el proyecto de prueba, habiéndose autenticado previamente con el usuario test.

Crear imagen

Lo primero que deberá hacer el usuario para crear una instancia es crear la imagen que será utilizada por esta. Desde [34] se pueden descargar formatos de imágenes soportados por OpenStack de la mayoría de los sistemas operativos Linux. La creación de la imagen se realiza desde el menú Project >Compute >Images >Create Image.

The screenshot shows the 'Create Image' form in OpenStack. The form is titled 'Create Image' and has a 'Close' button in the top right. It is divided into two tabs: 'Image Details' (selected) and 'Metadata'. The 'Image Details' section contains several fields: 'Image Name' (text input with 'image-test-1'), 'Image Description' (text input with 'Imagen de prueba'), 'Image Source' section with 'Source Type' (radio button for 'File'), 'File' (text input with 'Browse...' and 'cirros-0.4.0-x86_64-disk.img'), 'Format' (dropdown menu with 'QCOW2 - QEMU Emulator'), 'Image Requirements' section with 'Kernel' (dropdown with 'Choose an image'), 'Ramdisk' (dropdown with 'Choose an image'), 'Architecture' (text input with 'x86_64'), 'Minimum Disk (GB)' (text input with '10'), and 'Minimum RAM (MB)' (text input with '2048'). The 'Image Sharing' section has a 'Protected' section with 'Yes' and 'No' radio buttons. At the bottom, there are three buttons: 'Cancel', '< Back', and 'Next >', and a blue 'Create Image' button with a checkmark.

Figura D.9: Creación de una imagen (1/2).

En la primer pantalla se establecen los datos básicos para la creación de la imagen y en la siguiente se puede establecer medatada más específica de la misma.

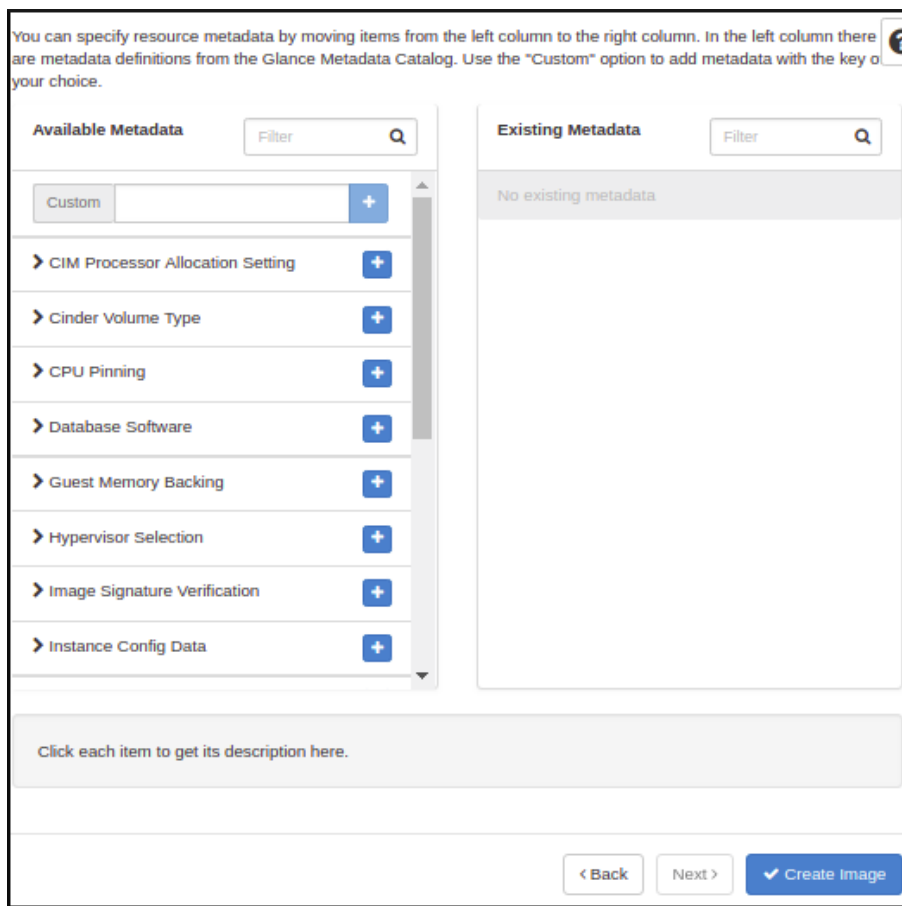


Figura D.10: Creación de una imagen (2/2).

Crear red

La creación de nuevas subredes se realiza accediendo a Project >Network >Networks >Create Network. Esta funcionalidad consta de tres pasos detallados a continuación:

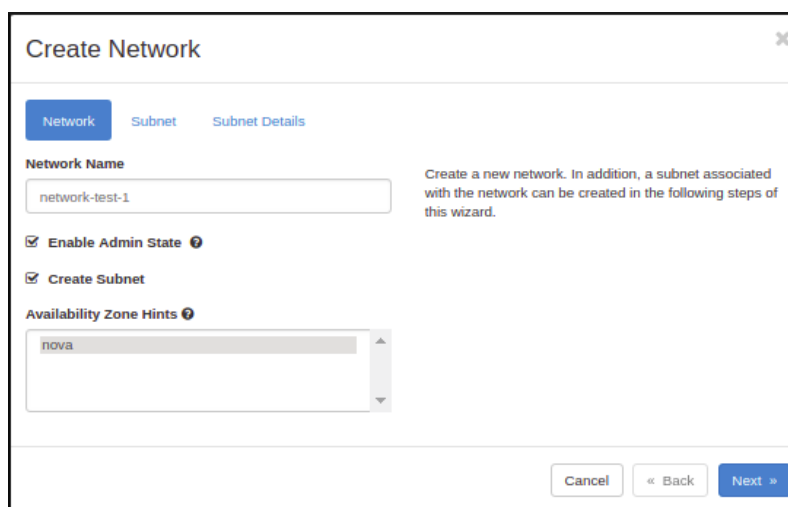


Figura D.11: Creación de una red (1/3).

Create Network

Network Subnet Subnet Details

Subnet Name

Network Address ⓘ

IP Version

Gateway IP ⓘ

Disable Gateway

Creates a subnet associated with the network. You need to enter a valid "Network Address" and "Gateway IP". If you did not enter the "Gateway IP", the first value of a network will be assigned by default. If you do not want gateway please check the "Disable Gateway" checkbox. Advanced configuration is available by clicking on the "Subnet Details" tab.

Cancel < Back Next >

Figura D.12: Creación de una red (2/3).

Create Network

Network Subnet Subnet Details

Enable DHCP

Allocation Pools ⓘ

DNS Name Servers ⓘ

Host Routes ⓘ

Specify additional attributes for the subnet.

Cancel < Back Create

Figura D.13: Creación de una red (3/3).

Crear router

La creación de nuevos routers se realiza accediendo a Project >Network >Routers >Create Router.

Figura D.14: Creación de un router.

Más detalles de estas configuraciones se pueden encontrar en [30].

Crear interfaz de router

Ir a Network >Routers y en la grilla desplegada seleccionar el router. En la pestaña de Interfaces se encuentra la opción para crear una nueva interfaz.

Figura D.15: Creación de una interfaz en un router.

Para la creación de una interfaz se debe ingresar la subred a la que estará conectada y opcionalmente la IP de la misma.

Crear key pair

La creación de key pairs se realiza en Project >Compute >Key Pairs >Create Key Pair.

Figura D.16: Creación de una key pair.

Luego al momento de crear una nueva instancia se debe seleccionar una clave. Se debe tener en cuenta que la asignación de key pairs a las instancias mediante

Horizon solamente se puede realizar en el momento de creación de las mismas. Más detalles se encuentran en [27].

Lanzar una instancia

La creación de nuevas instancias se realiza en Project >Instances >Launch Instance. En primer lugar se especifican aspectos básicos como el nombre y la descripción D.17.

The screenshot shows the 'Launch Instance' dialog box with the following details:

- Instance Name:** Instance-test-1
- Description:** Instancia de prueba 1
- Availability Zone:** nova
- Count:** 1
- Total Instances (10 Max):** 10%
- Progress:** 0 Current Usage, 1 Added, 9 Remaining
- Buttons:** Cancel, < Back, Next >, Launch Instance

Figura D.17: Lanzar una nueva instancia (1/5).

Luego se deberá indicar la imagen a ser utilizada para bootear la máquina. En este caso se elige la imagen imagen-test-1 creada anteriormente.

Launch Instance
✕

- Details
- Source
- Flavor *
- Networks *
- Network Ports
- Security Groups
- Key Pair
- Configuration
- Server Groups
- Scheduler Hints
- Metadata

Instance source is the template used to create an Instance. You can use an Image, a snapshot of an Instance (Image snapshot), a volume or a volume snapshot (if enabled). You can also choose to use persistent storage by creating a new volume. ?

Select Boot Source

Image

Volume Size (GB) *

10

▲
▼

Create New Volume

Yes No

Delete Volume on Instance Delete

Yes No

Allocated

Name	Updated	Size	Type	Visibility	
▶ Imagen-test-1	8/3/19 10:08 AM	12.13 MB	Iso	Private	↓

Available 2 Select one

Q Click here for filters. ✕

Name	Updated	Size	Type	Visibility	
▶ centos-7	8/3/19 10:28 AM	918.00 MB	Iso	Public	↑
▶ Image1	8/3/19 9:48 AM	12.13 MB	Iso	Public	↑

✕ Cancel
< Back
Next >
Launch Instance

Figura D.18: Lanzar una nueva instancia (2/5).

A continuación se determinan los recursos virtuales de la instancia mediante la selección de un flavor.

Launch Instance

Flavors manage the sizing for the compute, memory and storage capacity of the Instance.

Allocated

Name	VCPUS	RAM	Total Disk	Root Disk	Ephemeral Disk	Public
> test-1	2	4 GB	50 GB	50 GB	0 GB	No

Available 0

Select one

Click here for filters.

Name	VCPUS	RAM	Total Disk	Root Disk	Ephemeral Disk	Public
------	-------	-----	------------	-----------	----------------	--------

Cancel Back Next Launch Instance

Figura D.19: Lanzar una nueva instancia (3/5).

El siguiente paso requerido es la selección de una red.

Launch Instance

Networks provide the communication channels for Instances in the cloud.

Allocated 1

Select networks from those listed below.

Network	Subnets Associated	Shared	Admin State	Status
> 1 network-test-1	subnet-test-1	No	Up	Active

Available 1

Select at least one network

Click here for filters.

Network	Subnets Associated	Shared	Admin State	Status
> network-test-2	subnet-test-2	No	Up	Active

Cancel Back Next Launch Instance

Figura D.20: Lanzar una nueva instancia (4/5).

Finalmente, la última configuración mínimamente requerida para acceder en forma remota a la instancia es configurar una key pair.

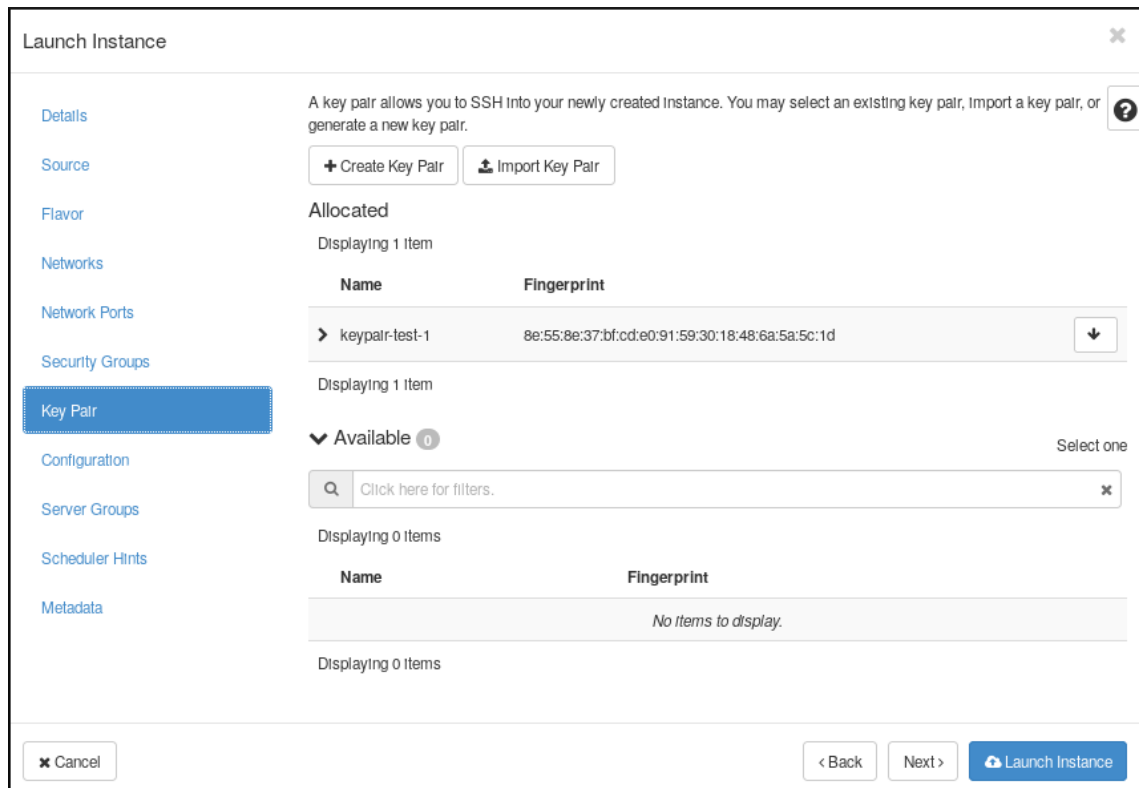


Figura D.21: Lanzar una nueva instancia (5/5).

Más detalles sobre el lanzamiento de una instancia se pueden ver accediendo a [37].

D.3. Acceso a una instancia

D.3.1. Por SPICE

Para acceder a la consola de la instancia creada a través del protocolo SPICE [57], es necesario dirigirse a Project >Compute >Instances >instances-test-1. Allí en la pestaña console deberíamos obtener el acceso, sin embargo debido a las limitaciones de red ya conocidas es necesario realizar un nuevo forwarding de puertos (en este caso el 6082) y acceder a través de localhost. Para esto:

1. Primero se realiza un forwarding desde la máquina local hasta lulu:

```
$ ssh -L 6082:localhost:6082 <usuario_fing>@lulu.fing.edu.uy
```

2. Una vez dentro de lulu, se realiza un nuevo forwarding desde la misma hasta la IP pública de el balanceador, a través del servidor renata.

```
$ ssh -L 6082:192.168.60.160:6082 openstack@192.168.60.242 -4
```

Luego en la pestaña de console, abrimos el link asociado a *'Click here to show only console'*, esto intentará cargar en el navegador una URL similar a la siguiente:
[https://192.168.60.160:6082/spice_auto.html?token=a3703173-3973-440d-babf-f8b662b2fd25&title=instance-test-1\(430c92de-3cab-4b57-be7e-6394793dc423\)](https://192.168.60.160:6082/spice_auto.html?token=a3703173-3973-440d-babf-f8b662b2fd25&title=instance-test-1(430c92de-3cab-4b57-be7e-6394793dc423))

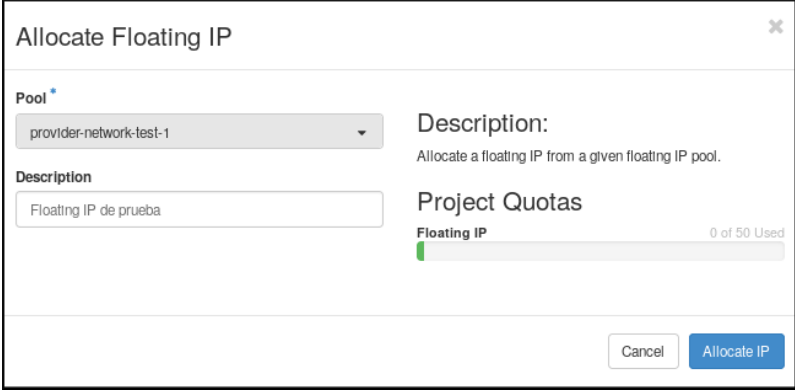
En la misma debemos sustituir la IP 192.168.60.160 por localhost para utilizar el forwarding realizado. Allí lograremos acceder a la consola SPICE.

D.3.2. Por SSH

Para lograr acceder en forma externa por SSH son necesarias algunas configuraciones extras.

D.3.2.1. Asociar una Floating IP a la instancia

Estas IPs son necesarias para poder acceder a las instancias desde redes externas. A continuación se detalla cómo realizarlo a partir de Project >Network >Floating IPs >Allocate IP To Project:



The screenshot shows a modal window titled "Allocate Floating IP". It contains the following elements:

- Pool:** A dropdown menu with "provider-network-test-1" selected.
- Description:** A text input field containing "Floating IP de prueba".
- Description:** A label "Description:" followed by the text "Allocate a floating IP from a given floating IP pool."
- Project Quotas:** A section titled "Project Quotas" with a sub-label "Floating IP" and a progress bar showing "0 of 50 Used".
- Buttons:** "Cancel" and "Allocate IP" buttons at the bottom right.

Figura D.22: Asignación de floating IP.

Luego se debe asociar con el puerto de la instancia creada desde Project >Network >Floating IPs >Associate.

Figura D.23: Asociación de floating IP.

D.3.2.2. Modificar security group

Los security groups en OpenStack son firewalls virtuales en donde se definen una serie de reglas a ser aplicadas al tráfico de las instancias. El grupo creado por defecto en la instalación de OpenStack contiene las siguientes reglas:

Displaying 4 items						
<input type="checkbox"/>	Direction ▲	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group
<input type="checkbox"/>	Egress	IPv4	Any	Any	0.0.0.0/0	-
<input type="checkbox"/>	Egress	IPv6	Any	Any	:::0	-
<input type="checkbox"/>	Ingress	IPv4	Any	Any	-	default
<input type="checkbox"/>	Ingress	IPv6	Any	Any	-	default

Figura D.24: Reglas security group por defecto.

Estas reglas permiten el tráfico de salida hacia cualquier IP y solamente se permite el tráfico de entrada de instancias del mismo security group.

Para tener conectividad SSH e ICMP con las instancias creadas se deben agregar las siguientes reglas accediendo por Project >Network >Security Groups >Manage Rules >Add Rule.

Add Rule

Rule
All ICMP

Direction
Ingress

Remote
CIDR

CIDR
0.0.0.0/0

Description:
Rules define which traffic is allowed to instances assigned to the security group. A security group rule consists of three main parts:
Rule: You can specify the desired rule template or use custom rules, the options are Custom TCP Rule, Custom UDP Rule, or Custom ICMP Rule.
Open Port/Port Range: For TCP and UDP rules you may choose to open either a single port or a range of ports. Selecting the "Port Range" option will provide you with space to provide both the starting and ending ports for the range. For ICMP rules you instead specify an ICMP type and code in the spaces provided.
Remote: You must specify the source of the traffic to be allowed via this rule. You may do so either in the form of an IP address block (CIDR) or via a source group (Security Group). Selecting a security group as the source will allow any other instance in that security group access to any other instance via this rule.

Cancel Add

Figura D.25: Agregar regla para tráfico ICMP.

Add Rule

Rule
SSH

Remote
CIDR

CIDR
0.0.0.0/0

Description:
Rules define which traffic is allowed to instances assigned to the security group. A security group rule consists of three main parts:
Rule: You can specify the desired rule template or use custom rules, the options are Custom TCP Rule, Custom UDP Rule, or Custom ICMP Rule.
Open Port/Port Range: For TCP and UDP rules you may choose to open either a single port or a range of ports. Selecting the "Port Range" option will provide you with space to provide both the starting and ending ports for the range. For ICMP rules you instead specify an ICMP type and code in the spaces provided.
Remote: You must specify the source of the traffic to be allowed via this rule. You may do so either in the form of an IP address block (CIDR) or via a source group (Security Group). Selecting a security group as the source will allow any other instance in that security group access to any other instance via this rule.

Figura D.26: Agregar regla para tráfico SSH.

D.3.2.3. SSH

Finalmente, para acceder a la consola de la instancia creada a través de SSH desde el servidor renata es necesario copiar la clave generada en el paso de key pair y luego acceder mediante:

```
$ ssh -i keypair-test-1.pem cirros@10.0.4.17
```

Una vez dentro, para tener conectividad a Internet se debe configurar el mismo proxy que fue asignado a todos los nodos de OpenStack, es decir:

```
$ export http_proxy="http://10.0.1.1:3128"
```

D.3.3. Por virsh

El administrador del Datacenter es capaz de acceder directamente a las instancias alojadas en los nodos de cómputo a través de la herramienta virsh [20] de libvirt. Para esto son útiles los siguientes comandos.

- Listar las instancias corriendo dentro de un nodo de cómputo:

```
[root@compute1 ~]# virsh list
```

- Obtener una consola dentro de la instancia:

```
[root@compute1 ~]# virsh console <instance_name>
```

Conclusiones

De esta sección se destaca la facilidad de uso que tiene la herramienta una vez que está desplegada y operativa. La curva de aprendizaje es significativamente baja dado que para una persona con ciertos conocimientos en redes o administración de sistemas el dashboard proporcionado por Horizon resulta muy intuitivo para realizar tareas como: agregar una nueva instancia, administración de usuarios y roles, creación de redes tenants y provider, entre otras. Además de lo funcional, los dashboards con información de los recursos asignados a los proyectos junto a qué porción de los mismos están siendo utilizados son muy útiles tanto para los usuarios como administradores.

Anexo E

Gestión

E.1. Recuperación ante fallas

E.1.1. Tareas de mantenimiento

Para la detección de problemas se destacan los siguientes comandos.

- Revisar el estado del cluster de galera desde el nodo de deploy:

```
# ansible -i /opt/openstack-ansible/inventory/dynamic_inventory.py
galera_container -m shell -a "mysql -h 127.0.0.1 -e 'show status
like \"%srep_cluster_%\";'"
```

- Revisar el estado del cluster de RabbitMQ desde el nodo de deploy:

```
# ansible -i /opt/openstack-ansible/inventory/dynamic_inventory.py
rabbitmq_container -m shell -a "rabbitmqctl cluster_status"
```

Este comando puede llegar a fallar por un bug que presenta Ansible y en su lugar se debe ejecutar el siguiente comando:

```
# ansible -i /opt/openstack-ansible/inventory/dynamic_inventory.py
rabbitmq_container -m shell -a "rabbitmqctl -n rabbit@<
hostname_contenedor_rabbit> cluster_status"
```

- En el firewall de los nodos físicos es importante verificar la existencia de reglas que filtren tráfico válido. Como se mencionó en la sección de preparación de nodos en los capítulos donde se detalla el proceso de instalación, se deben eliminar las siguientes reglas en caso de estar presentes:

```
# iptables -D INPUT -j REJECT --reject-with icmp-host-prohibited
# iptables -D FORWARD -j REJECT --reject-with icmp-host-prohibited
```


E.1.2. Problemas con Ceph

Para diagnosticar y solucionar potenciales problemas con Ceph se toman los documentos oficiales de la herramienta que se pueden encontrar en [7] presentando guías para: realizar el monitoreo del cluster, OSDs y PGs, llevar a cabo operaciones sobre el cluster, realizar una evaluación de problemas sobre los monitores u OSDs, entre otras. Algunos de los comandos utilizados se muestran a continuación.

- Ver el estado del cluster desde el contenedor `ceph_mon`:

```
# ceph health [detail]
```

- Desde los nodos OSD:

- Ver el status de todos los demonios:

```
# systemctl status ceph*.service ceph*.target
```

- Parar todos los demonios:

```
# systemctl stop ceph*.service ceph*.target
```

- Arrancar todos los demonios:

```
# systemctl start ceph.target
```

E.2. Escalamiento horizontal

E.2.1. Agregar nodo de Cómputo

Para agregar un nuevo nodo de cómputo en el Datacenter desplegado con OSA se deben seguir las siguientes instrucciones tanto para la versión Queens como Stein.

1. Configurar el nuevo nodo de cómputo de forma análoga a como se detalla en la sección C.3.1 del anexo C.3.
2. Editar el archivo de configuración principal `/etc/openstack_deploy/openstack_user_config.yml`, agregando el nuevo nodo (en este ejemplo sería el `compute2`) en la sección `compute_host` de la siguiente forma:

```
# nova hypervisors
compute_hosts:
  compute1:
    ip: 10.0.10.12
  compute2:
    ip: 10.0.10.22
```

3. Ejecutar los siguientes comandos desde el nodo de deploy:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible setup-hosts.yml --limit localhost,compute2
# ansible nova_all -m setup -a 'filter=ansible_local gather_subset="!
  all"'
# openstack-ansible setup-openstack.yml --limit localhost,compute2
```

4. Finalmente para verificar que la instalación quedó configurada correctamente se pueden ejecutar los siguientes comandos:

- 4.1. Listar los hipervisores disponibles en el Datacenter

```
[root@infra1-utility-container-161eebae ~]# openstack hypervisor
  list -f json
[
  {
    "Hypervisor Hostname": "compute1.openstack.local",
    "Host IP": "10.0.10.12",
    "State": "up",
    "ID": 1,
    "Hypervisor Type": "QEMU"
  },
  {
    "Hypervisor Hostname": "compute2.openstack.local",
    "Host IP": "10.0.10.22",
    "State": "up",
    "ID": 2,
    "Hypervisor Type": "QEMU"
  }
]
```

- 4.2. Listar las instancias virtualizadas por cada hipervisor

```
[root@infra1-utility-container-161eebae ~]# nova hypervisor-
  servers compute1
+-----+-----+
| Name          | Hypervisor Hostname |
+-----+-----+
| instance-0000000b | compute1.openstack.local |
| instance-00000010 | compute1.openstack.local |
+-----+-----+
```

```
[root@infra1-utility-container-161eebae ~]# nova hypervisor-
servers compute2
+-----+-----+
| Name                | Hypervisor Hostname    |
+-----+-----+
| instance-0000000e   | compute2.openstack.local |
| instance-00000012   | compute2.openstack.local |
| instance-00000013   | compute2.openstack.local |
| instance-00000014   | compute2.openstack.local |
+-----+-----+
```

E.2.2. Eliminar un nodo de cómputo

Para eliminar un nodo de cómputo hay que asegurarse que no exista ninguna instancia corriendo en él, por lo tanto hay que migrar todas las instancias.

Migrar instancias

La guía de referencia utilizada es [45]. Previo a comenzar el proceso de migración es necesario asegurar el acceso SSH entre los nodos de cómputo para el usuario `nova`. Para esto se deben ejecutar los siguientes comandos:

```
# su - nova
$ ssh <IP_nodo_computo>
$ exit
```

Ante la presencia de problemas con las conexiones SSH se puede utilizar el siguiente documento de como referencia [28].

El host al que se migra la instancia es elegido por el componente nova scheduler en base a su configuración. Para saber qué instancias es necesario migrar se puede ejecutar el comando mencionado en la sección anterior:

```
# nova hypervisor-servers <nombre_nodo>
```

Previo a realizar la migración es importante tener en cuenta que el resto de los nodos de cómputo soporten la carga de las instancias alojadas. Para esto se utiliza el comando:

```
# openstack host show <nodo_computo>
```

Este último indica para cada uno de los recursos CPU, RAM y disco: la cantidad total, la utilizada actualmente, la utilizada por todos los proyectos y al final lo utilizado por cada proyecto. Para realizar la migración se puede utilizar simplemente el comando:

```
# openstack server migrate VM_INSTANCE.
```

Por otro lado para obtener el estado de la instancia en el proceso de migración se puede utilizar el script `migrate_instance.sh`:

```
#!/bin/bash

# Provide usage
usage() {
    echo "Usage: $0 VM_ID"
    exit 1
}

[[ $# -eq 0 ]] && usage
VM_ID=$1

# Show the details for the VM
echo "Instance details:"
openstack server show ${VM_ID}

# Migrate the VM to an alternate hypervisor
echo -n "Migrating instance to alternate host "
openstack server migrate ${VM_ID}
while [[ "$(openstack server show ${VM_ID} -f value -c status)" != "
    VERIFY_RESIZE" ]]; do
    echo -n "."
    sleep 2
done
openstack server resize --confirm ${VM_ID}
echo " instance migrated and resized."

# Show the details for the migrated VM
echo "Migrated instance details:"
openstack server show ${VM_ID}

# Pause to allow users to examine VM details
read -p "Pausing, press <enter> to exit."

El mismo se invoca indicando por parámetro el identificador de la instancia en
cuestión.

# ./migrate_instance.sh <id_instancia>
```

En ambos casos los comandos se deben a ejecutar desde el contenedor de utility.

Retomando la acción principal, luego de migrar todas las instancias del nodo de computo a remove, se deben ejecutar los siguientes comandos:

- Se deben detener todos los servicios de cómputo y red corriendo en el nodo:

```
# systemctl stop nova-compute
# systemctl stop neutron-linuxbridge-agent
```

- Luego se debe clonar el repositorio de operaciones de OSA en el nodo deploy:

```
# git clone https://opendev.org/openstack/openstack-ansible-ops /opt/
  openstack-ansible-ops
```

- Ejecutar la playbook `remove_compute_node.yml` con el parámetro indicado:

```
# cd /opt/openstack-ansible-ops/ansible_tools/playbooks
# openstack-ansible remove_compute_node.yml -e host_to_be_removed="<
  name-of-compute-host>" 2>&1 | tee /var/log/openstack/remove-
  compute-node.log
```

- Finalmente actualizar el archivo `openstack_user_config.yml`, eliminando la entrada del nodo de cómputo removido.

E.2.3. Agregar nodo de Infraestructura

Versión Queens

Para agregar un nodo de infraestructura se utilizó la guía del documento [53]. Antes de comenzar se debe configurar el nuevo nodo tal como se indica en la preparación de nodos del anexo C.2. Luego desde el nodo deploy se deben seguir los siguientes pasos:

- Verificar el acceso por SSH al nuevo nodo, en caso de no contar con él, utilizar:

```
# ssh-copy-id <IP_nuevo_nodo>
```

- Respalda la instalación actual:

```
# source_series_backup_file="/openstack/backup-openstack-ansible-
  stein.tar.gz"
# tar zcf ${source_series_backup_file} /etc/openstack_deploy /etc/
  ansible/ /usr/local/bin/openstack-ansible.rc
```

- Agregar el nuevo nodo a la playbook principal `/etc/openstack_deploy/openstack_user_config.yml`:

```
###
### Infrastructure
###

_infrastructure_hosts: &infrastructure_hosts
    infra1:
        ip: 10.0.1.11
    infra2:
        ip: <IP_nuevo_nodo>
```

- Actualizar el inventario de nodos:

```
# /opt/openstack-ansible/inventory/dynamic_inventory.py > /dev/null
```

- Crear el archivo `/root/add_host.limit` con el siguiente contenido:

```
localhost
infra2
infra2-host_containers
```

- Ejecutar la playbook al igual que en el proceso de instalación:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible -vvv setup-everything.yml --limit @/root/add_host
.limit 2>&1 | tee /var/log/openstack/nuevo_infra.log
```

Durante la ejecución de la playbook final surgieron múltiples problemas relacionados con los distintos clusters y con el servidor de repositorio, entre otros. Esto imposibilitó culminar con el proceso correctamente y por lo tanto no se pudo lograr agregar un nodo de infraestructura al Datacenter.

Versión Stein

En este caso se utilizó la misma guía de referencia pero actualizada para la nueva versión [54]. En este caso se debe configurar el nuevo nodo siguiendo los pasos de la preparación de nodos del anexo C.3. Luego desde el nodo deploy se realizan las siguientes configuraciones:

- Verificar el acceso por SSH al nuevo nodo, en caso de no contar con él, utilizar:

```
[root@deploy ~]# ssh-copy-id <IP_nuevo_nodo>
```

- Respaldar la instalación actual:

```
# source_series_backup_file="/openstack/backup-openstack-ansible-stein.tar.gz"
# tar zcf ${source_series_backup_file} /etc/openstack_deploy /etc/ansible/ usr/local/bin/openstack-ansible.rc
```

- Agregar el nuevo nodo a la playbook principal /etc/openstack_deploy/openstack_user_config.yml:

```
###
### Infrastructure
###
_infrastructure_hosts: &infrastructure_hosts
    infra1:
        ip: 10.0.10.11
    infra2:
        ip: <IP_nuevo_nodo>
```

- Actualizar el inventario de nodos:

```
# /opt/openstack-ansible/inventory/dynamic_inventory.py > /dev/null
```

- Crear el archivo /root/add_host.limit con el siguiente contenido:

```
localhost
infra2
infra2-host_containers
rabbit_mq_container
ceph-mon_container
keystone_container
```

- Ejecutar la playbook al igual que en el proceso de instalación:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible -vvv setup-everything.yml --limit @/root/add_host.limit >&1 | tee /var/log/openstack/nuevo_infra.log
```

En esta versión sí fue posible finalizar el proceso con éxito. Cabe resaltar que en la documentación oficial, a la hora de configurar el archivo /root/add_host.limit no se mencionan los contenedores de RabbitMQ, Ceph y Keystone, sin embargo fue necesario agregarlos para que se comunicaran y configuraran correctamente junto con los ya creados en los nodos de infraestructura anteriores.

E.3. Actualización Queens a Rocky

En este caso se especifica cómo actualizar de la versión Queens a Rocky, siguiendo la guía [40]. En esta última se muestran dos opciones para realizar la actualización: mediante un script o en forma manual.

Actualizar mediante un script

En este caso se deben seguir los siguientes pasos:

- Acceder al directorio raíz del repositorio:

```
# cd /opt/openstack-ansible
```

- Moverse al último tag estable de la versión a actualizar, en este caso la última liberación de Rocky es la 18.1.16:

```
# git fetch
# git checkout 18.1.16
```

- Ejecutar el script:

```
# ./scripts/run-upgrade.sh
```

Al utilizar este método surgieron varios problemas que no permitieron completar satisfactoriamente el proceso de actualización. Para detectar con mayor precisión los inconvenientes y poder solucionarlos se tuvo que utilizar el método manual.

Actualizar en forma manual

Los siguientes pasos se corresponden a cada uno de los comandos que se ejecutan en el script `run-upgrade.sh`. Para cada uno de ellos se muestran los inconvenientes que surgieron junto con sus respectivas soluciones.

1. Posicionarse al directorio raíz del repositorio:

```
# cd /opt/openstack-ansible
```

2. Acceder a la última release estable de Rocky:

```
# git fetch
# git checkout 18.1.16
```

3. Definir variables a utilizar en la instalación:


```
# export MAIN_PATH="$(pwd)"
# export SCRIPTS_PATH="${MAIN_PATH}/scripts"
# export UPGRADE_PLAYBOOKS="${SCRIPTS_PATH}/upgrade-utilities/
  playbooks"
```

4. Remover variables del entorno existentes:

```
# unset ANSIBLE_INVENTORY
```

5. Preparar el ambiente para la nueva versión:

```
# ${SCRIPTS_PATH}/bootstrap-ansible.sh
```

6. Acceder al directorio playbooks

```
# cd /opt/openstack-ansible/playbooks
```

7. Preparar el ambiente antes de ejecutar los principales script ejecutando los siguientes comandos:

```
# openstack-ansible "${UPGRADE_PLAYBOOKS}/ansible_fact_cleanup.yml"
# openstack-ansible "${UPGRADE_PLAYBOOKS}/deploy-config-changes.yml"
# openstack-ansible "${UPGRADE_PLAYBOOKS}/user-secrets-adjustment.yml"
"
# openstack-ansible "${UPGRADE_PLAYBOOKS}/pip-conf-removal.yml"
# openstack-ansible "${UPGRADE_PLAYBOOKS}/ceph-galaxy-removal.yml"
# openstack-ansible "${UPGRADE_PLAYBOOKS}/molteniron-role-removal.yml"
"
```

8. Antes de instalar la infraestructura y OpenStack es necesario actualizar los nodos físicos:

```
# openstack-ansible setup-hosts.yml --limit '!galera_all:!'
  rabbitmq_all'
```

Se excluyen los grupos de Galera y RabbitMQ para que los cluster no sufran modificaciones o resets.

9. Por la excepción indicada en el paso anterior es necesario actualizar los contenedores correspondientes a Galera y RabbitMQ de forma independiente:

```
# openstack-ansible lxc-containers-create.yml -e \
  'lxc_container_allow_restarts=false' --limit 'galera_all:rabbitmq_all
  ,
```

10. Para que la configuración aplicada en el paso anterior tome efecto es necesario ejecutar el siguiente comando:

```
# openstack-ansible "${UPGRADE_PLAYBOOKS}/galera-cluster-rolling-  
restart.yml"
```

11. Actualizar la configuración de los servidores de repositorio:

```
# openstack-ansible repo-install.yml
```

12. Actualizar la configuración del HAproxy:

```
# openstack-ansible haproxy-install.yml
```

13. Luego de actualizar el servidor de repositorio se puede utilizar:

```
# openstack-ansible repo-use.yml
```

14. Actualizar la versión de MariaDB:

```
# openstack-ansible galera-install.yml -e 'galera_upgrade=true'
```

15. Actualizar la infraestructura:

```
# openstack-ansible unbound-install.yml  
# openstack-ansible memcached-install.yml  
# openstack-ansible rabbitmq-install.yml -e 'rabbitmq_upgrade=true'  
# openstack-ansible etcd-install.yml  
# openstack-ansible utility-install.yml  
# openstack-ansible rsyslog-install.yml
```

16. Limpiar todo el cache en Memcached:

```
# openstack-ansible "${UPGRADE_PLAYBOOKS}/memcached-flush.yml"
```

17. Actualizar OpenStack:

```
# openstack-ansible setup-openstack.yml
```

Los problemas que ocurrieron en la ejecución de estos pasos se listan a continuación:

- En el paso 8 se obtiene el siguiente mensaje de error:

```
"msg": "Failure talking to yum: Cannot find a valid baseurl for repo:  
base/7/x86\_64"
```

Luego de investigar el problema, se llegó a que la raíz del mismo era que el archivo `/etc/yum.conf` de los contenedores tenía configurada la variable `proxy=http://10.0.1.15:3142` indicando que se utilice el servidor de repositorios local. Esto genera un inconveniente similar al descrito en la sección [A.1](#). Para solucionarlo se modificó dicha variable en todos los contenedores del servidor de infraestructura, generando que se obtengan los repositorios de yum sin pasar por el proxy local.

- En el paso 14 en donde se actualiza MariaDB, a grandes rasgos lo que realiza es bajar el servicio, actualizar MariaDB y luego continuar haciendo modificaciones en la base de datos. El problema es que el script no vuelve a iniciar el servicio de bases de datos y por lo tanto al querer realizar cualquier operación sobre la misma retorna un error. Para solucionarlo en el momento que finaliza la actualización en forma manual se inicializa el servicio mencionado en el contenedor de galera con el siguiente comando:

```
# service mysql start
```

- En el paso 16 se obtuvo un error debido a que el script proporcionado está pensado para ejecutar sobre un sistema Ubuntu, no contemplando que el archivo que se requiere parsear tiene un formato distinto en el sistema CentOS. Este bug fue reportado en [\[33\]](#) y la solución al mismo fue cambiar el código de la última task en la playbook `/opt/openstack-ansible/scripts/upgrade-utilities/playbooks/memcached-flush.yml` por los siguientes comandos:

```
echo 'flush_all' | nc $(awk -F '-1' '/^OPTIONS/ {print $2}' {{
    memcached_conf_dest.get(ansible_os_family | lower) }} | awk '{
    print $1 }') $(awk -F '"' '/^PORT/ {print $2}' {{
    memcached_conf_dest.get(ansible_os_family | lower) }} )
```

Anexo F

Imágenes

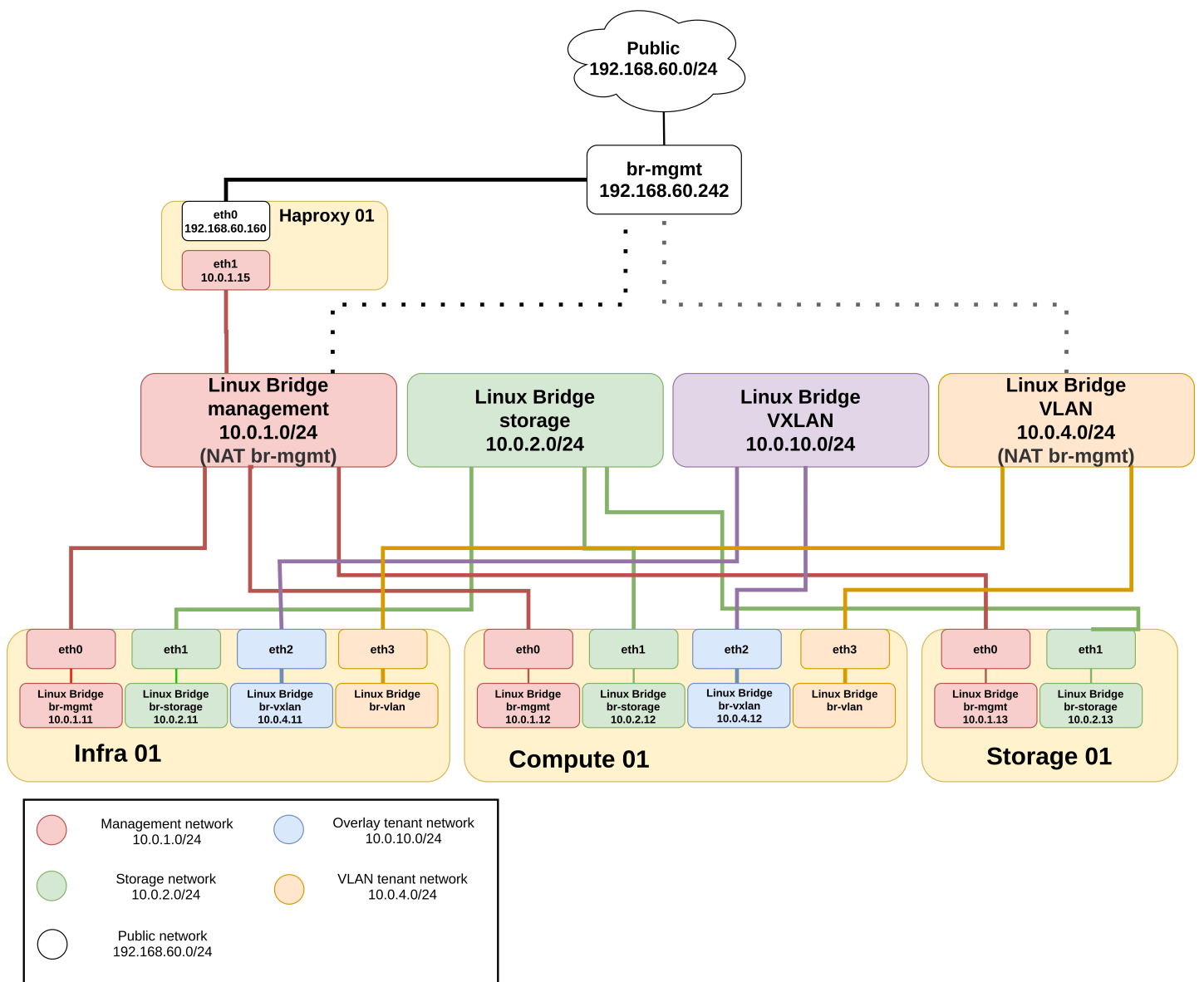


Figura F.1: Arquitectura diseñada para desarrollo

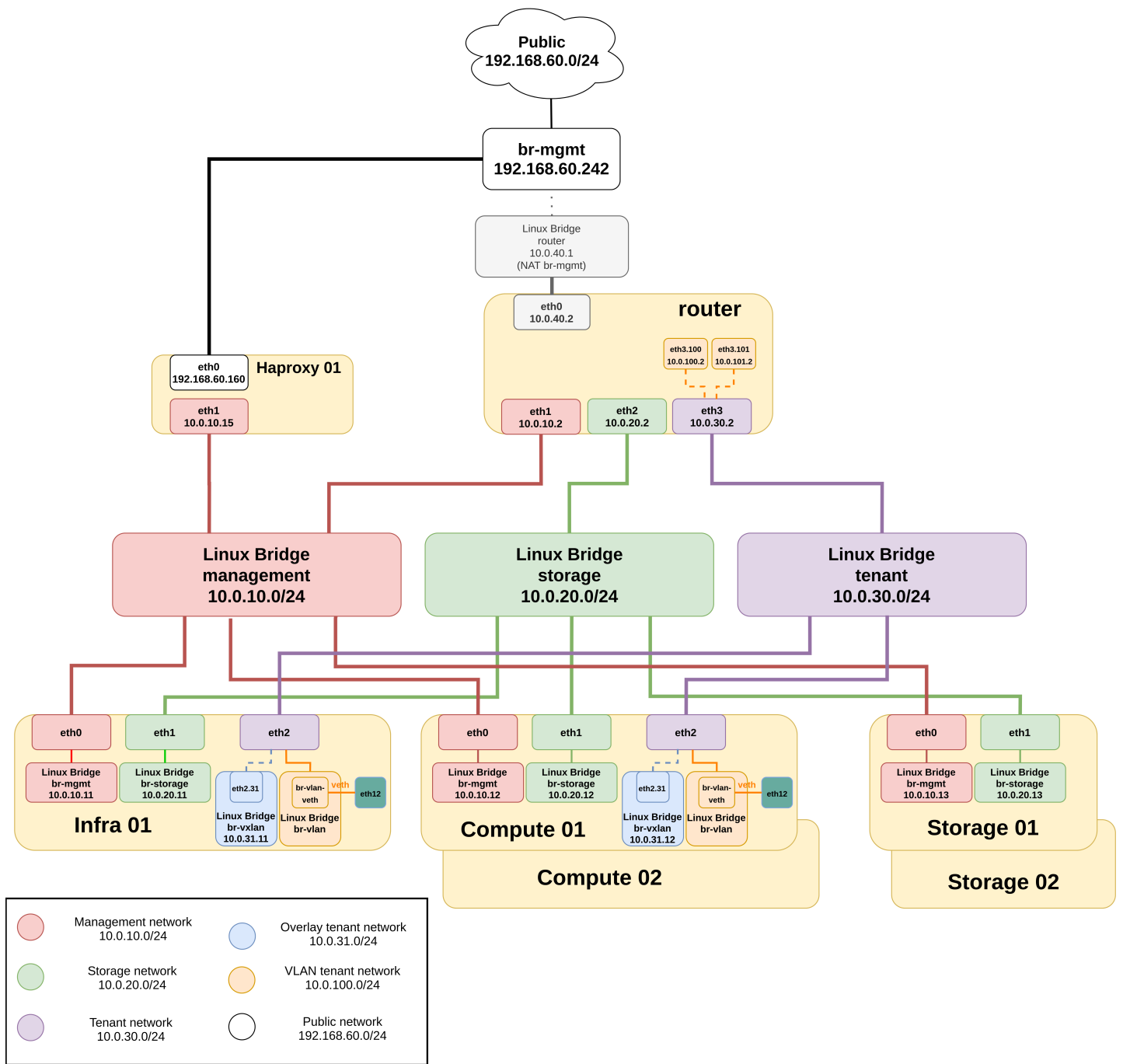


Figura F.2: Arquitectura diseñada para producción