

Despliegue de un Datacenter con Openstack

Matías Capucho, Santiago Elizondo
Universidad de la República, Montevideo, Uruguay.
Email: matias.capucho@fing.edu.uy, selizondo@fing.edu.uy

4 de Agosto de 2019

Resumen

El presente documento estudia los aspectos fundamentales de la plataforma Openstack. A su vez, relata el proceso de diseño, instalación, y configuración básica de un datacenter mediante la utilización de dicha plataforma. Se especifican los pasos necesarios a seguir y se detallan puntos sumamente relevantes a tener en cuenta. Por otro lado, se realiza un análisis del módulo de red Neutron enfocado en resolver la comunicación entre máquinas virtuales instanciadas en el datacenter.

Índice

1. Introducción	3
2. Marco teórico	4
2.1. Cloud computing	4
2.2. Virtualización	5
2.3. Contenerización	6
2.4. Datacenters	6
2.5. Redes	7
3. Openstack	9
3.1. Origen y definición	9
3.2. Módulos Core	9
3.2.1. Keystone	10
3.2.2. Nova	11
3.2.3. Neutron	13
3.2.4. Glance	15
3.2.5. Cinder	17
3.2.6. Swift	19
3.3. Tipos de nodos	20
3.4. Servicios de infraestructura	21
3.5. Métodos de instalación	22
3.5.1. Ansible	23
3.6. Arquitectura	23
3.6.1. Arquitectura de red	24
3.7. Configuración OSA	28
3.7.1. Convenciones	28
3.7.2. Inventario	29
3.7.3. openstack_user_config.yml	29
4. Instalación	30
4.1. Diseño de arquitectura	30
4.2. Ambiente de trabajo	32
4.2.1. Hardware utilizado	32
4.2.2. Conexión remota hacia el servidor renata	33
4.2.3. Virtualización con KVM	33
4.2.4. Especificaciones servidor renata	40
4.2.5. Acceso al exterior desde nodos	41
4.3. Preparación de nodos	42
4.4. Configuración	48
4.4.1. Configuración claves SSH	48
4.4.2. Archivos de configuración OSA	49

4.4.3. Generación de claves	52
4.4.4. Correcciones	52
4.5. Ejecución de playbooks	53
4.6. Verificación	54
5. Interacción	55
5.1. Configuraciones de administrador	56
5.2. Interacción de un usuario	61
5.3. Acceso a una instancia	69
5.3.1. Por SPICE	69
5.3.2. Por SSH	70
6. Inconvenientes	73
7. Análisis del módulo de red	75
7.1. Escenarios de prueba	75
7.1.1. Escenario 1: tráfico este-oeste (misma red tenant)	76
7.1.2. Escenario 2: tráfico este-oeste (distintas redes tenant)	76
7.1.3. Escenario 3: tráfico norte-sur (acceso hacia el exterior)	77
7.1.4. Escenario 4: tráfico norte-sur (acceso desde el exterior)	78
7.2. Linux bridge	79
7.2.1. Escenario 1	80
7.2.2. Escenario 2	87
7.2.3. Escenario 3	93
7.2.4. Escenario 4	98
7.3. Open vSwitch	101
7.3.1. Escenario 1	101
7.3.2. Escenario 2	113
7.3.3. Escenario 3	118
7.3.4. Escenario 4	123
8. Trabajo a futuro	127
9. Conclusiones	128

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 real. En función de esto resulta sumamente interesante investigar alternativas y procedimientos que deben ser llevados a cabo para desplegar y operar un datacenter.

Una tendencia que fue en aumento en los últimos años es la utilización de Openstack como plataforma de despliegue y gestión. Por lo tanto la finalidad de este informe será relatar una primera experiencia con dicha plataforma en una ambiente de prueba en el Instituto de Computación de la Facultad de Ingeniería.

Una vez alcanzado un ambiente estable, es de particular interés analizar cómo la herramienta resuelve la comunicación a nivel de red entre los planos virtuales y físicos del datacenter.

2

Marco teórico

A continuación se presentan conceptos introductorios relevantes para comprender el estudio que se desarrollará más adelante en el informe.

2.1. 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 implementación se aloja en los denominados datacenters, que integran adecuadamente los 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. Según la definición brindada por The National Institute of Standards and Technology (NIST) [15], la computación en la nube debe cumplir con 5 características esenciales:

- **On-demand self-service:** los servicios alojados deben poder obtener capacidad de cómputo a demanda, consumiendo los recursos requeridos y sin que exista interacción humana con proveedores.
- **Broad network access:** los servicios deben encontrarse disponibles a través de la red y accesibles mediante mecanismos estándares.
- **Resource pooling:** los recursos computacionales se mantienen en grupos pudiendo ser asignados a múltiples consumidores en forma dinámica según la demanda necesaria.
- **Rapid elasticity:** los recursos deben poder escalar en forma sencilla e incluso automática en función de la demanda. De esta forma, los consumidores de servicios tendrán una visión de la nube como una fuente de recursos ilimitada.
- **Measured service:** los recursos consumidos deben ser monitorizados y medidos con un determinado nivel de abstracción en función del servicio provisto. Deben existir reportes de consumo que brinden absoluta transparencia tanto al proveedor como al consumidor.

Estas propiedades ambiciosas demuestran que brindar un servicio de cloud no es para nada trivial y requiere grandes conocimientos de TI a la hora de diseñar y poner en marcha un datacenter.

En cuanto a los modelos de cloud computing, existen tres clasificaciones que se distinguen según el tipo de servicio provisto [66]:

- **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 manipular 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.

2.2. Virtualización

Uno de los principales conceptos que se encuentra involucrado en la implementación de cloud computing es el de virtualización [61][65]. 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** [61][56], 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.

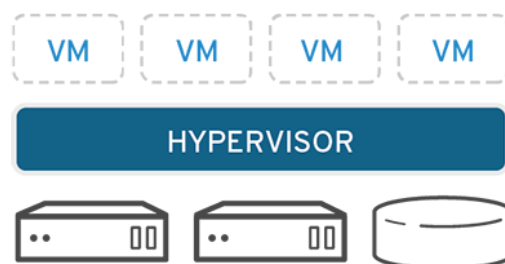


Figura 2.1: Hipervisores. Extraída de [61].

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.

KVM [60], siglas de Kernel-based Virtual Machine, se trata de una tecnología de virtualización open source 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.

2.3. Contenerización

Un contenedor [10] 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.

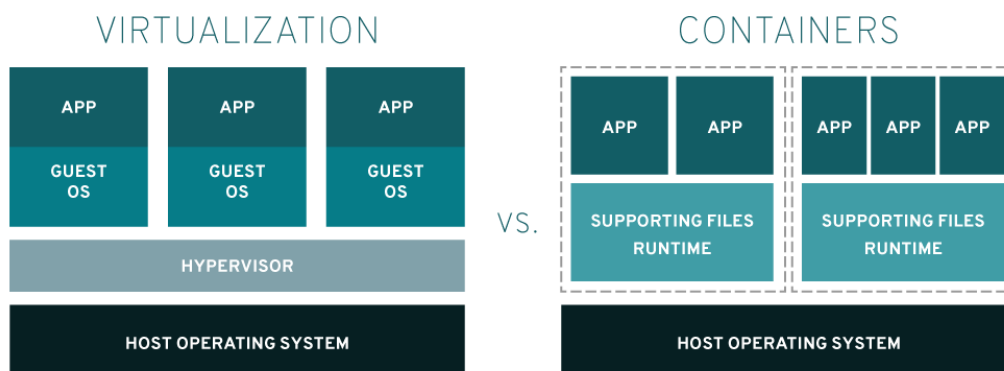


Figura 2.2: Virtualización vs Contenerización. Extraída de [62].

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 data-center de prueba mediante Openstack.

LXC [4][62] Un “Linux container” 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.

2.4. Datacenters

La infraestructura que se encuentra por debajo de la mencionada nube se conoce con el nombre de Datacenter. Un Datacenter [5] es un espacio físico que aloja múltiples componentes de hardware interconectados tales como servidores, racks, switches, routers, sistemas de almacenamiento, etc. Estos

últimos proveen una red de recursos de red, cómputo y almacenamiento, necesaria 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. En función de estos aspectos es que existe un estándar internacional especificado por la ANSI que califica y certifica el diseño de un datacenter. Existen entonces cuatro categorías bajo el estándar ANSI/TIA-942 que se resumen a continuación:

- **TIER 1:** especificado para pequeñas empresas y organizaciones con una infraestructura básica. Ofrece una escasa protección ante riesgos físicos externos, sin la implementación de ninguna redundancia.
- **TIER 2:** se corresponde a Datacenters que posean un mínimo nivel de redundancia a nivel de componentes pero no de distribución eléctrica. Además aumentan su protección en cuanto a eventos físicos, en comparación con el nivel anterior.
- **TIER 3:** suele indicarse para organizaciones que requieren un servicio con disponibilidad 24/7. Mantiene redundancia tanto a nivel de componentes como de distribución eléctrica. Tolera prácticamente cualquier tipo fallas físicas. Para mantener el sistema (ej: recambio de componentes) no es necesario paralizarlo.
- **TIER 4:** enfocado a grandes organizaciones mundiales. Mantiene el mayor nivel de tolerancia a fallas junto con redundancia de componentes y distribuciones eléctricas. Es posible que ocurra un mantenimiento del sistema junto con una falla inesperada sin que el servicio se vea afectado.

2.5. Redes

En la etapa de configuración e instalación y posteriormente en la utilización de Openstack, se refieren 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 fuera múltiples switches. A modo de ejemplo, dos host 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].

Openstack utiliza las VLANs con el fin de aislar el tráfico de datos de diferentes clientes, sin importar en qué servidor físico (nodo de cómputo) estén corriendo las máquinas de los mismos [19].

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 [12]. 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 Endpoint (VTEP) los cuales residen en los nodos físicos. Un listado con pros y contras de las redes de capa 2 y capa 3 se presenta en [43].

3

Openstack

3.1. Origen y definición

Openstack fue creado en los primeros meses del 2010. En ese momento Rackspace quería reescribir el código de infraestructura que corría en sus servidores cloud y además estaban considerando hacer open source el código cloud 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.

El primer Summit de diseño tuvo lugar en Austin, TX del 13 al 14 de julio de 2010, en donde participaron aproximadamente 25 compañías: AMD, Autonomic Resources, Citrix, Cloud.com, Cloudkick, Cloudscaling, CloudSwitch, Dell, enStratus, FathomDB, Intel, iomart Group, Limelight, Nicira, NTT DATA, Opscode, PEER 1, Puppet Labs, RightScale, Riptano, Scalr, SoftLayer, Sonian, Spiceworks, Zenoss y Zuora. El proyecto fue oficialmente anunciado el 21 de julio de ese mismo año.

Originalmente la misión de Openstack era “to produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable”. Luego en 2016 se actualizó la misma incluyendo interoperabilidad y mejor servicio a los usuarios finales. 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 [32]. 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 provisionados a través de APIs”*[50].

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).

3.2. Módulos Core

En esta sección se describirán los módulos Nova, Neutron, Glance, Cinder, Keystone y Swift, llamados core, profundizando en los aspectos más importantes de cada uno. Conceptualmente los módulos se relacionan de la siguiente forma:

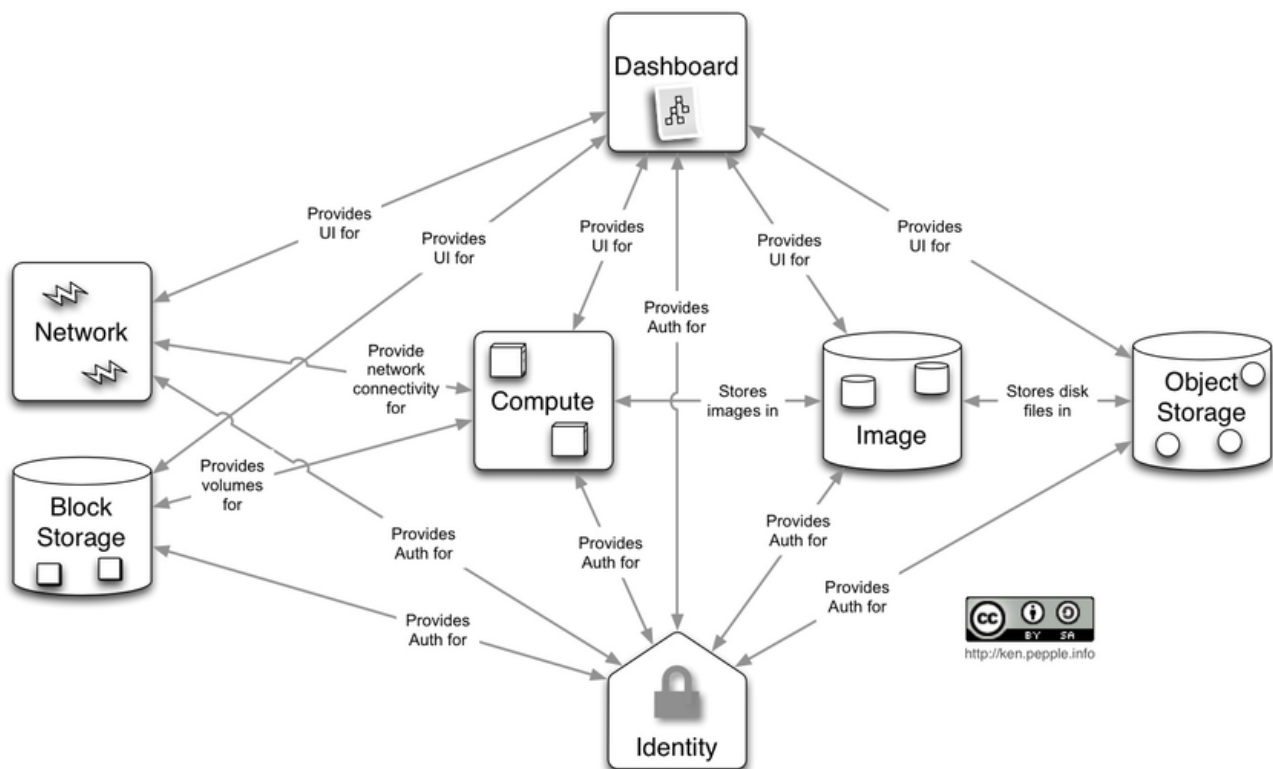


Figura 3.1: Relacionamiento entre módulos core

En [28] se muestra un esquema de una arquitectura lógica estándar de Openstack, en donde se puede apreciar las interacciones internas entre los componentes de un proyecto, las interacciones entre proyectos y las interacciones entre agentes externos y openstack.

3.2.1. Keystone

Keystone es el servicio de identidad que utiliza Openstack para la autenticación y autorización. El mismo se organiza como un conjunto de servicios internos, que se exponen en uno o varios endpoints, listados a continuación:

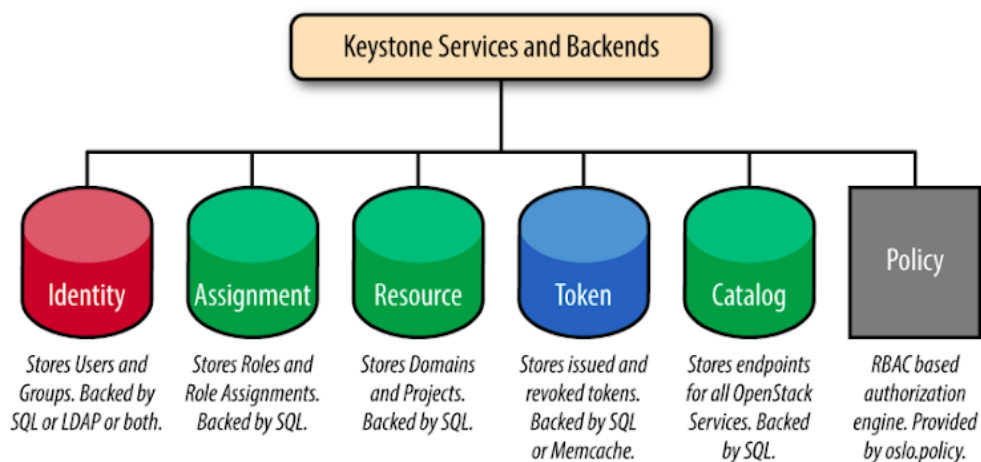


Figura 3.2: Servicios y backends soportados por Keystone. Extraída de [13].

- El **servicio de identidad** autenticación e información de usuarios y grupos. En una instalación estándar este servicio se encarga de todas las operaciones referentes a estos datos, sin embargo

en instalaciones más complejas se pueden configurar distintos backends para llevar a cabo estas tareas. Un ejemplo de esto puede ser LDAP.

- Un usuario representa a un consumidor individual de la API el cual necesariamente debe pertenecer a un dominio y es único dentro del mismo.
- Un grupo representa un conjunto de usuarios, los cuales al igual que los usuarios, deben pertenecer a un único dominio.
- El **servicio de recursos** provee información sobre proyectos y dominios.
 - Los proyectos representan la unidad base de propiedad en Openstack, debido a que todos los recursos deben pertenecer a un proyecto necesariamente. Los proyectos son únicos dentro de cada dominio.
 - Los dominios son grandes contenedores para los proyectos, grupos y usuarios. Por defecto Openstack crea el dominio “Default”. Dada la naturaleza de los dominios los mismos pueden ser utilizados para delegar la administración de los recursos.
- El **servicio de asignación** provee información sobre roles y asignaciones.
 - Un rol indica el nivel de autorización que un usuario final puede obtener. Un rol se puede otorgar a nivel de dominio o proyecto. Por otro lado un rol puede ser asignado a nivel de usuario o grupo.
 - Las asignaciones de roles son tripletas que contienen un rol, un recurso y una identidad.
- El **servicio de Tokens** válida y administra los tokens utilizados para las solicitudes de autenticaciones enviadas luego que las credenciales del usuario fueron validadas.
- El **servicio de catálogo** utilizado para el descubrimiento de endpoints.

Las definiciones mencionadas fueron extraídas de [35] y [14].

3.2.2. Nova

Nova es el proyecto que se encarga de proveer una forma para provisionar 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.

Nova se compone por un conjunto de demonios que permiten ofrecer los servicios mencionados. Adicionalmente para poder desarrollar las funciones básicas, este módulo requiere de los siguientes servicios core de openstack: keystone, glance, neutron y placement.

En la figura 3.3 se puede apreciar un esquema conceptual de los principales componentes de una instalación de nova estándar. Un dato relevante sobre los componentes de nova es que pueden ser escalados horizontalmente, siendo independiente el grado de escalado para cada servicio.

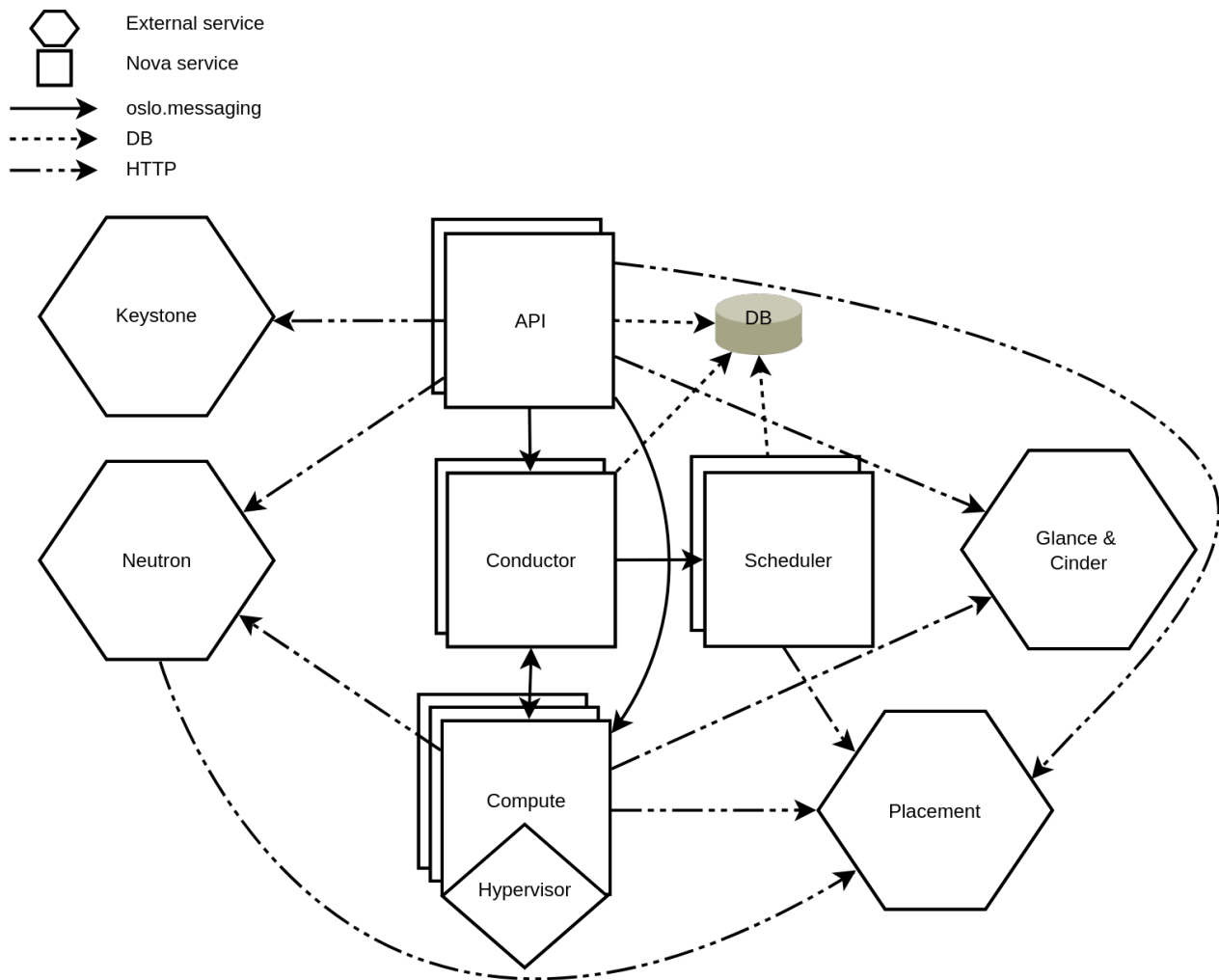


Figura 3.3: Principales componentes de Nova. Extraída de [44].

Internamente los componentes de nova se comunican mediante RPC mientras que la interfaz de cara al usuario es una API REST. Los principales componentes de nova son:

API Se encarga de recibir y responder las solicitudes HTTP y comunicarse con los otros componentes de nova mediante la cola de mensajes.

Scheduler Se encarga de decidir en qué host de cómputo se aloja cada instancia. Para tomar estas decisiones existen diversos algoritmos que pueden ser configurados, siendo algunos ejemplos el algoritmo simple donde selecciona el host con menor carga actual, chance en donde se elige un nodo de forma randómica, entre otros. Esta tarea puede ser muy sencilla como es el caso del algoritmo random o muy complicada para los casos donde se requiera realizar un uso eficiente de los recursos [57].

Compute El módulo nova-compute es principalmente un demonio que se encarga de administrar la comunicación con el hypervisor y con las máquinas virtuales. Esto lo lleva a cabo mediante las APIs del hypervisor. Algunos ejemplos de APIs son: libvirt para KVM o QEMU, XenAPI para XenServer y VMwareAPI para VMware.

Haciendo a un lado la complejidad del módulo, básicamente el demonio acepta acciones de la cola de mensajes para luego realizar una serie de comandos contra la API del hypervisor. Además se encarga de actualizar el estado de la base de datos [22].

Conductor En las primeras versiones de Openstack, todos los servicios del componente de cómputo tenían acceso directo a la base de datos hosteada en el nodo controlador. Esto presentaba dos grandes problemas: seguridad y performance. En lo que respecta a la seguridad, en el caso que un nodo de cómputo sea vea comprometido, el atacante tendrá acceso a la base de datos de Openstack. Por el lado de la performance, las llamadas realizadas desde el nodo de cómputo soportan un único hilo y son bloqueantes. Esto generaba un cuello de botella debido a no poder paralelizar las invocaciones.

Para mejorar estos dos aspectos se introdujo el servicio nova-conductor el cual se presenta como una capa por encima del servicio de cómputo. Nova-compute en lugar de acceder directamente a la BD, delega la responsabilidad al servicio nova-conductor. En otras palabras este servicio actúa como un proxy para el servicio nova-compute.

El problema de seguridad se resuelve dado que los nodos de cómputo dejan de acceder directamente a la BD y el de performance se resuelve gracias a que el servicio de nova-conductor es no bloqueante, permitiendo realizar múltiples invocaciones en paralelo [26]. Por los motivos de su existencia, este servicio no se debe instalar en los nodos de cómputo. [58].

Además puede servir como un lugar donde centralizar y permitir administrar las operaciones que involucran al scheduler y el servicio de computo cómo construir, cambiar el tamaño o migrar instancias. Esto se realiza con el fin de separar las responsabilidades entre los servicios de nova. En [58] se muestra un ejemplo de los beneficios que este cambio presenta.

Placement Este servicio se presenta como una API REST que se encarga de realizar un seguimiento de los proveedores de recursos. Los recursos pueden ser de distintas clases. A modo de ejemplo un proveedor puede ser un nodo de cómputo o un pool de storage.

3.2.3. Neutron

Neutron es el proyecto encargado de proveer y administrar recursos de red 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. El servicio de red 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 un red externa, la cual no es una SDN definida dentro de Openstack sino que es una red accesible por fuera de Openstack. 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 dichas VMs 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 3.4.

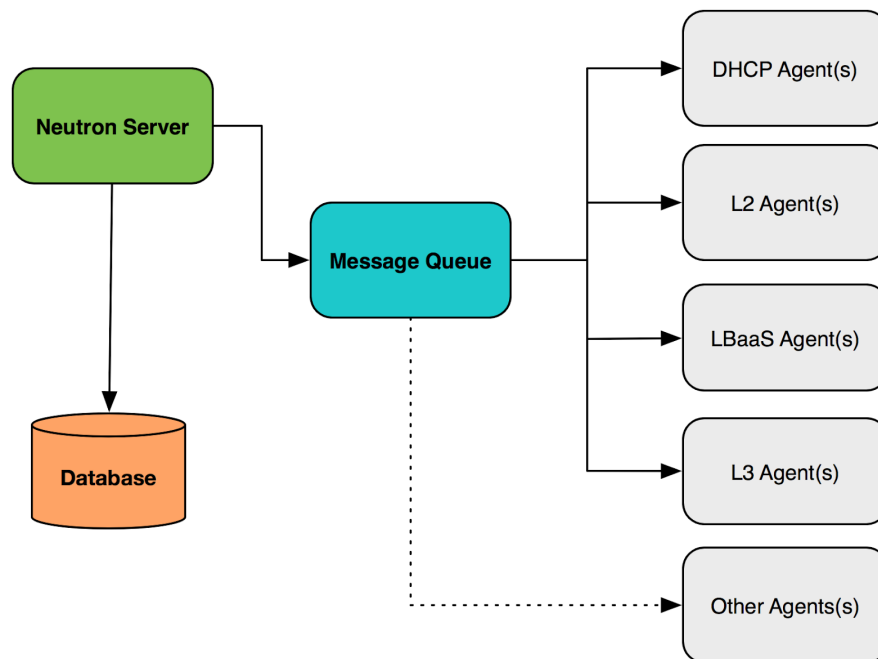


Figura 3.4: Extraída de [6]

Neutron-server El módulo 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 demanda según las funcionalidades y rendimiento requeridos. Estos agentes o plugins serán 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.

En la figura 3.4 se muestra como el servidor de Neutron se conecta con la base de datos que es donde se persisten los recursos de red creados. Por otro lado el servidor acepta solicitudes en la API que expone desde los usuarios y servicios.

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 proyecto. 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 mapearse 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.

Cuando se crea una red de proyecto los aspectos físicos de como 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.

Tipo de 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 [7]. 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 las probabilidades de fallas de red aumentan.

3.2.4. Glance

Glance es el nombre del proyecto utilizado por Openstack para la administración de imágenes. Este servicio permite a los usuarios, a través de una API RESTful, 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, comprendida por los siguientes componentes principales:

- **Cliente:** es quien realiza pedidos a través de la API REST provista.
- **API REST:** permite exponer todos los servicios ofrecidos por Glance.
- **Database Abstraction Layer:** es una API interna encargada de comunicar el servicio glance con la base de datos.
- **Glance Domain Controller:** se trata de un middleware que implementa las principales funcionalidades (autorización, notificaciones, políticas, conexiones a la base de datos).
- **Glance Store:** formado por un conjunto de drivers que permiten comunicar Glance con los diferentes backends de almacenamiento posibles.

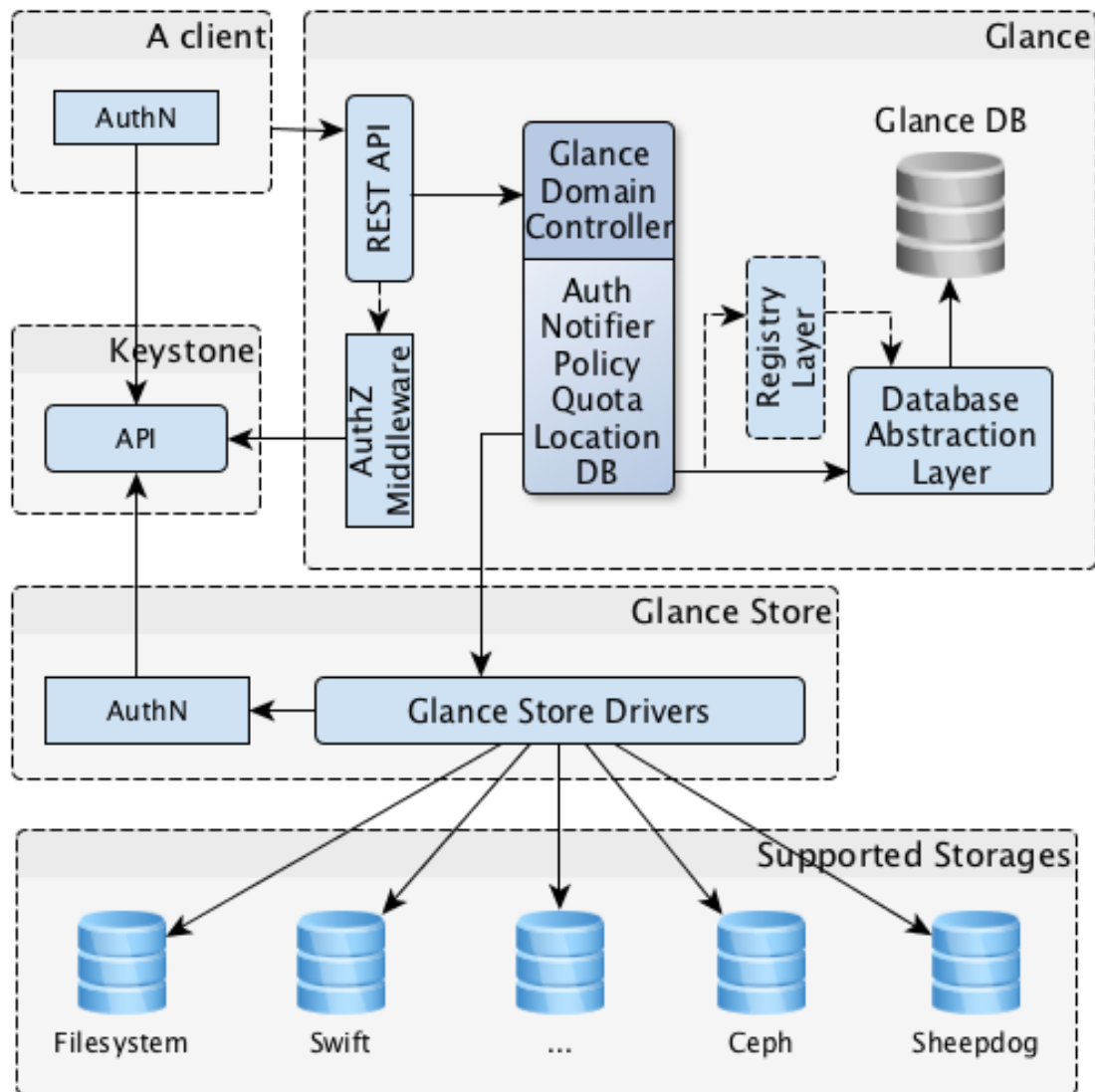


Figura 3.5: Componentes del módulo Glance. Extraída de [18].

Las imágenes son un concepto fundamental en Openstack debido a que son necesarias para crear instancias. Una instancia es una determinada máquina virtual que corre físicamente en un nodo de cómputo específico, siendo creada a partir de una imagen. Además de la imagen es necesario seleccionar un sabor (flavor) que determina las especificaciones de los recursos virtuales asignados a la VM, como pueden ser cantidad de CPUs, MB de memoria RAM y espacio en disco.

Creación de una VM Cuando se lanza una VM, en general y a grandes rasgos se llevan a cabo los siguientes pasos:

1. Se selecciona una imagen administrada por Glance y un flavor que indica tanto los recursos de cómputo necesarios a ser instanciados por Nova como de almacenamiento gestionados por Cinder.
2. El nodo de cómputo copia la imagen de base copiada desde Glance hacia el disco principal de la VM conectándose en forma remota con el servicio de Cinder.
3. El nodo de cómputo provee los recursos virtuales como vCPUs y memoria.
4. La instancia levanta y comienza a ejecutar. Cualquier modificación almacenada en el disco de la

instancia no altera la imagen de base original que seguirá estando disponible para lanzar nuevas instancias.



Figura 3.6: Extraída de [31].

Cuando la instancia deja de existir, se liberan todos los recursos con excepción del almacenamiento persistente en Cinder.

Cada imagen tiene asociada su metadata, que incluye conjunto mínimo de propiedades [20]:

- **architecture**: indica la arquitectura de CPU que debe ser soportada por el hipervisor.
- **instance_uuid**: en caso de tratarse de una snapshot de una instancia, es utilizada para determinar con cual de ellas se encuentra asociada.
- **kernel_id**: el identificador de la imagen que puede ser utilizada como kernel cuando se inicia una imagen AMI (Amazon Machine Image).
- **ramdisk_id**: el identificador de la imagen que puede ser utilizada como ramdisk cuando se inicia una imagen AMI.
- **os_distro**: el nombre común de la distribución del sistema operativo (en minúsculas).
- **os_version**: la versión del sistema operativo, especificada por el distribuidor.

La lista completa de propiedades se encuentra en [53]. Una de las características principales a especificar es el formato de disco o contenedor. El formato de disco de imagen de VM no es más que el formato de la imagen de disco subyacente, este puede ser alguno de los siguientes: raw, vhd, vhdx, vmdk, vdi, iso, ploop, qcow2, aki, ari, ami. Por su parte, el formato de contenedor refiere a aquellos formatos de imágenes que incluyen metadata en sí mismos, pueden ser: bare, ovf, aki, ari, ami, ova, docker.

3.2.5. 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.
- Atachear y desatachar volúmenes a máquinas virtuales.

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

A continuación se muestra la arquitectura de alto nivel de los componentes de Cinder y cómo interactúan:

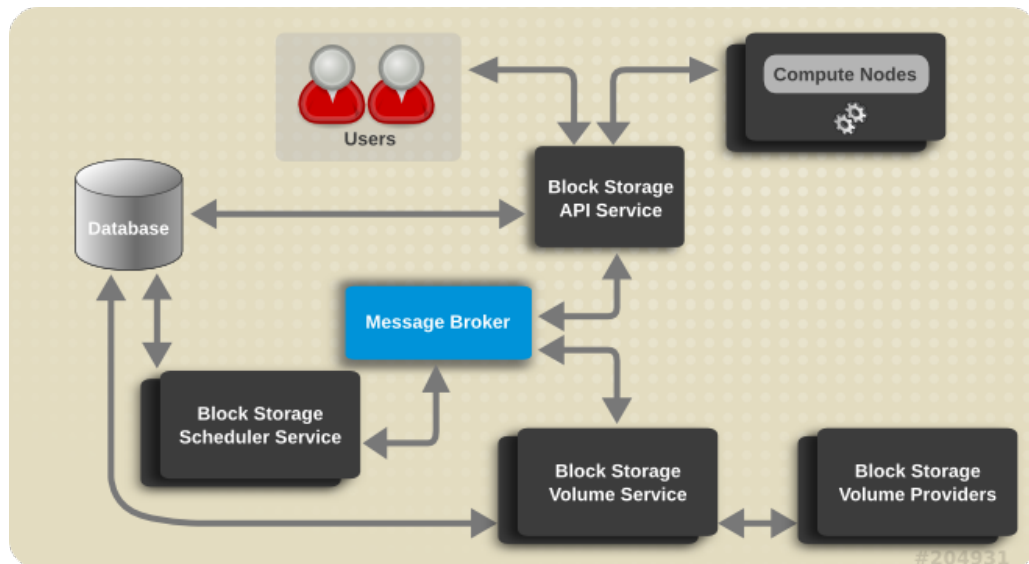


Figura 3.7: Principales componentes de Cinder. Extraído de [59].

El servicio **cinder-volume** administra la interacción con los dispositivos de almacenamiento de bloque. Al recibir una nueva solicitud del scheduler, el servicio crea, modifica o elimina volúmenes a demanda.

Este servicio incluye en su repositorio un conjunto de drivers para soportar diversos dispositivos de almacenamiento para proveer volúmenes. La instalación por defecto utiliza volúmenes locales administrados por Logical Volume Manager (LVM). Además los drivers pueden soportar diversos protocolos de transporte, en el caso de LVM son iSCSI y iSER [37]. En [54] muestran un listado de los drivers disponibles.

El servicio **cinder-api** recibe y responde las solicitudes del módulo. Este servicio al recibir un nuevo mensaje se encarga en primer lugar de verificar que se cumplan los requerimientos de identidad (tokens), luego en base al mensaje construye uno nuevo indicando las acciones a realizar sobre los volúmenes. El mensaje se envía al broker de mensajes para ser procesado por otros servicios del bloque de almacenamiento.

El servicio **cinder-backup** permite crear backups de los volúmenes a repositorios de almacenamiento externos al proyecto, como por ejemplo realizar los respaldos en Swift.

El servicio **cinder-scheduler** planifica y realiza el ruteo de las solicitudes a los servicios cinder-volume apropiados bajo un criterio definido. Dicho criterio puede ser sencillo como realizar un round robin entre los distintos servicios de volúmenes o ser más sofisticado utilizando filtros de planificación. Estos filtros pueden ser fijados sobre la capacidad, el tipo de volumen, las capacidades funcionales, entre otros.

3.2.6. Swift

Se trata del componente de almacenamiento de objetos de Openstack, implementado mediante un sistema distribuido de alta disponibilidad y accesible a través de una API REST. 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.

Swift maneja una jerarquía en tres niveles para organizar el almacenamiento de objetos [45]:

- **Account:** la cuenta es el nivel mas alto en la jerarquía, creando un namespace en el cual residen las instancias del siguiente nivel. A nivel de Openstack una cuenta está dada por un proyecto o usuario tenant.
- **Container:** no es mas que un contenedor de objetos, pero permite gestionar un control de acceso a estos mediante ACLs (no es posible tener ACLs directamente sobre objetos). Como contenedor, define un namespace para los objetos que almacena.
- **Objeto:** es el contenido a almacenar propiamente dicho, como documentos o imágenes, incluyendo la metadata asociada a estos. Todos los objetos tienen una URL que los identifica para poder ser accedidos.

La siguiente imagen ilustra la arquitectura del servicio Swift, en la cual se pueden identificar componentes clave que serán definidos a continuación [21][51].

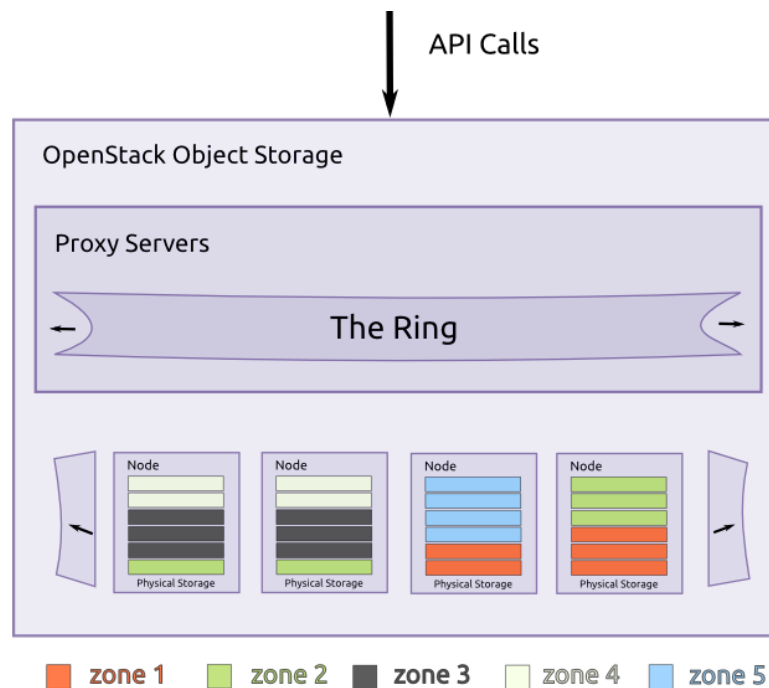


Figura 3.8: Arquitectura del módulo Swift. Extraída de [21].

Principales componentes

- **Proxy servers:** se encargan de comunicar los diferentes componentes del servicio Swift y brindan un acceso público a los usuarios, manejando todos los pedidos que llegan a la API. Para cada uno de los pedidos, busca la ubicación del elemento (cuenta, contenedor u objeto) dentro del anillo para luego acceder al nodo de almacenamiento adecuado.

- **Rings:** un anillo simboliza el mapeo entre los nombres de los elementos almacenados en el cluster y sus ubicaciones físicas, existiendo anillos separados para cada uno de los tipos de elementos. Cuando el sistema debe realizar una operación sobre un elemento debe interactuar con el anillo correspondiente para poder accederlo. A su vez, los anillos gestionan:
 - zonas: para generar aislamiento de información.
 - dispositivos: determina qué dispositivos se encuentra en uso y cuáles utilizar en caso de falla.
 - particiones: distribuidas en forma balanceada en todos los dispositivos.
 - réplicas: cada partición es replicada al menos 3 veces, para no tener un único punto de falla.
- **Zones:** las zonas son utilizadas para aislar la información almacenada y evitar una pérdida total en caso de fallas. Cada réplica de las mencionadas, intenta ser almacenada en una zona diferente. Cada zona puede ser representada desde un solo disco físico separado, hasta racks servidores completos.
- **Accounts and containers:** anteriormente se definieron en forma conceptual, pero como componentes cada cuenta o contenedor es implementado mediante una base de datos SQLite distribuida a lo largo del cluster.
- **Partitions:** cada partición puede estar compuesta por un conjunto de bases de datos de cuentas, bases de datos contenedores y objetos propiamente dichos. Se trata de un agrupamiento de elementos para poder ser trasladados dentro del cluster, facilitando operaciones de replicación.

3.3. 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 [8]:

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 Memcache. 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.

Un ejemplo de como quedan organizados los componentes de Neutron al utilizar nodos de control, red y computo se muestra a continuación:

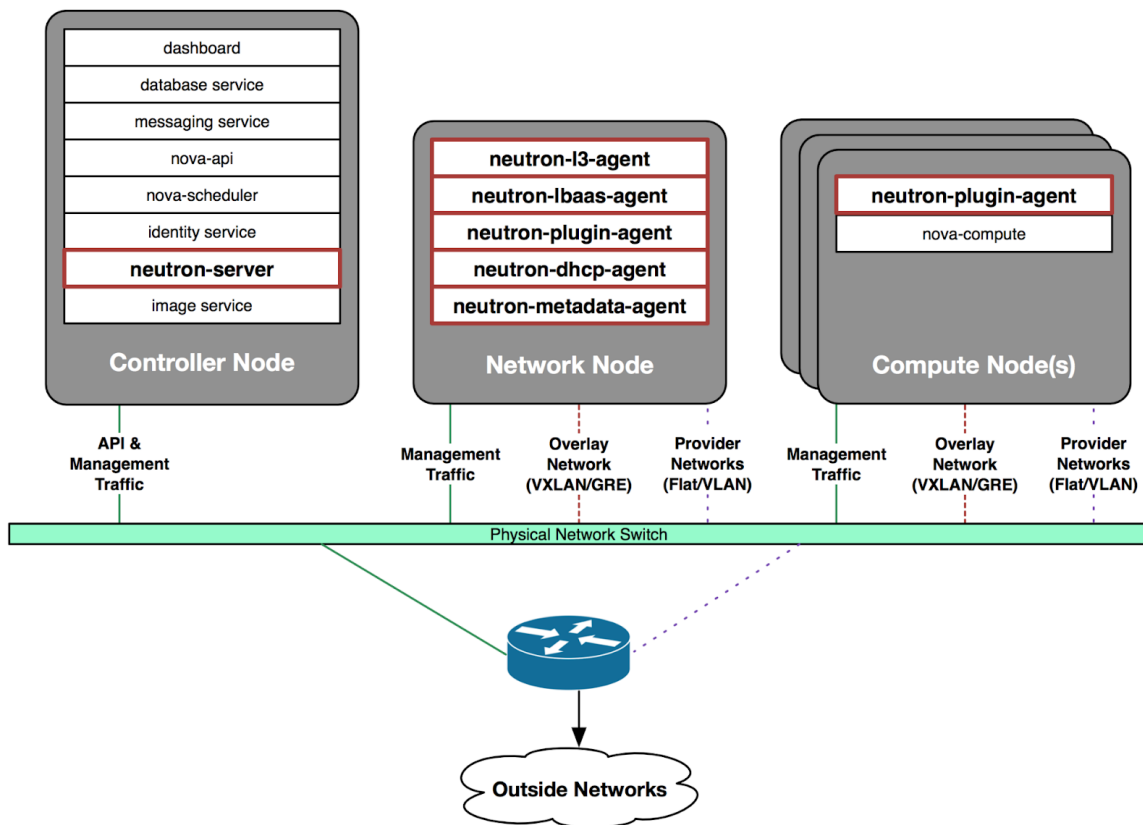


Figura 3.9: Arquitectura de Neutron. Extraída de [8].

En esta arquitectura, la API de Neutron es instalada en el nodo de control, los agentes encargados de realizar tareas específicas son instalados en un nodo dedicado de red y por último, cada nodo de cómputo tiene un agente de red encargado de brindar conectividad a las instancias que aloja.

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 balanceamiento 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.

3.4. Servicios de infraestructura

Estos servicios son transversales a todos los módulos del sistema y son mandatorios 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, el cual 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 [41].

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 [40].

3.5. 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, linux, redes y backends de almacenamiento. Las guías de instalación que se pueden encontrar en el sitio oficial de openstack consisten de cientos de configuración y comandos a ejecutar en donde es muy probable equivocarse y en consecuencia instalar incorrectamente los módulos de openstack.

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 esto existen varias tecnologías como puppet [55] o Ansible [47]. 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 [52].

Otra de las razones principales para utilizar herramientas de automatización además de facilitar la instalación 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). Dicho proyecto se encarga de proveer playbooks y roles para deployar y configurar un ambiente de Openstack. 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 probada para ambientes de cualquier tamaño, desde datacenters de testing hasta producción.

Antes de continuar profundizando en las particularidades de OSA se introducirá brevemente Ansible.

3.5.1. 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 el cual es legible por humano y 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 es 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 siguiente sección 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 [3].

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 setear el ambiente inicial, instalar los componentes de infraestructura, entre otros que se especificarán en las próximas secciones.

3.6. 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 3.10.

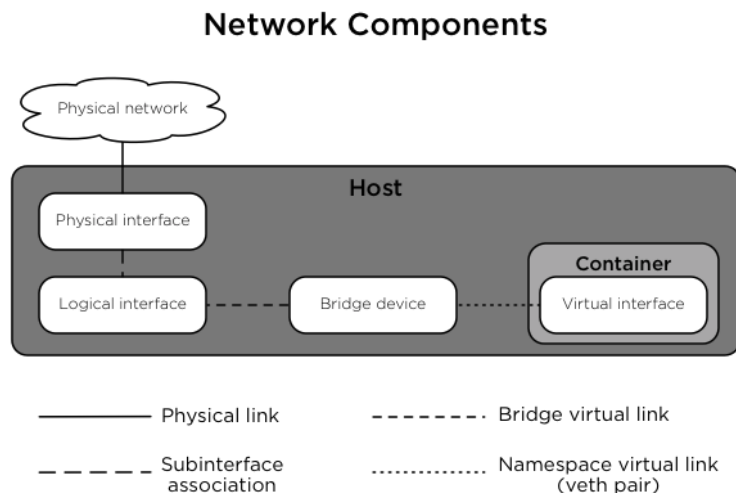


Figura 3.10: Componentes de red en Openstack. [25].

3.6.1. 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 bondeadas mejorando la disponibilidad de los servicios. 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 son VXLAN o GENEVE. La VLAN o interfaz utilizada para esta subred se asocia al bridge br-vxlan. Este bridge debe instanciarse 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 instanciarse en todos los nodos que alojan servicios de cómputo o almacenamiento.

Interfaces de red

Como se mencionó en OSA se pueden tener diversas configuraciones dependiendo de la cantidad de interfaces del host físico, algunas de ellas se muestran a continuación.

Network Interface Layout - Multiple Interfaces

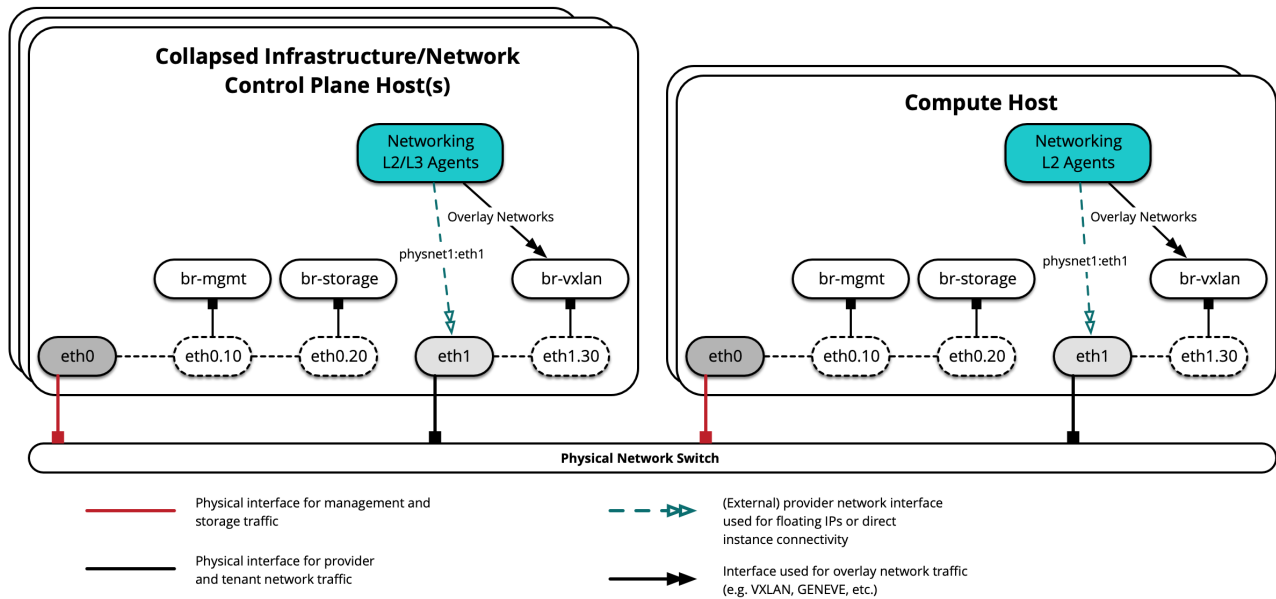


Figura 3.11: Diagrama de múltiples interfaces de red. Extraída de [42].

En la figura 3.11 se utilizan dos interfaces simples, subdivididas mediante la implementación de VLANs que finalmente se asocian a los bridges mencionados anteriormente.

Network Interface Layout - Multiple Bonded Interfaces

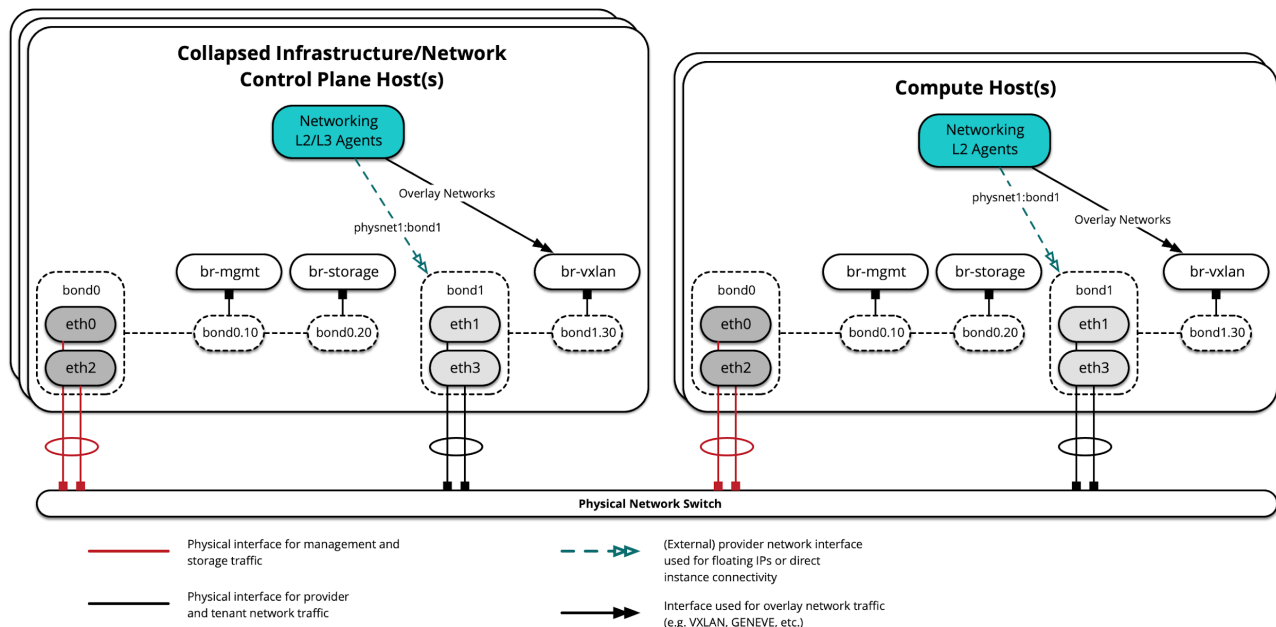


Figura 3.12: Diagrama de bonds de múltiples interfaces de red. Extraída de [42].

En el segundo, se cuenta con cuatro interfaces (provenientes de dos tarjetas físicas) que son bondeadas en forma cruzada para mejorar la disponibilidad y subdivididas mediante la implementación de VLANs para finalmente asociarlas a los bridges.

En ambos casos, se resalta la separación de las redes tenant con las redes internas para el funcionamiento de OSA y la ausencia de veth pairs asociadas a los bridges debido a que no se visualizan servicios desplegados en contenedores.

A continuación se presenta un diseño que ilustra servicios desplegados en contenedores y cómo varían las interconexiones de red para proveer conectividad.

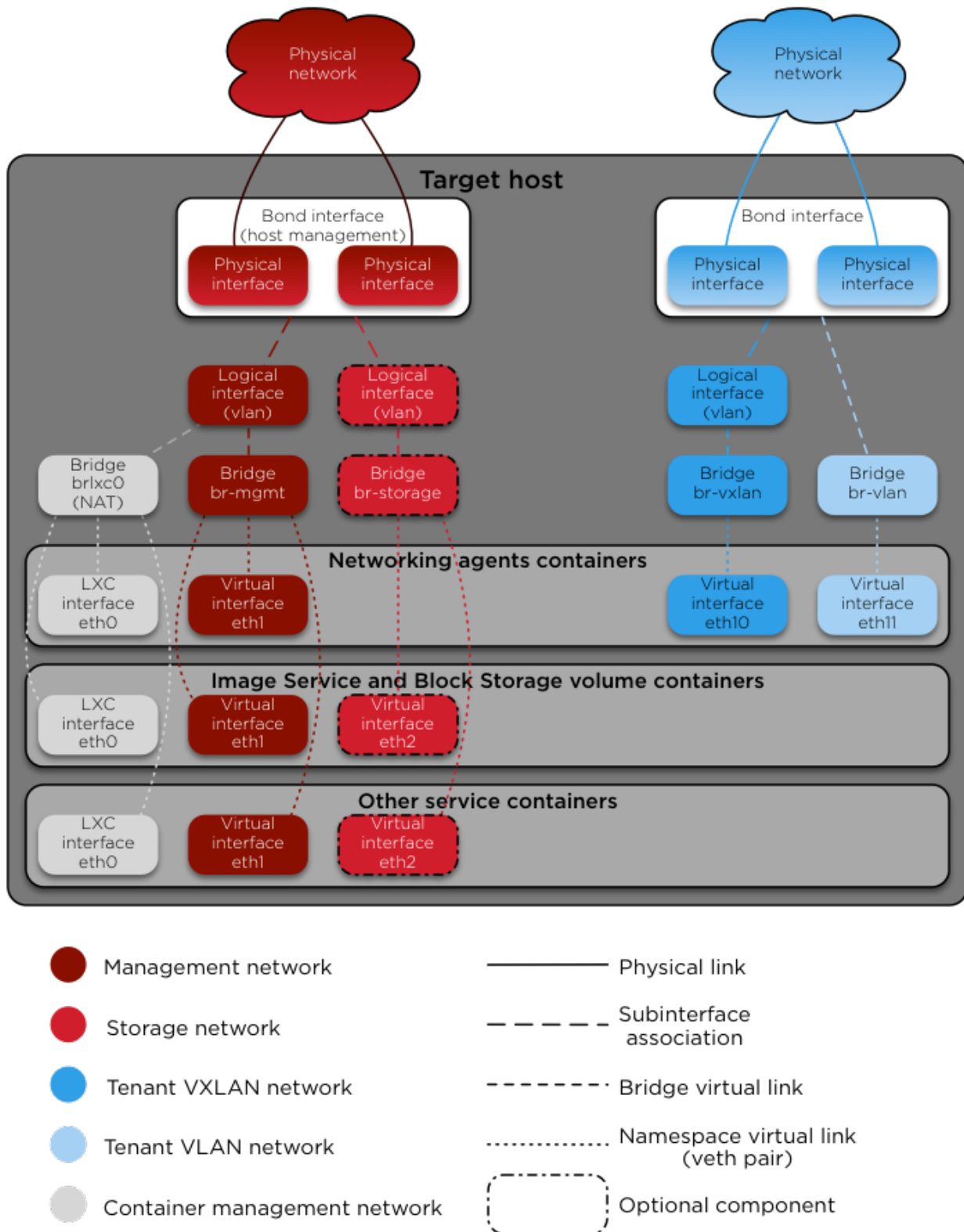


Figura 3.13: Despliegue de servicios Openstack en contenedores. Extraída de [17].

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.

A continuación se presenta el diagrama de una arquitectura modelo de Openstack Ansible en un ambiente de producción que involucra los conceptos mencionados en puntos anteriores.

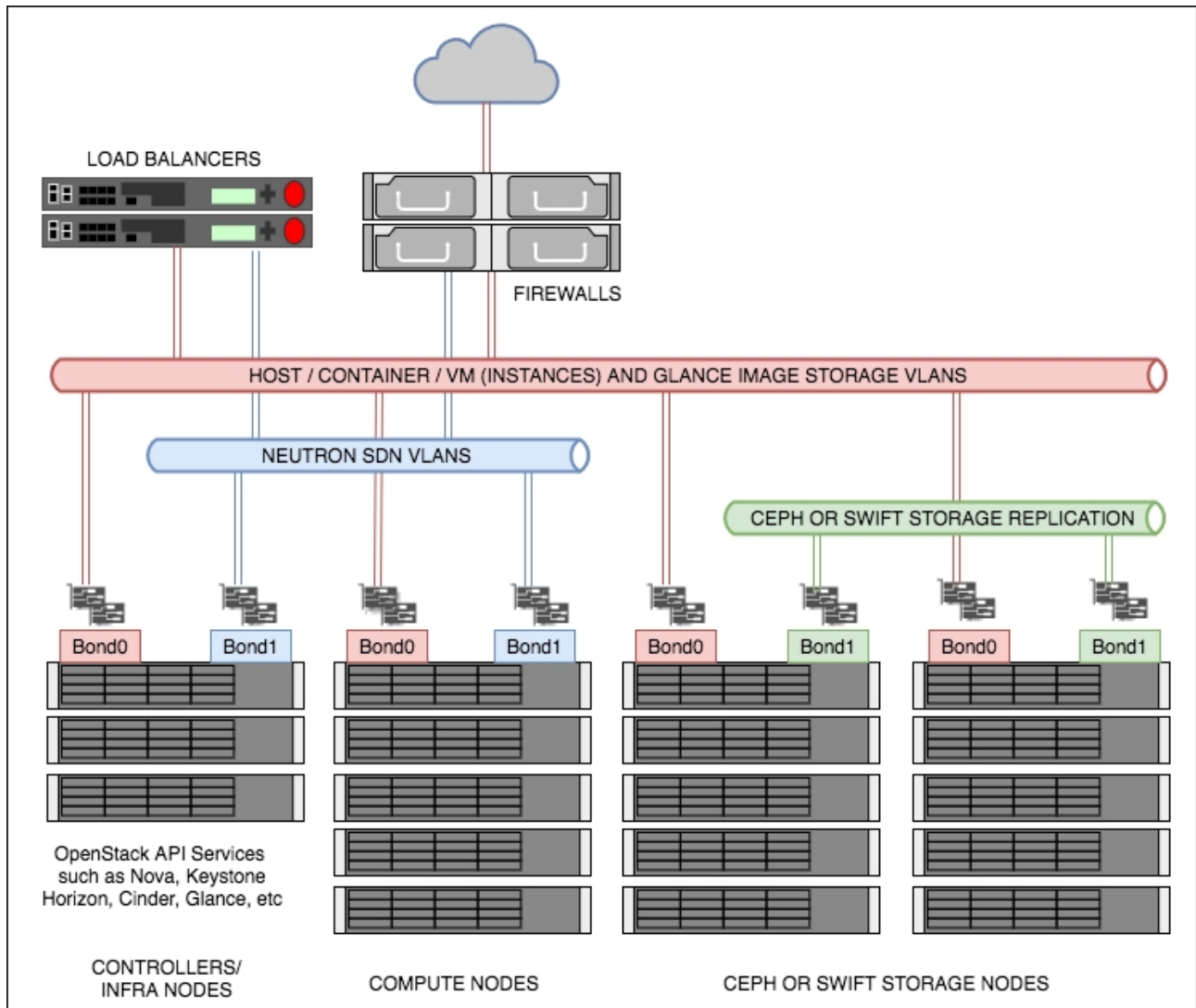


Figura 3.14: Extraída de [11]

3.7. 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.

3.7.1. 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`.

yaml mediante la ejecución del `scriptbootstrap-ansible.sh`.

- Las configuraciones realizadas por el administrador son indicadas en el directorio `/etc/openstack_deploy`.

3.7.2. 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 directorio `conf.d/`
 - El directorio `env.d/`

Este archivo es considerado 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.

3.7.3. `openstack_user_config.yml`

Es el principal archivo de configuración, creado por el operador de Openstack. Las especificaciones de cada sección se detallan en [48].

4

Instalación

4.1. Diseño de arquitectura

A continuación se presenta un diagrama 4.1 de la arquitectura diseñada para la instalación de Openstack Ansible. Al tratarse de un ambiente meramente de prueba, sólo se cuenta con un nodo de cada tipo definido previamente. 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 virtualizarán todos los nodos, con el fin de tener acceso externo a la plataforma.

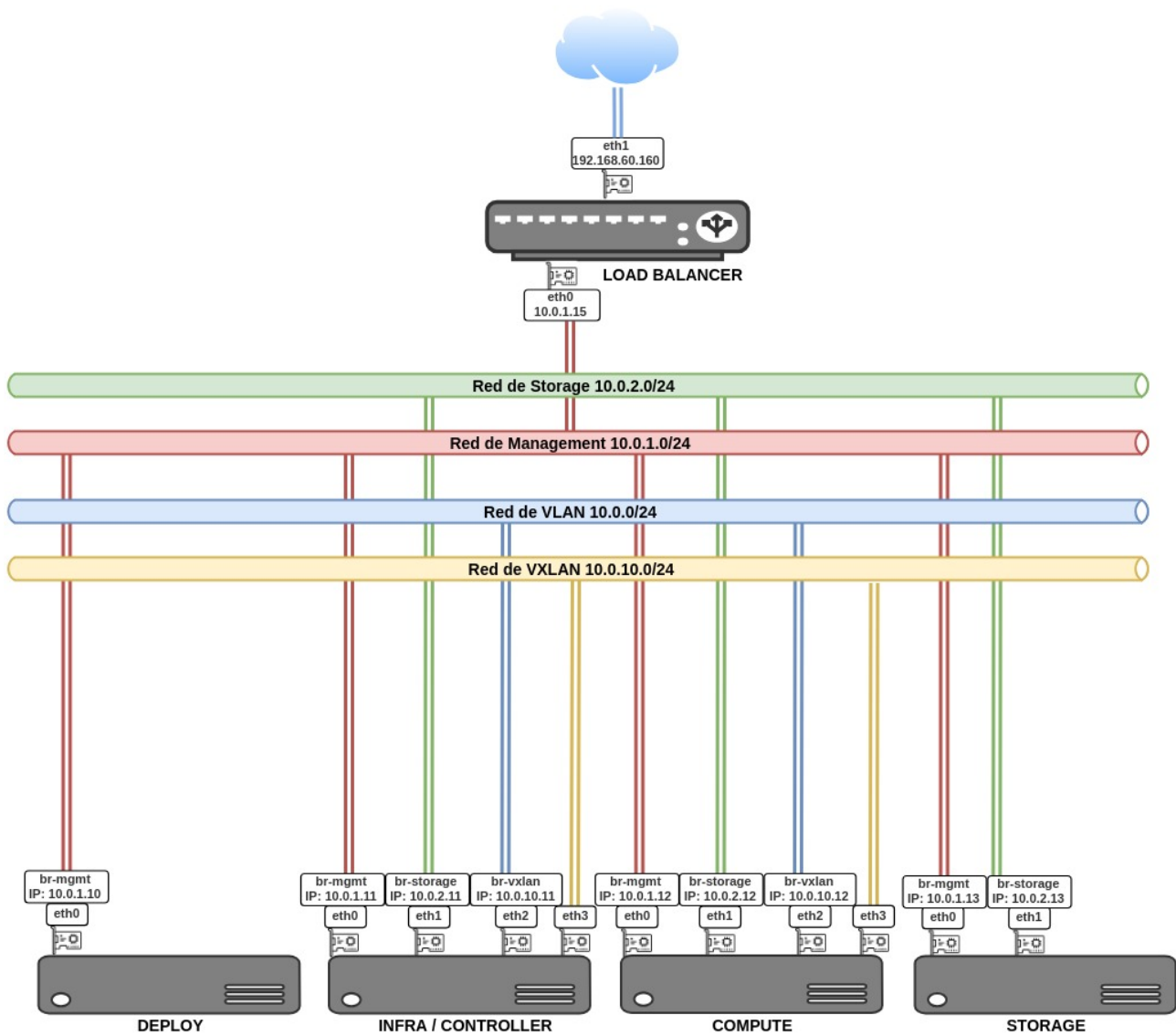


Figura 4.1: Arquitectura diseñada.

El siguiente diagrama muestra la disposición de los componentes de red en la arquitectura utilizada en donde el nodo de control y red se colapsaron a uno solo:

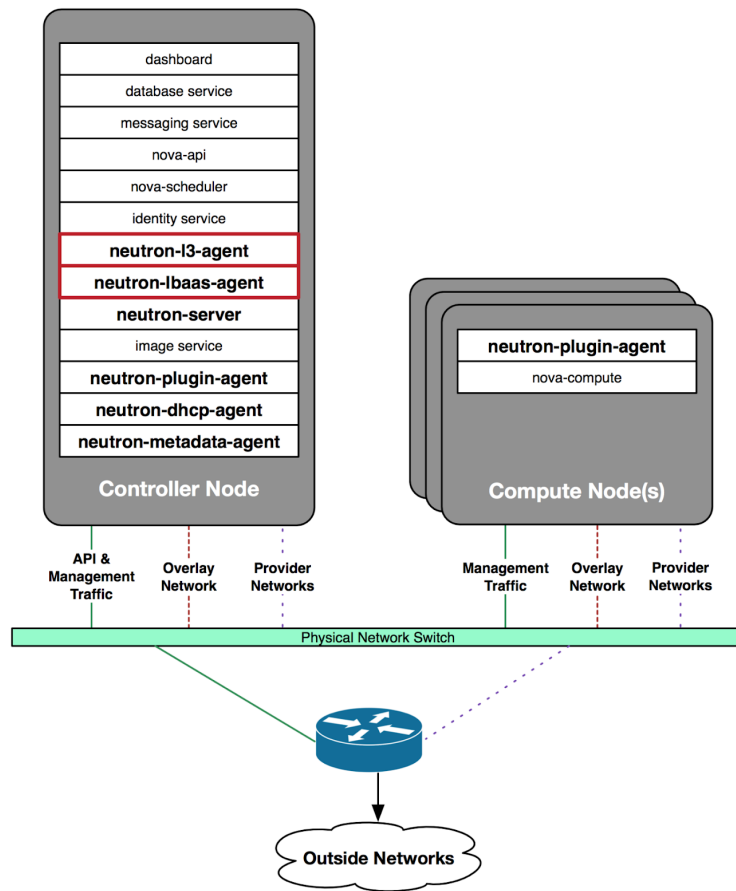


Figura 4.2: Disposición de componentes en Neutron. Extraída de [9].

4.2. Ambiente de trabajo

4.2.1. 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=en02
UUID=824cd835-662a-4d47-a148-512aec3dd237
DEVICE=en02
ONBOOT=yes
BRIDGE="br-mgmt"
NM_CONTROLLED="no"
```

4.2.2. 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.

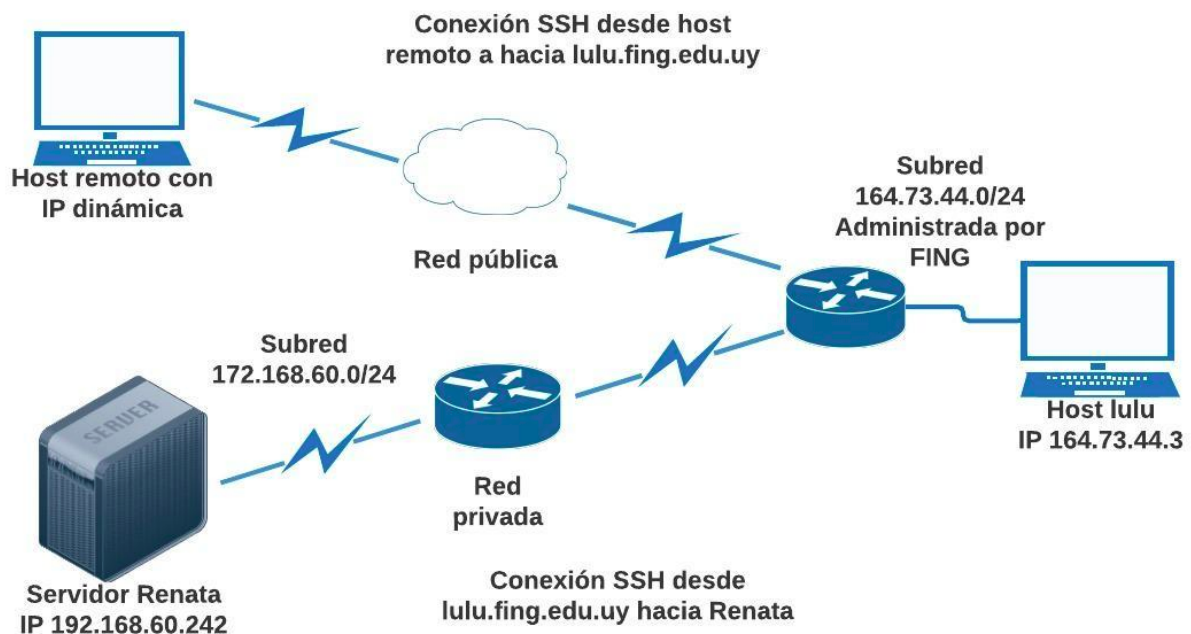


Figura 4.3: Acceso remoto al servidor renata.

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
```

4.2.3. 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.

Utilización virt-manager

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

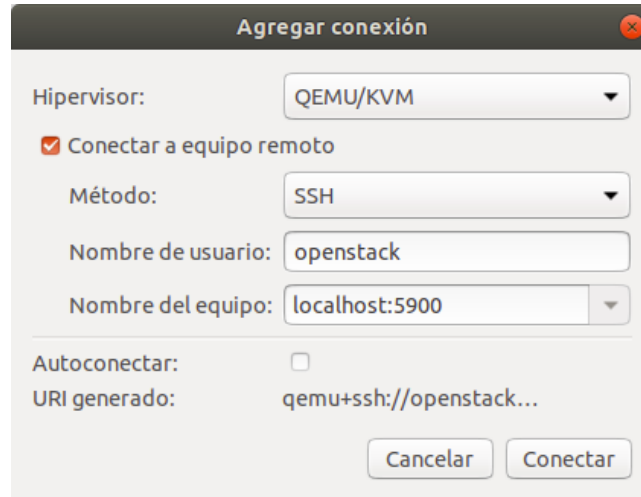


Figura 4.4: Nueva conexión en virt-manager.

Como un host remoto está a dos conexiones SSH del servidor Renata, la configuración que se muestra en la figura X 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.

Lo siguiente a realizar es crear con KVM las redes virtuales que se natean al bridge `br-mgmt` del servidor físico. Es necesario crear 4 redes, donde cada una se corresponde con las redes necesarias para el funcionamiento de Openstack. Estas redes son:

- NAT-Open (Red management Subred 10.0.1.0/24).
- NAT-Open-Storage (Red storage Subred 10.0.10.0/24).
- NAT-Open-Vxlan (Red vxlan Subred 10.0.2.0/24).
- NAT-Open-Vlan (Red vlan Subred 10.0.4.0/24).

Para crear estas redes 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 (+).

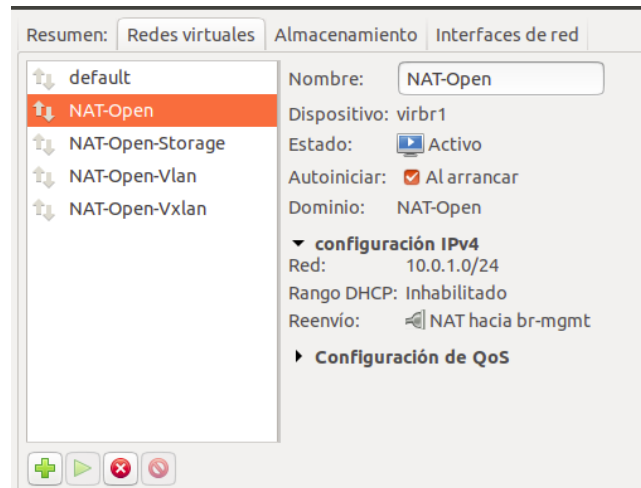
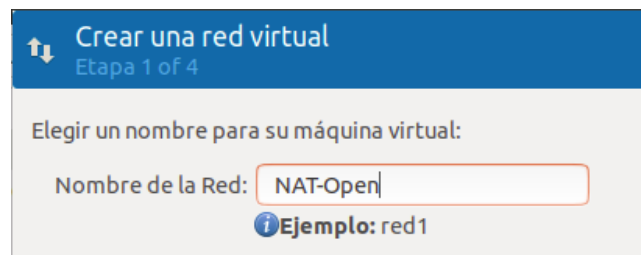


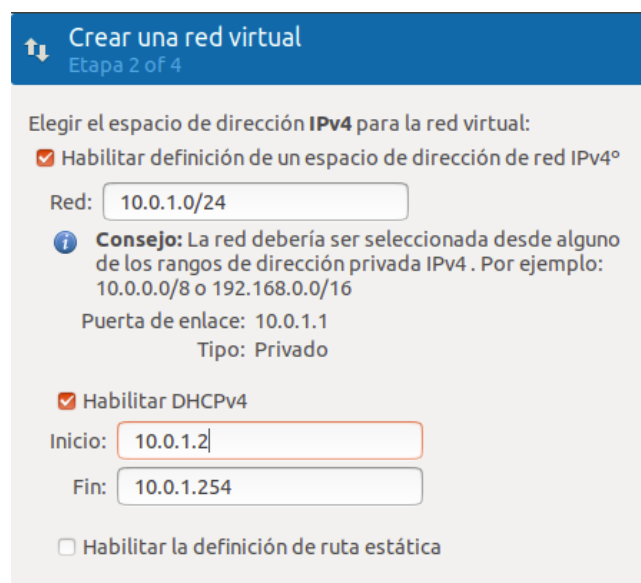
Figura 4.5: Configuración de redes virtuales en virt-manager.

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

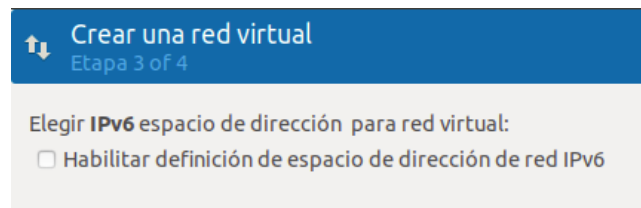
1. Seleccionar el nombre de la red



2. Seleccionar la subred y el rango para el DHCP



3. No habilitar direcciones IPv6

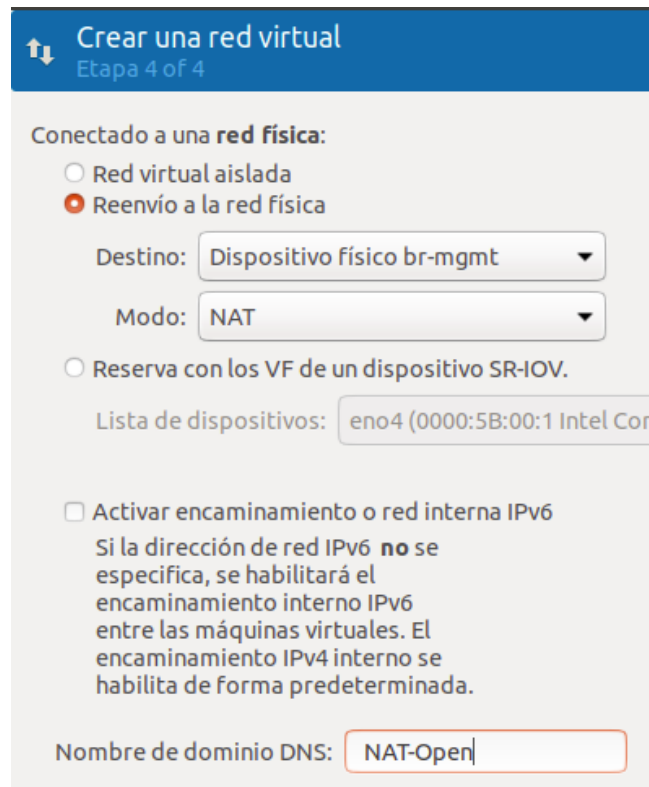


Crear una red virtual
Etapa 3 of 4

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: Dispositivo físico br-mgmt

Modo: NAT

☐ Reserva con los VF de un dispositivo SR-IOV.

Lista de dispositivos: eno4 (0000:5B:00:1 Intel Cor)

☐ 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: NAT-Open

El mismo procedimiento se deberá repetir con el resto de las redes listadas.

Lo último que es necesario crear con KVM son los nodos virtuales que se utilizaron para instalar Openstack. En la instalación realizada se utilizan 5 nodos con las especificaciones detalladas a continuación:

■ Nodo deploy:

- 1 interfaz en NAT-Open
- 2 CPUs
- 200 GB de disco
- 8 GB de RAM

■ Nodo haproxy1:

- 2 interfaces: una en NAT-Open y otra conectada al bridge br-mgmt de renata
- 4 CPUs
- 200 GB de disco
- 32 GB de RAM

■ Nodo infra1:

- 4 interfaces: una en cada NAT creada
- 8 CPUs
- 200 GB de disco
- 32 GB de RAM

■ Nodo compute1:

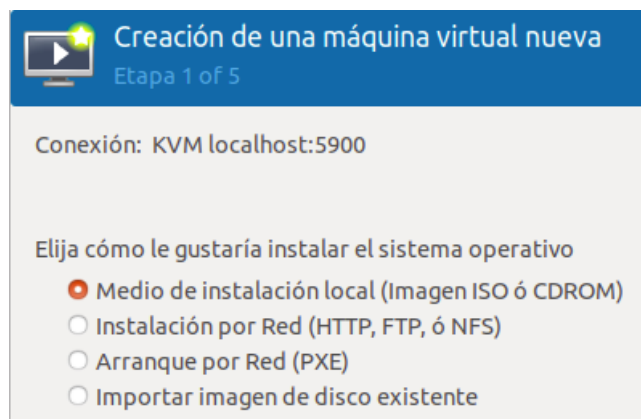
- 4 interfaces: una en cada NAT creada
- 8 CPUs
- 200 GB de disco
- 32 GB de RAM

■ Nodo storage1:

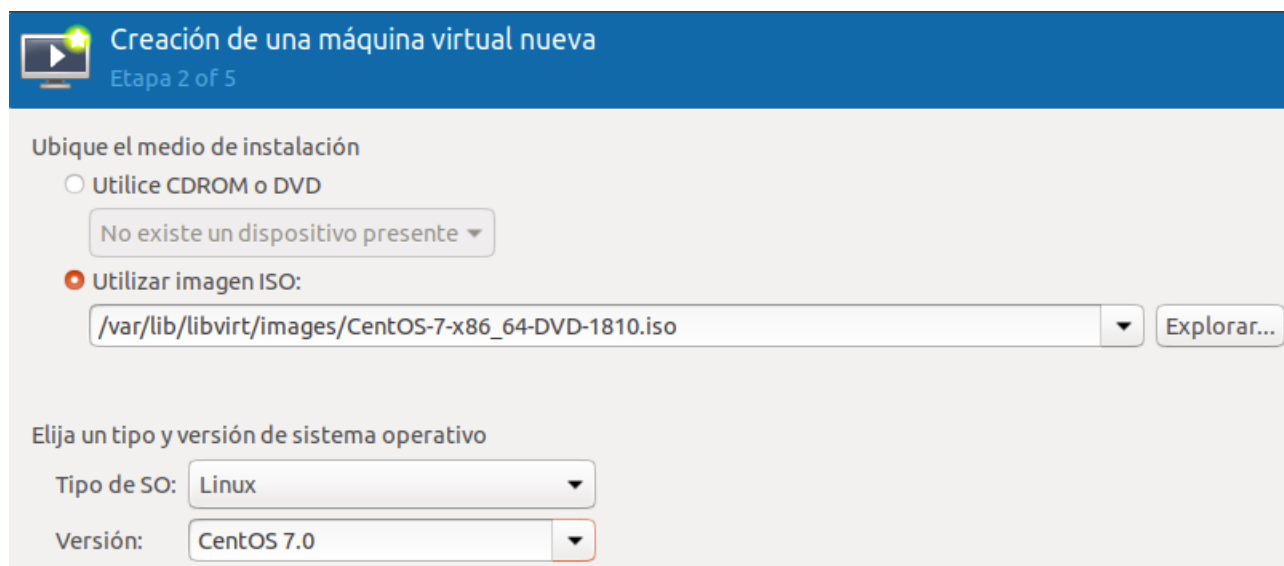
- 2 interfaces: una en NAT-Open y otra en NAT-Open-Storage
- 4 CPUs
- 2 discos: uno de 40 GB para el SO y otro de 200 GB para el volumen de cinder
- 32 GB de RAM

A continuación se detalla paso a paso cómo crear el nodo deploy, el resto se realiza de forma análoga salvando las diferencias de recursos e interfaces de red.

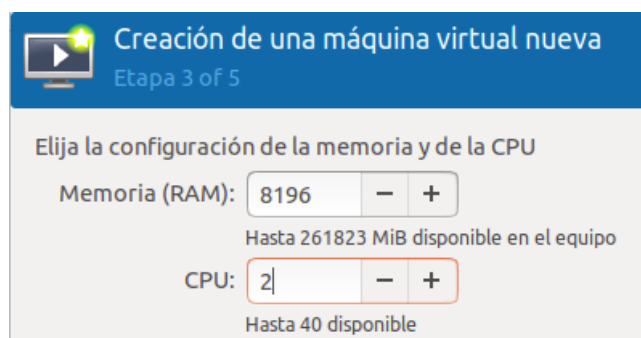
1. En el presente ejemplo se crea una VM suponiendo que se instalará el SO.



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

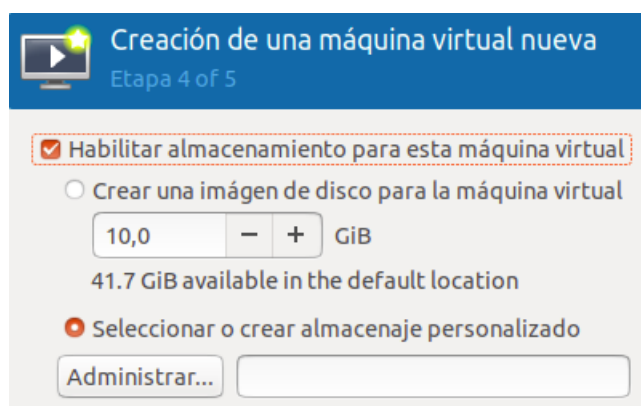


3. Seleccionar RAM y CPUs.

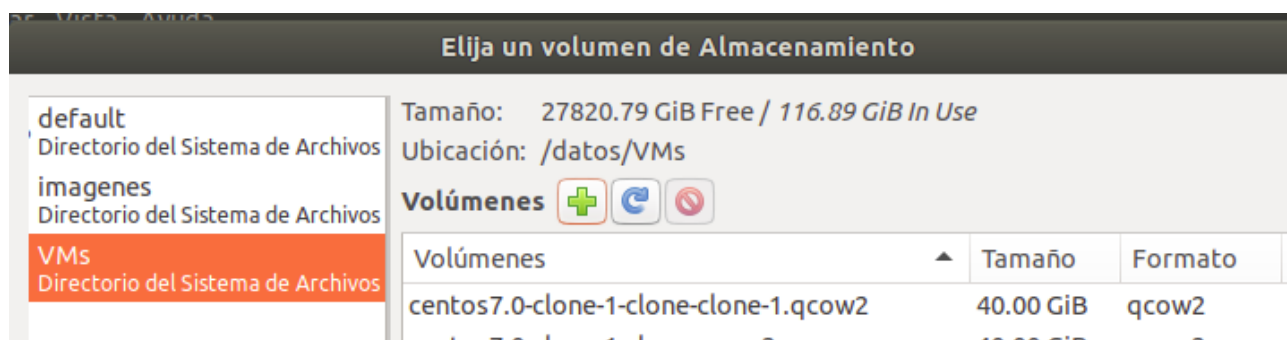


4. Para el almacenamiento se deberá crear un nuevo volumen de la siguiente forma:

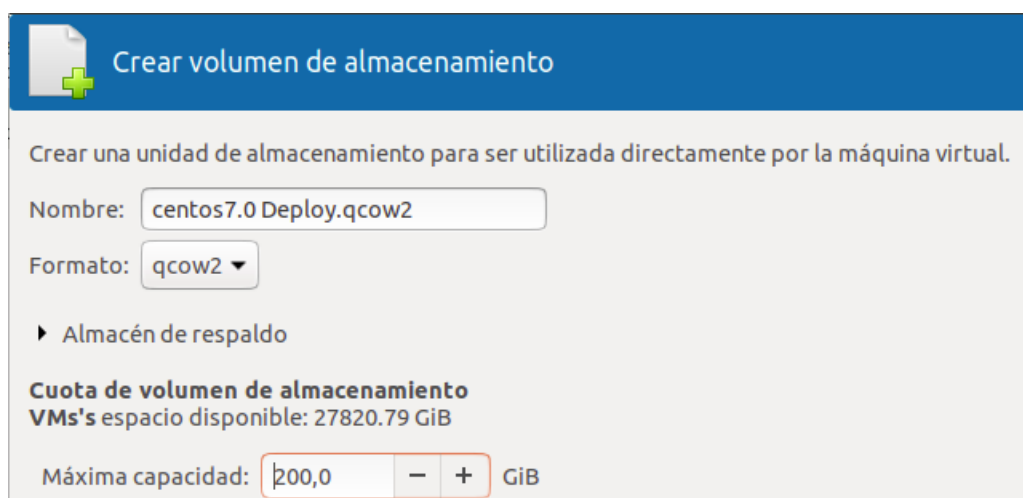
4.1. Elegir la segunda opción y presionar Administrar.



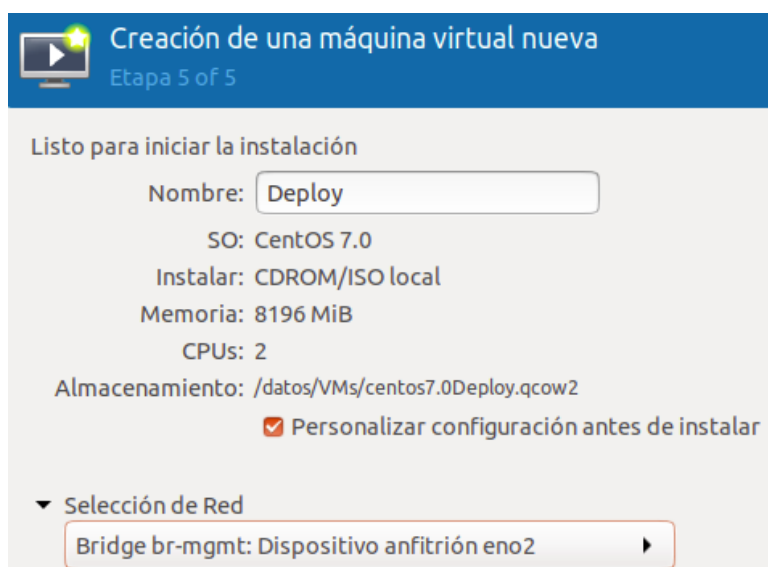
4.2. Esto desplegará una ventana mostrando directorios del sistema de archivos de renata. Se puede elegir un volumen existente o crear uno nuevo con el botón (+).



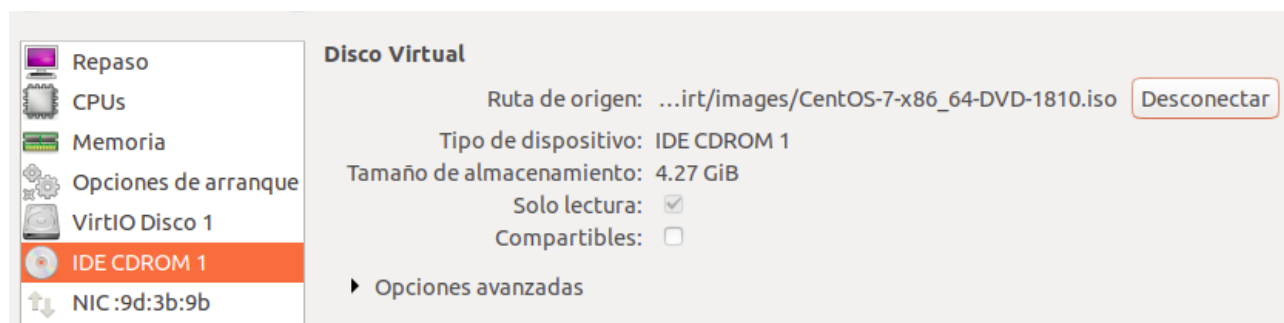
4.3. Al crearlo se debe especificar el nombre, el tipo y la capacidad.



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.



6. Se debe verificar en la opción IDE CDROM 1 que esté correctamente seteado 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 bootee con la imagen seleccionada.



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

4.2.4. 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 reverso desde el host lulu.fing.edu.uy al servidor Renata para establecer como proxy de Renata el host proxy.fing.edu.uy, siendo este último el proxy de la FING.

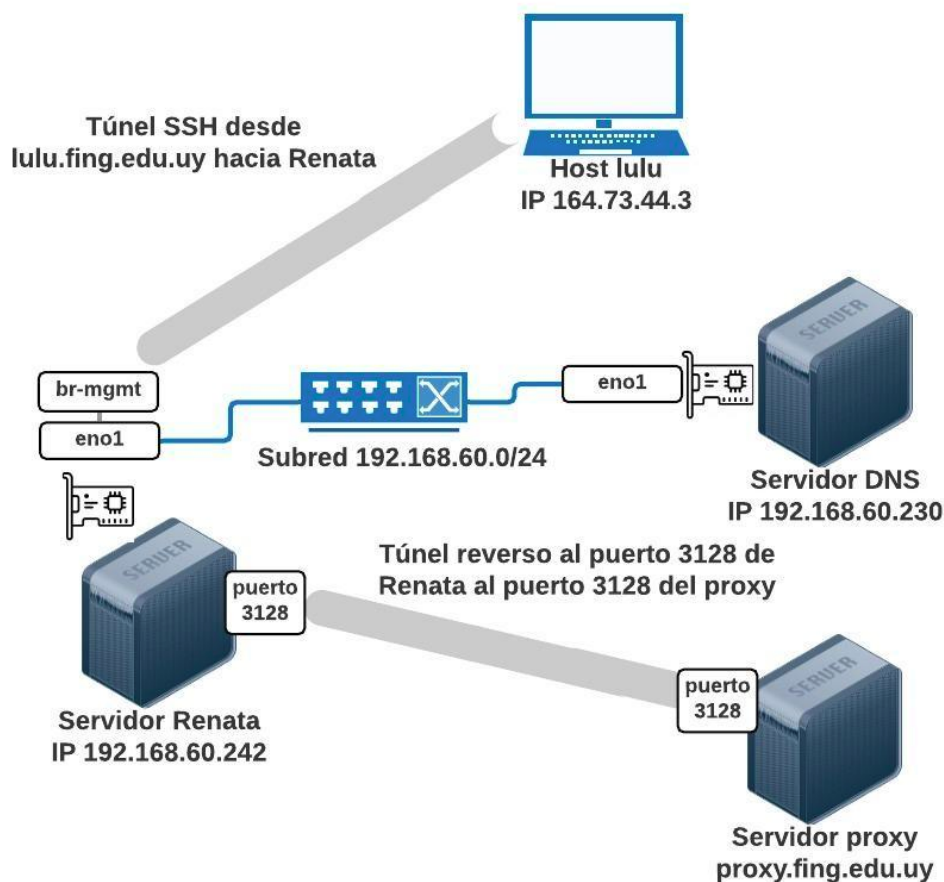


Figura 4.6: Túnel reverso y esquema de servidores.

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 setean 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 reverso 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 4.6 el servidor DNS utilizado se encuentra en la misma red local.

4.2.5. 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.

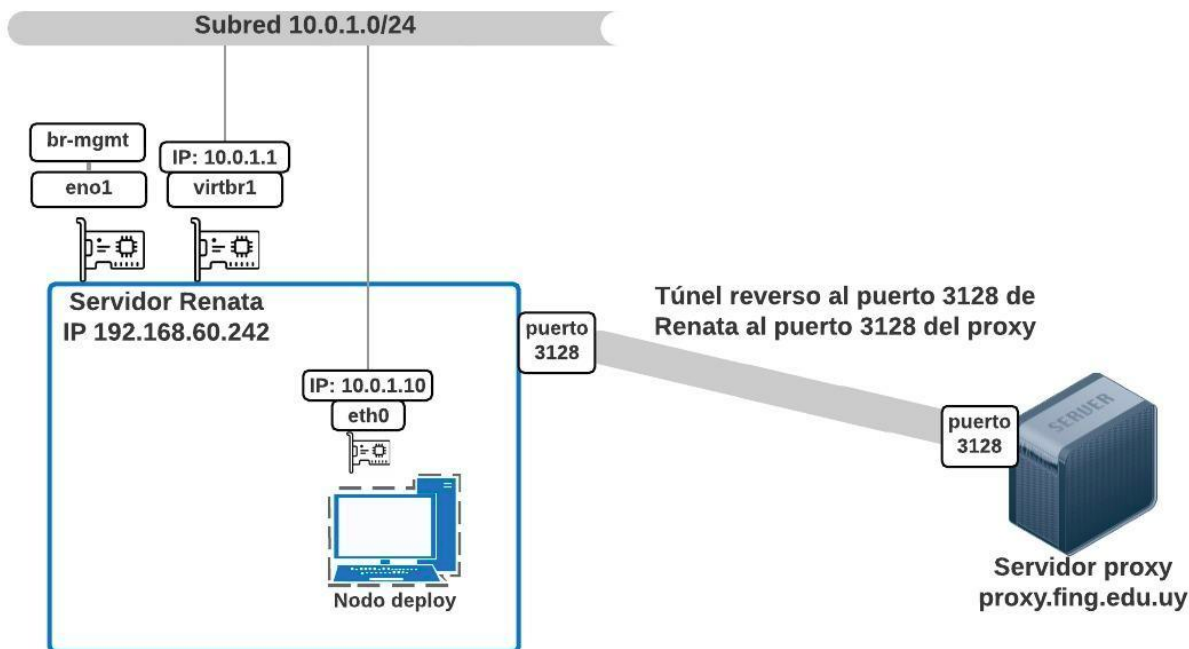


Figura 4.7: Salida a Internet en los nodos de Openstack.

4.3. 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.

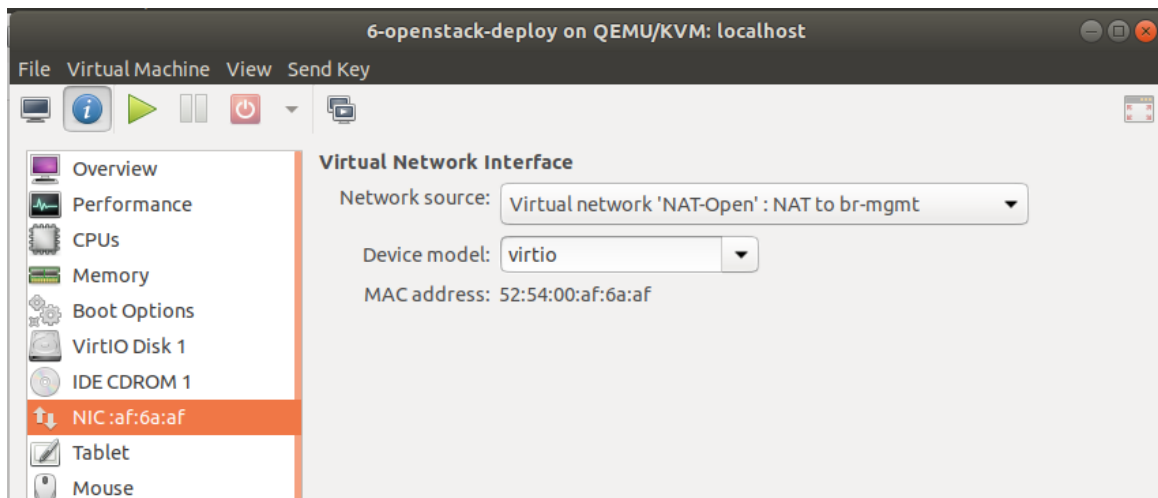


Figura 4.8

3. Por simplificación , cambiar el hostname de la máquina a ‘deploy’ 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	DEVICE=br-vxlan
BOOTPROTO=none	BOOTPROTO=none	BOOTPROTO=none
IPADDR=10.0.1.11	IPADDR=10.0.2.11	IPADDR=10.0.10.11
PREFIX=24	PREFIX=24	PREFIX=24
GATEWAY=10.0.1.1	ONBOOT=yes	ONBOOT=yes
DNS1=192.168.60.230	TYPE=Bridge	TYPE=Bridge
ONBOOT=yes	NM_CONTROLLED=no	NM_CONTROLLED=no
TYPE=Bridge		
NM_CONTROLLED=no		

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

TYPE="Ethernet"	TYPE="Ethernet"	TYPE="Ethernet"	TYPE="Ethernet"
BOOTPROTO=none	BOOTPROTO=none	BOOTPROTO=none	BOOTPROTO=none
DEVICE=eth0	DEVICE=eth1	DEVICE=eth2	DEVICE=eth3
ONBOOT=yes	ONBOOT=yes	ONBOOT=yes	ONBOOT=yes
NM_CONTROLLED=no	NM_CONTROLLED=no	NM_CONTROLLED=no	NM_CONTROLLED=no
BRIDGE=br-mgmt	BRIDGE=br-storage	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 'infra1' 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	DEVICE=br-vxlan
BOOTPROTO=none	BOOTPROTO=none	BOOTPROTO=none
IPADDR=10.0.1.12	IPADDR=10.0.2.12	IPADDR=10.0.10.12
PREFIX=24	PREFIX=24	PREFIX=24
GATEWAY=10.0.1.1	ONBOOT=yes	ONBOOT=yes
DNS1=192.168.60.230	TYPE=Bridge	TYPE=Bridge
ONBOOT=yes	NM_CONTROLLED=no	NM_CONTROLLED=no
TYPE=Bridge		
NM_CONTROLLED=no		

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

TYPE="Ethernet"	TYPE="Ethernet"	TYPE="Ethernet"	TYPE="Ethernet"
BOOTPROTO=none	BOOTPROTO=none	BOOTPROTO=none	BOOTPROTO=none
DEVICE=eth0	DEVICE=eth1	DEVICE=eth2	DEVICE=eth3
ONBOOT=yes	ONBOOT=yes	ONBOOT=yes	ONBOOT=yes
NM_CONTROLLED=no	NM_CONTROLLED=no	NM_CONTROLLED=no	NM_CONTROLLED=no
BRIDGE=br-mgmt	BRIDGE=br-storage	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' 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
BOOTPROTO=none
IPADDR=10.0.1.13
PREFIX=24
GATEWAY=10.0.1.1
DNS1=192.168.60.230
ONBOOT=yes
TYPE=Bridge
NM_CONTROLLED=no

```

```

DEVICE=br-storage
BOOTPROTO=none
IPADDR=10.0.2.13
PREFIX=24
ONBOOT=yes
TYPE=Bridge
NM_CONTROLLED=no

```

1.2. Configurar las interfaces eth0 y eth1.

```

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

```

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' 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:

```
$ lvmfdiskscan
```

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
```

HAproxy1

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' 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.

4.4. 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.

4.4.1. 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.

4.4.2. Archivos de configuración OSA

openstack_user_config.yml

A continuación se detalla cada bloque de configuración utilizado en la instalación realizada.

En la primera sección, `cidr_networks`, se describen las subredes utilizadas en la instalación de Openstack.

```
cidr_networks:
  container: 10.0.1.0/24
  tunnel: 10.0.10.0/24
  storage: 10.0.2.0/24
```

La red de container permite a los distintos servicios , ejecutados dentro de contenedores o servidores físicos, comunicarse entre sí. La red tunnel es utilizada cuando un usuario crea una nueva red tennant dentro de un proyecto de Openstack, en el caso que corresponda se creará una red VXLAN en este red física. Por último la red de storage la utilizaran los nodos que alojan el backend y los servicios de storage de Openstack. Los rangos elegidos son totalmente arbitrarios, a excepción de la subred 10.0.3.0/24 debido a que es utilizada internamente por OSA durante su ejecución como se mencionó en la sección de arquitectura de red.

Lo siguiente a configurar son las direcciones IP que están siendo utilizadas por los hosts físicos de la infraestructura en las redes definidas previamente, con el fin de reservarlas y que la instalación no utilice ninguna de ellas para la estructura de Openstack.

```
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
```

En la sección `global_overrides` se describen los bridges utilizados y detalles de las interfaces del ambiente, por ejemplo, cómo los containers se conectan a la red física de los hosts.

```
global_overrides:
  internal_lb_vip_address: 10.0.1.15
  external_lb_vip_address: 192.168.60.160
```

Estos parámetros refieren a las direcciones IPs pública y privada del balanceador de carga. La privada es utilizada internamente por los servicios de Openstack y la pública es la puerta de acceso para los usuarios.

```
tunnel_bridge: "br-vxlan"
management_bridge: "br-mgmt"
storage_bridge: "br-storage"
```

Con estas variables se define el nombre asignado a cada uno de los bridges creados en los hosts físicos.

La sección de `provider_networks` describe la relación entre la red de los contenedores y el ambiente físico de los servidores.

```
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"

```

La última sección describe en qué servidor o grupo de servidores corre cada servicio de Openstack y de infraestructura.

```

### 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 [49] 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.yml

En caso de utilizar un backend de storage LVM se debe indicar que este debe ser deployado 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

```

4.4.3. 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

```

4.4.4. Correcciones

SELinux

Durante el proceso de instalación se detectó un bug asociado a SELinux, el cual fue verificado en [16]. 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 [29] 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"
```

4.5. Ejecución de playbooks

Finalmente 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:

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 verbosa y el final es para mostrar la salida del comando en la consola y almacenarla en un archivo de log.

install-haproxy.yml

La configuración de los balanceadores de carga es lo siguiente a realizar. Esto se puede realizar sin tener explícitamente los contenedores o servidores físicos con los servicios instalados gracias a Ansible. OSA como se mencionó en la sección del inventario, a partir de los archivos de configuración conoce exactamente cómo quedará instalado cada uno de los servicios de Openstack, por ejemplo las IPs y puertos de cada servicio. Con el siguiente comando se ejecuta la playbook:

```
$ openstack-ansible -vvv haproxy-install.yml 2>&1 | tee /var/log/openstack/haproxyXX.log
```

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. Esta script además se encarga de instalar los servicios de infraestructura como son RabbitMQ o Galera DB para luego ser configurados en la playbook final. Con el siguiente comando se ejecuta la playbook:

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

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
```

4.6. Verificación

Luego de que la última playbook haya terminado su ejecución sin error, debemos verificar que la instalación fue exitosa. Esto se realizará 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`.

5

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 5.1: Vista del login de Horizon.

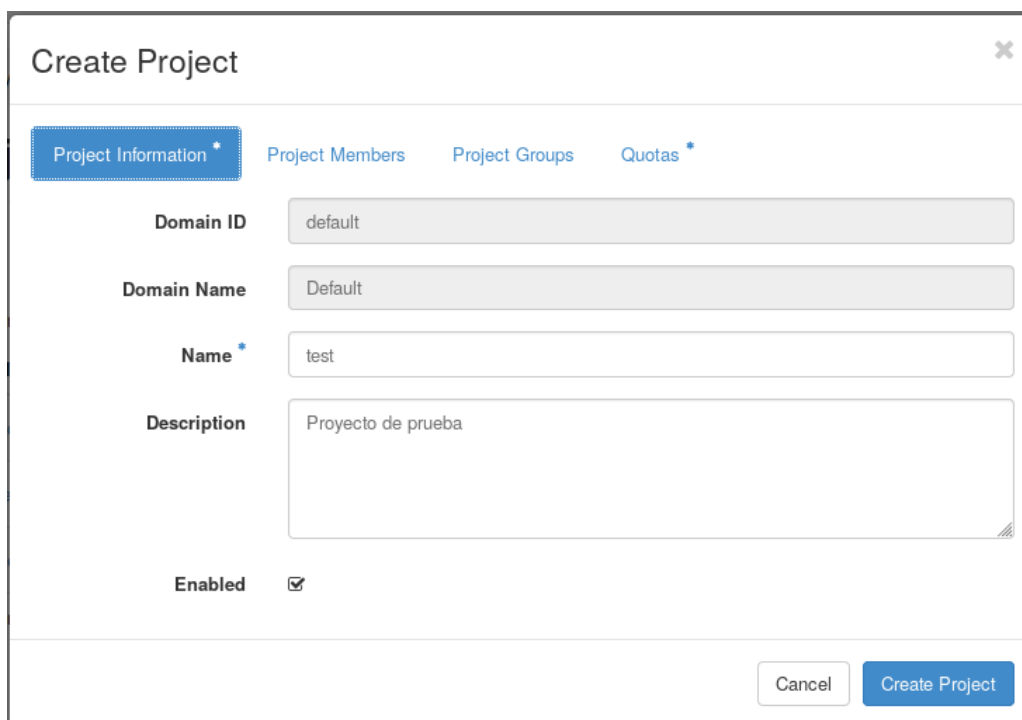
Una vez allí, se debe acceder con la cuenta de administrador para realizar las configuraciones iniciales utilizando las credenciales indicadas en el último paso de la sección de verificación.

5.1. 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 5.2.



The screenshot shows a 'Create Project' modal window. It features a tabbed interface with 'Project Information' as the active tab. The form includes input fields for 'Domain ID' (default), 'Domain Name' (Default), 'Name' (test), and 'Description' (Proyecto de prueba). An 'Enabled' checkbox is checked. The bottom right corner contains 'Cancel' and 'Create Project' buttons.

Figura 5.2: 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

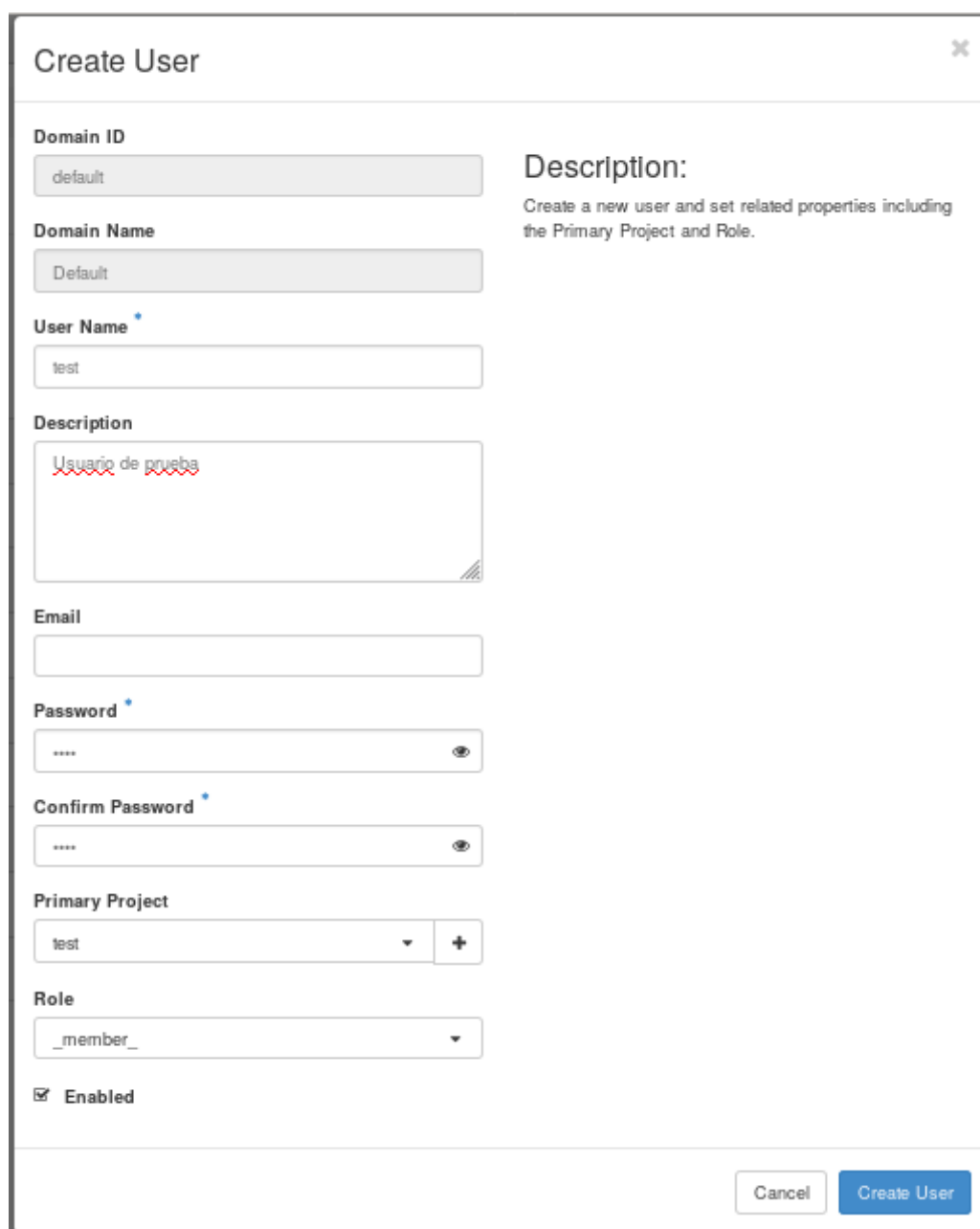
admin	member ▼	-
-------	----------	---

Cancel Create Project

Figura 5.3: 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 5.4.



Create User

Domain ID
default

Domain Name
Default

User Name *
test

Description
Usuario de prueba

Email

Password *

Confirm Password *

Primary Project
test

Role
member

☒ **Enabled**

Description:
Create a new user and set related properties including the Primary Project and Role.

Cancel Create User

Figura 5.4: Creación de un usuario.

Por más detalles se puede visitar [39].

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 5.5 y 5.6. Más detalles en [38].

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 5.5: 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

Q

admin

+

service

+

Selected Projects

Filter

Q

test

-

Cancel

Create Flavor

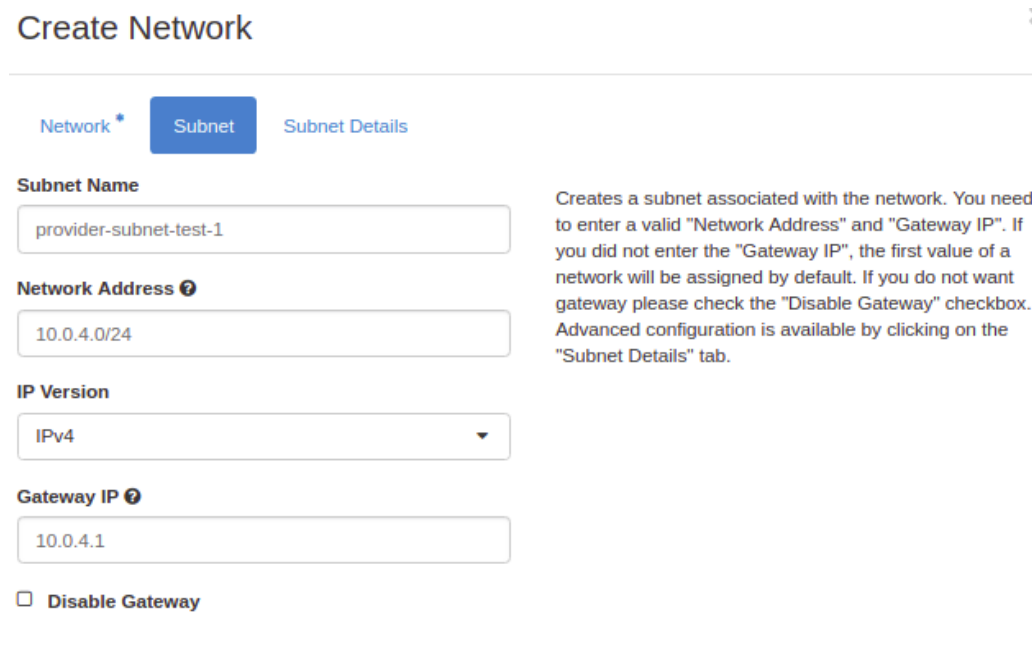
Figura 5.6: Creación de un flavor (2/2).

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 5.7 y 5.8.

Figura 5.7: 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
provider-subnet-test-1

Network Address ?
10.0.4.0/24

IP Version
IPv4

Gateway IP ?
10.0.4.1

☐ 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 5.8: Creación de una red provider (2/2).

5.2. 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 [30] 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.

Create Image

Image Details

Metadata

Image Details

Specify an image to upload to the Image Service.

Image Name

image-test-1

Image Description

Imagen de prueba

Image Source

Source Type

File

File

Browse...

cirros-0.4.0-x86_64-disk.img

Format

QCOW2 - QEMU Emulator

Image Requirements

Kernel

Choose an image

Ramdisk

Choose an image

Architecture

x86_64

Minimum Disk (GB)

10

Minimum RAM (MB)

2048

Image Sharing

Protected

Yes No

Cancel


< Back

Next >


Create Image


Figura 5.9: 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 metatada más específica de la misma.


You can specify resource metadata by moving items from the left column to the right column. In the left column there are metadata definitions from the Glance Metadata Catalog. Use the "Custom" option to add metadata with the key of your choice. 


Available Metadata


Filter 


Custom 


> CIM Processor Allocation Setting 


> Cinder Volume Type 


> CPU Pinning 

> Database Software 


> Guest Memory Backing 

> Hypervisor Selection 

> Image Signature Verification 

> Instance Config Data 

Existing Metadata

Filter 

No existing metadata

Click each item to get its description here.

< Back

Next >


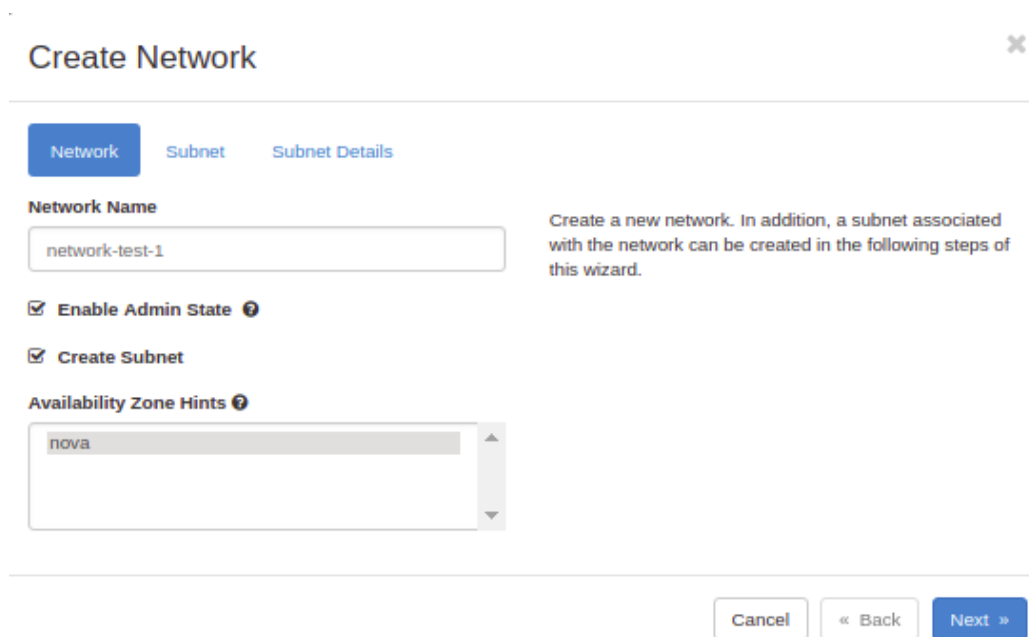
 Create Image

Figura 5.10: 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:

63



Create Network ✕

Network Subnet Subnet Details

Network Name
network-test-1

☒ **Enable Admin State** ⓘ

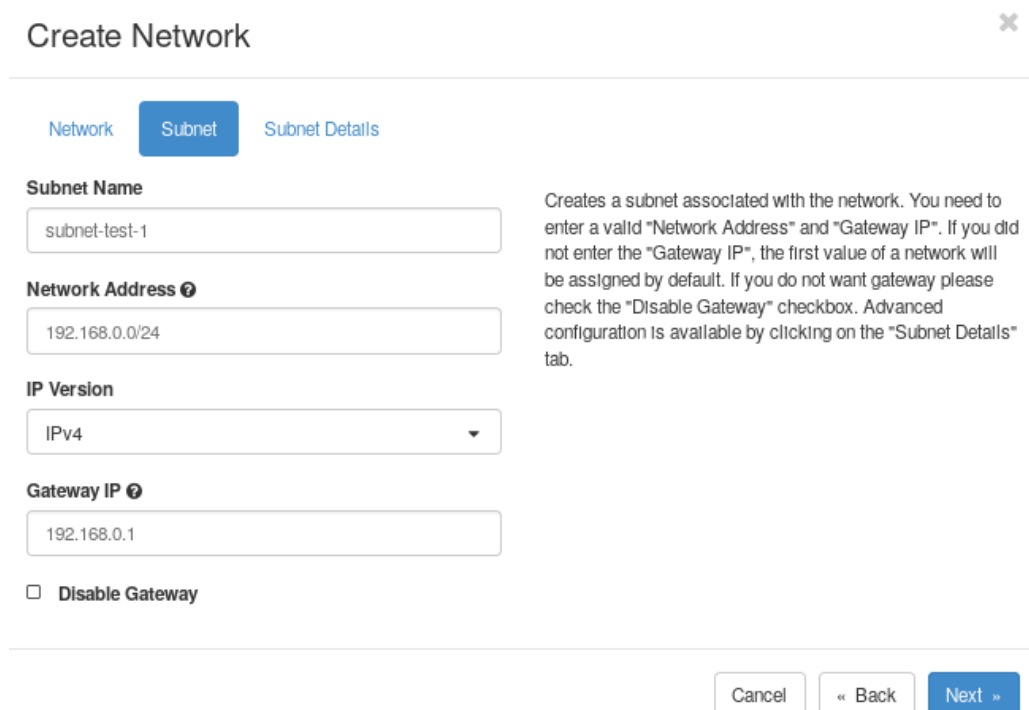
☒ **Create Subnet**

Availability Zone Hints ⓘ
nova

Create a new network. In addition, a subnet associated with the network can be created in the following steps of this wizard.

Cancel « Back Next »

Figura 5.11: Creación de una red (1/3).



Create Network ✕

Network **Subnet** Subnet Details

Subnet Name
subnet-test-1

Network Address ⓘ
192.168.0.0/24

IP Version
IPv4 ▼

Gateway IP ⓘ
192.168.0.1

☐ **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 5.12: Creación de una red (2/3).

Create Network ✕

Network Subnet **Subnet Details**

☒ **Enable DHCP** Specify additional attributes for the subnet.

Allocation Pools ?

DNS Name Servers ?

192.168.60.230

Host Routes ?

Cancel « Back Create

Figura 5.13: Creación de una red (3/3).

Crear router

La creación de nuevos routers se realiza accediendo a Project >Network >Routers >Create Router.

Create Router

Router Name

router-test-1

Description:

Creates a router with specified parameters.

☒ **Enable Admin State**

Availability Zone Hints ?

nova

Figura 5.14: Creación de un router.

Más detalles de estas configuraciones se pueden encontrar en [27].

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.

Add Interface

Subnet *

network-test-1: 192.168.0.0/24 (subnet-test-1) ▼

IP Address (optional) ⓘ

Description:

You can connect a specified subnet to the router.

If you don't specify an IP address here, the gateway's IP address of the selected subnet will be used as the IP address of the newly created interface of the router. If the gateway's IP address is in use, you must use a different address which belongs to the selected subnet.

✕ Cancel + Create Key Pair

Figura 5.15: 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.

Create Key Pair ✕

Key Pair Name *

Key Pairs are how you login to your instance after it is launched. Choose a key pair name you will recognize. Names may only include alphanumeric characters, spaces, or dashes.

✕ Cancel + Create Key Pair

Figura 5.16: 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 [24].

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 5.17.

Launch Instance

Please provide the initial hostname for the instance, the availability zone where it will be deployed, and the instance count. Increase the Count to create multiple instances with the same settings.

Instance Name *
Instance-test-1

Description
Instancia de prueba 1

Availability Zone
nova

Count *
1

Total Instances (10 Max)
10%

0 Current Usage
1 Added
9 Remaining

Cancel < Back Next > Launch Instance

Figura 5.17: 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

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

Create New Volume
Yes No

Volume Size (GB) *
10

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

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 5.18: Lanzar una nueva instancia (2/5).

A continuación se determinan los recursos virtuales de la instancia mediante la selección de un flavor.

Launch Instance

Details

Source

Flavor

Networks *

Network Ports

Security Groups

Key Pair

Configuration

Server Groups

Scheduler Hints

Metadata

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
------	-------	-----	------------	-----------	----------------	--------

< Back Next > Launch Instance

Figura 5.19: Lanzar una nueva instancia (3/5).

El siguiente paso requerido es la selección de una red.

Launch Instance

Details

Source

Flavor

Networks

Network Ports

Security Groups

Key Pair

Configuration

Server Groups

Scheduler Hints

Metadata

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

< Back Next > Launch Instance

Figura 5.20: 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.

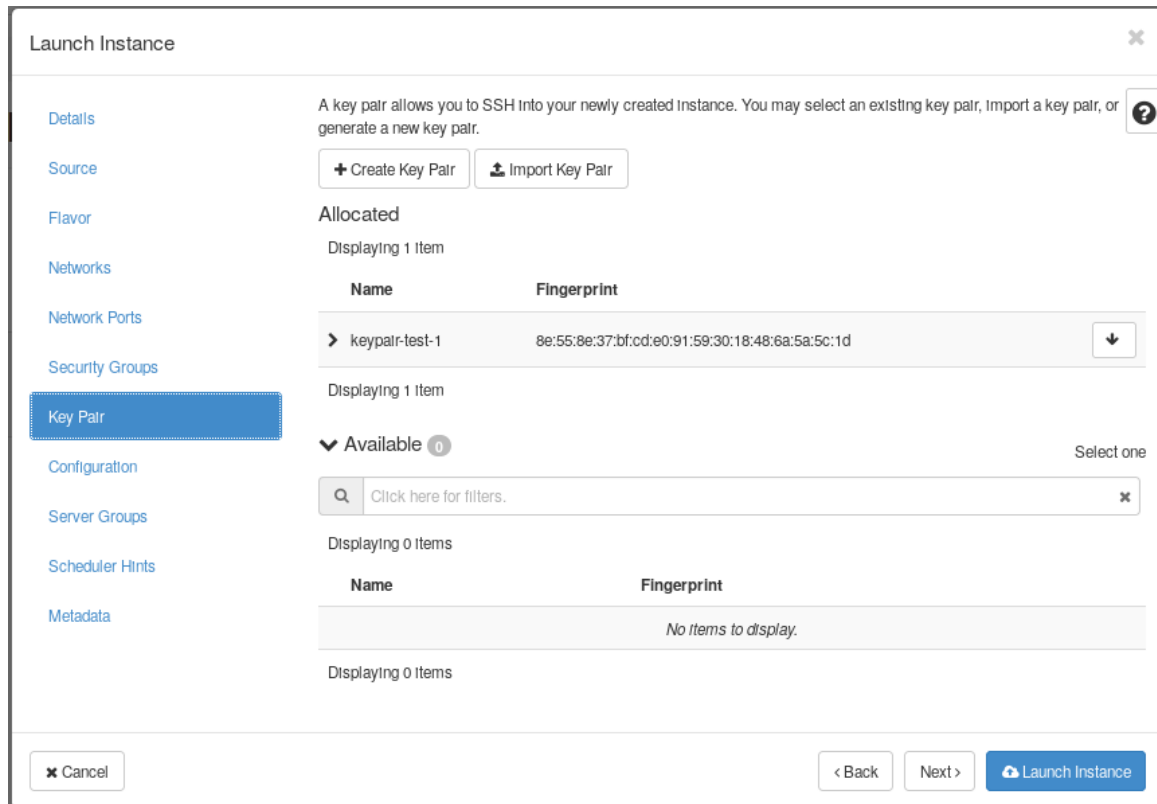


Figura 5.21: Lanzar una nueva instancia (5/5).

Más detalles sobre el lanzamiento de una instancia se pueden ver accediendo a [36].

5.3. Acceso a una instancia

5.3.1. 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.

5.3.2. Por SSH

Para lograr acceder en forma externa por SSH son necesarias algunas configuraciones extras.

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:

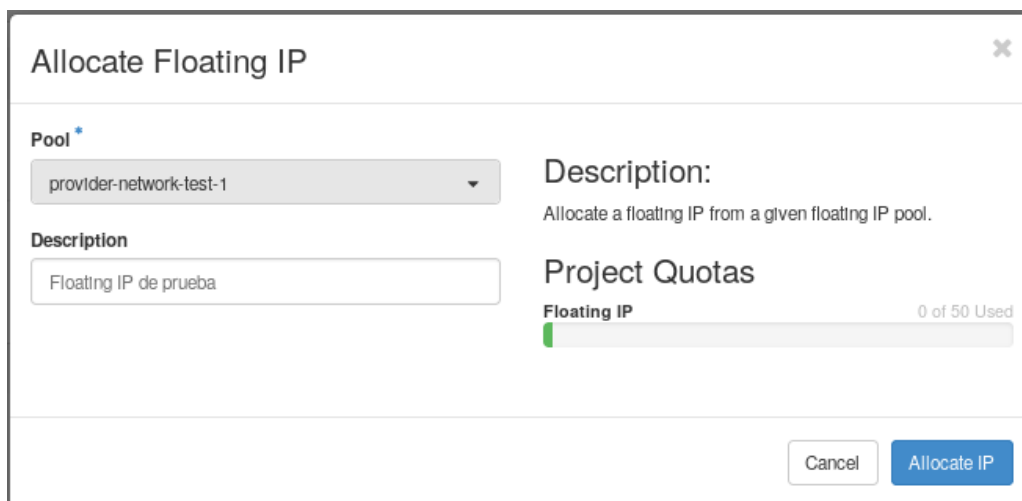


Figura 5.22: Asignación de floating IP.

Luego se debe asociar con el puerto de la instancia creada desde Project >Network >Floating IPs >Associate.

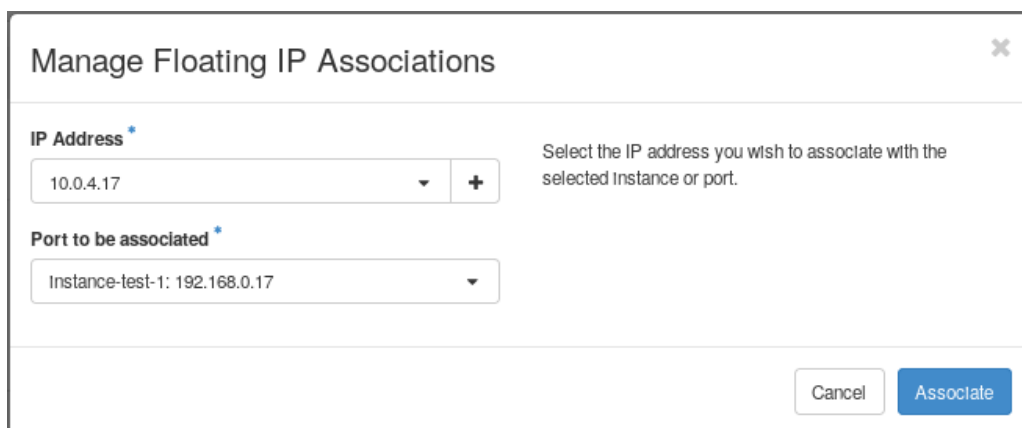


Figura 5.23: Asociación de floating IP.

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 5.24: 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 5.25: 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 5.26: Agregar regla para tráfico SSH.

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"
```

6

Inconvenientes

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 de correcciones, 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 en playbook setup-infrastructure

Durante la instalación de la playbook mencionada 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.el7.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.el7.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. Para esto se deben seguir 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 rango. En función de esta observación es que se especificó el rango 10.0.10.0/24 para la red vxlan.

7

Análisis del módulo de red

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 OpenVSwitch, 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.

7.1. 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

Cuadro 7.1: Sabores

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

Cuadro 7.2: Imágenes

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

Cuadro 7.3: Redes provider

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

Cuadro 7.4: Subredes provider

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.

7.1.1. 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.

Composición del escenario

- Red tenant 1 de tipo VXLAN (7.5).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (7.6).
- Instancia 2 alojada en el nodo de cómputo 2 asociada a la red tenant 1 (7.6).

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

Cuadro 7.5: 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

Cuadro 7.6: Detalles de las instancias

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

Composición del escenario

- Red tenant 1 de tipo VXLAN (7.7).
- Red tenant 2 de tipo VXLAN (7.8).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (7.9).
- Instancia 2 alojada en el nodo de cómputo 2 asociada a la red tenant 2 (7.9).
- Router X asociado a las redes tenant 1 y 2 (7.10).

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

Cuadro 7.7: Detalles de la subred 1

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

Cuadro 7.8: 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-1	192.168.2.102

Cuadro 7.9: Detalles de las instancias

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

Cuadro 7.10: Detalles del router

7.1.3. 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 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.

Composición del escenario

- Red tenant 1 de tipo VXLAN (7.11).
- Red provider vlan de tipo VLAN (7.12).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (7.13).
- Router Y asociado a las redes tenant 1 y provider vlan (7.14).

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

Cuadro 7.11: 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

Cuadro 7.12: Detalles de la subred provider vlan

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

Cuadro 7.13: Detalles de las instancias

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

Cuadro 7.14: Detalles del router

7.1.4. 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.

Composición del escenario

- Red tenant 1 de tipo VXLAN (7.15).
- Red provider vlan de tipo VLAN (7.16).
- Instancia 1 alojada en el nodo de cómputo 1 asociada a la red tenant 1 (7.17).
- Router Z asociado a las redes tenant 1 y provider vlan (7.18).

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

Cuadro 7.15: 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

Cuadro 7.16: 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

Cuadro 7.17: Detalles de las instancias

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

Cuadro 7.18: Detalles del router

7.2. Linux bridge

Introducción del linux bridge.

7.2.1. Escenario 1

Tráfico Este - Oeste (misma red tenant)

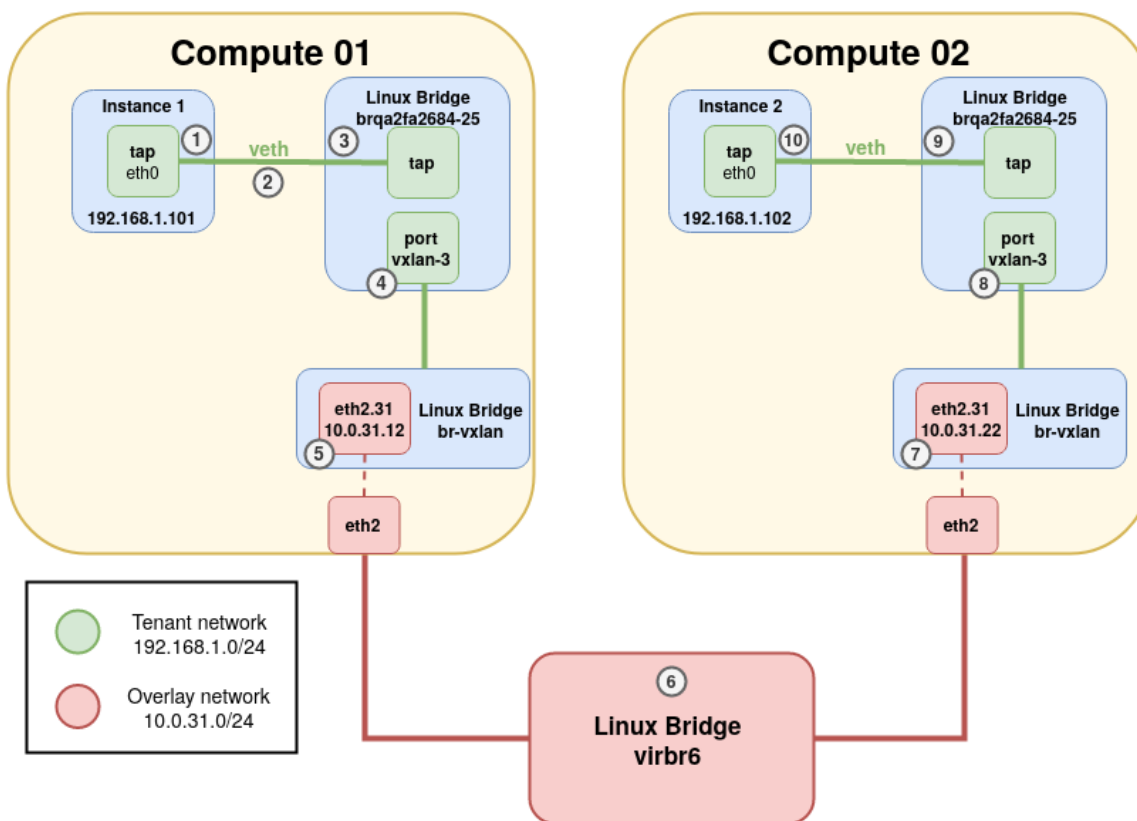


Figura 7.1: Diagrama de arquitectura para el escenario 1 de Linux Bridge

Análisis de componentes

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

Nodos de cómputo

- El hypervisor 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 siguientes comandos:

- 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 el 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-9
ebb-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"

```

- o 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

```

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

<p>▶ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)</p> <p>▼ Ethernet II, Src: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc), Dst: Broadcast (ff:ff:ff:ff:ff:ff)</p> <p>▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)</p> <p>▶ Source: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)</p> <p>Type: ARP (0x0806)</p> <p>▼ Address Resolution Protocol (request)</p> <p>Hardware type: Ethernet (1)</p> <p>Protocol type: IPv4 (0x0800)</p> <p>Hardware size: 6</p> <p>Protocol size: 4</p> <p>Opcode: request (1)</p> <p>Sender MAC address: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)</p> <p>Sender IP address: 192.168.1.101</p> <p>Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)</p> <p>Target IP address: 192.168.1.102</p>
--

Figura 7.2: Paquete ARP request capturado en la interfaz eth0 de la instancia 1

- 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	0.011416	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

<p>▶ Frame 1: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)</p> <p>▶ Ethernet II, Src: RealtekU_be:9c:39 (52:54:00:be:9c:39), Dst: IPv4mcast_01:01:01 (01:00:5e:01:01:01)</p> <p>▶ Internet Protocol Version 4, Src: 10.0.31.12, Dst: 239.1.1.1</p> <p>▶ User Datagram Protocol, Src Port: 44739, Dst Port: 8472</p> <p>▼ Virtual eXtensible Local Area Network</p> <p>▶ Flags: 0x0800, VXLAN Network ID (VNI)</p> <p>Group Policy ID: 0</p> <p>VXLAN Network Identifier (VNI): 3</p> <p>Reserved: 0</p> <p>▼ Ethernet II, Src: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc), Dst: Broadcast (ff:ff:ff:ff:ff:ff)</p> <p>▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)</p> <p>▶ Source: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)</p> <p>Type: ARP (0x0806)</p> <p>▼ Address Resolution Protocol (request)</p> <p>Hardware type: Ethernet (1)</p> <p>Protocol type: IPv4 (0x0800)</p> <p>Hardware size: 6</p> <p>Protocol size: 4</p> <p>Opcode: request (1)</p> <p>Sender MAC address: fa:16:3e:d2:5b:cc (fa:16:3e:d2:5b:cc)</p> <p>Sender IP address: 192.168.1.101</p> <p>Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)</p> <p>Target IP address: 192.168.1.102</p>

Figura 7.3: 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

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

Figura 7.4: 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
```

11. 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 7.5: 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.

1. 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 7.6: 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 7.7: Paquete ICMP request encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1

4. El paquete es recibido por linux bridge br-vxlan del nodo de cómputo 2 a través de la sub interfaz 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.
5. 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.

7.2.2. Escenario 2

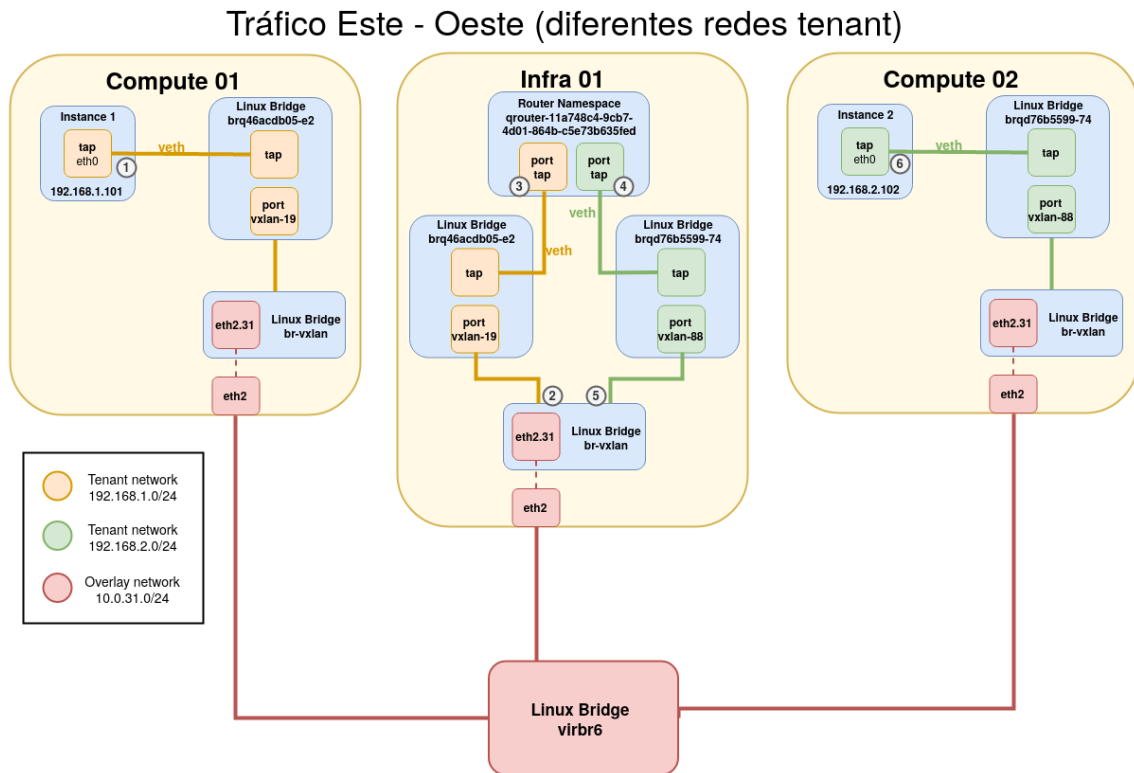


Figura 7.8: Diagrama de arquitectura para el escenario 2 de Linux Bridge

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

- 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 11a748c4-9cb7-4
d01-864b-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-91e1-531
b42952adc\"}]",
  "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 d_port los primeros diez caracteres del identificador del puerto que se pueden listar con:

```
[root@infra1-utility-container-eebf40f8 ~]# openstack port list --device-id <
device_id>, siendo el device_id el identificador del router.
```

- 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
```

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 7.9: 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 0.001350	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 7.10: 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.

7.2.3. Escenario 3

Tráfico Norte - Sur (acceso desde el exterior)

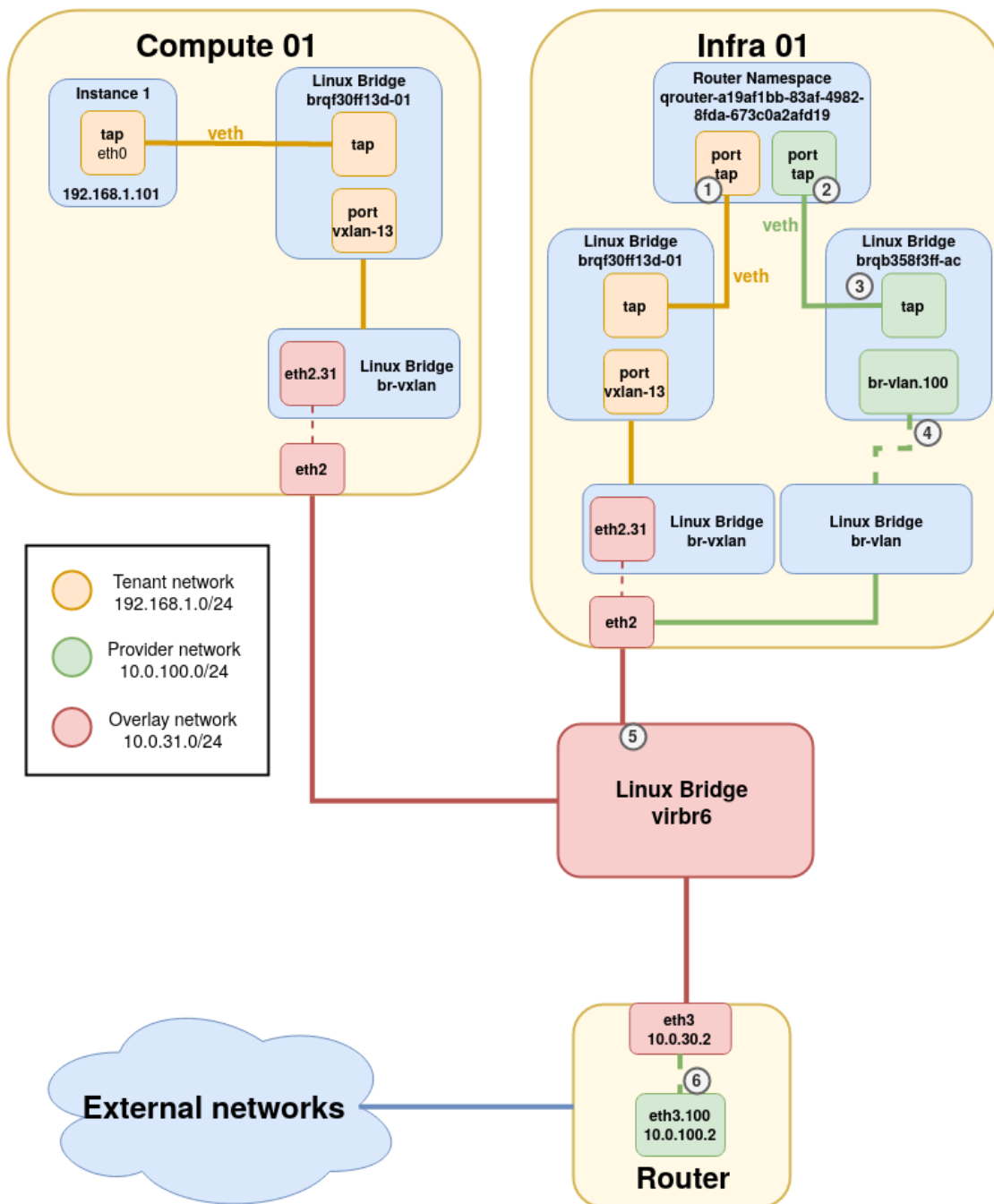


Figura 7.11: Diagrama de arquitectura para el escenario 3 de Linux Bridge

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 redes de Neutron en este escenario son:

- La red tenant:

```
[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:

```
[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.

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 7.12: 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 7.13: 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.

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) ...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 7.14: Paquete ICMP echo request capturado en la interfaz br-vlan del nodo de red

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.

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

7.2.4. Escenario 4

Tráfico Norte - Sur (acceso desde el exterior)

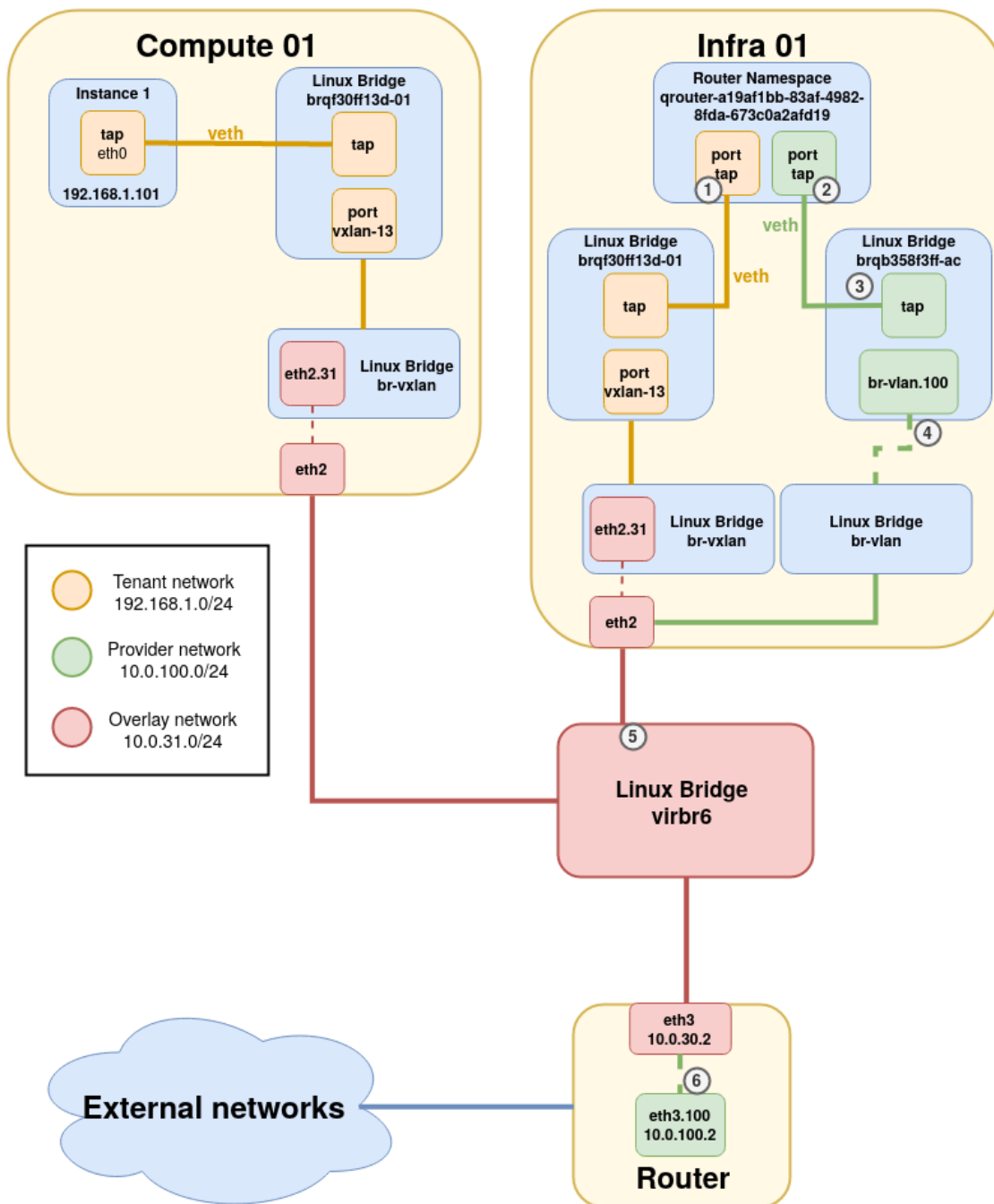


Figura 7.15: Diagrama de arquitectura para el escenario 4 de Linux Bridge

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-673c0a2afd19 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).

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).

2	0.792744	10.0.100.2	10.0.100.64	ICMP	102 Echo (ping) request	id=0x3b8f, seq=1/256,
3	0.794303	10.0.100.64	10.0.100.2	ICMP	102 Echo (ping) reply	id=0x3b8f, seq=1/256,

Frame 2: 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:73:85:77 (fa:16:3e:73:85:77)

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: 10.0.100.64

Internet Control Message Protocol

Figura 7.16: 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 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 7.17: 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 7.18: 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.

7.3. 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.

7.3.1. Escenario 1

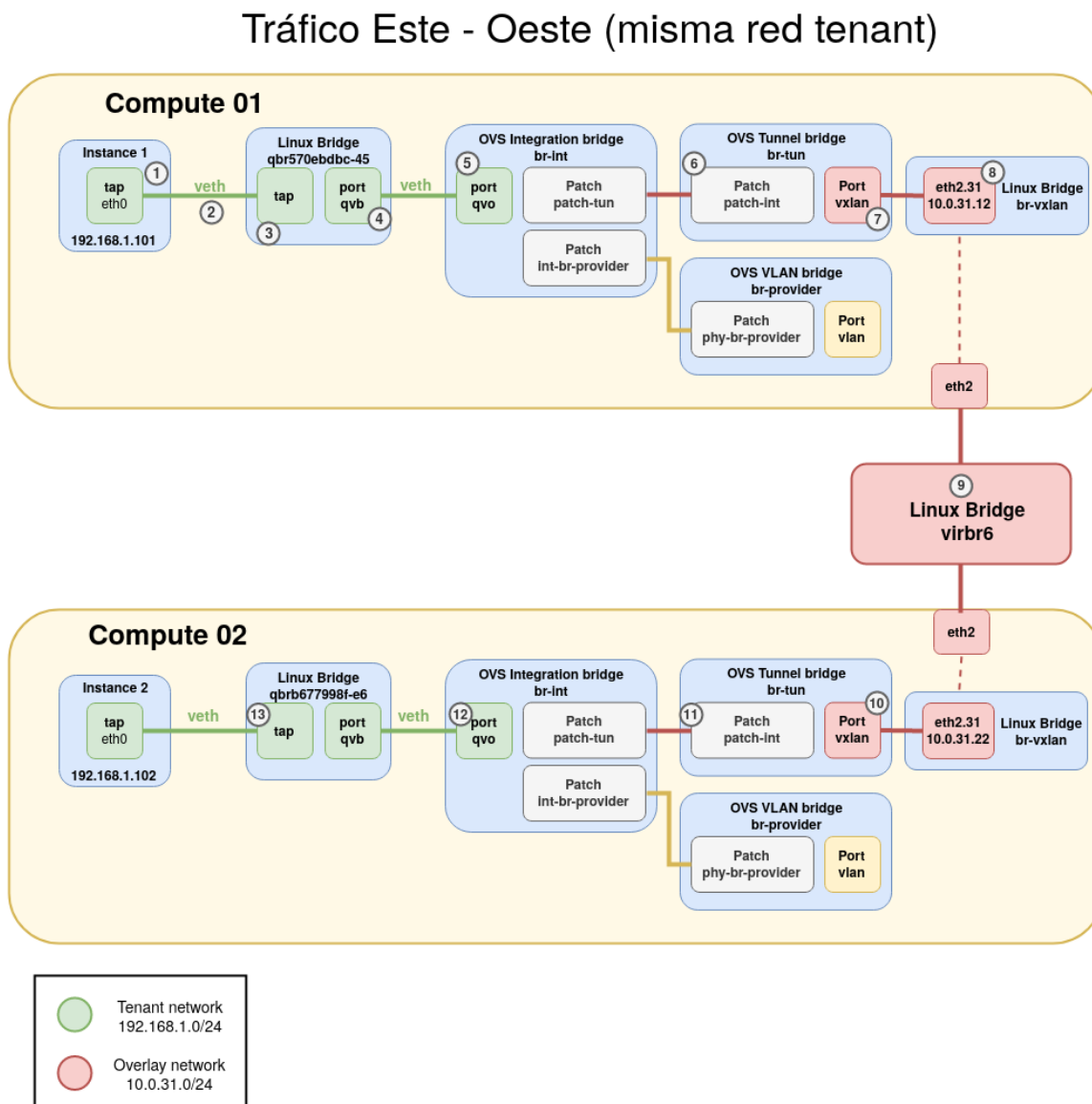


Figura 7.19: Diagrama de arquitectura para el escenario 1 de Open vSwitch

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 hypervisor 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 4974712b-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 hypervisor 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
27d2-60fe-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`.

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 7.20: 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 actuar 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=0
   xb8003f26cd0d7430,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,d1_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,d1_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)
9. table=2,priority=0,d1_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 (`d1_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.

7. 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 7.21: ARP request encapsulado en VXLAN capturado en la interfaz `br-vxlan` del nodo de cómputo 1

8. 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,d1_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,d1_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)
9. table=2,priority=0,d1_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",d1_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 precondición 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=0
  x3f05eaa5b010cd3f,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-0
  a001f0c"
7. table=22,priority=1,d1_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,d1_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)
10. table=2,priority=0,d1_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.

18. 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 7.22: ARP reply encapsulado en VXLAN capturado en la interfaz br-vxlan del nodo de cómputo 1

19. 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"`
20. 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 7.23: Paquete ARP reply capturado en la interfaz eth0 de la instancia 1

21. 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 0.009034	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)
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: 84
  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 7.24: Paquete ICMP request capturado en la interfaz eth0 de la instancia 1

- 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.
- Este último recibe el pedido por el puerto qvo (5), el cual etiqueta el paquete con el VLAN ID 6.
- 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`.
- 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=0
  xb8003f26cd0d7430,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,d1_dst=fa:16:3e:92:57:
  e4 actions=load:0->NXM_OF_VLAN_TCI[],load:0x13->NXM_NX_TUN_ID[],output:"vxlan-0
  a001f16"
7. table=22,priority=1,d1_vlan=6 actions=strip_vlan,load:0x13->NXM_NX_TUN_ID[],output
  : "vxlan-0a001f0b",output:"vxlan-0a001f16"
8. table=0,priority=0 actions=drop
9. table=2,priority=0,d1_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)
10. table=2,priority=0,d1_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 “`vxlan-0a001f16`” se alcanza al bridge `br-vxlan` (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 7.25: Paquete ICMP request encapsulado en VXLAN 19 capturado en el bridge br-vxlan en el nodo de cómputo 1

6. 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.

7.3.2. Escenario 2

Tráfico Este - Oeste (diferentes redes tenant)

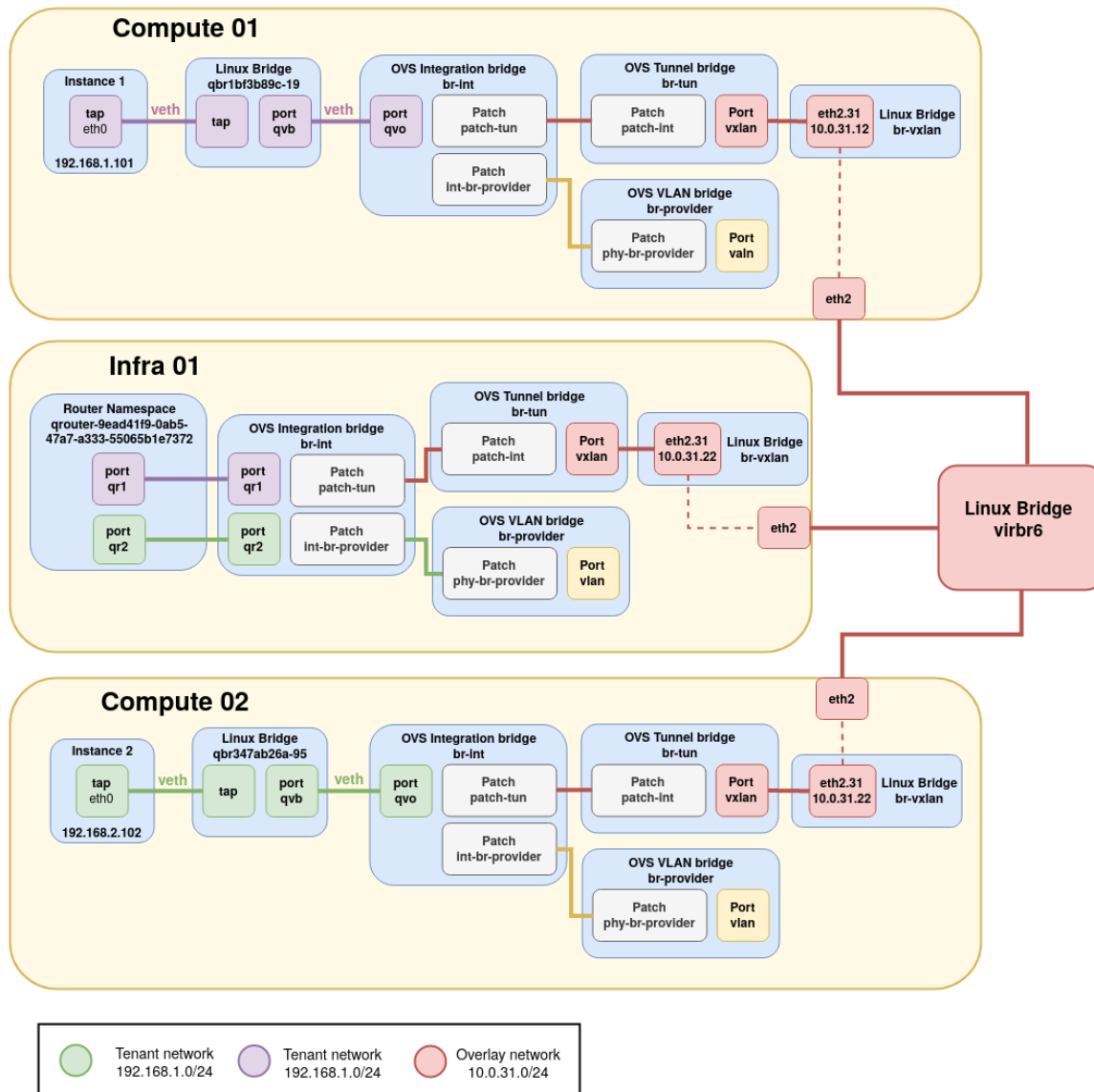


Figura 7.26: Diagrama de arquitectura para el escenario 2 de Open vSwitch

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": "c2f9aeefe-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-a333-55065b1e7372 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 "tcp: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"

```

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 7.27: 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-a333-55065b1e7372 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 7.28: 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.

7.3.3. Escenario 3

Tráfico Norte - Sur (acceso hacia el exterior)

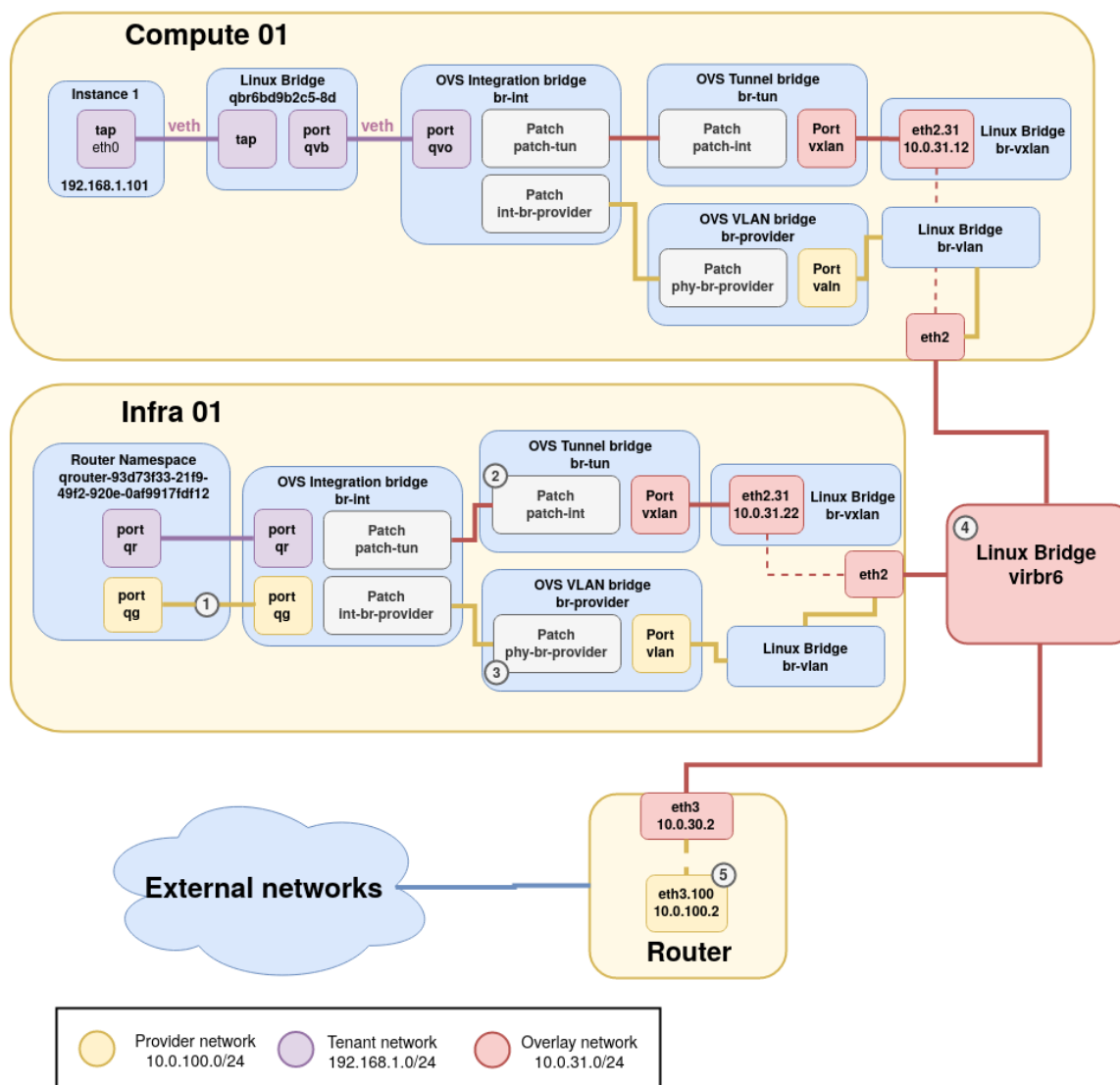


Figura 7.29: Diagrama de arquitectura para el escenario 3 de Open vSwitch

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

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-0af9917fdf12 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.

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,
▶ 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 ▶ 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 7.30: 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-0af9917fdf12 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 llegó el mensaje.

3. Las reglas del bridge br-tun descartarán 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
```

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

- 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.
- 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.

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

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

- 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 ICMPecho-reply es similar al descrito pero en sentido contrario.

7.3.4. Escenario 4

Tráfico Norte - Sur (acceso desde el exterior)

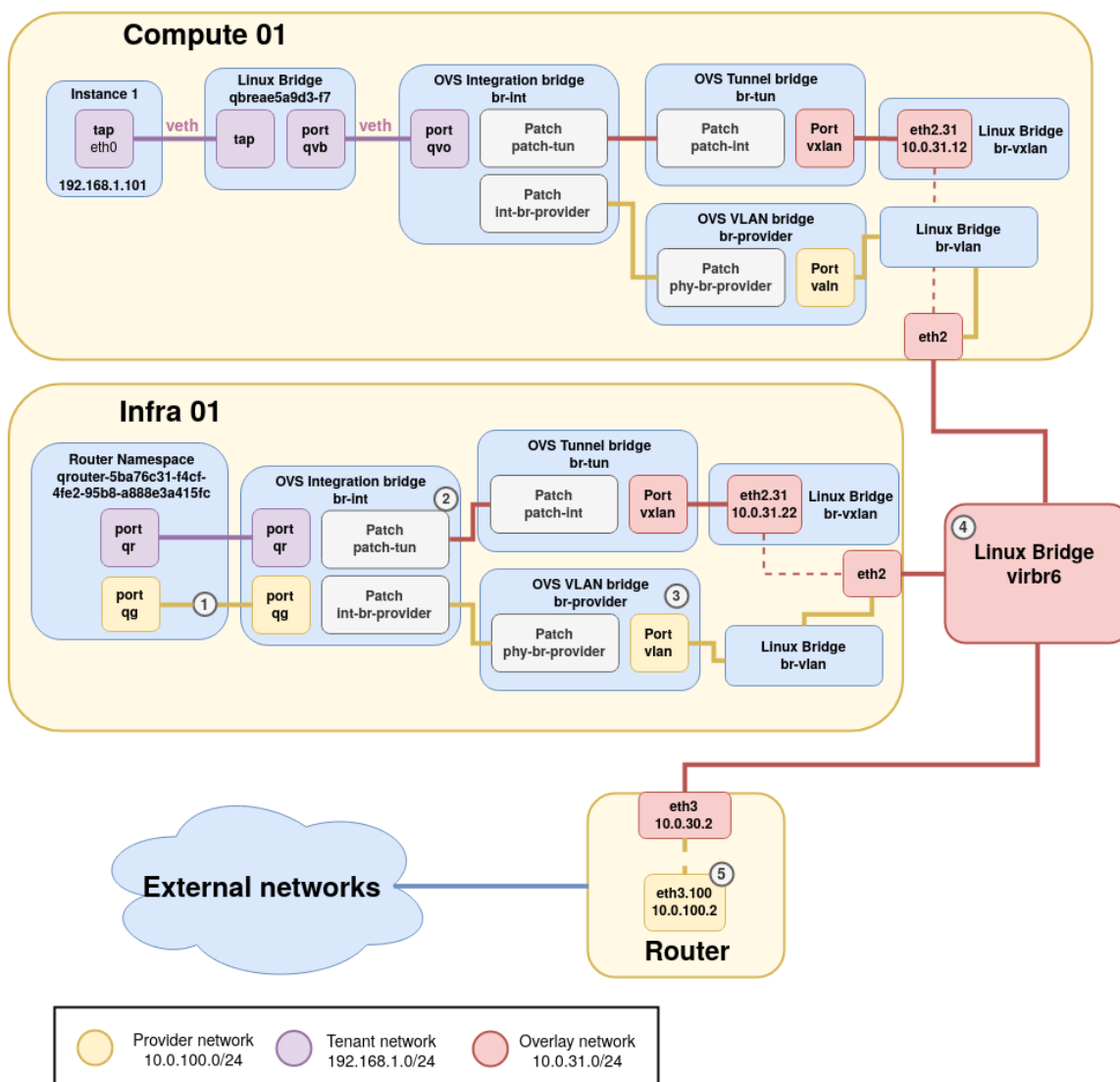


Figura 7.33: Diagrama de arquitectura para el escenario 4 de Open vSwitch

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 qrouter-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).

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
ult 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	0.000903	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 7.34: Paquete ICMP echo request taggeado con el VLAN ID 100 capturado en la interfaz eth3 del router físico

- 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).
- 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 7.35: 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 7.36: 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.

8

Trabajo a futuro

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 inadmisibles. 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 como fue mencionado en la descripción del ambiente de trabajo.

Gestión de Openstack en operación

Es sumamente relevante investigar la gestión en caliente de un datacenter implementado mediante Openstack. Esto involucra tener un manual con pasos claros a realizar en circunstancias tales como escalar horizontalmente el datacenter, reemplazar nodos core y recuperar el funcionamiento ante eventos externos.

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 operativa 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 y contenerización, 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.

El proceso de instalación resulta sumamente complejo debido a la cantidad de herramientas y configuraciones a tener en cuenta, es por eso que la utilización de una herramienta de automatización de tareas resulta inevitable. Como se menciona en el documento, se utilizó Ansible para facilitar el proceso ya que permite ejecutar instalaciones reiteradas veces 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. Esto condujo a que lograr todo el trabajo llevará un tiempo mayor al estimado.

Finalmente se resalta el gran apoyo e inversión a nivel internacional en esta plataforma, y los niveles de crecimiento que tuvo y continúa teniendo en esta década, indicando que es algo por lo que apostar a futuro.

Bibliografía

- [1] 802.1Q-2014 - Bridges and Bridged Networks. <http://www.ieee802.org/1/pages/802.1Q-2014.html>. Accedido: 2019-06-15.
- [2] Containers on Virtual Machines or Bare Metal? White Paper, VMware, 2018.
- [3] Ansible. Module Index. https://docs.ansible.com/ansible/latest/modules/modules_by_category.html. Accedido: 2019-07-05.
- [4] ArchLinux. Linux Containers. https://wiki.archlinux.org/index.php/Linux_Containers. Accedido: 2019-06-20.
- [5] 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.
- [6] James Denton. *Learning Openstack Networking*, chapter 1 Introduction to Openstack Networking, page 15. Packt Publishing, 3rd edition, 2018.
- [7] James Denton. *Learning Openstack Networking*, chapter 1 Introduction to Openstack Networking. Packt Publishing, 3rd edition, 2018.
- [8] James Denton. *Learning Openstack Networking*, chapter 1 Introduction to Openstack Networking, pages 21–23. Packt Publishing, 3rd edition, 2018.
- [9] James Denton. *Learning Openstack Networking*, chapter 1 Introduction to Openstack Networking, page 31. Packt Publishing, 3rd edition, 2018.
- [10] Docker. What is a Container? <https://www.docker.com/resources/what-container>. Accedido: 2019-06-20.
- [11] Kevin Jackson, Cody Bunch, Egle Sigler, and James Denton. *Openstack Cloud Computing Cookbook*, chapter 1 Installing Openstack with Ansible, page 16. Packt Publishing, 4th edition, 2018.
- [12] 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, August 2014.
- [13] Steve Martinelli, Henry Nash, and Brad Topol. *Identity, Authentication, and Access Management in OpenStack: Implementing and Deploying Keystone*, chapter 1 Fundamental Keystone Topics, page 13. O'Reilly Media, Inc., 1st edition, 2015.
- [14] 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.
- [15] Peter M. Mell and Timothy Grance. SP 800-145, The NIST Definition of Cloud Computing. Special Publication, National Institute of Standards & Technology, 2011.

- [16] Openstack. AIO Build Fails on SELinux File Context Tasks. <https://bugs.launchpad.net/openstack-ansible/+bug/1782798>. Accedido: 2019-07-05.
- [17] Openstack. Appendix E: Container networking. <https://docs.openstack.org/project-deploy-guide/openstack-ansible/ocata/app-networking.html#network-diagrams>. Accedido: 2019-07-05.
- [18] Openstack. Basic architecture. <https://docs.openstack.org/glance/queens/contributor/architecture.html>. Accedido: 2019-07-05.
- [19] Openstack. Basic networking. <https://docs.openstack.org/mitaka/networking-guide/intro-basic-networking.html>. Accedido: 2019-06-15.
- [20] Openstack. Common Image Properties. <https://docs.openstack.org/glance/queens/user/common-image-properties.html>. Accedido: 2019-06-20.
- [21] Openstack. Components. <https://docs.openstack.org/swift/latest/admin/objectstorage-components.html>. Accedido: 2019-07-05.
- [22] Openstack. Compute service overview. <https://docs.openstack.org/nova/latest/install/get-started-compute.html>. Accedido: 2019-06-20.
- [23] Openstack. Conductor as a place for orchestrating tasks. <https://docs.openstack.org/nova/rocky/user/conductor.html>. Accedido: 2019-06-20.
- [24] 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.
- [25] Openstack. Container networking. <https://docs.openstack.org/openstack-ansible/rocky/reference/architecture/container-networking.html>. Accedido: 2019-07-05.
- [26] Openstack. Control plane architecture. <https://docs.openstack.org/arch-design/design-control-plane.html>. Accedido: 2019-06-20.
- [27] Openstack. Create and manage networks. <https://docs.openstack.org/horizon/latest/user/create-networks.html>. Accedido: 2019-08-03.
- [28] Openstack. Design. <https://docs.openstack.org/arch-design/design.html>. Accedido: 2019-06-20.
- [29] Openstack. Drop SELinux support for CentOS 7. <https://review.opendev.org/#/c/603860/>. Accedido: 2019-07-05.
- [30] Openstack. Get images. <https://docs.openstack.org/image-guide/obtain-images.html>. Accedido: 2019-08-04.
- [31] Openstack. Images and instances. <https://docs.openstack.org/glance/queens/admin/troubleshooting.html>. Accedido: 2019-06-20.
- [32] Openstack. Introduction. <https://docs.openstack.org/project-team-guide/introduction.html>. Accedido: 2019-06-15.
- [33] Openstack. Introduction to Object Storage. <https://docs.openstack.org/swift/latest/admin/objectstorage-intro.html>. Accedido: 2019-07-05.
- [34] Openstack. Inventory. <https://docs.openstack.org/openstack-ansible/rocky/reference/inventory/inventory.html>. Accedido: 2019-06-20.

- [35] Openstack. Keystone Architecture. <https://docs.openstack.org/keystone/latest/getting-started/architecture.html>. Accedido: 2019-06-20.
- [36] Openstack. Launch and manage instances. <https://docs.openstack.org/horizon/latest/user/launch-instances.html>. Accedido: 2019-08-03.
- [37] Openstack. LVM. <https://docs.openstack.org/cinder/latest/configuration/block-storage/drivers/lvm-volume-driver.html>. Accedido: 2019-06-20.
- [38] Openstack. Manage flavors. <https://docs.openstack.org/horizon/latest/admin/manage-flavors.html>. Accedido: 2019-08-03.
- [39] Openstack. Manage projects and users. <https://docs.openstack.org/horizon/latest/admin/manage-projects-and-users.html>. Accedido: 2019-08-03.
- [40] Openstack. Memcached. <https://docs.openstack.org/mitaka/install-guide-ubuntu/environment-memcached.html>. Accedido: 2019-06-20.
- [41] Openstack. Message queuing. <https://docs.openstack.org/security-guide/messaging.html>. Accedido: 2019-06-20.
- [42] Openstack. Network architectures. <https://docs.openstack.org/openstack-ansible/stein/user/network-arch/example.html>. Accedido: 2019-07-05.
- [43] Openstack. Networking concepts. <https://docs.openstack.org/arch-design/design-networking/design-networking-concepts.html>. Accedido: 2019-06-15.
- [44] Openstack. Nova System Architecture. <https://docs.openstack.org/nova/latest/user/architecture.html>. Accedido: 2019-07-05.
- [45] Openstack. Object Storage API overview. https://docs.openstack.org/swift/latest/api/object_api_v1_overview.html. Accedido: 2019-07-05.
- [46] Openstack. Object Storage characteristics. <https://docs.openstack.org/swift/latest/admin/objectstorage-characteristics.html>. Accedido: 2019-07-05.
- [47] Openstack. OpenStack-Ansible Documentation. <https://docs.openstack.org/openstack-ansible/latest/>. Accedido: 2019-07-05.
- [48] Openstack. openstack_user_config settings reference. <https://docs.openstack.org/openstack-ansible/queens/reference/inventory/openstack-user-config-reference.html>. Accedido: 2019-06-20.
- [49] Openstack. Settings Reference. <https://docs.openstack.org/horizon/queens/configuration/settings.html>. Accedido: 2019-08-04.
- [50] Openstack. Software. <https://www.openstack.org/software/>. Accedido: 2019-06-15.
- [51] Openstack. Swift Architectural Overview. https://docs.openstack.org/swift/latest/overview_architecture.html. Accedido: 2019-07-05.
- [52] Openstack. The OpenStack Marketplace. <https://www.openstack.org/marketplace/distros/>. Accedido: 2019-07-05.
- [53] Openstack. Useful image properties. <https://docs.openstack.org/glance/latest/admin/useful-image-properties.html>. Accedido: 2019-06-20.
- [54] Openstack. Volume drivers. <https://docs.openstack.org/cinder/latest/configuration/block-storage/volume-drivers.html>. Accedido: 2019-06-20.

- [55] Openstack. Welcome to the Puppet OpenStack Guide! <https://docs.openstack.org/puppet-openstack-guide/latest/>. Accedido: 2019-07-05.
- [56] Oracle. What are Hypervisors? https://docs.oracle.com/cd/E50245_01/E50249/html/vmcon-hypervisor.html. Accedido: 2019-06-20.
- [57] Ken Pepple. *Deploying OpenStack*, chapter 4 Understanding Nova. O'Reilly Media, Inc., 2011.
- [58] Red Hat. Conductor. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/4/html/Configuration_Reference_Guide/section-conductor.html. Accedido: 2019-06-20.
- [59] Red Hat. Openstack Block Storage (Cinder). https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/6/html/Component_Overview/section-blockStorage.html. Accedido: 2019-06-20.
- [60] Red Hat. What is KVM? <https://www.redhat.com/en/topics/virtualization/what-is-KVM>. Accedido: 2019-06-20.
- [61] Red Hat. What is virtualization? <https://www.redhat.com/en/topics/virtualization/what-is-virtualization>. Accedido: 2019-06-20.
- [62] Red Hat. What's a Linux container. <https://www.redhat.com/en/topics/containers/whats-a-linux-container>. 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] VMware. Virtualization. <https://www.vmware.com/solutions/virtualization.html>. Accedido: 2019-06-20.
- [66] 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.