En las unidades anteriores se han visto dos elementos fundamentales para configurar la infraestructura como servicio, esto es máquinas virtuales y almacenamiento, pero falta un último elemento: la virtualización de redes. La virtualización de redes consiste en desacoplar las redes virtuales del hardware de red para lograr una mejor integración de los entornos virtuales.

El componente de OpenStack que se encarga de gestionar la virtualización de redes es neutron y es un componente que se incluyó en OpenStack Folsom con el nombre de quantum. OpenStack neutron está sometido a un importante desarrollo y va incluyendo paulatinamente diferentes tecnologías de red como “backends” mediante plugins

En nuestro caso, vamos a utilizar Open vSwitch como la tecnología de capa 2 por debajo de neutron encargada de la virtualización de la red.

Open vSwitch

Open vSwitch (OVS) es una implementación de software libre de un switch virtual distribuido multicapa.

Las máquinas GNU/Linux incluyen ya un software para la utilización de switches L2, software conocido como “linux bridge”, incluido en el kérnel linux y manejado con las herramientas del espacio de usuario “bridge-utils” y en particular con la instrucción *brctl*. OpenvSwitch es un desarrollo nuevo e independiente que se incluyó en la versión 3.3 del kérnel linux y que proporciona mayor funcionalidad para entornos de virtualización de redes, como muy bien se explica en Why OpenvSwitch?.

Características necesarias

En un entorno con varios nodos de computación como tenemos en OpenStack, precisamos de algunas características de la red como las siguientes:

- Máquinas que se ejecutan en el mismo nodo de computación, pero que estén en diferente red local.
- Máquinas que se ejecutan en diferentes nodos de computación, pero que virtualmente estén en la misma red local.
- Máquinas que puedan pasar “en vivo” de ejecutarse en un nodo de computación a otro.

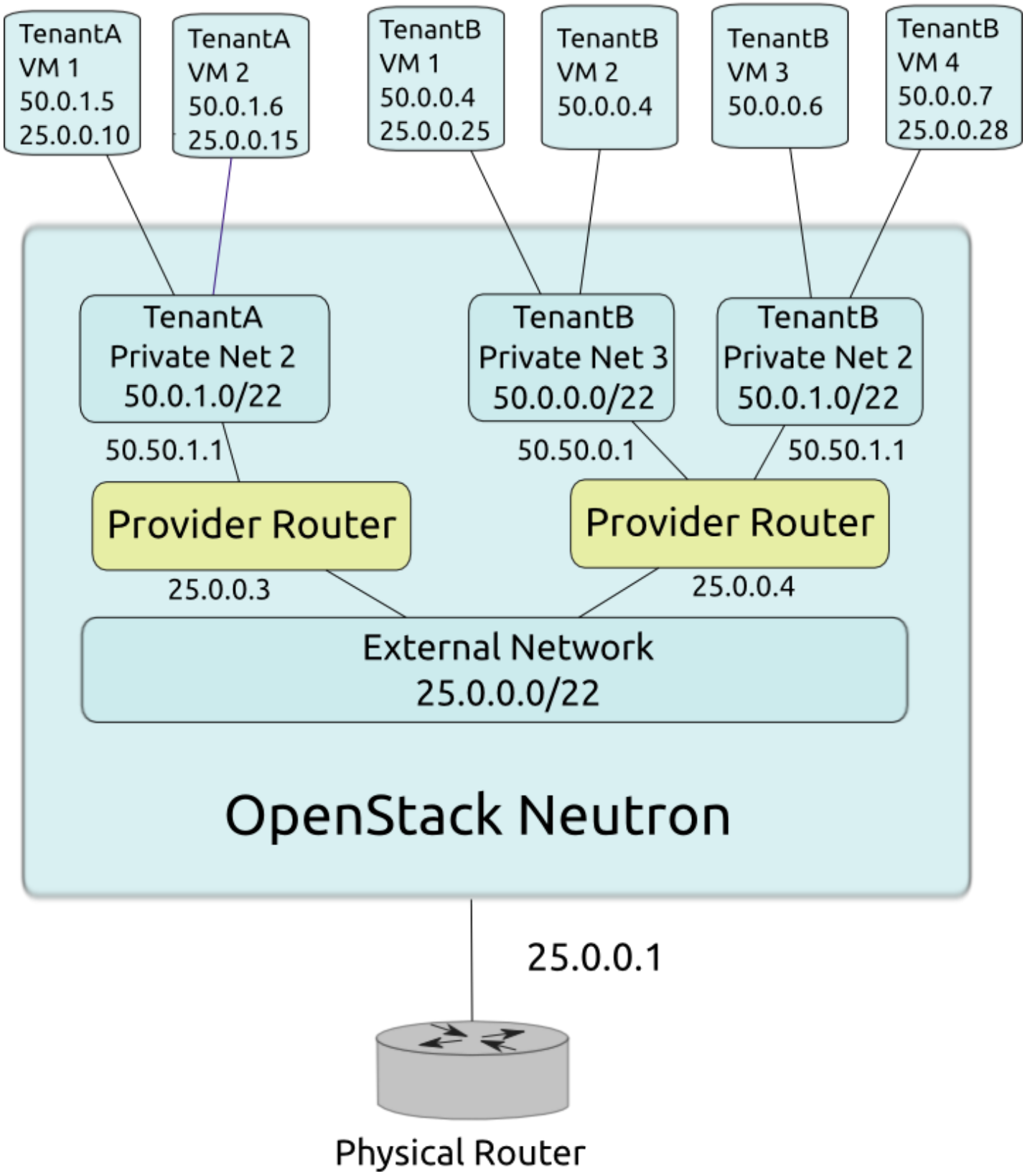
- Cuando sea preciso, un servidor DHCP por cada subred que no interfiera con los de otras subredes
- Reglas de cortafuegos definidas por máquina, no por red.
- Tráfico seguro entre nodos de computación
- Conexión de las máquinas virtuales con el exterior.

Tipo de despliegue

Dependiendo del uso que vaya a tener nuestro cloud, optaremos por alguno de los tipos de despliegue que soporta neutron, en particular debemos saber si:

- Todas las instancias de todos los proyectos estarán dentro de una misma VLAN, lo que OpenStack denomina Single Flat Network.
- Cada proyecto podrá definir las redes privadas que quiera, pero compartirán el o los routers de acceso al exterior, lo que OpenStack denomina Provider Router with Private Networks
- Cada proyecto podrá definir las redes privadas y routers que quiera, de forma independiente al resto de proyectos, lo que OpenStack denomina Per-tenant Routers with Private Networks

En este caso vamos a explicar este último caso, que es el más general e interesante para una nube de infraestructura compartida por diferentes usuarios.



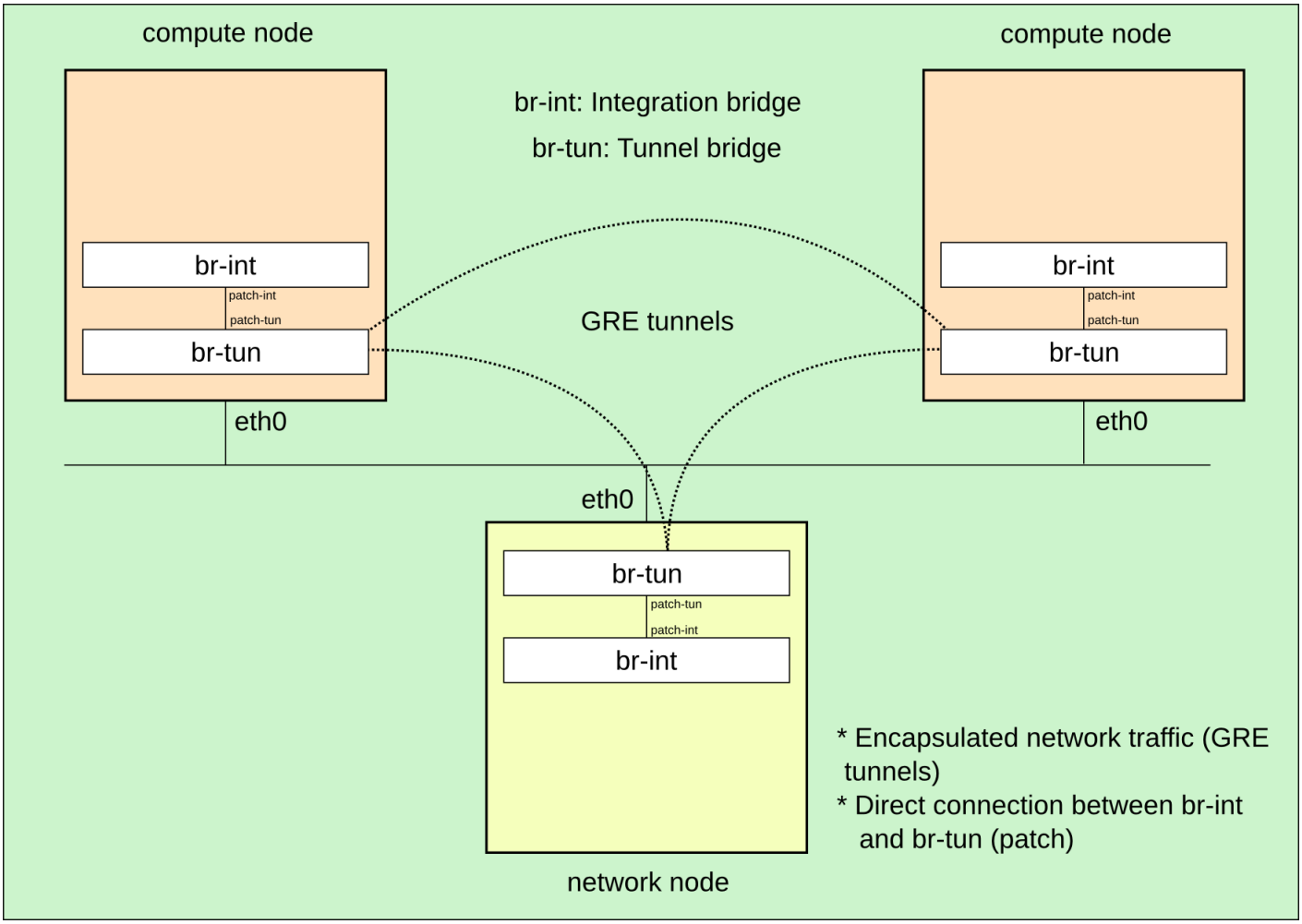
Redes privadas

Puesto que el despliegue completo incluye muchos elementos y puede resultar demasiado complejo, vamos a ver en primer lugar la configuración de la capa 2 con OpenvSwitch y posteriormente los elementos de la capa 3.

Bridge de integración y túneles GRE

El objetivo a la hora de utilizar virtualización de redes es conseguir que las instancias de una misma red se comuniquen entre sí utilizando el mismo direccionamiento y compartiendo algunos elementos comunies (servidor DHCP, puerta de enlace, etc.), pero que permenezcan completamente separadas del resto de redes virtuales. Esto debe producirse independientemente del hipervisor (nodo de computación) en el que se estén ejecutando las máquinas virtuales y una de las formas de conseguir esto con OpenvSwitch es crear un switch virtual en cada nodo, que se denomina switch de integración o br-int y comunicar cada nodo con los demás a través de túneles GRE, que encapsulan el tráfico entre los nodos.

Un esquema que representa esta situación tras una instalación limpia de OpenStack neutron se ve en la siguiente imagen:



Verificación inicial

Puesto que vamos a ir mostrando paso a paso cómo se van creando algunos bridges y puertos de conexión, vamos a mostrar cual es la situación de partida.

Inicialmente no hay ningún bridge linux:

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
```

En el nodo de red hay tres puentes Open vSwitch (br-int, br-tun y br-ex). En este apartado trabajaremos con el bridge de integración (br-int) y el bridge para los túneles (br-tun):

```
# ovs-vsctl list-br
br-ex
br-int
br-tun
```

Esa misma salida en un nodo de computación, muestra que hay dos bridges Open vSwitch:

```
# ovs-vsctl list-br
br-int
br-tun
```

El único dispositivo conectado inicialmente a br-int es br-tun a través de una interfaz especial denominada puerto patch:

```
# ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:000032e2b607a841
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_D...
 1(patch-tun): addr:fa:82:c1:68:15:38
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-int): addr:32:e2:b6:07:a8:41
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

En el caso de br-tun existe un puerto para cada túnel GRE:

```
# ovs-ofctl show br-tun
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000d2c93b314b42
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 1(patch-int): addr:56:63:30:9a:59:a7
      config:     0
      state:      0
      speed: 0 Mbps now, 0 Mbps max
```

```
 2(gre-1): addr:26:7c:0f:93:22:ee
      config:     0
      state:      0
      speed: 0 Mbps now, 0 Mbps max
 3(gre-2): addr:26:7c:0f:a3:12:ae
      config:     0
      state:      0
      speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-tun): addr:d2:c9:3b:31:4b:42
      config:     0
      state:      0
      speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

Creación de una red privada

Utilizando el cliente de línea de comandos de neutron y con las credenciales de un usuario normal, creamos una red privada y una subred asociada a ella con el rango de direcciones IP 10.0.0.0/24:

```
$ neutron net-create privada
Created a new network:
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| admin_state_up | True                                |
| id            | f84a328e-407e-4ca6-87bb-ec148853c585 |
| name          | privada                              |
| shared        | False                                |
| status        | ACTIVE                               |
| subnets       |                                      |
| tenant_id     | 4beb810ce40f49659e0bca732e4f1a3c     |
+---------------+--------------------------------------+

$ neutron subnet-create f84a328e-407e-4ca6-87bb-ec148853c585 10.0.0.0/24
Created a new subnet:
+-----------------+------------------------------------------------------+
| Field           | Value                                                |
+-----------------+------------------------------------------------------+
| allocation_pools | {"start": "10.0.0.2", "end": "10.0.0.254"} |
| cidr            | 10.0.0.0/24                                  |
| dns_nameservers |                                                      |
| enable_dhcp     | True                                                 |
| gateway_ip      | 10.0.0.1                                           |
| host_routes     |                                                      |
| id              | 08a639c6-8493-4a4b-bb59-c14a15087b0c                 |
| ip_version      | 4                                                    |
| name            |                                                      |
| network_id      | 4047ea25-2aad-45b1-9ea7-c2ae2463ed8d                 |
| tenant_id       | 5ca029df72e344499484b2c1767d6472                     |
+-----------------+------------------------------------------------------+
```

Creación automática de la red virtual al levantar una instancia:

La red anteriormente definida no se crea realmente hasta que no se lance una instancia, para la cual se crean todos los dispositivos de red necesarios y se interconectan adecuadamente:

```
$ nova boot --image 4ddb27ab-b3cc-4a65-ac52-f4ce7894e4ed \
--flavor 1 \
--key_name clave-openstack \
--nic net-id=f84a328e-407e-4ca6-87bb-ec148853c585 \
test
```

Comprobamos que se han creado dos puertos nuevos en la nueva subred (mostramos el listado de todos los puertos y filtramos por la subred anteriormente creada):

```
$ neutron port-list|grep 08a639c6-8493-4a4b-bb59-c14a15087b0c
| 27a7722f-... |    | fa:16:3e:34:42:61 | {"subnet_id": "08a639c6-...", "ip_address": "10.0.0.3"}
  |
| 4d56b393-... |    | fa:16:3e:7a:cd:33 | {"subnet_id": "08a639c6-...", "ip_address": "10.0.0.2"}
  |
```

Comprobamos qué puerto se corresponde con el servidor DHCP y cual con la instancia:

```
$ neutron port-show 27a7722f-d8fe-4083-a6ed-8f6ab429e0a9|grep device_owner
| device_owner          | network:dhcp

$ neutron port-show 4d56b393-326c-4231-ad9d-ed4434f920d3|grep device_owner
| device_owner          | compute:None
```

Es decir el puerto con IP 10.0.0.3 se corresponde con el servidor DHCP y el otro con la instancia recién lanzada.

Comprobamos que aparecen un nuevo puerto en el br-int del nodo de red:

```
# ovs-ofctl show br-int
...
 179(tap27a7722f-d8): addr:00:00:00:00:00:00
           config:     PORT_DOWN
       state:      LINK_DOWN
       speed: 0 Mbps now, 0 Mbps max
...

# ovs-vsctl show
    ...
Port "tap27a7722f-d8"
        tag: 85
        Interface "tap27a7722f-d8"
            type: internal
...
```

“tag” es una etiqueta de tipo VLAN que identifica a todos los puertos de la misma red dentro de un nodo de OpenStack, sin embargo no tiene por qué coincidir con el etiquetado que tengan los puertos de la misma red que estén en otro nodo de OpenStack, por ejemplo en el nodo de computación:

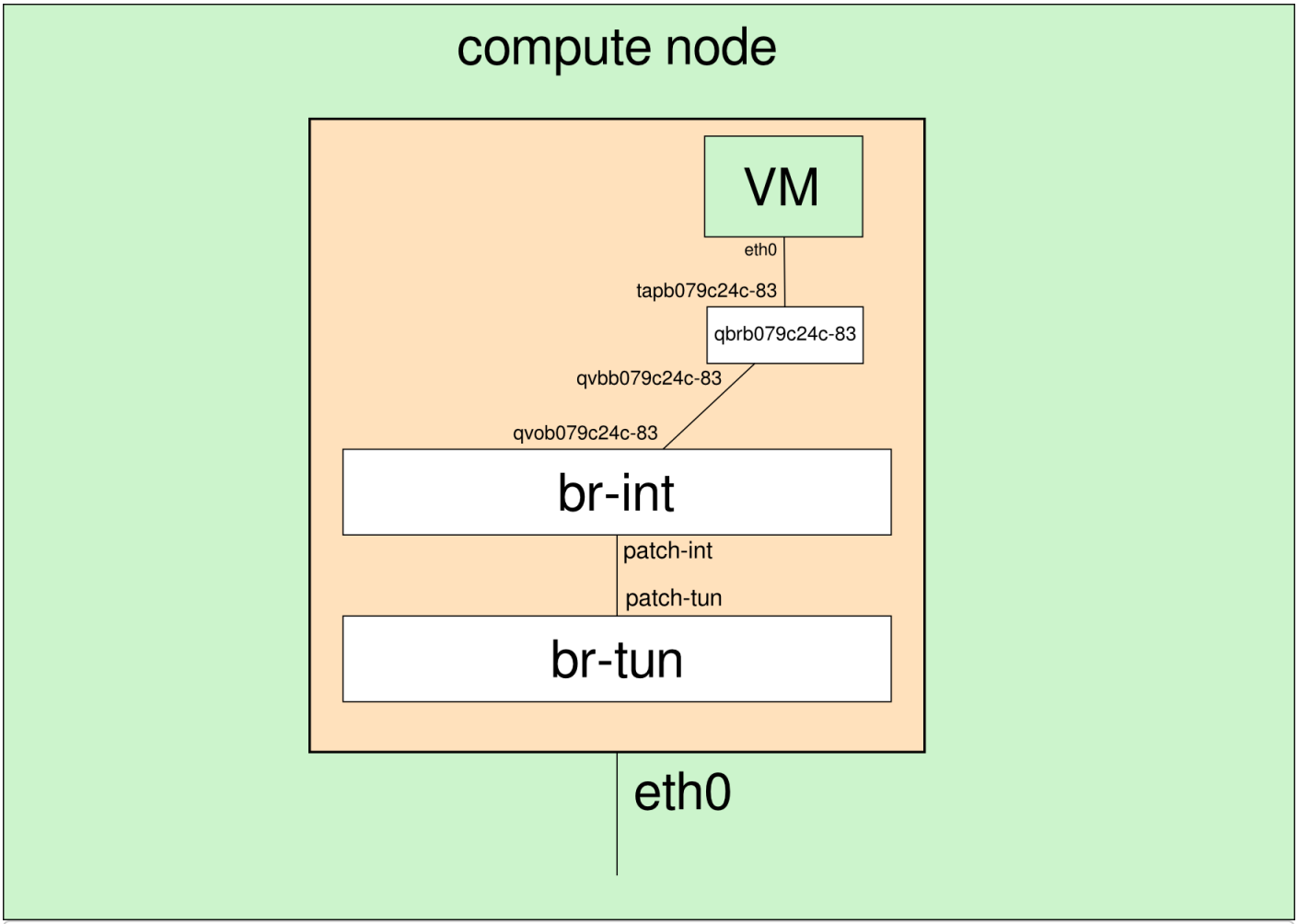
```
# ovs-ofctl show br-int
...
 364(qvo4d56b393-32): addr:26:a2:cb:55:12:93
       config:     0
       state:      0
       current:    10GB-FD COPPER
...

# ovs-vsctl show
...
    Port "qvo4d56b393-32"
        tag: 137
        Interface "qvo4d56b393-32"
...
```

Vemos que la denominación de los puerto en OVS está relacionado con el identificador único del puerto en OpenStack, concretamente con los 11 primeros caracteres. Explicaremos más adelante cómo se ponen en contacto a través de los túneles GRE puertos que inicialmente están definidos en VLANs diferentes.

Podemos comprobar que en el nodo de computación en el que se está ejecutando la instancia aparecen 4 interfaces de red nuevas relacionadas con el identificador 4d56b393-32:

```
# ip link show |grep 4d56b393-32
1582: qbr4d56b393-32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFA
ULT group default
1583: qvo4d56b393-32: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
ovs-system state UP mode DEFAULT group default  qlen 1000
1584: qvb4d56b393-32: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
qbr4d56b393-32 state UP mode DEFAULT group default qlen 1000
1585: tap4d56b393-32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master qbr4d56b
393-32 state UNKNOWN mode DEFAULT group default qlen 500
```

La forma en la que se interconectan todas estas interfaces se muestra en la siguiente imagen y es necesario añadir un bridge linux por cada instancia por la forma de definir las reglas de seguridad por instancia:




```
qbr: Linux bridge
qvo: Puerto en br-int
qvb: Puerto en linux bridge
tap: Interfaz de red de la máquina
```

Esto es algo que choca bastante al principio y hace más difícil entender los elementos implicados, pero parece ser que existe una incompatibilidad actualmente entre la utilización de Open vSwitch, iptables e interfaces tap, tal como se explica en OpenStack Cloud Administration Guide: *Ideally, the TAP device vnet0 would be connected directly to the integration bridge, br-int. Unfortunately, this isn't possible because of how OpenStack security groups are currently implemented. OpenStack uses iptables rules on the TAP devices such as vnet0 to implement security groups, and Open vSwitch is not compatible with iptables rules that are applied directly on TAP devices that are connected to an Open vSwitch port.*

Comprobamos el nuevo bridge linux que se ha creado:

```
# brctl show qbr4d56b393-32
bridge name     bridge id               STP enabled     interfaces
qbr4d56b393-32          8000.56fd37a4d600       no      qvb4d56b393-32
                                                        tap4d56b393-32
```

Servidor DHCP

Se utiliza linux network namespaces, un modo de virtualización de la red

En el nodo de red ejecutamos la instrucción (filtramos por el uuid de la red creada):

```
# ip netns |grep 4047ea25-2aad-45b1-9ea7-c2ae2463ed8d
qdhcp-4047ea25-2aad-45b1-9ea7-c2ae2463ed8d
```

Vemos que se ha creado un espacio de nombres que empieza por dhcp, podemos ejecutar comandos en ese espacio de nombres de forma independiente del sistema:

```
# ip netns exec qdhcp-4047ea25-2aad-45b1-9ea7-c2ae2463ed8d ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
265: tap27a7722f-d8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN mode
DEFAULT group default
    link/ether fa:16:3e:34:42:61 brd ff:ff:ff:ff:ff:ff
```

Podemos ver también el proceso DHCP (con dnsmasq) que se está ejecutando en el nodo de red para darle IP dinámica a las instancias:

```
 # ps aux |grep 4047ea25-2aad-45b1-9ea7-c2ae2463ed8d|grep -v grep
 nobody   32591  0.0  0.0  38852  2732 ?        S    17:29   0:00 dnsmasq \
 --no-hosts --no-resolv --strict-order --bind-interfaces \
 --interface=tap27a7722f-d8 --except-interface=lo \
 --pid-file=/var/lib/neutron/dhcp/4047ea25-2aad-45b1-9ea7-c2ae2463ed8d/pid \
 --dhcp-hostsfile=/var/lib/neutron/dhcp/4047ea25-2aad-45b1-9ea7-c2ae2463ed8d/host \
 --addn-hosts=/var/lib/neutron/dhcp/4047ea25-2aad-45b1-9ea7-c2ae2463ed8d/addn_hosts \
 --dhcp-optsfile=/var/lib/neutron/dhcp/4047ea25-2aad-45b1-9ea7-c2ae2463ed8d/opts \
 --leasefile-ro --dhcp-range=set:tag0,10.0.0.0,static,86400s \
 --dhcp-lease-max=256 --conf-file=/etc/neutron/dnsmasq-neutron.conf \
 --domain=openstacklocal
```

Hay que destacar que la instrucción anterior incluye el parámetro “–no-resolv” por lo que no se utilizará el fichero /etc/resolv.conf de la máquina donde se ejecuta dnsmasq para definir los servidores DNS en los clientes DHCP. Esto quiere decir que habrá que definir de forma explícita estos servidores DNS en cada subred privada.

Podemos acceder al directorio:

```
# cd /var/lib/neutron/dhcp/4047ea25-2aad-45b1-9ea7-c2ae2463ed8d
```

Y comprobar que existe una reserva para la MAC de la instancia:

```
# cat /var/lib/neutron/dhcp/4047ea25-2aad-45b1-9ea7-c2ae2463ed8d/host
fa:16:3e:7a:cd:33,host-192-168-14-2.openstacklocal,10.0.0.2
```

Conexión con otra instancia de la misma red

Para comprobar la conectividad entre instancias dentro de una misma red privada, creamos una nueva instancia (test2) que se ejecutará en el segundo nodo de computación y que virtualmente está en la misma red privada que la instancia “test”, aunque todo el tráfico pase por el bridge de integración de cada nodo de computación, el bridge túnel y el túnel GRE.

Si listamos los puertos del proyecto al crear la nueva instancia, veremos que aparece un puerto nuevo:

```
$ neutron port-list
+--------------------------------------+------+-------------------+-------------------------------
-------------------------------------------------------+
| id                                   | name | mac_address       | fixed_ips
                                                       |
+--------------------------------------+------+-------------------+-------------------------------
-------------------------------------------------------+
| 19ce29da-7bde-4c64-8562-98d0d680d6b1 |      | fa:16:3e:9f:d8:b7 |{"subnet_id": "08a639c6-8493-4
a4b-bb59-c14a15087b0c", "ip_address": "10.0.0.4"}    |
...
```

Luego tiene que haber nuevas interfaces de red en el segundo nodo de computación:

```
  # ip link show |grep 19ce29da-7b
  1384: qbr19ce29da-7b: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DE
FAULT group default
  1385: qvo19ce29da-7b: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast maste
r ovs-system state UP mode DEFAULT group default qlen 1000
  1386: qvb19ce29da-7b: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast maste
r qbr19ce29da-7b state UP mode DEFAULT group default qlen 1000
  1387: tap19ce29da-7b: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master qbr19c
e29da-7b state UNKNOWN mode DEFAULT group default qlen 500
```

Siguiendo el esquema de trabajo de la primera imagen, esta instancia se ha creado en el segundo nodo de computación, por lo que comprobamos que qvo6a6ac39c-61 está conectado a br-int:

```
# ovs-ofctl show br-int
...
 325(qvo19ce29da-7b): addr:02:a0:f5:63:c9:54
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
...
```

Y vemos el nuevo bridge linux:

```
# brctl show qbr6a6ac39c-61
bridge name     bridge id               STP enabled     interfaces
qbr6a6ac39c-61          8000.96a4ec71dea6       no              qvb6a6ac39c-61
                                                                tap6a6ac39c-61
```

Reglas del grupo de seguridad

Comprobamos las reglas de iptables que se han creado en el nodo de computación al levantar la instancia test (b079c24c-8386-47db-a730-35b3a99419e8), en este caso se crean reglas utilizando sólo los 10 primeros caracteres del uuid:

```
# iptables -S |grep b079c24c-8
-N neutron-openvswi-ib079c24c-8
-N neutron-openvswi-ob079c24c-8
-N neutron-openvswi-sb079c24c-8
-A neutron-openvswi-FORWARD -m physdev --physdev-out tapb079c24c-83 --physdev-is-bridged -j neutr
on-openvswi-sg-chain
-A neutron-openvswi-FORWARD -m physdev --physdev-in tapb079c24c-83 --physdev-is-bridged -j neutro
n-openvswi-sg-chain
-A neutron-openvswi-INPUT -m physdev --physdev-in tapb079c24c-83 --physdev-is-bridged -j neutron-
openvswi-ob079c24c-8
-A neutron-openvswi-ib079c24c-8 -m state --state INVALID -j DROP
-A neutron-openvswi-ib079c24c-8 -m state --state RELATED,ESTABLISHED -j RETURN
-A neutron-openvswi-ib079c24c-8 -s 10.0.0.3/32 -p udp -m udp --sport 67 --dport 68 -j RETURN
-A neutron-openvswi-ib079c24c-8 -j neutron-openvswi-sg-fallback
-A neutron-openvswi-ob079c24c-8 -p udp -m udp --sport 68 --dport 67 -j RETURN
-A neutron-openvswi-ob079c24c-8 -j neutron-openvswi-sb079c24c-8
-A neutron-openvswi-ob079c24c-8 -p udp -m udp --sport 67 --dport 68 -j DROP
-A neutron-openvswi-ob079c24c-8 -m state --state INVALID -j DROP
-A neutron-openvswi-ob079c24c-8 -m state --state RELATED,ESTABLISHED -j RETURN
-A neutron-openvswi-ob079c24c-8 -j neutron-openvswi-sg-fallback
-A neutron-openvswi-sb079c24c-8 -s 10.0.0.2/32 -m mac --mac-source FA:16:3E:79:3B:AC -j RETURN
-A neutron-openvswi-sb079c24c-8 -j DROP
-A neutron-openvswi-sg-chain -m physdev --physdev-out tapb079c24c-83 --physdev-is-bridged -j neut
ron-openvswi-ib079c24c-8
-A neutron-openvswi-sg-chain -m physdev --physdev-in tapb079c24c-83 --physdev-is-bridged -j neutr
on-openvswi-ob079c24c-8
```

Entre las que destacamos las siguientes:

- Se crean tres nuevas cadenas, para los procesos de entrada (i), salida (o) y prevenir spoofing (s):

  -N neutron-openvswi-ib079c24c-8 -N neutron-openvswi-ob079c24c-8 -N neutron-openvswi-sb079c24c-8

- Se permiten conexiones entrantes relacionadas o establecidas:

  -A neutron-openvswi-ib079c24c-8 -m state –state RELATED,ESTABLISHED -j RETURN

- Se permiten las respuestas DHCP (DHCPOFFER y DHCPACK) desde el servidor DHCP de la red:

-A neutron-openvswi-ib079c24c-8 -s 10.0.0.3/32 -p udp -m udp –sport 67 –dport 68 -j RETURN

- Se permiten peticiones DHCP (DHCPDISCOVER y DHCPREQUEST):

  -A neutron-openvswi-ob079c24c-8 -p udp -m udp –sport 68 –dport 67 -j RETURN

- Se permiten conexiones salientes relacionadas o establecidas:

  -A neutron-openvswi-ob079c24c-8 -m state –state RELATED,ESTABLISHED -j RETURN

- Se verifica la correspondencia entre la dirección IP y la MAC:

  -A neutron-openvswi-sb079c24c-8 -s 10.0.0.2/32 -m mac –mac-source FA:16:3E:79:3B:AC -j RETURN

Conectividad entre las instancias de la misma red

Como las reglas de cortafuegos se aplican a las instancias de forma individual y no hay ninguna regla que permita acceso desde equipos del mismo segmento de red, no es posible inicialmente acceder desde una instancia a otra de la misma red, es necesario permitir explícitamente cada proceso que se desee.

Dependiendo de la versión de OpenStack varían las reglas de seguridad que se definen por defecto. En el caso de OpenStack Icehouse se definen las siguientes reglas de salida (egress):

```
$ neutron security-group-rule-list
+--------------------------------------+----------------+-----------+----------+-----------------
-+--------------+
| id                                   | security_group | direction | protocol | remote_ip_prefix
 | remote_group |
+--------------------------------------+----------------+-----------+----------+-----------------
-+--------------+
| a008aa84-5543-4a24-8398-751b3aa3d1a8 | default        | egress    |          |
 |              |
| ffa55db3-00ae-4e8f-8e03-92f76d998ba7 | default        | egress    |          |
 |              |
+--------------------------------------+----------------+-----------+----------+-----------------
-+--------------+
```

Añadimos una nueva regla para que puedan comunicarse por ICMP las máquinas de la red 10.0.0.0/24:

```
$ neutron security-group-rule-create --direction ingress --protocol icmp --remote-ip-prefix 10.0.
0.0/24 default
```

Que produce la siguiente regla de iptables:

```
-A neutron-openvswi-ib079c24c-8 -s 10.0.0.0/24 -p icmp -j RETURN
```

Y podemos comprobar que ya es posible hacer ping entre instancias de la misma red privada.

Para permitir de forma explícita la comunicación a algún puerto UDP o TCP se utilizaría una regla de seguridad como la siguiente para el servicio ssh (22/tcp):

```
$ neutron security-group-rule-create --direction ingress --protocol tcp --port-range-min 22 --por
t-range-max 22 --remote-ip-prefix 10.0.0.0/24 default
```

Reglas permisivas en la red privada

Si queremos que las instancias tengan acceso a todos los puertos del resto de instancias de su misma red privada, podemos definir reglas de seguridad permisivas para esta situación, abriendo todo el rango de puertos UDP y TCP:

```
$ neutron security-group-rule-create --direction ingress --protocol tcp --port-range-min 1 --port
-range-max 65535 --remote-ip-prefix 10.0.0.0/24 default
$ neutron security-group-rule-create --direction ingress --protocol udp --port-range-min 1 --port
-range-max 65535 --remote-ip-prefix 10.0.0.0/24 default
```

Interconexión de VLAN a través de un túnel GRE con OVS

Tenemos inicialmente que los puertos de una determinada red se etiquetan en cada nodo en una VLAN diferente, sin embargo es evidentemente que tiene que haber un mecanismo que “traduzca” el tráfico de un nodo a otro al pasar a través del túnel. Si hacemos un volcado de los flujos definidos en br-tun en cada nodo y filtramos por la etiqueta vlan utilizada (dl_vlan), veremos que todo el tráfico de una misma red lleva el mismo identificador de túnel:

En el nodo de red:

```
# ovs-ofctl dump-flows br-tun |grep dl_vlan=85
 cookie=0x0, duration=11762.174s, table=21, n_packets=3, n_bytes=230, idle_age=11755, dl_vlan=85
actions=strip_vlan,set_tunnel:0x53,output:4,output:2,output:3
```

En uno de los nodos de computación:

```
# ovs-ofctl dump-flows br-tun|grep dl_vlan=137
 cookie=0x0, duration=11862.572s, table=21, n_packets=234, n_bytes=11708, idle_age=8524,
dl_vlan=137 actions=strip_vlan,set_tunnel:0x53,output:4,output:3,output:2
```

Podemos comprobar que en ambos casos se identifica el tráfico de cada VLAN con el túnel 0x53.

Redes independientes

Utilizamos otro proyecto y creamos una red y una subred con el mismo direccionamiento 10.0.0.0/24 para verificar que neutron gestiona adecuadamente esta situación sin que se produzcan interferencias.

Al lanzar la primera instancia, comprobamos los puertos que se han creado:

```
$ neutron port-list
+--------------------------------------+------+-------------------+-------------------------------
-------------------------------------------------+
| id                                   | name | mac_address       | fixed_ips
                                                 |
+--------------------------------------+------+-------------------+-------------------------------
-------------------------------------------------+
| 03b89c06-87a6-4532-ac7c-397e0b537bfa |      | fa:16:3e:f8:83:5a | {"subnet_id": "16786d0d-2252-
4b86-9f5b-e1e5741b0a14", "ip_address": "10.0.0.2"} |
| 319ca3b4-5fe7-4f45-9b8a-eef0dfda0d87 |      | fa:16:3e:52:c2:89 | {"subnet_id": "16786d0d-2252-
4b86-9f5b-e1e5741b0a14", "ip_address": "10.0.0.3"} |
+--------------------------------------+------+-------------------+-------------------------------
-------------------------------------------------+
```

De nuevo aparecen dos nuevos puertos y lógicamente se ha creado toda la estructura similar a la de antes, en particular un nuevo proceso DHCP en el nodo de red:

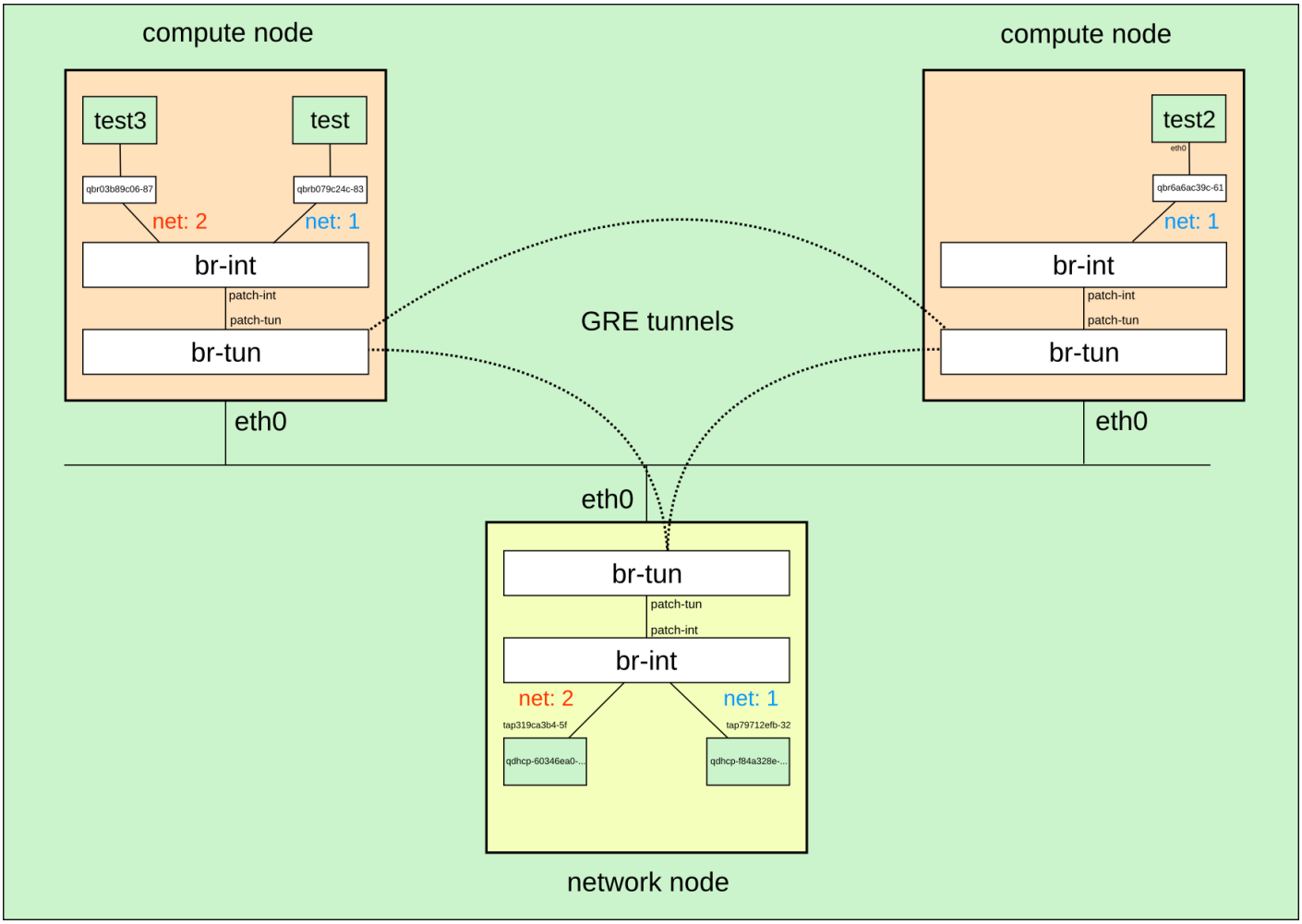
```
# ip netns
qdhcp-f84a328e-407e-4ca6-87bb-ec148853c585
qdhcp-60346ea0-ed9d-40aa-a615-5dd3e33892bc

# ps aux |grep --color 60346ea0-ed9d-40aa-a615-5dd3e33892bc
nobody    1142  0.0  0.0  12888   640 ?        S    13:26   0:00 dnsmasq --no-hosts --no-resolv -
-strict-order --bind-interfaces --interface=tap319ca3b4-5f --except-interface=lo --pid-file=/var/
lib/neutron/dhcp/60346ea0-ed9d-40aa-a615-5dd3e33892bc/pid --dhcp-hostsfile=/var/lib/neutron/dhcp/
60346ea0-ed9d-40aa-a615-5dd3e33892bc/host --dhcp-optsfile=/var/lib/neutron/dhcp/60346ea0-ed9d-40a
a-a615-5dd3e33892bc/opts --leasefile-ro --dhcp-range=tag0,10.0.0.0,static,86400s --dhcp-lease-max
=256 --conf-file= --domain=openstacklocal
```

Vemos que se ha etiquetado una nueva VLAN con el número 2:

```
# ovs-vsctl show
...
    Port "qvo03b89c06-87"
        tag: 76
        Interface "qvo03b89c06-87"
...
```

En la que están conectados el servidor DHCP y el nuevo bridge linux.

Un esquema que representa la creación de las tres instancias es el siguiente:



Conexión con el exterior (Capa 3)

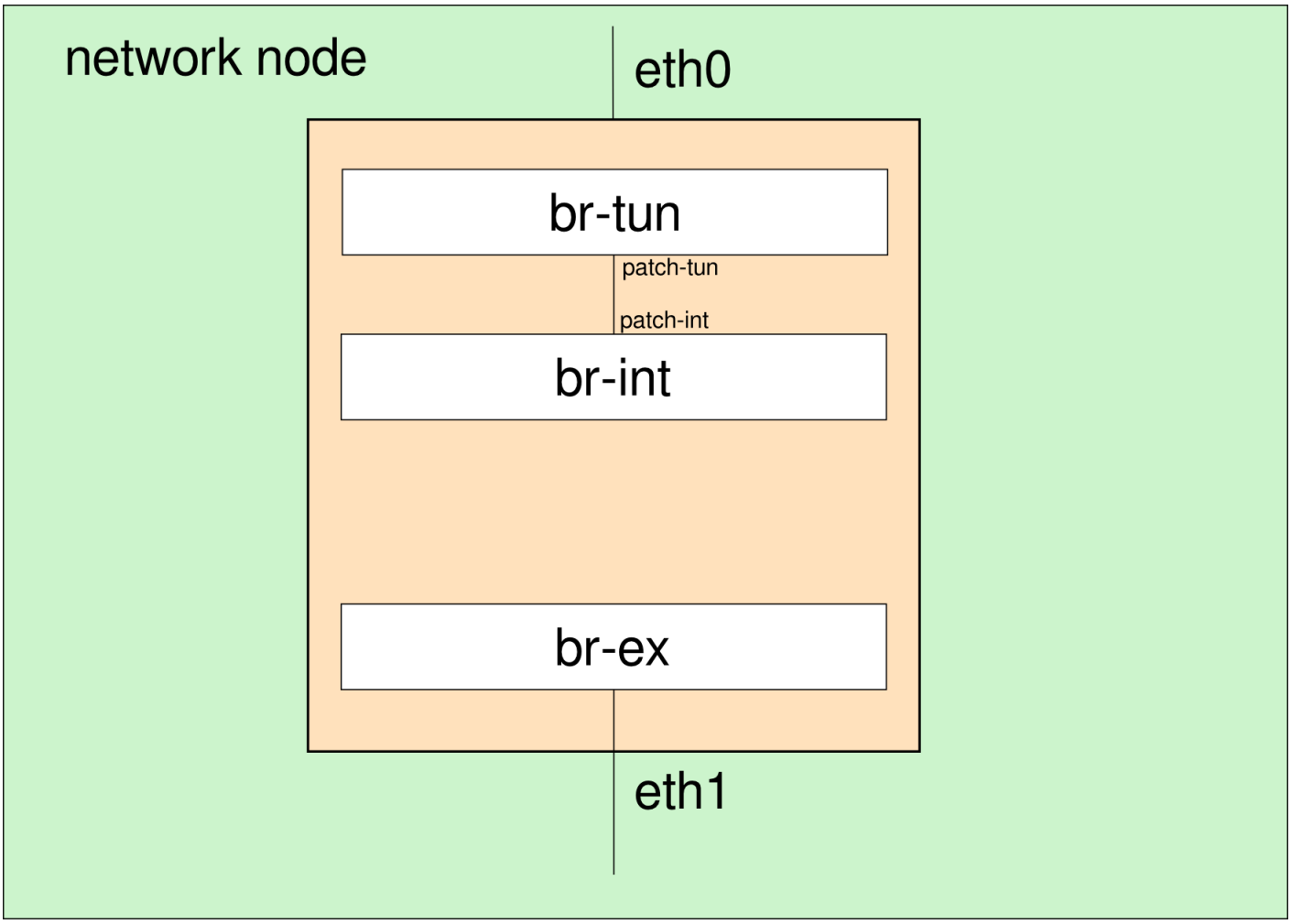
En la sección anterior explicamos los mecanismos que permiten la conectividad entre instancias que se ejecutan en diferentes nodos de computación, así como el aislamiento entre diferentes redes privadas mediante VLAN. En esta sección vamos a explicar de forma detallada el proceso que se sigue para conectar las instancias con el exterior.

Red externa

Supongamos que partimos de una situación inicial en la que tenemos el bridge exterior ya creado en el nodo de red y la interfaz eth1 conectada a él:

```
# ovs-ofctl show br-ex
OFPT_FEATURES_REPLY (xid=0x2): dpid:000000270e035840
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC SET_DL_DST
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 1(eth1): addr:00:27:0e:ff:a8:41
      config:     0
  state:      0
  current:    1GB-FD AUTO_NEG
  advertised: 10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-HD 1GB-FD COPPER AUTO_NEG AUTO_PAUSE AUTO_PAU
SE_ASYM
  supported:  10MB-HD 10MB-FD 100MB-HD 100MB-FD 1GB-HD 1GB-FD COPPER AUTO_NEG
          speed: 100 Mbps now, 1000 Mbps max
 LOCAL(br-ex): addr:00:27:0e:03:58:40
      config:     0
          state:      0
          speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

Esta situación se representa esquemáticamente en la siguiente figura, en la que puede observarse que el bridge exterior está conectado a la interfaz de red física eth1 y desconectado de los otros dos bridges:



La dirección IP de br-ex es la 192.168.0.68 y está en una red 192.168.0.0/24 del que podemos utilizar el rango 192.168.0.15-192.168.0.45 como direcciones IP flotantes.

Creamos como administrador de neutron una red externa:

```
# neutron net-create ext_net -- --router:external=True
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 3292768d-b916-45bc-8ce5-cbab121d6d01 |
| name                      | ext_net                              |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 3                                    |
| router:external           | True                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 635dd2732ceb468e8518e556248c23d0     |
+---------------------------+--------------------------------------+
```

Y asociamos una subred en la que especificamos el depósito de IP asignables:

```
# neutron subnet-create --allocation-pool \
start=192.168.0.15,end=192.168.0.45 ext_net 192.168.0.0/24 -- \
--enable_dhcp=False
Created a new subnet:
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| allocation_pools | {"start": "192.168.0.15", "end": "192.168.0.45"}     |
| cidr             | 192.168.0.0/24                                       |
| dns_nameservers  |                                                      |
| enable_dhcp      | False                                                |
| gateway_ip       | 192.168.0.1                                          |
| host_routes      |                                                      |
| id               | be85bfa6-6dd5-4b59-bdce-614e669c9f3e                 |
| ip_version       | 4                                                    |
| name             |                                                      |
| network_id       | 3292768d-b916-45bc-8ce5-cbab121d6d01                 |
| tenant_id        | 635dd2732ceb468e8518e556248c23d0                     |
+------------------+------------------------------------------------------+
```

Podemos entrar ahora en horizon con un usuario cualquiera y podemos observar en “Topología de red” que ha aparecido una nueva red no gestionada por el usuario

Tenemos por un lado redes privadas gestionadas por cada proyecto y una red exterior gestionada por el administrador, ¿qué elemento de red nos hace falta ahora para poder conectar las redes privadas con la red exterior?

Routers

En el despliegue de OpenStack que estamos utilizando cada proyecto puede definir sus propias redes privadas y routers, estos routers son los dispositivos que permiten conectar las redes privadas con la red exterior y realmente lo que van a hacer es conectar el bridge exterior con el bridge de integración.

Como usuario normal, definimos un nuevo router para conectar la red privada con el exterior:

```
$ neutron router-create r1
Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| external_gateway_info |                                      |
| id                    | cc5fd2f5-59d6-484d-a759-819917a5610c |
| name                  | r1                                   |
| status                | ACTIVE                               |
| tenant_id             | 4beb810ce40f49659e0bca732e4f1a3c     |
+-----------------------+--------------------------------------+
```

Conexión a la red exterior

Conectamos el router a la red exterior, lo que se denomina en neutron configurar la puerta de enlace:

```
$ neutron router-gateway-set cc5fd2f5-59d6-484d-a759-819917a5610c \
3292768d-b916-45bc-8ce5-cbab121d6d01
Set gateway for router cc5fd2f5-59d6-484d-a759-819917a5610c
```

Los routers también se crean en un espacio de nombres de red propio, por lo que podemos comprobar con la instrucción *ip netns* que aparece un nuevo espacio de nombres:

```
# ip netns |grep cc5fd2f5-59d6-484d-a759-819917a5610c
qrouter-cc5fd2f5-59d6-484d-a759-819917a5610c
```

Si vemos las direcciones IP de las interfaces de red en ese espacio de nombres, comprobaremos que existe una interfaz de red con el prefijo “qj-“ y de nuevo 11 caracteres asociados al nuevo puerto creado:

```
# ip netns exec qrouter-cc5fd2f5-59d6-484d-a759-819917a5610c ip addr show
27: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
28: qg-f14d9e65-26: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether fa:16:3e:b8:0a:f9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.15/24 brd 192.168.0.255 scope global qg-f14d9e65-26
    inet6 fe80::f816:3eff:feb8:af9/64 scope link
       valid_lft forever preferred_lft forever
```

Si comprobamos con las credenciales del administrador del cloud los puertos, veremos que aparece un nuevo puerto asociado a la primera dirección IP de nuestra red exterior (192.168.0.15):

```
# neutron port-list|grep 192.168.0.15

| f14d9e65-2608-4b1c-b17d-58c59ce84e8a |      | fa:16:3e:b8:0a:f9 | {"subnet_id": "be85bfa6-6dd5-
4b59-bdce-614e669c9f3e", "ip_address": "192.168.0.15"} |
```

Desde cualquier equipo de la red exterior al que está conectado el nodo de red podemos comprobar que tenemos conexión con el router recién creado:

```
$ ping -c 3 192.168.0.15
PING 192.168.0.15 (192.168.0.15) 56(84) bytes of data.
64 bytes from 192.168.0.15: icmp_req=1 ttl=64 time=0.517 ms
64 bytes from 192.168.0.15: icmp_req=2 ttl=64 time=0.163 ms
64 bytes from 192.168.0.15: icmp_req=3 ttl=64 time=0.175 ms

--- 192.168.0.15 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.163/0.285/0.517/0.164 ms
```

Conexión a la red privada

Conectamos el router a la red privada (es necesario conocer el id de la subred) mediante la instrucción:

```
$ neutron router-interface-add cc5fd2f5-59d6-484d-a759-819917a5610c d4bb2d0e-2af7-44fe-9729-4e3b9
5766e28
Added interface db992449-635f-4267-9d53-3ddb24a9acc1 to router cc5fd2f5-59d6-484d-a759-819917a561
0c.
```

Y podremos comprobar que se ha creado un nuevo puerto en el proyecto asociado a la dirección IP 10.0.0.1 (por defecto se reserva esta dirección IP en cada subred para el router):

```
$ neutron port-list
+--------------------------------------+------+-------------------+-----------------------------
-----------------------------------------------+
```

```
| id                                   | name | mac_address       | fixed_ips
|
+--------------------------------------+------+-------------------+-----------------------------
-------------------------------------------------+
...
| db992449-635f-4267-9d53-3ddb24a9acc1 |      | fa:16:3e:6b:cf:fa |
{"subnet_id": "d4bb2d0e-2af7-44fe-9729-4e3b95766e28", "ip_address": "10.0.0.1"}
|
+--------------------------------------+------+-------------------+-----------------------------
-------------------------------------------------+
```

En el espacio de nombres del router aparece una nueva interfaz de red asociada a este puerto con la dirección IP 10.0.0.1:

```
# ip netns exec qrouter-cc5fd2f5-59d6-484d-a759-819917a5610c ip addr show
...
29: qr-db992449-63: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
link/ether fa:16:3e:6b:cf:fa brd ff:ff:ff:ff:ff:ff
inet 10.0.0.1/24 brd 10.0.0.255 scope global qr-db992449-63
inet6 fe80::f816:3eff:fe6b:cffa/64 scope link
   valid_lft forever preferred_lft forever
```

La denominación de la interfaz sigue el mismo criterio que en casos anteriores, aunque en este caso se utiliza el prefijo “qr-“ y posteriormente los 11 primeros caracteres del puerto con el que se relaciona.

Esta interfaz de red está conectada al bridge de integración, como puede verificarse mediante:

```
# ovs-ofctl show br-int|grep qr-db992449-63
8(qr-db992449-63): addr:7d:02:00:00:00:00
```

Puesto que tenemos una regla de seguridad que permite todo el protocolo ICMP entre los equipos de la red 10.0.0.0/24, podemos hacer ping desde el router a la instancia:

```
# ip netns exec qrouter-cc5fd2f5-59d6-484d-a759-819917a5610c ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.378 ms
...
```

OpenStack neutron añade automáticamente una regla de SNAT para que las instancias de la red privada puedan acceder al exterior utilizando la dirección IP exterior del router:

```
# ip netns exec qrouter-cc5fd2f5-59d6-484d-a759-819917a5610c iptables -t nat -S|grep SNAT
-A neutron-l3-agent-snat -s 10.0.0.0/24 -j SNAT --to-source 192.168.0.15
```

Es decir, todas las máquinas de la red privada pueden acceder al exterior aunque todavía no tengan asociada una dirección IP flotante. Las IPs flotantes se utilizan para

poder iniciar una conexión desde el exterior a una instancia completa, por lo que se utilizan reglas de DNAT de iptables.

IP flotante

Las IP flotantes se definen como las direcciones IPs asociadas a la red exterior, que en un momento determinado se relacionan con una instancia y con su dirección IP fija, pero que pueden relacionarse con otra instancia cuando se desee. Las IPs flotantes permiten que las instancias sean accesibles desde el exterior.

Solicitamos la asignación de una IP flotante al proyecto:

```
$ neutron floatingip-create ext_net
Created a new floatingip:
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    |                                      |
| floating_ip_address | 192.168.0.16                         |
| floating_network_id | 3292768d-b916-45bc-8ce5-cbab121d6d01 |
| id                  | bd31743b-79df-41a4-a40c-62756e868a88 |
| port_id             |                                      |
| router_id           |                                      |
| tenant_id           | 4beb810ce40f49659e0bca732e4f1a3c     |
+---------------------+--------------------------------------+
```

La IP flotante se asocia a un determinado puerto (en este caso al de la máquina test) mediante:

```
$ neutron floatingip-associate bd31743b-79df-41a4-a40c-62756e868a88 b079c24c-8386-47db-a730-35b3a
99419e8
Associated floatingip bd31743b-79df-41a4-a40c-62756e868a88
```

Si ahora vemos las reglas de la tabla nat de iptables en el espacio de nombres del router observaremos las reglas de DNAT asociadas a la IP flotante 192.168.0.16:

```
# ip netns exec qrouter-cc5fd2f5-59d6-484d-a759-819917a5610c iptables -t nat -S
...
-A neutron-l3-agent-OUTPUT -d 192.168.0.16/32 -j DNAT --to-destination 10.0.0.2
...
-A neutron-l3-agent-PREROUTING -d 192.168.0.16/32 -j DNAT --to-destination 10.0.0.2
```

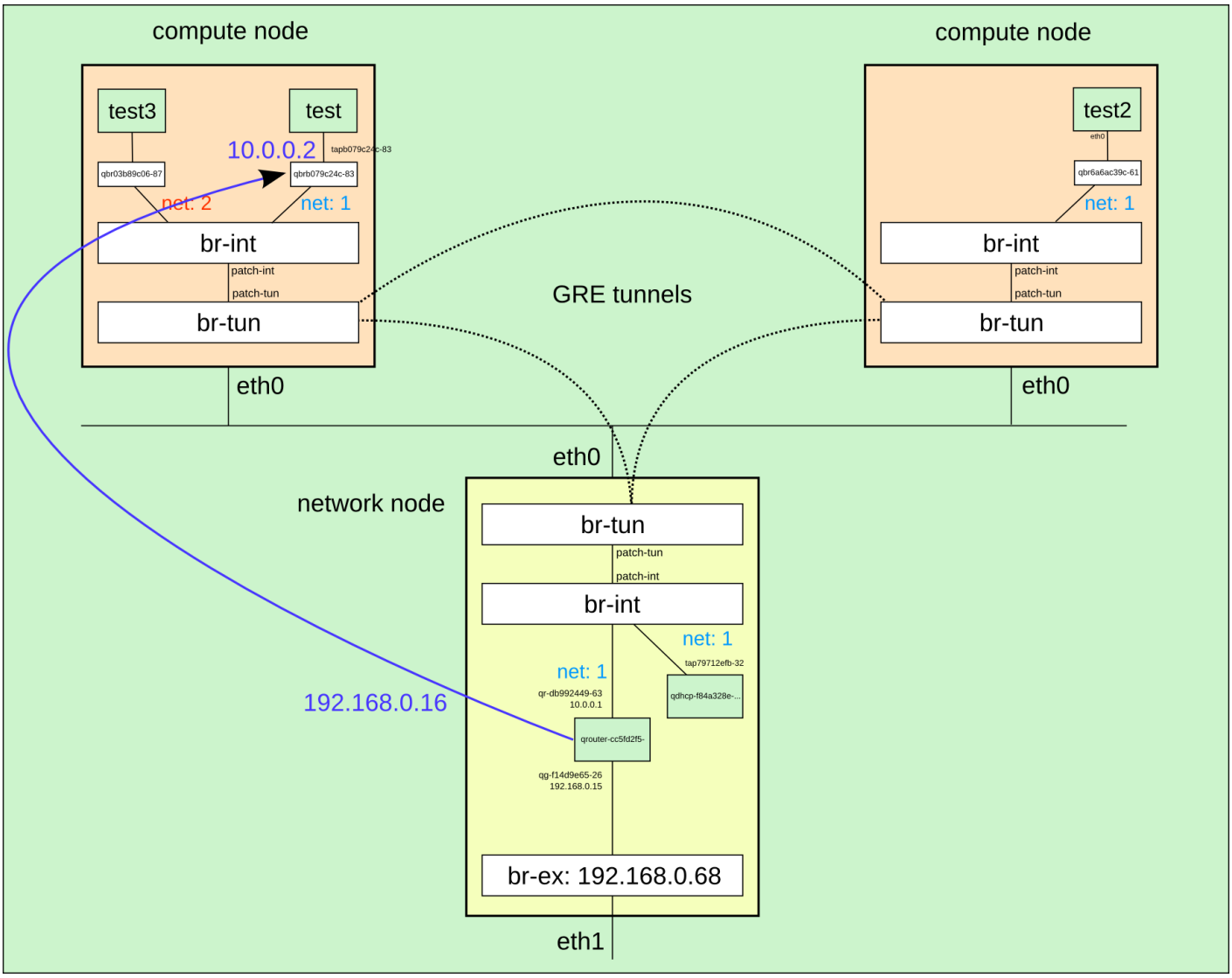
Creamos reglas de seguridad para poder acceder a la instancia por ssh desde la red 192.168.0.0/24:

```
$ neutron security-group-rule-create --direction ingress --protocol tcp --port-range-min 22 --por
t-range-max 22 --remote-ip-prefix 192.168.0.0/24 default
```

Que lógicamente se relaciona con una nueva regla de cortafuegos que se crea en la cadena ib079c24c-8 de la interfaz de red tapb079c24c-83 de la instancia “test” en el nodo de computación:

```
# iptables -S|grep ib079c24c-8
...
-A neutron-openvswi-ib079c24c-8 -s 192.168.0.0/24 -p tcp -m tcp --dport 22 -j RETURN
```

Finalmente podemos ver la situación generada con todas sus interfaces de red y direcciones IP en el siguiente diagrama:



Servidor de metadatos

Al igual que otras plataformas de cloud computing OpenStack proporciona un servicio de metadatos para pasarle a la instancia cierta información específica. Este servicio de

metadatos se utiliza principalmente durante el arranque de la máquina virtual, pero puede utilizarse en cualquier momento que sea preciso.

La instancia espera que el servidor de metadatos esté accesible a través de la URL http://169.254.169.254, puesto que estamos utilizando neutron y cada proyecto gestiona sus propias redes aisladas entre sí, OpenStack tiene que proporcionar un mecanismo para que la URL anterior esté accesible en todas las redes y esto habitualmente se hace mediante un proxy de metadatos en neutron. A continuación vamos a explicar cómo funciona.

En el nodo de red hay que comprobar que exista un proceso del agente de metadatos:

```
neutron   2559  0.0  0.4 178264 69548 ?        Ss   17:37   0:04 \
/usr/bin/python2.7 /usr/bin/neutron-metadata-agent \
--config-file=/etc/neutron/metadata_agent.ini \
--config-file=/etc/neutron/neutron.conf \
--log-file=/var/log/neutron/neutron-metadata-agent.log
```

El servicio de metadatos puede estar tanto asociado a un router del proyecto como a un dispositivo DHCP, aunque lo más habitual es que esté asociado a un router.

Buscamos el identificador del router del proyecto:

```
# neutron router-list |grep "router de test"
| 3e98276c-f107-48a5-97de-f144e10df3df | router de test | {"network_id": "a86fc437-de50-4034-8e1a
-f37a7b05537f", "enable_snat": true} |
```

El espacio de nombres de red correspondiente en el nodo de red:

```
# ip netns |grep 3e98276c-f107-48a5-97de-f144e10df3df
qrouter-3e98276c-f107-48a5-97de-f144e10df3df
```

Y podemos encontrar el proceso del proxy de metadatos funcionando dentro de ese espacio de nombres de red:

```
# ip netns exec qrouter-3e98276c-f107-48a5-97de-f144e10df3df netstat -putan
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address
State       PID/Program name
tcp        0      0 0.0.0.0:9697            0.0.0.0:*    LISTEN      19551/python2.7
```

El proxy de metadatos es un proceso que escucha peticiones en el puerto 9697/tcp, por lo que hay una regla de iptables que redirige las peticiones que los clientes hagan al 169.254.169.254:80:

```
# ip netns exec qrouter-3e98276c-f107-48a5-97de-f144e10df3df iptables -t nat -L -n -v|grep 9697
24  1440 REDIRECT   tcp  --  *      *       0.0.0.0/0    169.254.169.254      tcp dpt:80 redir po
```

```
rts 9697
```

(En este caso se han redirigido 24 paquetes)

El proxy de metadatos se conecta al agente de metadatos del nodo de red a través de un socket UNIX y el agente de metadatos manda la petición al servidor de metadatos a través de la API de nova utilizando una clave secreta compartida que se configura en ambos nodos (8775/tcp en el nodo controlador).

Finalmente podemos comprobar que el servidor de metadatos está esperando peticiones en el puerto 8775/tcp del nodo controlador:

```
# netstat -putan | grep 8775
tcp        0      0 0.0.0.0:8775            0.0.0.0:*    LISTEN      2657/python
```

Referencias

- What's Software-Defined Networking (SDN)?
  Understanding the differences between Software Defined Networking, network virtualization and Network Functions Virtualization
- OpenStack Networking Administration Guide
- RDO - Networking in too much detail