openstack
Mitaka

openstack™
CLOUD SOFTWARE

Francisco Javier Ramiréz Rodríguez

Índice

Introducción

Esta memoria es el resultado del proyecto de fin de ciclo de 2º de ASIR, del IES Gonzalo Nazareno. Realizado por Francisco Javier Ramírez Rodríguez. El proyecto consiste en la instalación de OpenStack mitaka en dos nodos virtuales sobre **Ubuntu 14.04 LTS.**

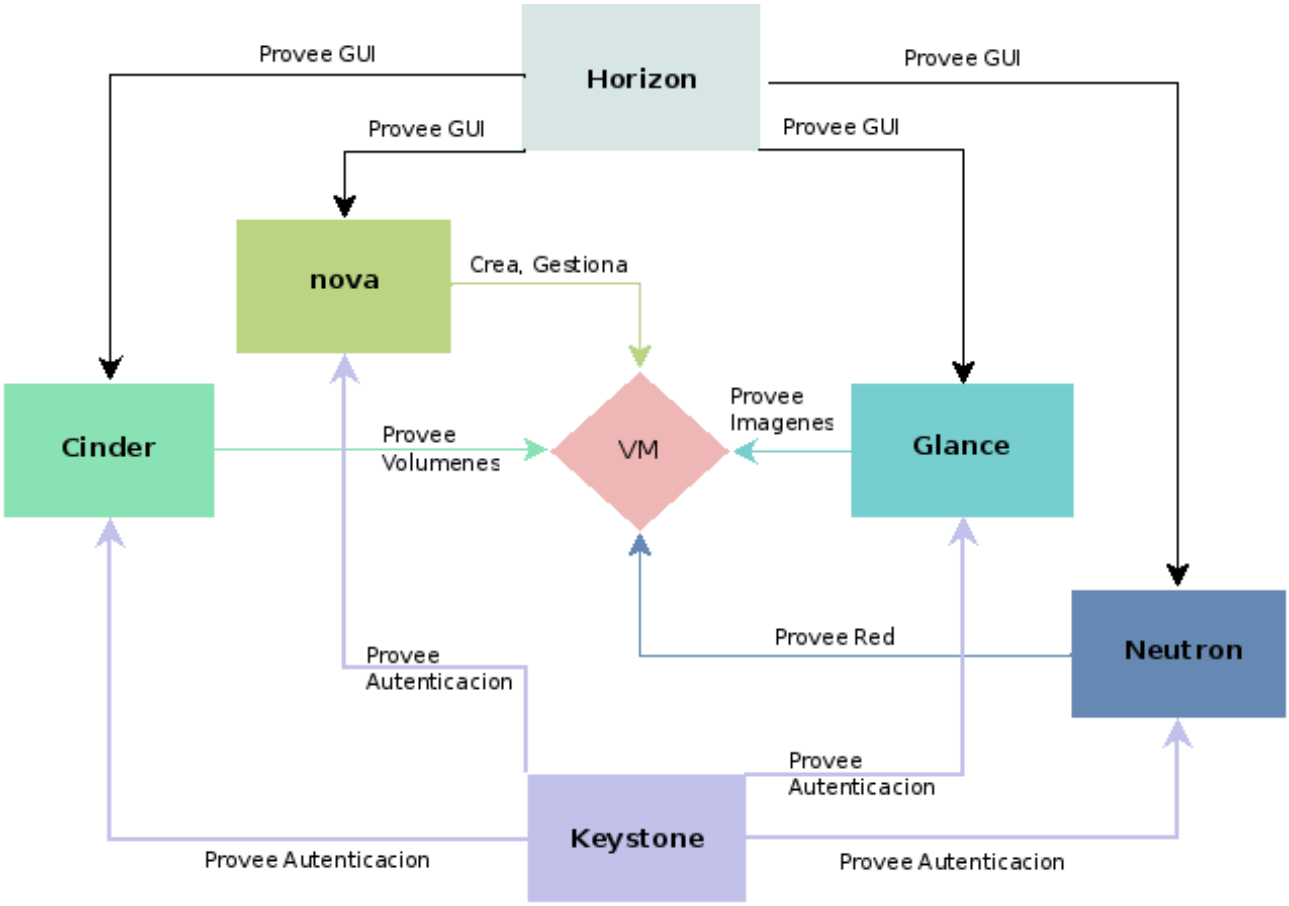
Conceptos

Open**S**tack: El proyecto OpenStack se define así mismo como una plataforma de cloud computing hecha con software libre para desplegar nubes públicas y privadas, desarrollada con la idea de ser sencilla de implementar, masivamente escalable y con muchas prestaciones. OpenStack proporciona una solución de Infraestructura como servicio (IaaS) a través de un conjunto de servicios interrelacionados.

Componentes

OpenStack no es un solo producto, sino un conjunto de componentes que pueden combinarse en función de las características y necesidades de cada caso, hay algunos componentes fundamentales y otros opcionales. OpenStack tiene una gran flexibilidad y permite implementar tanto cloud privado como público. Cada componente de OpenStack es totalmente autónomo y funcional y utiliza el protocolo AMQP de gestión de colas para comunicarse con el resto de componentes (**Rabbit**) y una API web RESTful para comunicarse con procesos "externos" o los usuarios.

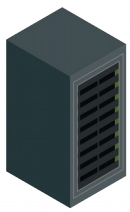
Arquitectura OpenStack



Entorno

Nuestro entorno constará de dos nodos virtualizados bajo kvm, con las siguientes características:

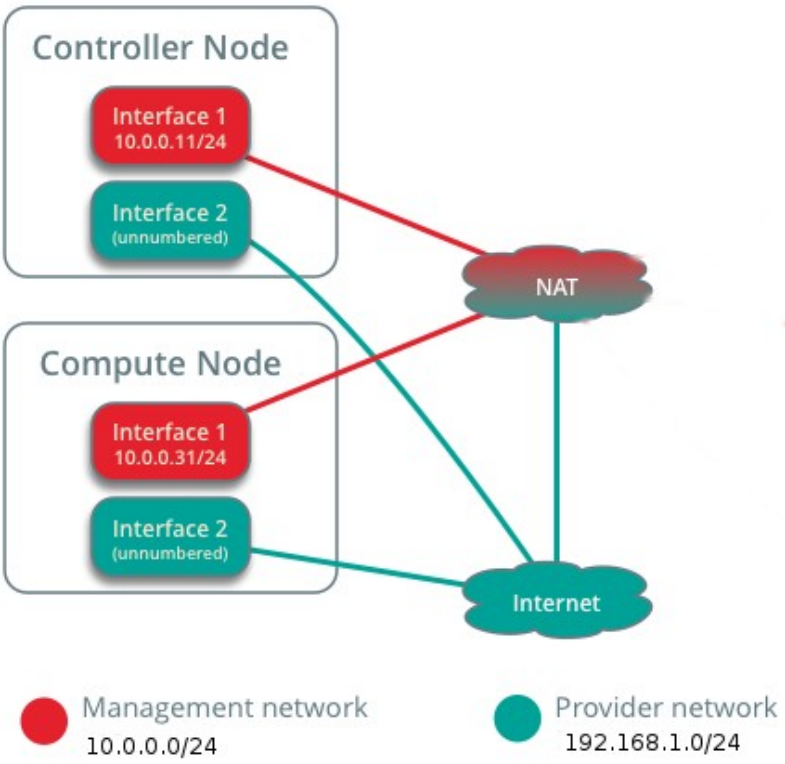
Controller Node: 1 procesador, 4 GB memoria y 15 GB almacenamiento.



Compute Node: 1 procesador, 3 GB memoria y 20 GB almacenamiento.

Esquema de Red.

Como ya hemos comentado nuestro esquema constará de dos nodos. Openstack utiliza dos redes independientes. la red **"management"** que se trata de la red interna utilizada por los componentes para su comunicación entre ellos. y la red "provider" la cual es la encargada de proveer de acceso a internet tanto a los nodos como a las instancias.



Configuraciones

Una vez descrito el proyecto y aclarados los conceptos sobre este pasaremos a la instalación y configuraciones propiamente dichas. Lo primero ha hacer en cada uno de nuestros nodos será la configuración de sus tarjetas de red.

Nodo controller.

Editamos su fichero de configuración **/etc/network/interfaces**

```
auto eth0
iface eth0 inet static
        address 10.0.0.11
        netmask 255.255.255.0
        gateway 10.0.0.1

auto eth1
iface eth1 inet manual
up ip link set dev eth1 up
down ip link set dev eth1
```

Nodo compute.

Editamos su fichero de configuración **/etc/network/interfaces**

```
auto eth0
iface eth0 inet static
        address 10.0.0.31
        netmask 255.255.255.0
        gateway 10.0.0.1

auto eth1
iface eth1 inet manual
up ip link set dev eth1 up
down ip link set dev eth1
```

Tambien es recomendable definir en ambos nodos el fichero **/etc/hosts** para resoluciones de nombres estáticas.:

Nodo controller

```
# controller
10.0.0.11       controller

# compute
10.0.0.31       compute
```

Nodo compute

```
# controller
10.0.0.11       controller

# compute
10.0.0.31       compute
```

Network Time Protocol (NTP)

Una vez configuradas las interfaces de redes de nuestros dos nodos, pasaremos a la configuración de un servidor de hora, con el cual nos aseguraremos que nuestros nodos se encontrarán perfectamente sincronizados.

CONTROLLER NODE:

Instalamos el siguiente paquete:

```
root@controller# aptitude install chrony
```

Una vez instalado pasaremos a las configuraciones, para ello editamos el fichero **/etc/chrony/chrony.conf**

server NTP_SERVER iburst (sustituimos NTP_SERVER por un servidor de hora valido).

Guardamos los cambios y reiniciamos el servicio.

```
root@controller# /etc/init.d/chrony restart
```

COMPUTE NODE:

Instalamos el siguiente paquete:

```
root@controller# aptitude install chrony
```

Una vez instalado pasaremos a las configuraciones, para ello editamos el fichero **/etc/chrony/chrony.conf**

Pero a diferencia del nodo controlador en este definimos como servidor de hora a nuestro nodo controllller.

server controller iburst

Guardamos los cambios y reiniciamos el servicio.

```
root@controller# /etc/init.d/chrony restart
```

Habilitar repositorios Mitaka

Por defecto en Ubuntu 14.04 no vienen añadidos los repositorios de OpenStack Mitaka, para habilitarlos ejecutamos los siguientes comandos:

```
root@controller# apt-get install software-properties-common
root@controller# add-apt-repository cloud-archive:mitaka
```

Para que los cambios surjan efectos debemos actualizar la lista de paquetes en nuestro equipo para ello:

```
root@controller# apt-get update && apt-get dist-upgrade
```

Por ultimo y para tener nuestro entorno completamente configurado para llevar a cabo las configuraciones propiamente de openstack debemos instalar el cliente de openstack de python el cual nos permitirá interactuar con las apis que nos ofrece OpenStack.

```
root@controller# apt-get install python-openstackclient
```

Base de Datos: MariaDB

Cada uno de los componentes de OpenStack requiere de una base de datos la cual utiliza para almacenar información. Openstack nos permite la posibilidad de utilizar tanto bases de datos relacionales como no relaciones en mi caso utilizare MariaDB la cual es un gestor de base de datos derivado de MySQL con licencia GPL.

```
root@controller# apt-get install mariadb-server python-pymysql
```

Una vez instalado los paquetes correspondientes pasaremos a configurar MariaDB, dentro del directorio **/etc/mysql/conf.d** generamos un nuevo fichero de configuración **openstack.conf** con la siguiente configuración

```
[mysqld]
bind-address = 10.0.0.11
default-storage-engine = innodb
innodb_file_per_table
collation-server = utf8_general_ci
character-set-server = utf8
```

Guardamos la configuración y reiniciamos el servicio.

```
root@controller# service mysql restart
```

RabbitMQ

OpenStack utiliza una cola de mensajes para coordinar las operaciones y la información de estado entre los servicios . El servicio de cola de mensajes normalmente se ejecuta en el nodo del controlador . OpenStack es compatible con varios servicios . incluyendo la cola de mensajes RabbitMQ . Qpid . y ZeroMQ . En nuestra instalación utilizaremos Rabbit.

```
root@controller# aptitude install rabbitmq-server
```

Una vez instalado el paquete lo siguiente es añadir un usuario "openstack" con la siguiente instrucción.

```
root@controller# rabbitmqctl add_user openstack RABBIT_PASS
Creating user "openstack" ...
...done.
```

Una vez generado le asignaremos permisos a este usuario tanto de escritura como lectura

```
root@controller#  rabbitmqctl set_permissions openstack ".*" ".*" ".*"
```

Memcached

Se trata de un sistema distribuido de cache. el cual nos permite almacenar datos u objetos. con el fin de agilizar tareas. En openstack es utilizado primordialmente por Keystone (del cual hablaremos posteriormente) memcached es utilizado para almacenar los token de keystone. Comúnmente este servicio corre en el nodo controller. Por defecto este servicio escucha peticiones en el puerto 11211.

Instalamos los paquetes necesarios:

```
root@controller#  apt-get install memcached python-memcache
```

Una vez instalado pasaremos a llevar a cabo las configuraciones pertinentes. Para ello editamos el fichero de configuración `/etc/memcached.conf` y configuraremos memcached para que atienda peticiones en la interfaz ethO de nuestro nodo controller la cual pertenece a la red interna o management. para esto en el fichero de configuración debemos buscar el parámetro -L y especificarle la IP de nuestro nodo controller. Por defecto viene configurado para que solo acepte peticiones de localhost.

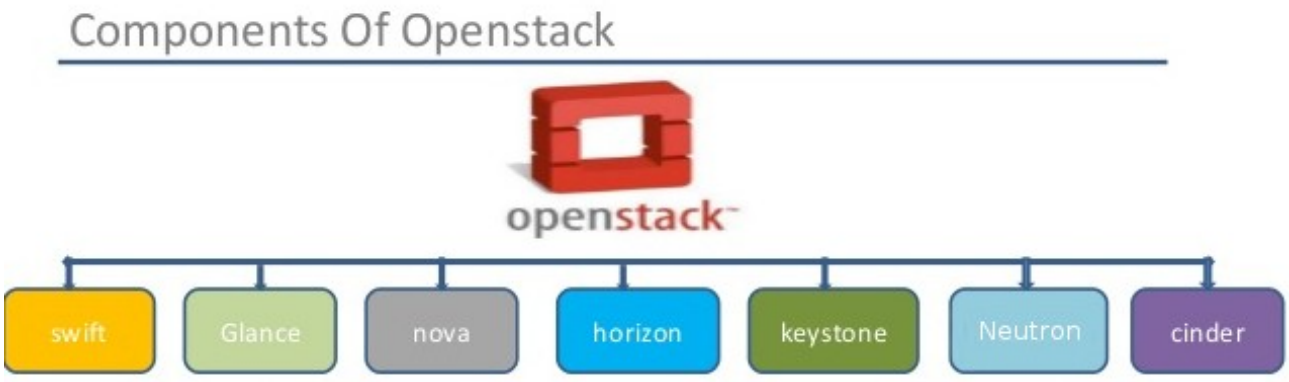
-l 127.0.0.1 → -l 10.0.0.11

Una modificada la linea correspondiente reiniciamos el servicio memcached.

```
root@controller# service memcached restart
```

Componentes OpenStack

Como comentamos anteriormente openstack esta compuesto por una serie de componentes independientes que conviven entre si y cada uno de estos es el encargado de una tarea. a continuación mostrare los componentes principales de openstack. Para posteriormente ser explicados con detalle.



Servicios de identidad (Keystone)

KeyStone es el servicio de identidad que utiliza OpenStack para la autenticación. dicho sistema puede ser integrado con ldap. además puede ser usado a través de nombre de usuario y contraseña estándar. sistemas basados en tokens e inicios de sesión. dispone de una API de identidad (actualmente la versión 3.0) la cual puede ser usada por herramientas de terceros para determinar la accesibilidad de los recursos. En definitiva KeyStone nos va a permitir la autenticación y autorización entre los servicios. siendo instalado en el nodo Controllller.

Para entender cómo funciona KeyStone o Servicio de identidad debemos tener en cuenta los siguientes componentes:

- **Usuario**: Pueden ser personas, sistemas o servicios que usen la nube de OpenStack, KeyStone será el encargado de validar las peticiones de éste. Los usuarios disponen de un login.
- **Credenciales**: Es el dato que va a usar el usuario para su autenticación, puede ser un login y contraseña, un token, una clave de la API...
- **Autenticación**: Es el acto de confirmar la identidad del usuario usando las credenciales provistas. Una vez validada la autenticación se provee de un token pasar ser usado en las peticiones del usuario, de esta forma no necesita autenticarse de nuevo.
- **Token**: Es un bit de texto usado para acceder a los recursos, es generado cuando se valida la autenticación, cada toquen tiene un alcance el cual nos permite saber a que recursos puede acceder un usuario.
- **Tenant**: Es el contenedor usado para agrupar y aislar los recursos. Los tentant pueden ser clientes, organizaciones, cuentas o proyectos.
- **Servicio**: cada uno de los servicios que conforman OpenStack,
- **Endpoint**: Es el direccionamiento IP o URL para acceder a los servicios.
- **Roles**: Para especificar o limitar las operaciones de usuarios KeyStone usa roles, los cuales contienen un grupo de privilegios para determinar el alcance del usuario.

Instalación keystone

Como comentábamos anteriormente cada componente de openstack requiere de un base de datos donde posteriormente se generaran tablas en las cuales almacenarán sus datos.

Creación base de datos.

```
root@controller#  mysql -u root -p

mysql> CREATE DATABASE keystone;

mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost'  IDENTIFIED BY 'KEYSTONE_DBPASS';

mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%'  IDENTIFIED BY 'KEYSTONE_DBPASS';
```

Para la configuración inicial de keystone es necesario generar un token temporal. para ello:

```
root@controller#  openssl rand -hex 10
```

Previamente a la instalación de los paquetes necesarios. debemos deshabilitar el arranque automatico de keystone tras su instalación. Para ello:

```
root@controller#  echo "manual" > /etc/init/keystone.override
```

Instalación y configuración de los componentes

```
root@controller# apt-get install keystone apache2 libapache2-mod-wsgi
```

***** Como observamos instalamos apache2 y el modulo wsgi de este, esto es debido a que en esta versión Mitaka, keystone a pasado a configurarse mediante apache2 a través de un virtualhost el cual escucha peticiones en los puertos de escucha de kesytone

*** Mod_wsgi** se trata de un módulo de Apache, que nos ofrece la posibilidad de servir aplicaciones hechas en Python.

Una vez instalado los paquetes pasamos a las configuraciones. Los ficheros de configuración de OpenStack vienen diferenciados con secciones por lo que realizando búsquedas dentro de los ficheros de configuración podemos agilizar bastante la tarea de configuración. **/etc/keystone/keystone.conf**

En la sección **[DEFAULT]** debemos especificar el token aleatorio generado anteriormente.

[DEFAULT]

admin_token = **ADMIN_TOKEN**

En la sección **[database]** debemos configurar el acceso a la base de datos previamente creada.

[database]

connection = mysql+pymysql://keystone:**KEYSTONE_DBPASS**@controller/keystone

En la sección **[token]** debemos configurar como provedor de token **'fernet'**

[token]

provider = **fernet**

***El proyecto de identidad Keystone utiliza tokens Fernet que emplean claves privadas compartidas en su carga útil para que estos tokens no tengan que ser almacenados y no se produzcan réplicas en la base de datos. Esto reduce la carga situada en la base de datos de Keystone, lo que permite que el servicio crezca con el fin de aceptar muchas más llamadas de la API.

Guardamos los cambios y realizamos la sincronización con la base de datos de keystone.

root@controller# su -s /bin/sh -c "keystone-manage db_sync" keystone

Por ultimo inicializamos los token fernet.

root@controller# keystone-manage fernet_setup --keystone-user keystone --keystone-group keystone

Configuración Apache2 + Mod_WSGI

Lo primero es editar el fichero de configuración **/etc/apache2/apache2.conf** en el cual especificaremos como **ServerName** a nuestro nodo controller.

ServerName **controller**

En Segundo lugar vamos a generar un nuevo virtualhost en el cual configuraremos keystone mediante el módulo wsgi. Accedemos al directorio **/etc/apache2/sites-available** y en el generaremos un fichero llamado **wsgi-keystone.conf** con el siguiente contenido.

```
Listen 5000
Listen 35357

<VirtualHost *:5000>
    WSGIDaemonProcess keystone-public processes=5 threads=1 user=keystone group=keystone
display-name=%{GROUP}
    WSGIProcessGroup keystone-public
    WSGIScriptAlias / /usr/bin/keystone-wsgi-public
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    ErrorLogFormat "%{cu}t %M"
    ErrorLog /var/log/apache2/keystone.log
    CustomLog /var/log/apache2/keystone_access.log combined

    <Directory /usr/bin>
        Require all granted
    </Directory>
</VirtualHost>

<VirtualHost *:35357>
    WSGIDaemonProcess keystone-admin processes=5 threads=1 user=keystone group=keystone
display-name=%{GROUP}
    WSGIProcessGroup keystone-admin
    WSGIScriptAlias / /usr/bin/keystone-wsgi-admin
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    ErrorLogFormat "%{cu}t %M"
    ErrorLog /var/log/apache2/keystone.log
    CustomLog /var/log/apache2/keystone_access.log combined

    <Directory /usr/bin>
        Require all granted
    </Directory>
</VirtualHost>
```

Guardamos la configuración y generamos un enlace simbólico haciendo referencia al directorio

/etc/apache2/sites-enabled.

root@controller# ln -s /etc/apache2/sites-available/wsgi-keystone.conf /etc/apache2/sites-enabled

Por ultimo y para terminar reiniciamos el servicio de Apache2 y eliminar la base de datos sqlite que por defecto trae consigo Keystone.

root@controller# service apache2 restart

root@controller# rm -f /var/lib/keystone/keystone.db

Creación Servicio Keystone y Endpoints.

Keystone nos ofrece un catalogo de servicios y sus ubicaciones. por lo que cada componente que añadamos a nuestro entorno debe ser añadido al catalogo de keystone. Para comunicarnos con keystone es necesario declara una serie de variables.

export OS_TOKEN = **ADMIN_TOKEN**

export OS_URL = **http://controller:35357/v3**

export OS_IDENTITY_API_VERSION = **3**

Una vez exportadas las variables creamos el servicio de keystone.

root@controller# openstack service create --name keystone --description "OpenStack Identity" identity

```
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Identity               |
| enabled     | True                             |
| id          | 4ddaae90388b4ebc9d252ec2252d8d10 |
| name        | keystone                         |
| type        | identity                         |
+-------------+----------------------------------+
```

Como comentábamos anteriormente Keystone maneja un catalogo en el cual asocia API Endpoints con los servicios/componentes de nuestro entorno. OpenStack usa una **variante de 3 EndPoint** para cada servicio: **admin, internal y public**. **Admin** permite modificar los usuarios y los tenants por defecto, mientras que los otros no permiten estas operaciones. En ambientes de Producción, las variantes pueden residir en redes distintas por razones de seguridad. Por Ejemplo el **Endpoint público** puede ser accedido desde internet para que los usuarios administren sus nubes, y el **Endpoint interno** solo accesible para operadores dentro de la organización.

root@controller# openstack endpoint create --region RegionOne identity **public** http://controller:5000/v3

root@controller# openstack endpoint create --region RegionOne identity **internal** http://controller:5000/v3

root@controller# openstack endpoint create --region RegionOne identity **admin** http://controller:5000/v3

Creación dominio, proyectos, usuarios y roles

El servicio de identidad proporciona servicios de autenticación para cada servicio OpenStack . El servicio de autenticación utiliza una combinación de dominios , proyectos (tenants), usuarios y roles.

Creación Dominio Default

root@controller# openstack domain create --description "Default Domain" default

Creación proyecto Admin

root@controller# openstack project create --domain default --description "Admin Project" admin

Creación Usuario Admin

root@controller# openstack user create --domain default --password-prompt admin

User Password: ********
Repeat User Password: ********

Creación Rol Admin

root@controller# openstack role create admin

Por ultimo añadimos el **rol de admin** al proyecto y usuario **'admin'**.

root@controller# openstack role add --project admin --user admin admin

Ahora generaremos un nuevo proyecto "**service**" que contendrá un usuario único para cada componente que añadamos a nuestro entorno.

root@controller# openstack project create --domain default --description "Service Project" service

Una vez terminadas las configuraciones podemos comprobar que keystone esta funcionando de manera correcta solicitando un token. Nos solicitara las password del usuario admin establecida en su creación.

root@controller# openstack --os-auth-url http://controller:35357/v3 --os-project-domain-name default --os-user-domain-name default --os-project-name admin --os-username admin token issue

```
Password:********

+------------+-----------------------------------------------------------------+

| Field      | Value                                                           |
+------------+-----------------------------------------------------------------+
| expires    | 2016-02-12T20:14:07.056119Z                                     |
| id         | mjEO643d-e_j-XXq9AmIegIbA7UHGPv |
|            | wenN21qtOMjCtyhejjkJEQnVOfj3nclRQgAYRsfSU_Mrsic74dñp9cb7HEpoBb4 |
|            | o6ozsA_NmFWEpLeKy0uNn_WeKbAhYygrsmQGA49dclHVnz-O85fg5r2ws       |
| project_id | 3435g45e850143a096806dfaefa9afdc                                |
| user_id    | ac33776331494012961f6c0d92d79dc16                               |
+------------+-----------------------------------------------------------------+
```

Creación Script Openstack Admin.

Generamos un nuevo fichero de nombre admin-openrc con el siguiente contenido:

```
export OS_PROJECT_DOMAIN_NAME = default
export OS_USER_DOMAIN_NAME = default
export OS_PROJECT_NAME = admin
export OS_USERNAME = admin
export OS_PASSWORD = ADMIN_PASS
export OS_AUTH_URL = http://controller:35357/v3
export OS_IDENTITY_API_VERSION = 3
export OS_IMAGE_API_VERSION = 2
```

Cada vez que queramos interactuar con las apis de openstack deberemos realizar un source sobre el fichero recientemente creado.

root@controller# openstack token issue

```
+------------+----------------------------------------------------------------+
| Field      | Value                                                          |
+------------+----------------------------------------------------------------+
| expires    | 2016-02-12T20:44:35.659723Z                                    |
| id         | gAAAAABWvjYj-Zjfg8WXFaQnUd1DMYTBVrKw4h3fIagi5NoEmh21U72SrRv2trl |
|            | JWFYhLi2_uPR31Igf6A8mH2Rw9kv_bxNo1jbLNPLGzW_u5FC7InFqx0yYtTwa1e |
|            | eq2b0f6-18KZyQhs7F3teAta143kJEWuNEYET-y7u29y0be1_64KYkM7E       |
| project_id | 343d245e850143a096806dfaefa9afdc                               |
| user_id    | ac3377633149401296f6c0d92d79dc16                               |
+------------+----------------------------------------------------------------+
```

Servicio Gestión de Imágenes (Glance)

Glance es el servicio que proporciona el registro. descubrimiento y entrega de imágenes de máquinas virtuales. Las imágenes almacenadas se pueden utilizar como plantillas para un rapido despliegue de nuevos servidores sin tener que realizar la instalación del sistema operativo. También puede ser utilizado para almacenar un número ilimitado de backups.

Las imagenes gestionadas por Glance pueden encontrarse en uno de los siguientes estados:

- **queued:** el identificador de la imagen ha sido reservado en el registro del servicio Glance pero los datos de la imagen no se han añadido al servicio.
- **saving:** indica que los datos de la imagen se están añadiendo en ese momento al servicio Glance.
- **active:** la imagen está completamente disponible en Glance. Esto ocurre cuando los datos de la imagen se han añadido al servicio o el tamaño de la imagen se ha establecido en cero durante la creación.
- **killed:** ha ocurrido un error cuando se estaban añadiendo los datos de la imagen y la imagen no puede ser accesible.
- **deleted:** Glance ha mantenido la información de la imagen, pero no está disponible para su uso. Una imagen en este estado será eliminada de forma automática más adelante.
- **pending_delete:** es similar al estado deleted, pero Glance todavía no ha borrado los datos de la imagen. Una imagen en este estado es recuperable.

Instalación de Glance

De igual manera que con cada componente deberemos generar una base de datos para glance.

Creación base de datos.

root@controller# mysql -u root -p

mysql> CREATE DATABASE glance;

mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost' IDENTIFIED BY 'GLANCE_DBPASS';

mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' IDENTIFIED BY 'GLANCE_DBPASS';

Una vez generada la base de datos deberemos de generar al usuario GLANCE. para ello debemos cargar el script de variables anteriormente creado.

root@controller# source admin-openrc

Creación Usuario glance

root@controller# openstack user create --domain default --password-prompt glance

User Password: ********
Repeat User Password: ********

Añadir usuario Glance rol Admin.

root@controller# $ openstack role add --project service --user glance admin

Añadir usuario Glance rol Admin.

root@controller# openstack role add --project service --user glance admin

Creación Servicio Keystone y Endpoints.

De igual manera que con keystone debemos registrar a glance como servicio. para ello:

root@controller# openstack service create --name glance --description "OpenStack Image" image

Endpoint public.

root@controller# openstack endpoint create --region RegionOne image **public** http://controller:9292

Endpoint internal.

root@controller# openstack endpoint create --region RegionOne image **internal** http://controller:9292

Endpoint admin

root@controller# openstack endpoint create --region RegionOne image **admin** http://controller:9292

Una vez registrado glance en keystone como servicio etc. podemos llevar a cabo las configuraciones de glance.

Instalación y configuración de los componentes

root@controller# aptitude install glance

Una vez instalado pasamos a editar sus ficheros de configuración glance consta de dos ficheros

/etc/glance/glance-api.conf y /etc/glance/glance-registry.conf

Editamos /etc/glance/glance-api.conf

En la sección **[database]** debemos configurar el acceso a la base de datos previamente creada.

[database]

connection = mysql+pymysql://glance:**GLANCE_DBPASS**@controller/glance

En la sección **[keystone_authtoken]** configuramos la integración de glance con keystone.

[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = glance
password = **GLANCE_PASS**

En la sección **[paste_deploy]** le indicamos a glance que el flavor es keystone..

[paste_deploy]

flavor = keystone

En la sección **[glance_store]** le indicamos el directorio en el cual almacenara las imágenes.

[glance_store]

stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/

Editamos /etc/glance/glance-registry.conf

En la sección **[database]** debemos configurar el acceso a la base de datos previamente creada.

[database]

connection = mysql+pymysql://glance:**GLANCE_DBPASS**@controller/glance

De igual manera que anteriormente debemos configurar la integración de este servicio con keystone

[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = glance
password = **GLANCE_PASS**

En la sección **[paste_deploy]** le indicamos a glance que el flavor es keystone..

```
[paste_deploy]
flavor = keystone
```

Con esto ya tendriamos terminadas las configuraciones y ya podemos realizar la sincronización con la base de datos para que glance genere su estructura de tablas pertinentes. para ello:

```
root@controller# su -s /bin/sh -c "glance-manage db_sync" glance
```

Cuando finalice la sincronización, reiniciamos ambos servicios.

```
root@controller# service glance-registry restart
root@controller# service glance-api restart
```

Servicios de computación (Nova)

Nova es el componente principal del IaaS (infraestructura como servicio) y es el encargado de administrar los pools de recursos que tenemos disponibles. hay que tener en cuenta que Nova NO es un hypervisor, sino el gestor de recursos de hipervisores tales como: *Xen, KVM, etc...*

Nodo controller

Creación base de datos.

```
root@controller#  mysql -u root -p
mysql> CREATE DATABASE nova;
mysql> CREATE DATABASE nova_api;
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost'  IDENTIFIED BY 'NOVA_DBPASS';
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%'  IDENTIFIED BY 'NOVA_DBPASS';
mysql> GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'localhost'  IDENTIFIED BY 'NOVA_DBPASS';
mysql> GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'%'  IDENTIFIED BY 'NOVA_DBPASS';
```

Una vez generada la base de datos deberemos de generar al usuario NOVA. para ello debemos cargar el script de variables anteriormente creado.

root@controller# source admin-openrc

Creación Usuario Nova

root@controller# openstack user create --domain default --password-prompt nova

User Password: ********
Repeat User Password: ********

Añadir usuario Nova rol Admin.

root@controller# $ openstack role add --project service --user nova admin

Creación Servicio Nova y Endpoints.

De igual manera que con los demas componentes debemos registrar a nova como servicio. para ello:

root@controller# openstack service create --name nova   --description "OpenStack compute" compute

Endpoint public.

root@controller# openstack endpoint create --region RegionOne compute public http://controller:8774/v2.1/%\(tenant_id\)s

Endpoint internal.

root@controller# openstack endpoint create --region RegionOne compute internal http://controller:8774/v2.1/%\(tenant_id\)s

Endpoint admin

root@controller# openstack endpoint create --region RegionOne compute admin http://controller:8774/v2.1/%\(tenant_id\)s

Una vez registrado nova en keystone como servicio etc. podemos llevar a cabo las configuraciones de nova , propiamente dichas.

Instalación y configuración de los componentes

root@controller# apt-get install nova-api nova-conductor nova-consoleauth nova-novncproxy nova-scheduler

Editamos el fichero /etc/nova/nova.conf

En la sección **[DEFAULT]** habilitamos las siguientes opciones.

[DEFAULT]

enabled_apis = osapi_compute,metadata

rpc_backend = rabbit

auth_strategy = keystone

my_ip = 10.0.0.11 (**debemos especificar la interfaz management**)

use_neutron = True

firewall_driver = nova.virt.firewall.NoopFirewallDriver

Como observamos anteriormente nova consta de dos bases de datos por lo que la conexión debemos realizarla contra ambas.

En las secciones **[api_database]** y en la sección **[database]** configuramos la conexión con la BD.

[api_database]

connection = mysql+pymysql://nova:**NOVA_DBPASS**@controller/nova_api

[database]

connection = mysql+pymysql://nova:**NOVA_DBPASS**@controller/nova

En la sección **[oslo_messaging_rabbit]** configuramos los parámetros de rabbit.

rabbit_host = controller

rabbit_userid = openstack

rabbit_password = **RABBIT_PASS**

En la sección **[keystone_authtoken]** configuramos la integración de nova con keystone.

[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = nova
password = **NOVA_PASS**

En la sección **[vnc]** . configurar el proxy de VNC para utilizar la dirección IP de la interfaz de gestión del nodo del controlador :

[vnc]

vncserver_listen = **$my_ip**

vncserver_proxyclient_address = **$my_ip**

En la sección **[glance]** debemos indicarle a nova la localización se servicio de imagenes nova.

[glance]

api_servers = http://controller:9292

En la sección [oslo_concurrency] configuramos el **lock_path**

[oslo_concurrency]

lock_path = /var/lib/nova/tmp

Nota: Debido a un error de empaquetamiento , quitar la opción **logdir** de la sección **[DEFAULT]** .

Con esto ya tendriamos terminadas las configuraciones y ya podemos realizar la sincronización con la base de datos para que nova genere su estructura de tablas pertinentes. para ello:

```
root@controller# su -s /bin/sh -c "nova-manage api_db sync" nova
root@controller# su -s /bin/sh -c "nova-manage db sync" nova
```

Por ultimo reiciniciamos los servicios.

```
root@controller# service nova-api restart
root@controller# service nova-consoleauth restart
root@controller# service nova-scheduler restart
root@controller# service nova-conductor restart
root@controller# service nova-novncproxy restart
```

Nodo compute

En el nodo de computo también debemos configurar nova para ello. instalamos el paquete necesario.

```
root@compute# apt-get install nova-compute
```

Editamos el fichero **/etc/nova/nova.conf**.

En la sección **[DEFAULT]** añadimos la siguiente configuración:

```
[DEFAULT]
rpc_backend = rabbit
auth_strategy = keystone
my_ip = 10.0.0.31        (MANAGEMENT_INTERFACE_IP_ADDRESS)
use_neutron = True
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```

En la sección **[oslo_messaging_rabbit]** configuramos los parámetros de rabbit.:

rabbit_host = controller

rabbit_userid = openstack

rabbit_password = **RABBIT_PASS**

En la sección **[keystone_authtoken]** integramos a nova con keystone.

[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = nova
password = **NOVA_PASS**

En la sección **[vnc]** configuramos la conexión remota a las consolas:

[vnc]

enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = **$my_ip**
novncproxy_base_url = http://controller:6080/vnc_auto.html

En la sección **[glance]** le indicamos al servicio nova la localización de glance.

[glance]

api_servers = http://controller:9292

En la sección **[oslo_concurrency]** configuramos el lock_path.

[oslo_concurrency]

lock_path = /var/lib/nova/tmp

Lo siguiente es indicarle a nuestro nodo de computo que tipo de virtualización va a utilizar en nuestro caso utilizaremos KVM (qemu) Para ello lo primero es verificar que nuestro nodo de computo soporta la aceleración por hardware. Para ello:

root@compute# egrep -c '(vmx|svm)' /proc/cpuinfo

Si el resultado es uno o superior no debemos de realizar ninguna configuración adicional, en caso contrario de no ser asi debemos añadir la siguiente configuración.

Editamos /etc/nova/nova-compute.conf

[libvirt]

virt_type = qemu

Por ultimo reiniciamos el servicio.

root@compute# service nova-compute restart

Para verificar que nova funciona de manera correcta cargamos el script de variables y ejecutamos el siguiente comando:

root@controller# source admin-openrc

root@controller# openstack compute service list

```
+----+------------------+------------+----------+---------+-------+----------------------------+--------------------------------+
| Id | Binary           | Host       | Zone     | Status  | State | Updated At
|
+----+------------------+------------+----------+---------+-------+----------
|  1 | nova-consoleauth | controller | internal | enabled | up    | 2016-05-
|  2 | nova-scheduler   | controller | internal | enabled | up    | 2016-05-
|  3 | nova-conductor   | controller | internal | enabled | up    | 2016-05-
|  4 | nova-compute     | compute    | nova     | enabled | up    | 2016-05-
-------+------------------------------+----------------+--------------+------------+----------+------------+
```

Servicios de Redes (Neutron)

Tenemos que tener en cuenta que en versiones anteriores de OpenStack, **podemos encontrarnos con Nova-network** como el servicio de administración de red, pero desde Juno éste, **es sustituido por Neutron**, aunque podemos usarlo. Neutron nos permite hacer uso de su API a través de plugins y agentes capaces de conectar/ desconectar puertos, aprovisionar redes y subredes, etc, todo ello vinculado al proveedor que vayamos a usar en nuestra nube, como por ejemplo switches físicos o virtuales de Cisco, productos de NEC OpenFlow, Open vSwitch, Linux bridge, Ryu Network Operating System o VMware NSX.

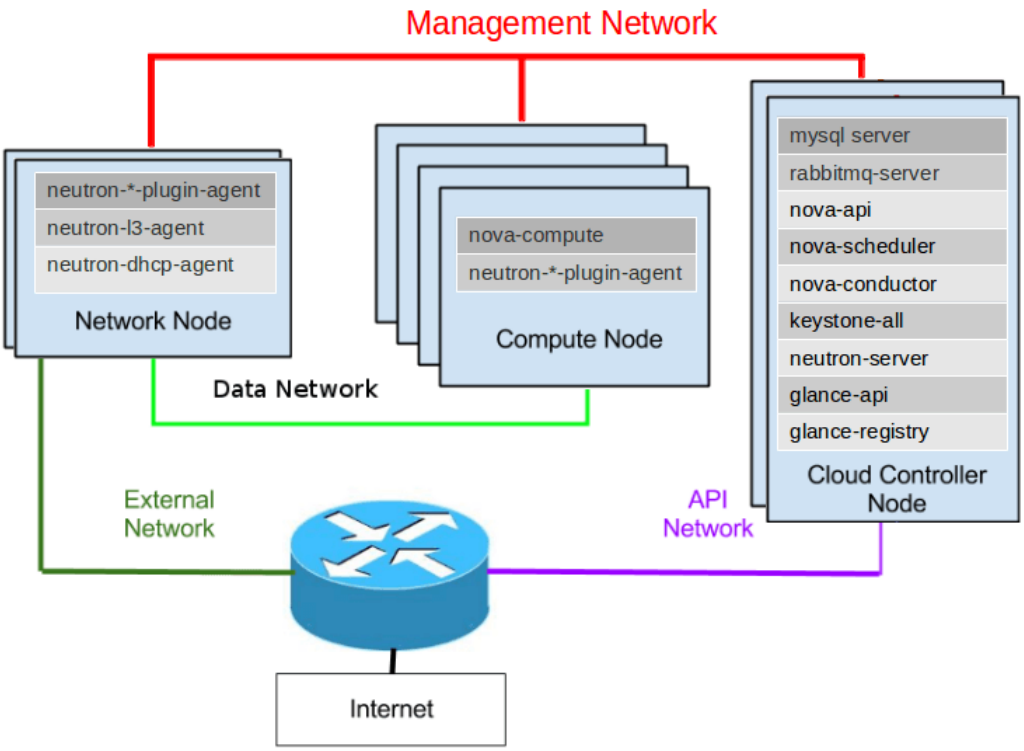
Los tres principales componentes de neutron son:

Neutron L3 agent

- **DHCP agent**
- **Neutron plugin agent.**
- **Neutron L3 agent**

Normalmente neutron se instala de manera independiente en un nodo (nodo de red) en nuestra implementación neutron será instalado en nuestro nodo controlador.

Aquí muestro un esquema de como es la comunicación de neutrón con los demas nodos tanto **controller** como **compute.**



Como comentábamos anteriormente neutron es un componente bastante complejo, debido a la gran variedad de posibilidades de configuración que nos ofrece, la documentación oficial de openstack nos ofrece una seria de escenarios de redes que podrían ser posibles configuraciones en función de los requisitos de cada uno, en mi caso la configuración seleccionada ha sido **Linux_bridge con VXLAN**

Nodo controller

Creación base de datos.

root@controller# mysql -u root -p

mysql> CREATE DATABASE **neutron**;

mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'nova'@'localhost' IDENTIFIED BY 'NEUTRON_DBPASS';

mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' IDENTIFIED BY 'NEUTRON_DBPASS';

Una vez generada la base de datos deberemos de generar al usuario NEUTRON. para ello debemos cargar el script de variables anteriormente creado.

root@controller# source admin-openrc

Creación Usuario Neutron

root@controller# openstack user create --domain default --password-prompt neutron

User Password: ********
Repeat User Password: ********

Añadir usuario Neutron rol Admin.

root@controller# $ openstack role add --project service --user neutron admin

Creación Servicio Neutron y Endpoints.

De igual manera que con los demás componentes debemos registrar a nova como servicio. para ello:

root@controller# openstack service create --name neutron   --description OpenStack Networking" network

Endpoint public.

root@controller# openstack endpoint create --region RegionOne network public http://controller:9696

Endpoint internal.

root@controller# openstack endpoint create --region RegionOne network internal http://controller:9696

Endpoint admin

root@controller# openstack endpoint create --region RegionOne network admin http://controller:9696

Una vez registrado neutron en keystone como servicio etc. podemos llevar a cabo las configuraciones de neutron , propiamente dichas.

Instalación y configuración de los componentes

Neutron nos ofrece dos arquitecturas de redes diferentes:

Provider Network: Se trata de la configuración mas básica, la cual solo soporta añadir instancias a la red 'provider' red externa, por lo que solo el usuario **admin** podría gestionar esta infraestructura

Self-service Network: A diferencia de la anterior con esta opción disponemos de un abanico mas amplio de posibilidades, es decir, cada usuario puede gestionar su propia infraestructura de red (crear router, crear subredes) y la posibilidad de generar ip flotante con las que proveer de acceso a internet a las instancias. Nosotros elegiremos la segunda opción.

En nuestro nodo controlador instalamos los siguientes paquetes:

root@controller# apt-get install neutron-server neutron-plugin-ml2 neutron-linuxbridge-agent neutron-l3-agent neutron-dhcp-agent neutron-metadata-agent

Una vez instalado pasamos a realizar las configuraciones. tambien cabe destacar que nuestro esta

conformado por un cojunto de plugins los cuales se encargan cada uno de una tarea concreta.

Editamos el fichero /etc/neutron/neutron.conf

En la sección **[DEFAULT]** añadimos la siguiente configuración:

[DEFAULT]

rpc_backend = rabbit

auth_strategy = keystone

core_plugin = ml2

service_plugins = router

allow_overlapping_ips = True

notify_nova_on_port_status_changes = True

notify_nova_on_port_data_changes = True

En la sección **[DEFAULT]** añadimos los datos de conexión con la base de datos:

[database]

connection = mysql+pymysql://neutron:**NEUTRON_DBPASS**@controller/neutron

En la sección **[oslo_messaging_rabbit]** configuramos los parámetros de rabbit.:

rabbit_host = controller

rabbit_userid = openstack

rabbit_password = **RABBIT_PASS**

En la sección **[keystone_authtoken]** integramos a neutron con keystone.

[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = **NEUTRON_PASS**

En la sección **[nova]** integramos a nova para ser notificado de cambios en la topología de red.

[nova]

auth_url = http://controller:35357
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = **NOVA_PASS**

Configuración Plugins (nodo controller)

Configuración del Modulo Layer 2 (ML2)

ML2 El plug-in utiliza el mecanismo de puente de Linux para construir la infraestructura de capa 2

(puenteo y conmutación) para el establecimiento de una red instancias virtuales .

Editamos el fichero /etc/neutron/plugins/ml2/ml2_conf.ini

En la sección **[ml2]** especificamos las siguientes opciones de configuración.

[ml2]
type_drivers* = flat , vxlan

(* En esta opción le indicamos los tipos de drivers que posteriormente seran lanzados desde el espacio de nombres `neutron.ml2.type_drivers`)

tenant_network_types* = vxlan

(* En esta opción indicamos que se habilite vxlan para la red self-service)

mechanism_drivers* = linuxbridge,l2population

(* En esta opción le indicamos el mecanismo de drivers ha usar que serán lanzados desde el espacio de nombres `neutron.ml2.mechanism_drivers`)

extension_drivers = port_security

En la sección **[ml2_type_flat]** especificamos las siguientes opciones de configuración.

[ml2_type_flat]

flat_networks = provider

En la sección **[ml2_type_vxlan]** especificamos el rango de redes para la red self-service. Es decir el numero de subredes posibles.

[ml2_type_vxlan]

vni_ranges = 1:1000

En la sección **[securitygroup]** habilitamos la opción **ipset.***

[securitygroup]

enable_ipset = True

ipset*
Se trata de una extensión de iptables que permite la creación de reglas de firewall que responden a "conjuntos " enteros de direcciones IP de forma simultánea . Estos conjuntos residen en las estructuras de datos indexados para aumentar la eficiencia , sobre todo en sistemas con una gran cantidad de reglas .

Configuración del agente Linux Bridge

Editamos el fichero **/etc/neutron/plugins/ml2/linuxbridge_agent.ini**

En la sección **[linux_bridge]** se realiza un mapeo entre la interfaz física eth1 que proveerá de acceso al exterior.

[linux_bridge]

physical_interface_mappings = provider:**PROVIDER_INTERFACE_NAME** (eth1)

En la sección [vxlan] habilitamos las redes vxlan y ademas indicarle la interfaz encargada de gestionarlas

[vxlan]

enable_vxlan = True
local_ip = **OVERLAY_INTERFACE_IP_ADDRESS** (10.0.0.11)
l2_population = True

En la sección **[securitygroup]** habilitaremos los grupos de seguridad (iptables).

[securitygroup]

enable_security_group = True
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver

Configuración del agente layer-3

El Layer-3 (L3) es el agente encargado de proporcionar servicios de enrutamiento y NAT para las redes virtuales self-service.

Editamos el fichero **/etc/neutron/l3_agent.ini**

En la sección **[DEFAULT]** configuramos el driver de linux_bridge junto con el puente a la red externa

[DEFAULT]

interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
external_network_bridge = (esta opción se establece sin valor, para permitir multiples conexiones a la red externa por un solo agente)

Configuración del agente DHCP

Este agente es el encargado de proover de un servidor DHCP para las redes virtuales.

Editamos el fichero de configuración **/etc/neutron/dhcp_agent.ini**

En la sección **[DEFAULT]** configuramos el controlador de interfaz de puente de Linux. el driver de Dnsmasq DHCP.y habilitar la opción enable_isolated_metadata la cual permite a las instancias acceder al servidor de metadatos mediante la red.

[DEFAULT]

interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver

dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq

enable_isolated_metadata = True

Configuración servidor de metadatos

El servidor de metadatos es el encargado de proveer información a las instancias.

Editamos el fichero **/etc/neutron/metadata_agent.ini**

En la sección **[DEFAULT]** configuramos el servidor de metadatos.

[DEFAULT]

nova_metadata_ip = controller
metadata_proxy_shared_secret = **METADATA_SECRET**

(*Replazar **METADATA_SECRET** por un valor adecuado).

Integración de Nova con Neutron

Es necesario configurar nova para integrarlo con neutron. Para ello realizamos lo siguiente:

Editamos el fichero **/etc/nova/nova.conf**

En la sección **[neutron]** añadimos la siguiente configuración.

[neutron]

url = http://controller:9696
auth_url = http://controller:35357
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = **NEUTRON_PASS**

service_metadata_proxy = True
metadata_proxy_shared_secret = **METADATA_SECRET**

Nodo compute

Instalación y configuración de los componentes

root@compute# apt-get install neutron-linuxbridge-agent

Editamos el fichero de configuración **/etc/neutron/neutron.conf**

En la sección **[DEFAULT]** configuramos las siguientes directivas.

[DEFAULT]

rpc_backend = rabbit

auth_strategy = keystone

En la sección **[oslo_messaging_rabbit]** configuramos los parámetros de rabbit.

[oslo_messaging_rabbit]

rabbit_host = controller
rabbit_userid = openstack
rabbit_password = **RABBIT_PASS**

En la sección **[keystone_authtoken]** configuramos la integración de neutron con keystone.

[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = **NEUTRON_PASS**

Debemos de configurar en el nodo de computo a neutron

Editamos el fichero **/etc/nova/nova.conf**

En la sección **[neutron]** configuramos los parámetros:

[neutron]

url = http://controller:9696
auth_url = http://controller:35357
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = **NEUTRON_PASS**

Configuración Plugins (nodo computo)

Configuración del agente Linux Bridge

Editamos el fichero de configuración **etc/neutron/plugins/ml2/linuxbridge_agent.ini**

En la sección **[linux_bridge]** se realiza un mapeo entre la interfaz física eth1 que proveerá de acceso al exterior.

[linux_bridge]
physical_interface_mappings = provider:**PROVIDER_INTERFACE_NAME** (eth1)

En la sección [vxlan] habilitamos las redes vxlan y ademas indicarle la interfaz encargada de gestionarlas

[vxlan]
enable_vxlan = True
local_ip = **OVERLAY_INTERFACE_IP_ADDRESS** (10.0.0.31)
l2_population = True

En la sección **[securitygroup]** habilitaremos los grupos de seguridad (iptables).

[securitygroup]

enable_security_group = True
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver

Por ultimo reiniciamos los servicios en el nodo de computo.

root@compute# service nova-compute restart

root@compute# service neutron-linuxbridge-agent restart

Con esto ya tendriamos realizadas todas las configuraciones necesarias de neutron, tanto en nuestro nodo controller como nuestro nodo compute. Ahora poblaremos la base de datos de neutron.

root@controller# su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.conf --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head" neutron

Y por ultimo reiniciamos los servicios.

root@controller# service nova-api restart

root@controller# service neutron-server restart

root@controller# service neutron-linuxbridge-agent restart

root@controller# service neutron-dhcp-agent restart

root@controller# service neutron-metadata-agent restart

root@controller# service neutron-l3-agent restart

Para verificar que todo funciona de manera correcta, cargamos las variables de admin, y ejecutamos el siguiente comando.

root@controller# neutron agent-list

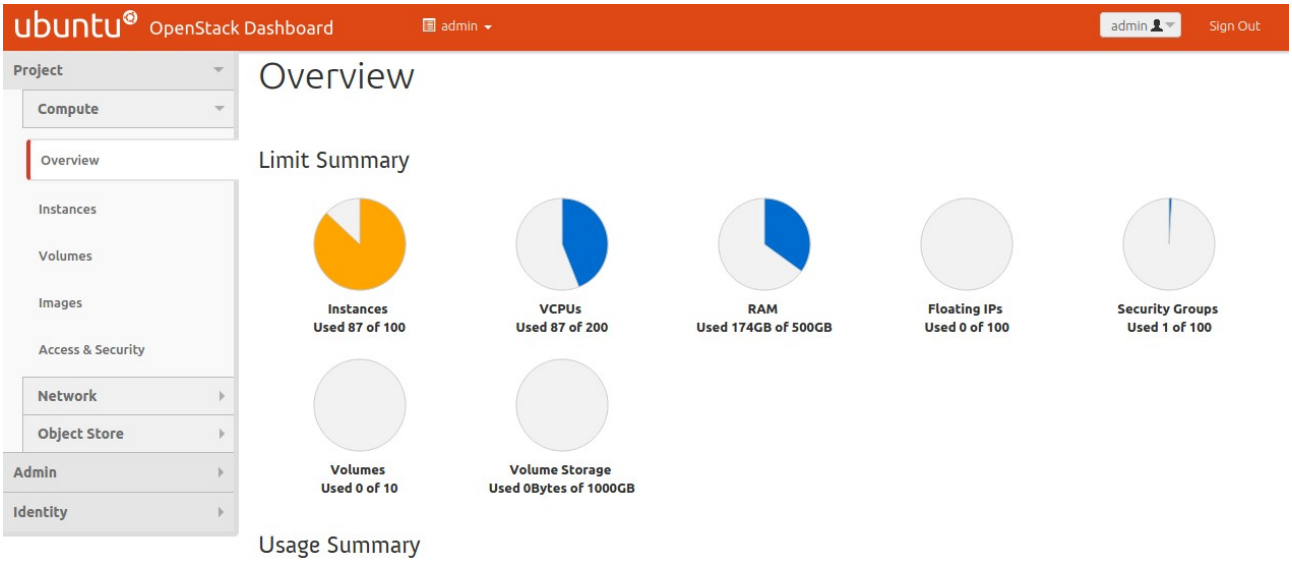
```
+-------+---------------+--------------------------+-------------------+------

| id                                   | agent_type         | host       | alive
+--------------------------------------+--------------------+------------+------
+-------+---------------+--------------------------+
| 5892e9ffga746-447c-br71-411e898f6a92 | Linux bridge agent | compute    | :-)
| 12wyh952-a748-456b-bf71-p41e22246a92 | Linux bridge agent | controller | :-)
| 83199ff-zz36-2456-84f4-067af123a0dc4 | L3 agent           | controller | :-)
| cc3663c9-3dsa-0087a-9282-66c8f051709 | DHCP agent         | controller | :-)
| p789sd81-afd6-4b3d-b923-e22240517099 | Metadata agent     | controller | :-)
+--------------------------------------+--------------------+------------+----
+-------+---------------+--------------------------+------+------------+----
```

Panel Web (Horizon)

Horizon es uno de los componentes principales de OpenStack este con ofrece una interfaz web con la cual interactuar con los componentes de openstack de nuestra infraestructura. Horizon esta desarrollado en django. estas son algunas de las características principales de horizon.

Horizon es el panel de control web (dashboard) de OpenStack

- Es una aplicación web desarrollada en Django
- Implementa las funcionalidades básicas de los componentes principales de OpenStack: Nova, Glance, Swift, etc.
- Ideal para que usuarios noveles utilicen OpenStack
- Como todos los componentes de OpenStack está sometido a un fuerte desarrollo, por lo que cambia bastante con cada versión.



Instalación de Horizon

Horizon normalmente es instalado en el nodo controller.

root@controller# apt-get install openstack-dashboard

Cuando termine la instalación deberemos llevar a cabo una serie de configuraciones:

Editamos el fichero de configuración **/etc/openstack-dashboard/local_settings.py**

- Configuramos horizon para que se ejecute en el nodo controller.

  ```
  OPENSTACK_HOST = "controller"
  ```

- Configuramos horizon para que permita a los hosts acceder a el.

  ```
  ALLOWED_HOSTS = ['*', ]
  ```

- Configuramos memcached para que almacene en cache las sesiones.

  ```
  SESSION_ENGINE = 'django.contrib.sessions.backends.cache'

  CACHES = {
        'default': {
              'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
              'LOCATION': 'controller:11211',
        }
   }
  ```

- Habilitamos keystone para todos los dominios.

  ```
  OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
  ```

- Configuramos las versiones de las APIS.

  ```
  OPENSTACK_API_VERSIONS = {
     "identity": 3,
     "image": 2,
     "volume": 2,
  }
  ```

- Configurar como dominio predeterminado default

  ```
  OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "default"
  ```

- Configurar que el rol por defecto mediante horizon para nuevos usuarios sea user.

  OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"

- Opcionalmente podemos modificar la zona horaria.

  TIME_ZONE = "TIME_ZONE"

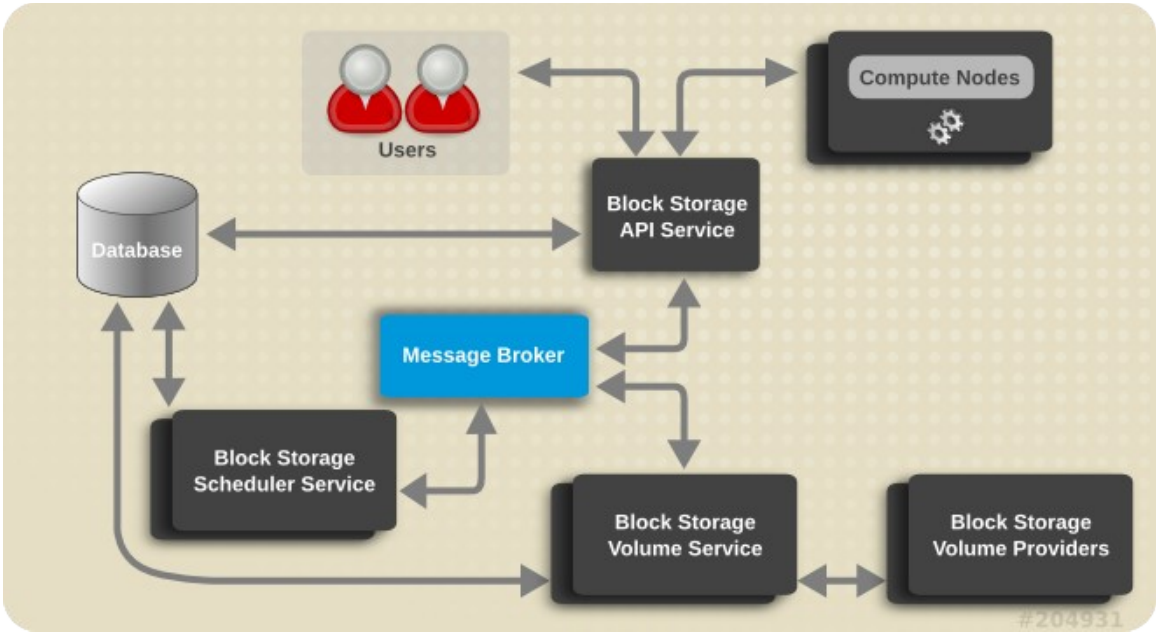
Para finalizar reiniciamos el servicio de apache2..

```
root@controller#  service apache2 reload
```

Para acceder a nuestro panel Web accedemos a **`http://controller/horizon.`**

Servicio de Almacenamiento de Bloques (Cinder)

OpenStack Block Storage (Cinder) proporciona dispositivos de almacenamiento a nivel de bloque persistentes para usar con instancias de OpenStack Compute. El sistema de almacenamiento de bloques gestiona la creación, aplicación y el desprendimiento de los dispositivos de bloque a los servidores. Volúmenes de almacenamiento de bloque se integran plenamente en OpenStack Compute y el Dashboard que permite a los usuarios en la nube gestionar sus propias necesidades de almacenamiento. Además del almacenamiento del servidor local de Linux, puede utilizar las plataformas de almacenamiento incluyendo Ceph, CloudByte, Coraid, EMC (VMAX y VNX), GlusterFS, Hitachi Data Systems, IBM Storage (familia Storwize, controlador de volumen SAN, XIV Storage System, y GPFS) , Linux LIO, NetApp, Nexenta, Scality, SolidFire, HP (StoreVirtual y 3PAR StoreServ familias) y almacenamiento puro. La gestión Snapshot ofrece una potente funcionalidad para realizar copias de seguridad de los datos guardados en volúmenes de almacenamiento en bloque.



Instalación de Cinder

De nuevo y al igual que con los demás componentes es necesario crear una base de datos para cinder y registrarlo como servicio en keystone, endpoints etc. Cabe destacar que en despliegues más complejos cinder es instalado en un nodo independiente, nodo de almacenamiento. En nuestro entorno será instalado en el nodo controller.

Nodo controller

Creación base de datos.

root@controller# mysql -u root -p

mysql> CREATE DATABASE cinder;

mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' IDENTIFIED BY 'CINDER_DBPASS';

mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' IDENTIFIED BY 'CINDER_DBPASS';

Una vez generada la base de datos deberemos de generar al usuario CINDER, para ello debemos cargar el script de variables anteriormente creado.

```
root@controller#  source admin-openrc
```

Creación Usuario cinder

```
root@controller#  openstack user create --domain default --password-prompt cinder

User Password: ********
Repeat User Password: ********
```

Añadir usuario Cinder rol Admin.

```
root@controller#  openstack role add --project service --user cinder admin
```

Creación Servicio Keystone y Endpoints.

De igual manera que con keystone debemos registrar a cinder como servicio. Cinder tiene una peculiaridad requiere de dos entradas en keystone como servicio. para ello:

```
root@controller# openstack service create --name cinder --description "OpenStack Block Storage" volume

root@controller# openstack service create --name cinderv2 --description "OpenStack Block Storage" volumev2
```

Endpoint public. (v1)

```
root@controller#openstack endpoint create --region RegionOne volume public http://controller:8776/v1/%\(tenant_id\)s
```

Endpoint internal. (v1)

```
root@controller#openstack endpoint create --region RegionOne volume internal http://controller:8776/v1/%\(tenant_id\)s
```

Endpoint admin (v1)

root@controller# openstack endpoint create --region RegionOne volume admin http://controller:8776/v1/%\(tenant_id\)s

Endpoint public. (v2)

root@controller#openstack endpoint create --region RegionOne volumev2 public http://controller:8776/v2/%\(tenant_id\)s

Endpoint internal. (v2)

root@controller#openstack endpoint create --region RegionOne volumev2 internal http://controller:8776/v2/%\(tenant_id\)s

Endpoint admin (v2)

root@controller# openstack endpoint create --region RegionOne volumev2 admin http://controller:8776/v2/%\(tenant_id\)s

Una vez registrado cinder en keystone como servicio etc. podemos llevar a cabo las configuraciones de cinder.

Instalación y configuración de los componentes

root@controller# apt-get install cinder-api cinder-scheduler

Editamos /etc/cinder/cinder.conf

En la sección **[database]** debemos configurar el acceso a la base de datos previamente creada.

[database]

connection = mysql+pymysql://cinder:**CINDER_DBPASS**@controller/cinder

En la sección **[DEFAULT]** añadimos la siguiente configuración:

[DEFAULT]

rpc_backend = rabbit

auth_strategy = keystone

my_ip = 10.0.0.11

En la sección **[oslo_messaging_rabbit]** configuramos los parámetros de rabbit.:

rabbit_host = controller

rabbit_userid = openstack

rabbit_password = **RABBIT_PASS**

En la sección **[keystone_authtoken]** integramos a nova con keystone.

[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = **CINDER_PASS**

En la sección **[oslo_concurrency]** configuramos el loch path.

[oslo_concurrency]

lock_path = /var/lib/cinder/tmp

Ya tenemos configurado cinder en el nodo controller. lo siguiente que vamos a hacer es poblar la base de datos.

root@controller# su -s /bin/sh -c "cinder-manage db sync" cinder

Integración de cinder con nova (nodo controller)

En nuestro nodo controlador editamos el fichero de nova **/etc/nova/nova.conf**

En la sección **[cinder]** añadimos la siguiente configuración.

```
[cinder]
os_region_name = RegionOne
```

Para terminar reiniciamos los servicios implicados en el nodo controller.

```
root@controller# service nova-api restart
root@controller# service cinder-api restart
root@controller# service cinder-scheduler restart
```

Nodo compute

Como comentaba antes normalmente cinder se instala en un nodo de almacenamiento, como en nuestro caso nuestro entorno esta compuesta por dos nodos las instalaciones que ahora se mostrarán se harán en el nodo compute

```
root@compute# apt-get install lvm2
```

Una vez instalado generaremos un grupo de volúmenes con un disco en mi caso **/dev/sdb (con un tamaño de 15GB)** previamente deberemos crear un volumen fisico con eso disco.

```
root@controller# pvcreate /dev/sdb
Physical volume "/dev/sdb" successfully created
```

Ahora generamos el grupo de volumenes llamado "cinder-volumes".

```
root@controller# vgcreate cinder-volumes /dev/sdb
Volume group "cinder-volumes" successfully created
```

Cinder utilizara este grupo de volúmenes para generar volúmenes lógicos los cuales seran asociados a instancias.

Para asegurarnos de que solo las instancias puedan tener acceso a este grupo de volúmenes y sus posteriores volúmenes lógicos, para evitar inconcluencias. Para ello editamos el fichero de configuración de LVM **/etc/lvm/lvm.conf** en el cual indicaremos que permita el acceso a nuestro grupo de volúmenes y que rechace a los demás dispositivos.

```
devices {
filter = [ "a/sdb/", "r/.*/"]
```

Instalación de los componentes

Instalamos los paquetes necesarios:

```
root@compute# apt-get install cinder-volume
```

Una vez instalado editamos el fichero **/etc/cinder/cinder.conf**

En la seccion **[database]** realizamos la conexión con la base de datos.

```
[database]
connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
```

En la seccion **[DEFAULT]** añadimos las siguientes configuraciones.

```
[DEFAULT]
rpc_backend = rabbit
auth_strategy = keystone
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS (10.0.0.31)
enabled_backends = lvm
glance_api_servers = http://controller:9292
```

En la sección **[oslo_messaging_rabbit]** añadimos los parámetros de rabbit.

```
[oslo_messaging_rabbit]

rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS
```

En la seccion **[keystone_authtoken]** integramos a cinder con keystone.

```
[keystone_authtoken]

auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
```

En la sección **[lvm]** configuraremos el backend indicandole que sea lvm y el protocolo ISCSI.

```
[lvm]

volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
iscsi_protocol = iscsi
iscsi_helper = tgtadm
```

En la sección **[oslo_concurrency]** configuramos el lock_path.

```
[oslo_concurrency]

lock_path = /var/lib/cinder/tmp
```

Por ultimo reiniciamos los servicios implicados.

```
root@compute# service tgt restart

root@compute# service cinder-volume restart
```

Para verificar que todo funciona de manera correcta, cargamos las variables.

root@controller# source admin-openrc

root@compute# cinder service-list

```
+------------------+------------+------+---------+-------+----------------------------+----+

|      Binary      |    Host    | Zone |  Status | State |
+------------------+------------+------+---------+-------|
| cinder-scheduler | controller | nova | enabled |   up  |
| cinder-volume    | compute    | nova | enabled |   up  |
+------------------+------------+------+---------+-------+
```

Creación Redes Virtuales

Con esto ya tendríamos instalados todos lo componentes necesarios para una instalación básica, ahora es el momento de generar las redes iniciales, en función a nuestro escenario de red elegido sera necesario la creación de dos redes la red PROVIDER y la red SELF-SERVICE

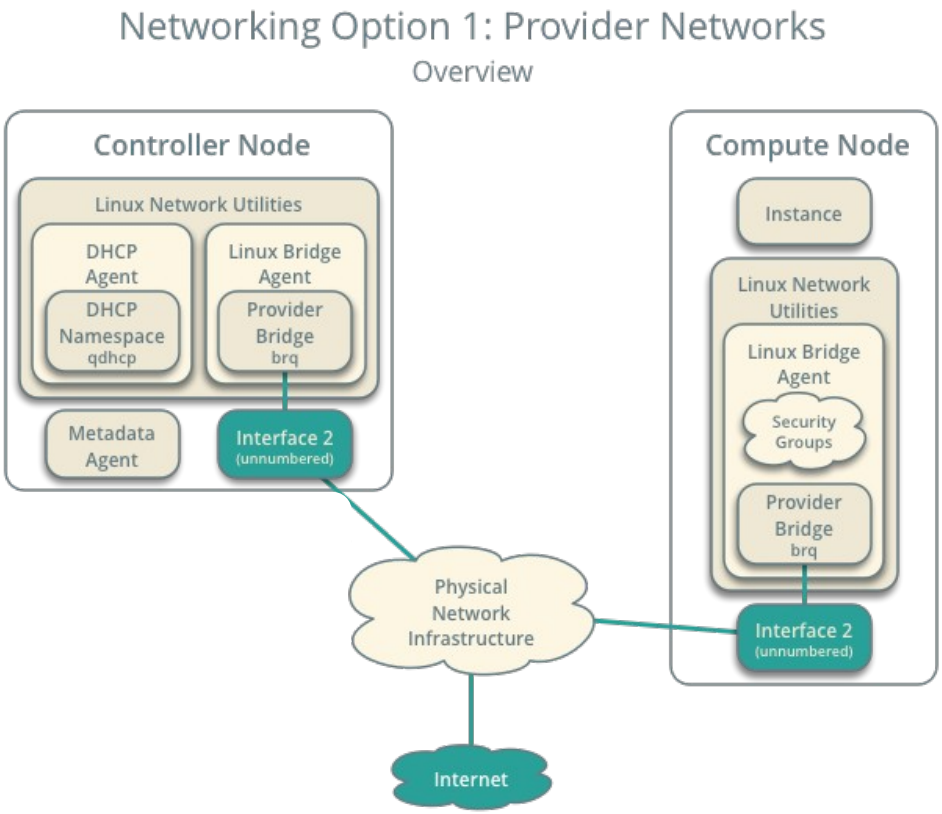
Creación de la red Provider

Cargamos las variables de admin, ya que esta red solo puede ser gestionada por el usuario admin.

root@controller# source admin-openrc

root@controller# neutron net-create --shared --provider:physical_network provider --provider:network_type flat provider

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 5tthefcd-8zxe-12w4-c118-d10f06t9c5ad |
| mtu                       | 1500                                 |
| name                      | provider                             |
| port_security_enabled     | True                                 |
| provider:network_type     | flat                                 |
| provider:physical_network | provider                             |
| provider:segmentation_id  |                                      |
| router:external           | False                                |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 425b2f6e18d0927390425b2f6e18d092     |
+---------------------------+--------------------------------------+
```



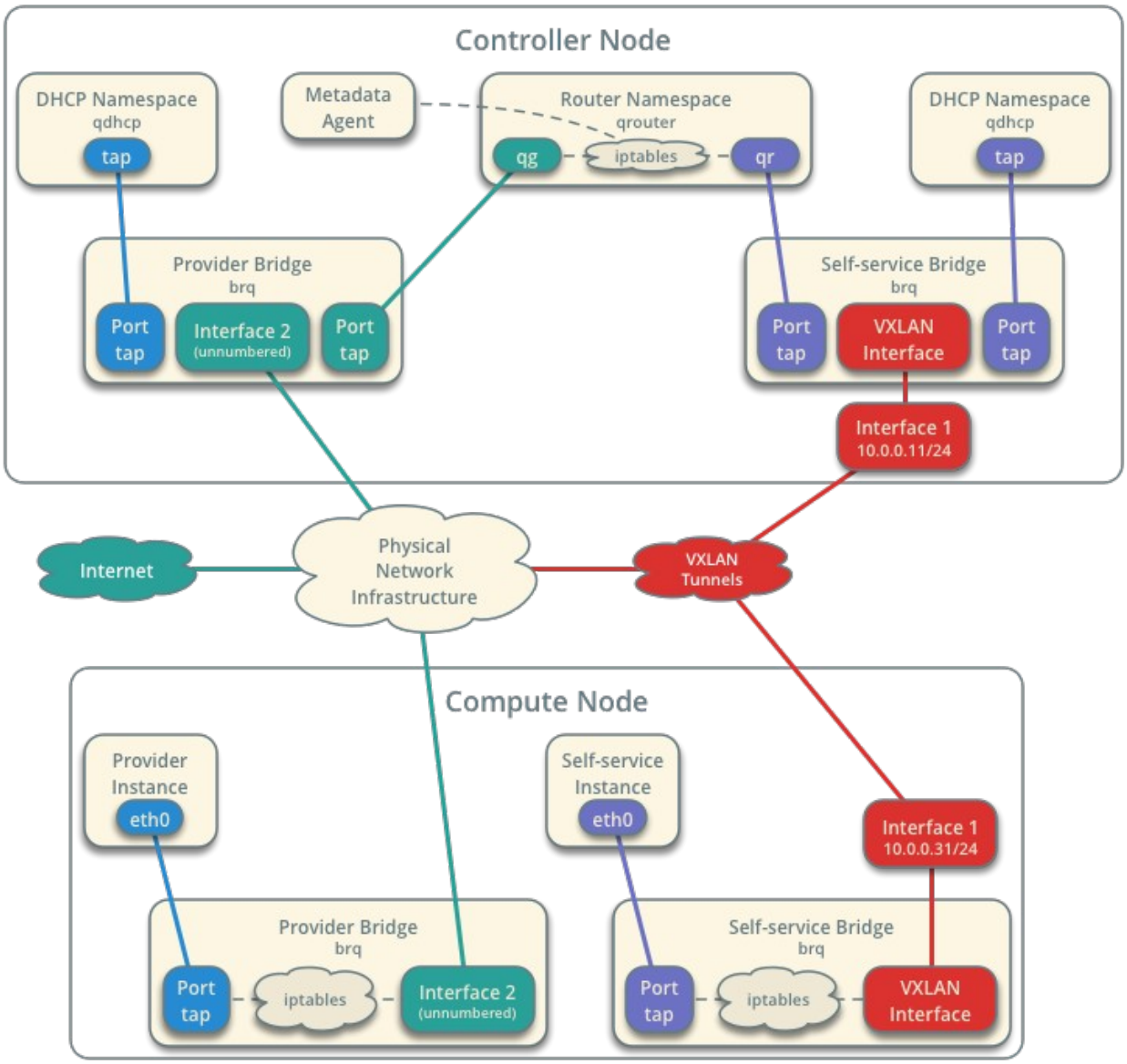
Creación de la subred Provider

root@controller#neutron subnet-create --name provider --allocation-pool start=192.168.1.50,end=192.168.1.55 --dns-nameserver 8.8.4.4 --gateway 192.168.1.1 provider 192.168.1.0/24

```
+-------------------+-----------------------------------------------------+
| Field             | Value                                               |
+-------------------+-----------------------------------------------------+
| allocation_pools  | {"start": "192.168.1.50", "end": "192.168.1.55"}    |
| cidr              | 192.168.1.0/24                                      |
| dns_nameservers   | 8.8.4.4                                             |
| enable_dhcp       | True                                                |
| gateway_ip        | 192.168.1.1                                         |
| host_routes       |                                                     |
| id                | 4dd72sa8-132b-2514-de52-c4c012ds2ft6                |
| ip_version        | 4                                                   |
| ipv6_address_mode |                                                     |
| ipv6_ra_mode      |                                                     |
| name              | provider                                            |
| network_id        | 0e62efcd-8cee-46c7-b163-d8df05c3c5ad                |
| subnetpool_id     |                                                     |
| tenant_id         | v6f7hh8sc658iryhdf347bd9f62023fn                    |
+-------------------+-----------------------------------------------------+
```

Networking Option 2: Self-service Networks

Connectivity



Creación de la red Self-service

Esta red sera generada para el proyecto demo. Cargamos las sources de demo.

root@controller# source demo-openrc

root@controller# neutron net-create selfservice

```
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | True                                 |
| id                      | 3482f524-8bff-4871-80d4-5774c2730728 |
| mtu                     | 0                                    |
| name                    | selfservice                          |
| port_security_enabled | True                                 |
| router:external         | False                                |
| shared                  | False                                |
| status                  | ACTIVE                               |
| subnets                 |                                      |
| tenant_id               | f5b2ccaa75ac413591f12fcaa096aa5c     |
+-------------------------+--------------------------------------+
```

Creación de la subred Self-service

root@controller# neutron subnet-create --name selfservice --dns-nameserver 8.8.4.4 --gateway 172.16.1.1 selfservice 172.16.1.0/24

```
+-------------------+------------------------------------------------+

| Field             | Value                                          |
+-------------------+------------------------------------------------+
| allocation_pools  | {"start": "172.16.1.2", "end": "172.16.1.254"} |
| cidr              | 172.16.1.0/24                                  |
| dns_nameservers   | 8.8.4.4                                        |
| enable_dhcp       | True                                           |
| gateway_ip        | 172.16.1.1                                     |
| host_routes       |                                                |
| id                | 3482f524-8bff-4871-80d4-5774c2730728           |
| ip_version        | 4                                              |
| ipv6_address_mode |                                                |
| ipv6_ra_mode      |                                                |
| name              | selfservice                                    |
| network_id        | 7c6f9b37-76b4-463e-98d8-27e5686ed083           |
| subnetpool_id     |                                                |
| tenant_id         | f5b2ccaa75ac413591f12fcaa096aa5c               |
+-------------------+------------------------------------------------+
```

Creación Router

La red self-service se interconecta con la red provider mediante un router virtual el cual realiza nat bidireccional.

root@controller# source admin-openrc

Ahora conectamos el router con la red provider.

root@controller# neutron net-update provider --router:external

Updated network: provider

Ahora conectamos de nuevo con las sources de demo generaremos el router

root@controller# neutron router-create router

```
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| external_gateway_info |                                      |
| id                    | 89dd2083-a160-4d75-ab3a-14239f01ea0b |
| name                  | router                               |
| routes                |                                      |
| status                | ACTIVE                               |
| tenant_id             | f5b2ccaa75ac413591f12fcaa096aa5c     |
+-----------------------+--------------------------------------+
```

Ahora añadimos la subnet self-service a una de las interfaces del router virtual.

root@controller# neutron router-interface-add router selfservice

Por ultimo en el router indicamos como puerta de enlace la de la red provider.

root@controller# neutron router-gateway-set router provider

Con esto ya tendriamos las redes inciales completamente configuradas. por ultimo vamos a instanciar una maquina cirros para verificar la operación.

Instanciación Maquina Cirros

Vamos a llevar a cabo el proceso de instanciación.

Lo primero seria descargarnos una imagen cirros, para ello:

root@controller# source admin-openrc

root@controller# wget http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img

Una vez descargada añadiremos esta imagen al servicio de imagenes Glance, con las siguientes caracteristicas, formato de imagen **QEMU Copy On Write 2 (QCOW2)**, contenedor **BARE** y la visibilidad pública para que sea visiable para todos los proyectos.

root@controller# openstack image create "cirros" --file cirros-0.3.4-x86_64-disk.img --disk-format qcow2 --container-format bare --public

```
+------------------+------------------------------------------------------+
| Property         | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | 133eae9fb1c98f45894a4e60d8736619                     |
| container_format | bare                                                 |
| created_at       | 2015-03-26T16:52:10Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/cc5c6982-4910-471e-b864-1098015901b5/file |
| id               | cc5c6982-4910-471e-b864-1098015901b5                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | cirros                                               |
| owner            | ae7a98326b9c455588edd2656d723b9d                     |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
| size             | 13200896                                             |
| status           | active                                               |
| tags             |                                                      |
| updated_at       | 2015-03-26T16:52:10Z                                 |
| virtual_size     | None                                                 |
| visibility       | public                                               |
+------------------+------------------------------------------------------+
```

Generamos un par de claves

root@controller# ssh-keygen -q -N ""

root@controller# openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey

root@controller# openstack keypair list

```
+-------+-------------------------------------------------+

| Name  | Fingerprint                                     |
+-------+-------------------------------------------------+
| mykey | a4:f7:gg:de:23:5t:pp:l0:a1:ss:c3:ff:z1:k7:34:cd |
+-------+-------------------------------------------------+
```

Generamos reglas de seguridad. (PING y SSH)

root@controller# openstack security group rule create --proto icmp default

root@controller# openstack security group rule create --proto tcp --dst-port 22 default

Con esto ya tendríamos todos los requisitos necesarios para instanciar.

Instanciación

root@controller# openstack server create --flavor **m1.tiny** --image **cirros** --nic net-id=**SELFSERVICE_NET_ID** --security-group **default** --key-name **mykey** **instancia**

root@controller# openstack server list

```
| ID                                   | Name                 | Status | Networks                |
+--------------------------------------+----------------------+--------+-------------------------+
| 113c5892-e58e-4093-88c7-e33f502eaaa4 | instancia            | ACTIVE | selfservice=172.16.1.3 |
+--------------------------------------+----------------------+--------|-------------------------|
```