Trabajo final

Despliegue automático de una aplicación escalable sobre una nube Openstack utilizando los servicios de orquestación de Heat.

Miguel Marcos García
Jaime Calleja del Castillo
Javier Hernández Sánchez

Contenido

1. Despliegue escenario físico

Durante este trabajo simularemos las principales tareas de automatización y gestión realizadas por los proveedores de infraestructura. Los clientes finales pueden interactuar y desplegar sus servicios sobre la infraestructura física de los proveedores mediante portales web. Estos portales son extremadamente intuitivos y sencillos de usar, pero la realidad es que detrás de estos portales se esconden complejas tareas de automatización por parte de los proveedores.

Para simular la infraestructura de nuestro proveedor cloud utilizaremos el siguiente escenario, disponible en la **Figura 1**. Además instalaremos todas las herramientas OpenStack para gestionar nuestra infraestructura física.

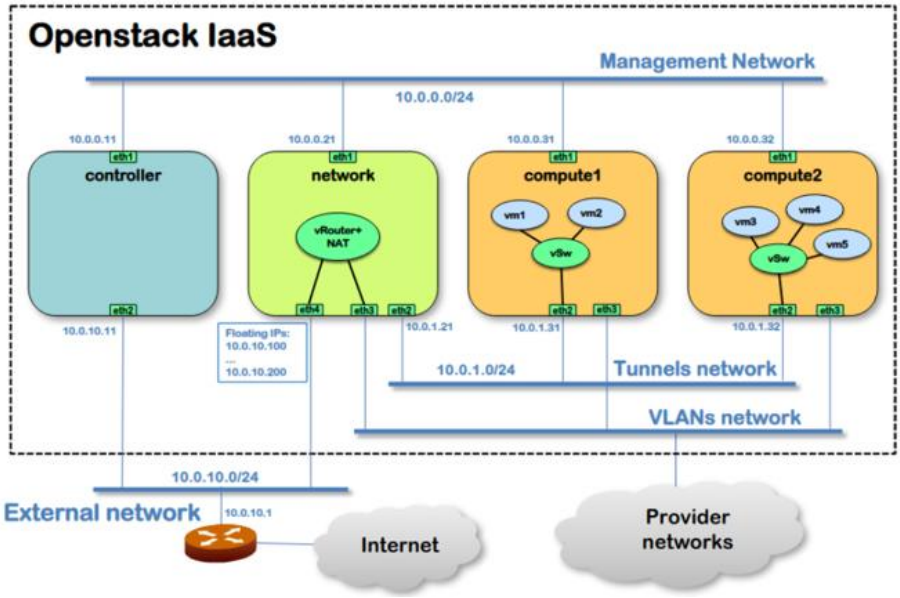

Figura 1 - Escenario físico Openstack del proveedor de infraestructura

Para desplegar el escenario físico del proveedor cloud y los componentes OpenStack ejecutaremos los siguientes comandos de la **Figura 2**. VNX es una herramienta de despliegue de redes virtuales sobre Linux, desarrollada por el departamento de sistemas telemáticos (DIT) de la Universidad Politécnica de Madrid. Mediante esta herramienta desplegaremos el escenario físico del proveedor sobre un único computador Linux.

```
/lab/cnvr/bin/get-openstack-tutorial.sh
cd /mnt/tmp/openstack_lab-stein_4n_classic_ovs-v06
sudo vnx -f openstack_lab.xml --create
sudo vnx -f openstack_lab.xml -x start-all,load-img
sudo vnx_config_nat ExtNet enp2s0
```

Figura 2 - Despliegue de infraestructura física

Para automatizar y facilitar los despliegues de ahora en adelante emplearemos shell scripts de Linux. Para ejecutar los comandos VNX de la **Figura 2** emplearemos el script *cloud.sh* de la **Figura 3**.

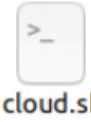

Figura 3 – Script de despliegue escenario físico

2. Configuración inicial de Openstack

Tras desplegar el escenario físico, el proveedor ya podría ofrecer sus recursos de red y computación. Para ello, deberá seguir los pasos de la **Figura 4**.

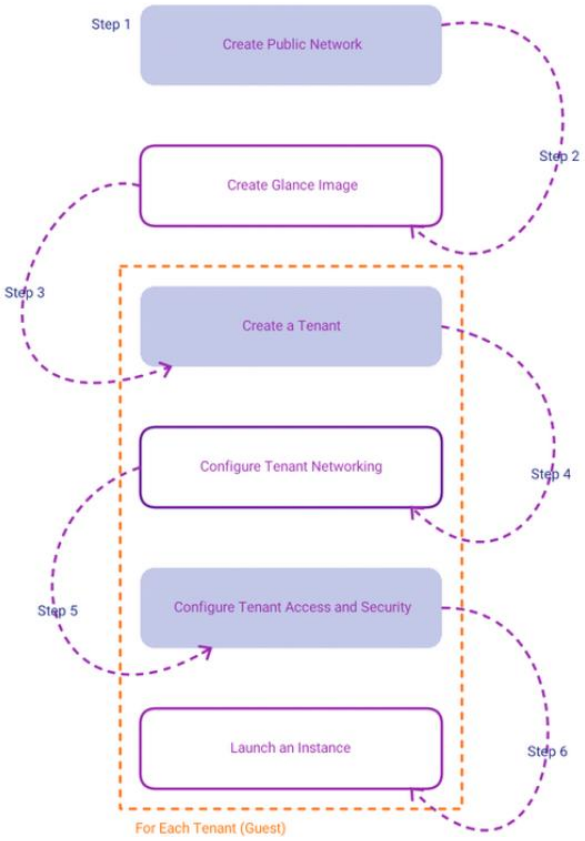

Figura 4 - Tareas a realizar por el proveedor de infraestructura.

En primer lugar debemos crear la red pública de acceso sobre Openstack. Después, cargaremos en Glance las imágenes que queremos ofrecer a nuestros clientes para el despliegue de máquinas virtuales. Estas dos tareas ya han sido realizadas previamente con el shell script *cloud.sh,* por lo que no será necesario ejecutarlas de nuevo.

No obstante, tendremos que cargar una última imagen más. Para ello ejecutaremos *load_image.sh,* script encargado de subir una imagen customizada a Openstack. Esta imagen cuenta con ciertas herramientas preinstaladas por nosotros. Más tarde se verá en detalle cómo realizar dicho proceso, por ahora simplemente cargaremos la imagen en Openstack.

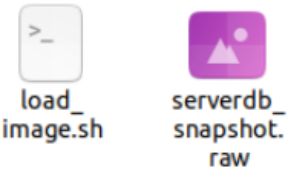

Figura 5 – Script de carga de imágenes

En la **Figura 6** se muestra en detalle el script *load_image.sh*. En primer lugar, será necesario autentificarse como usuario administrador en Openstack. Después con el comando "*openstack image create"* se cargará la imagen en Glance. Tanto la imagen como el script deberán encontrarse en el

mismo directorio para que el proceso de carga funcione correctamente. En la Figura 5 se muestran ambos ficheros, la imagen tiene un formato *qcow2*.

```
export OS_USERNAME=admin
export OS_PASSWORD=xxxx
export OS_PROJECT_NAME=admin
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
export OS_AUTH_TYPE=password

openstack image create --container-format bare --disk-format qcow2 \
  --file serverdb_snapshot.raw ServerDBSnapshot
```

Figura 6 - Script load_image.sh

3. Creación de un entorno de usuario

Tras la puesta en marcha del escenario físico y la realización de los dos primeros pasos de la Figura 4, ya podríamos ofrecer los recursos de red y computación a nuestros clientes. Pero, para facilitar los despliegues de nuestros clientes y reducir costes asociados debemos automatizar cada tarea. Por tanto, para la automatización emplearemos el servicio de orquestación *Heat*.

Dado que Openstack permite la creación de múltiples proyectos y cuentas de usuario, crearemos un entorno virtual por cliente, consiguiendo así un aislamiento completo entre ellos. Para automatizar la creación de nuevos clientes, hemos desarrollado *create_tenant.sh,* script que debemos ejecutar cada vez que queramos instanciar un nuevo cliente en Openstack. Este script tiene como contenido los comandos de la Figura 7. En primer lugar se realizará la autentificación como usuario administrador, para después desplegar un nuevo entorno de usuario mediante un *stack* de *Heat*.

```
#!/bin/sh

#--------------------------------------------------------------------------
# Credenciales admin Openstack

# source /root/bin/admin-openrc.sh # Admin credentials
export OS_USERNAME=admin
export OS_PASSWORD=xxxx
export OS_PROJECT_NAME=admin
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
export OS_AUTH_TYPE=password

admin_project_id=$(openstack project show admin -c id -f value)
default_secgroup_id=$(openstack security group list -f value | grep default
| grep $admin_project_id | cut -d " " -f1)

#--------------------------------------------------------------------------
# Launch HEAT Template for creating a project and its associated user

# Login into the controller or login into an external machine with
OpenStack installed for remote access

# Source as the admin (already done)
# source ~/keystone_admin

# Ensure we have copied the HOT (Heat Stack) and the environment file into
the current directory of the controller or remote machine
openstack stack create -t create-tenant.yaml -e create-tenant-env.yaml TNT00
```

Figura 7 - Script create_tenant.sh

Con los archivos de la **Figura 8** Heat automatizará la creación de nuevos clientes. Los archivos *create-tenant.yaml* y *create-tenant-env.yaml* corresponden a la plantilla y variables de entrada empleadas por *Heat*. Para instanciar nuevos clientes tan sólo debemos cambiar las variables de entrada del fichero *create-tenant-env.yaml* con la información del nuevo cliente y ejecutar el script *create_tenant.sh*.

Figura 8 - Heat templates empleados para la creación de nuevos clientes

La plantilla Heat empleada tiene el aspecto de la **Figura 9**. En ella creamos los siguientes recursos, un nuevo proyecto, un usuario administrador asociado al proyecto y dos roles de administrador tanto para el nuevo usuario, como para la cuenta admin de Openstack.

```
heat_template_version: 2016-04-08

description: Tenant init for GBM Cloud

parameters:
  project_name:
    type: string
    description: Name of Project (Tenant)
  admin_name:
    type: string
    description: Username of Project Admin
  admin_password:
    type: string
    description: Password for Project Admin
    hidden: true

resources:
  tenant:
    type: OS::Keystone::Project
    properties:
      name: { get_param: project_name }
      domain: default
  tenant_admin:
    type: OS::Keystone::User
    properties:
      name: { get_param: admin_name }
      password: { get_param: admin_password }
  tenant_admin_role:
    type: OS::Keystone::UserRoleAssignment
    properties:
      user: { get_resource: tenant_admin }
      roles:
        - {project: {get_resource: tenant}, role: admin}
  default_admin_role:
    type: OS::Keystone::UserRoleAssignment
    properties:
      user: admin
      roles:
        - {project: {get_resource: tenant}, role: admin}
```

Figura 9 - Heat template para la creación de clientes

Tras ejecutar *create_tenant.sh* se despliega el stack de Heat mostrado en la **Figura 10**. Con él se crea un nuevo proyecto ***TNT00*** con un usuario administrador asociado, ***tnt00user***. En la siguiente imagen se muestra el stack desplegado; además, en la barra de navegación superior aparece el nombre del nuevo proyecto y del nuevo usuario.

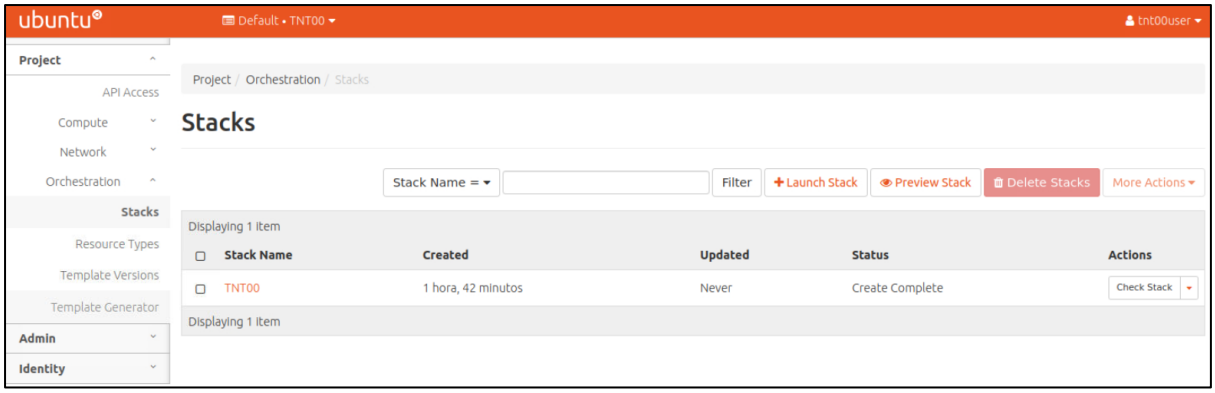

Figura 10 - Tenant TNT00

4. Despliegue del escenario virtual del cliente

De ahora en adelante, emplearemos este cliente o tenant para el resto de despliegues. El escenario virtual que vamos a desplegar en TNT00 es el escenario de la **Figura 11**.

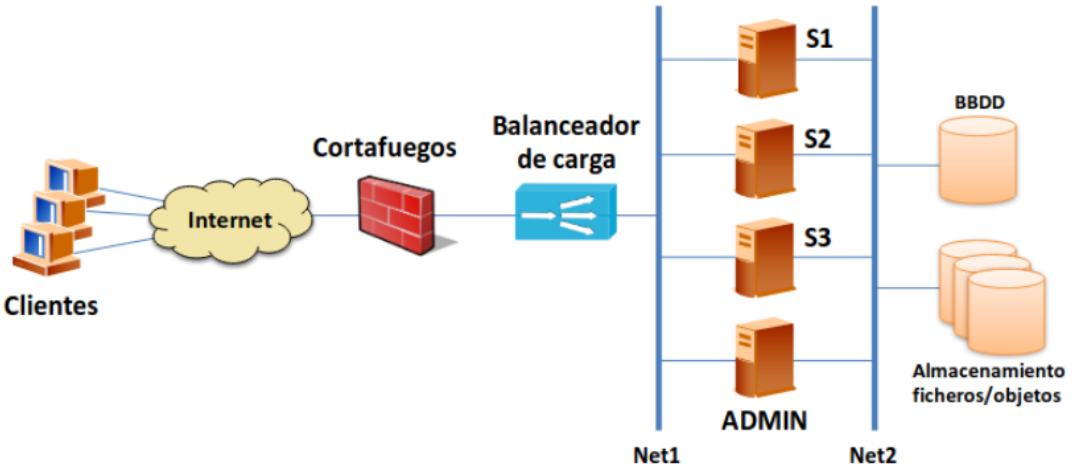

Figura 11 - Escenario virtual del cliente TNT00

El escenario virtual estará compuesto de un balanceador que distribuirá la carga entre tres servidores web situados en Net1. Además habrá un router que comunicará nuestro balanceador con el exterior. Este router tendrá una serie de reglas de seguridad instaladas actuando como cortafuegos. En Net1 también habrá un servidor de administración que será accesible desde el exterior mediante ssh. Por último aparece Net2, en la cual estarán instalados los servidores de base de datos y almacenamiento. Necesitaremos dos IPs flotantes, una para el balanceador de carga y otra para el servidor de administración para facilitar el acceso desde el exterior. El escenario de la **Figura 12** es la topología real del escenario que desplegaremos sobre Openstack.

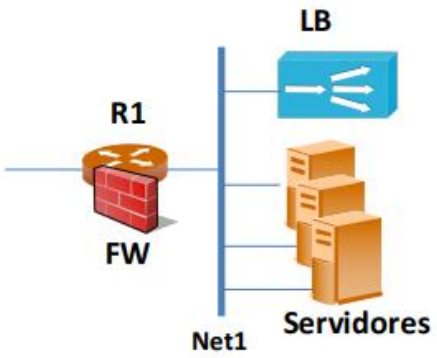

Figura 12 - Implementación del escenario virtual en Openstack

En primer lugar, con otro stack de Heat lanzaremos las máquinas virtuales, las redes Net1 y Net2, el router y el resto de recursos Openstack. Para lanzar el escenario virtual bastaría con ejecutar el script de la **Figura 13**, *init_tenant.sh*. Este script desplegará un nuevo *stack* de *Heat* que arrancará todo por nosotros.

Figura 13 – Ficheros de inicialización del escenario virtual I

En la **Figura 14** se muestra *init_tenant.sh*. Como se puede observar es un script bastante similar a *create_tenant.sh*, pero esta vez en las credenciales introducimos el nuevo proyecto y usuario TNT00.

```
#!/bin/sh

#--------------------------------------------------------------------------------
# Credenciales admin TNT00 Project

# source /root/bin/admin-openrc.sh # Admin credentials
export OS_USERNAME=tnt00user
export OS_PASSWORD=password
export OS_PROJECT_NAME=TNT00
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
export OS_AUTH_TYPE=password

TNT00_project_id=$(openstack project show TNT00 -c id -f value)
default_secgroup_id=$(openstack security group list -f value | grep default
| grep $TNT00_project_id | cut -d " " -f1)

#--------------------------------------------------------------------------------
# Launch HEAT Template for initiate the tenant TNT00

# Login into the controller or login into an external machine with
OpenStack installed for remote access

# Ensure we have copied the HOT (Heat Stack) and the environment file into
the current directory of the controller or remote machine
openstack stack create -t init-tenant.yaml -e init-tenant-env.yaml
INIT_TNT00
```

Figura 14 - Script init_tenant.sh

Este stack arrancará el escenario de la **Figura 15** con los tres servidores, el servidor de administración, el servidor de base de datos, las redes Net1 y Net2, y el router que comunicará la red Net1 con el exterior. También se asigna una IP flotante al servidor de administración.

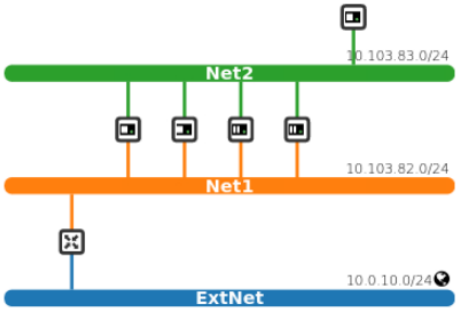

Figura 15 - Escenario virtual

El stack desplegado contiene los parámetros de entrada mostrados en la **Figura 16**. Entre ellos se encuentran los nombres de los distintos recursos y el rango IP de nuestras subredes (CIDR).

```
heat_template_version: 2016-04-08

description: Tenant init for GBM Cloud

parameters:
  private_net1_name:
    type: string
    description: Name of Private Tenant Network
  private_subnet1_name:
    type: string
    description: Name of Private Subnet
  private_subnet1_cidr:
    type: string
    description: CIDR of Private Subnet
  private_subnet1_gateway:
    type: string
    description: Gateway of Private Subnet
  private_net1_pool_start:
    type: string
    description: Start of private network IP address allocation pool
  private_net1_pool_end:
    type: string
    description: End of private network IP address allocation pool
  private_net2_name:
    type: string
    description: Name of Private Tenant Network
  private_subnet2_name:
    type: string
    description: Name of Private Subnet
  private_subnet2_cidr:
    type: string
    description: CIDR of Private Subnet
  private_net2_pool_start:
    type: string
    description: Start of private network IP address allocation pool
  private_net2_pool_end:
    type: string
    description: End of private network IP address allocation pool
  dns:
    type: comma_delimited_list
    label: DNS nameservers
    description: Comma separated list of DNS nameservers
  public_net_name:
    type: string
    description: Name of Private Network
  router_name:
    type: string
    description: Name of Router
  VM1_name:
    type: string
    description: Name of Virtual Machine
  VM2_name:
    type: string
    description: Name of Virtual Machine
  VM3_name:
    type: string
    description: Name of Virtual Machine
  VM4_name:
    type: string
    description: Name of Virtual Machine
  VM5_name:
    type: string
    description: Name of Virtual Machine
  key_name:
    type: string
    description: Name of SSH Key
  image_name:
    type: string
    description: Name of Image to use for the VM instance (Should exist before hand)
  flavor_name:
    type: string
    description: Name of flavor to use for the VM instance (Should exist before hand)
  sec_group_name:
    type: string
    description: Name of Security Group
```

Figura 16 - Parámetros de entrada Heat del escenario

En la **Figura 17**, se muestra la configuración de la red Net1. La red Net2 se configura de forma análoga.

```
resources:
  private_net1:
    type: OS::Neutron::Net
    properties:
      name: { get_param: private_net1_name }
      shared: false
  private_subnet1:
    type: OS::Neutron::Subnet
    depends_on: [private_net1]
    properties:
      name: { get_param: private_subnet1_name }
      network_id: { get_resource: private_net1 }
      cidr: { get_param: private_subnet1_cidr }
      gateway_ip: { get_param: private_subnet1_gateway }
      dns_nameservers: { get_param: dns }
      allocation_pools:
        - start: { get_param: private_net1_pool_start }
          end: { get_param: private_net1_pool_end }
  private_net2:
    type: OS::Neutron::Net
    properties:
      name: { get_param: private_net2_name }
      shared: false
  private_subnet2:
    type: OS::Neutron::Subnet
    depends_on: [private_net2]
    properties:
      name: { get_param: private_subnet2_name }
      network_id: { get_resource: private_net2 }
      cidr: { get_param: private_subnet2_cidr }
      dns_nameservers: { get_param: dns }
      allocation_pools:
        - start: { get_param: private_net2_pool_start }
          end: { get_param: private_net2_pool_end }
```

Figura 17 - Configuración Net1 y Net2

En la **Figura 18** se muestra la configuración del router encargado de comunicar ExtNet y Net1.

```
  router1:
    type: OS::Neutron::Router
    depends_on: [private_net1, private_subnet1]
    properties:
      name: { get_param: router_name }
      external_gateway_info:
        network: { get_param: public_net_name }
  router1_interface1:
    type: OS::Neutron::RouterInterface
    depends_on: [router1, private_subnet1]
    properties:
      router_id: { get_resource: router1 }
      subnet_id: { get_resource: private_subnet1 }
```

Figura 18 - Configuración del router

En la siguiente imagen, **Figura 19**, se muestra la configuración de las claves ssh e IP flotante del servidor de administración. Además configuramos unas reglas de seguridad para todos nuestros servidores. Como se puede observar permitimos tráfico ICMP, ssh y HTTP. Esta serie de reglas de seguridad han sido configuradas para todos los servidores igual, para permitir la realización de pruebas. En producción tendríamos que inhabilitar algunas de ellas dependiendo del tipo de máquina.

```
server_security_group:
  type: OS::Neutron::SecurityGroup
  properties:
    description: Security Group for VM
    name: { get_param: sec_group_name }
    rules:
      - remote_ip_prefix: 0.0.0.0/0
        protocol: tcp
        port_range_min: 22
        port_range_max: 22
      - remote_ip_prefix: 0.0.0.0/0
        protocol: icmp
      - remote_ip_prefix: 0.0.0.0/0
        protocol: tcp
        port_range_min: 80
        port_range_max: 80
server_ssh_key:
  type: OS::Nova::KeyPair
  properties:
    name: { get_param: key_name }
    save_private_key: true
floating_ip:
  depends_on: [private_subnet1,port_vm4net1, router1, router1_interface1, server_admin]
  type: OS::Neutron::FloatingIP
  properties:
    floating_network_id: { get_param: public_net_name }
    port_id: { get_resource: port_vm4net1 }
```

Figura 19 - Configuración de security groups, ssh keys y floating IP

Después configuramos los puertos de nuestras máquinas. En la **Figura 20** tan sólo mostramos la configuración de uno de ellos por simplicidad.

```
port_vm1net1:
  depends_on: [private_subnet1]
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: private_net1}
    security_groups: [{ get_resource: server_security_group }]
    fixed_ips:
      - subnet_id: { get_resource: private_subnet1 }
```

Figura 20 - Configuración de puertos

A continuación se muestra la configuración de los servidores web. Como se puede observar en la **Figura 21**, la máquina obtiene como entrada la imagen y su respectivo *flavor*. Además conectamos también la máquina a las redes *Net1* y *Net2* con sus respectivos puertos. Por último se le proporciona un script cloud-init de arranque a nuestra máquina virtual para la configuración inicial partiendo de la imagen base proporcionada. En el script cloud-init se configura el servidor web y se habilita la segunda interfaz de red *ens4*.

```
server1:
  depends_on: [server_security_group,private_subnet1,private_subnet2,port_vm1net1,port_vm1net2]
  type: OS::Nova::Server
  properties:
    name: { get_param: VM1_name }
    image: { get_param: image_name }
    flavor: { get_param: flavor_name }
    networks:
    - port: { get_resource: port_vm1net1 }
    - port: { get_resource: port_vm1net2 }
    user_data: |
      #cloud-config
      packages:
      - aptsh
      - apache2
      runcmd:
      - ifconfig ens4 up
      - dhclient ens4
      - echo "<h1>Server 1</h1>" | sudo tee /var/www/html/index.html
    user_data_format: RAW
```

Figura 21 - Configuración servidores web

Por defecto Openstack sólo habilita una interfaz de red por máquina, *ens3*. Pero, si recordamos la topología de red hay varios servidores conectados a dos redes distintas simultáneamente, por lo que surge la necesidad de activar una segunda interfaz, *ens4*. Si ejecutamos el comando "*ifconfig*" en alguna de nuestras máquinas observaremos que sólo la interfaz *ens3* se encuentra activa. Para observar todas las interfaces, incluidas las deshabilitadas, ejecutamos "*ifconfig -a*". Comprobamos como la segunda interfaz *ens4* se encuentra desactivada. Para poder activar *ens4* ejecutamos "*ifconfig ens4 up*" y después forzamos la obtención de su IP mediante el comando "*dhclient ens4*". Una vez realizados estos pasos podemos comprobar que las dos interfaces se encuentran perfectamente operativas y funcionan correctamente.

En segundo lugar arrancaremos el servidor web. Este servidor escuchará peticiones entrantes sobre el puerto 80.

En la **Figura 22** se muestra el servidor de administración. Configuración bastante similar a la configuración de los servidores web.

```
server_admin:
  depends_on: [server_security_group,server_ssh_key,private_subnet1,private_subnet2,port_vm4net1,port_vm4net2]
  type: OS::Nova::Server
  properties:
    name: { get_param: VM4_name }
    key_name: { get_resource: server_ssh_key }
    image: { get_param: image_name }
    flavor: { get_param: flavor_name }
    networks:
    - port: { get_resource: port_vm4net1 }
    - port: { get_resource: port_vm4net2 }
    user_data: |
      #cloud-config
      hostname: adminserver
      fqdn: adminserver.example.com
      manage_etc_hosts: true
      packages:
      - aptsh
      runcmd:
      - ifconfig ens4 up
      - dhclient ens4
    user_data_format: RAW
```

Figura 22 - Configuración servidor administración

En la **Figura 23** se muestra el servidor de base de datos. Configuración bastante similar al resto de servidores salvo por su imagen fuente. El servidor de base de datos empleará una imagen customizada con MongoDB instalado. Más adelante explicaremos como poder configurar nuestras propias imágenes.

```
server_db:
  depends_on: [server_security_group,private_subnet2,port_vm5net2]
  type: OS::Nova::Server
  properties:
    name: { get_param: VM5_name }
    image: ServerDBSnapshot
    flavor: { get_param: flavor_name }
    networks:
    - port: { get_resource: port_vm5net2 }
    user_data: |
      #cloud-config
      hostname: dbserver
      fqdn: dbserver.example.com
      manage_etc_hosts: true
      packages:
      - aptsh
      runcmd:
      - echo db > /root/db.txt
    user_data_format: RAW
```

Figura 23 - Configuración servidor de base de datos

Finalmente, en la **Figura 24** se muestra la única salida proporcionada por nuestro stack, la clave SSH del servidor admin.

```
outputs:
  tenant_ssh_key:
    description: Private Key for Tenant Machine
    value: { get_attr: [server_ssh_key, private_key] }
```

Figura 24 - Heat output

Una vez hemos arrancado los servidores y el resto de componentes, creamos el balanceador de carga ejecutando los siguientes scripts de la **Figura 25**.

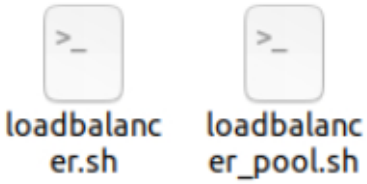

Figura 25 - LoadBalancer

Primero creamos el balanceador de carga con el script *loadbalancer.sh* mostrado en la **Figura 26**. Se omiten los credenciales de administración ya que son iguales para todos los scripts. Se añaden tres reglas de seguridad al balanceador de carga habilitando el puerto 80 y 443 para comunicaciones HTTP y HTTPS respectivamente, además de comunicaciones ICMP para la realización de pruebas. ICMP debe ser desactivado tras llevarlo a producción.

```
# Load balancer (https://docs.openstack.org/mitaka/networking-guide/config-lbaas.html)
neutron lbaas-loadbalancer-create --name lb Subnet1

# neutron lbaas-loadbalancer-show lb

neutron security-group-create lbaas
neutron security-group-rule-create \
  --direction ingress \
  --protocol tcp \
  --port-range-min 80 \
  --port-range-max 80 \
  --remote-ip-prefix 0.0.0.0/0 \
  lbaas
neutron security-group-rule-create \
  --direction ingress \
  --protocol tcp \
  --port-range-min 443 \
  --port-range-max 443 \
  --remote-ip-prefix 0.0.0.0/0 \
  lbaas
neutron security-group-rule-create \
  --direction ingress \
  --protocol icmp \
  lbaas

vip_port_id=$(neutron lbaas-loadbalancer-show lb -c vip_port_id -f value)

neutron port-update \
  --security-group lbaas \
  $vip_port_id

# Add HTTP listener

neutron lbaas-listener-create \
  --name lb-http \
  --loadbalancer lb \
  --protocol HTTP \
  --protocol-port 80
```

Figura 26 - LoadBalancer

Finalmente ejecutamos *loadbalancer_pool.sh,* **Figura 27**, que añadirá los servidores web al pool del balanceador para que sean accesibles desde él. También añadirá una IP flotante al balanceador para que este sea accesible desde el exterior.

```
# Crear la pool de servidores
neutron lbaas-pool-create \
  --name lb-pool-http \
  --lb-algorithm ROUND_ROBIN \
  --listener lb-http \
  --protocol HTTP

# Añadir servidores al pool
# https://docs.openstack.org/python-openstackclient/pike/cli/command-
list.html
# openstack server list
string=$(openstack server show Server1 -c addresses -f value)
string=$(echo "${string%%;*}")
private_ip_vm1=$(echo "${string#Net1=}")

string=$(openstack server show Server2 -c addresses -f value)
string=$(echo "${string%%;*}")
private_ip_vm2=$(echo "${string#Net1=}")

string=$(openstack server show Server3 -c addresses -f value)
string=$(echo "${string%%;*}")
private_ip_vm3=$(echo "${string#Net1=}")
```

```
neutron lbaas-member-create \
  --name lb-http-member-1 \
  --subnet Subnet1 \
  --address $private_ip_vm1 \
  --protocol-port 80 \
  lb-pool-http
neutron lbaas-member-create \
  --name lb-http-member-2 \
  --subnet Subnet1 \
  --address $private_ip_vm2 \
  --protocol-port 80 \
  lb-pool-http
neutron lbaas-member-create \
  --name lb-http-member-3 \
  --subnet Subnet1 \
  --address $private_ip_vm3 \
  --protocol-port 80 \
  lb-pool-http

# Monitoriza si algún servidor se ha caído para retirarlo del pool
#neutron lbaas-healthmonitor-create \
#  --delay 5 \
#  --max-retries 2 \
#  --timeout 10 \
#  --type HTTP \
#  --pool lb-pool-http

#floating_ip=$(openstack floating ip create ExtNet -c floating_ip_address -f
value)
vip_port_id=$(neutron lbaas-loadbalancer-show lb -c vip_port_id -f value)
floating_ip_id=$(openstack floating ip create --project TNT00 --subnet
ExtSubNet ExtNet -c id -f value)
neutron floatingip-associate $floating_ip_id $vip_port_id

# neutron lbaas-loadbalancer-stats lb
```

Figura 27 - Pool de servidores del balanceador e IP flotante

Una vez arrancado el balanceador de carga este será accesible desde el exterior gracias a su IP flotante. Comprobamos que el balanceador se encuentra accesible desde el exterior en la **Figura 28**.

```
javier.hernandez.sanchez@l111:~/Desktop/CNVR/Práctica Final 2$ curl 10.0.10.137
<h1>Server 2</h1>
javier.hernandez.sanchez@l111:~/Desktop/CNVR/Práctica Final 2$ curl 10.0.10.137
<h1>Server 3</h1>
javier.hernandez.sanchez@l111:~/Desktop/CNVR/Práctica Final 2$ curl 10.0.10.137
<h1>Server 1</h1>
javier.hernandez.sanchez@l111:~/Desktop/CNVR/Práctica Final 2$ curl 10.0.10.137
```

Figura 28 - Balanceador de carga en funcionamiento

Finalmente lanzamos el componente restante de nuestro escenario, el Firewall de Openstack. Para ello ejecutaremos *firewall.sh*, disponible en la **Figura 29**.

Figura 29 - Script firewall.sh

En la **Figura 30** se muestra la configuración del firewall. Este script crea una serie de reglas de seguridad. Posteriormente estas reglas se añaden al router de nuestro escenario. Se habilitan tres reglas, dos de entrada y una de salida. Las reglas de entrada sólo permiten tráfico SSH con dirección de destino al servidor administrador y tráfico HTTP con dirección al balanceador de carga. Cabe mencionar que dichas direcciones deben ser IPs internas, puesto que las reglas del firewall se aplican tras realizar la traducción de IPs.

```
# Firewall (https://docs.openstack.org/newton/networking-guide/fwaas-v2-scenario.html#configure-firewall-as-a-service-v2)

# openstack server list

# SSH Admin Server traffic rule
string=$(openstack server show ServerAdmin -c addresses -f value)
string=$(echo "${string%%;*}")
string=$(echo "${string#Net1=}")
server_admin_internal_ip=$(echo "${string%%,*}") # The firewall is applied after the NAT so we need the internal IP
openstack firewall group rule create --protocol tcp \
  --destination-ip-address $server_admin_internal_ip \
  --destination-port 22 \
  --action allow --name ssh_ingress

# HTTP Server traffic rule
load_balancer_internal_address=$(neutron lbaas-loadbalancer-show lb -c vip_address -f value)
openstack firewall group rule create --protocol tcp \
  --destination-ip-address $load_balancer_internal_address \
  --destination-port 80 \
  --action allow --name http_ingress

# All internal traffic is allowed to egress
openstack firewall group rule create --protocol any \
  --source-ip-address 10.103.82.0/24 \
  --action allow --name egress

# Firewall policy
openstack firewall group policy create ingressfirewallpolicy
openstack firewall group policy add rule ingressfirewallpolicy ssh_ingress
openstack firewall group policy add rule ingressfirewallpolicy http_ingress
openstack firewall group policy create egressfirewallpolicy
openstack firewall group policy add rule egressfirewallpolicy egress
# FWaaS always adds a default deny all rule at the lowest precedence of each policy.
Consequently, a firewall policy with no rules blocks all traffic by default.
string=$(openstack router show RTR1TNT82CL3 -c interfaces_info -f value)
string=$(echo "${string%%,*}")
string=$(echo "${string#*:}")
string=$(echo "${string#*"'"}")
string=$(echo "${string%"'"*}")
internal_router_port=$(echo "${string%"'"*}")

openstack firewall group create --port $internal_router_port --ingress-firewall-policy ingressfirewallpolicy --egress-firewall-policy egressfirewallpolicy --project TNT00 --name firewall
```

Figura 30 - Firewall

En la **Figura 31** se realiza un testeo rápido del firewall. Se muestran dos comandos, en primer lugar una petición HTTP exitosa con dirección IP flotante del balanceador, y después un ping fallido bloqueado satisfactoriamente por nuestro firewall.

```
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ curl 10.0.10.193
<h1>Server 2</h1>
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ ping 10.0.10.193
PING 10.0.10.193 (10.0.10.193) 56(84) bytes of data.
^C
--- 10.0.10.193 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4093ms
```

Figura 31 - Testeando el firewall I

Después, **Figura 32** accedemos al admin mediante ssh, comprobando que el tráfico ssh no es bloqueado por el firewall.

```
javier.hernandez.sanchez@l091:~/Desktop/CNVR/Práctica Final 2$ ssh vnx@10.0.10.184
vnx@10.0.10.184's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-170-generic x86_64)
```

Figura 32 - Testeando el firewall II

5. Customización de imágenes

Para configurar nuestras propias imágenes debemos conectaremos a una máquina con conexión a internet. Nos conectaremos por ejemplo al servidor de administración. Accedemos mediante ssh como se muestra en la **Figura 33**.

```
javier.hernandez.sanchez@l091:~/Desktop/CNVR/Práctica Final 2$ ssh vnx@10.0.10.184
vnx@10.0.10.184's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-170-generic x86_64)
```

Figura 33 – Accediendo al servidor de administración para la customización de imágenes

Una vez dentro del servidor realizamos la instalación de los programas necesarios para la imagen *custom*. En las figuras **Figura 34**, **Figura 35**, **Figura 36**, **Figura 37** se realiza la instalación de MongoDB que será la única herramienta que necesitemos en los servidores de base de datos.

```
vnx@adminserver:~$ curl -fsSL https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
[sudo] password for vnx:
OK
vnx@adminserver:~$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mon
godb-org-4.4.list
deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse
vnx@adminserver:~$ sudo apt update
Hit:1 http://nova.clouds.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://nova.clouds.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://nova.clouds.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu xenial-security InRelease
Ign:5 https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 InRelease
Get:6 https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 Release [3,094 B]
Get:7 https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 Release.gpg [801 B]
Get:8 https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4/multiverse amd64 Packages [32.0 kB]
Get:9 https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4/multiverse arm64 Packages [24.6 kB]
Fetched 60.5 kB in 1s (54.2 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
153 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Figura 34 - Actualizando repositorios apt

En las figuras **Figura 34**, **Figura 35** se actualizan los repositorios apt para que pueda instalarse MongoDB satisfactoriamente. A veces apt se queda bloqueado por algún proceso, por lo que puede ser necesario ejecutar los comandos de la **Figura 35** para evitarlo.

```
vnx@adminserver:~$ sudo rm /var/lib/apt/lists/lock
vnx@adminserver:~$ sudo rm /var/cache/apt/archives/lock
vnx@adminserver:~$ sudo rm /var/lib/dpkg/lock*
vnx@adminserver:~$ sudo dpkg --configure -a
vnx@adminserver:~$ sudo apt update
```

Figura 35 - Actualizando apt

En la **Figura 36** realizamos la instalación de MongoDB con el comando mostrado. Apt es capaz de encontrar el repositorio Mongo gracias a los pasos previos.

```
vnx@adminserver:~$ sudo apt-get install mongodb
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libboost-filesystem1.58.0 libboost-program-options1.58.0 libboost-system1.58.0 libboost-thread1.58.0 libgoogle-perftools4 libpcrecpp0v5 libsnappy1v5
  libtcmalloc-minimal4 libunwind8 libv8-3.14.5 libyaml-cpp0.5v5 mongodb-clients mongodb-server
The following NEW packages will be installed:
  libboost-filesystem1.58.0 libboost-program-options1.58.0 libboost-system1.58.0 libboost-thread1.58.0 libgoogle-perftools4 libpcrecpp0v5 libsnappy1v5
  libtcmalloc-minimal4 libunwind8 libv8-3.14.5 libyaml-cpp0.5v5 mongodb mongodb-clients mongodb-server
0 upgraded, 14 newly installed, 0 to remove and 153 not upgraded.
Need to get 58.0 MB of archives.
After this operation, 196 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://nova.clouds.archive.ubuntu.com/ubuntu xenial/main amd64 libpcrecpp0v5 amd64 2:8.38-3.1 [15.2 kB]
Get:2 http://nova.clouds.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libboost-system1.58.0 amd64 1.58.0+dfsg-5ubuntu3.1 [9,146 B]
Get:3 http://nova.clouds.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libboost-filesystem1.58.0 amd64 1.58.0+dfsg-5ubuntu3.1 [37.5 kB]
Get:4 http://nova.clouds.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libboost-program-options1.58.0 amd64 1.58.0+dfsg-5ubuntu3.1 [138 kB]
Get:5 http://nova.clouds.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libboost-thread1.58.0 amd64 1.58.0+dfsg-5ubuntu3.1 [47.0 kB]
```

Figura 36 - Instalación de MongoDB

Finalmente, comprobamos que MongoDB está instalado con el comando de la **Figura 37**. La versión v2.6.10 de MongoDB ha sido instalado satisfactoriamente.

```
vnx@adminserver:~$ mongo --version
MongoDB shell version: 2.6.10
```

Figura 37 - Comprobación MongoDB

Una vez instalado MongoDB debemos parar nuestra instancia, pero, previamente debemos autentificarnos con los comandos de la **Figura 38**. Además en dicha imagen se listan los servidores activos de Openstack.

```
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_USERNAME=tnt00user
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_PASSWORD=password
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_PROJECT_NAME=TNT00
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_USER_DOMAIN_NAME=Default
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_PROJECT_DOMAIN_NAME=Default
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_AUTH_URL=http://controller:5000/v3
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_IDENTITY_API_VERSION=3
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_IMAGE_API_VERSION=2
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ export OS_AUTH_TYPE=password
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ openstack server list
+--------------------------------------+------------+--------+-----------------------------------------------------+-----------------------------+-------
-----+
| ID                                   | Name       | Status | Networks                                            | Image                       | Flavor
     |
+--------------------------------------+------------+--------+-----------------------------------------------------+-----------------------------+-------
-----+
| 1a5aded8-ba99-4124-a1de-c01c75154dee | Server1    | ACTIVE | Net1=10.103.82.12; Net2=10.103.83.53                | xenial-server-cloudimg-amd64-vnx | m1.sma
ller |
| 8ee09381-cc08-4687-9a94-461da1dc8fcd | Server2    | ACTIVE | Net1=10.103.82.91; Net2=10.103.83.13                | xenial-server-cloudimg-amd64-vnx | m1.sma
ller |
| 1f139440-9be4-41f5-8830-f6804558f36d | ServerDB   | ACTIVE | Net2=10.103.83.92                                   | xenial-server-cloudimg-amd64-vnx | m1.sma
ller |
| a47747f3-5913-4a4d-a870-4bdc226505c6 | Server3    | ACTIVE | Net1=10.103.82.45; Net2=10.103.83.56                | xenial-server-cloudimg-amd64-vnx | m1.sma
ller |
| b7407443-13c9-44bc-bf7a-2e5e769c5023 | ServerAdmin | ACTIVE | Net1=10.103.82.66, 10.0.10.130; Net2=10.103.83.133 | xenial-server-cloudimg-amd64-vnx | m1.sma
ller |
+--------------------------------------+------------+--------+-----------------------------------------------------+-----------------------------+-------
-----+
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$
```

Figura 38 - Autentificación y listado de servidores activos

Paramos nuestra instancia con el comando de la **Figura 39**. Además se puede observar los servidores encendidos y apagados para comprobar que el comando ha funcionado exitosamente.

```
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ openstack server stop ServerAdmin
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ openstack server list
+--------------------------------------+------------+---------+-----------------------------------------------------+-----------------------------+------
------+
| ID                                   | Name       | Status  | Networks                                            | Image                       | Flavo
r     |
+--------------------------------------+------------+---------+-----------------------------------------------------+-----------------------------+------
------+
| 1a5aded8-ba99-4124-a1de-c01c75154dee | Server1    | ACTIVE  | Net1=10.103.82.12; Net2=10.103.83.53                | xenial-server-cloudimg-amd64-vnx | m1.sm
aller |
| 8ee09381-cc08-4687-9a94-461da1dc8fcd | Server2    | ACTIVE  | Net1=10.103.82.91; Net2=10.103.83.13                | xenial-server-cloudimg-amd64-vnx | m1.sm
aller |
| 1f139440-9be4-41f5-8830-f6804558f36d | ServerDB   | ACTIVE  | Net2=10.103.83.92                                   | xenial-server-cloudimg-amd64-vnx | m1.sm
aller |
| a47747f3-5913-4a4d-a870-4bdc226505c6 | Server3    | ACTIVE  | Net1=10.103.82.45; Net2=10.103.83.56                | xenial-server-cloudimg-amd64-vnx | m1.sm
aller |
| b7407443-13c9-44bc-bf7a-2e5e769c5023 | ServerAdmin | SHUTOFF | Net1=10.103.82.66, 10.0.10.130; Net2=10.103.83.133 | xenial-server-cloudimg-amd64-vnx | m1.sm
aller |
+--------------------------------------+------------+---------+-----------------------------------------------------+-----------------------------+------
------+
```

Figura 39 - Parar instancia y listado de servidores activos

Después creamos una snapshot de nuestra instancia customizada con el comando de **Figura 40**.

```
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ openstack server image create --name ServerDBSnapshot ServerAdmin
```

Figura 40 - Crear snapshot de nuestra imagen customizada

En la **Figura 41** listamos todas las imágenes disponibles en Openstack. Como se puede observar aparece nuestra snapshot.

```
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ openstack image list
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| 3111517c-6ea1-434a-8ab9-6398e9cdf9ab | ServerDBSnapshot                | active |
| 33c71dc6-2865-4a28-ab34-df2ee77f9d8e | cirros-0.3.4-x86_64-vnx         | active |
| 3cbe4278-8f30-4beb-8d36-dab8f33e379c | xenial-server-cloudimg-amd64-vnx | active |
+--------------------------------------+---------------------------------+--------+
```

Figura 41 - Listado de imágenes Openstack

Finalmente en la **Figura 42** guardamos y exportamos la snapshot.

```
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ openstack image save --file serverdb_snapshot.raw 3111517c-6ea1-434a-8ab9-6398e9cdf9ab
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$
```

Figura 42 - Exportar imagen customizada

El comando de la **Figura 42** debería generar un fichero imagen de formato *qcow2* como el archivo de la **Figura 43**.

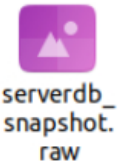

Figura 43 - Imagen customizada

Una vez exportada nuestra imagen ya podríamos usarla en nuestro escenario. Para ello destruiremos el escenario físico con *destroy_cloud.sh* y volveríamos a desplegarlo con *cloud.sh*. Inmediatamente después el script *load_image.sh* mostrado anteriormente para cargar nuestra imagen.

Finalmente tras arrancar completamente el escenario realizamos todas las comprobaciones pertinentes. En la **Figura 44** se muestran las tres imágenes que ofreceremos a nuestros clientes en su escenario virtual.

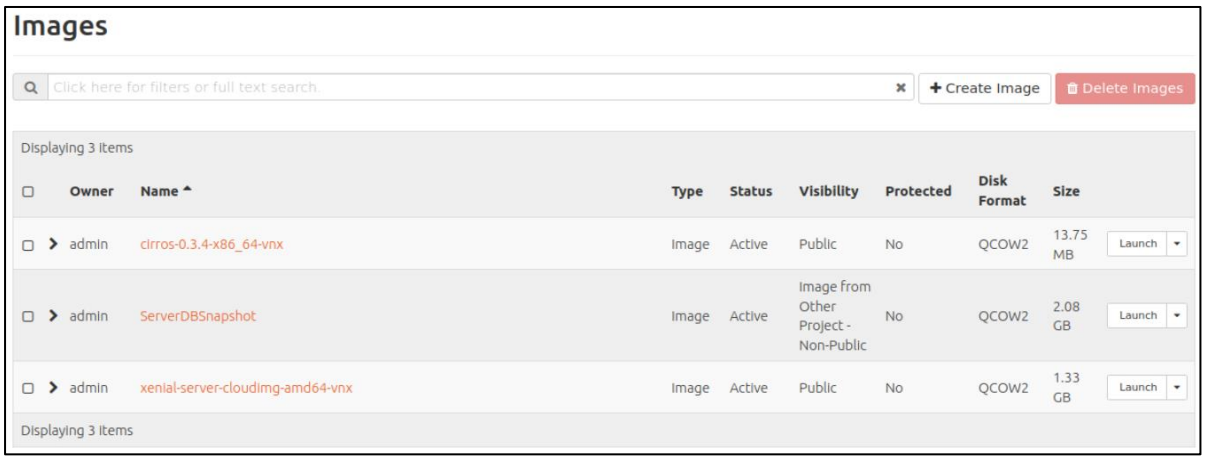

Figura 44 - Imágenes disponibles en Openstack tras realizar todos los pasos de despliegue

Finalmente, nos conectamos al servidor de base de datos para comprobar que la instalación ha funcionado correctamente. En la **Figura 45** nos conectamos al servidor de administración, puesto que es el único servidor accesible desde el exterior mediante ssh. Debemos hacer un ssh anidado para acceder al servidor de base de datos.

```
javier.hernandez.sanchez@l060:~/Desktop/CNVR/Práctica Final 2$ ssh vnx@10.0.10.139
The authenticity of host '10.0.10.139 (10.0.10.139)' can't be established.
ECDSA key fingerprint is SHA256:ckoJE+EuU4MH0xZLWb7+1ajDWwdS+kIUM/RYCOxLP7o.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

Figura 45 - Ssh servidor de administración

Después, en la **Figura 46** accedemos al servidor de base de datos con la IP de su interfaz de red Net2.

```
vnx@adminserver:~$ ssh vnx@10.103.83.173
The authenticity of host '10.103.83.173 (10.103.83.173)' can't be established.
ECDSA key fingerprint is SHA256:LsRRkDEEJwc9QZLa/kkG8IwbfhhKTPeWNlnQxrQJS60.
Are you sure you want to continue connecting (yes/no)? yes
```

Figura 46 - Ssh servidor de base de datos

Finalmente, en la **Figura 47**, comprobamos que la imagen customizada contiene MongoDB instalado.

```
vnx@dbserver:~$ mongo --version
MongoDB shell version: 2.6.10
vnx@dbserver:~$
```

Figura 47 - Comprobación final instalación de MongoDB

6. Resumen de despliegue

A continuación, se ofrece un resumen con los pasos de ejecución para el despliegue:

1. *cloud.sh*
2. *create-tenant.sh*
3. *init-tenant.sh*
4. *loadbalancer.sh*
5. *loadbalancer-pool.sh*
6. *firewall.sh*

Para destruir el escenario ejecutaremos:

- *destroy_cloud.sh*

Importante, los scripts deben ejecutarse en el orden correcto. Además, se debe comprobar antes de ejecutar cualquier script que su script previo ha terminado su tarea exitosamente.

7. Referencias

[1] https://www.openstack.org/

[2] https://whatcloud.wordpress.com/2017/01/09/ose4eg/

[3] https://whatcloud.wordpress.com/2017/01/26/ose5uh/

[4] https://docs.openstack.org/heat/pike/template_guide/software_deployment.html

[5] https://docs.openstack.org/nova/rocky/admin/migrate-instance-with-snapshot.html