Capítulo 3

Cloud computing
Plataformas IaaS

Javier Esparza Peidro - jesparza@dsic.upv.es

Contenido

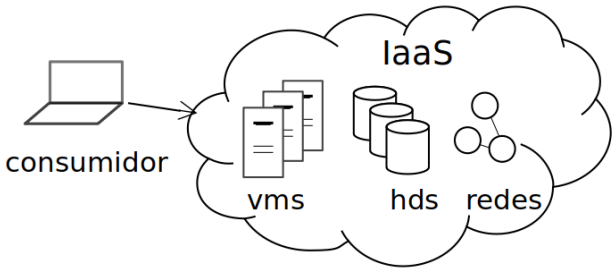

Figura 3.1: Modelo de servicio IaaS

3.1. Introducción

De acuerdo con la definición de NIST [Mell et al., 2011] un entorno cloud de tipo IaaS proporciona recursos de *procesamiento*, *almacenamiento* y *redes* (ver figura 3.1) que permiten al consumidor desplegar y ejecutar software, incluyendo sistemas operativos y aplicaciones. El consumidor no tiene control de la infraestructura subyacente, pero sí es responsable de los sistemas operativos, el almacenamiento y las aplicaciones, y posee cierto control sobre la conectividad de los sistemas.

Por tanto, se trata de un modelo en el que el usuario debe poseer conocimientos avanzados de gestión de sistemas informáticos, es un modelo pensado para administradores de sistemas o arquitectos de sistemas.

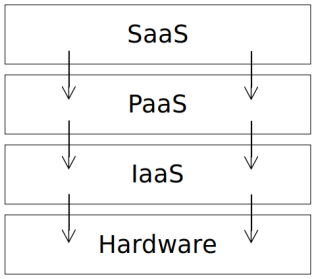

Figura 3.2: Modelos de servicio como sistema a capas

Un entorno cloud IaaS permite gestionar una infraestructura virtual, en lugar de una infraestructura física, tal y como se hacía en los centros de datos tradicionales.

Considerando los distintos modelos de servicio proporcionados por un entorno cloud [Mell et al., 2011], los servicios IaaS ocupan la posición de más bajo nivel, y suelen utilizarse como base para la construcción de servicios de más alto nivel, como PaaS o SaaS (ver figura 3.2).

3.1.1. Computación

En un IaaS las capacidades de computación generalmente están representadas por medio de *máquinas virtuales*, habitualmente denominadas *instancias*. De este modo, el IaaS debe gestionar distintos hipervisores para proporcionar instancias con los requisitos de CPU (incluso GPU) y memoria requeridos por el consumidor en un breve espacio de tiempo.

Generalmente un IaaS proporciona distintas opciones preestablecidas a la hora de escoger las prestaciones de una instancia. Por ejemplo, el servicio Amazon AWS EC2[1] ofrece más de 50 modelos distintos, y cada uno con diferentes configuraciones de núcleos y memoria. El consumidor deberá analizar sus requerimientos y escoger el tipo de instancia más adecuado.

Para agilizar el aprovisionamiento de las instancias, es habitual crearlas a partir de cierta *imagen* que posee su propio sistema operativo y aplicaciones preinstaladas. El IaaS crea una copia de la imagen y la monta en la instancia como un dispositivo de almacenamiento orientado a bloques (ver figura 3.3). La instancia arranca a partir de este dispositivo, que constituye su espacio de almacenamiento primario.

Las instancias pueden ser arrancadas, paradas, pausadas, reanudadas y destruidas. Una instancia se comporta como un host tradicional y es posible conectarse a ésta utilizando los mecanismos habituales, como SSH.

[1] https://aws.amazon.com/es/ec2/instance-types/

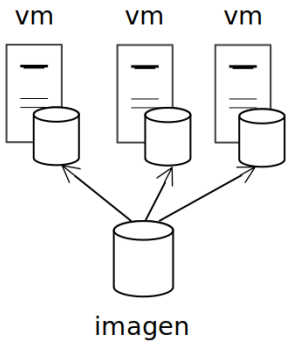

Figura 3.3: La imagen es clonada y montada en la máquina virtual

El consumidor suele monitorizar el consumo de recursos de las instancias adquiridas y puede solicitar la creación de nuevas instancias o destruirlas, según sus necesidades. Es muy común que el IaaS facilite herramientas automáticas de escalado. Son reglas que permiten al consumidor indicar cuándo deben crearse o destruirse instancias. Estas reglas suelen incluir dos partes:

- Las condiciones que se deben verificar para que se aplique la regla: suelen incluir la comprobación de métricas de consumo de recursos, como % de CPU utilizado, % de memoria utilizada o el número de operaciones de entrada/salida por segundo (IOPS).
- Las acciones a tomar cuando las condiciones se verifiquen: suelen incluir la creación de réplicas de ciertas instancias, si el consumo de recursos sobrepasan cierto límite, o la destrucción de instancias en caso contrario.

3.1.2. Almacenamiento

Es habitual que los IaaS proporcionen múltiples opciones de almacenamiento:

- Almacenamiento de tipo bloque: suelen suministrarse entorno al concepto de *volumen* o *disco virtual*. Es el tipo de almacenamiento primario de una instancia y puede ser efímero o persistente. En el primer caso, cuando la instancia se destruye, también lo hace todo su estado, incluyendo sus volúmenes. En el segundo caso, el ciclo de vida de la instancia y del volumen son independientes; cuando la instancia se destruye, el volumen se preserva y puede ser utilizado por otras instancias.
- Sistema de ficheros compartido: se trata de sistemas de almacenamiento como NFS o CIFS, que permiten a múltiples instancias compartir un sistema de ficheros.

- Almacenamiento SQL: se trata de motores de base de datos relacionales, generalmente adaptados para funcionar en un entorno cloud. Suelen estar replicados y optimizados para soportar importantes cargas de trabajo y grandes volúmenes de datos.
- Almacenamiento NoSQL: existe una gran variedad de motores que ofrecen opciones de almacenamiento diferentes al modelo relacional, como repositorios de objetos o documentos, almacenes de tipo clave-valor o almacenes orientados a grafos.

3.1.3. Redes

Un IaaS debe proporcionar conectividad a sus instancias. Esta capacidad puede ser ofrecida de diversas maneras. Una opción es implementar un espacio de direccionamiento plano, donde todas las instancias se ven entre sí. Otra opción consiste en permitir la creación de redes y subredes virtuales, y habilitar un mecanismo para conectar instancias a dichas redes. Esta última opción, aunque más compleja, es realmente potente, pues permite el diseño de complejas arquitecturas de red totalmente personalizadas.

Resulta bastante habitual que las instancias reciban una dirección IP privada y que no sean directamente accesibles desde el exterior. Si se desea hacer pública una instancia es necesario solicitarlo explícitamente. En estos casos, o bien la instancia recibe una dirección IP pública, o bien se utilizan mecanismos orientados a la aplicación, por ejemplo usando balanceadores de carga o subdominios, que finalmente redirigen las comunicaciones a la instancia en cuestión.

3.1.4. Regiones

Por cuestiones de disponibilidad, es muy común que los recursos puedan ser creados en distintas regiones. Las regiones representan ubicaciones geográficamente dispersas. Si se distribuyen los recursos de un consumidor en diferentes zonas se incrementa la fiabilidad de la solución global.

Por ejemplo en AWS y Google Cloud una zona representa un centro de datos, mientras que una región es un concepto más amplio, que representa un área geográfica. De este modo, una misma región puede contener una colección de zonas.

3.2. Anatomía de un IaaS

Tomando como punto de inicio la arquitectura de referencia descrita en [Liu et al., 2011], un IaaS puede descomponerse de manera general en las siguientes

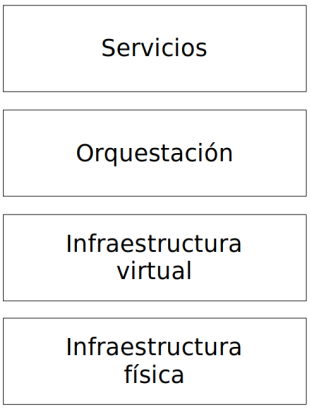

Figura 3.4: Anatomía de un IaaS

capas (ver figura 3.4):

- *La infraestructura física*: contiene todos los recursos físicos del sistema, incluyendo computadores, redes, dispositivos de almacenamiento, etc., así como la infraestructura y suministros (energía, refrigeración, etc.) necesarios.

- *La infraestructura virtual*: los recursos físicos son virtualizados. Esto configura un gran pool de recursos virtuales que constituye las piezas básicas para suministrar infraestructura a los consumidores del servicio. Además de compartición de recursos y escalabilidad, la virtualización permite migrar servidores virtuales entre máquinas físicas, con el objetivo de balancear la carga.

- *Orquestación*: incluye todos los servicios necesarios para gestionar adecuadamente la infraestructura virtual. La gestión de los recursos virtuales se lleva a cabo siguiendo ciertas políticas que persiguen cubrir las expectativas de los consumidores por un lado, y efectuar un uso óptimo de la infraestructura por otro. Todas las operaciones sobre los recursos virtuales son automatizadas. Esto favorece el aprovisionamiento instantáneo de recursos y su mantenimiento, así como una monitorización contínua de su uso y estado de salud.

- *Servicios*: en esta capa se publica la API que permite consumir los recursos ofrecidos por el entorno IaaS.

En los siguientes apartados se dan más detalles acerca de estas capas.

3.3. Infraestructura física

Al final, el software se ejecuta en máquinas físicas, se comunica a través de redes físicas y almacena la información en dispositivos de almacenamiento físicos. Esta capa contiene todos los recursos físicos del sistema.

Es innegable que en el nivel físico ciertas tareas deben ser efectuadas de manera manual. Por ejemplo, conectar un nuevo servidor, un switch de red o un servidor NAS en el centro de datos. Sin embargo, dejando a un lado las tareas de conexión física, debido al gran volumen de dispositivos existentes en un centro de datos, es necesario disponer de herramientas que permitan automatizar su instalación, configuración y operación.

En este sentido, las herramientas se pueden clasificar en diversas categorías, por ejemplo:

- *Aprovisionamiento*: automatizan la instalación del sistema operativo y aplicaciones de un gran número de máquinas. Algunos ejemplos son Clober, Kickstart, Spacewalk, Crowbar, etc.
- *Gestión de la configuración*: automatizan la configuración de máquinas a gran escala. En general, permiten aplicar distintas configuraciones a todas las máquinas de un centro de datos. Algunos ejemplos son Ansible, Cfengine, Chef, Puppet, etc.
- *Monitorización*: monitorizan de manera contínua los recursos del sistema. Gestionan los logs del sistema, habilitan diversas métricas de rendimiento y detectan caídas. Algunos ejemplos son Nagios, Zabbix, Grafana, Zenoss, etc.

3.3.1. Ansible

Ansible es una herramienta que permite automatizar tareas sobre cierta infraestructura (física o virtual), potencialmente compuesta por cientos o miles de nodos. Ansible es capaz de configurar sistemas, instalar o actualizar software, etc. Para ello, a partir de una máquina maestra (nodo de control) que posee las herramientas adecuadas, se conecta con las máquinas a configurar (nodos gestionados) a través de OpenSSH (existen otras alternativas) y ejecuta en ellas los comandos necesarios para alcanzar cierto estado. (ver figura 3.5)

Este modo de funcionamiento recibe la denominación de configuración PUSH, en contraste con la configuración PULL, en la que los nodos gestionados contactan periódicamente con el servidor y descargan la configuración más reciente. Una ventaja de la aproximación PUSH es que los nodos gestionados no requieren la instalación de ningún software adicional, únicamente OpenSSH y Python.

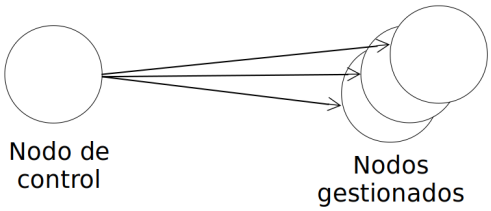

Figura 3.5: Ansible

Por su parte, el nodo de control requiere la instalación de OpenSSH, Python y Ansible.

Ansible permite ejecutar comandos arbitrarios en cualquier nodo gestionado. Además, incluye un catálogo muy extenso de módulos que permiten ejecutar cualquier tipo de tarea sobre el nodo remoto. Los módulos son declarativos, es decir, se especifica el estado que se desea alcanzar en los nodos gestionados. Los módulos son idempotentes, es decir, si la misma acción se aplica múltiples veces, el resultado siempre es el mismo.

3.3.1.1. Instalación

Ansible está desarrollado en Python. Se puede instalar de diversas maneras[2]. Puede instalarse de manera local a través de pip o a través del gestor de paquetes del sistema operativo. También se puede instalar en un entorno virtual.

```
> pip install ansible
```

```
> sudo apt update
> sudo apt install software-properties-common
> sudo add-apt-repository --yes --update ppa:ansible/ansible
> sudo apt install ansible
```

Ansible debe ser capaz de conectarse a los nodos gestionados utilizando SSH. Por tanto primero es necesario asegurarse de que el servidor OpenSSH está instalado en los nodos gestionados.

```
> apt install openssh-server
```

Además, será necesario que los nodos gestionados permitan autenticación automática a través de clave pública. En el fichero `/etc/ssh/sshd_config` se habilita la siguiente propiedad.

[2] https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

```
PubkeyAuthentication yes
```

A continuación es necesario generar un par de claves pública/privada en el nodo controlador y registrar la clave pública en el fichero `~/.ssh/authorized_keys` del usuario con el que se desea trabajar (usualmente `root` o usuario `sudo`) en los nodos gestionados.

Para la primera parte se utiliza `ssh-keygen`; para la segunda parte se puede hacer de manera manual, o bien utilizar la herramienta `ssh-copy-id`:

```
> ssh-keygen
> ssh-copy-id -i <clave_pública> <host remoto>
> ssh-copy-id -i .ssh/id_rsa.pub root@remotemachine
> ssh root@remotemachine
```

3.3.1.2. Configuración

En Ansible es habitual definir opciones por defecto en un fichero de configuración. De este modo, no es necesario establecerlas cada vez que se ejecuta Ansible. Existen diversas opciones para ello. A continuación se lista el orden de búsqueda que sigue Ansible cada vez que se ejecuta:

1. Analiza el fichero especificado en la variable de entorno `ANSIBLE_CONFIG`
2. Analiza el fichero en el directorio actual `./ansible.cfg`
3. Analiza el fichero en el `HOME` del usuario actual `~/.ansible.cfg`
4. Analiza el fichero de configuración global `/etc/ansible/ansible.cfg`

A continuación se lista un ejemplo de fichero de configuración:

```
[defaults]
inventory = hosts
remote_user = root
private_key_file = .ssh/id_rsa
host_key_checking = False
```

3.3.1.3. Comandos add hoc

En Ansible se pueden ejecutar comandos ad hoc para tareas puntuales. Lo habitual es utilizar playbooks. La sintaxis de un comando ad hoc es como sigue:

```
> ansible [pattern] -m [module] -a "[module options]"
```

El patrón determina los nodos sobre los que se efectuará la acción. Se discutirá este extremo en la siguiente sección.

El módulo determina la acción a efectuar. Dependiendo del módulo será necesario (o no) especificar más opciones. Ansible posee un catálogo inmenso de módulos que permiten efectuar todo tipo de acciones sobre los nodos.

Por ejemplo, a continuación se ejecutará una primera conexión contra el nodo local, ejecutando el módulo ping, que permite comprobar si la conexión se efectúa con éxito:

```
> ansible localhost -m ping
```

El módulo `command` permite ejecutar comandos arbitrarios sobre el nodo destino. Es necesario además suministrar en un argumento adicional el comando por medio del flag `-a`. El módulo `command` es el módulo por defecto, es decir, se asume si no se especifica ningún módulo.

```
> ansible localhost -m command -a uptime
> ansible localhost -a uptime
> ansible localhost -a 'ls -la'
```

Por defecto, Ansible intentará conectarse a la máquina utilizando el usuario utilizado para efectuar la conexión SSH. Para cambiar de usuario se utiliza el flag `-u`.

```
> ansible localhost -a "/sbin/reboot" -u root
```

Muchos comandos requieren privilegios de superusuario. Se pueden reclamar dichos privilegios utilizando los flags `-b`/`--become` (become root) y `-K`/`--ask-become-pass`.

```
> ansible localhost -a "/sbin/reboot" -u root --become [--ask-become-pass]
```

3.3.1.4. Inventario

Ansible efectúa operaciones sobre una colección de nodos gestionados. Estos nodos están recogidos en el *inventario*. El inventario contiene las direcciones de los nodos, y opcionalmente la información de autenticación; además es posible organizar los nodos en grupos, y aplicar acciones a todo el conjunto.

Una manera sencilla de especificar el inventario es a través de un fichero. A continuación se muestra un ejemplo:

```
www.example.com
db-[a:f].example.com
```

Patrón	Descripción
all	Todos los hosts en el inventario
host1	El host con nombre host1
host1:host2 ó host1,host2	Especifica una colección de hosts

Cuadro 3.1: Patrones reconocidos por Ansible

Se pueden especificar variables, aportando más información sobre cada host. Algunas variables tienen significados especiales para Ansible. Por ejemplo:

```
test1 ansible_host=192.168.122.149
test3 ansible_host=192.168.122.149
test2 ansible_host=192.168.122.149 ansible_port=22 ansible_user=root ansible_private
```

Es muy habitual agrupar los hosts en el inventario, por ejemplo:

```
[webservers]
www01.example.com
www02.example.com
www[03:50].example.com
```

Para especificar el inventario de nodos, se utiliza el flag `-i`. Entonces se pueden aplicar acciones de manera selectiva. Por ejemplo, a continuación se ejecuta el módulo ping sobre la máquina test1:

```
> ansible test1 -i hosts -m ping
```

A la hora de especificar los destinatarios de la acción, Ansible reconoce algunos patrones muy útiles, recogidos en la tabla 3.1.

3.3.1.5. Playbooks

En Ansible las configuraciones sobre los nodos gestionados suelen efectuarse a partir de *playbooks*. Un playbook es una lista de tareas que se puede ejecutar de manera ordenada múltiples veces. Los playbooks se utilizan para definir scripts que permiten instalar, configurar y actualizar sistemas de manera reusable.

Un playbook contiene una lista de *plays*. Cada play determina una colección de *tareas* y los nodos sobre los que se deben ejecutar. Si un playbook debe ejecutar distintas tareas sobre distintos hosts es necesario especificar distintos plays. Cada tarea ejecuta un *módulo* Ansible. La figura 3.6 ilustra esta relación de conceptos.

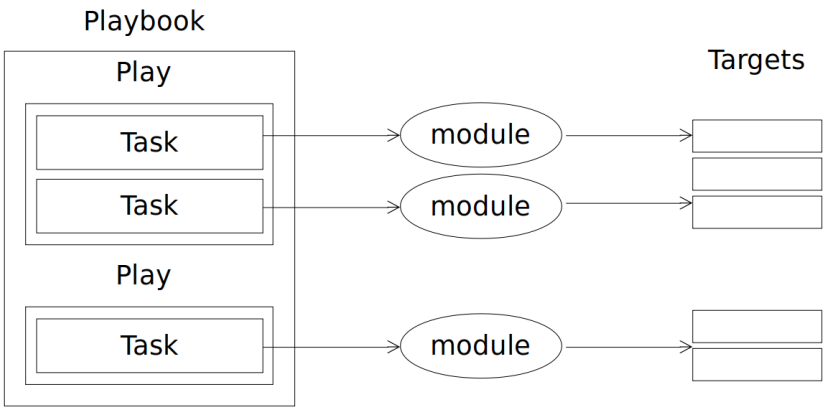

Figura 3.6: Estructura de un playbook en Ansible

Un playbook se define utilizando YAML[3] A continuación se muestra el ejemplo `ping.yml` con un play que ejecuta ping sobre todos los nodos definidos en el inventario:

```
# Very simple play
- name: Ping all hosts
  hosts: all
  tasks:
  - name: Make ping
    ping:
```

Para ejecutar un playbook utilizamos el comando `ansible-playbook`:

```
> ansible-playbook -i hosts ping.yml
```

Se pueden especificar múltiples opciones para un play[4]. En la tabla 3.2 se listan algunas de ellas.

A continuación se lista un playbook con varios plays:

```
---
# Multiple plays
- name: Add user
  hosts: master
  become: true
  tasks:
```

[3] https://yaml.org/

[4] https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html

Propiedad	Descripción
become	Booleano que determina si se ejecutan todas las tareas como superusuario
become_user	Determina el nombre del superusuario
gather_facts	Booleano que determina si se obtiene información de los nodos
hosts	Patrón que determina sobre qué nodos se ejecutarán las tareas
name	Identificador del play
remote_user	Determina el usuario que se usará para la conexión
tasks	Lista de tareas a ejecutar
vars	Diccionario con variables

Cuadro 3.2: Opciones soportadas por los plays de Ansible

```
  - name: Create user
    user:
      name: alumno
      group: users,admin
      shell: /bin/bash
      home: /home/alumno

- name: Install python
  hosts: slaves
  become: true
  tasks:
  - name: Install Python
    apt:
      name: python3
      state: present
```

3.3.1.6. Módulos

Son las unidades de código ejecutadas por Ansible. Se trata de scripts que efectúan alguna acción sobre un nodo. Ansible posee un catálogo[5] muy amplio de módulos. Cada módulo acepta unos parámetros diferentes, y devuelve unos resultados. Los módulos son diseñados para ser idempotentes, es decir, si se ejecutan múltiples veces el resultado debe ser el mismo.

Es posible obtener documentación sobre cada módulo utilizando el comando

[5] https://docs.ansible.com/ansible/2.8/modules/list_of_all_modules.html

Propiedad	Descripción
file	Para modificar las propiedades de un fichero, o eliminarlo
command, shell	Para ejecutar comandos
apt	Para instalar paquetes Debian/Ubuntu
copy	Para copiar ficheros de la máquina local a la máquina remota
user	Para gestionar cuentas de usuario
service	Para gestionar los servicios en ejecución
debug	Para mostrar información por pantalla
lineinfile	Comprueba la existencia de una fila en un fichero o la reemplaza
git	Para gestionar repositorios git
setup	Muestra información de los hosts remotos
template	Copia ficheros local a la máquina remota reemplazando plantillas

Cuadro 3.3: Módulos populares de Ansible

`ansible-doc`.

```
> ansible-doc user
```

En las tabla 3.3 se listan algunos de los módulos más populares.

3.3.1.7. Variables

Cada vez que se ejecuta Ansible, es posible definir variables[6]. Las variables se pueden definir en YAML y pueden contener strings, listas, diccionarios y combinaciones de estos. También se pueden definir a partir de los valores devueltos por las tareas. Resultan muy útiles en los siguientes escenarios:

- Para definir parámetros en las tareas.
- Para ejecutar tareas de manera condicional.
- En plantillas.
- En bucles.

[6] https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Las variables se pueden definir de los siguientes modos:

- En el inventario:

```
test ansible_host=192.168.1.1
```

- En un play de un playbook con la propiedad vars:

```
---
- hosts: server
  vars:
    user_name: pepe
```

- En un fichero myvars.yml:

```
---
var1: valu1
var2: valu2
```

Y se importan en un play de un playbbok:

```
---
- hosts: server
  vars_files:
  - /vars/myvars.yml
```

- En línea de comandos con el flag `-e`/`--extra-vars`:

```
> ansible-playbook -i hosts -e "var1:value1 var2:value2" playbook.yml
```

A partir de los resultados devueltos por otras tareas utilizando la propiedad `register`. El resultado es un objeto con diversos campos, por ejemplo `stdout` contiene la salida del comando, mientras que `rc` contiene el código de retorno.

```
---
- hosts: web_servers
  tasks:
    - name: Run a shell command and register its output as a variable
      ansible.builtin.shell: /usr/bin/foo
      register: foo_result
```

```
    ignore_errors: true

  - name: Run a shell command using output of the previous task
    ansible.builtin.shell: /usr/bin/bar
    when: foo_result.rc == 5
```

- A partir de los *facts* obtenidos de cada host. Por defecto, para cada nodo gestionado, Ansible obtiene información sobre éste y la almacena en la variable `ansible_facts`. Se trata de un diccionario con gran cantidad de información sobre el nodo gestionado.

```
---
- name: ansible facts
  hosts: all
  tasks:
  - debug:
      var: ansible_facts
```

Una vez definida la variable, es posible referenciarla utilizando la sintaxis de plantillas Jinja2[7]. Jinja2 incluye diversos filtros[8] que permiten manipular el contenido de las variables (es necesario utilizar "para evitar confusiones con los diccionarios). Por ejemplo:

```
---
- hosts: server
  vars:
    user_name: pepe
    tasks:
      - user:
          name: "{{ user_name }}"
          group: users,admin
          shell: /bin/bash
          home: /home/alumno
```

3.3.1.8. Condicionales

En un playbook puede resultar muy útil ejecutar tareas de manera condicional[9]. Esto se puede hacer con la propiedad `when`, que evalúa una expresión

[7] https://jinja.palletsprojects.com/en/3.0.x/

[8] https://jinja.palletsprojects.com/en/3.0.x/templates/builtin-filters

[9] https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#playbooks-conditionals

Jinja2 (sin utilizar {{ }}) para cada host, y en caso de obtener `True` se ejecuta la tarea asociada. Es muy habitual utilizar en la expresión variables registradas previamente.

A continuación se muestran algunos ejemplos:

- Comprobando el valor de un fact:

```
tasks:
  - name: Shut down Debian flavored systems
    ansible.builtin.command: /sbin/shutdown -t now
    when: ansible_facts['os_family'] == "Debian"
```

- Comprobando el valor de una variable registrada previamente.

```
---
- hosts: all
  vars:
    epic: true
    monumental: "yes"
  tasks:
    - name: Run the command if "epic" or "monumental" is true
      ansible.builtin.shell: echo "This certainly is epic!"
      when: epic or monumental | bool

    - name: Run the command if "epic" is false
      ansible.builtin.shell: echo "This certainly isn't epic!"
      when: not epic
```

- Comprobando el valor de una tarea ejecutada previamente.

```
---
- name: Test play
  hosts: all
  tasks:
      - name: Register a variable
        ansible.builtin.shell: cat /etc/motd
        register: motd_contents
      - name: Use the variable in conditional statement
        ansible.builtin.shell: echo "motd contains the word hi"
        when: motd_contents.stdout.find('hi') != -1
```

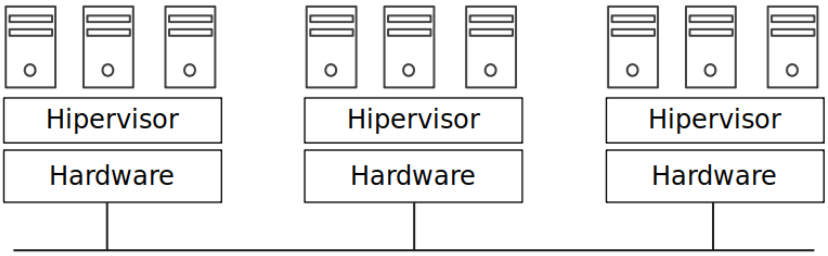

Figura 3.7: Nodos físicos con hipervisor interconectados

3.4. Infraestructura virtual

La infraestructura física es virtualizada. De este modo se consigue una mejor gestión y utilización de los recursos disponibles. Virtualizar la infraestructura física implica virtualizar, entre otros, los tres recursos computacionales fundamentales:

- Virtualización de la ejecución.
- Virtualización de la red.
- Virtualización del almacenamiento.

3.4.1. Virtualización de la ejecución

En general, la infraestructura virtual se construye a partir de nodos físicos que poseen un hipervisor y que permiten la ejecución de múltiples máquinas virtuales, tal y como se ilustra en la figura 3.7. En este contexto, soluciones multi-hipervisor como *libvirt* son muy recomendables, ya que posibilita trabajar con diversas tecnologías de virtualización.

3.4.2. Virtualización de la red

En una infraestructura virtual, las instancias pueden ejecutarse en cualquier nodo físico, y pueden comunicarse con otras máquinas virtuales, ejecutándose en el mismo u otros nodos físicos (ver figura 3.8).

Existen diversas alternativas para implementar esta conectividad, por ejemplo:

- Proporcionar un espacio de direccionamiento plano, donde cualquier máquina virtual puede comunicarse con cualquier otra.
- Proporcionar herramientas para la definición de una infraestructura de red virtual, compuesta por redes y subredes, conectadas por medio de puentes y routers.

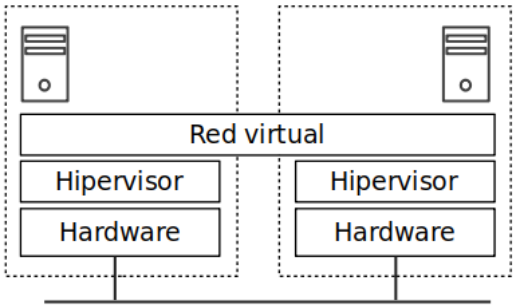

Figura 3.8: Máquinas virtuales conectadas a una red virtual

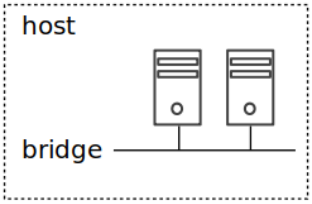

Figura 3.9: El bridge de Linux visto como un switch

En cualquier caso, es necesario virtualizar la red física, permitiendo la creación y configuración de múltiples redes virtuales independientes.

Las máquinas virtuales que se ejecutan en un mismo host pueden comunicarse entre sí fácilmente utilizando soluciones como el *bridge de Linux*, que permite retransmitir tramas de red entre distintas interfaces. Para comunicar máquinas virtuales ejecutándose en distintos hosts es necesario contar con herramientas más sofisticadas, que retransmitan paquetes de red de manera inteligente. Para ello se suelen utilizar soluciones SDN (*Software Defined Network*), que habilitan la programación de la configuración de red. *Open vSwitch* es un ejemplo de esta aproximación.

3.4.2.1. El bridge de Linux

El bridge de Linux está incluido en el kernel de Linux y permite interconectar distintos segmentos de red Ethernet retransmitiendo los paquetes en el nivel 2 de red. En la práctica, se puede considerar como un switch software que interconecta distintos dispositivos de red en un mismo host (ver figura 3.9).

A pesar de que el código del bridge de Linux está incluido en el kernel, es necesario instalar las herramientas de usuario que permiten operar con él:

```
> sudo apt install bridge-utils
```

Para revisar la configuración actual del bridge se utiliza el siguiente comando:

```
> sudo brctl show
```

Para crear o eliminar un nuevo bridge se utiliza los siguientes comandos:

```
> sudo brctl addbr <bridge>
> sudo brctl delbr <bridge>
```

Una vez creado, es posible conectar nuevos dispositivos de red, o bien desconectarlos, con los siguientes comandos:

```
> sudo brctl addif <bridge> <dev>
> sudo brctl delif <bridge> <dev>
```

Las soluciones de virtualización utilizan de manera habitual el bridge de Linux para interconectar localmente las máquinas virtuales. Por ejemplo, *libvirt* crea una red NAT por defecto utilizando precisamente el bridge de Linux.

```
> sudo brctl show
bridge name bridge id STP enabled interfaces
virbr0 8000.52540063f368 yes virbr0-nic
```

Si se arrancan distintos dominios en el mismo host, como se ilustra en la figura 3.10, se obtiene la siguiente configuración:

```
> sudo brctl show
bridge name bridge id     STP enabled   interfaces
virbr0 8000.52540063f368 yes       virbr0-nic
                  vnet0
                  vnet1
```

3.4.2.2. Open vSwitch

[Kreutz et al., 2014]

Open vSwitch[10] fue creado para cubrir las carencias del bridge de Linux. Open vSwitch implementa un switch virtual (software) inteligente, que trabaja en distintas capas de red y que permite la interconexión de máquinas virtuales ejecutándose en el mismo o diferentes hosts (ver figura 3.11). El switch puede ser fácilmente programado y extendido. Soporta múltiples tecnologías de virtualización basadas en Linux como Xen/XenServer, KVM y VirtualBox.

[10] https://www.openvswitch.org/

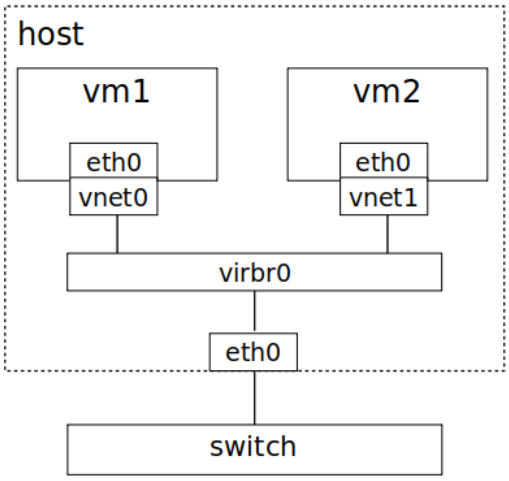

Figura 3.10: Libvirt y el bridge de Linux

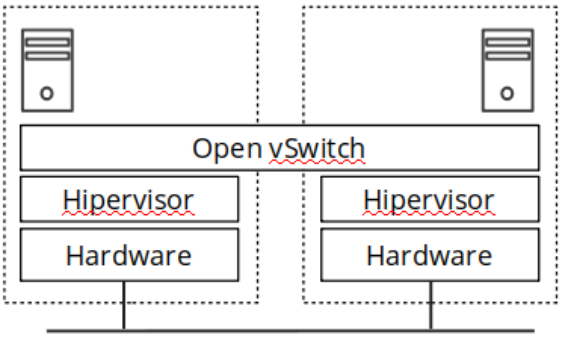

Figura 3.11: Open vSwitch

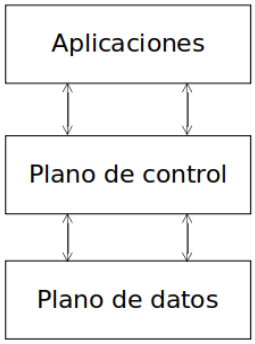

Figura 3.12: Modelo SDN (Software Defined Network)

Open vSwitch implementa con éxito el protocolo OpenFlow[11], un protocolo de comunicaciones que permite implantar un modelo SDN (Software Defined Network). Este modelo se caracteriza por desacoplar el plano de control del plano de datos. El plano de datos (o de redireccionamiento) incluye la infraestructura de red, ya sea física o virtual, que se encarga de retransmitir las tramas de red. El plano de control determina cuándo, dónde y cómo se retransmiten las tramas de red, y generalmente es implementado por uno o varios controladores. El controlador contacta con la infraestructura y la configura. Las aplicaciones pueden contactar con el controlador indicando sus preferencias o necesidades. La figura 3.12 ilustra este modelo.

Respecto a la arquitectura, Open vSwitch posee los siguientes componentes fundamentales (ver figura 3.13) :

- El módulo del kernel Open vSwitch: implementa el plano de datos como un módulo del kernel. Esta parte debe ser extremadamente rápida. Este es el motivo por el que está implementado como un módulo del kernel.
- Las herramientas de usuario: implementan el plano de control. Es posible identificar los siguientes elementos:
  - El demonio *ovs-vswitchd*, que implementa el switch.
  - El demonio *ovsbd-server*, que implementa un servidor de base de datos ligero que Open vSwitch consulta para obtener su configuración.
  - Los clientes *ovs-dpctl*, *ovs-vsctl*, *ovs-appctl*, que permiten enviar comandos al módulo del kernel y a los demonios de Open vSwitch, respectivamente.

[11] https://opennetworking.org/sdn-resources/customer-case-studies/openflow/

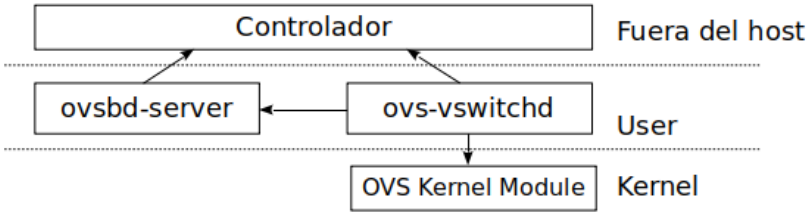

Figura 3.13: Arquitectura de Open vSwitch

Para instalar Open vSwitch se utiliza el siguiente comando:

```
> sudo apt install openvswitch-switch
```

Para comprobar si el switch está operativo:

```
> sudo systemctl status openvswitch-switch
```

Para obtener la configuración actual del switch:

```
> sudo ovs-vsctl show
```

Para añadir o eliminar dispositivos de red se utiliza el siguiente comando:

```
> sudo ovs-vsctl add-port <switch> <dev>
> sudo ovs-vsctl del-port <switch> <dev>
```

Por ejemplo, para conectar varias máquinas virtuales en el mismo host, y conseguir un resultado como el mostrado en la figura 3.14, se seguirían los siguientes pasos:

1. Se crea el switch en el host.

   ```
   > sudo ovs-vsctl add-br ovsbr
   > sudo ovs-vsctl list-br
   > sudo ovs-vsctl show
   ```

2. Se conecta las máquinas virtuales con el switch creado. En libvirt esto se puede conseguir añadiendo una nueva interfaz de red conectada al switch. La máquina virtual poseerá dos interfaces de red, una conectada a la red NAT `default` y otra conectada al nuevo switch virtual. Será necesario añadir a la plantilla xml de creación del dominio un fragmento similar al siguiente:

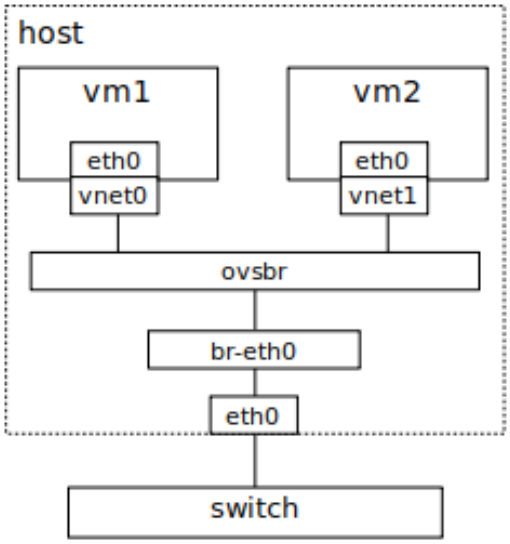

Figura 3.14: Varias máquinas virtuales en el mismo host

```
<interface type='bridge'>
  <source bridge='ovsbr'/>
  <virtualport type="openvswitch" />
</interface>
```

3. Por último, es necesario configurar la nueva interfaz en el interior del dominio. A priori no existe ningún servidor DHCP que asigne la configuración de red de manera automática. Por tanto, es necesario hacerlo de manera manual, por ejemplo se puede editar el fichero `/etc/network/interfaces` añadiendo la siguiente configuración:

   ```
   auto <iface>
   iface <iface> inet static
   address x.x.x.x
   netmask 255.255.255.0
   gateway x.x.x.x
   ```

Para obtener los puertos de conexión con el switch se pueden utilizar los siguientes comandos:

```
> sudo ovs-vsctl list-ports ovsbr
> sudo ovs-ofctl show ovsbr
```

Para eliminar el switch se utiliza el comando:

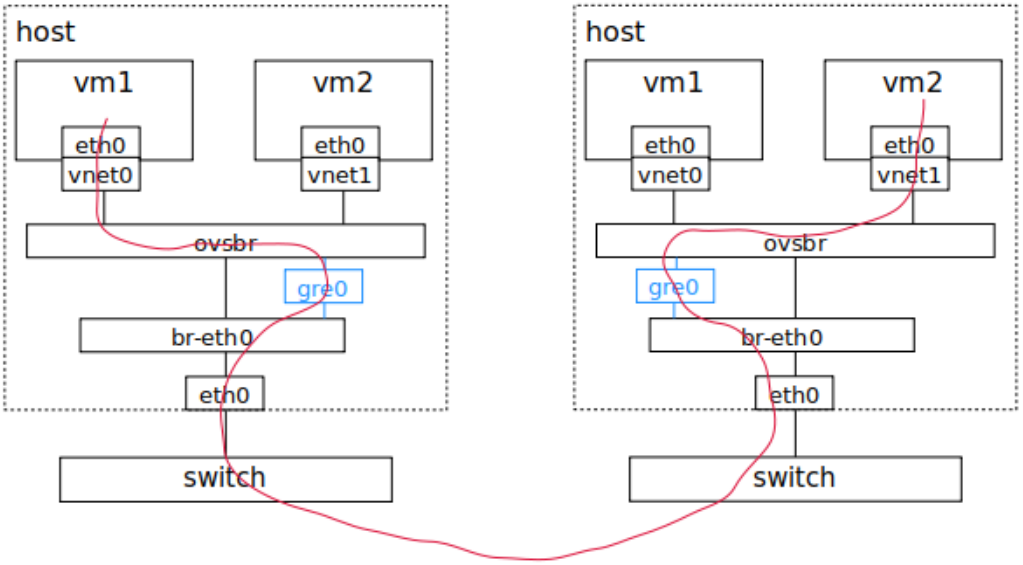

Figura 3.15: Varias máquinas virtuales en distintos hosts

```
> sudo ovs-vsctl del-br ovsbr
```

Para interconectar máquinas virtuales ejecutándose en distintos hosts es necesario utilizar túneles. De este modo, cada host dispone de un switch virtual al que se conectan las máquinas virtuales ejecutándose en dicho host. Uno de los puertos de los switch se utiliza para interconectarlos por medio de túneles. La figura 3.15 ilustra esta configuración. Para añadir puertos de esta naturaleza se utilizan comandos similares al siguiente:

```
> sudo ovs-vsctl add-port ovsbr gre0 \\
 -- set interface gre0 type=gre options:remote_ip=<IP of eth0 on host2>
```

3.4.3. Virtualización del almacenamiento

En una infraestructura virtual, el espacio de almacenamiento resulta útil por diversos motivos:

- Las imágenes a partir de las que se crean las máquinas virtuales deben residir en algún lugar.
- Las máquinas virtuales requieren un espacio de almacenamiento primario, en el que reside el sistema operativo y las aplicaciones.
- Algunas aplicaciones podrían requerir espacio de almacenamiento adicional, para almacenar datos de manera persistente.

El espacio de almacenamiento físico puede proceder de diversas fuentes:

- De los nodos físicos que componen el centro de datos.
- De soluciones de almacenamiento dedicadas, tipo NAS (*Network Attached Storage*) o SAN (*Storage Area Network*).

La virtualización del almacenamiento es una técnica que permite abstraer todos los recursos de almacenamiento físico bajo un único pool de almacenamiento lógico. En general es posible identificar dos tipos de almacenamiento:

- Almacenamiento de tipo bloque: el almacenamiento se monta como una unidad de disco, y es accedido a través de bloques de bytes.
- Almacenamiento de tipo fichero: el almacenamiento es accedido como un directorio compartido, y los ficheros son transferidos a través de la red.

3.4.3.1. NFS

NFS (Network File System) es un protocolo de compartición de ficheros, que permite a un cliente acceder a ficheros remotos del mismo modo que lo haría a un fichero local. Se trata de un protocolo cliente-servidor.

En el servidor se instala el siguiente paquete:

```
> sudo apt install nfs-kernel-server
```

En el ordenador cliente se instalan los siguientes paquetes:

```
> sudo apt install rpcbind nfs-common
```

El procedimiento es sencillo, el servidor exporta un directorio, y los clientes lo montan como un sistema de ficheros local. Para exportar el directorio, el servidor debe incluir una nueva línea en el fichero `/etc/exports` con la sintaxis:

```
<dir-remoto> <cliente>(opción1, ... opciónN)
```

Por ejemplo:

```
/dir-remoto  ip-ordenador-cliente(rw,sync,no_root_squash,no_subtree_check)
```

Si el directorio exportado no existe previamente, es recomendable cambiar su propietario por `nobody:nogroup`.

```
> sudo chown nobody:nogroup /dir-remoto
```

A continuación se explican algunas opciones típicas:

- `rw`: El cliente puede leer y escribir.
- `sync`: NFS escribe los cambios en disco antes de responder.
- `no_subtree_check`: evita que en cada petición se compruebe que el fichero está disponible.
- `no_root_squash`: por defecto las peticiones del root desde el cliente se traducen a peticiones del usuario no privilegiado nobody:nogroup en el servidor. Esta opción deshabilita este comportamiento.

Una vez incluida la configuración en el fichero `/etc/exports` es necesario ejecutar el siguiente comando:

```
> sudo exportfs -a
```

El servidor NFS debe ser iniciado:

```
> sudo service nfs-kernel-server start
> sudo systemctl restart nfs-kernel-server
```

En el cliente, se crea un directorio y se monta en él el directorio remoto:

```
> mkdir dir-local
> sudo mount ip-del-servidor:/dir-remoto /dir-local
```

Se puede observar si está montado con:

```
mount -t nfs
```

Para que el directorio remoto se monte automáticamente al iniciar el sistema operativo es necesario modificar el fichero `/etc/fstab`:

```
ip-server:/dir-remoto /dir-local nfs auto,noatime,nolock,bg,nfsvers=4,intr,tcp,actim
```

Para desmontar el directorio remoto:

```
> sudo umount /dir-local
```

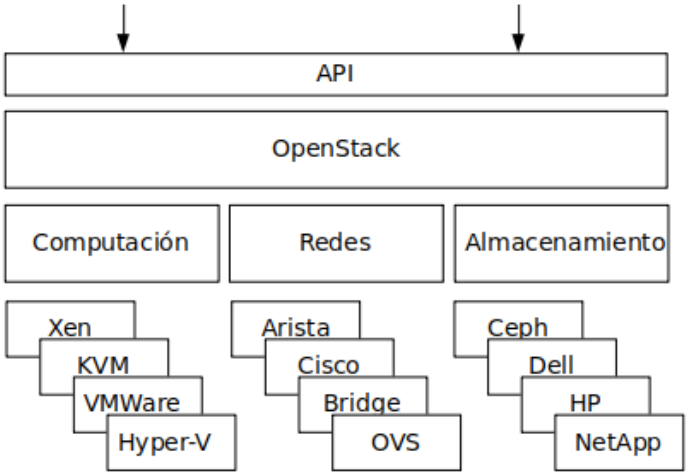

Figura 3.16: OpenStack proporciona una API homogénea

3.5. Estudio de un IaaS: OpenStack

OpenStack[12] es un framework open source que permite construir clouds públicos y privados a partir de un pool de recursos de computación, almacenamiento y redes. Es habitual definirlo como un sistema operativo cloud, pues permite gestionar todos estos recursos de manera óptima y transparente para el consumidor.

Algunas compañías que han utilizado OpenStack como base para construir su IaaS público son Rackspace, Red Hat, Dell, HP, IBM, Cisco, Canonical, ...

OpenStack abstrae los recursos hardware y software que configuran el cloud y proporciona una API común para gestionarlos. Para ello, en general OpenStack no implementa nuevas tecnologías, sino que hace uso de tecnologías existentes, ya sean open source o propietarias, y crea una capa de abstracción que permite gestionarlas de manera homogénea, desacoplando al consumidor de las tecnologías y recursos subyacentes. Esta aproximación permite sustituir o reemplazar tecnologías y recursos a bajo nivel, sin que ello tenga impacto sobre el consumidor final. La figura 3.16 ilustra esta configuración.

Respecto a los recursos computacionales, OpenStack soporta múltiples hipervisores[13], como Xen, KVM, QEMU, LXC, ESXi, Hyper-V, etc. Respecto a los recursos de red, OpenStack puede gestionar diversas tecnologías, como Arista Networks, Cirsco Nexus, Linux bridging, Open vSwitch (OVS), etc. Respecto a los recursos de almacenamiento, OpenStack gestiona almacenamiento de tipo bloque y almacenamiento de objetos. El primero se utiliza para dotar de sistemas de ficheros para las máquinas virtuales. Por lo tanto, la latencia de este tipo de

[12] https://www.openstack.org/
[13] https://wiki.openstack.org/wiki/HypervisorSupportMatrix

almacenamiento debe de ser baja. OpenStack aprovecha soluciones de terceros como Ceph, Dell, EMC, HP, IBM, NetApp, etc. Respecto al almacenamiento de objetos, OpenStack implementa su propio motor. Este motor es utilizado de manera habitual para almacenar grandes cantidades de datos, como backups de volúmenes, o imágenes de máquinas virtuales. En este tipo de almacenamiento los requisitos en cuanto a latencia no son tan exigentes.

3.5.1. Arquitectura de OpenStack

OpenStack está diseñado como un sistema modular que se compone de múltiples servicios[14]. Cada servicio es responsable de ciertas tareas bien delimitadas. Algunos de estos servicios son clave para el correcto funcionamiento del sistema, y reciben la denominación de servicios core, otros son considerados extensiones. Cada escenario particular impone unas necesidades diferentes, y por tanto puede requerir la instalación de diferentes servicios.

Cada servicio se gestiona como un proyecto diferente, con un nombre en clave distinto, y puede ser instalado y configurado de manera independiente. A continuación se listan los proyectos más relevantes que configuran una instalación de OpenStack, junto con su nombre en clave:

- *Identity* (*Keystone*): gestiona los usuarios del sistema, sus permisos y registra los distintos servicios del sistema.
- *Compute* (*Nova*): gestiona las instancias (máquinas virtuales).
- *Image* (*Glance*): gestiona las imágenes
- *Dashboard* (*Horizon*): proporciona una GUI web
- *Network* (*Neutron*): gestiona las redes que interconectan las instancias.
- *Block Storage* (*Cinder*): gestiona los volúmenes (discos virtuales).
- *Object Storage* (*Swift*): proporciona un almacén persistente de objetos.
- *Telemetry* (*Ceilometer*): obtiene las métricas del sistema.
- *Orchestration* (*Heat*): gestiona composiciones de instancias.

Cada servicio implementa su propia API y proporciona herramientas CLI que permiten interaccionar con él. Los servicios colaboran entre sí a través de dichas APIs para ofrecer las funcionalidades requeridas. En la figura 3.17 se ilustra la relación que existe entre servicios.

[14] https://docs.openstack.org/arch-design/

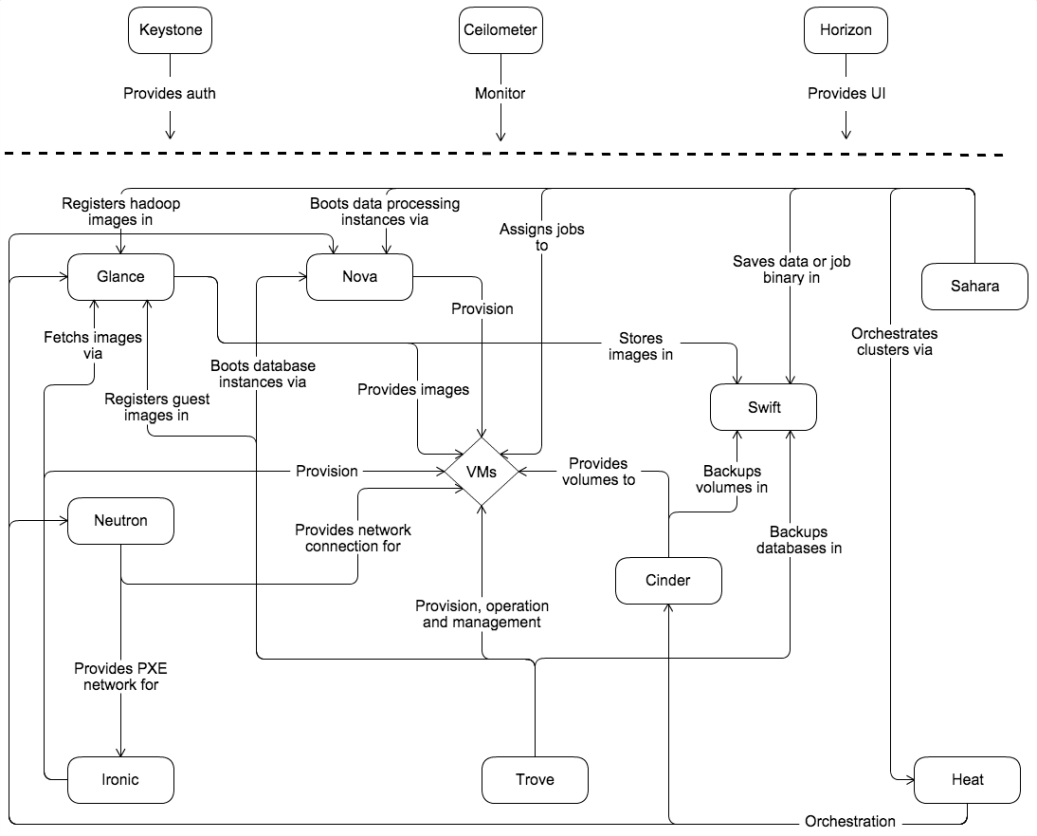

Figura 3.17: Relación de servicios OpenStack

Existe gran flexibilidad a la hora de distribuir estos servicios en distintos nodos. Una manera habitual de hacerlo consiste en agruparlos en un nodo denominado *controlador*. Este nodo suele estar replicado para habilitar alta disponibilidad y gestiona todos los recursos del sistema. El resto de nodos proporcionan los distintos recursos físicos. En este sentido, es habitual disponer de nodos que proporcionan recursos de computación (*Compute*), de almacenamiento (*Block Storage*) y redes (*Network*).

Esta cuestión es ampliamente configurable, ya que los recursos pueden ser implementados de muy diversas maneras. En ocasiones, el recurso es creado y gestionado por el propio servicio. En otras ocasiones, el servicio únicamente recubre una solución de un tercero (proveedores o *vendors*). Por ejemplo, en el caso de *Compute* (*Nova*), el servicio puede delegar la creación de instancias a distintos hipervisores como Xen, KVM, Hyper-V, etc. En el caso de *Block Storage* (*Cinder*) el almacenamiento puede ser proporcionado a través de distintas soluciones iSCSI, NFS, VMWare, etc. En este sentido, OpenStack permite gestionar colecciones de recursos respaldados por muy diversas tecnologías de manera homogénea (ver figura 3.16). Si la tecnología es actualizada, ampliada o reemplazada el consumidor de los servicios no tiene por qué verse afectado.

En la figura 3.18 se muestra una configuración sencilla[15] con cuatro nodos. Además del nodo controlador que contiene los servicios fundamentales, es posible identificar otros tres nodos que proporcionan recursos al sistema: uno de ellos proporciona computación, y los otros dos almacenamiento, bien en forma de bloque o en forma de repositorio de objetos.

La figura 3.19 muestra una configuración más seria[16], con múltiples nodos proporcionando gran cantidad de recursos.

3.5.2. Instalación

En esta sección se procederá a instalar una versión de desarrollo/test de OpenStack, a través del paquete *DevStack*[17]. *DevStack* comprende una colección de scripts y ficheros de configuración que automatizan la instalación de los componentes principales de OpenStack en una única máquina (física o virtual). Para ello, es necesario disponer de una instalación de Linux Ubuntu o CentOS/RHEL, y un mínimo de 5,5GB de memoria.

El primer paso consiste en contar con una máquina (física o virtual) que contenga un sistema operativo compatible con OpenStack (*Ubuntu latest LTS*,

[15] https://docs.openstack.org/newton/install-guide-ubuntu/overview.html

[16] https://docs.openstack.org/arch-design/use-cases/use-case-general-compute.html

[17] https://docs.openstack.org/devstack/latest/

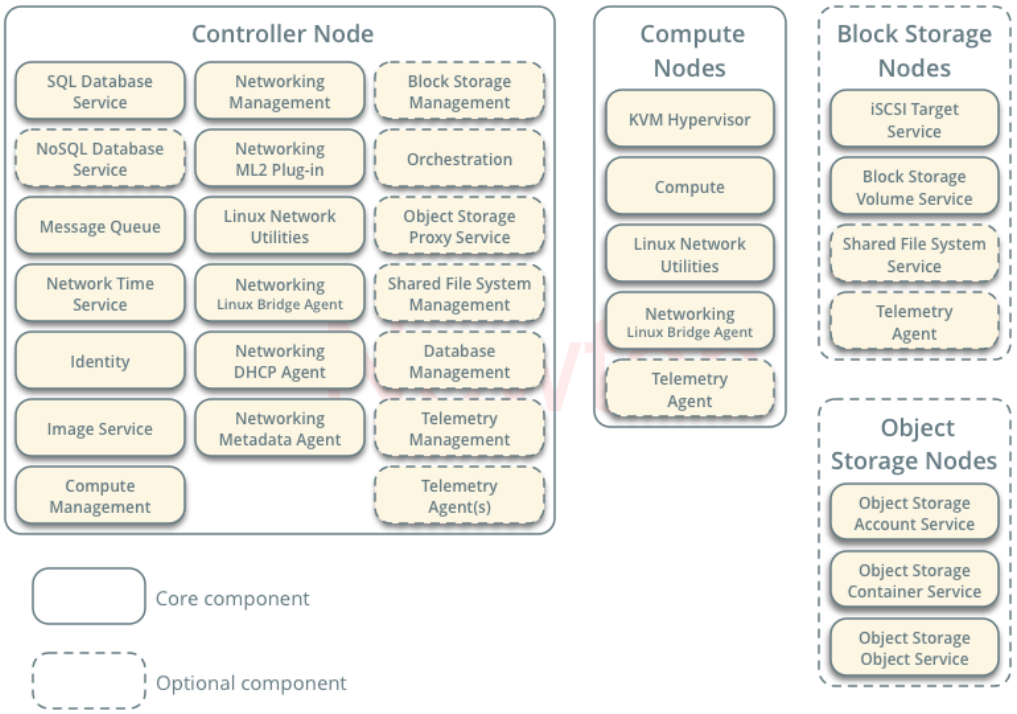

Figura 3.18: Configuración simple de OpenStack

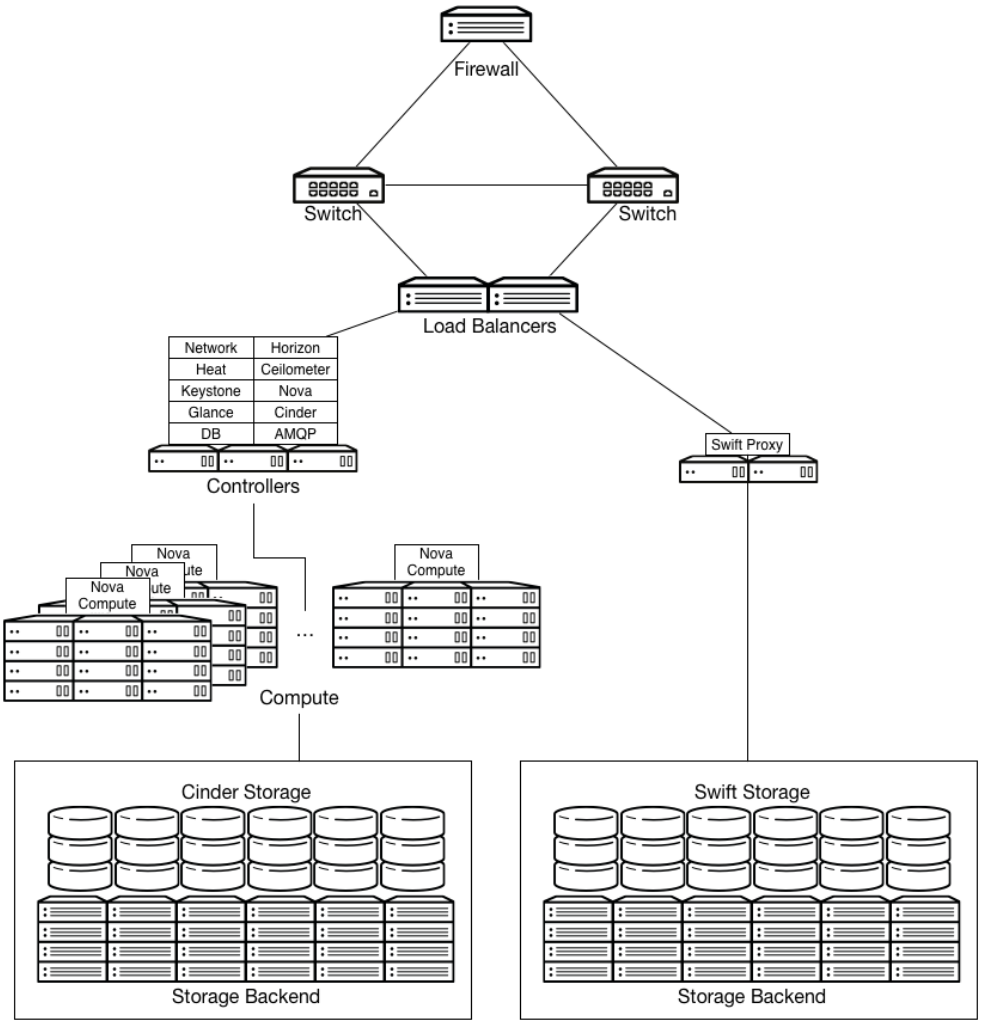

Figura 3.19: Configuración compleja de OpenStack

Fedora, *CentOS/RHEL 8* o *OpenSUSE*). En las instrucciones que siguen se asume que se parte de una instalación de *Ubuntu Server 20.04*.

A continuación es necesario instalar GIT.

```
> sudo apt-get install -y git
```

El siguiente paso consiste en añadir el usuario `stack`, y asignarle privilegios `sudo`.

```
> sudo useradd -s /bin/bash -d /opt/stack -m stack
> echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
> sudo su - stack
```

A continuación se clona el repositorio de DevStack.

```
> git clone https://opendev.org/openstack/devstack
> cd devstack
```

Cuando se instala OpenStack, DevStack utiliza las opciones de configuración especificadas en el fichero `local.conf`[18]. El siguiente paso consiste en crear dicho fichero. Inicialmente, se creará con el siguiente contenido:

```
[[local|localrc]]
ADMIN_PASSWORD=devstack
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
HOST_IP=<ip-host>
```

En este ejemplo se fija `devstack` como password de todos los servicios. Es importante reemplazar el contenido de la variable `HOST_IP` por la dirección IP de la máquina en la que se desea instalar OpenStack.

A continuación se comienza con la instalación del sistema:

```
> ./stack.sh
```

Este proceso puede durar varias horas, dependiendo de la potencia de la máquina y de la velocidad de conexión.

Una vez instalado el sistema, si se reinicia, será necesario volver a configurarlo. Para ello, sería necesario seguir las siguientes instrucciones:

```
> sudo su - stack
> cd devstack
> ./unstack.sh
> ./clean.sh
> ./stack.sh
```

[18] https://docs.openstack.org/devstack/latest/configuration.html

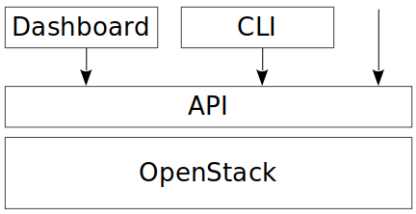

Figura 3.20: Mecanismos de acceso a OpenStack

3.5.3. Acceso a OpenStack

Una vez se dispone de una instalación de OpenStack es posible acceder a ésta utilizando tres vías:

- A través del Dashboard (*Horizon*). Se trata de una GUI web que tras un proceso de autenticación permite acceder a gran parte de las funcionalidades de OpenStack.
- A través de la herramienta CLI `openstack`. Se trata de una utilidad en línea de comandos que permite acceder a todas las funcionalidades de OpenStack.
- A través de la API RESTful publicada por OpenStack. Es el mecanismo de acceso primario a OpenStack. Las herramientas anteriores al final acceden a esta API.

La figura 3.20 ilustra la relación que existe entre estos tres mecanismos de acceso. Como se puede observar, la API publicada por OpenStack es consumida tanto por Dashboard como por la herramienta CLI.

3.5.4. Dashboard

El Dashboard representa un mecanismo de acceso a OpenStack muy adecuado para usuarios finales. Se trata de una GUI de muy sencilla gestión, que proporciona herramientas gráficas para monitorizar y utilizar gran parte de las funcionalidades ofrecidas por el sistema.

Para acceder al Dashboard únicamente es necesario introducir la URL correspondiente en el navegador. Aparecerá un formulario de autenticación donde el usuario deberá suministrar sus credenciales (ver figura 3.21)

Una vez suministradas las credenciales, el usuario tendrá acceso a un gran número de funcionalidades (ver figura 3.22), que se explorarán en las siguientes secciones.

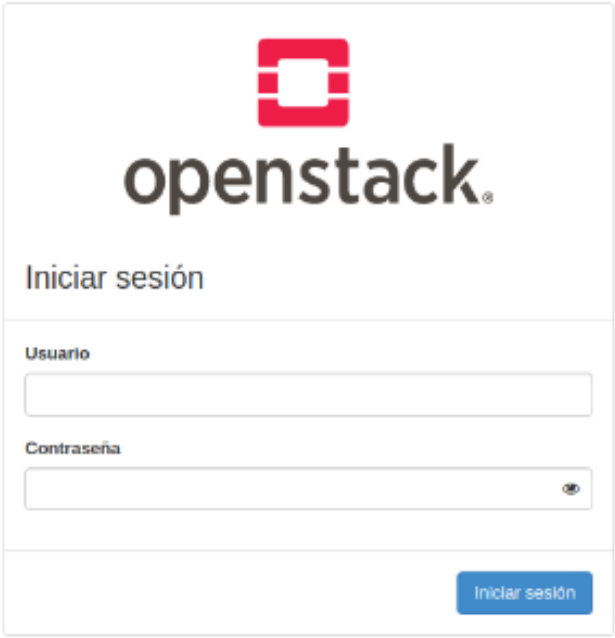

Figura 3.21: Autenticación

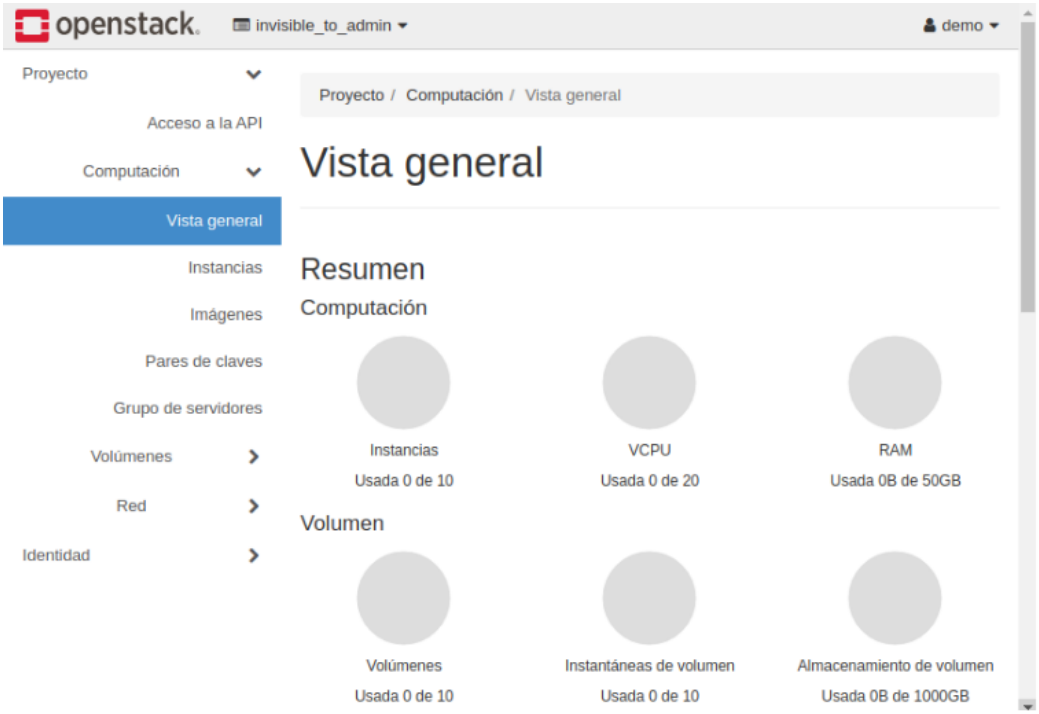

Figura 3.22: Vista general

Componente	Herramienta CLI	Paquete Python	Descripción
Autenticación	`keystone`	`python-keystoneclient`	Gestiona identidades
Computación	`nova`	`python-novaclient`	Gestiona las instancias
Almacenamiento	`cinder`	`python-cinderclient`	Gestiona discos
Imágenes	`glance`	`python-glanceclient`	Gestiona imágenes
Redes	`neutron`	`python-neutronclient`	Gestiona redes
Métricas	`ceilometer`	`python-ceilometerclient`	Monitorizar consumo recursos

Cuadro 3.4: Herramientas CLI OpenStack

3.5.5. Herramienta CLI

La herramienta CLI `openstack` es un paquete Python (`python-openstackclient`) que se utiliza en línea de comandos y que proporciona acceso a todas las funcionalidades del sistema. Favorece la automatización de tareas repetitivas por medio de scripting y resulta por tanto resulta adecuada para administradores de sistemas.

En realidad, esta herramienta es un recubrimiento completo de la API RESTful de OpenStack. Internamente, cada comando utiliza la herramienta `curl` para efectuar una petición HTTP sobre la API de OpenStack.

Además de la herramienta CLI openstack, cada componente de OpenStack dispone de su propia herramienta CLI, que permite acceder a las funcionalidades específicas de dicho componente en cuestión. En la tabla 3.4 se listan algunas de ellas.

Para utilizar la herramienta CLI primero es necesario instalarla[19]. Es posible hacerlo utilizando el gestor de paquetes del sistema operativo o bien utilizando `pip`:

```
> sudo apt-get install python-openstackclient
> pip install python-openstackclient
```

Para comprobar que la herramienta ha sido correctamente instalada:

```
> openstack --version
```

[19] https://docs.openstack.org/ocata/user-guide/common/cli-install-openstack-command-line-clients.html

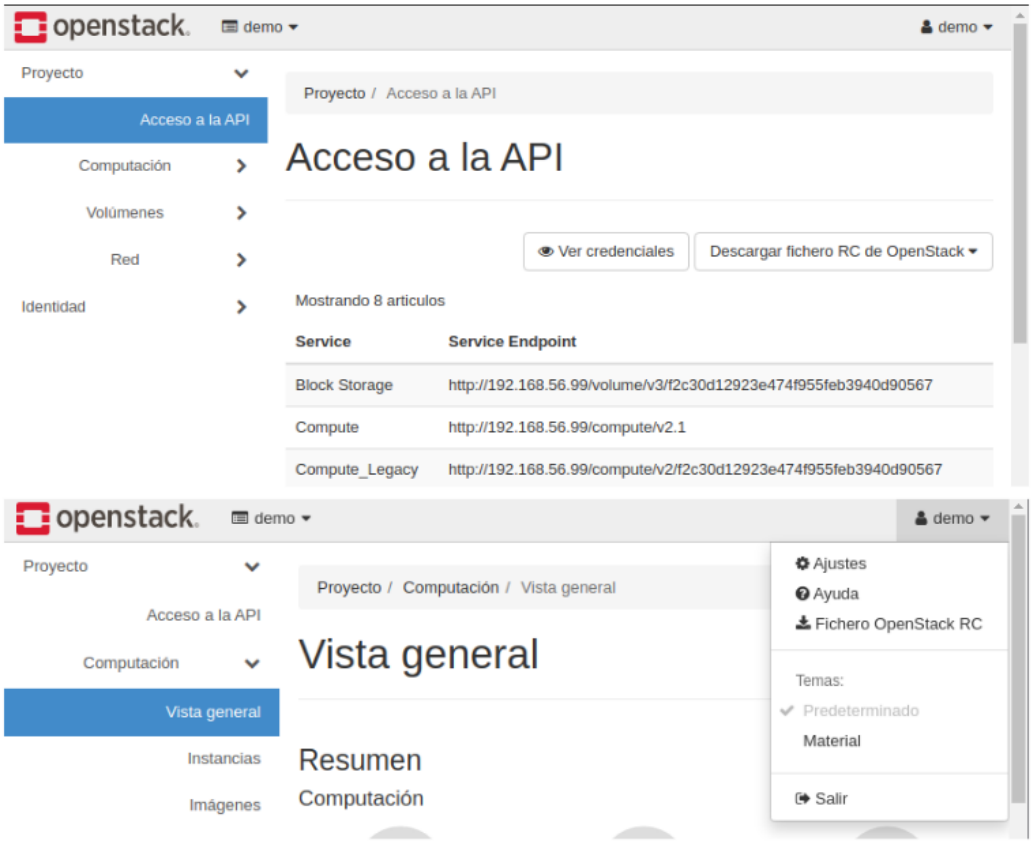

Figura 3.23: Obtener script *openrc.sh*

Para que la herramienta CLI pueda acceder a OpenStack primero es necesario proporcionar las credenciales del usuario. Esto se efectúa por medio de una colección de variables de entorno como `OS_USERNAME`, `OS_PASSWORD`, etc. Para facilitar la configuración de estas variables de entorno, es habitual definir un script `openrc.sh`. Una manera sencilla de obtener este fichero es a través de *Dashboard*, a través de la ruta "Acceso a la API" o bien a través de las opciones del usuario autenticado (ver figura 3.23).

A continuación es necesario ejecutar dicho comando con source, e introducir el password del usuario:

```
> source openrc.sh
```

Si la operación tiene éxito, será posible ejecutar cualquier comando soportado por la herramienta CLI.

```
> openstack project list
```

```
> openstack image list
> openstack help
```

Se recomienda acceder a la página web oficial[20] para revisar el listado de comandos disponibles. Debido a la gran cantidad de opciones disponibles, resulta común tener a mano algún documento de referencia rápida[21].

3.5.6. API RESTful

Es el mecanismo de acceso primario a todas las funcionalidades de OpenStack. Cada componente de OpenStack publica un subconjunto de esta API[22]. Por lo tanto, para ejecutar un determinado comando primero hay que localizar a dicho componente y posteriormente redirigir la petición en cuestión.

Es perfectamente posible acceder directamente a esta API utilizando herramientas que permiten efectuar peticiones HTTP, como por ejemplo curl.

Asumiendo que OpenStack está escuchando en la URL http://127.0.0.1:8080, en el siguiente ejemplo se listan todas las imágenes disponibles para el usuario demo:

```
> curl -s -X POST http://127.0.0.1:8080/image \\
  -d '{"auth": {"passwordCredentials": {"username": "demo", "password":"devstack"},'
  -H "Content-type: application/json"
```

Esta facilidad de acceso a la API de OpenStack permite la implementación de librerías (bindings) en múltiples lenguajes de programación. Por ejemplo, en Python resulta muy sencillo trabajar con el *OpenStack Python SDK*[23]. Se trata de una colección de paquetes que permiten trabajar con objetos Python, en lugar de lidiar directamente con las llamadas a la API REST.

3.5.7. El modelo OpenStack

En OpenStack los recursos son accedidos por una colección de usuarios. El usuario `admin` representa al administrador del sistema y posee privilegios especiales. El resto de usuarios son creados por `admin` y a priori no poseen privilegios especiales. En una instalación a través de DevStack es habitual contar con el usuario `admin` y un usuario de prueba `demo`.

[20] https://docs.openstack.org/python-openstackclient/latest/cli/index.html

[21] https://docs.openstack.org/ocata/user-guide/cli-cheat-sheet.html

[22] https://docs.openstack.org/api-quick-start/

[23] https://docs.openstack.org/mitaka/user-guide/sdk.html

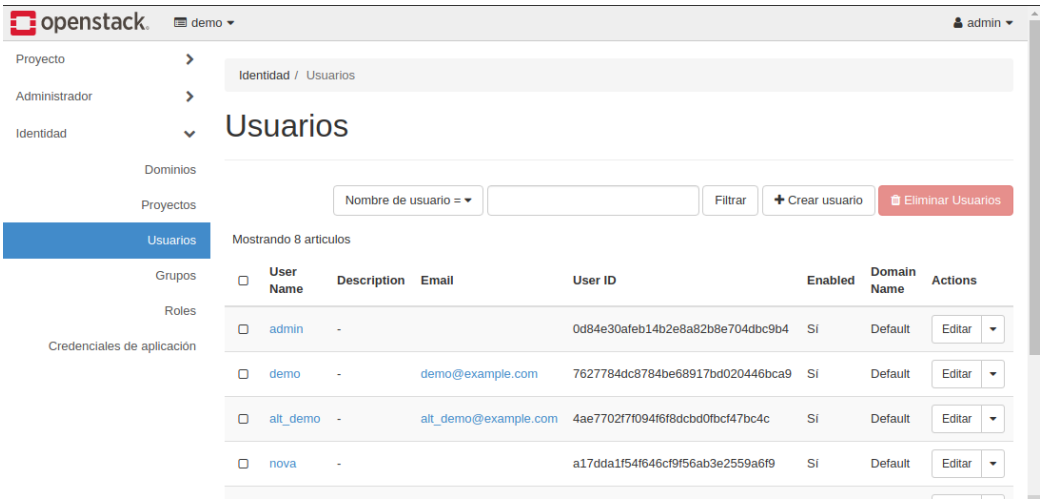

Figura 3.24: Gestionar usuarios

Para gestionar los usuarios del sistema es necesario autenticarse como `admin`. Desde Dashboard se puede acceder a través de la ruta `Identidad > Usuarios` (figura 3.24).

Con la herramienta CLI se utilizan los comandos:

```
> openstack user list
> openstack user create --project <project> --passsword <password> <new-user>
> openstack user delete <user-name>
```

Todos los recursos gestionados por OpenStack se configuran en base a *tenants* o *proyectos*. Un *proyecto* puede ser considerado un espacio o compartimento estanco al que se pueden asignar diversas cuotas o *límites máximos* de los recursos que puede utilizar un conjunto de usuarios.

Para gestionar todos los proyectos existentes en el sistema es necesario autenticarse como `admin`. En Dashboard, en el panel lateral se accede a la ruta `Identidad > Proyectos`. Desde esta sección es posible obtener toda la información sobre los proyectos, tal y como se ilustra en la figura 3.25.

En la herramienta CLI se pueden gestionar los proyectos con los siguientes comandos:

```
> openstack project list
> openstack project show <project>
> openstack project create <project-name>
> openstack project delete <project-name>
```

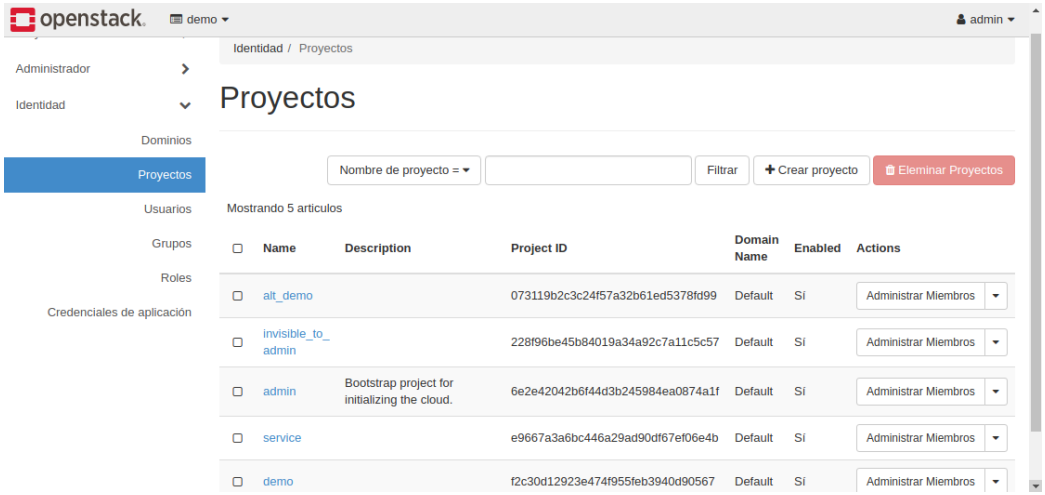

Figura 3.25: Gestionar proyectos

Cada usuario dispone de un proyecto primario, y puede ser miembro de proyectos adicionales. Los usuarios son añadidos a los proyectos, con distintos niveles de privilegios, denominados *roles*. Roles habituales en cualquier instalación son *Member* (el rol común) y *admin*, éste último con privilegios especiales.

Es posible listar todos los roles en Dashboard, a través de la ruta `Identidad > Roles`. En la figura 3.26 se puede observar esta sección.

Con la herramienta CLI se utilizan los comandos:

```
> openstack role list [--user <user-name>] [--project <project-id>]
> openstack role show <role-name>
> openstack role create <role-name>
> openstack role add --user <user-name> --project <project-id> <role-name>
> openstack role remove --user <user-name> --project <project-id> <role-name>
```

Cada proyecto dispone de una asignación máxima de recursos. Estos límites reciben la denominación de *cuotas* y son fácilmente configurables. Cuando se crea un proyecto, se aplican ciertas cuotas por defecto, que pueden ser modificadas bien en el proceso de creación o bien en un momento posterior.

Algunos ejemplos de cuotas serían:

- Número de CPUs (vCPUs) virtuales utilizadas
- Máxima memoria RAM utilizada
- Número de instancias (máquinas virtuales) creadas
- Número de volúmenes (discos virtuales) creados y tamaño total de estos

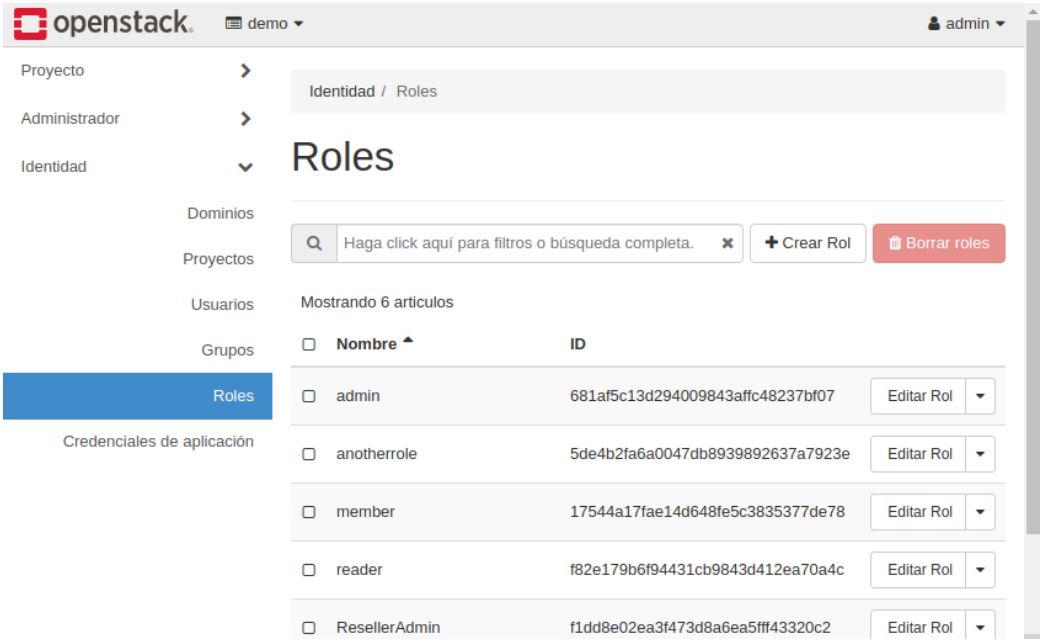

Figura 3.26: Gestionar roles

- Número de redes, puertos, routers, subredes
- etc.

OpenStack contabiliza todos los recursos consumidos por un proyecto y comprueba que estos límites no sean excedidos. En este sentido, un proyecto podría ser considerado como un pool de recursos disponibles para un grupo de usuarios (un único usuario, un departamento de una compañía, etc.).

Las cuotas pueden ser aplicadas también sobre usuarios concretos dentro de un proyecto. De este modo, las cuotas del proyecto pueden ser repartidas entre sus miembros de acuerdo a distintas políticas.

Las cuotas aplicadas a un proyecto están disponibles en Dashboard bajo la ruta `Identidad > Proyectos`. Tras seleccionar un proyecto es posible manipular sus cuotas, tal y como se muestra en la figura 3.27.

También es posible acceder a dichas cuotas utilizando la herramienta CLI:

```
> openstack quota list|show|set --project <proyect-id/name>
> openstack quota show --project <proyect-id/name>
> openstack quota set <project-id/name> -parameters
```

Respecto a los recursos de computación, OpenStack permite gestionar múltiples *instancias*. Cada instancia es una máquina virtual que puede ser ejecutada en un hipervisor diferente.

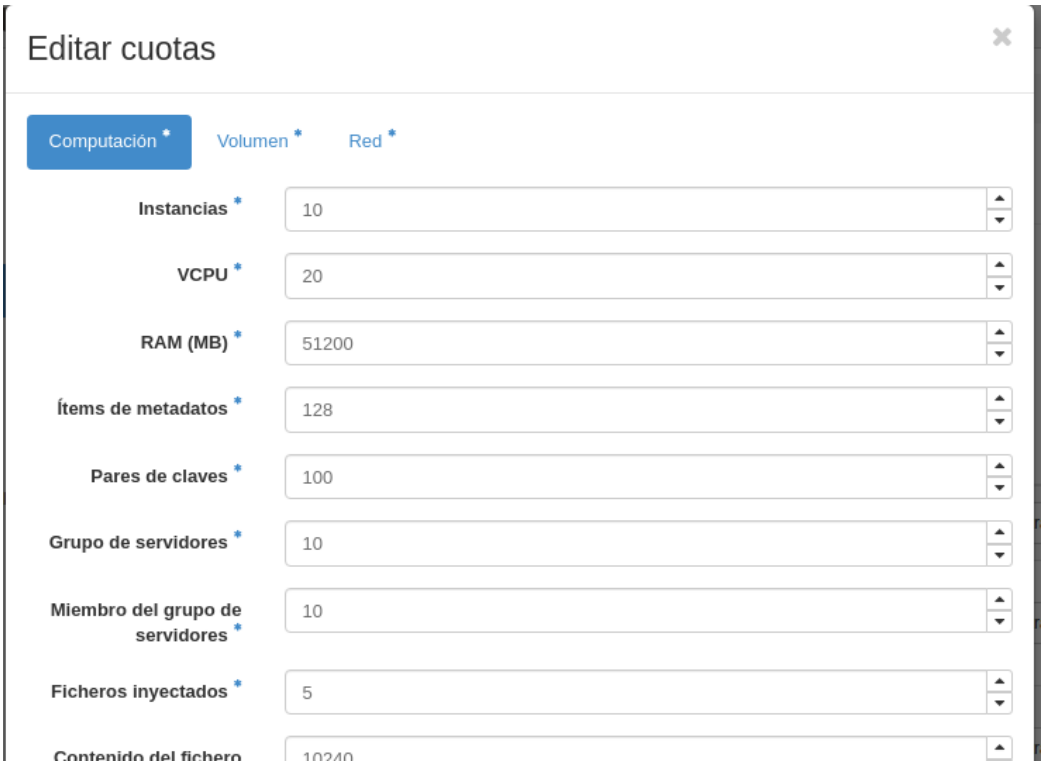

Figura 3.27: Gestionar cuotas

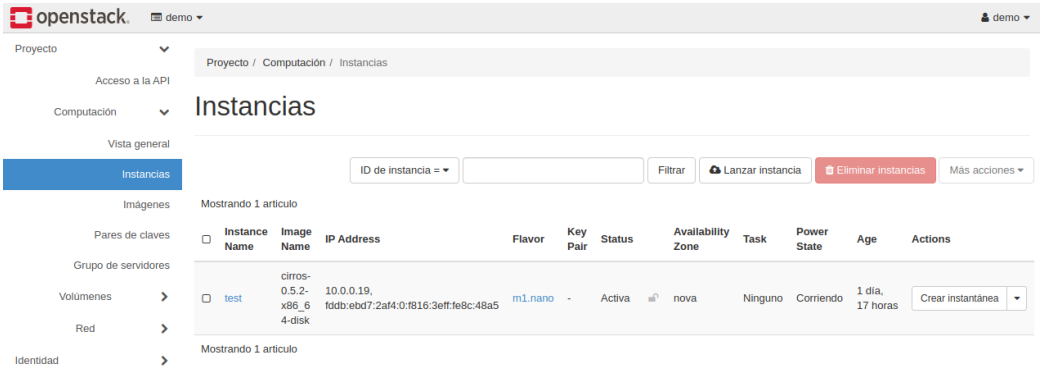

Figura 3.28: Gestionar instancias

Las instancias se gestionan en Dashboard a través de la ruta `Proyecto > Computación > Ins` tal y como se muestra en la figura 3.28.

Para gestionar las instancias a través de la herramienta CLI se utilizan los siguientes comandos:

```
> openstack server create|list|show|delete|start|stop|pause|suspend ...
```

Una instancia se crea a partir de una plantilla, que puede ser una *imagen*, una instantánea (*snapshot*) de una instancia, un *volumen* o una instantánea de un volumen. Cuando se crea la instancia, se crea una copia de la plantilla. Para esto se utilizan diversos mecanismos, dependiendo del tipo de plantilla utilizada. La plantilla original permanece inmutable. Por defecto, la copia de la plantilla es provisional, esto es, se destruye automáticamente cuando la instancia se destruye. Es posible crear un volumen nuevo a partir de dicha copia. En este caso, el almacenamiento utilizado sería permanente.

A cada instancia se le asignan una serie de recursos máximos, que incluyen el número de CPUs virtuales o vCPUs, la RAM y el tamaño del disco. Esta asignación de recursos se efectúa a través de *sabores* (*flavours*). OpenStack viene preconfigurado con una colección de sabores. En la tabla 3.5 se listan algunos de ellos.

Una vez arrancada la instancia, es posible acceder a la misma a través de la consola. Será necesario introducir las credenciales de un usuario válido.

En todo momento se puede efectuar una instantánea de la instancia en ejecución. Esta instantánea captura el estado de la instancia y puede ser recuperado en un instante posterior.

Es habitual crear instancias a partir de una *imagen*. Una imagen es un paquete que contiene el sistema operativo y todo el software necesario para ejecutar una instancia. OpenStack soporta múltiples formatos de imagen como RAW, VHD, VMDK, VDI, ISO, QCOW, AKI, ARI, AMI ...

Nombre	vCPUs	RAM	Disco
m1.nano	1	128MB	1GB
m1.micro	1	192MB	1GB
m1.tiny	1	512MB	1GB
m1.small	1	2GB	20GB
m1.medium	2	4GB	40GB
m1.large	4	8GB	80GB
m1.xlarge	8	16GB	160GB

Cuadro 3.5: Sabores predefinidos en OpenStack

Las imágenes pueden ser creadas de manera manual, pero también es posible obtenerlas a partir de múltiples proveedores de sistemas operativos[24] como por ejemplo Debian[25], Ubuntu[26] o CentOS[27]. DevStack instala a modo de ejemplo imágenes muy livianas que poseen el sistema operativo CirrOS. Estas imágenes permiten efectuar pruebas sobre el sistema, pero no están pensadas para un uso en producción. De hecho, CirrOS no dispone de gestor de paquetes que facilite la instalación de nuevo software.

También es posible crear una imagen a partir de una instantánea de una instancia. De este modo, resulta sumamente sencillo construir imágenes personalizadas: se crea una instancia, se configura con el software requerido, se crea una instantánea de la instancia y finalmente se transforma en imagen; a partir de ese instante será posible crear cualquier número de instancias utilizando como origen la nueva imagen.

En Dashboard es posible gestionar todas las imágenes disponibles a través de la ruta `Proyecto > Computación > Imágenes`. Se muestra esta sección en la figura 3.29.

A través de la herramienta CLI se utilizan los comandos:

```
> openstack image list/show/create/delete/...
```

Para poder acceder a una máquina virtual a nivel de usuario es necesario contar con ciertas cuentas en el sistema operativo. Estas cuentas pueden estar preconfiguradas en la imagen o pueden ser añadidas dinámicamente en el proceso de creación de la instancia. Para ello, OpenStack permite la creación de pares de claves, de manera que la clave pública puede ser inyectada automáticamente en la

[24] https://docs.openstack.org/image-guide/obtain-images.html
[25] http://cdimage.debian.org/cdimage/openstack/
[26] https://cloud-images.ubuntu.com/
[27] http://cloud.centos.org/centos/

Figura 3.29: Gestionar imágenes

Figura 3.30: Gestionar claves

instancia creada. Entonces cualquier usuario podrá conectarse a dicha instancia utilizando la clave privada.

En Dashboard es posible gestionar todas las imágenes disponibles a través de la ruta `Proyecto > Computación > Pares de claves`. Se muestra esta sección en la figura 3.30.

A través de la herramienta CLI se utilizan los comandos:

```
> openstack keypair list/show/create/delete/...
```

Por defecto las instancias creadas son efímeras. Esto significa que al crear la instancia se crea una copia de su imagen y esta copia constituye el espacio de almacenamiento temporal de la instancia. Cuando la instancia se destruye también se destruye su espacio de almacenamiento.

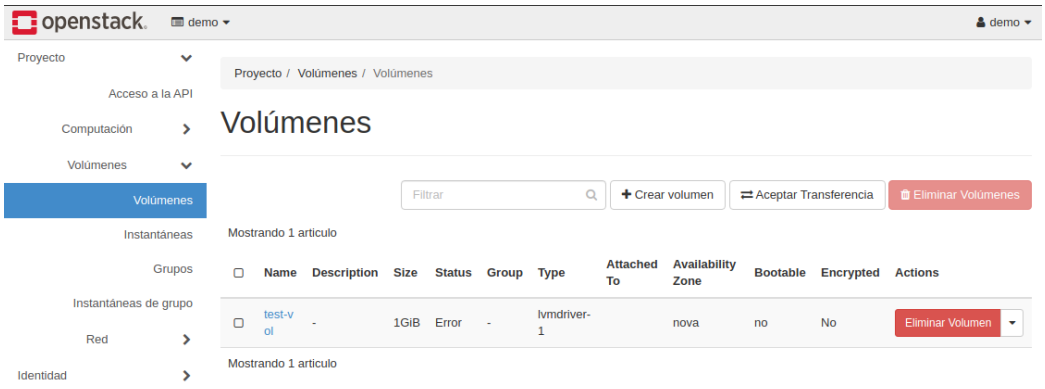

Figura 3.31: Gestionar volúmenes

Los *volúmenes* son discos virtuales que permiten a una instancia trabajar con almacenamiento permanente. Los volúmenes son entidades que se gestionan de manera independiente a las instancias, pero pueden ser asignados a éstas, e incluso pueden ser creados al mismo tiempo que la instancia.

Es posible crear volúmenes vacíos, o a partir de una imagen. En este último caso, el volumen posee todos los datos incluidos en la imagen.

En Dashboard es posible gestionar todos los volúmenes disponibles a través de la ruta `Proyecto > Volúmenes > Volúmenes`. Se muestra esta sección en la figura 3.31.

En la herramienta CLI:

```
> openstack volume create|delete|list|show ...
```

OpenStack permite diseñar complejas arquitecturas de red, que consten de diversas redes públicas y privadas, interconectadas por medio de routers.

Para gestionar adecuadamente las redes, existen diversos elementos que es necesario comprender:

- Grupos de seguridad
- Redes privadas
- Direcciones IP flotantes
- Redes públicas
- Routers

Figura 3.32: Gestionar grupos de seguridad

Respecto a los grupos de seguridad, son colecciones de reglas que determinan qué conexiones de entrada y salida son válidas para acceder a una determinada instancia. En este sentido, representan la configuración del firewall de dicha instancia, aunque en general son restricciones que se aplican a nivel de switch.

Los grupos de seguridad se asignan a una o varias instancias. Una instancia puede acumular más de un grupo de seguridad. Por defecto, OpenStack crea un grupo de seguridad que permite tráfico de entrada y salida.

En Dashboard es posible gestionar los grupos de seguridad a través de la ruta `Proyecto > Red > Grupos de seguridad`. En la figura 3.32 se muestra esta sección.

En CLI:

```
> openstack security group create|delete|list|show|...
```

Las *redes privadas* son redes virtuales que se crean en el contexto de un proyecto (tenant). Pueden ser creadas por cualquier miembro del proyecto. Una red privada admite la conexión de cualquier número de instancias. La red dispone de un servicio DHCP que asigna direcciones IP privadas a todas las instancias conectadas.

Por defecto, una instancia conectada a una red privada tiene acceso al exterior pero no es visible desde fuera de la red, y por lo tanto no es posible conectarse a ella vía SSH. Sin embargo, todas las instancias conectadas a la misma red privada pueden verse entre sí sin problemas.

En Dashboard es posible gestionar las redes a través de la ruta `Proyecto > Red > Redes`. Se muestra esta sección en la figura 3.33.

En CLI se utilizan los comandos:

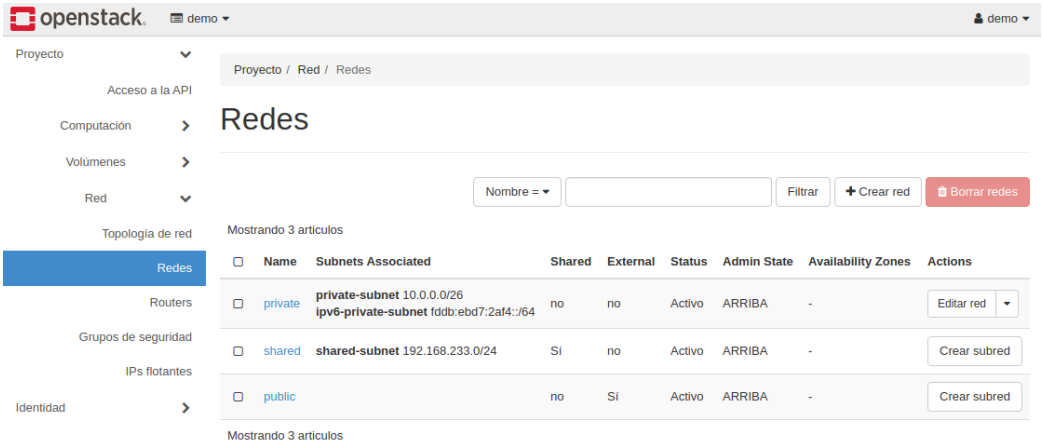

Figura 3.33: Gestionar de redes

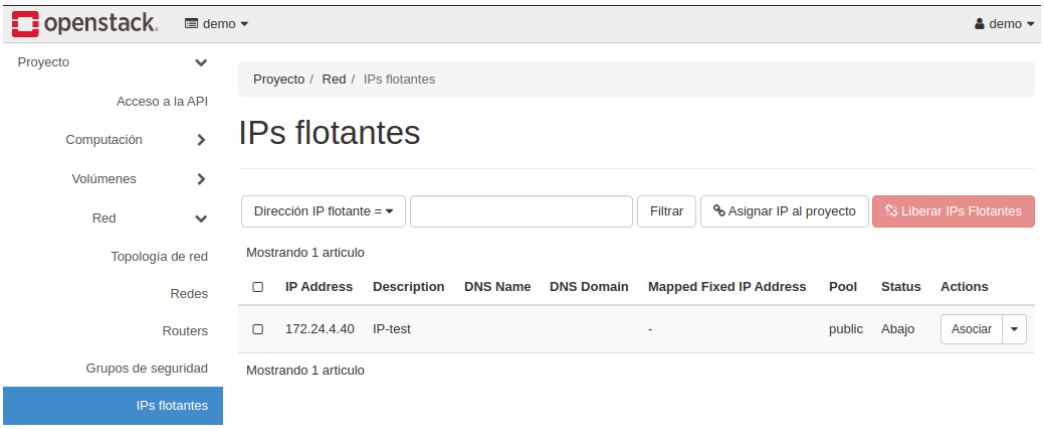

Figura 3.34: Gestionar de IPs flotantes

```
> openstack network create|delete|list|show|...
```

Por defecto, una instancia conectada a una red privada no es visible desde el exterior. Es posible alterar esta condición por medio de *direcciones IP flotantes*. Una dirección IP flotante permite acceder a la instancia desde el exterior de la red privada. Una vez asignada la dirección IP flotante la instancia será visible y será posible conectarse con ésta a través de SSH.

En Dashboard es posible gestionar las redes a través de la ruta `Proyecto > Red > IPs flotantes`. Se muestra esta sección en la figura 3.34.

En CLI se utilizan los comandos:

```
> openstack floating ip create|delete|list|show|...
```

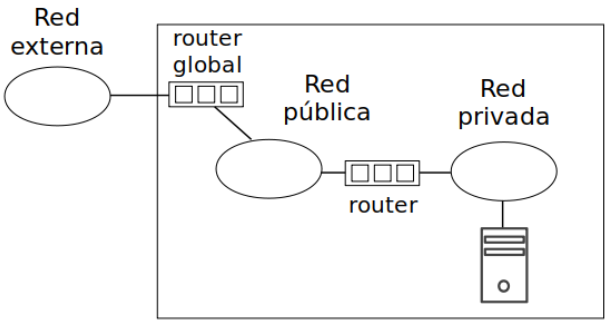

Figura 3.35: Configuración de una red pública

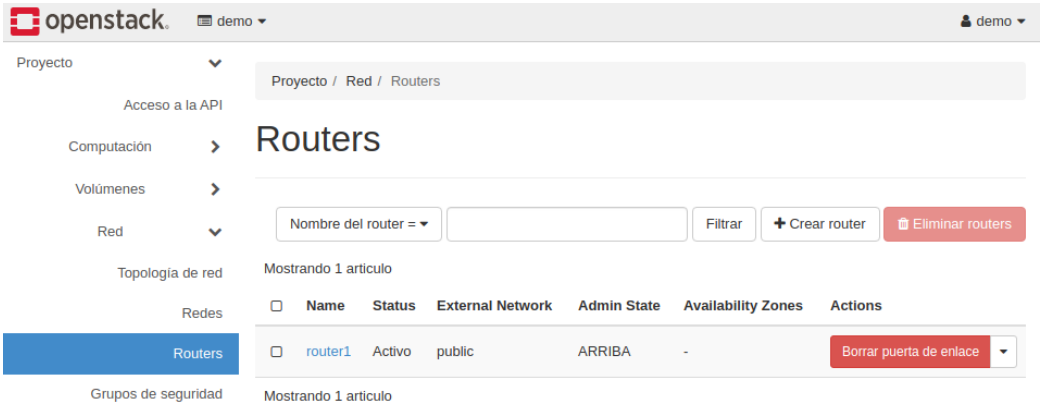

Figura 3.36: Gestionar routers

Las *redes públicas* son redes globales que sólo pueden ser creadas por `admin`. Estas redes representan la interfaz de red externa de OpenStack y pueden ser utilizadas por cualquier número de redes privadas. Las instancias conectadas a este tipo de red reciben una dirección IP pública y son visibles desde fuera. Sin embargo, no es habitual conectar instancias directamente a esta red, sino *routers* que interconectan la red pública con otras redes privadas, dando visibilidad desde el exterior a las instancias conectadas a las redes privadas. La figura 3.35 ilustra esta configuración.

Los *routers* son dispositivos virtuales con múltiples *puertos* que permiten interconectar redes. En OpenStack es muy habitual interconectar una red pública con una red privada por medio de un router.

En Dashboard es posible gestionar las redes a través de la ruta `Proyecto > Red > Routers`. Se muestra esta sección en la figura 3.36.

En CLI se utilizan los comandos:

```
> openstack router create|delete|list|show|...
```

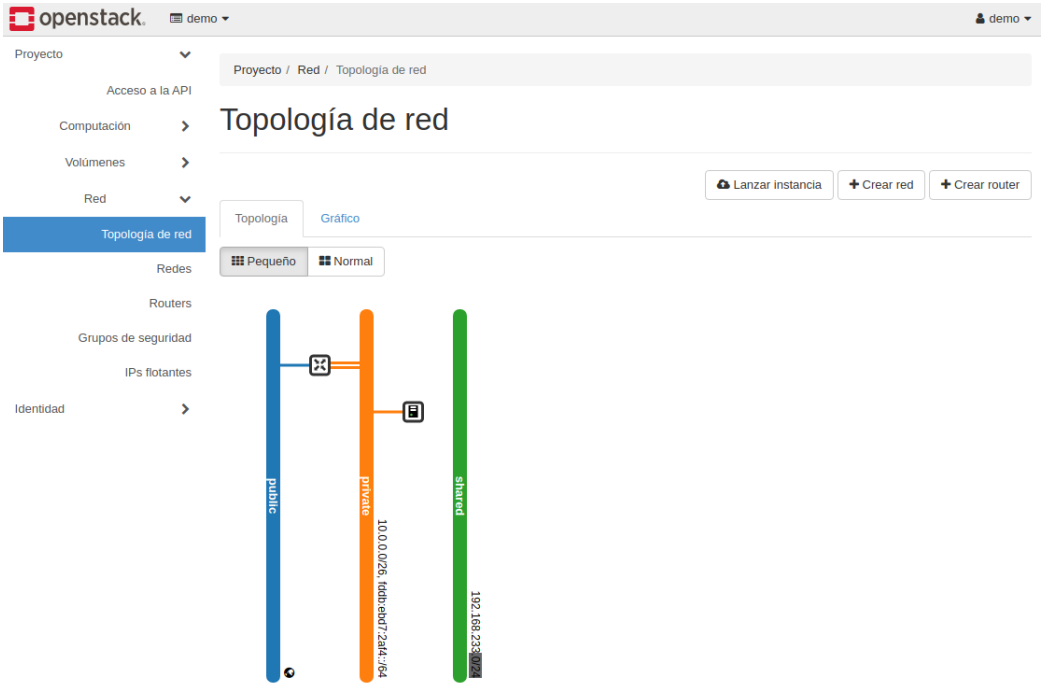

Figura 3.37: Topología de red

```
> openstack router add port|remove port
```

Resulta especialmente interesante visualizar gráficamente la arquitectura de redes creadas. Esta opción está disponible a través de la ruta `Proyecto > Red > Topología de red`. Se muestra esta sección en la figura 3.37.

Referencias

[Kreutz et al., 2014] Kreutz, D., Ramos, F. M. V., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2014). Software-defined networking: A comprehensive survey.

[Liu et al., 2011] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D., et al. (2011). Nist cloud computing reference architecture. *NIST special publication*, 500(2011):1–28.

[Mell et al., 2011] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.