UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Diseño y despliegue de Funciones de Red Virtuales (VNFs) utilizando OpenStack

Autor
Alejandro Toledo Juan

Director
Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 15 de Noviembre de 2018

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Diseño y despliegue de Funciones de Red Virtuales (VNFs) utilizando OpenStack

Autor
Alejandro Toledo Juan

Director
Jorge Navarro Ortiz

Departamento de Teoría de la Señal, Telemática y Comunicaciones

—

Granada, 15 de Noviembre de 2018

Diseño y despliegue de Funciones de Red Virtuales (VNFs) utilizando OpenStack

Alejandro Toledo Juan

Palabras clave: Cloud, Devstack, Heat, IaaS, IT, OpenStack, Tacker, VIM, VM, VNF

Resumen

Hoy día son pocos aquellos que no han escuchado hablar del término *cloud computing* o el concepto de nube. *Cloud computing* es un conjunto de principios y enfoques que permite proporcionar infraestructura informática, servicios, plataformas y aplicaciones que provienen de la nube a los usuarios. Las nubes son grupos de recursos virtuales orquestados por software y automatizados para que los usuarios puedan acceder a estos a pedido a través de los portales de autoservicios a los que dan soporte el escalado automático y la asignación dinámica de recursos.

El uso de los entornos *cloud* ahorra tiempo y esfuerzo a los departamentos de IT, ampliando las implementaciones personalizadas al darle a las unidades empresariales el poder para solicitar e implementar sus propios recursos. Para ello se crean portales de autoservicio con herramientas de escalado automático y asignación de recursos que permiten ser más flexibles y seguros a los entornos de IT actuales, permitiéndosle gestionar y compartir los activos de la empresa u organización, así como datos de forma más escalable e incrementando la movilidad y fiabilidad de los sistemas a la par que la productividad de su actividad.

En este trabajo se pretende realizar la instalación y configuración de una nube basada en el software OpenStack, con el objetivo de ejecutar en el mismo una serie de funciones de red virtualizadas (VNFs). Estas funciones se deberán poder instanciar y gestionar de forma automática.

Este proyecto busca una visión para profesionales de IT que quieren una descripción general de alto nivel de *OpenStack*, y que desean saber si *OpenStack* es la respuesta adecuada para satisfacer las necesidades de IT de su organización. En él también ayudaremos a cualquier persona que quiera establecer un entorno de prueba basado en esta solución a pequeña escala a adquirir experiencia trabajando con OpenStack.

Project Title: Design and Dimensioning Virtual Network Functions (VNFs) using OpenStack

Alejandro Toledo Juan

Keywords: Cloud, Devstack, Heat, IaaS, IT, OpenStack, Tacker, VIM, VM, VNF

Abstract

Nowadays, few are those who have not heard the term cloud computing or the concept of cloud. Cloud computing is a set of principles and approaches that make posible to provide computing infrastructure, services, platforms and applications that come from the cloud to users. Clouds are groups of virtual resources orchestrated and automated by software. Users access them on demand through self-service portals with tools for supporting automatic scaling and dynamic allocation of resources.

The usage of cloud environments saves time and effort for IT departments, expanding custom implementations by giving business units the power to request and implement their own resources. To this end, self-service portals are created with automatic scaling tools and allocation of resources that confess them to be more flexible and secure in current IT environments, allowing them to manage and share the assets of the company or organization, as well as data in a more scalable way, increasing system the mobility and reliability of the systems at the same time as their productivity.

In this work we intend to perform the installation and configuration of a cloud based on the OpenStack platform, in order to execute a series of Virtual Network Functions (VNFs). These functions should be able to be instantiated and managed automatically.

This work seeks a vision for IT professionals who want an OpenStack high-level description to know if OpenStack is the right solution to satisfy their IT requirements in their organizations. At the same time this will help to implement a small scale test environment to gain experience working with OpenStack.

Yo, **Alejandro Toledo Juan**, alumno de la titulación de GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alejandro Toledo Juan

Granada a 15 de Noviembre de 2018.

D. **Jorge Navarro Ortiz**, Profesor del Área de Ingeniería Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Diseño y despliegue de Funciones de Red Virtuales (VNFs) utilizando OpenStack***, ha sido realizado bajo su supervisión por **Alejandro Toledo Juan**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 15 de Noviembre de 2018.

El director:

Jorge Navarro Ortiz

Agradecimientos

Si se siente la gratitud y no se expresa, es como envolver un regalo y no darlo. Ha sido largo el camino realizado hasta llegar a este punto, más de lo que me gusta reconocer.

"Agadezcamos a los que nos befenician y beneficiemos a los agradecidos". Mahoma.

Siento la necesidad de agradecer en primer lugar el tiempo, atención y dedicación que Jorge Navarro ha puesto en este proyecto.

"Imposibles de olvidar jamás. Para obrar, el que da debe olvidar pronto, y el que recibe, nunca". Séneca

Me resulta demasiado insignificante agradecer por todo el esfuerzo, paciencia y sacrificios que mis padres han tenido que realizar para poder estar hoy escribiendo en este punto. Ojalá algún día puedan sentirse tan orgullosos de mí como yo lo estoy de ellos y sienta que nada fue en vano.

"Cuando la gratitud es tan absoluta las palabras no sobran nunca".

Por último y más importante, no puedo acabar este capítulo de agradecimientos sin acordarme de la persona que más a cambiado mi vida. Desde que entro en ella cada vez estoy más cerca de cómo y de lo que quiero ser. Buena parte de estar hoy escribiendo estos renglones es mérito suyo, por lo que espero tenerla a mi lado el tiempo que ella lo desee. Hasta el momento en el que escribo, sé que su pensamiento está en mí, apoyándome como sólo ella sabe hacerlo, a pesar de a veces pueda no transmitirle esa sensación.

Índice general

Índice de figuras

Índice de cuadros

Lista de acrónimos

API Application Programming Interface

CLI Command Line Interface

ETSI European Telecommunication Standars Institute

HOT Heat Orchestration Template

IaaS Infraestructure as a Service

IT Information Technology

LVM Linux Volume Manager

NAT Network Address Translation

NFV Network Function Virtualization

NFVI NFV Infraestructure

NFVO NFV Orchestrator

NSD Network Services Descriptors

PaaS Platform as a Service

SaaS Software as a Service

SDN Software Defined Network

SDS Software Defined Storage

TIC Tecnologías de Información y comunicación

TOSCA Topology and Orchestration Specification for Cloud Applications

VIM Virtual Infrastructure Manager

VM Virtual Machine

VNF Virtual Network Function

VNFD VNF Descriptors

VNFFGD VNF Forwarding Graph Descriptors

VNFM VNF Manager

Capítulo 1

Introducción

En la actualidad los servicios IT han alcanzado un papel crítico en el desarrollo de nuestra sociedad, haciendo que surjan tecnologías como la computación en la nube, la virtualización, o la orquestación de estos elementos virtuales, que ahora marcan el camino a seguir en los despliegues de cualquier infraestructura de red o centro de datos. Dentro de este ámbito, es imprescindible dotar a todos los servicios que existen de mecanismos de gestión y aprovechamiento de los recursos para complacer las demandas de los usuarios asegurando alta disponibilidad.

Este primer capítulo tiene como fin hacer una primera presentación de los aspectos relacionados con el trabajo fin de grado. En primer lugar vamos a describir algunos detalles que nos ayudarán a contextualizar las tecnologías implicadas para, a continuación, exponer las metas que nos proponemos alcanzar a lo largo del desarrollo del mismo.

Hablaremos también de la situación actual de los desarrollos *cloud* basados en OpenStack y la relevancia que tiene en los despliegues en entornos de producción hoy día.

Finalmente, subrayaremos la estructura del resto de la memoria incluyendo la descripción de cada uno de los capítulos y apéndices incluidos en esta memoria.

1.1. Contexto y motivación

Cloud computing es un paradigma de computación que permite ofrecer recursos informáticos a través de la red. Estos recursos (redes, almacenamiento, cómputo, servidores, aplicaciones, servicios) se ofrecen bajo demanda y pueden ser suministrados con una mínima interacción y gestión por parte del proveedor del servicio.

Este es el punto en el que OpenStack entra en juego. El proyecto OpenStack se define así mismo como una plataforma de *cloud computing* hecha con software libre para desplegar nubes públicas y privadas orientadas a ofrecer infraestructuras como servicio (*Infrastructure as a Service, IaaS*) a los usuarios, desarrollada con la idea de ser sencilla de implementar, masivamente escalable y con muchas prestaciones.

OpenStack está creciendo a un ritmo sin precedentes. Prueba de ello es el hecho de que el 65 % de las implementaciones de *cloud* en producción y el 96 % de quienes están llevando acabo despliegues de VNFs (*Virtual Network Functions*) lo usan, convirtiéndose así en una parte esencial de las nuevas infraestructuras de red [1]. Otra muestra de la relevancia de este proyecto es el hecho de que, según el informe de trabajos de código abierto de *The Linux Foundation and Dice* [2], el 64 % de los gerentes de contratación dice que la experiencia con OpenStack y otras tecnologías en la nube están impulsando las decisiones de contratación en trabajos relacionados con el despliegue de infraestructuras de *data centers*, *cloud* y virtualización.

Por tanto, un motivo fundamental a la hora de considerar esta herramienta para el presente trabajo es el creciente uso en despliegues profesionales de tecnologías IT y el papel fundamental que OpenStack tiene hoy día en el desarrollo de despliegues de redes virtuales, gestión de VNFs, centros de datos, *edge computing*, contenedores y entornos *cloud*.

Otro aspecto importante a la hora de desarrollar el proyecto es la cantidad de nuevas tecnologías que se pueden usar y con las que se trabaja en este tipo de entornos, conociendo así el alcance que puede llegar a tener OpenStack.

También resulta relevante el cambio de enfoque respecto a los centros de datos convencionales. En el enfoque tradicional de implementación de centros de datos con máquinas físicas existen muchos recursos desaprovechados, en contraposición con el enfoque de computación en la nube, donde los recursos se comparten globalmente permitiendo una mayor elasticidad y reduciendo así costes en espacio y consumo entre otros.

Otro buen motivo para trabajar con esta herramienta es el hecho de que es *Open Source*. OpenStack es una plataforma de *cloud computing* cuyo desarrollo se basa en Licencia Apache 2.0 [3], una licencia libre que nos permite acceder al código fuente y modificarlo o realizar aportaciones y que además no hace uso de funcionalidades de pago.

1.2. Objetivos y alcance del proyecto

El proyecto actual se lleva acabo como reto personal en el que se pretende conseguir crear un entorno de *cloud* privado preparado para dotarla en un

futuro cualquier servicio que se quisiera dar, surtiendo a la infraestructura de mecanismos de orquestación y escalabilidad de funciones de red y recursos, de modo que se facilite la gestión de la infraestructura, pudiendo abordar la provisión de recursos en esta de una forma automatizada y dinámica.

Para el abordaje del mismo son de gran valor los conocimientos adquiridos durante la formación del grado en el despliegue y diseño de redes para conocer las necesidades que tendremos a la hora de realizar el planteamiento y jerarquización de los escenarios y recursos implicados en el mismo así como la gestión de las máquinas tanto físicas como virtuales.

En primer lugar, comenzaremos haciendo un repaso de las tecnologías y conceptos implicados en la realización del presente trabajo y también de nociones varias que nos ayudarán a comprender el alcance que puede llegar a tener el uso de OpenStack.

Pasaremos después a ver las distintas opciones o herramientas similares que tenemos a la hora de desplegar nuestro entorno de trabajo. Una vez seleccionadas algunas de las principales opciones que existen, haremos las comparaciones oportunas con OpenStack justificando así la elección de esta herramienta para el despliegue de nuestra infraestructura.

Antes de realizar dicho despliegue, será importante hacer una planificación exhaustiva tanto de los recursos hardware, software y otros posibles que puedan estar implicados así como de los costes que conllevan.

Con todo ello, plantearemos el diseño de la infraestructura razonando los motivos que nos han llevado a optar por esa vía para a continuación realizar la instalación de la misma.

Una vez este a punto nuestra infraestructura, veremos como podemos realizar la gestión de la plataforma para crear nuestra *cloud* privada y gestionar distintas VNFs tanto vía web con la ayuda de uno de los proyectos de OpenStack, como desde línea de comandos y de manera automatizada, creando redes, cortafuegos, máquinas virtuales y asignando recursos a estas máquinas entre otras posibilidades que nos permitirán sacarle el máximo partido a los escenarios creados.

De este modo, trataremos distintos métodos para realizar la gestión de algunas funciones de red, de forma que a medida que avancemos la forma de gestionarlas sea cada vez más automatizada, haciendo el desarrollo escalable.

Queremos por tanto crear una nube que pueda orquestar funciones de red virtuales de manera automatizada. Para abordar la puesta a punto de las partes implicadas en el proyecto que irán aportando valor a la infraestructura a medida que se avance se pretende:

- Crear un escenario en el que aparezcan todos los elementos necesarios para la puesta a punto de cualquier entorno. A saber: creación de

proyectos, usuarios, IPs flotantes, creación de redes, subredes, gestión de imágenes, creación de grupos de seguridad, pares de claves público-privadas y volúmenes de almacenamiento. Todos ellos se crearán con el fin de ser usados por una o varias instancias o máquinas virtuales.

- El punto anterior será la base de partida. Teniendo en mente las partes implicadas en la creación de máquinas virtuales o instancias, iremos replicando este escenarios con distintos métodos para ilustrar en orden creciente el valor que aporta dar el salto a cada realización del escenario. En primer lugar crearemos un escenario desde la interfaz web, pasaremos después a hacerlo desde línea de comandos para después crear todos estos recursos o VNFs mediante *scripting*.

- Una vez realizadas las tareas anteriores pasaremos a usar herramientas de automatización como es el caso de Heat, para escalar estas funciones de red de forma automatizada, viendo además como combinada con *scripting* podremos monitorizar instancias para ser capaces de recuperarlas frente a fallos.

- Por último, con nuestra infraestructura a punto y el uso de Tacker, seremos capaz de registrarla creando así un *site* que para nosotros será una infraestructura virtual a gestionar, y una vez realizado este paso, veremos como crear VNFs más avanzadas con mecanismos de auto-recuperación ante fallos entre otros.

Todos los conceptos introducidos en esta parte se irán tratando a lo largo de la memoria en detalle, obteniendo así un conocimiento amplio a alto nivel de OpenStack y sus posibilidades.

1.3. Estructura de la memoria

Este documento consta de nueve capítulos que describimos a continuación:

- **Capítulo 1: Introducción**. Breve explicación del contexto y las razones de llevar a cabo este proyecto así como los objetivos a alcanzar.

- **Capítulo 2: Conceptos previos**. En este capítulo se abordarán conceptos y tecnologías relacionadas con las posibilidades que OpenStack ofrece haciendo énfasis en aquellas que atañen a este proyecto.

- **Capítulo 3: Estado del arte**. Exploración de las principales soluciones del mercado que existen como alternativa y justificación de la elección.

- **Capítulo 4: Planificación y costes**. Estimación del tiempo, recursos empleados y costes asociados para llevar a cabo el proyecto.

- **Capítulo 5: Herramientas utilizadas**. Análisis en profundidad de OpenStack y los principales proyectos que lo componen haciendo énfasis en aquellos que usaremos.

- **Capítulo 6: Diseño**. Explicación y justificación de los distintos escenarios planteados para la orquestación de las funciones de red.

- **Capítulo 7: Implementación**. Documentación del proceso seguido para la instalación de la infraestructura y la puesta a punto de los distintos escenarios y servicios planteados.

- **Capítulo 8: Pruebas y resultados**. Realización de diferentes test para corroborar el correcto funcionamiento del despliegue realizado.

- **Capítulo 9: Conclusiones y líneas futuras**. Culminación de la memoria haciendo y repaso de los objetivos alcanzados y prospección de futuras vías para la ampliación del trabajo.

Al final de la memoria, podremos encontrar además distintos apéndices:

- **Apéndice A: Arquitectura lógica de OpenStack**. Arquitectura lógica en la que se muestran los componentes de los principales proyectos de OpenStack y la relación que hay entre ellos.

- **Apéndice B: Archivo de configuración para la instalación**. El proceso de instalación de OpenStack con DevStack tiene como elemento principal un archivo de configuración donde se describen los recursos que queremos que tenga nuestra nube. Este tema merece una mención especial y se propone como anexo al proceso de instalación que se verá en la sección 7.2.2.

- **Apéndice C: Creación de un escenario mediante *scripting* en bash**. Generación de un script para automatizar la tarea de configurar un escenario de red desde la línea de comandos.

Capítulo 2

Conceptos previos

Este capítulo trata de recoger y exponer los principales conceptos y tecnologías implicadas en los despliegues en los que se usa OpenStack.

Para ello iremos desde el concepto de virtualización hasta llegar al cloud computing, pasando por la tecnología de contenedores a la par que analizamos los beneficios de estos y algunos inconvenientes para acabar hablando de las funciones de red virtualizadas, VNFs, y el rol que OpenStack tiene en ella.

2.1. De la virtualización al Cloud Computing

A lo largo de los años, la informática corporativa ha visto muchos desarrollos que han derivado en el aumento de la computación en la nube tal como la conocemos:

- En la década de 1990, la informática corporativa se centraba en servidores en un centro de datos.
- En la década de 2000, la informática corporativa se basaba principalmente en la virtualización.
- En la década de 2010, estamos viendo el aumento de la computación en la nube para aprovechar la informática corporativa.

2.1.1. Virtualización

Para introducir los contenidos que trataremos, vamos a hablar en primer lugar de qué es la virtualización. Cuando se habla de virtualización, la mayoría de las personas piensa en máquinas virtuales. Una máquina virtual

(Virtual Machine, VM) es una computadora en la que el software, así como el hardware, se crean como una solución de software que simula un sistema de computación y puede ejecutar programas como si fuese una computadora real. El objetivo de la virtualización es proporcionar una versión virtual, en lugar de física, de los activos de IT esenciales:

- **Hardware**: Este es el uso "tradicional", donde el sistema operativo se instala en una representación de software de los recursos de hardware.
- **Almacenamiento**: También conocido como almacenamiento definido por software (*Software Defined Storage, SDS*). En SDS, se crea una capa entre los discos duros reales y las computadoras que acceden a estos discos duros, con el objetivo de hacerlos más accesibles.
- **Redes**: Las actuales redes definidas por software (*Software Defined Networks, SDN*). En SDN, los usuarios de la nube pueden crear una infraestructura de red lógica en la parte superior de la red física, lo que hace que sea más fácil satisfacer sus necesidades.

2.1.2. Beneficios de la virtualización

En OpenStack, se utilizan todos los tipos de virtualización. La virtualización tiene muchos beneficios para las IT modernas:

- Permite un uso más eficiente de la capacidad del servidor físico. Si se observa la utilización típica de la CPU de los servidores de hardware, lo más probable es que veamos una utilización de CPU inferior al 10 o incluso al 5 %. La virtualización permite ejecutar diferentes máquinas virtuales sobre CPUs físicas, lo que significa que el uso de la CPU se puede optimizar para colocarnos en torno al 80 %.
- Permite un uso más eficiente del espacio físico destinado al centro de datos. Este es un gran problema porque el espacio del centro de datos es caro. Imaginemos que deseamos construir un centro de datos en el medio de una gran ciudad. Los metros cuadrados en las grandes ciudades en general son muy caros. Esta es la razón por la cual la virtualización ayuda en este sentido, porque el uso de máquinas virtuales significa que necesitaremos menos hardware físico, y usar menos hardware físico significa usar menos metros cuadrados con el consecuente ahorro económico.
- Permite un consumo de energía más eficiente. Como resultado del punto anterior, también podremos usar la energía de una manera más eficiente, si pensamos en las facturas de electricidad de los centros de

datos corporativos grandes, que pueden llegar a millones. Por tanto, si optimizamos los servidores reduciendo por ejemplo a la mitad la cantidad de servidores físicos que se necesitan, mejoraremos en esto aspecto, utilizando así la mitad de la energía necesaria para ejecutar estos servidores, lo que permite a las compañías ahorrar una gran suma.

2.1.3. Implicaciones de la Virtualización

Cuando se trata de virtualización, los activos físicos están representados por software:

- Los servidores y las estaciones de trabajo se están convirtiendo en máquinas virtuales.
- Las redes y el almacenamiento también se pueden virtualizar, convirtiéndose así en SDN y SDS.
- Con todos estos tipos de virtualización, un administrador puede construir un *Data Center* definido por software.

2.1.4. Limitaciones de la virtualización

Aún así, la virtualización también tiene algunas limitaciones:

- La ausencia de soluciones de autoservicio. En virtualización, es el administrador quien proporciona la infraestructura que está creando el almacenamiento, la red y las máquinas virtuales. Como usuario final, solo se podrá solicitar la virtualización de los recursos para a continuación esperar a que el administrador tenga tiempo para cumplir con esta solicitud.
- Escalabilidad limitada. Por supuesto, hay escalabilidad, porque se pueden agregar servidores a la virtualización para alojar más máquinas virtuales, y también se pueden agregar redes, así como almacenamiento. Pero el hecho de que sea el administrador el encargado de estas tareas, hace que la escalabilidad se encuentre restringida.
- La virtualización es relativamente pesada, ya que cada VM tiene su propio *kernel*. Si estamos ejecutando veinte máquinas virtuales sobre un hipervisor, estamos ejecutando veinte *kernels* diferentes. Si se necesitan sin embargo, máquinas virtuales con diferentes sistemas operativos, no hay realmente otra solución para hacerlo de manera más eficiente. Por el contrario, si estamos ejecutando sistemas operativos

similares, podemos hacerlo de manera más eficiente, y ese es exactamente el objeto de la tecnología de contenedores.

2.1.5. Virtualización basada en host vs basada en hipervisor

La virtualización basada en hipervisor, también denominada *Bare-Metal* debido a que se instala directamente en un servidor físico sin necesidad de un sistema operativo, es la solución más óptima pare entornos de producción. La solución más común utilizada para ejecutar máquinas virtuales sobre Linux en un centro de datos es KVM *(Kernel-based Virtual Machine)*. KVM es un tipo de virtualización basada en hipervisor (como es el caso de otras soluciones como *VMware ESXi*). Cuando se trata de virtualización basada en hipervisor, la máquina virtual se ejecuta sobre un kernel de virtualización pequeño y altamente optimizado, que permite a la máquina virtual abordar el hardware de manera más eficiente tal como se esquematiza en la parte izquierda de la Fig.2.1.

Otra solución de virtualización común es la basada en *host*, como por ejemplo *VirtualBox*. La principal diferencia con la anterior radica en que esta solución necesita en primer lugar un sistema operativo sobre el que los desarrolladores podrán ejecutar una aplicación en la parte superior de su escritorio con la que poder crear máquinas virtuales.

La solución de virtualización no es tan eficiente como la virtualización basada en hipervisor, la cuál ofrece mayor rendimiento, fiabilidad, estabilidad, escalabilidad y más funcionalidades, motivos por los cuales es utilizada como método de virtualización en las infraestructuras como servicio.

2.1.6. Tecnología de contenedores

Como solución al problema del peso de la virtualización, surgió la tecnología de contenedores. En IT, el objetivo fundamental es ejecutar una aplicación y no una máquina virtual. Al fin y al cabo, es el usuario de la aplicación quien tiene una necesidad. Esta necesidad es ejecutar una aplicación y ningún usuario pedirá por lo general una máquina virtual (aunque actualmente existen empresas que están desplegando precisamente este tipo de servicios). Ese es, pues, uno de los principales objetivos de diseño de la tecnología de contenedores.

Un contenedor es un método de virtualización en el nivel del sistema operativo. Esto permite que varias instancias de un sistema operativo se ejecuten en el mismo kernel, lo que permite un uso más eficiente de los recursos disponibles.

Usar contenedores facilita a los usuarios finales ejecutar aplicaciones en

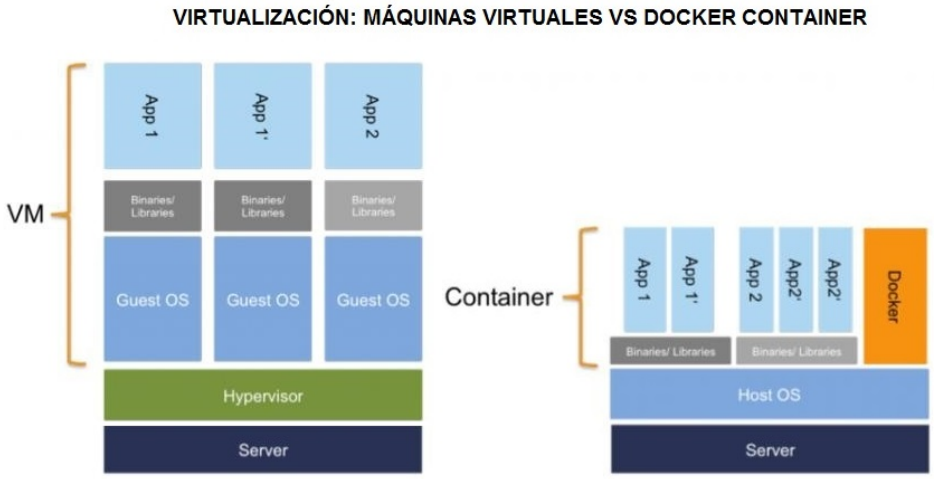


Figura 2.1: Containers vs VMs.

Fuente: Crisp Research, "Container-Technologien auf dem Vormarsch: Docker in a Nutshell", 2014

diferentes plataformas. Las tecnologías de contenedores, como *Docker*, eliminan la necesidad de considerar las diferentes características de la plataforma. El contenedor simplemente se ejecuta como un entorno aislado en la parte superior de la plataforma. Todo lo específico de la aplicación se incluye en un paquete que se ejecuta en la parte superior de cualquier plataforma de hardware. Esto permite al desarrollador centrarse en la aplicación, no en la plataforma subyacente.

Por tanto, el desafío dentro de un entorno de IT moderno típico es el hecho de la existencia de una gran cantidad de entornos de hardware y multitud de *stacks* o pilas que necesitan soporte. Cuando hablamos de una pila o *stack*, estamos hablando de un entorno completo que se usa para interactuar con los contenedores, como se puede ver a la derecha de la Fig. 2.1 en comparación de la pila usada para las máquinas virtuales que vemos en la izquierda.

Algunas pilas existentes se ilustran en la Fig.2.2, como un sitio web estático, una base de datos de usuario (*user DB*), una interfaz web, una cola o una base de datos de análisis *(analytics BD*). Estas pilas son más que sólo la aplicación, hay un entorno completo que se necesita para ejecutar la pila en la parte superior de un sistema operativo. El propósito de la contenerización es ponerlo todo en un contenedor, de forma que creamos un paquete autónomo donde todo está incluido con el fin de poder ser manipulado usando operaciones estándar y ejecutarse consistentemente en prácticamente cualquier plataforma de hardware como puede ser una VM

Figura 2.2: Docker stacks.

Fuente: Docker

(*Virtual Machine*), un servidor de QA (*Quality Assurance*), un centro de datos de clientes, una nube pública, un clúster de producción, una PC portátil, etc.

Por lo tanto, al trabajar con contenedores, los desarrolladores de aplicaciones pueden eliminar los problemas relacionados con la ejecución de aplicaciones sobre diferentes sistemas operativos.

De esta forma, en general, la tecnología de contenedores está haciendo que las IT sean más eficientes debido a las ventajas que ofrece:

- Se pueden implementar fácilmente muchas instancias de una aplicación para que se ejecute sobre un kernel único.
- Proporciona entornos aislados y, por lo tanto, seguros.
- A su vez, esto hace que los contenedores en ejecución sean un proceso seguro: un usuario de un contenedor no puede acceder a los recursos que están asignados a otro contenedor.
- Debido al hecho de que muchos contenedores se pueden usar sobre el mismo kernel, los recursos se pueden usar de una manera más eficiente.
- La implementación es mucho más llevadera debido al tamaño relativamente pequeño de los contenedores.

¿Cuáles son por tanto las diferencias entre los contenedores y la virtualización? Básicamente que los contenedores y la virtualización se utilizan en diferentes contextos:

- Usaremos contenedores si queremos ejecutar múltiples copias de una sola aplicación, con lo que podremos balancear la carga.

- Por otro lado, usaremos máquinas virtuales si necesitamos la flexibilidad de ejecutar múltiples aplicaciones o si queremos poder ejecutarlas en cualquier sistema operativo.

Tanto los contenedores como las máquinas virtuales se pueden ejecutar juntos en una nube IaaS.

2.2. Cloud Computing

La computación en la nube o *Cloud Computing*, es un tipo de computación que se apoya en Internet para proporcionar recursos de procesamiento compartido y datos a computadoras y otros dispositivos bajo demanda. Permite el acceso bajo demanda a un grupo compartido de recursos informáticos, como redes, servidores, almacenamiento, aplicaciones y servicios, que normalmente están alojados en centros de datos de terceros. La computación en la nube es una metáfora para hacer referencia a un concepto. Para un usuario, los elementos de red que representan los servicios prestados por el proveedor son invisibles, como oscurecidos por una nube. La conclusión es que, desde la perspectiva del usuario, realmente no importa qué recursos se ejecutan o dónde, todo lo que importa son los recursos mismos.

OpenStack pertenece a la categoría de *computación en la nube IaaS*. Sin embargo, OpenStack evoluciona continuamente, ampliando su alcance. En ocasiones, el enfoque de OpenStack va más allá de IaaS. Vamos a profundizar más sobre IaaS en cloud y los diferentes tipos de cloud.

2.2.1. Tipos de cloud computing

El concepto de *computación en la nube* es muy amplio, hasta el punto de que podemos afirmar que no existe tal cosa como la computación en la nube. Si le pide a un usuario final que explique qué es la computación en la nube y luego le hace la misma pregunta al administrador del sistema, obtendremos dos descripciones diferentes. En general, hay tres enfoques importantes cuando se trata de computación en la nube dependiendo de los tipos de recursos que se gestionen. En la Fig.2.3 se puede ver esta clasificación tomando como referencia un despliegue tradicional donde se gestionan todos los recursos. Otro punto de vista que lleva a la misma división es, de acuerdo al usuario final de la nube (Fig.2.4, cuya pirámide inversa sería la cantidad de proveedores de servicio que existen):

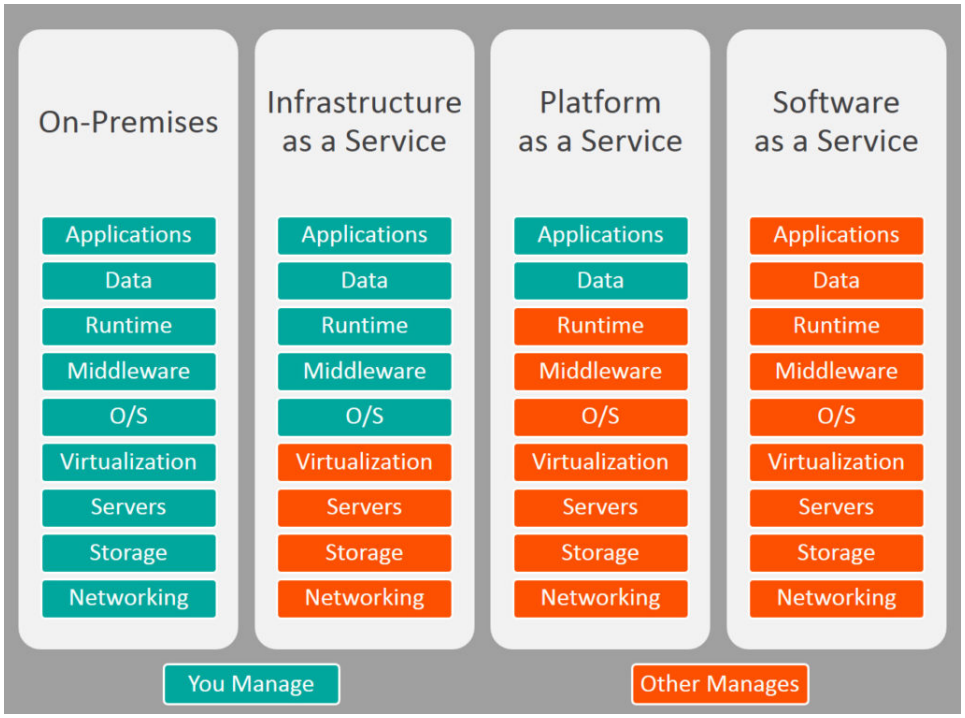


Figura 2.3: Clasificación de cloud computing en base a los recursos que se gestionan.

Fuente: Stephen Watts, "SaaS vs PaaS vs IaaS: What's The Difference and How To Choose", bmc blogs, 2017

- **IaaS (*Infrastructure as a Service*)**: Es una infraestructura que se utiliza para proporcionar máquinas virtuales. Va más allá de la virtualización porque la nube agrega escalabilidad y aspectos bajo demanda a la virtualización ofreciendo un control total sobre la infraestructura disponiendo de mecanismos de procesamiento, almacenamiento, red y otros recursos computacionales. Ejemplos de este tipo de arquitectura son: Amazon Web Services, Rackspace Cloud Servers o Mirantis Cloud Platform.

- **PaaS (*Platform as a Service*)**: El proveedor suministra la red, los servidores, el almacenamiento, el sistema operativo y el middleware para alojar una aplicación, como casos de uso tenemos: Google App Engine, Windows Azure, OpenShift o Heroku.

- **SaaS (*Software as a Service*)**: El proveedor da acceso a una aplicación alojada en la nube como ocurre por ejemplo en Google Apps o Dropbox.

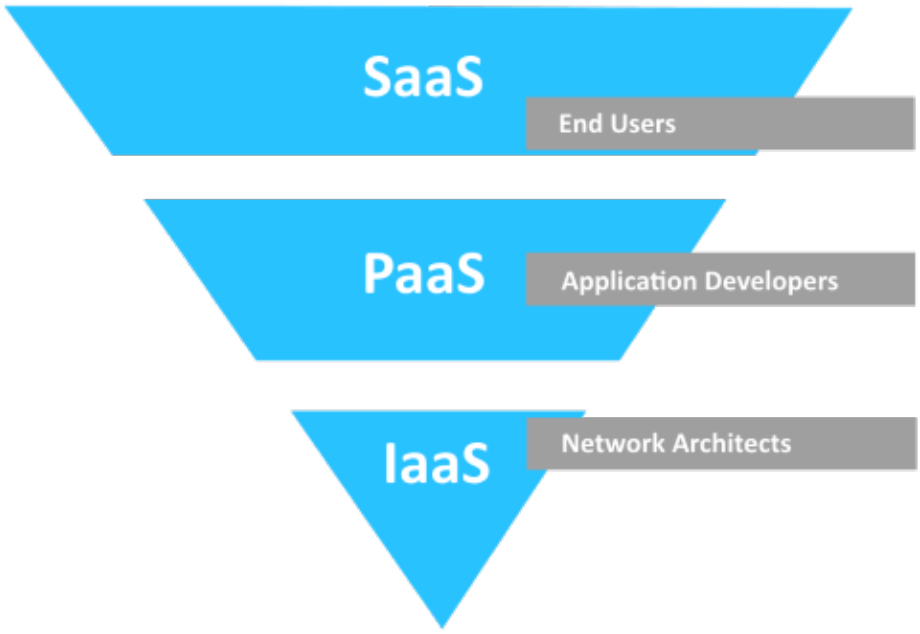


Figura 2.4: Clasificación de cloud computing en función del usuario.

Fuente: GBS Soluciones de Negocio

En la Fig.2.5 podemos ver representado todo lo que realmente podemos hacer en Cloud Computing. Vemos una nube a la que se puede acceder desde teléfonos, portátiles, servidores, escritorios y tabletas, realmente es accesible desde cualquier dispositivo. Dentro de la nube tenemos aplicaciones, plataformas e infraestructura. La infraestructura es la nube IaaS. En la nube IaaS, se están proporcionando recursos de cómputo, almacenamiento en bloque y redes. Luego está la plataforma, que es la plataforma como servicio en la nube (PaaS), en la que se proporcionan el almacenamiento de objetos, servicios de identificación, el tiempo de ejecución de los procesos, la cola y la base de datos. Y por último, en la capa más alta tenemos la nube de aplicaciones que conocemos como nube de software como servicio (SaaS). En ella se están ofreciendo aplicaciones que pueden ser muy diversas: monitorización, contenidos específicos, colaboración, comunicación o finanzas entre otros.

2.2.2. Beneficios del cloud computing

La computación en la nube presenta diversos beneficios. Hace que las IT sean flexibles para los usuarios así como para los administradores:

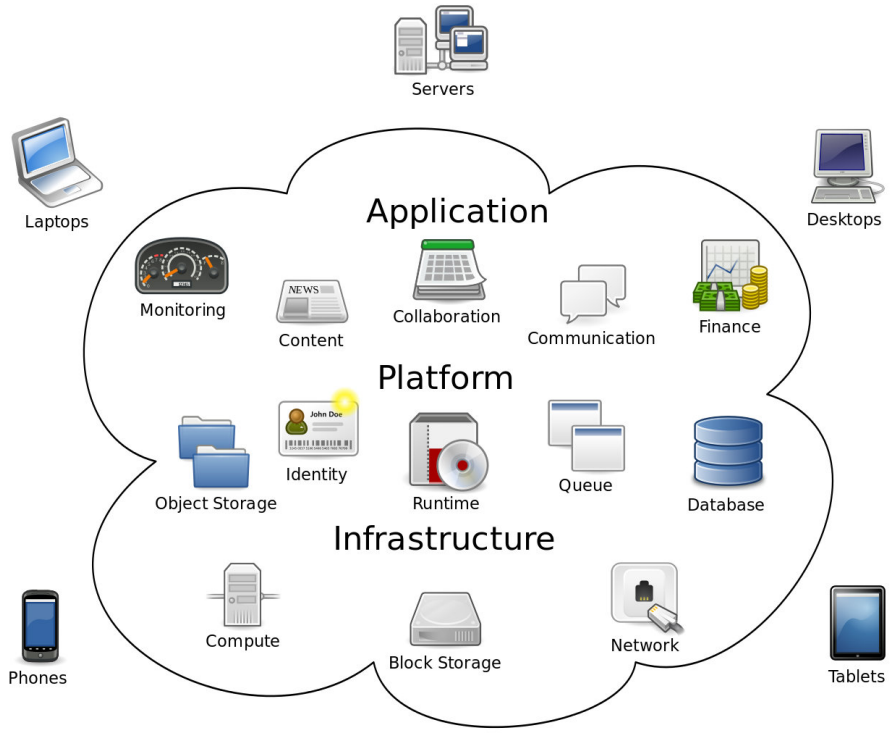


Figura 2.5: Entorno Cloud Computing.

Fuente:Sam Johnston, “Cloud Computing”, licensed underCC BY-SA 3.0 Unported, retrieved from Wikipedia)

- Proporciona fácil acceso a los activos de IT esenciales. Esto hace que sea lo más fácil posible para los usuarios emplear estos activos.

- Permite el auto-despliegue o auto-escalado. Los usuarios ya no necesitan un administrador de IT para implementar un nuevo servidor.

- Proporciona escalabilidad. Cuando se quede sin recursos físicos, es fácil agregar nuevos recursos.

2.2.3. Modelo de Servicio IaaS

Como hemos comentado ya, OpenStack se basa inicialmente en un modelo IaaS. Las nubes IaaS se pueden ofrecer de diferentes maneras, utilizando diferentes modelos de servicio:

- **Nube privada (*Private Cloud*)**. Una empresa crea una infraestructura IaaS que es solo para uso privado.
- **Nube pública (*Public Cloud*)**. La capacidad de la nube se ofrece como un servicio básico, como es el caso de la telefonía que ofrece un proveedor de telecomunicaciones. Los clientes contratan por tanto una parte de la infraestructura de la nube pública.
- **Nube híbrida (*Hybrid Cloud*)**. Este tipo de servicio IaaS es un modelo en el que una sesión en la nube consta de componentes de IaaS privados y públicos.

2.2.4. Funcionamiento del modelo IaaS

El funcionamiento de la computación IaaS es el siguiente: un usuario final accede al portal de la nube para derivar una máquina virtual. Mientras se prepara la instancia, el usuario final toma decisiones sobre redes, almacenamiento y seguridad. El almacenamiento persistente es opcional. Al final de la vida útil de la VM, esta desaparecerá.

La nube IaaS proporciona una plataforma flexible para ejecutar contenedores, así como máquinas virtuales, de una manera flexible:

- La nube IaaS ofrece un portal de autoservicio.
- También se proporciona almacenamiento escalable y conexión en red.

La computación en la nube IaaS funciona mejor para entornos que necesitan soluciones de IT flexibles. Si no necesitamos escalabilidad y autoservicio, quizás sea mejor optar por otro tipo de virtualización.

2.3. VNF

Para hablar del concepto de VNF nos apoyaremos en el de NFV. *OpenStack Foundation* describe NFV (*Network Function Virtualization*) simplemente como una nueva forma de definir, crear y administrar funciones y servicios de red al reemplazar dispositivos físicos dedicados, con software que puede ser automatizado y administrado por OpenStack.

Al igual que las máquinas virtuales tomaron el lugar de muchos servidores dedicados en los centros de datos, NFV simplemente continúa con la mentalidad de reemplazar hardware físico inflexible y patentado por software. Una muestra de ello podemos verla en la Fig.2.6.

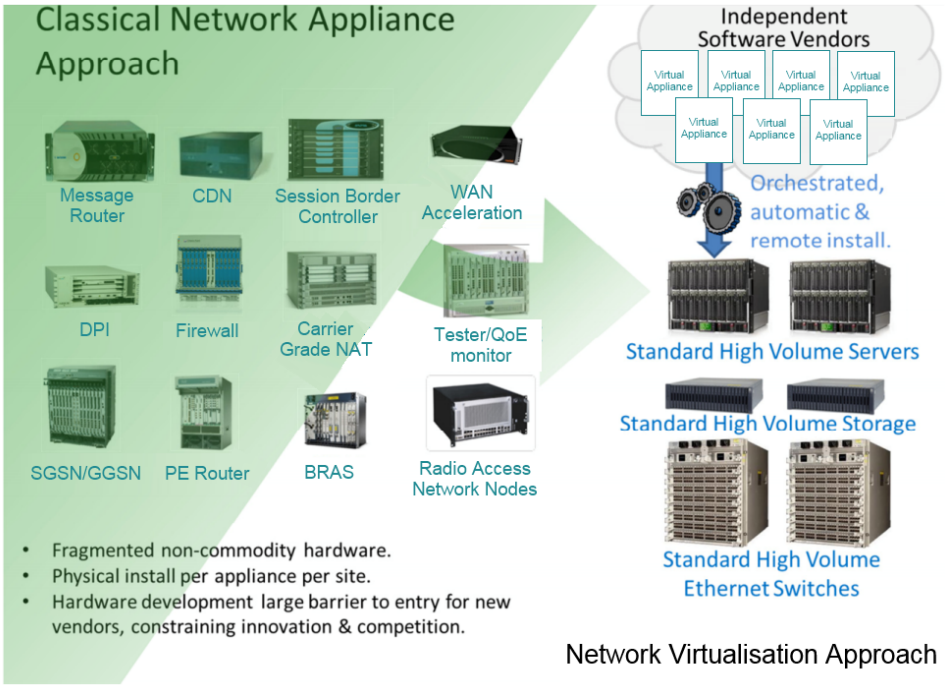


Figura 2.6: Infraestructura física propietaria vs NFV.

Fuente: ETSI

Es aquí donde entran en juego las funciones de red virtuales (*Virtual Network Function, VNF*). Las VNF son responsables de completar tareas de red específicas, que pueden ser máquinas virtuales, un contenedor, abarcar múltiples contenedores, múltiples máquinas virtuales o servidores *bare metal*, *routers*, *firewalls* u otras funciones descritas todas en la parte superior de la infraestructura informática y de la red subyacente.

Más ejemplos de VNF son puertas de enlace o *gateways* móviles, *routers*,

servicios de red de entrega de contenido, cortafuegos, aceleradores WAN, DNS e incluso funciones básicas de paquete, todas ellas implementadas de manera virtual.

2.3.1. Beneficios de VNF

En general, la mayoría de los beneficios del uso de VNFs y NFV se centran en agilidad, flexibilidad y simplicidad. Hoy en día, la industria de las telecomunicaciones es más competitiva que nunca debido a los márgenes decrecientes, los costos y la demanda de nuevos servicios en aumento. Estas tecnologías prometen ofrecer algunas soluciones viables nuevas. Algunos de los beneficios principales de ellas se detallan a continuación:

- Flexibilidad en la red a través del aprovisionamiento automatizado.
- Aprovechamiento de la tecnología de código abierto.
- La flexibilidad de los controladores y complementos.
- Completo acceso y uso de APIs que permiten un desarrollo más rápido.
- El uso de hardware COTS (*Commercial Off-The-Shelf*) frente a los dispositivos patentados.
- Eficiencia operativa en la orquestación en centros de datos, regiones y empresas.
- Amplia gama disponible de documentación.

Los beneficios de VNF en OpenStack (frente a otras plataformas) aumentan la eficiencia y reducen los requisitos de Capex y Opex (inversión y gasto), potencia y espacio [4].

2.3.2. Diferencias entre NFV y SDN

Aunque NFV y SDN son similares, ambas tienen claras diferencias. Tanto SDN como NFV son una forma de proporcionar funciones de red y automatización de redes heredadas, pero el objetivo de SDN es diferente de NFV.

SDN consiste en la softwarización de las redes con el fin de simplificarlas, hacerlas más flexibles y reducir costes. Para ello usa un controlador encargado de ejecutar el software que gestiona elementos de red física como *switches*, *firewalls* y *routers*, implementándose en la parte superior de la infraestructura de red física sobre la que se ejecuta la nube.

NFV consume estas SDN como parte de una solución más grande y luego agrega funcionalidad adicional sobre las SDN. Algunos ejemplos de esto son *firewalls* virtuales, filtros de contenido, aplicaciones antivirus, balanceadores de carga, *routers*, etc. Estas funciones adicionales se denominan VNF o funciones de red virtuales. Aunque SDN juega un papel importante en el aprovisionamiento de recursos de NFV, la misión de NFV es virtualizar las funciones del nivel de aplicación sobre las de SDN [5].

2.3.3. ETSI

En los últimos años, la arquitectura NFV ha crecido más allá de ser simplemente una prueba de concepto para el administrador de infraestructura virtual (*Virtual Infrastructure Manager*, VIM) principal que se utiliza para orquestar herramientas de gestión y orquestación de servicios automatizados para la infraestructura de NFV. Sin embargo, para definir mejor las especificaciones de lo que debería ser una plataforma de NFV, los grupos de expertos de OpenStack y los líderes de la industria de las telecomunicaciones han creado un gabinete específico alrededor de las NFV en Telecomunicaciones.

Fundada en 2012 por siete de los mayores operadores de redes de telecomunicaciones, la ETSI (*European Telecommunication Standars Institute*) creó el Grupo de especificación industrial de facto para NFV. Tras 5 años y más de 100 publicaciones después, lo que comenzó como una prueba de concepto ahora se ha convertido en la base para el estudio de mecanismos de de interoperabilidad. Los resultados de estas pruebas producen estándares para las organizaciones miembro y la industria europea de TIC en general.[6]

La ETSI actualmente está trabajando en la definición de la arquitectura y los requisitos para las VNF con el fin de:

- Maximizar la estabilidad y garantizar la confiabilidad del servicio.
- Integración fluida con plataformas existentes y servicios heredados EMS, NMS, OSS, BSS y de orquestación de red.
- Desarrollo de soluciones altamente efectivas y portátiles que tengan una amplia aplicación para proveedores de servicios.
- Maximizar la eficiencia en la migración a nuevas plataformas virtualizadas.
- Simplificar y optimizar las operaciones de telecomunicaciones.

2.3.4. El rol de OpenStack en NFV

En la imagen de la Fig.2.7 podemos ver la base para construir todos los sistemas NFV. En la parte de la izquierda se muestra la arquitectura general de NFV y los principales componentes mientras que en la derecha tenemos como esos componentes funcionan conjuntamente.

El rol de OpenStack es del VIM. El VIM orquesta el hardware contenido en el cuadro de infraestructura NFVI (NFV Infrastructure) para ejecutar las VNFs.

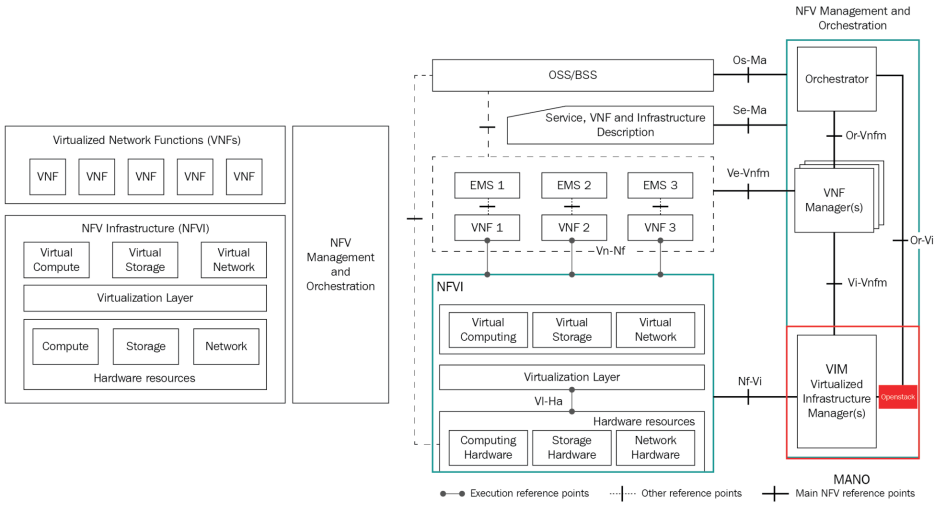


Figura 2.7: ETSI MANO Architectural Framework. Arquitectura para NFV.

Fuente: ETSI NFV architecture

Capítulo 3

Estado del arte

A pesar de ser la elección de OpenStack un requisito del proyecto, los motivos de su elección frente al resto de plataformas libres y de pago son varios.

En este capítulo vamos a proceder a describir los servicios y características *cloud* que ofrecen algunos de los proveedores de servicio IaaS principales y justificaremos la elección de OpenStack haciendo una pequeña comparativa.

También veremos algunas de las empresas más fuertes que ofrecen un servicio de computación en la nube basado en OpenStack para a continuación justificar la vía por la que optaremos.

Para finalizar el capítulo veremos algunos datos que nos darán una clara imagen de la importancia de OpenStack en los despliegues de IT y el panorama TIC actual.

3.1. Plataformas de Cloud Computing

Al desplegar un proyecto de IaaS pretendiendo crear nuestra propia nube, inmediatamente se nos vienen a la cabeza proveedores de servicio de pago como Amazon Web Services, Rackspace Cloud Server, Mirantis, Microsoft Windows Azure, VMware o Google Cloud.

Existen otros proveedores que ofrecen plataformas open source como es el caso de Eucalyptus, OpenNebula o CloudStack.

Sin mas dilación, vamos a ver las características principales de algunos de estos proveedores (dos de pago y dos open source) los cuales hemos elegido por su relevancia dentro de los servicios *cloud*.

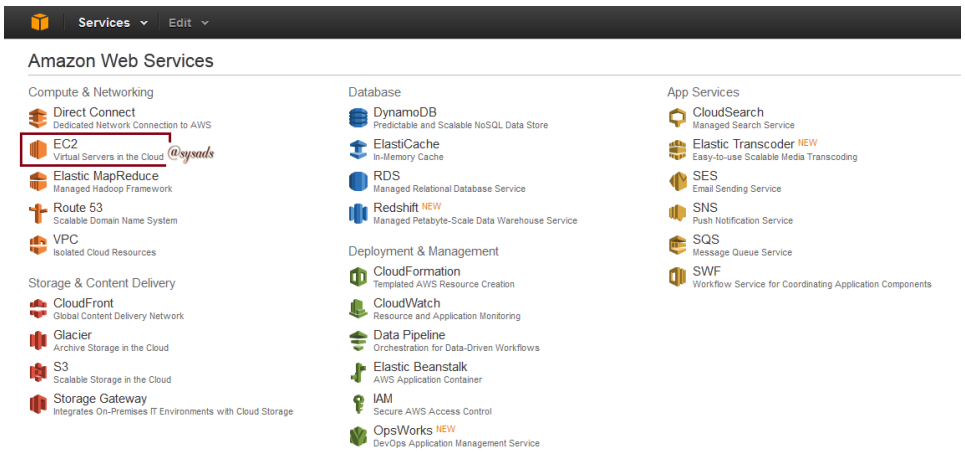


Figura 3.1: Panel de control de Amazon Web Services.

Fuente: Amazon

3.1.1. Amazon Web Services

Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona capacidad informática con tamaño modificable en la nube. Está diseñado para facilitar a los desarrolladores recursos informáticos escalables vía web a través de una sencilla interfaz cuyo panel de control podemos ver en la Fig.3.1. Según la propia empresa ofrecen [7]:

- Reducción del tiempo necesario para obtener y arrancar nuevas instancias de un servidor en minutos, lo que permite escalar rápidamente la capacidad, ya sea aumentándola o reduciéndola, según cambien sus necesidades.
- Modelo económico consecuente. Pago sólo por la capacidad que se utiliza realmente.
- Amazon EC2 proporciona a los desarrolladores las herramientas necesarias para crear aplicaciones resistentes a errores y para aislarse de los casos de error más comunes.
- Compatibilidad con otros servicios de AWS.
- Fiabilidad. El servicio se ejecuta en los centros de datos e infraestructura de red acreditados de Amazon.
- Seguridad. Mecanismos de VPC, listas de acceso, VPN IPsec e instancias aisladas.
- Asequibilidad. Amazon EC2 ofrece ventajas financieras dentro su corporación.

3.1.2. VMware

VMware es una compañía suministradora de servicios de virtualización por software donde las máquinas virtuales proporcionan un ambiente de ejecución similar, a todos los efectos, a un computador físico. Entre sus servicios de virtualización ofrece [8]:

- Virtualización de servidores.
- Virtualización de redes.
- Virtualización de escritorios.
- Virtualización de aplicaciones.
- Virtualización de almacenamiento.

3.1.3. VMware vSphere

Dentro de las opciones que nos brinda VMware cabe destacar VMware vSphere, diseñado para organizaciones que desean optimizar los activos de infraestructura tecnológica existentes y demorar las costosas expansiones del centro de datos, *VMware® vSphere® Standard Edition* proporciona una solución de consolidación básica de las aplicaciones, a fin de reducir drásticamente los costes de hardware, además de acelerar la implementación de las aplicaciones sin tener que programar ningún tiempo de inactividad.

VMware vSphere permite a los usuarios ejecutar aplicaciones críticas para el negocio con confianza y responder con mayor rapidez a las necesidades empresariales acelerando el cambio hacia la computación en la nube para los centros de datos.

Esta herramienta ofrece una serie de servicios [9] como vemos en la Fig.3.2 que la dotan de ventajas entre las que destacamos:

- Mayor eficiencia gracias a la automatización y mejora del rendimiento del hardware pasando del 15 al 80 % de utilización.
- Disminución de gastos de propiedad y operativos.
- Escalabilidad y disponibilidad.
- Plataforma basada en estándares.

Figura 3.2: Servicios de VMware vSphere.

Fuente: VMware

3.1.4. Apache CloudStack

Apache CloudStack es un proyecto de alto nivel de la Fundación de Software Apache. El proyecto desarrolla software de código abierto y nubes con infraestructura como servicio (IaaS). Este proyecto provee las herramientas necesarias para entregar nubes públicas de manera fiable y escalable. Entre sus características se encuentran las siguientes [10]:

- Trabaja con hipervisores como Xen, KVM, HyperfV o Vmware ESXiconVsphere.
- Provee una interfaz sencilla para el manejo de toda la operativa de un IaaS por medio de una interfaz visual o de comandos.
- Tiene una API Nativa.
- Compatibilidad con Amazon S3 y EC2.
- Maneja almacenamiento para instancias basado en los hipervisores además de plantillas, capturas de instancias e Imágenes ISO como almacenamiento secundario.
- Servicios de redes desde la capa de enlace a la de aplicación como por ejemplo: DHCP, NAT, cortafuegos y VPNs.
- Está separada en tres partes: Red, cómputo y almacenamiento.
- Soporte multi-usario o multi-tenant: a un proyecto pueden acceder varios usuarios concurrentemente.

3.1.5. OpenNebula

OpenNebula surge como un proyecto de investigación llevado a cabo desde la Universidad Complutense de Madrid. Esta enfocado en la computación distribuida, virtualización y plataformas IaaS bajo una licencia Apache 2.0.[11]

Esta plataforma esta dotada con algunas funcionalidades que convierten a OpenNebula en un gestor de VDC (Virtual Data Center) con el que encargarse desde la capa más básica de red y almacenamiento, hasta los procesos de gestión de usuarios, tiempos de uso, explotación de recursos y escalabilidad:

- Gestión de recursos flexible y despliegue de máquinas pre-configuradas.
- Gestión de usuarios: uso, facturación, provisionamiento.

- Gestión de perfiles de seguridad.
- Gestión de redes.
- Gestión de almacenamiento.
- Mecanismos de alta disponibilidad y clusters.
- Creación de virtual centers, zonas o nubes híbridas.
- Aplicaciones en Market App.
- Servicios de monitorización.
- API para integración.
- Sistema de Hooks.
- Compatible con hipervisores Xen, KVM, QUEMU, VMWare ESXI, ESX Server.

3.1.6. Comparativa de las plataformas Cloud de pago con OpenStack

El principal motivo de la elección de OpenStack frente a otras plataformas de pago como VMware o AWS, es precisamente el económico. La filosofía del proyecto es que se realice mediante una plataforma *cloud* de software libre apta para la investigación.

Además OpenStack cuenta con todos los servicios que estas plataformas de pago ofrecen. La principal diferencia radica en el hecho de que las plataformas de pagos ofrecen sus propia infraestructura y el despliegue y uso es mucho más sencillo e intuitivo. Basta con contratar aquellos servicios que queramos y estos serán administrados por nosotros.

Esto no es obviamente algo que deseemos pues el objetivo de nuestro proyecto es realizar el despliegue de nuestra propia infraestructura y adaptarla a aquellos recursos y servicios que queramos y de los que podamos disponer.

3.1.7. Comparativa de las plataformas Cloud open source con OpenStack

Eucalyptus, OpenNebula y CloudStack, son los principales competidores a la hora de realizar nuestra elección frente a OpenStack.

El primer caso, Eucalyptus, fue absorbida por HP y hoy día ya no existe como plataforma libre, razón por la cuál no hemos hablado de dicho proyecto.

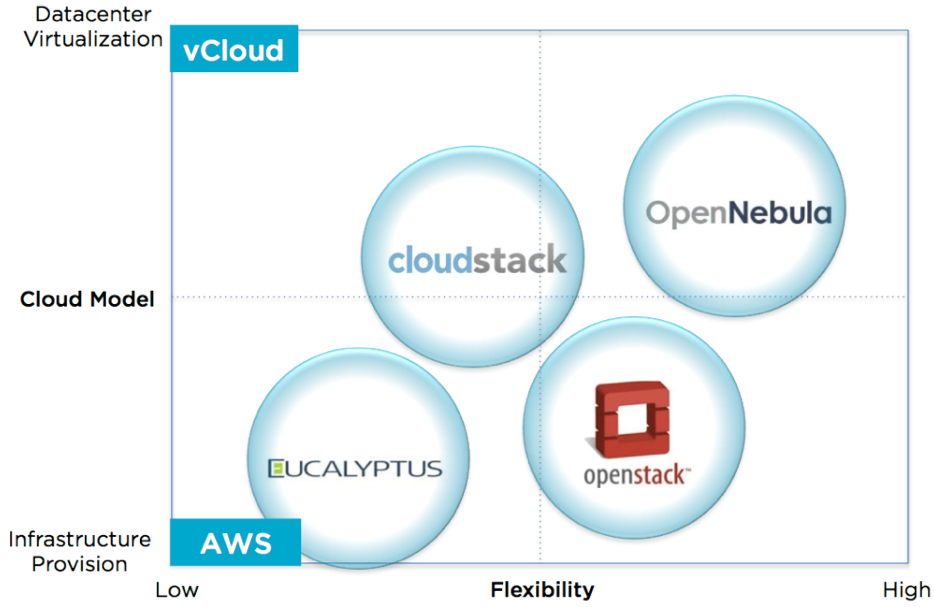


Figura 3.3: Comparativa entre clouds open source en función del uso y la flexibilidad.

Fuente: Ignacio M.Llorente, "Eucalyptus, CloudStack, OpenStack and OpenNebula: A Tale of Two Cloud Models", OpenNebula, 2013

Con respecto a OpenNebula y CloudStack, una característica fundamental para tomar la decisión se representa en la Fig.3.3 extraída de la web de OpenNebula [12] donde podemos ver una comparativa de las distintas herramientas en función de dos criterios:

- El modelo de cloud:
  - Virtualización del centro de datos. Entender la nube como una extensión de la virtualización en el centro de datos.
  - Provisión de infraestructura: Herramienta de aprovisionamiento para suministrar recursos virtualizados según demanda.
- La flexibilidad.

Cómo vemos, OpenStack se postula como la plataforma más flexible para construir nuestra IaaS. Además, posee todas las características de sus competidores incorporando una actualización constante de los proyectos que forman parte de esta herramienta y que podemos incorporar a nuestro proyecto de forma modular en función de nuestras necesidades.

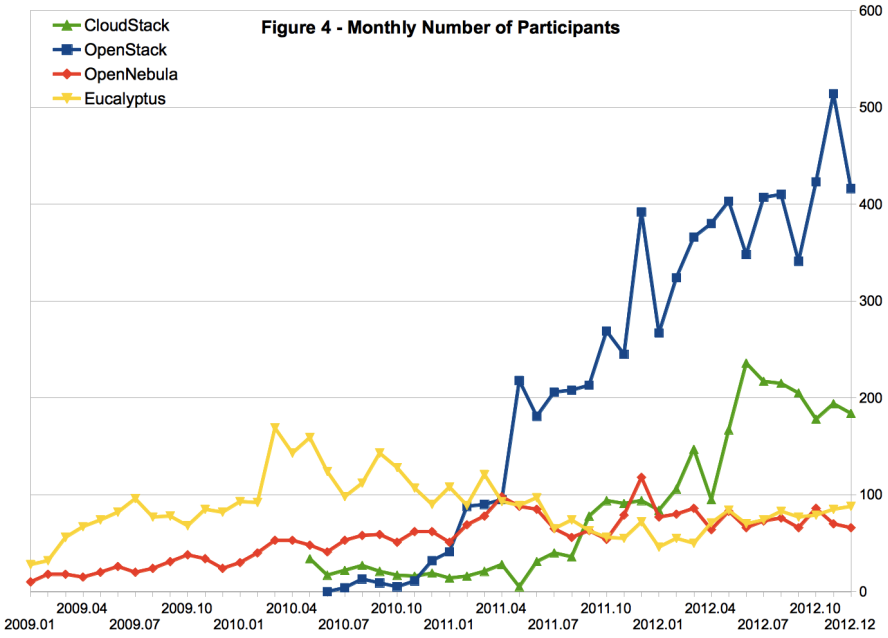


Figura 3.4: Número de participantes mensual.

Fuente: Qingye Jiang, "OpenStack vs OpenNebula vs Eucalyptus vs CloudStack", Open Source IaaS Community Analysis, 2013

Para acabar la comparación, un estudio realizado en 2013 por Qingye Jiang [13] en el que hace una comparativa de estas cuatro plataformas, ya mostraba la clara tendencia de comunidad, empresas y desarrolladores a usar OpenStack. En la Fig.3.4 vemos una extracción del estudio en el que se aprecia el número de participantes mensual en los proyectos y como OpenStack irrumpió desde un inicio tomando clara ventaja al resto de soluciones.

3.2. Vendedores de OpenStack

Son varios los proveedores que ofrecen sus productos y servicios junto con OpenStack. Estos proveedores abarcan desde distribuidores de hipervisores con estrecha integración a su plataforma hasta proveedores en la nube que ofrecen versiones de cloud públicas, privadas o híbridas. En lo que sigue de la sección, nombraremos algunas de las más representativas del mercado [14].

3.2.1. VMware Integrated OpenStack

De esta solución podemos destacar:

- VMware Integrated OpenStack viene con un proveedor de hipervisores.
- Se puede obtener de forma gratuita con una licencia *enterprise plus* de VMware.
- Se integra muy bien en el cliente web vSphere y despliega un OpenStack enfocado a un entorno productivo en solo unos pocos clics.
- Está estrechamente integrado con el hipervisor ESXi Hypervisor.
- La última versión integrada de OpenStack es *Kilo*, de 2015, ya sin soporte.

Su popularidad radica en el hecho de que existen muchas empresas que ya han hecho una inversión considerable en términos de licencias de VMware para hipervisores [15].

3.2.2. Rackspace Cloud

Rackspace necesita una mención especial, no solo porque ejecutan una famosa nube pública basada en OpenStack, sino también porque si no fuera por ellos, ni siquiera tendríamos OpenStack ya que fueron Rackspace y NASA los que comenzaron este proyecto en 2010. Sin embargo, todavía están en la versión *Icehouse* (de 2014, sin soporte) con el hipervisor Xen [16].

3.2.3. HP Helion

En el segmento de software libre y código abierto (FOSS) para productos en la nube. OpenStack y Eucalyptus fueron dos productos que resolvieron los mismos problemas. HP adquirió Eucalyptus y lo ha agregó a sus ofertas en la nube de Helion. Actualmente incorpora también OpenStack pudiendo elegir entre ambas opciones [17].

3.2.4. Cisco OpenStack

Cisco tiene una distribución OpenStack que se ejecuta en su chasis y proporciona soluciones en la nube, en su mayoría privadas, para las empresas que permiten una fácil implementación de una instalación compatible con OpenStack en sus centros de datos [18].

3.2.5. Mirantis OpenStack

Mirantis OpenStack es uno de los más flexibles y al mismo tiempo, una distribución abierta de OpenStack que además ofrece servicio de soporte técnico. Además esta empresa es un referente en la adopción de soluciones que marcan el futuro de la gestión de los centros de datos.

En cuanto a los hipervisores, se abarca desde Xen, Hyper-V, ESXi, LXC (contenedores Linux), QEMU y KVM. Por lo tanto, si se desean opciones más respaldadas en términos de hipervisores, esta será la elección ideal [19].

3.2.6. SwiftStack

SwiftStack es un ejemplo de implementación parcial de OpenStack. Solo implementa Swift, uno de los servicios de almacenamiento de objetos de OpenStack [20].

3.2.7. IBM Cloud Manager

El administrador de IBM Cloud pertenece al gigante tecnológico IBM, que proporciona integración con el hipervisor z/VM ejecutándose en mainframes. También proporciona conjuntos de herramientas de gestión junto con su distribución. Su lanzamiento actual se basa en *Juno* [21], de 2014 y también fuera de soporte.

3.2.8. Suse Cloud

Basada en OpenStack y Crowbar, esta oferta de nube privada admite implementaciones mixtas de hipervisor en la nube basadas en la versión de OpenStack de *Icehouse* [22].

3.2.9. Sobre los vendedores

De nuevo, a la hora de implementar nuestra IaaS, no vamos a optar por ninguno de los vendedores citados sino que partiendo de la instalación de Ubuntu Server en nuestros servidores, realizaremos nuestro propio despliegue por varios motivos:

- El primero vuelve a ser el económico. Como proyecto de investigación y prueba de concepto que estamos realizando, la barrera económico es clave en el desarrollo del proyecto.

- Todas las soluciones citadas dan soporte sólo a algunos de los proyectos que forman OpenStack. De este modo, proyectos menos conocidos o usados por su naturaleza, como el caso de Tacker, no tendrían cabida. Además, así podremos incorporar sólo aquellos proyectos que nos ayuden a cumplir nuestros objetivos.

- Otro motivo es la naturaleza académica del proyecto. Realizando nuestro propio despliegue alcanzaremos un conocimiento mucho más profundo acerca de los aspectos relativos al despliegue de una nube IaaS.

- Por último, la mayoría de distribuidores se basan en versiones de OpenStack que ya no tiene soporte. En nuestro caso elegiremos para el desarrollo **Queens**, que es la última versión estable [23] del proyecto e incorpora actualizaciones de todos los servicios y proyectos de OpenStack.

3.3. El papel de OpenStack en el ámbito IT

El proyecto OpenStack surge de la colaboración global de desarrolladores tecnológicos y de computación en la nube que crearon esta plataforma. Cientos de las marcas más grandes del mundo incluyendo AT&T, Bloomberg, Best Buy, Comcast, eBay, PayPal, SAP, Time Warner Cable, Verizon, Visa, Walmart, Wells Fargo y Yahoo, por nombrar algunos, confían en OpenStack para el funcionamiento diario de sus negocio, reduciendo costos y ganando en movilidad.

OpenStack es utilizado por el 50 % de la lista de las compañías que más ingresos generan en EE.UU. conocida como *US Fortune 100* [24], que abarca industrias que incluyen todo tipo de servicios, finanzas, fabricación, medios de comunicación, investigación, gubernamental, universitarios, venta minorista, tecnología y telecomunicaciones. Para mostrar este hecho en términos económicos, en la Fig.3.5 se muestra un estudio del portal de estadísticas *Statista* [25] donde se estima que las ganancias que generará la plataforma solo este año serán de aproximadamente 3.46 billones de dólares americanos con una tendencia anual creciente.

OpenStack está basado en estándares abiertos. Uno de los principales elementos de OpenStack es la API abierta (*Application Program Interface*), que es un conjunto de definiciones de rutina, protocolos y herramientas para crear software y aplicaciones. Debido a que la API OpenStack está tan bien desarrollada, atrae a los desarrolladores, facilitando su trabajo.

Lanzado en 2010, el proyecto OpenStack, cuenta con una de las comunidades de código abierto de más rápido crecimiento en el mundo, respaldado por esta comunidad vibrante de desarrolladores y algunos de los nombres

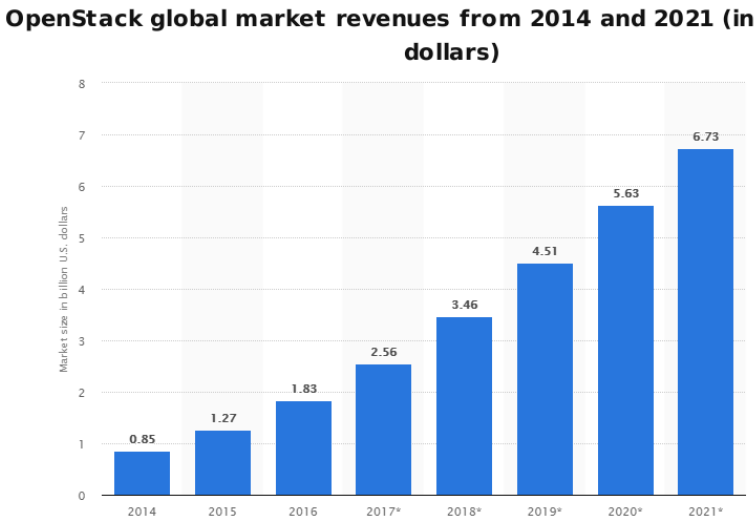


Figura 3.5: Ingresos globales de mercado de OpenStack de 2014 a 2021 en miles de millones de dólares estadounidenses .

Fuente: Statista

más grandes en la industria hasta la fecha. Posee una contribución de más de 20 millones de líneas de código por más de 90000 personas y 600 empresas en 185 países. En la Fig.3.6 podemos ver un resumen de las dimensiones del proyecto hasta la fecha.

Por tanto, el hecho de ser una plataforma open source unido a que es la herramienta más usada para entornos *cloud* en la actualidad y la que cuenta con la mayor comunidad en su ámbito y un número enorme de empresas contribuidoras fundamentales en cualquier desarrollo tecnológico de hoy día, hace que la elección sea sencilla.

3.3.1. Casos de éxito

OpenStack es el producto de nube más exitoso, las implementaciones con él son demasiadas para detallarlas. En esta sección, veremos algunos de los casos de uso más comunes donde se puede ver OpenStack en acción.[26]

- **Enterprise Private Cloud**. Uno de los casos de uso más comunes si nos encontramos en la organización de IT de cualquier empresa que opte por ofrecer una nube privada a sus diferentes unidades de negocios, que ahora demandan agilidad, flexibilidad y menor tiempo de lanzamiento al mercado, es sin duda OpenStack.

  Algunas de las empresas que lo han adoptado son eBay, Alcatel-Lucent,

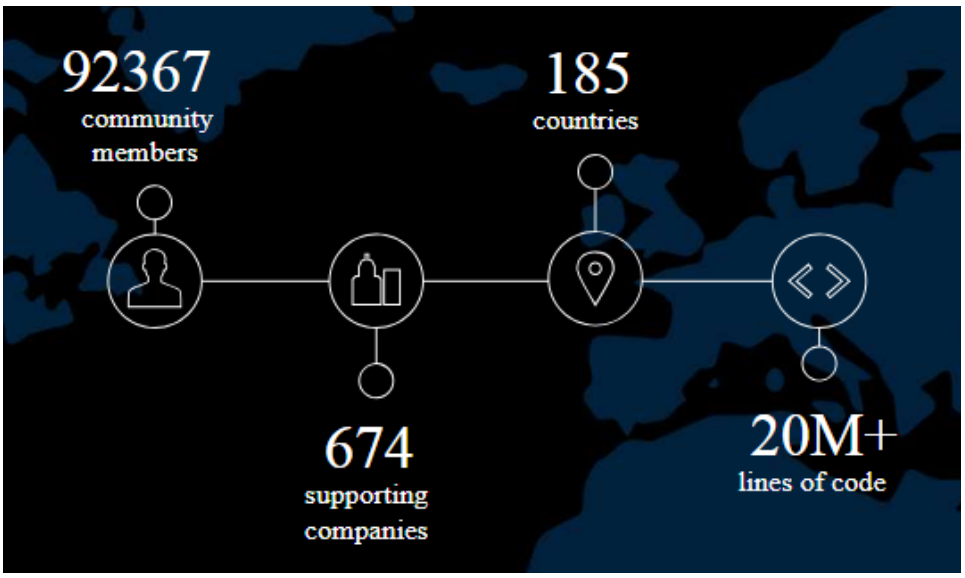


Figura 3.6: Alcance de OpenStack.

BMW, PayPal, NASA y Sony, entre otras.

- **Proveedores de servicio**. Si miramos líneas de negocio de proveedores de servicios, como un proveedor de centro de datos, es posible que este desee comenzar a ofrecer servicios en la nube. La naturaleza distribuida de OpenStack puede ayudarlo a crear una nube con el fin de proveer del servicio de un centro de datos virtualizado a sus usuarios, agregando algunas integraciones adicionales por encima de la oferta estándar de OpenStack, como conjuntos de herramientas y SLA.

  Hay varios proveedores de servicios que usan OpenStack hoy como AT & T, Telstra, CCS (Cisco Cloud Services), Korea Telecom, Dream Host, y más.

- **Escuelas / Centros de investigación**. Incluso las escuelas usan OpenStack en sus laboratorios para proporcionar rápidamente diferentes tipos de cargas de trabajo para los estudiantes, el personal y los profesores de investigación que están llevando a cabo proyectos o investigando en varios campos de su estudio. No depender del equipo de IT reduce en gran medida el tiempo requerido para comenzar un proyecto.

  Algunos de los ejemplos notables son CERN, MIT, CSAIL, etc.

- **Proveedores web, SaaS**. Este tipo de empresas necesitan agilidad. Hay varios cientos de ellos y sus criterios de éxito dependen de cuán rápido puedan incorporar nuevas características a sus productos, por

lo tanto, el *Dev/Test* y todo el paradigma *DevOps* para ellos se convierte en la clave para la supervivencia y OpenStack puede ayudarlos a lograrlo. Estas compañías inevitablemente usan OpenStack o un equivalente para abordar esta tarea.

Algunos ejemplos en este segmento serían MercadoLibre.com o Platform 9.

Capítulo 4

Planificación y costes

El primer paso, una vez que hemos esbozado la idea del proyecto que tenemos entre manos, consiste en realizar una planificación detallada en la que establezcamos y definamos de manera precisa las tareas necesarias para abordarlo identificando todos los aspectos y dependencias que pueden influir en el correcto desarrollo del mismo.

Para lograr la implantación de nuestra *cloud* y dar así un servicio de IaaS, tenemos que pasar por diversas fases de desarrollo que comentaremos en el siguiente punto. Además, se necesita de una planificación temporal que acompañe a las tareas y que veremos también en este capítulo.

No podemos olvidar los recursos necesarios para la elaboración del proyecto tanto humanos, como hardware, software y servicios. Estos recursos se han escogido, en ocasiones, basándonos en el uso de los mismos durante la realización de los estudios de Grado.

Finalmente concluiremos este punto con un desglose del coste estimado de los distintos elementos del proyecto de los que iremos hablando a lo largo del capítulo y que darán un coste final total como colofón.

4.1. Fases de desarrollo

En esta sección vamos a listar todas las tareas que se van a llevar a cabo, agrupándolas en distintas fases y acompañándolas de una pequeña descripción.

Las distintas fases de desarrollo junto las tareas de las que se componen, así como las fechas de inicio y fin y el número de días empleado para cada una se muestran en la Fig.4.1. Como se puede apreciar existe una clara relación entre las tareas y la estructura de la memoria 1.3. Además tenemos los correspondientes diagramas de Gantt que muestran la contemporización de

Número Tarea	Descripción	Inicio	Fin	Duración (días)
	Estudio del estado del arte	01/11/17	24/11/17	18
Tarea 1	Evaluación de alternativas similares	01/11/17	24/11/17	18
	Análisis	24/12/17	19/01/18	20
Tarea 2	Inmersión teórica en OpenStack	27/11/17	05/01/18	30
Tarea 3	Establecimiento de requisitos	25/12/17	12/01/18	15
Tarea 4	Selección de módulos de OpenStack	08/01/18	19/01/18	10
	Diseño	21/01/18	27/04/18	70
Tarea 5	Diseño de la infraestructura	22/01/18	09/02/18	15
Tarea 6	Adquisición de equipos y recursos	28/02/18	27/04/18	43
	Implementación	05/02/18	20/07/18	120
Tarea 7	Pruebas de implementación	11/02/18	30/03/18	35
Tarea 8	Puesta en marcha de equipos	23/04/18	11/05/18	15
Tarea 9	Instalación del software necesario	14/05/18	20/05/18	5
Tarea 10	Despliegue de la infraestructura	20/05/18	08/06/18	15
Tarea 11	Creación de escenarios	11/06/18	20/07/18	30
	Pruebas	11/06/18	03/08/18	40
Tarea 12	Comprobación de escenarios	11/06/18	03/08/18	40
Tarea 13	Batería de pruebas	05/08/18	16/08/18	9
	Elaboración de documentación	01/11/17	07/09/18	223
Tarea 14	Recolección de documentación	01/11/17	17/08/18	208
Tarea 15	Memoria	01/05/18	24/08/18	84
Tarea 16	Presentación	24/08/18	07/09/18	11
Total días				223

Figura 4.1: Planificación de tareas.

tareas por meses, en la Fig.4.2 y para una mejor visualización por trimestres en la Fig.4.3.

En estas figuras podemos ver que la duración estimada del proyecto en días laborales desde que se inicia la primera tarea hasta que finaliza la última, entre las cuales está planificada la realizarán algunas tareas en paralelo, hacen un total de 223 días, de los cuales se estima un trabajo de 3 horas diarias en días laborales, dato que tendremos en cuenta en el apartado 4.3.1 de costes humanos.

4.1.1. Estudio del estado del arte

Dentro de esta primera fase, se evaluarán distintas alternativas similares a OpenStack para determinar si es la herramienta idónea para abordar el proyecto.

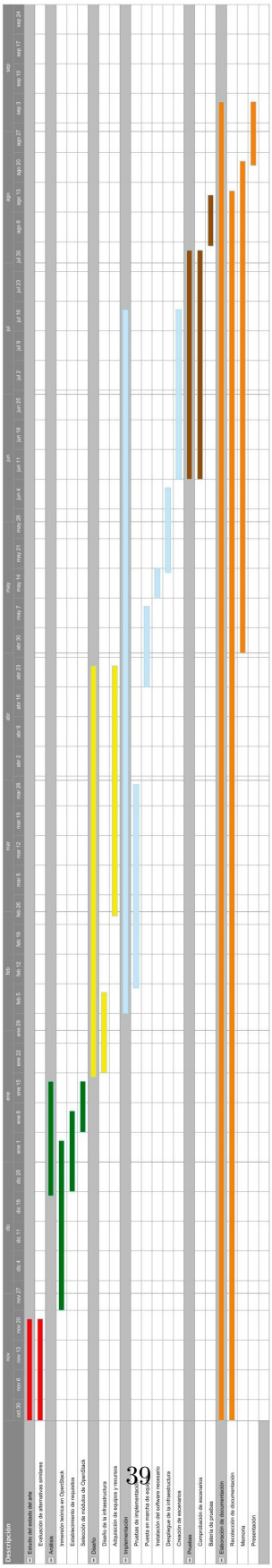


Figura 4.2: Diagrama de Gantt mensual.

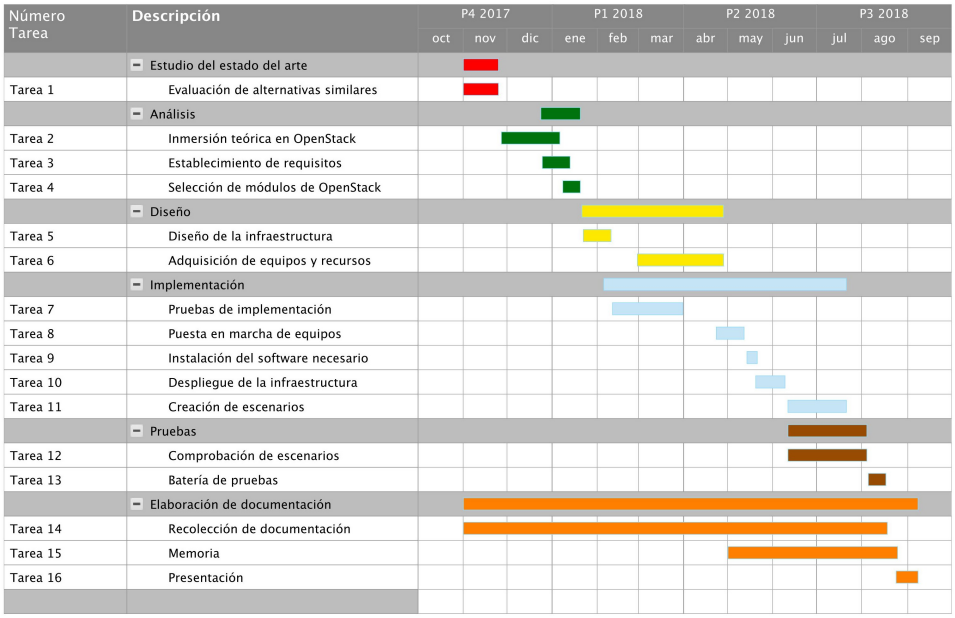


Figura 4.3: Diagrama de Gantt trimestral.

4.1.2. Análisis

Esta fase de análisis es vital, ya que nos servirá par adquirir un conocimiento profundo de OpenStack a nivel teórico y definir los requisitos que necesitaremos para llevar a buen puerto este proyecto.

Una vez realizadas estas tareas tendremos el nivel necesario para decidir que módulos de OpenStack necesitamos instalar para poder gestionar los recursos de orquestación de funciones de red y otros que necesitemos.

En la memoria, el resultado de este apartado se ve reflejado a lo largo de todos los capítulos en los que se analizan distintos temas en función del apartado en el que nos encontremos.

4.1.3. Diseño

Ya definida la planificación de los requisitos, especificaciones y módulos a estudiar para nuestro entorno IaaS, podemos diseñar la infraestructura de red que soportará nuestro servicio a nivel teórico.

En esta fase y en paralelo se irán adquiriendo los recursos hardware y software que necesitaremos tras el estudio de la fase anterior.

4.1.4. Implementación

Con todo a punto, será el momento de desplegar el entorno con el que trabajaremos en OpenStack. Para ello en primer lugar se realizarán una serie de pruebas de instalación e implementación en un ordenador personal para una vez puesta en marcha los distintos equipos necesarios, poder instalar el software de OpenStack y hacer que el despliegue de la infraestructura sea una tarea más ágil y precisa.

En este apartado se crearán también los distintos escenarios que justifican la elaboración de esta memoria.

4.1.5. Pruebas

El despliegue de la infraestructura finalizará con la comprobación del correcto funcionamiento de los distintos escenarios planteados en los que se harán una serie de pruebas para asegurarnos de que el entorno creado se comporta de acuerdo a las necesidades de creación de funciones de red y recursos que tenemos.

4.1.6. Documentación

Esta última fase comienza desde el principio de la elaboración del proyecto. Por un lado desde el inicio hasta prácticamente el final del proyecto se irá recolectando información y fuentes que iremos necesitando para superar las distintas fases aquí explicadas.

Se incluye aquí la realización tanto de la memoria técnica como de la presentación para la defensa final que se expondrá ante la comisión de evaluación.

4.2. Recursos

En esta sección se hará un desglose en distintas subsecciones de los recursos que puedan estar implicados en este proyecto, tanto humanos como de hardware y software, facilitando así la estimación de costes en la sección posterior.

4.2.1. Recursos humanos

Los recursos humanos de este proyecto serán las personas implicadas en el mismo, a saber:

- D. Jorge Navarro Ortiz. Profesor titular de universidad del departamento de *Teoría de la señal, telemática y comunicaciones* de la *Universidad de Granada*. Tutor del presente trabajo fin de grado.

- Alejandro Toledo Juan. Estudiante del *Grado en Ingeniería de Tecnologías de Telecomunicación* en la *Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación*, *Universidad de Granada*. Autor del presente trabajo fin de grado.

4.2.2. Recursos hardware

Para la desarrollo del proyecto se requerirán los siguientes recursos hardware:

- Ordenador portátil Sony Vaio PCG-81112M con procesador Intel Core i7-4710HQ a 2,5 GHz, con una memoria RAM de 8 GB y disco duro de 500 GB. Este portátil será utilizado para la realización de pruebas de implementación de OpenStack antes de la instalación definitiva en los servidores.

- Servidor. Asus G11 CD-K-SP012T. Procesador Intel Core i7-7700 a 3,6 GHz, con una RAM de 16 GB DDR4 y disco duro de 1TB más 128 GB SSD SATA3. Este servidor será el principal a la hora del desarrollo e implementación de nuestra nube.

- Cable RJ-45 Cat.5e. NANOCABLE: RJ45, Cat.5e UTP AWG24, latiguillo de 20 mts. Cable de red con latiguillo incluido de categoría 5e que usaremos para conectar a la toma de red que nos dará conexión a Internet.

- Regleta APC P5BV-SP Regleta con Proteccion 5 Tomas y Coaxial 230V. Regleta que usaremos para conectar la alimentación del servidor a la toma eléctrica.

4.2.3. Recursos software

Del mismo modo, he aquí el listado del software necesario para el desarrollo del TFG.

- Sistema operativo Ubuntu Server 16.04 LTS. Se ha optado por este sistema ya que es el aconsejado por los creadores de OpenStack para realizar la instalación *all-in-one* con DevStack y por el uso habitual del mismo durante el Grado y a nivel personal.

- Openstack. El software necesario para la instalación de todos los componentes necesarios lo encontramos en los repositorios de GitHub de *all-in-one* DevStack.[27]
- Overleaf. Procesador de textos online gratuito basado en LATEX que usaremos para la redacción de la memoria.
- Zotero. Herramienta para la gestión de referencias bibliográficas.
- Smartsheet. Es un software online que nos permite diseñar diagramas de Gantt en distintos formatos.
- Lucidchart. Esta herramienta disponible online, nos permite realizar todo tipo de diagramas: diagramas de red, diagramas de flujo, esquemas, diagramas UML...

4.3. Estimación de costes

4.3.1. Costes humanos

Actualmente en España no existe un salario de referencia para un profesional del sector[28], por tanto la estimación que aparece en la tabla 4.1 se ha realizado en base a lo que suele cobrar un ingeniero de telecomunicaciones en función de su experiencia. Así, el precio por hora del trabajo en el proyecto realizado por Alejandro Toledo Juan, será de 20 €/hora mientras que el realizado por D.Jorge Navarro Ortiz y que se corresponde con las tutorías será de 50 €/hora.

Concepto	**Precio/hora**	**Horas**	**Total**
Trabajo en el proyecto	20 €/hora	669	13380 €
Tutorías	50 €/hora	14	700 €
Total			**14080 €**

Cuadro 4.1: Estimación de costes de recursos humanos.

4.3.2. Costes hardware

En la tabla 4.2 se muestra un desglose de los costes de los recursos hardware. Para el coste del servidor y el PC, se ha tenido en cuenta que el tiempo de vida útil está en torno a los 4 años, es decir, 48 meses. Se estima pues el coste de estos en el proyecto calculando la parte proporcional de ese coste para 5 meses que será el tiempo que se estima estén operativos:

Precio total del PC Portátil: 783.49€. Precio proporcional: 81.16€. Precio total del Servidor: 1132.78€. Precio proporcional: 117.99€.

Concepto	Descripción	Cantidad	Precio (€)
PC Portátil	Sony Vaio: i7, 8 GB RAM	1	81.16
Servidor	Asus: i7, 16 GB RAM	1	117.99
Cable RJ-45 Cat.5e	NANOCABLE: RJ45, Cat,5e UTP AWG24, latiguillo de 20 mts	1	13.56
Regleta	APC P5BV-SP Regleta con Protección 5 Tomas y Coaxial 230V	1	15.99
Total			**222.38**

Cuadro 4.2: Estimación de costes de recursos hardware.

4.3.3. Costes software

Como podemos ver en la tabla 4.3 Todas las tecnologías que se pretenden usar en el proyecto están licenciadas bajo una licencia Open Source y por tanto no tienen ningún coste asociado.

Concepto	Descripción	Cantidad	Precio (€)
Sistema Operativo	Ubuntu Server 16.04 LTS	3	0
Openstack	All-in-one Devstack	3	0
Overleaf	Procesador de texto en LATEX online	1	0
Zotero	Gestor de referencias bibliográficas	1	0
Smartsheet	Diseño de diagramas de Gantt	1	0
Lucidchart	Diseño de diagramas	1	0
Total			**0**

Cuadro 4.3: Estimación de costes de recursos software.

4.3.4. Costes de servicios

Además de los costes ya citados, existen otros derivados de los servicios necesarios para el desarrollo de este trabajo y que tienen que entrar también en la planificación de gastos. Estos aparecen en la tabla 4.4.

Vamos ahora a justificar la estimación de los mismos:

- Como podemos ver en la Fig.4.1, la puesta en marcha de los equipos comenzará el 23/04/18. En la puesta en marcha se contempla la solicitud de acceso al aula donde se alojará el servidor, de la IP pública necesaria para el acceso externo y la propia solicitud de los equipos hasta su posterior recepción. Se prevé disponer del servidor ya instalado el 11/05/2018.

  La desinstalación de la infraestructura una vez finalizado el proyecto y la presentación, está planificada para el 14/07/2018. Esto hace un

total de 127 días, o bien, 3048 horas.

En la tabla 4.4 se ha calculado el precio de la luz teniendo en cuenta que el 11/05/2017, un año antes en la misma fecha que se prevé esté instalado el servidor, el precio medio de 1kWh era de 0.09808€/kW en Endesa.[29]

Mencionar que el precio se ha estimado en base al pico de potencia consumido por lo que será previsiblemente menor al que aparece.

- Línea de acceso a Internet, que será la que nos permita la conexión remota y el acceso a la red externa en el entorno creado. La línea tiene un coste de 30€/mes y estará disponible durante 5 meses.

Concepto	**Descripción**	**Consumo total(kW/h)**	**Precio (€)**
Consumo energético	1 servidores de 150 W	457.2	44.84
Internet	Línea de acceso durante 5 meses a 30€/mes		150
Total			**194.84**

Cuadro 4.4: Coste estimado de los servicios.

4.4. Presupuesto final

Concepto	**Precio (€)**
Recursos humanos	14080
Recursos hardware	222.38
Recursos software	0
Servicios	194.84
Total	**14497.22**

Cuadro 4.5: Presupuesto final estimado.

Una vez identificados todos los recursos que se van a utilizar y lo costes asociados a estos, podemos hacer un presupuesto final en el que aparecen todos ellos 4.6.

Finalmente el coste total estimado del proyecto es de **14497.22€**, CATORCE MIL CUATROCIENTOS NOVENTA Y SIETE CON VEINTI DOS EUROS.

4.5. Desviaciones del proyecto

En este apartado vamos a tratar algunas desviaciones que se han producido a la hora de ajustarse a la planificación del proyecto inicial, debido

principalmente a que cuando se planificó el desarrollo del proyecto estaba en otra situación profesional.

Desde hace unos meses trabajo como ingeniero en un centro de datos por lo que la cantidad de horas diarias planificadas en los recursos humanos ha sido menor. Esto unido a que mi desempeño laboral se realiza en Madrid, haciendo complicado el acceso físico al servidor que era una tarea bastante recursiva ha hecho que el proyecto se aplazase de Septiembre a Noviembre.

El principal impacto que tiene en el proyecto es el económico, pues supone un coste adicional en el consumo energético del servidor y la extensión de mantenimiento de servicios, por lo que se añaden los siguientes sobrecostes al coste final inicial:

4.6. Presupuesto final

Concepto	**Precio (€)**
Total incial	14497.22
Sobrecostes de consumo	21.18
Sobrecostes línea internet	60
Total	**14578.4**

Cuadro 4.6: Presupuesto final estimado.

Por tanto finalmente el proyecto asciende a la cuantía de **14578.4€**, CATORCE MIL QUINIENTOS SETENTA Y OCHO CON CUARETA EUROS.

Los costes de recursos humanos que son el mayor coste generado no se ven afectados por las desviaciones pues no supone un mayor número de horas si no una distribución distinta de estas.

Capítulo 5

Herramientas utilizadas

En este capítulo vamos a ver los orígenes de OpenStack, de donde viene y como podemos contribuir con el proyecto.

Después discutiremos el rol de la *OpenStack Foundation* y cómo los diferentes grupos de *OpenStack Foundation* están trabajando juntos para hacer de OpenStack la solución en la nube más importante de nuestros días.

Además veremos una descripción general de alto nivel de OpenStack y estudiaremos como usar OpenStack para hacer que la administración de la infraestructura de IT sea más flexible.

Para ello, iremos tratando los diferentes proyectos de OpenStack que forman el núcleo de proyectos principales, como son *Nova, Neutron, Glance* o *Cinder* entre otros.

Listaremos también otros proyectos y servicios importantes de OpenStack haciendo énfasis finalmente en aquellos relevantes para la orquestación de VNFs y la consecución de nuestros objetivos.

5.1. Orígenes de OpenStack

OpenStack comenzó en 2010, como un proyecto conjunto de *Rackspace Hosting* y NASA *(National Aeronautics and Space Administration)*. La NASA contribuyó con su plataforma *Nebula*, que luego se convirtió en *Nova*. Rackspace contribuyó con su plataforma *Cloud Files*, que más tarde acabó derivando en Swift.

En abril de 2011, se produjo el lanzamiento de *OpenStack Bexar* en Ubuntu. Más tarde ese mismo año, Debian incluyó *OpenStack Cactus* en su distribución. En 2012, Red Hat anunció una vista previa de su distribución OpenStack también. Desde entonces, muchos otros le siguieron, incluyendo

Oracle, HP y Vmware.

5.2. OpenStack Foundation

The OpenStack Foundation, es una fundación que promueve el desarrollo global, la distribución y la adopción del sistema en la nube, proporcionando recursos compartidos para hacer crecer la nube de OpenStack. También permite a los proveedores de tecnología y desarrolladores ayudar en la producción de software en la nube. [30]

Existen dentro del gran número de de empresas que conforman la *OpenStack Foundation* distintos estatus según el tipo de rol que tengan como miembros. Dependiendo del rol pueden tener una participación mayor o menor en el abastecimiento de fondos para capacitar y promover la comunidad y el software de OpenStack además de encargarse de la gestión del proyecto. Entre las empresas que participan se encuentran Intel, Huawei, Rasckpace, RedHat, SUSE, Mirantis, Cisco, Citrix, Lenovo, Juniper y un sin fin de compañías más.[31]

5.3. El Core de los proyectos en Openstack

OpenStack incluye numerosos proyectos, cada uno de ellos con un estado de adopción diferente. Al mismo tiempo, se están agregando nuevos proyectos de forma regular.

Si deseamos aprender acerca de OpenStack, es importante familiarizarse con los diferentes productos que existen. Dos veces al año, la *OpenStack Summit* toma decisiones sobre la implantación de nuevos proyectos, por lo que se agregan nuevos de forma regular.

En el sitio web de OpenStack [3], se puede obtener una visión general actualizada de todos los proyectos que existen. *OpenStack Foundation* distingue entre servicios principales y servicios opcionales. Los servicios principales que forman el core son *Nova, Neutron, Swift, Cinder, Keystone y Glance*. Una de las cosas interesantes que podemos ver en la web, es la adopción actual. Así, por ejemplo, vemos como Nova es una parte esencial de OpenStack, porque el 93 % de todas las nubes OpenStack lo usan.

También podemos ver que hay servicios opcionales, y algunos de ellos bastante usados en los despliegues de OpenStack, como es el caso de *Horizon*, por ejemplo, que existe desde hace 7 años y lo adoptan el 86 % de todas las implementaciones de OpenStack y otros que van surgiendo como *Magnum* y *Congress*. Por tanto, para cualquier servicio, si vemos los detalles de ellos, podremos ver información sobre los mismos en la *wiki* del proyecto,

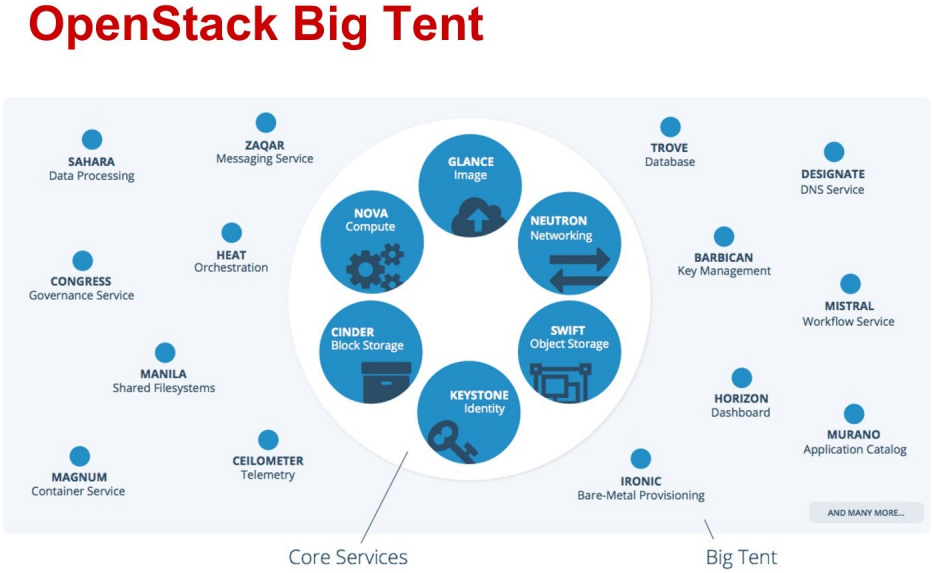


Figura 5.1: Big Tent and Core Services.

Fuente: OpenStack

indicadores de uso, puntuación, o si es un proyecto que está comenzando o ya maduro.

Actualmente existen una serie de servicios dentro de OpenStack conocidos como *Big Tent and Core Services* que podemos ver en la Fig.5.1 entre los que se encuentran el núcleo de servicios de Openstack antes citados, y otros servicios que son habituales encontrar en los despliegues de OpenStack y que varían en función del uso que se le vaya a dar a la infraestructura.

Estos servicios junto a otros relevantes se describen en los siguientes apartados. Además en el anexo A podemos ver el esquema lógico de OpenStack con los componentes y relación que existe entre la mayoría de proyectos que trataremos.

5.3.1. Núcleo de servicios

- **Nova Compute**. Es la interfaz para el hipervisor. Se asegura de que las máquinas virtuales se puedan ejecutar en algún lugar de la nube.

- **Neutron Networking**. Neutron es un proyecto de OpenStack usado para proporcionar conectividad de red como un servicio en la nube a través del uso de switches software como *Open vSwitch* estableciendo así SDN (Software Defined Network) lógicas.

- **Swift Object Storage**. Con Swift podemos usar el almacenamiento en la nube de una forma inteligente, un almacenamiento que no esté vinculado a dispositivos físicos, sino que se organice en objetos binarios que se pueden dispersar por toda la nube de una manera distribuida y replicada.

- **Cinder Block Storage**. Como administradores, podemos usarla para proporcionar almacenamiento persistente a la máquina virtual que estamos implementando en la nube.

- **Keystone Identity**. Keystone nos permite combinar todo lo anterior. Podemos crear usuarios, roles o perfiles y servicios. Con esta herramienta definiremos que usuario tiene acceso a qué servicio específico y cómo los servicios específicos pueden comunicarse entre sí.

- **Glance Image**. Es el servicio de imágenes, necesario para implementar una instancia de una imagen en la nube.

5.3.2. Nova

Nova es quizás el proyecto más importante en OpenStack dentro del núcleo de los mismos [32]. Este proyecto es el responsable de administrar el ciclo de vida de las instancias de cómputo. Ahora bien, ¿qué es una instancia de cómputo? El término de cómputo del inglés *compute* es simplemente otro nombre para Nova, que podrá ser cualquier equipo o servidor en el que esté instalado el servicio y una instancia es una máquina virtual, por tanto, Nova se encarga de ejecutar la máquina virtual.

Para ello, Nova se conecta al hipervisor pero no es el hipervisor en sí mismo. Esto lo hace muy interesante ya que independientemente del hipervisor que se esté ejecutando, bien sea Xen, KVM, VMware, vSphere o cualquier otro, Nova lo puede tratar.

Nova instalará un agente en el hipervisor para asegurarse de que sea compatible con nuestro entorno OpenStack y se hará responsable de crear, planificar y desmantelar las máquinas virtuales bajo demanda, incluyendo los procesos del servicio Nova que se ejecutan en el controlador de la nube, así como los agentes Nova, que se ejecutan en el hipervisor.

5.3.3. Neutron

El siguiente proyecto central de OpenStack es *Neutron*. Neutron posibilita la definición de redes lógicas definidas por software. Las SDN permiten a los usuarios definir su propia red para las instancias que se implementen.[33]

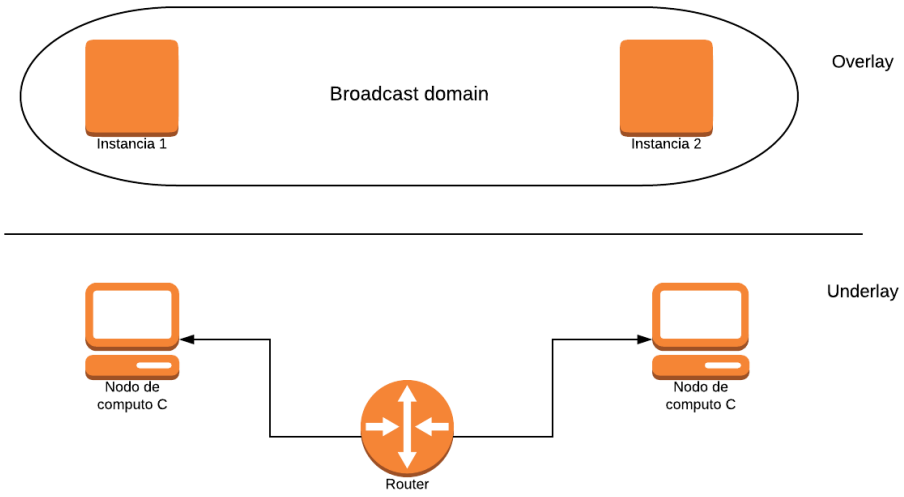


Figura 5.2: SDN en Openstack.

Imaginemos un entorno típico de OpenStack como el de la Fig.5.2. En este supuesto entorno hay dos nodos C de cómputo diferentes. Estos nodos se conectarán mediante el uso de una red física que simulamos como un router en la figura. Llamamos a esta parte, la red subyacente, *underlay network*. En esta red física se realizará enrutamiento y lo necesario para que funcione. Esta parte de la configuración de la red física será normalmente lo que el usuario de OpenStack no conocerá.

El nivel de usuario, que será el nivel más alto, es lo que llamaremos la red de superposición o *network overlay*. En el nivel de usuario, puede haber distintas instancias ejecutándose en lugares diferentes (instancias 1 y 2 ejecutándose cada una en un nodo), pero el usuario puede desear implementar estas instancias como si se usaran en el mismo dominio de difusión (*broadcast domain*), formando así una red lógica.

En un dominio de difusión, las instancias estarán en la misma red, por lo que, ¿cómo puede ser que estén en el mismo dominio de difusión si en la red subyacente tenemos diferentes redes físicas? Esto es exactamente de lo que Neutron se está ocupando mediante el uso de redes definidas por software. Para hacerlo, Neutron necesita interconectar la arquitectura de la red física, para lo cual utiliza una arquitectura conectable (*pluggable architecture*). Esta arquitectura conectable admite muchos proveedores y tecnologías de redes. Por lo tanto, si se ha realizado una gran inversión en alguna tecnología de red patentada, es probable que haya un buen *plugin* de Neutron disponible. La mayoría de los proveedores de red tienen *plugins* de Neutron, por lo que no tendremos por norma general problemas con esto.

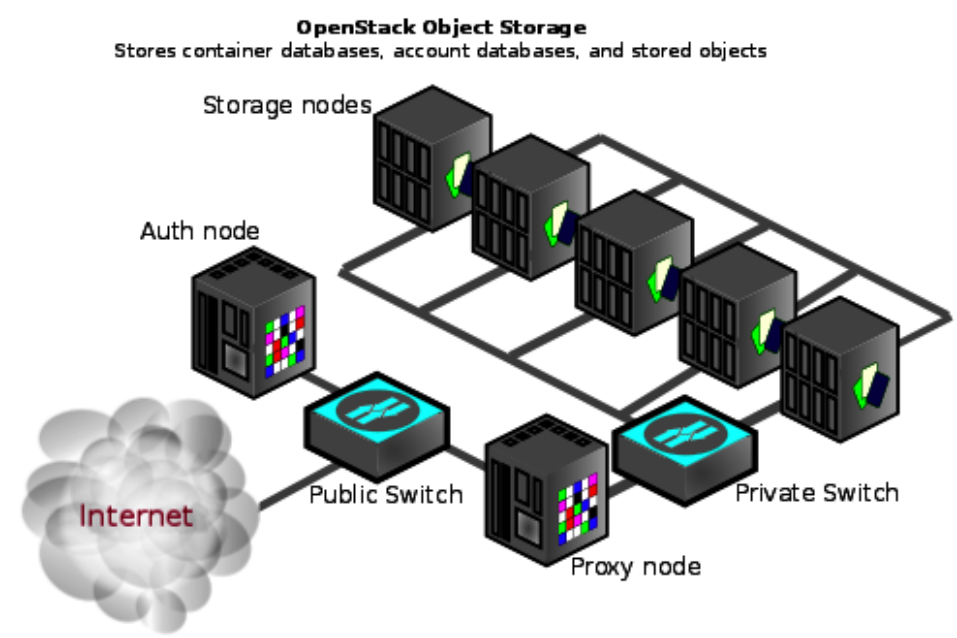


Figura 5.3: Almacenamiento de objetos distribuido con Swift.
Fuente: Mirantis

Además, Neutron proporciona una API para que los usuarios definan las redes y los archivos adjuntos en ellas. Como resultado, para un programador o administrador será relativamente sencillo crear su propio entorno de SDN, como veremos más adelante.

5.3.4. Swift

El siguiente proyecto central de OpenStack del que hablaremos es *Swift*. Swift está diseñado para proporcionar escalabilidad en el nivel de almacenamiento. Funciona con objetos binarios para almacenar datos de una manera distribuida y replicada. [34]

El almacenamiento finalmente termina en un disco duro y un disco duro es un dispositivo físico. El problema con los dispositivos físicos es que son limitados y no muy escalables, por lo que si todo nuestro almacenamiento en la nube estuviese en un servidor que proporciona diferentes discos duros, no tendríamos escalabilidad. Para tenerla, Swift ofrece un modelo de almacenamiento basado en objetos.

Para entender el funcionamiento de Swift nos fijaremos en la Fig.5.3. La idea sobre Swift es que tenemos una aplicación que usaremos para escribir datos. En un entorno OpenStack, la aplicación no escribe un archivo en

el disco duro, sino que esta puede interactuar con el almacenamiento de objetos de Swift que se está comunicando a su vez con muchos nodos de almacenamiento.

En la Fig.5.3 podemos ver distintos nodos de almacenamiento (*Storage nodes*). Swift está usando un proxy y cuando el proxy de Swift recibe datos de la aplicación, va a crear objetos binarios. Estos objetos binarios son los fragmentos de datos.

Digamos que tenemos objetos binarios A, B y C. En Swift, el objeto binario A se puede almacenar en uno de los nodos de almacenamiento, el objeto binario B en otro y el C en otro diferente. Si solo se almacena una vez, el sistema no será tolerante a fallos, por lo que Swift incluye un algoritmo de replicación para almacenar los objetos binarios en múltiples nodos. Por defecto, esto se hará tres veces, es decir, cada objeto se almacenará en tres nodos diferentes, aunque podríamos almacenarlo en un número mayor de nodos si así lo requerimos.

Este proceso hace que el almacenamiento sea más eficiente ya que en el momento en que la aplicación necesita recuperar los datos, se dirigirá al proxy Swift nuevamente y el proxy Swift estará utilizando un algoritmo avanzado para determinar exactamente dónde residen los objetos binarios, enviando llamadas a todos los nodos de almacenamiento involucrados. Debido a todos estos nodos de almacenamiento que podrán funcionar en paralelo, los datos llegarán al proxy de Swift, y por lo tanto, a la aplicación, de una manera muy rápida. Y así es como se consigue con OpenStack hacer que el almacenamiento sea escalable.

Aún tiene más ventajas, imaginemos que estos nodos de almacenamiento son de un terabyte cada uno. Cuando nos estamos quedando sin almacenamiento disponible, es bastante fácil agregar un par de nodos de almacenamiento Swift más, e incluso se pueden requilibrar los objetos binarios que están escritos en la configuración de almacenamiento de Swift.

La aplicación que se usa para hablar con Swift, está utilizando una API RESTful. REST es una forma estándar de comunicación en un entorno OpenStack como ya sabemos. Esto significa que la aplicación no está escribiendo un archivo en un sistema de archivos, sino que está utilizando una llamada RESTful API, que es entendida por Swift proxy. La API RESTful es el lenguaje nativo de OpenStack, y eso convierte a Swift en la opción nativa para el almacenamiento de objetos en OpenStack.

5.3.5. Glance

El siguiente proyecto importante de OpenStack es Glance. Glance se usa para almacenar imágenes de disco de la máquina virtual [35]. Las máquinas

virtuales, que son las instancias, no están instaladas, sino que se generan a partir de una imagen que podemos descargar[36]. Es como arrancar Linux desde un *live CD*. Las imágenes se pueden descargar fácilmente o se pueden crear para que coincidan con las necesidades específicas dentro de una organización.

En la web se proporcionan diferentes imágenes en la nube para diferentes sistemas operativos, que incluyen CentOS, Debian, Fedora, básicamente, todas las principales distribuciones de Linux, e incluso incluye Microsoft Windows.

Una imagen interesante es la imagen de *CirrOS* y lo es porque es muy ligera, solo de trece megabytes, motivo por el cual la usaremos para realizar nuestros despliegues posteriormente debido a la limitación principalmente de RAM de nuestro entorno.

Así pues, Glance es como nuestra tienda de imágenes. Esto significa que si un administrador desea iniciar una instancia, iniciará la instancia desde el almacén de imágenes de Glance y es por ello que Glance es un pilar básico de OpenStack.

Para que todo sea escalable, el almacén de imágenes de Glance normalmente usa el almacenamiento de objetos Swift como un *back-end*. Por supuesto no hay porque utilizar el almacenamiento de objetos de Swift, pero usarlo lo hace más escalable. La otra opción sería usar el almacenamiento local como *backend* para Glance, pero estaríamos usando entonces almacenamiento local y por tanto vinculado a un servidor físico que contiene las imágenes física y que no es muy escalable.

5.3.6. Cinder

El siguiente proyecto esencial de OpenStack es Cinder [37] encargado de proporcionar almacenamiento persistente a las instancias.

Por defecto, el almacenamiento de las instancias es efímero. De nuevo el ejemplo de arrancar desde un live CD en Linux es bastante ilustrativo. Si trabajamos desde un *live CD* e intentamos cambiar algún archivo de configuración, ¿dónde se cambiaría? porque la imagen del *live CD* está cargada en la RAM y realmente no existe en ningún disco duro de almacenamiento.

Este es el mismo problema que tenemos con las imágenes de Glance y es por eso que en OpenStack, el almacenamiento instantáneo es efímero. Cinder es el componente que permite a los administradores asignar almacenamiento persistente a las instancias.

Cinder también puede usar diferentes *backends*. Puede ser un almacenamiento local, que de forma predeterminada será LVM de Linux (*Linux Volume Manager*) o puede incluir almacenamiento de objetos Swift y Ceph,

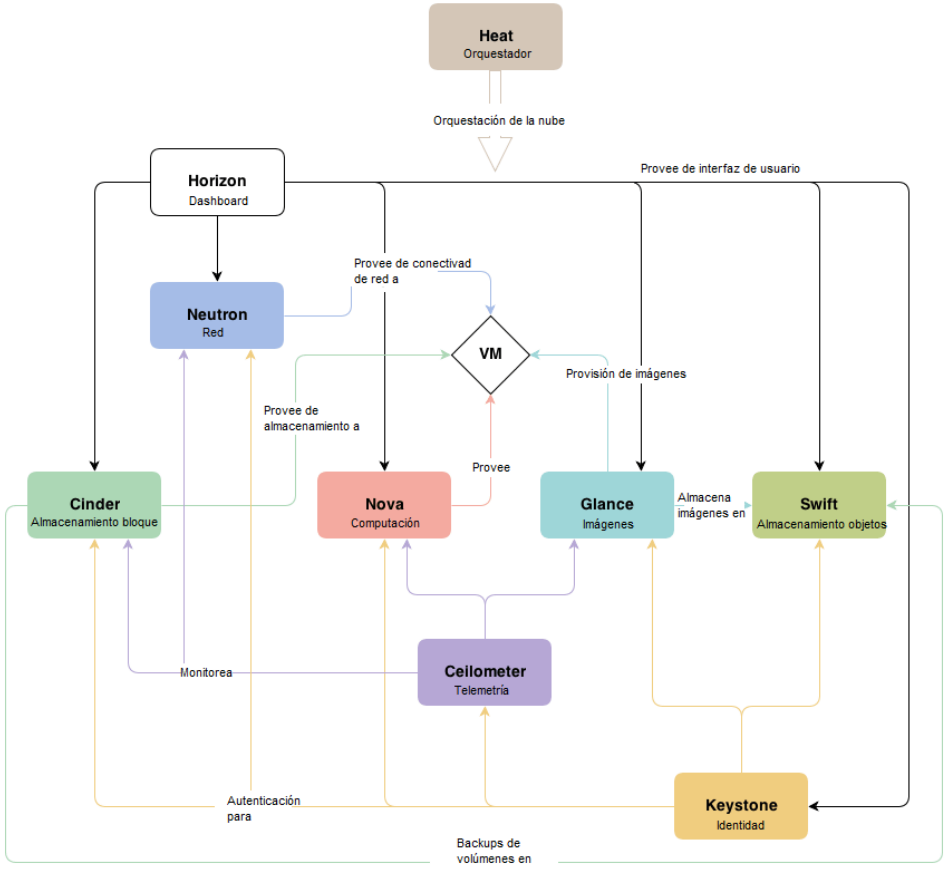


Figura 5.4: Arquitectura conceptual de OpenStack.

Fuente: Daniel Romero Sanchez, "Openstack desde cero - Keystone", DBigCloud, 2015

incrementando así la escalabilidad.

5.3.7. Keystone

Ninguna nube OpenStack podría existir sin los servicios proporcionados por el proyecto *Keystone*. Keystone se usa para la autenticación, autorización y enumeración de todos los servicios y *backends*.[38]

Con servicio nos referimos en este caso a cualquiera de los proyectos que se implementa en OpenStack. De este modo, si queremos poder ejemplo acceder a Nova, Nova debe definirse como un servicio en Keystone y el *backend* proporciona una URL que da acceso al servicio específico. Como administradores, también podremos consultar estos servicios y *backends* como veremos.

Keystone es un elemento central en OpenStack tal como refleja la Fig.5.4

donde aparecen los principales proyectos de OpenStack que hemos ido enumerando y las relaciones comentadas que existen entre ellos.

En Keystone, los usuarios y roles se crean y se asignan a proyectos, que también se conocen como *tenants*. Un proyecto, también conocido como tenant, por lo general, es un cliente de OpenStack. Si OpenStack se usa como una nube pública, pueden ser por ejemplo diferentes empresas que están contratando espacio en la nube en OpenStack y con el fin de distinguir entre los recursos disponibles para un cliente y otro, OpenStack utiliza esta noción de proyectos o tenants.

En un entorno de proyecto, normalmente se crearán cuentas de usuario a las que se asignarán roles específicos y dependiendo del rol asignado a una cuenta de usuario, estos tendrán más o menos privilegios para crear recursos en OpenStack. De todo ello se ocupa Keystone, por lo tanto, Keystone es un repositorio central para todos los servicios y resultados de *backends*.

De forma predeterminada, Keystone utiliza una base de datos, que es la base de datos MariaDB (derivada de MySQL), para almacenar información, lo cual no significa que no podamos cambiarla a otras que queramos usar como OracleDB o bien, usar un directorio LDAP.

5.3.8. *The big tent*

Otros proyectos interesantes de OpenStack que no forman parte de los principales pero que tienen un alto índice de adopción son:

- **Ceilometer.** Ceilometer es conocido como el servicio de telemetría y se usa para medición y registro. Recopila datos que pueden ser consultados por un software de terceros, pero también se pueden usar para obtener estadísticas de uso que se pueden pasar al servicio de Heat. Según la información que obtiene de Ceilometer, Heat puede ampliar o reducir automáticamente la cantidad de recursos en la nube [39]. Actualmente este proyecto está obsoleto y otros están surgiendo para sustituirlo como Panko, Senlin, Aodh o Gnocchi.

- **Trove.** Trove es un *Database as a Service* para OpenStack. Provee a OpenStack de motores de bases de datos relacionales y no relacionales. Su objetivo es ofrecer la funcionalidad de base de datos a los usuarios sin la necesidad de ocuparse de tareas de administración complejas. Como resultado, usuarios y administradores pueden implementar instancias de base de datos fácilmente.[40]

- **Sahara.** Sahara es Big Data en OpenStack. Está configurado para ofrecer una fácil implementación de Hadoop sobre OpenStack. Sahara se ocupa de los parámetros de Hadoop, como la versión, la topología

del clúster y el hardware. El usuario proporciona estos parámetros y Sahara implementará automáticamente el clúster. Sahara puede agregar y eliminar nodos fácilmente al clúster según sea necesario.[41]

- **Ironic.** Ironic es el programa de aprovisionamiento *bare metal* de OpenStack y fue desarrollado para implementar máquinas físicas y no virtuales. Ironic es una API con un conjunto de complementos que interactúan con los hipervisores de *bare metal*. Utiliza PXE (*Preboot Execution Enviornment*) o IPMI (*Intelligent Platform Management Interface*) para aprovisionar y encender máquinas o apagarlas. Ironic también funciona con complementos específicos del proveedor para aportar funcionalidades adicionales.[42]

- **Zaqar.** Zaqar es el servicio de gestión de colas de mensajes en OpenStack. Es una alternativa a una instalación independiente del servicio de mensajería AMQP (*Advanced Message Queuing Protocol*) o ZeroMQ. Zaqar puede usar una API REST basada en HTTP, así como una API basada en websocket para la comunicación. Además permite el manejo de colas de múltiples usuarios y proporciona una alta disponibilidad interna y escalabilidad, lo cual es un reto en los servicios de mensajería independientes.[43]

- **Manila.** Originalmente está basado en Cinder. Manila es el servicio de sistema de archivos de OpenStack. Ofrece una alternativa al almacenamiento local o de objetos, proporcionando así una solución de almacenamiento flexible.[44]

- **Designate.** El proyecto Designate proporciona DNS como servicio para OpenStack: acceso a API REST para administración de dominios y registros, utilizable en entorno multi-tenant, integrado con Keystone para autenticación, se integra con Nova y Neutron para la generación automática de registros DNS y soporte nativo para Bind9 y PoweDNS.[45]

- **Barbican.** Barbican implementa seguridad y administración de claves en la nube, proporcionando una API REST. Está diseñado para almacenamiento, aprovisionamiento y administración de todo tipo de claves, incluidas contraseñas, claves de cifrado y certificados X.509.[46]

- **Magnum.** El proyecto Magnum fue desarrollado para permitir la administración de contenedores en OpenStack. Su objetivo es integrar los motores de orquestación de contenedores como recursos de primera clase en OpenStack. Un ejemplo importante de este tipo de motor es Kubernetes. Magnum permite integrar Kubernetes en un entorno OpenStack para que pueda ejecutar contenedores orquestados de Kubernetes encima de OpenStack. El resultado es que se pueden integrar

contenedores en OpenStack de manera similar a como las instancias se integran encima de Nova.[47]

- **Murano.** El proyecto Murano proporciona un catálogo de aplicaciones para OpenStack. El objetivo de este proyecto es permitir que los desarrolladores de aplicaciones y los administradores de la nube publiquen las aplicaciones disponibles en un catálogo. Esto permite a los usuarios consultar una lista de aplicaciones en la nube de una manera fácil de navegar.[48]

- **Congress.** Congress es un proyecto de *Policy as a Service*. Proporciona mecanismos de gobierno y cumplimiento como servicios en la nube. El administrador puede definir políticas y Congress se encargará de implementar estas políticas.[49]

- **Heat.** Se usa para desplegar un conjunto de instancias en un entorno OpenStack. Esto nos permite desplegar múltiples máquinas virtuales relacionadas de una manera sencilla. Heat usa HOT (Heat Orchestration Template) o AWS CloudFormation template para definir las pilas o conjuntos de instancias orquestando de esta forma el resto de proyecto (ver Fig.5.4). Este proyecto lo trataremos en detalle en la sección 5.6.

5.3.9. Detrás de los proyectos de OpenStack

Detrás de todos estos servicios de OpenStack que forman el núcleo básico, también hay algo más. Siempre se requieren algunos servicios vitales y OpenStack se incluyen entre ellos:

- La sincronización de tiempo o *timestamp*. Esta es necesaria ya que muchos servicios utilizan *timestamps* para la comunicación, por ejemplo en Keystone. Keystone está enviando tickets que se basan en *timestamps* y si los servicios no llegan a un acuerdo en el momento adecuado, entonces, no habrá comunicación.

- Otro servicio es la base de datos de MariaDB de la que hemos estado hablando cuando hablamos sobre Keystone, usada para almacenar toda la información relacionada con la nube. Este ítem es muy interesante porque como administradores, podremos consultar la base de datos y obtener información sobre lo que está sucediendo fuera de ella.

- También está la cola de mensajes o *message queue*. La cola de mensajes es un componente esencial al que los servicios acceden para intercambiar mensajes de forma ordenada entre los distintos servicios necesarios. Es como un servidor de correo SMTP en el manejo del

correo electrónico, un servicio que se encarga de entregar el mensaje al destino correcto. OpenStack incorpora AMQP (*Advanced Message Queuing Protocol*) como servicio de mensajería, siendo el agente predeterminado RabbitMQ.

En una implementación de OpenStack que está automatizada, todo estos componentes se crearán automáticamente. Si por contra optamos por implementar OpenStack manualmente, deberemos encargarnos nosotros mismos de estos tres servicios esenciales.

5.4. La API RESTful

Para comprender OpenStack, hay que entender el funcionamiento de la *API RESTful*. REST es un método genérico que OpenStack usa para proporcionar acceso a todos los componentes del mismo. Todas las APIs de OpenStack son RESTful, lo que proporciona un acceso uniforme. Esto facilita el trabajo para los desarrolladores, ya que se usan los mismos estándares en todas partes y un desarrollador puede usar el mismo lenguaje en todo lo que hace.

El acceso a la API RESTful se puede utilizar al implementar comandos, pero también directamente usando *curl*, un proyecto de software consistente en una biblioteca (*libcurl*) y un intérprete de comandos (*curl*) orientado a la transferencia de archivos y que soporta protocolos como FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, FILE y LDAP entre otros.[50]

Vamos a hacer una pequeña demostración de lo que la API RESTful hace posible. Imaginemos que estamos en una máquina OpenStack. En la máquina OpenStack, podemos usar un comando como este:

```
stack@wimunet4:/opt/stack# curl -s -X POST http://10.5.1.201:5000/v2
    .0/tokens -H "Content-Type: application/json" -d ' ("auth": {"
    tenantName": "'"$OS_TENANT_NAME"'", "passwordCredentials": {"
    username": "'"$OS_USERNAME"'", "password": "'"$OS_PASSWORD"'"}}}'
    | python -m json.tool
```

De este modo, estamos haciendo una llamada a la API de forma directa. Como administradores, no nos interesará hacer esto, pero es una forma de exponer lo que sucede detrás de la interfaz web de Horizon, o de la línea de comandos.

Al usar este comando, estamos accediendo a un *endpoint* de la API utilizando el método POST, por lo que, estamos haciendo un POST a *tokens* y estamos especificando en el *content-type* que el tipo de datos sea JSON, proporcionando además credenciales de autenticación. Por último, redirigimos el resultado a Python, que está utilizando la herramienta JSON.

Como resultado obtendremos:


```
          ],
          "endpoints links": [],
          "name":"Image Service",
         "type":"image"
     },
          "endpoints": [
                {
                    "adminURL:"http://10.5.1.201:35357/v2.0",
                    "id":"5797558684004446bbe92448ffd3f6fw",
                    "internalURL":"http://10.5.1.201:5000/v2.0"
                    "publicURL":"http://10.5.1.201:5000/v2.0"
                    "region":"RegionOne"
                }
           ],
           "endpointss_links": [],
           "name":"keystone",
           "type":"identity"
     }
],
"token":{
    "audit_ids": [
          "3RAGfK16Ssui3kA6M2Cgmg"
      ],
     "expire":"2018-07-28T14:51:152",
     "id":"bb9ad8edda2e434a9b261c539a8bf1ea",
     "issued_at":"2018-07-28T14:51:152.86034ZZ",
     "tenant": {
           "description":"deafult tenant",
           "enabled":"true",
           "id":"7db988ced11842e3af279aaa2425f362",
           "name":"demo"
       }
},
"user": {
     "id":"ae8380b700344341994183c0cd6db4f9",
     "name":"demo",
     "roles": [
            {
                "name": "_member_"
            }
       ],
      "roles_links": [],
      "username": "demo"
```


Esta es la información en crudo que OpenStack proporciona al administrador. Más adelante veremos la existencia de una línea de comandos que hace esta tarea más sencilla y la existencia de una interfaz web, Horizon, que la hace mucho más amigable e intuitiva para el usuario.

Como conclusión, saber que uno de los puntos fuertes de OpenStack es que todo es compatible con la API RESTful y debido a ello es más fácil comunicarse con los diferentes servicios.

5.5. Horizon

Al tratar la API RESTful y en la Fig.5.4 vemos un proyecto del que aún no hemos hablado, Horizon [51]. Horizon es el dashboard de OpenStack y proporciona una interfaz web para una fácil administración de instancias y otras propiedades de OpenStack tanto para los administradores de la nube como para los usuarios que se creen y que quieran gestionar los recursos proporcionados de dicha nube.

Horizon es uno de los componentes más populares de OpenStack. Se usa con mucha frecuencia porque es mucho más fácil de usar que la poderosa interfaz de línea de comando y esa es la razón por la cual los usuarios finales usan Horizon.

Como nota importante, al trabajar desde Horizon notaremos que la perspectiva que se obtiene como usuario administrador es diferente a la perspectiva que se verá como un usuario tenant o invitado. El usuario administrador es responsable de la administración de los tenants así como de la infraestructura de la nube, mientras que un usuario tenant es responsable de administrar el entorno del tenant, de ahí las diferencias que se apreciarán.

5.5.1. API REST vs CLI vs Horizon

Vamos a hacer una pequeña comparativa entre la API, la línea de comandos o CLI *(Command Line Interface)* y Horizon. Hemos visto previamente un comando insertado con el que usamos la API. Esta API puede ser utilizada por cualquiera o cualquier aplicación puede usarla para solicitar información directamente de OpenStack y la respuesta se proporciona en el formato JSON, lo cual no es muy legible.

Si OpenStack no está siendo usado por un desarrollador, probablemente no elija la opción de la API. Es por eso que desde la CLI también podemos usar comandos de OpenStack. Por ejemplo, la lista de puntos finales de OpenStack, que presenta más o menos la misma información que la llamada a la API que acabamos de ver en el apartado anterior usando curl:

```
stack@jorge-All-Series:/opt/stack# openstack endpoint list
```

Y que nos devuelve como resultado los datos de la Fig.5.5, donde mostramos la salida parcial para una mejor visualización de algunos de los servicios que se listan como *endpoints*.

Pero aún así, esta forma de trabajo requiere trabajar con comandos complicados, motivo por el que existe Horizon. En la Fig.5.6 podemos ver el listado de los *endpoints* clicando en "Acceso a la API" al acceder al escritorio o *dashboard* de Horizon. Esta forma de trabajar es mucho más intuitiva al

```
stack@wimunet04:~/devstack$ openstack endpoint list --fit-width
+----------------------------------+-----------+--------------+----------------+---------+-----------+-----------------------------------------+
| ID                               | Region    | Service Name | Service Type   | Enabled | Interface | URL                                     |
+----------------------------------+-----------+--------------+----------------+---------+-----------+-----------------------------------------+
| 08c8bf1e0f9b46c88b8333aa734b642a | RegionOne | neutron      | network        | True    | public    | http://10.5.1.201:9696/                 |
| 08ddc4fb10e94cd4bf446e54d961ef9f | RegionOne | panko        | event          | True    | admin     | http://10.5.1.201:8977                  |
| 0e21f7f55f46462 8b62fcf77dbe34c37 | RegionOne | cinderv3     | volumev3       | True    | public    | http://10.5.1.201/volume/v3/$(project_id)s |
| 1a64319a79da41c7be007e0816a5cf5b | RegionOne | cinder       | block-storage  | True    | public    | http://10.5.1.201/volume/v3/$(project_id)s |
| 1e80bdc6f9074c4e83c0a0bf9894b084 | RegionOne | mistral      | workflowv2     | True    | admin     | http://10.5.1.201:8989/v2               |
| 20e08cab906f44ad8aeb67a5d0688bcf | RegionOne | keystone     | identity       | True    | admin     | http://10.5.1.201/identity              |
| 29197209dcab42a7be97c8de8714eba0 | RegionOne | heat-cfn     | cloudformation | True    | public    | http://10.5.1.201/heat-api-cfn/v1       |
| 2ed6545d89284d6485289f72dd0b7925 | RegionOne | aodh         | alarming       | True    | admin     | http://10.5.1.201:8042                  |
| 36980114ec1a4d5eb78ba459ce272fec | RegionOne | heat-cfn     | cloudformation | True    | admin     | http://10.5.1.201/heat-api-cfn/v1       |
| 3d6061b8ed594ca4a38b1c3521118e72 | RegionOne | panko        | event          | True    | public    | http://10.5.1.201:8977                  |
| 4a0d0e525fe1412f99370c7725639e69 | RegionOne | barbican     | key-manager    | True    | internal  | http://10.5.1.201/key-manager           |
```

Figura 5.5: Lista de endpoints.

ser gráfica que el resto y será la preferida en algunos casos.

5.6. Heat

En el capítulo 7 donde vamos a ver como implementar nuestro entorno, veremos como utilizar el dashboard de Horizon y como trabajar desde la CLI (*Command Line Interface*). En los comienzos del cloud computing, aprovisionar de recursos mediante la línea de comandos pronto dejó de ser viable y los administradores comenzaron a realizar *scripting* en bash u otro lenguaje con apoyo en la CLI para automatizar la creación de entornos. Pero pronto empezó a dar problemas este método. ¿Qué ocurre si ejecutamos un script accidentalmente, o si queremos eliminar sólo algunos recursos, o bien hacer un único cambio? Esta es la motivación del surgimiento de Heat. Además, esta evolución hacia Heat la veremos claramente tras leer el capítulo 7 de implementación, en el que se verá la administración del entorno con estas herramientas, cubriendo así uno de los objetivos del proyecto.

Heat se introdujo en 2013 con la versión *Grizzly* de OpenStack. Se basa en una pila o *stack* que no es mas que una descripción de los parámetros de orquestación donde se indican todos los recursos de OpenStack a aprovisionar tal como veíamos aún sin haber hablado de ello en la Fig.5.4. Heat se ocupa por tanto del aprovisionamiento y la configuración sin necesidad de preocuparnos de las dependencias o el orden de ejecución, creando así una

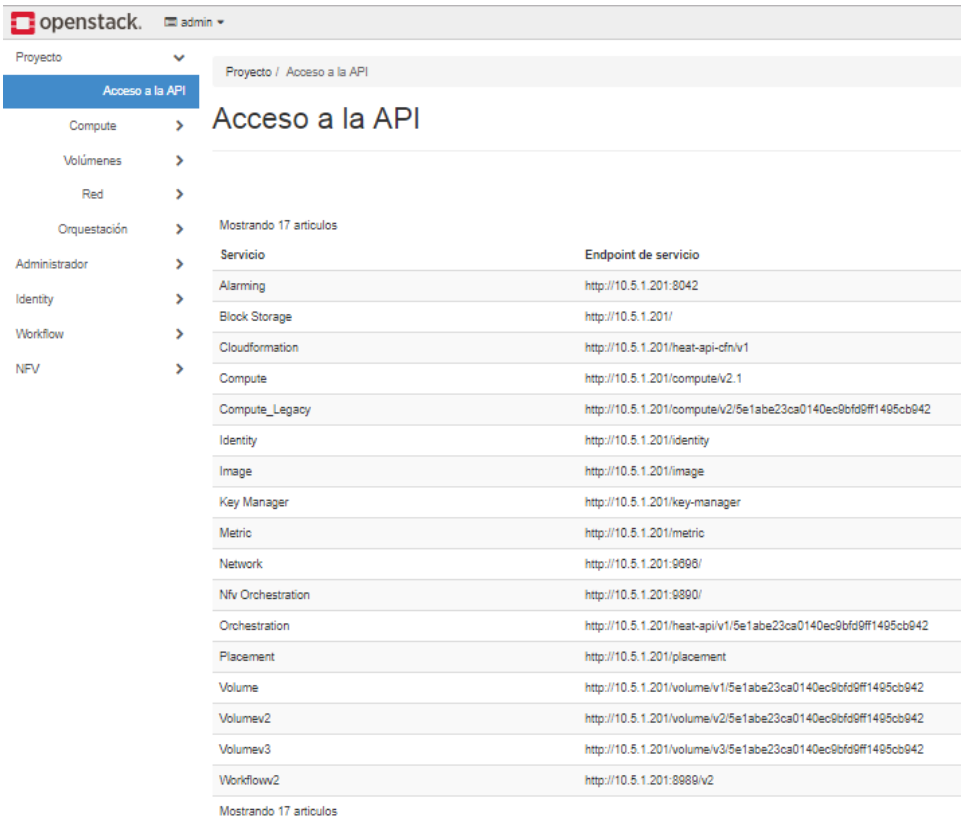


Figura 5.6: Lista de endpoints desde Horizon.

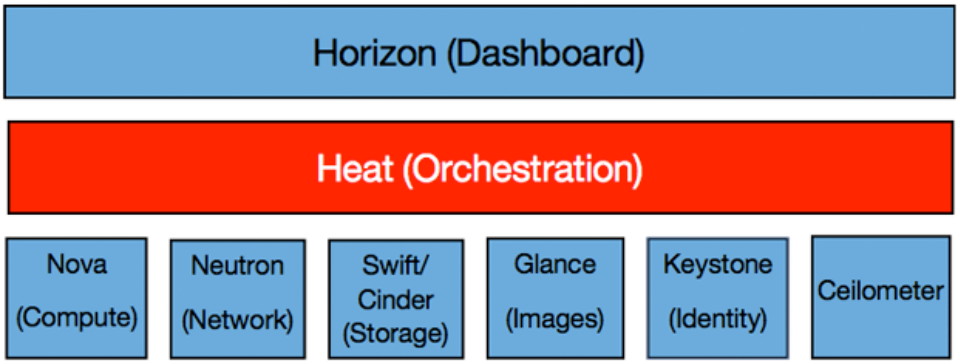


Figura 5.7: Heat: capa de orquestación.

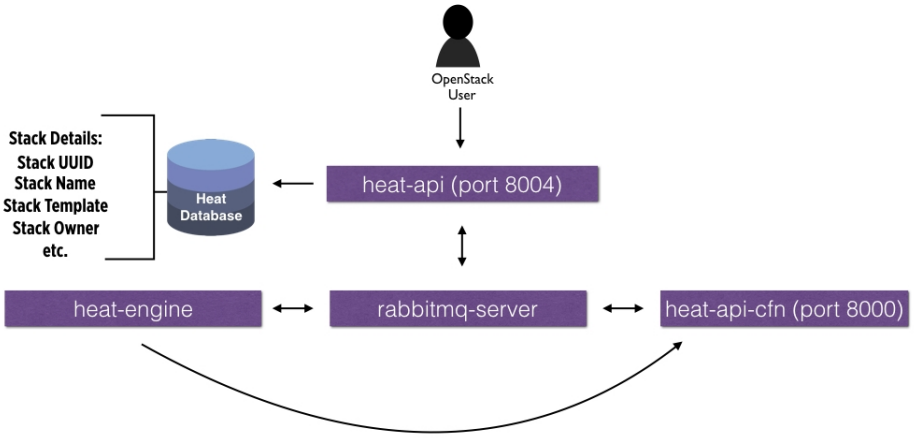


Figura 5.8: Arquitectura de Heat.

Fuente: Matt Dorn, "Preparing for the Certified OpenStack Administrator Exam"Packt Publishing, 2017

capa de abstracción entre los servicios básicos (Fig.5.7) que podemos usar para crear VNFs. Además, Heat es compatible con el formato de plantillas de AWS CloudFormation.

5.6.1. Arquitectura de Heat

Hay cuatro componentes esenciales de un proyecto en Heat que podemos ver en la Fig.5.8, cada uno con una única función [52]:

- **heat**. Es la CLI que se comunica con la API de heat, heat-api.
- **heat-api**. Es el componente que proporciona una API REST nativa de OpenStack que procesa las solicitudes y las envía al motor heat-engine.
- **heat-api-cfn**. Este componente proporciona una API de consulta del estilo AWS que es compatible con AWS CloudFormation y procesa las solicitudes y las envía a heat-engine.
- **heat-engine**. Es el cerebro de la operación y hace el trabajo principal de orquestar el lanzamiento de plantillas o *templates* y proporcionar respuesta al usuario de la API.

5.6.2. Conceptos sobre Heat

Ahora que conocemos los componentes de Heat vamos a presentar una serie de conceptos más sobre esta herramienta de orquestación:

- **Resources**. Son objetos que se crearán o modificarán durante la orquestación. Los recursos son las VNFs en sí y pueden ser redes, routers, subredes, instancias, volúmenes, direcciones IP flotantes, grupos de seguridad y más.

- **Stack**. En Heat, una pila o *stack* es una colección de recursos.

- **Parameters**. Permiten al usuario proporcionar información a la plantilla durante la implementación. Por ejemplo, si se desea ingresar el nombre de una instancia durante la orquestación, ese nombre podría ingresarse como un parámetro en la plantilla y cambiarse cada vez que se ejecute.

- **Template**. Plantilla en la que se indica como se define y describe un stack con código.

- **Output**. Como su nombre indica, nos estamos refiriendo a las salidas que proporcionan información al usuario, definidas en las plantillas y que podemos consultar.

5.6.3. Funcionamiento de Heat

Ahora que entendemos los conceptos básicos sobre Heat, podemos comprender mejor cómo funciona.

1. La orquestación se define en un *template* al describir los recursos en un formato legible y fácilmente inteligible.

2. El usuario crea el *stack* que hará que la herramienta heat-cli apunte al archivo y a los parámetros de la plantilla.

3. La herramienta heat-cli se comunica con heat-api.

4. El heat-api envía la solicitud al heat-engine.

5. El heat-engine procesa la solicitud comunicándose con las otras APIs de OpenStack y devuelve la salida al usuario.

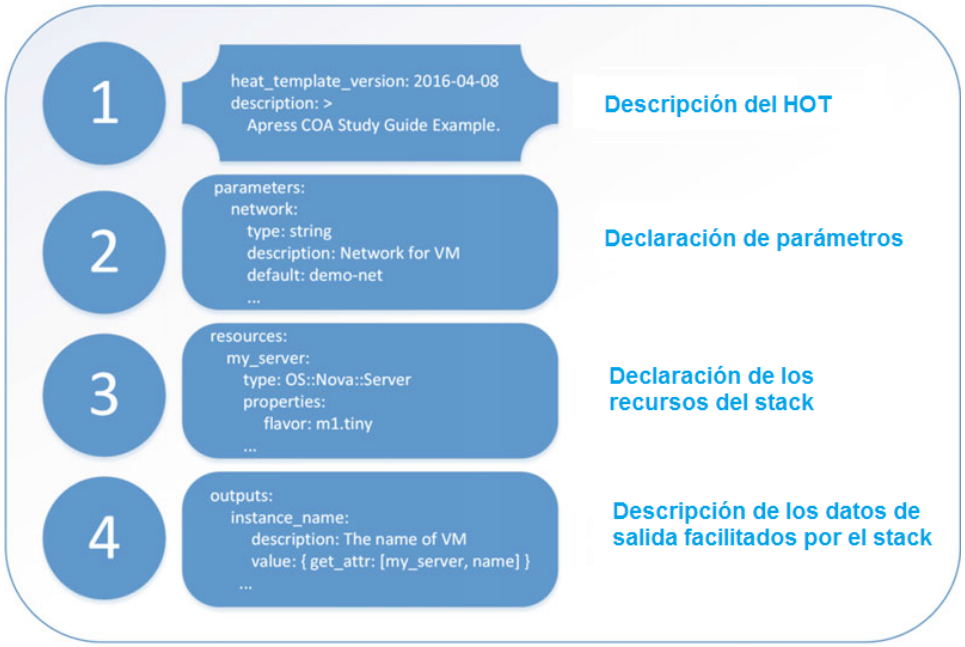


Figura 5.9: Heat Orchestration Template.

Fuente: Andrey Markelov, "Certified OpenStack Administrator Study Guide", apress, 2016

5.6.4. Heat Orchestration Template

Todos estos elementos aparecen al crear una plantilla como se aprecia en la Fig.5.9, que muestra la anatomía de un *template* en Heat y que se conoce con el nombre de HOT (*Heat Orchestration Template*).

Una plantilla de Heat suele ser un archivo YAML, un lenguaje de serialización basado en JSON cuyo fin es su fácil comprensión y que contiene los siguientes campos [53]:

- Version.
- Description.
- Parameters.
- Resources.
- Output.

Sólo son obligatorios tres de los campos antes mencionados para obtener una plantilla básica: *version*, *description* y *resources*.

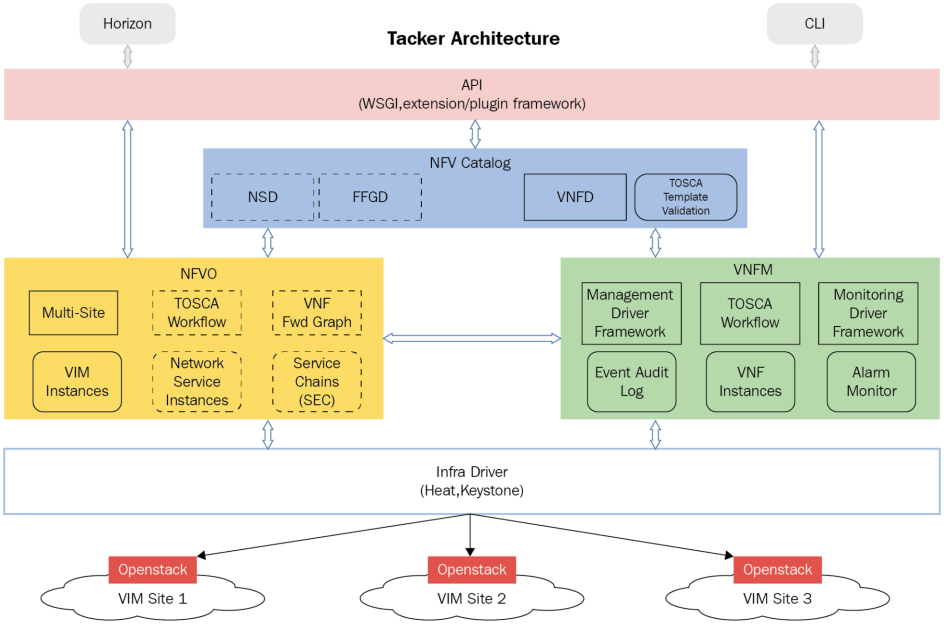


Figura 5.10: Arquitectura de Tacker.

Fuente: Openstack Wiki

5.7. Tacker

Tacker es un proyecto oficial de OpenStack que construye un gestor genérico de VNFs que se conoce como VNF Manager (VNFM) y un orquestador de NFV, NFV Orchestrator (NFVO), para implementar y operar con los servicios de red y las funciones de red virtual (VNF), en una plataforma que proporcione la infraestructura como OpenStack.

Tacker se basa en la *ETSI MANO Architectural Framework* , Fig.2.7, para formar su arquitectura, Fig.5.10, y proporciona un *stack* funcional para la orquestación de servicios de red extremo a extremo utilizando VNFs.[54]

5.7.1. Motivación de Tacker

En la mayoría de las demandas en telecomunicaciones, la alta disponibilidad y fiabilidad en su infraestructura es crítica.

Al igual que la mayoría de las plataformas cloud y aplicaciones nativas en la nube, esto se logra a través de la escalabilidad horizontal de la arquitectura a consta de dejar a un lado las prácticas tradicionales de escalabilidad vertical en las que únicamente se requiere cada vez de más hardware y mayor capacidad.

A pesar de no poder asegurar proporcionar el 99.999 % del tiempo de actividad del hardware en un entorno cloud, si podemos crear mecanismos que hagan que los servidores trabajen como un clúster con la finalidad de repartir el trabajo y asegurar alta disponibilidad.

De esto trata la escalabilidad horizontal, cuya naturaleza es distribuida. Más allá de desplegar una infraestructura celular usando distintos nodos de Nova en OpenStack, Tacker ayuda a aumentar la disponibilidad.

5.7.2. Funciones de Tacker

Dentro de los recursos y funciones de red virtual extremo a extremo que proporciona encontramos:

Catálogo NFV:

- VNF Descriptors, (VNFD).
- Network Services Descriptors, (NSD).
- VNF Forwarding GraphQQ Descriptors, (VNFFGD).

Todos estos descriptores se basan OASIS TOSCA [55] y proporcionan algunos requisitos para definir descriptores de servicio de red (NSD), descriptores VNF (VNFD), los cuales usaremos en el proyecto, y descriptores de gráficos de reenvío VNF (VNFFGD) utilizando *TOSCA templates*. Estas plantillas se analizan y se traducen en plantillas de Heat, transmitiéndose después a un controlador que interactúa con la infraestructura de OpenStack, Heat y Keystone. La relación de estos descriptores con la arquitectura de Tacker podemos verla en la Fig.5.11.

VNFM:

- Ciclo de vida básico de una VNF (crear / actualizar / eliminar).
- Posicionamiento mejorado basado en plataforma (EPA) de cargas de trabajo NFV de alto rendimiento.
- Supervisión del estado de las VNFs implementadas.
- Auto-recuperación y auto-escalado de VNFs basado en políticas.
- Facilitar la configuración inicial de cada VNF.

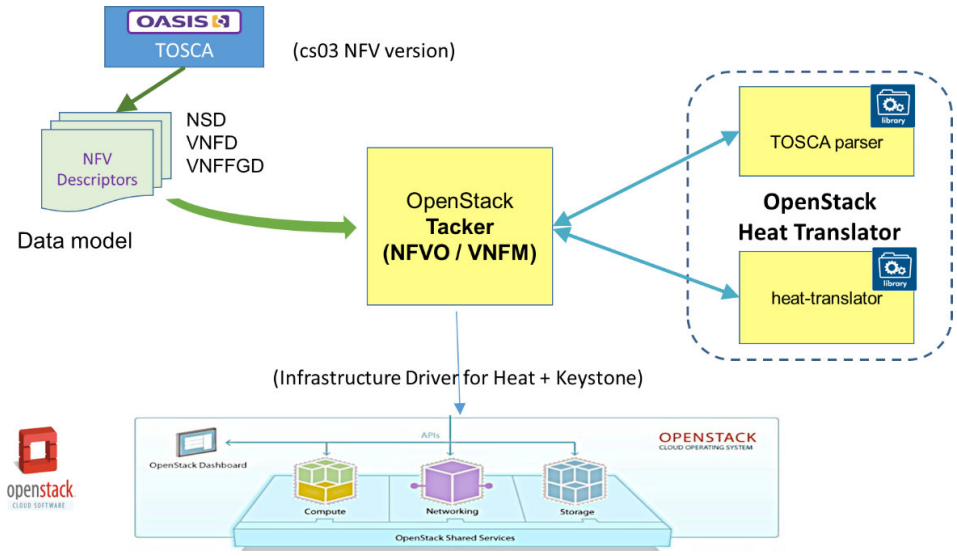


Figura 5.11: Descriptores en arquitectura de Tacker.

Fuente: Openstack Wiki

NFVO:

- Implementación de servicios de red extremo a extremo mediante el uso de plantillas que incorporan VNFs.
- Políticas de despliegue de VNF para garantizar su formación de manera eficiente.
- Conexión de VNFs mediante SFC (*Service Function Chaining*), especificadas en un descriptor VNF *Forwarding Graph Descriptor*.
- Verificaciones de recursos VIM y asignación de recursos.
- Capacidad de organizar VNFs en múltiples VIM y regiones múltiples (POPs).

5.7.3. VNFDs

Entre los recursos que proporciona Tacker, hemos hablado de los conocidos como VNFD (*Virtual Network Function Descriptor*). Estos descriptores son plantillas de TOSCA que definen lo que harán las VNFs que creemos. Como adelanto a lo que se verá en la sección correspondiente dentro del capítulo de implementación 7.6 mostramos aquí la estructura de una descriptor VNFD [56]:

```
tosca_definitions_version:
   This defines the TOSCA definition version on which the template is
       based.
   The current version being tosca_simple_profile_for_nfv_1_0_0.

tosca_default_namespace:
   This is optional. It mentions default namespace which includes
       schema,
   types version etc.

description:
   A short description about the template.

metadata:
   template_name: A name to be given to the template.

topology_template:
   Describes the topology of the VNF under node_template field.
   node_template:
       Describes node types of a VNF.
       VDU:
           Describes properties and capabilities of Virtual Deployment
           Unit.
       CP:
           Describes properties and capabilities of Connection Point.
       VL:
           Describes properties and capabilities of Virtual Link.
```

Tanto en las plantillas de TOSCA como de Heat, veremos que todas las descripciones estarán en inglés, pues no se permiten ciertos caracteres como por ejemplo las tíldes.

Como se aprecia en la descripción tenemos diversos campos:

- **tosca_definitions_version**.En esta parte se define la versión de la plantilla de TOSCA en la que nos basaremos. La versión actual es tosca_simple_profile_for_nfv_1_0_0 y será la que utilicemos.
- **tosca_default_namespace**. Este apartado es opcional. En él se mencionan los *namespaces* predeterminados.
- **description**. Descripción de lo que hace la plantilla.
- **metadata**. Aquí se puede especificar metadatos de los recursos.
- **template_name**. Nombre que queramos darle a la plantilla.
- **topology_template**. Descripción de la topología de la VNF a crear.
- **node_template**. Describe el tipo de nodo que será nuestra VNF.
- **VDU (Virtual Deployment Unit)**. Aquí se detallas las propiedades y capacidades de la implementación de la unidad virtual.

- **CP (Conecction Points)**. Se utiliza para definir las propiedades y capacidades de la conexión.
- **VL (Virtual Link)**. Describe como se realizará la conexión virtual.

Capítulo 6

Diseño

El objetivo de este apartado será explicar el diseño por el que se ha optado tanto físico como lógico y los motivos que han hecho que tomemos esas decisiones.

Se presentaran en primer lugar el diseño físico de la infraestructura y a continuación los distintos escenarios a alto nivel para y los requerimientos de los mismos para alcanzar nuestras metas.

6.1. Diseño físico del entorno

Para el dimensionado del entorno físico, se ha tenido en primer lugar en cuenta el objetivo del proyecto. Dado que la motivación del mismo es crear una infraestructura en la que podríamos implementar cualquier servicio y dotarla de mecanismos de orquestación de funciones de red que proporcionen escalabilidad y redundancia para asegurar alta disponibilidad, en principio se pensó el instalar cuatro máquinas distintas: dos en una localización geográfica y otras dos en otra diferente. En cada una de las localizaciones tendríamos un servidor dedicado con los servicios de autenticación (Keystone) y dashboard (Horizon), que compartirían los datos de todos los usuarios y proyectos disponibles, y en los otros dos el resto de servicios de OpenStack. Además cada localización tendría una IP pública de acceso distinta, creando así un entorno multi-región como el que aparece en la Fig.6.2

Puesto que sólo pudimos disponer de un equipo y una dirección IP pública, para la configuración de la infraestructura física hemos optado por realizar la configuración de la Fig.6.1, donde tenemos un servidor interconectado ala red pública y que simulara un entorno completo en un emplazamiento geográfico.

El acceso al exterior se hará pro tanto a través de una de las interfaces

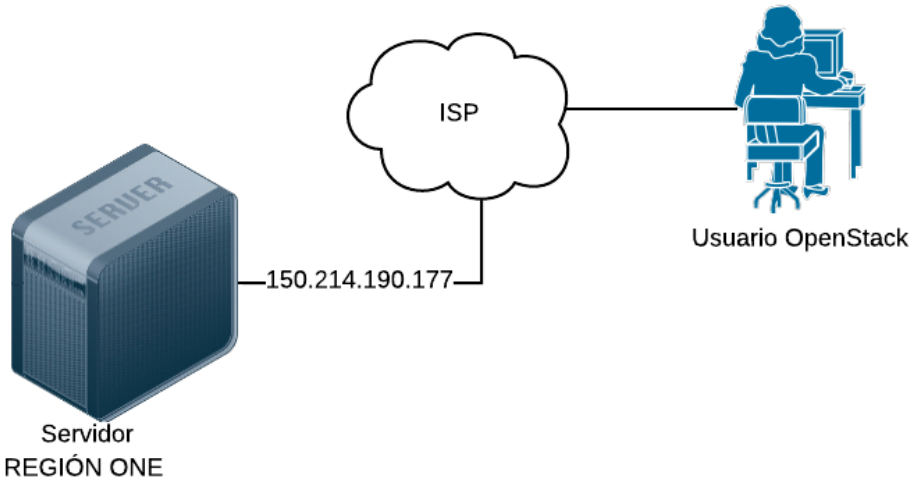


Figura 6.1: Diseño físico de la infraestructura.

de red de este servidor y que conectará con el ISP que nos da acceso a la red pública con IP 150.214.190.177. Mencionar que el servidor se instalará en el despacho 2.21 de la ETSIIT.

6.2. Diseño de los entornos para los distintos escenarios

El objetivo de estos diseños será esquematizar con los proyectos de OpenStack y los recursos hardware que tenemos los distintos escenarios que vamos a crear desde Horizon, la CLI, Heat y Tacker. En estos dos últimos se añadirán mecanismos a nuestra cloud que proporcionen redundancia de VMs, orquestación y automatización, de forma que conseguimos una administración centralizada de nuestra nube, pudiendo disponer de la recuperación de los servicios que se quisieran implementar en la nube ante posibles desastres.

Si nos fijamos en la Fig.6.1, veremos que la *Region One*, será el nombre que le daremos a la localización de nuestra IaaS y que contendrá todos los servicios que necesitamos de OpenStack en el servidor que podría tener todos los nodos AZ complementarios que se desearan. Aunque en nuestro caso sólo dispondremos de uno, sería relativamente sencillo escalar el escenario y añadir más nodos si dispusiéramos de ellos.

Debida a las limitaciones presentadas, en la sección 9.1.1, se propone como línea futura el despliegue de un entorno en producción donde se pue-

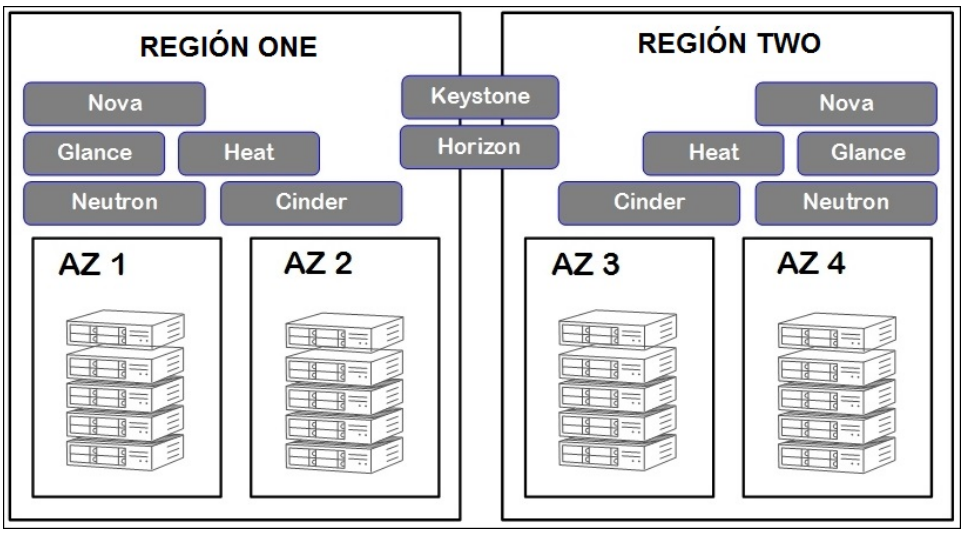


Figura 6.2: Entorno Multi-Región.

Fuente: Walter Bentley, "OpenStack Administration with Ansible 2 - Second Edition" Packt Publishing, 2016

da disponer de más nodos, en la que se puedan hospedar servicios en un nodo central, los servicios de Keystone y Horizon siendo este el nodo de administración que permitiría gestionar de manera centralizada el resto.

A continuación vamos a pasar a hacer una descripción a alto nivel del diseño físico-lógico de los distintos escenarios que se van a crear para alcanzar los objetivos propuestos

6.2.1. Diseño creado para el despliegue de un entorno mediante Horizon y CLI

En este primer escenario el propósito será crear una réplica del escenario de diseñado cuyo esquema podemos ver en la Fig.6.3. Los iconos que aparecen son propios de AWS y están elegidos por la falta de estandarización de los mismos en OpenStack y la compatibilidad que AWS tiene con OpenStack.

En el podemos ver nuestra *underlay network* que en todos los casos será la infraestructura IaaS montada con OpenStack que hará las veces de cloud privada y que será totalmente transparente para los usuario que gestionen los recursos que ofrece.

Por otro lado tenemos la *overlay network*, en la que vemos una pasarela hacia Internet, que será la que nos conecte con la interfaz física de nuestros servidor para lo que crearemos una interfaz virtual (con IP 10.5.1.201), y que

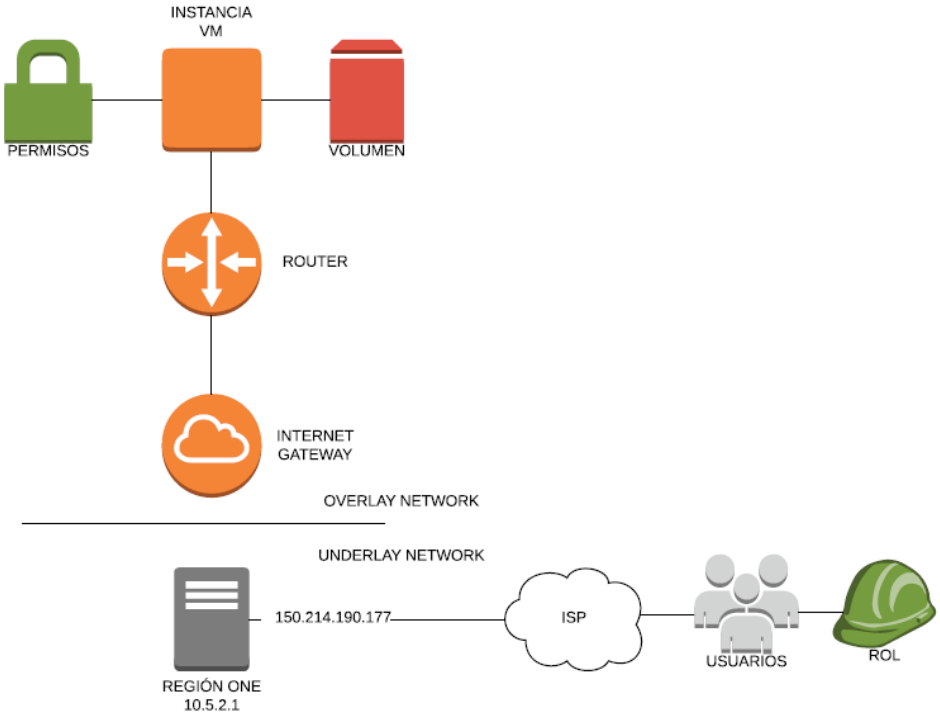


Figura 6.3: Diseño del primer escenario a crear para gestionar con Horizon y CLI.

nos permitirá conectarnos con la interfaz de red conectada con el ISP con IP 150.214.190.177 a través de la cuál los usuarios externos podrían acceder a los recursos de la nube.

Para conseguir este fin necesitamos crear una vez estando operativa nuestra nube:

- Un tenant y usuarios que puedan gestionar dicho tenant asignándoles para ello un rol específico. Una vez creado podremos continuar con los siguientes ítemes.
- Un router que separe la red pública lógica de la red privada donde ubicaremos la instancia.
- Permisos desde el punto de vista de acceso y restricción de tráfico para la instancia.
- Un volumen de almacenamiento persistente que estará asociado a la instancia
- La propia instancia que será una VM de CirrOS.

Estos serán lo hitos principales, además veremos como para alcanzarlos necesitaremos crear algunos recursos adicionales más. A la hora de realizar el diseño se han tenido en cuenta los criterios de esquemas de red jerárquico por capas conforme a lo aprendido durante el Grado en la especialidad de Telemática. En el tendremos la capa de acceso, donde implementaremos las instancia, y la capa de distribución con core colapsado que estará formada por un único router que nos dará acceso a la red externa.

El diseño del entorno de OpenStack que abarca la *overlay network* de nuestro entorno se llevará a cabo desde cero desde la interfaz web de Horizon y desde línea de comandos, para conocer así mejor la herramienta y ver los hitos de gestionar nuestros entornos de una u otra manera.

6.2.2. Diseño creado para la creación VNFs con Heat

En este segundo escenario, que en realidad será el tercero ya que en el paso anterior crearemos dos similares pero con distintas herramientas, el objetivo será diferente. Partiendo de un diseño de nuestra *overlay network* similar al de los escenarios anteriores, en este caso usaremos las *templates* de Heat para auto-escalar instancias dentro de una red ya creada como ilustramos en la Fig6.4.

Los recursos que necesitamos para el auto-escalado estarán creados de antemano, los objetivos para esta parte son dos:

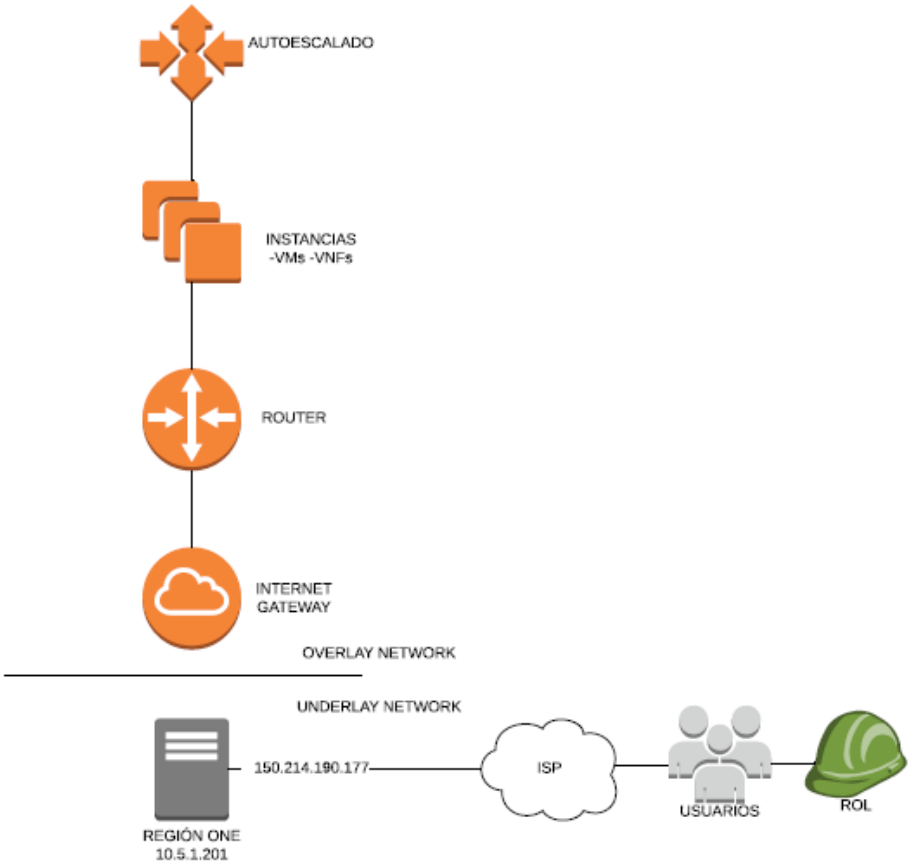


Figura 6.4: Diseño del segundo escenario a crear para gestionar con Heat.

- Auto-escalado de instancias.
- Monitorización de las instancias mediante scripting para posibilitar la recuperación de estas ante fallos.

6.2.3. Diseño creado para la creación VNFs con Tacker

En este último escenario, vamos a trabajar desde proyecto de administración y con el usuario que hará como tal. El motivo de ello es que se pretende gestionar el *site* creado que hemos denominado *Region One* desde una capa superior para ver como se podrían gestionar de forma relativamente sencilla otros *sites* que pudiésemos tener en nuestra IaaS.

En este escenario cuyo diseño podemos ver en la Fig.6.5 se crearan una serie de *TOSCA templates* para la creación de VNFDs que permitirán crear VNFs. El alcance de esta propuesta abarca:

- Registro de nuestra IaaS como una VIM.
- Creación de una VNF que permitirá crear varias VMs conectadas a distintas redes.
- Creación de una VNF que creará una VM capaz de auto-recuperarse ante fallos mediante la monitorización automatizada de la misma y en la que se insertarán datos adicionales a la VM para ilustrar la posibilidad de personalizar nuestra VM.
- Creación de una VNF basada que hará las veces de router con funcionalidades de firewall y que se podría usar para interconectar distintas VIM o *sites*.

Los recursos que necesitamos para esta parte se crearán de antemano de manera automática en la instalación de nuestra IaaS.

6.2.4. Requisitos del diseño lógico

Como cometamos en el primer escenario, para llevar a cabo estos diseños necesitaremos crear algunos recursos dependiendo del caso entre los que se encuentran:

- Creación del entorno SDN. Se crearán los routers, redes y subredes necesarios para la implementación de los escenarios tal y como se verá en el siguiente capítulo.

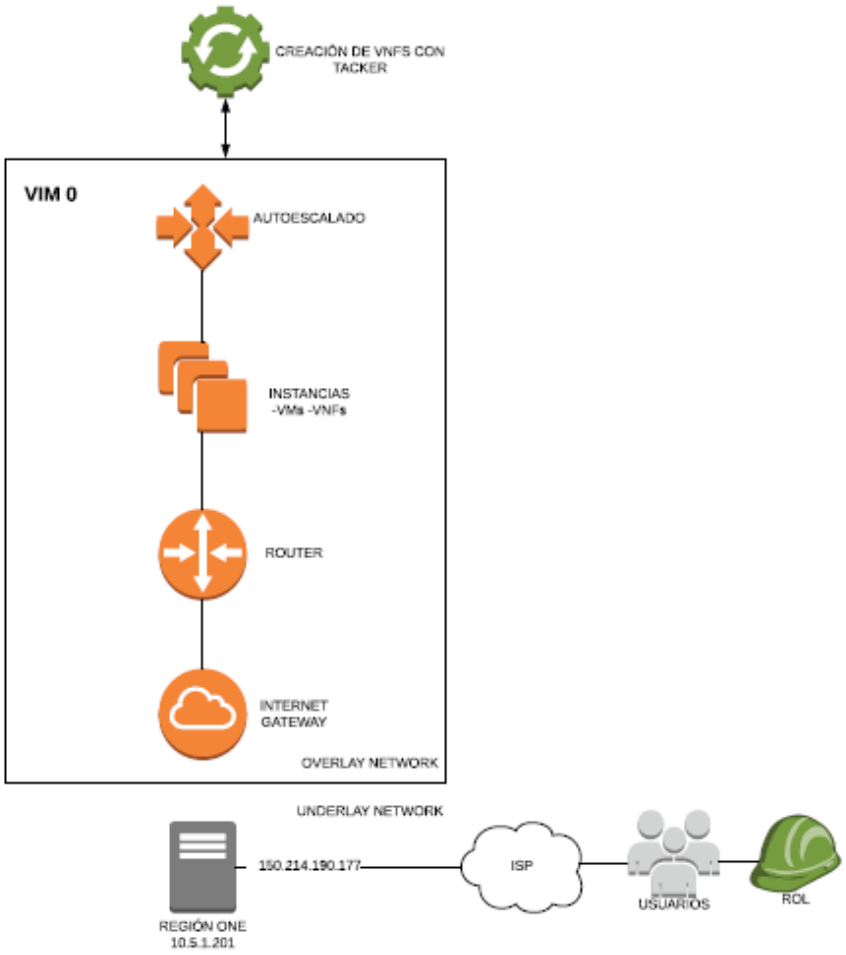


Figura 6.5: Diseño del tercer escenario a crear para gestionar con Tacker.

- Floating IP. Lo ideal para crear IPs flotantes sería disponer de varias IPs públicas. Dado que no es el caso, se ha creado en el servidor una interfaz virtual en las tarjeta de red para poder abordar el problema de disponer de una sola IP pública. El concepto de IP flotante se verá con detalle en la sección 7.3.4 del siguiente capítulo.

- Listas de acceso / Firewall. Debemos de establecer listas de acceso que actuarán de firewall para declarar el tráfico tanto entrante como saliente que deseamos permitir. En OpenStack esto se hace a través de los *security groups*. Crearemos reglas par permitir cualquier tráfico saliente y sólo el tráfico IP, SSH, y HTTP entrante.

- Instancias. Las instancias se preparan para ser capaces de alojar cualquier servicio que quisiéramos dar.

- Key Pair. También necesitaremos crear un par de claves público-privadas para acceder a las instancias de forma segura.

- Imágenes. La forma más sencilla de obtener una imagen de una máquina virtual que funcione con OpenStack es descargar una ya creada. En este caso será Cirros como veremos más adelante, debido a su pequeño tamaño.

- Almacenamiento. Por defecto, las instancias que creemos no tendrán espacio para el almacenamiento. Con el servicio de Cinder conseguiremos proveer de almacenamiento persistente a las imágenes.

- HOT. Plantilla de Heat donde definiremos los parámetros necesarios para la orquestación de VNFs.

- TOSCA Template. Plantilla que contendrá el código necesario para crear describir las VNFs que usaremos durante la implementación de esta parte.

Capítulo 7

Implementación

En este capítulo vamos a tratar la puesta en marcha de nuestra IaaS y la creación e implementación en su caso de los escenarios que crearemos en OpenStack para crear una cloud que incluirá gestión de VNFs.

Comenzaremos describiendo el despliegue del entorno en OpenStack para explicar después el proceso de instalación de forma detallada.

A continuación, se irán creando escenarios en orden de dificultad, automatización y dificultad creciente tratando todas las posibilidades que OpenStack ofrece para la creación de recursos, desde la interfaz gráfica de Horizon, pasando por la gestión desde CLI, *scripting*, orquestación de VNFs con Heat y *scripting* y finalmente gestión de VNFs con Tacker.

Nos basaremos en la versión *Queens* de OpenStack que es la versión estable más actualizada hasta la fecha [23].

A lo largo de este capítulo introduciremos en detalle aquellos conceptos que aún no se han presentado y que aportan valor al proyecto y trataremos de forma más práctica haciendo menos énfasis a nivel teórico, los puntos que resulten más atractivos de presentar de este modo.

7.1. Desplegando OpenStack

En este punto vamos a ver las opciones que tenemos para implementar nuestra nube IaaS con OpenStack y como lo haremos finalmente con el uso de una solución todo en uno (*all-in-one*). Eso significa que los diferentes servicios de OpenStack se ejecutarán en la misma máquina, que puede ser una máquina física o una máquina virtual y servirá de guía para implementar OpenStack en Ubuntu.

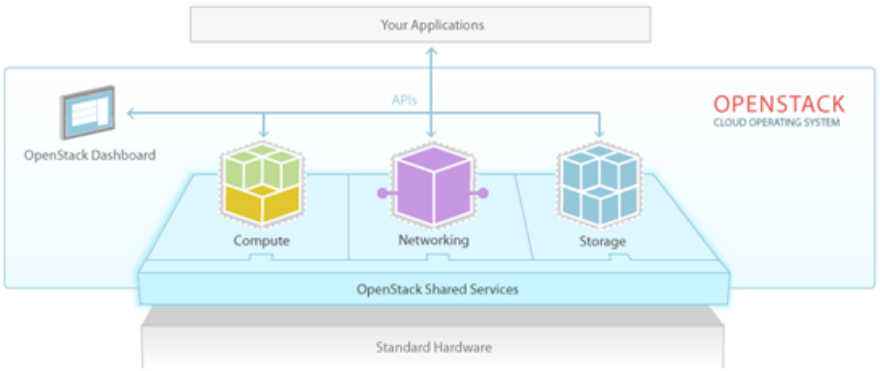


Figura 7.1: Tipos de nodos en OpenStack.

Fuente: OpenStack wiki

7.1.1. Métodos de despliegue

OpenStack se puede implementar de varias maneras:

- Manualmente. OpenStack se puede implementar manualmente mediante la aplicación de los pasos descritos en la documentación pertinente [57].
- Scripted. También se puede implementar de forma guiada mediante el uso de una solución como Packstack o DevStack, que será la que nosotros usemos.
- Despliegue a gran escala. Otra opción es el despliegue de OpenStack a gran escala, utilizando soluciones de implementación más avanzadas, como Triple O, Director [58][59].

7.1.2. El rol de los tipos de nodo

Normalmente, existen diferentes roles según el tipo de nodo en una nube de OpenStack como vemos en la Fig.7.1 Los roles que se utilizan dependen de la configuración existente:

- Nodo controlador (*Controller node*). Ejecuta los servicios de control. En este nodo encontraremos servicios como Keystone, la cola de mensajes, la base de datos MariaDB y todo lo esencial para controlar la nube.
- Nodo controlador de red (*Network controller node*). Proporciona servicios de red a la nube. Es el tipo de nodo que proporciona SDN y está

conectado a redes internas y externas.

- Nodo de cómputo (*Compute nodes*). Son los nodos que realmente van a ejecutar las instancias en OpenStack. Básicamente, los *compute nodes* se conectan con los hipervisores y en ellos algunas partes de Nova se están ejecutando para interactuar con el hipervisor, lo que permite a OpenStack programar el funcionamiento de máquinas virtuales en un nodo de cómputo específico.

- Nodos de almacenamiento (*Storage nodes*). Los nodos de almacenamiento son todo lo que involucra el almacenamiento de objetos (servicios como Swift o Ceph) y en un clúster de almacenamiento avanzado, estamos hablando fácilmente de docenas de *storage nodes* para garantizar la disponibilidad del mismo.

Aunque hagamos referencia a un nodo, en un entorno en producción redundante podrá haber tantos como sean necesarios de cada tipo para asegurar alta disponibilidad.

7.1.3. Despliegues con DevStack

DevStack es la solución estándar para la creación de un entorno de pruebas con OpenStack y se ejecuta en múltiples distribuciones de Linux incluyendo Ubuntu, Fedora, OpenSUSE, Debian, CentOS o RedHat entre otras. DevStack proporciona una serie de *scripts* para montar un entorno OpenStack completo basado en la configuración que especificaremos en un archivo destinado a ese fin.

DevStack es muy popular entre los desarrolladores, ya que se usa para pruebas de desarrollo y operativas. Es importante señalar que DevStack no se creó para su uso en producción.

Se pueden usar diferentes escenarios al configurar un entorno de laboratorio basado en Ubuntu con DevStack:

- All-in-one Single VM.

- All-in-one Single Machine.

- Multi-node Lab.

La dos primeras son la instalación todo en uno con la salvedad de realizarla en una máquina virtual o una máquina física. La última, puede usar varias VMs o máquinas físicas.

Optaremos por la segunda opción para realizar la implementación de este proyecto por motivos obvios de disponibilidad de equipos y presupuesto, es decir, una configuración all-in-one en una máquina física, lo cual significa que no haremos una distinción estricta entre los tipos de nodos vistos sino que tendremos una máquina en el entorno creado que proporcione las funciones de todos estos nodos.

7.2. Proceso de instalación

Usaremos Ubuntu Server 16.04.3 LTS como sistema operativo. Para ello, el departamento de Telemática cedió un servidor dedicado ya que en el proceso de instalación se hacen cambios sustanciales en la configuración del equipo y además se requiere de altos requisitos hardware para que el funcionamiento del despliegue sea óptimo.

Vamos a comenzar con la implementación de OpenStack mediante DevStack.

7.2.1. Puesta a punto del servidor

En primer lugar, una vez instalado el sistema operativo en el servidor, haremos una serie de verificaciones.

Para empezar, necesitamos verificar la memoria RAM disponible, ya que si no tenemos 8 GB como mínimo, la instalación fallará. Inicialmente teníamos 16 GB y tras la instalación:

```
jorge@wimunet4::~$ free
              total        used        free      shared  buff/cache   available
Mem:       16298968     5222208     8664664        9464     2412096    10721112
Swap:       2097148           0     2097148
```

Puesto que el servidor está dentro de la ETSIIT, tras solicitar una IP pública se nos asignó la 150.214.190.177. Una vez obtenida configuramos el servidor modificando el archivo /etc/network/interfaces de nuestra máquina como sigue:

```
jorge@wimunet4::~$ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto enp3s0
iface enp3s0 inet static
        address 150.214.190.177
        netmask 255.255.254.0
        gateway 150.214.190.222
```

```
        dns-nameservers 150.214.204.10

auto enp3s0:1
iface enp3s0:1 inet static
        address 10.5.1.201
        netmask 255.255.255.0
```

De este modo conseguimos:

- Acceso a Internet con la IP pública 150.214.190.177.
- Creación de la interfaz virtual *enp3s0:1* que nos permitirá asignar a las instancias que creemos IPs flotantes (ver sección 7.3.4) dentro de la red 10.5.1.201/24 para su acceso desde el exterior resolviendo así la problemática de tener una única IP pública.

Una vez preparado el servidor comprobaremos que tenemos conexión con la red pública:

```
jorge@wimunet4:~$ ping www.google.es
PING www.google.es (216.58.201.163) 56(84) bytes of data.
64 bytes from mad08s06-in-f3.1e100.net (216.58.201.163): icmp_seq=1
    ttl=54 time=12.2 ms
64 bytes from mad08s06-in-f3.1e100.net (216.58.201.163): icmp_seq=2
    ttl=54 time=12.2 ms
^C
--- www.google.es ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 12.239/12.251/12.264/0.111 ms
```

7.2.2. Instalando OpenStack

En este punto nos encontramos ya en disposición de instalar OpenStack.

En primer lugar vamos a ver la creación de un usuario con el nombre *stack* y permisos de administrador:

```
jorge@wimunet4:~$  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, instalamos los paquetes *cloud-init* que contienen un conjunto de paquetes que son útiles cuando se trabaja con DevStack en una máquina Ubuntu. Básicamente son scripts de Python, Git y algunos más.

```
stack@wimunet4:~$ sudo apt-get install cloud-init
```

En esta parte de la configuración, pasamos a clonar todos los paquetes actuales de OpenStack que están disponibles en Git [27] a la máquina local,

con lo que conseguiremos poder comenzar con la instalación en la misma. Además indicamos que la versión que queremos instalar sea Queens con la opción -b que nos permite seleccionar el *branch*.

```
stack@wimunet4:~$ git clone https://git.openstack.org/openstack-dev/
    devstack -b stable/queens
```

Tras esto, se habrá creado un directorio llamado *devstack*:

```
stack@wimunet4:~$ cd devstack
```

Dentro de él existen distintos archivos, los que más nos interesarán son *stack.sh, unstack.sh y clean.sh*, que se utilizan para instalar, desinstalar y limpiar la configuración existente respectivamente. Necesitamos crear un archivo *local.conf*, al que antes hacíamos referencia y que contendrá la configuración necesaria para que la instalación se realice acorde a nuestra necesidades. El archivo de configuración lo podemos ver en el apéndice B.

Una vez creados ambos archivos podemos ya ejecutar el script *stack.sh*, que leerá los parámetros de configuración de *local.conf* y construirá nuestra IaaS con DevStack-OpenStack basándose en estos parámetros.

```
1 stack@wimunet4:~/devstack$ ./stack.sh
```

El tiempo promedio de ejecución del script es largo y dependerá de la capacidad del hardware de nuestro servidor. Cuando finaliza la instalación de manera correcta, nos mostrará un reporte de la misma, Fig.7.2. En el podemos ver el tiempo que ha demorado cada componente, nuestra dirección IPv4 e IPv6 y como se ha creado un usuario *demo* que será otro usuario del tenant *admin* creado además del usuario administrador con el mismo nombre. Por último, podemos ver que la versión instalada es *Queens*, como ya hemos comentado, la versión estable más reciente que incorpora versiones actualizadas de los proyectos de OpenStack.[60]

Después de implementar OpenStack en Ubuntu, abrimos un navegador en la dirección IP de la máquina donde acabamos de alojar OpenStack, cuya IP pública es 150.214.190.177, en caso de que estemos accediendo a Horizon desde una red externa, mientras que desde el servidor podremos acceder directamente a la IP privada, 10.5.1.201 y conectarnos. Se mostrará entonces la página de la Fig.7.3 donde podremos iniciar sesión como *admin* utilizando la contraseña establecida.

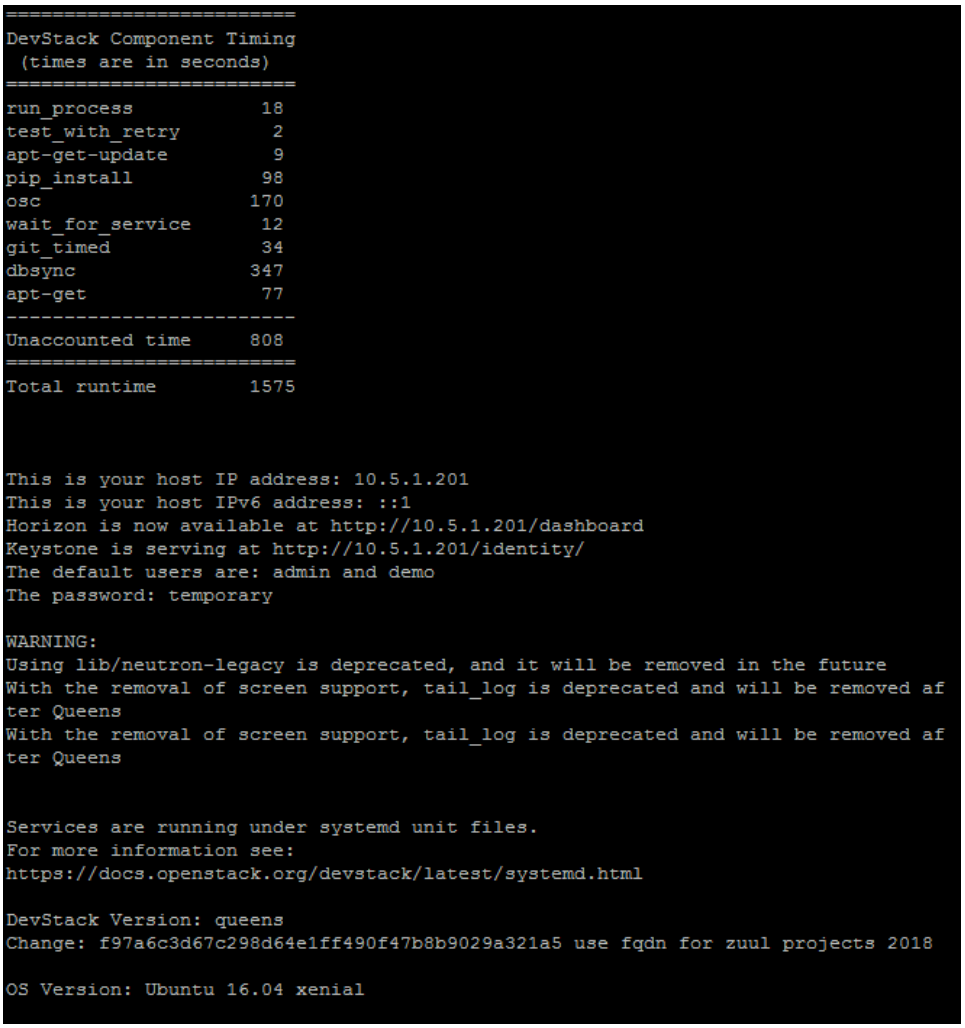


Figura 7.2: Reporte de la instalación.

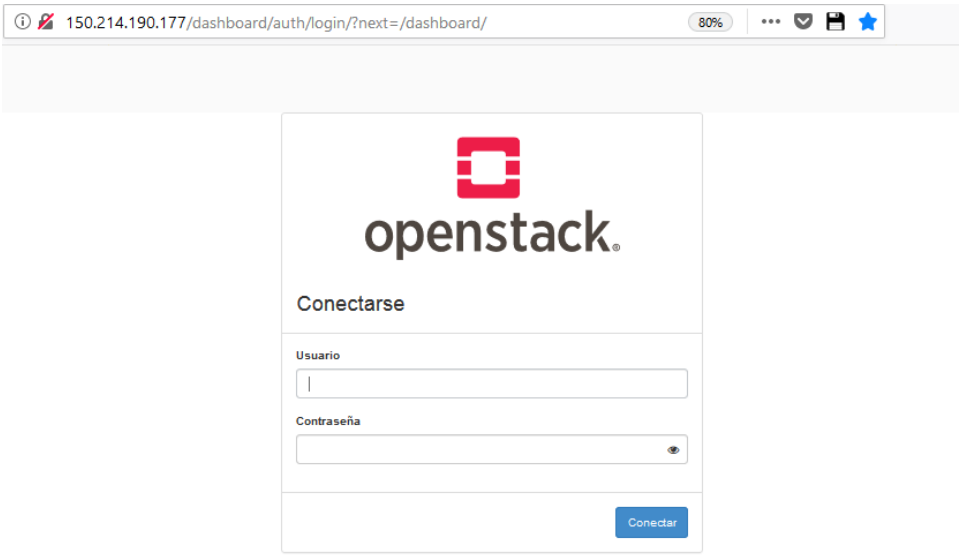


Figura 7.3: Página de acceso al dashboard de Horizon.

7.3. Configuración de un escenario mediante Horizon

Horizon es la herramienta más importante para los administradores que se están iniciando en OpenStack pues proporciona una interfaz web que facilita el trabajo como administrador de OpenStack.

En este apartado procederemos con la implementación de instancias desde la interfaz web de Horizon. Trataremos el propósito y uso de tenants, usuarios y roles, diferenciaremos entre ámbitos administrativos en Horizon y discutiremos los componentes necesarios para implementar instancias. Veremos lo fácil que resulta desplegar instancias y también sus implicaciones, comenzando con la creación de un proyecto o tenant y acabando con una cloud operativa.

Antes de acceder a la interfaz web, vamos a ver el estado del servidor web Apache. Los repositorios de Ubuntu, incluyen el servidor de código abierto Apache, el más utilizado en sistemas operativos Linux. Apache sirve las páginas web a los navegadores de los usuarios al acceder a la dirección IP o al nombre de dominio asociado a nuestro sistema Ubuntu. Además Apache se ejecuta en segundo plano como un servicio del sistema, lo que significa que podemos iniciarlo, detenerlo y reiniciarlo usando el servicio nativo de comandos en Ubuntu. Para iniciar y comprobar su estado podemos escribir en un terminal:

Figura 7.4: Panel inicial de Proyectos tras acceder a Horizon.

```
stack@wimunet4:~/devstack$ sudo service apache2 start
stack@wimunet4:~/devstack$ sudo service apache2 status
```

Nuestro servidor web Apache aloja Horizon, es decir, los procesos del *dashboard*. El servidor web está activo y ejecutándose, lo que significa que podemos ir a la interfaz web y ver qué aspecto tiene Horizon.

Abriendo un navegador web, Firefox en mi caso, podremos conectarnos a la dirección IP del servidor web Horizon usando la IP pública 150.214.190.177, como ya vimos en la última parte del apartado anterior.

Firefox resuelve automáticamente el nombre DNS de Horizon. En el *prompt* de inicio de sesión, podemos iniciar sesión como usuario administrador. Como todavía no hemos creado nada, solo hay un usuario disponible, el usuario administrador con las contraseñas que proporcionamos. En realidad también hay instalado un cliente de demostración, demo, con el que podremos iniciar igualmente sesión como un usuario cliente pero que no usaremos.

Tras introducir las credenciales para el usuario *admin* accederemos al escritorio donde se nos mostrará el panel de la Fig.7.4 en el que podemos ver un listado de los proyectos que existen en nuestro entorno.

De modo que en Horizon existen diferentes puntos de vista. Si somos administradores de OpenStack, veremos diferentes elementos que si nos autenticamos como un usuario tenant. El usuario tenant solo ve el entorno para ese tenant específico, mientras que el usuario administrador, puede ver todos los entornos.

Una de las partes más importante para el usuario administrador es la pestaña de identidad, "Identity". Una de las primeras cosas que haremos como administrador será crear un entorno donde los tenants puedan iniciar sesión, como veremos después.

Además, está la pestaña de administración "Administrador", en la que el administrador puede gestionar la infraestructura de la nube, infraestructura en la que están disponibles los hipervisores o *Host Agrgregates*, que es un grupo de hipervisores, las imágenes de Glance para arrancar instancias, o bien gestionar las redes públicas. Estas son típicamente las tareas administrativas.

También tenemos una pestaña de proyectos, "Proyecto", que no es algo en lo que se trabajará como administrador, porque normalmente se separará entre el entorno del *tenant*, que contiene el entorno en el que van a derivar las máquinas virtuales y el entorno de administración, que realmente está destinado a proporcionar infraestructura.

Antes de comenzar a trabajar en Horizon, vamos a familiarizarnos con

algunos conceptos esenciales de OpenStack:

- **Service**. Un servicio es un proyecto. Es un componente esencial de OpenStack, tal como lo define *OpenStack Foundation*.

- **Tenant**. Un *tenant* es un espacio de la infraestructura reservado a un cliente. Si OpenStack se usa como una nube pública, se pueden crear varios tenants. También se conoce como proyecto.

- **Role**. Con un rol, nos referimos a un conjunto de permisos que se usan para permitir a los usuarios hacer lo que necesiten.

- **User**. Un usuario es una entidad administrativa que ha sido asignada a un rol específico en OpenStack.

7.3.1. Creación de proyectos o tenants

A continuación vamos a ver cómo crear un tenant desde Horizon. Normalmente, la primera tarea para el administrador de OpenStack es crear los entornos de tenants, también conocidos como proyectos, lo cual haremos desde la pestaña "Identity".

Haciendo clic en la pestaña "Identity" podemos ver que hay cuatro elementos diferentes: los proyectos, los usuarios, los grupos y los roles, como se aprecia en la Fig.7.4.

Podemos ver que los proyectos activos entre los más relevantes se encuentra el proyecto de administración, "admin", el proyecto de servicios, "service" y el proyecto para la gestión de NFVs, "nfv".

Los proyectos que no se vayan a usar como puede ser el caso de "demo" podemos eliminarlos marcando el proyecto y clicando en el botón rojo situado en la parte superior derecha "Eliminar Proyectos".

El proyecto de administración es donde se encuentra el administrador y el proyecto de servicios es un tenant de los servicios de OpenStack. Esto quiere decir que todos los servicios están creados en un proyecto también, lo cual está relacionado con los permisos, al ponerlos todos en un proyecto específico, es fácil darles acceso a estos servicios a los distintos tenants.

Vamos a crear ahora nuestro propio proyecto haciendo clic sobre icono "Crear Proyecto". De este modo crearemos el entorno donde los usuarios que especifiquemos de la nube puedan crear sus instancias o los recursos que necesiten.

Vamos a llamarlo "horizon-project", Fig.7.5. La descripción no es necesaria. En las pestañas "Miembros del proyecto" y "Grupos del proyecto"

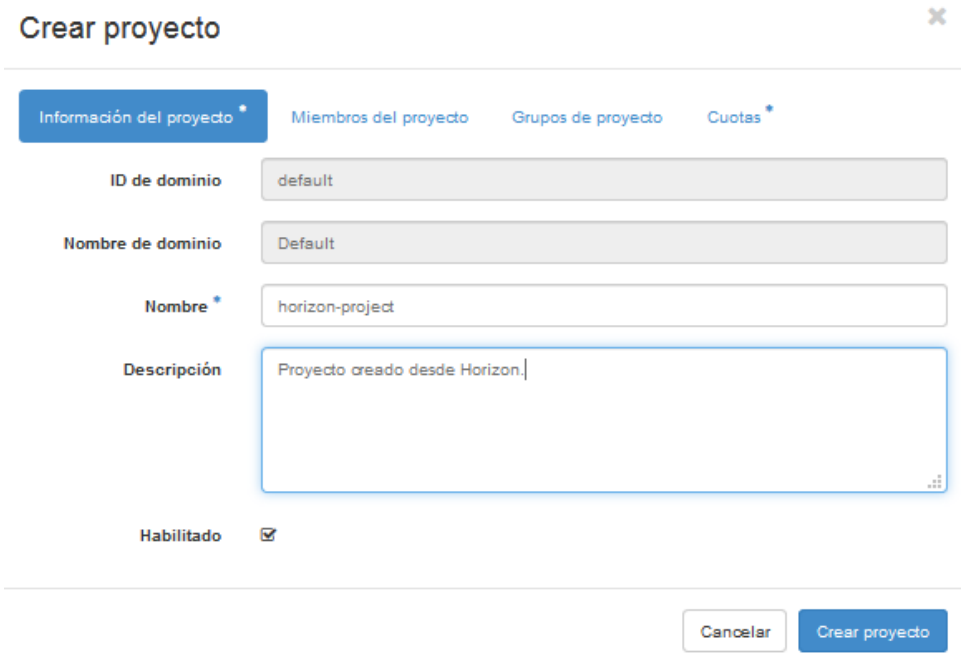


Figura 7.5: Creación de un proyecto en Horizon.

podemos especificar los miembros y grupos del proyecto que deseemos. En este caso dejaremos los que vienen por defecto para después ver como se añaden los que creemos nosotros y no los que se instalaron como demo. Podemos pasar directamente al apartado de "Cuotas".

La cuota del proyecto es importante ya que se utiliza para determinar que los usuarios de su proyecto no pueden obtener acceso ilimitado a los recursos en la nube. Por lo tanto, podemos limitar la cantidad de instancias, la cantidad de RAM que se puede usar, estableciéndola en 4096 MB por ejemplo, porque tenemos alrededor de 9 GB de RAM solamente tras la instalación de DevStack, la cantidad de direcciones IP flotantes, la cantidad de redes, etc.

Después de imponer las limitaciones que vemos en la Fig.7.6, habremos especificado las propiedades de cuota del proyecto. Haciendo clic en "Crear proyecto" crearemos el tenant que se agregará a los que ya existían.

7.3.2. Creación de usuarios

El siguiente paso será crear un usuario. Dentro de "Identity" clicamos en "Usuarios" y "Crear usuario", Fig.7.7.

Vamos a darle un nombre, "horizon-user". No necesitamos una descrip-

Figura 7.6: Cuota del proyecto.

ción ni un correo electrónico pero sí una contraseña. Los usuarios también deben asignarse a un proyecto, de modo que en la lista "Proyecto principal", podemos seleccionar el proyecto que acabamos de crear. También podemos definir en el desplegable "Rol", el tipo de usuario o rol que tendrá. Especificamos que sea *Member*. Un usuario Member puede crear todo lo necesario en el entorno de la nube. Después de especificar las propiedades del usuario, podemos hacer clic en "Crear usuario" y con esto, dicho usuario podrá ya iniciar sesión.

Las pestañas "Grupos" y "Roles" de "Identity" se usan con menor frecuencia. Sus funciones son las de agrupar diferentes usuarios y definir nuevos roles respectivamente, pero normalmente para la mayoría de los entornos en la nube, basta con los roles de miembro y administrador.

Si cerramos sesión e iniciamos de nuevo pero esta vez con el usuario "horizon-user" creado, entraremos directamente a la vista general del usuario que vemos en la Fig.7.8, donde el proyecto se abre de manera predeterminada y en el que podremos ver un reporte del uso y las capacidades de nuestro entorno cloud en cuanto a instancias, RAM y el resto de recursos principales.

7.3.3. Recursos necesarios para crear una instancia

En este apartado, vamos a explicar qué se necesita para implementar una instancia en un entorno de nube OpenStack.

Partimos de nuestro *Compute Node*. Dicho nodo interactúa con el hipervisor KVM o cualquier hipervisor que se necesite y se derivará una instancia a partir de él, pero las diferentes partes de la instancia provienen de otro lugar. Para ejecutar una instancia se necesitas redes, necesitamos una imagen, ajustes de seguridad y otros. Todos estos requisitos son provistos por diferentes servicios de OpenStack.

La instancia en sí proviene del servicio Nova, que se ocupa de crear las instancias, así como de las capas del hipervisor y de proporcionar seguridad. Luego está la creación de redes. Para encargarnos de las redes, necesitamos configurar Neutron. En Neutron, necesitamos una red privada, que se reservará para un inquilino específico que ejecutará la instancia. Y luego está la imagen, que es proporcionada por el servicio de imágenes Glance.

Y aún hay más. Al proporcionar una instancia, está se ejecuta como una instancia de solo lectura, lo que significa que el almacenamiento será efímero y no se podrá almacenar en ningún lugar, sería similar a arrancar un sistema operativo desde un *live* CD. Si queremos que una instancia pueda almacenar información en algún lugar, tendremos que gestionar también el almacenamiento de la misma que es proporcionado por el servicio de Cinder.

Así pues si queremos crear una instancia, no es tan simple como arran-

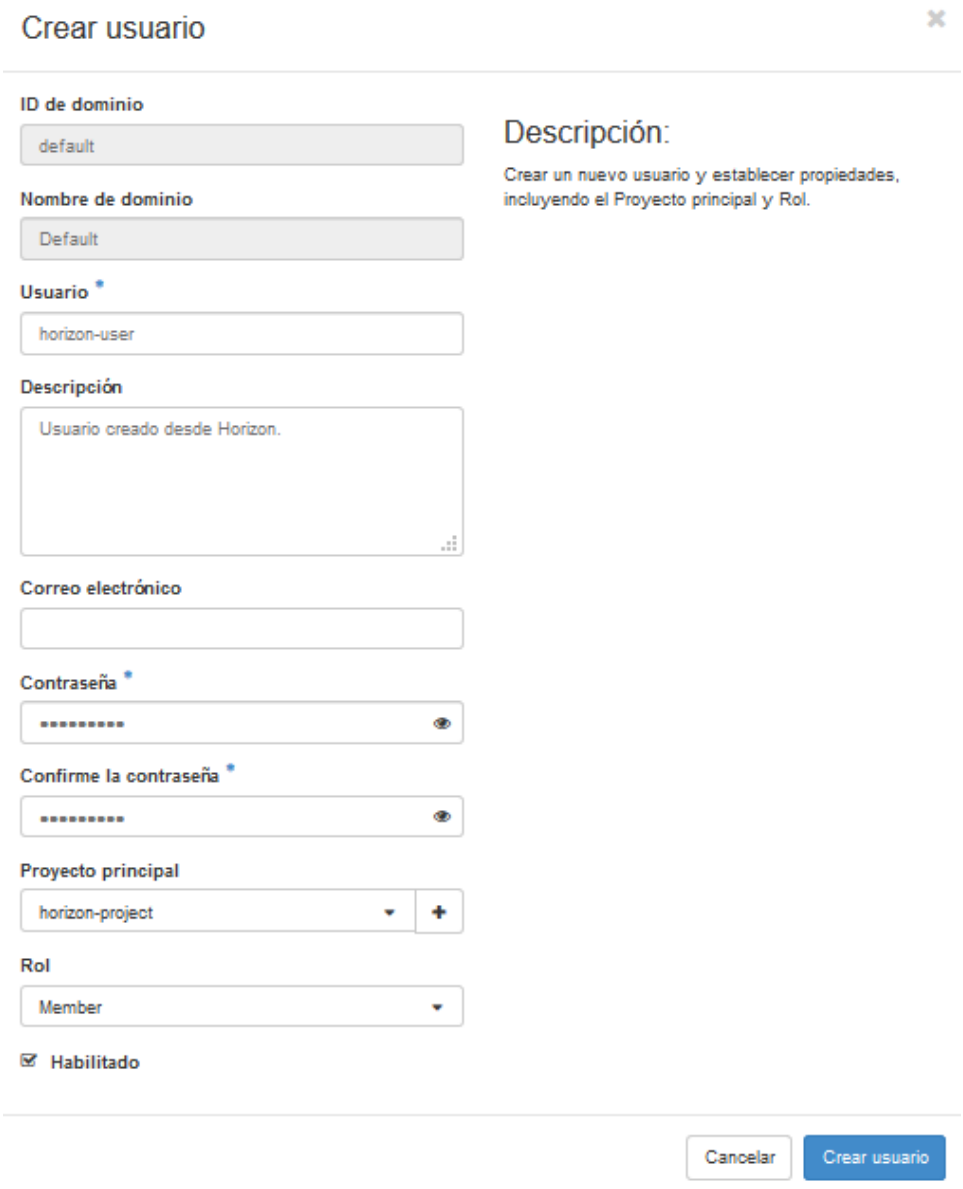


Figura 7.7: Creación de un usuario.

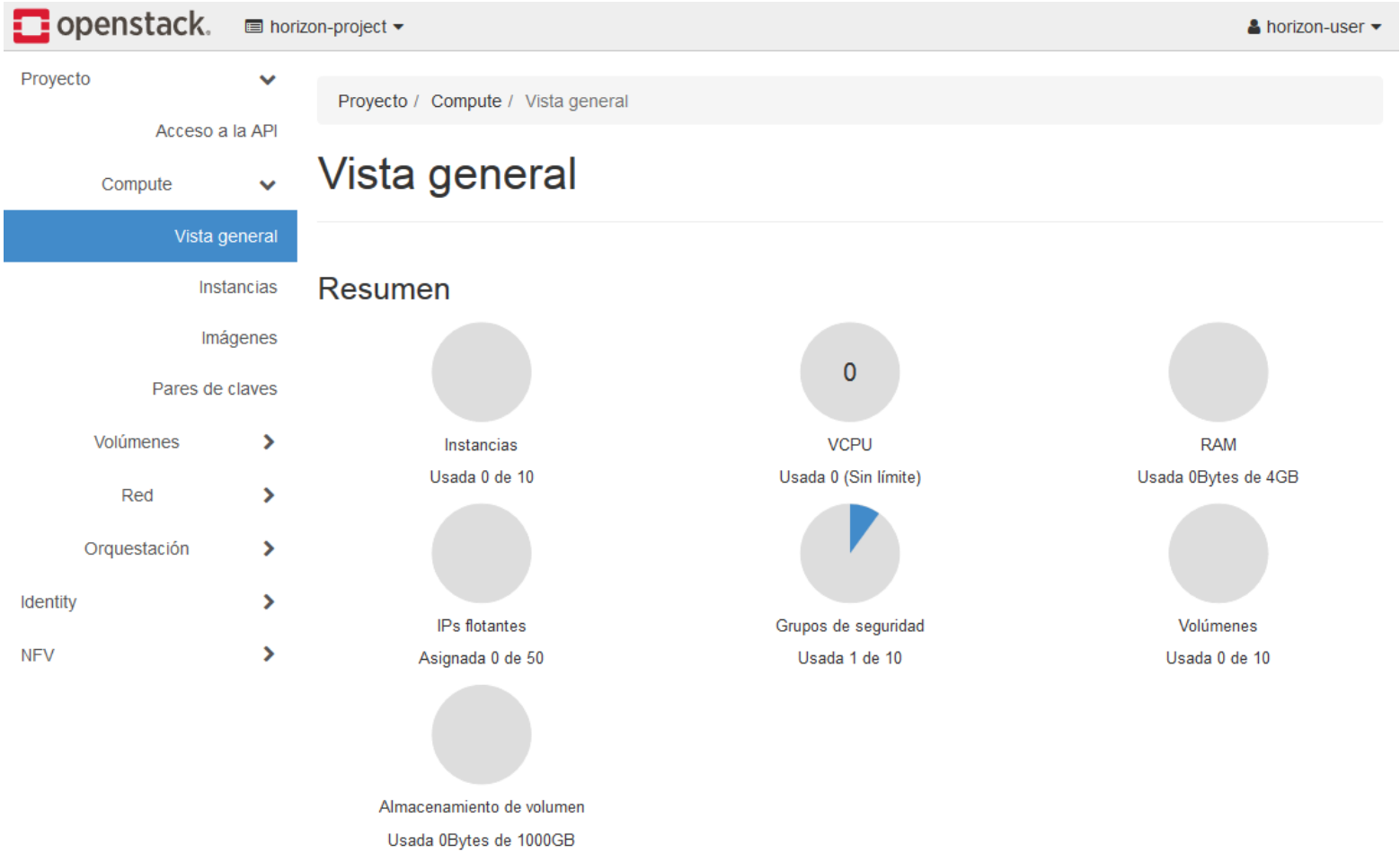


Figura 7.8: Vista general del proyecto horizon-project.

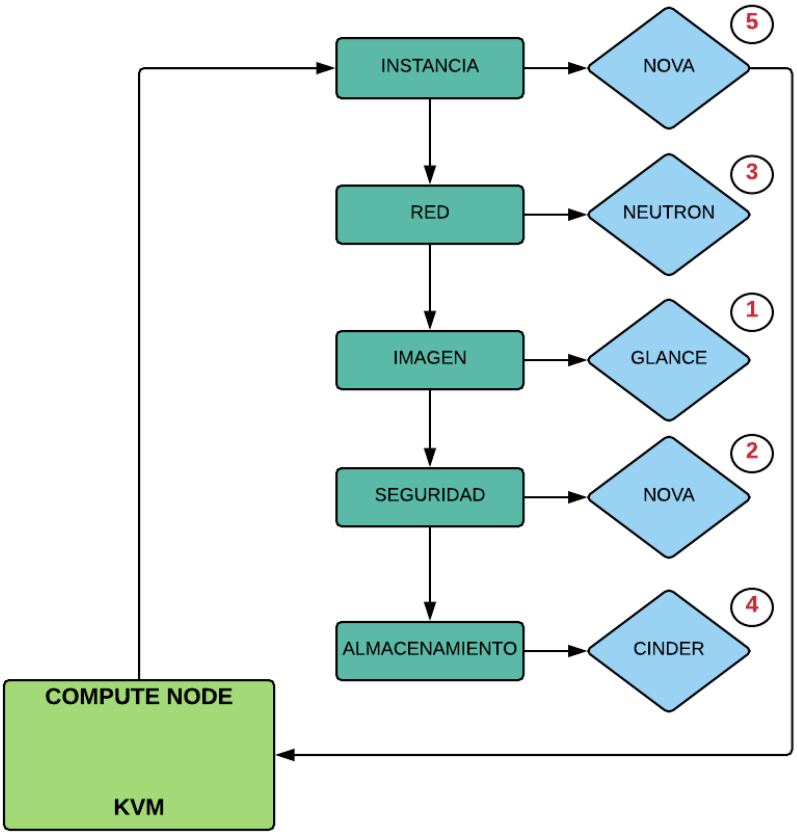


Figura 7.9: Recursos necesarios para la creación de una Instancia.

carla sin más, necesitamos preparar otras elementos antes. En primer lugar la imagen en Glance que vamos a arrancar, un símil podría ser el ya nombrado de un *live* CD de Linux. Después, debemos asegurarnos de que en Nova tenemos configurada y activa la seguridad pues de lo contrario no podremos acceder a la imagen. También tenemos que atender al *networking* y también al almacenamiento persistente si así lo queremos. Una vez realizadas estas tareas, podremos *bootear* desde Nova la instancia. Un esquema de los pasos podemos verlo en la Fig.7.9.

De modo que los pasos a seguir son:

- Configurar la red.
- Asignar direcciones IP flotantes.
- Definir un grupo de seguridad en la nube.
- Crear un par de claves SSH.
- Crear una imagen de Glance.
- Eligir un *flavor*, el cual contendrá las capacidades hardware reservadas para la instancia.
- Iniciar la instancia.

7.3.4. IP flotante

Las redes son una parte muy importante del entorno cloud. Antes incluso de comenzar a crear instancias en nuestra nube, como administradores de un tenant tendremos que crear el entorno de SDN. Esto implica que hemos de asegurarnos de que las instancias se puedan conectar a una red interna privada y también asignar direcciones IP flotantes para que puedan ser alcanzadas externamente.

En primer lugar vamos a definir el concepto de IP flotante en OpenStack. Una dirección IP flotante es un servicio proporcionado por Neutron. No se utiliza para la asignación ningún servicio DHCP ni está configurado de manera estática dentro de la instancia o la máquina donde se ejecute. De hecho, el sistema operativo anfitrión no tiene idea de que se le asignó una dirección IP flotante. La entrega de paquetes a la interfaz con la dirección flotante asignada es responsabilidad del agente L3 de Neutron.

A las instancias con una dirección IP flotante asignada se puede acceder desde la red pública mediante la IP flotante. Una dirección IP flotante y una dirección IP privada se pueden usar al mismo tiempo en una única interfaz de red. Es probable que la dirección IP privada se use para la comunicación

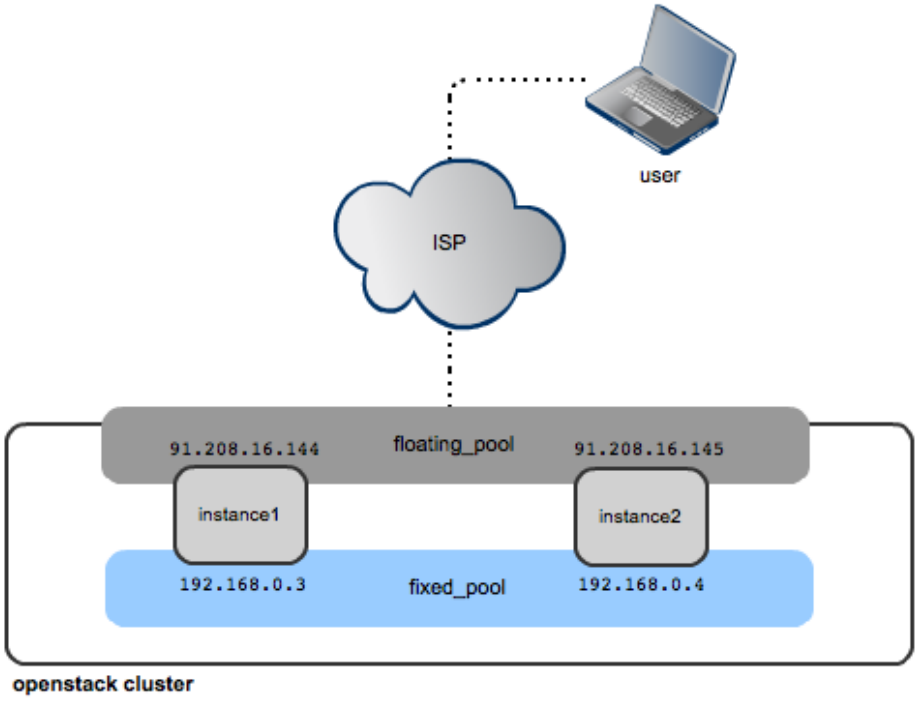


Figura 7.10: Esquema de uso de IPs flotantes.

Fuente: Mirantis

o acceso a otras instancias dentro de la red privada mientras que la dirección IP flotante se usará para acceder a la instancia desde redes públicas.

Para explicar su funcionamiento pondremos un ejemplo de uso. En la Fig.7.10 podemos ver un diagrama que esquematiza un posible escenario donde se refleja el tema que estamos tratando. En la parte inferior de esta figura se puede ver que hay dos instancias: instancia 1 e instancia 2, y tienen direcciones IP que provienen de la red privada. La red privada es como una red detrás de un router NAT. Todo lo que está detrás del router NAT no se puede abordar directamente, por lo tanto, no se puede acceder a las direcciones IP 192.168.0.3 y 192.168.0.4 directamente desde Internet. Es por eso que en OpenStack, al crear nuestras SDN para hacer que las instancias sean accesibles, se necesitan direcciones IPs flotantes como se puede ver en la parte superior de la figura.[61]

Por tanto, una dirección IP flotante es una dirección IP que está reservada para una instancia y está expuesta en el lado externo del router NAT. De este modo, para que el tráfico externo llegue a nuestra instancia creada, el tráfico externo deberá ir directamente a la dirección IP flotante de la ins-

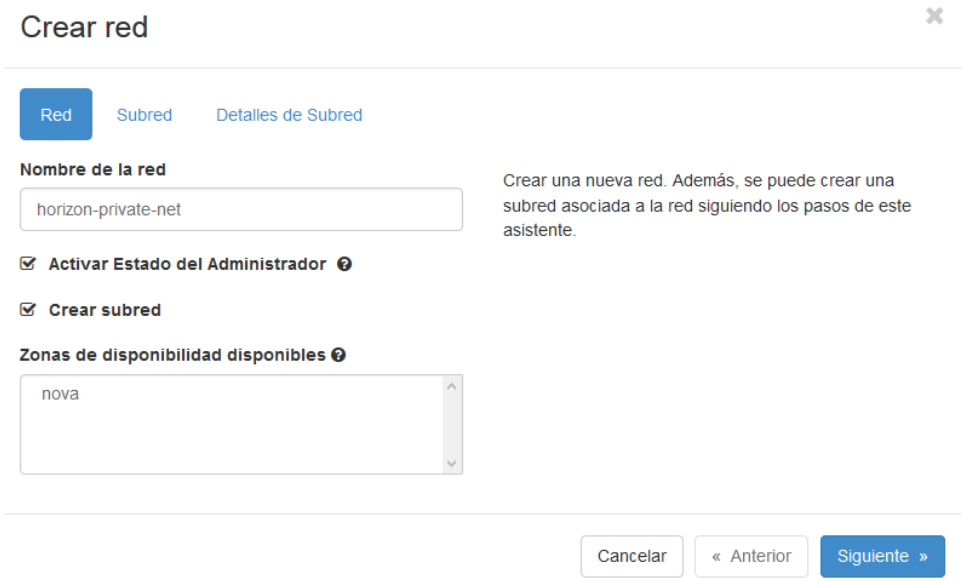


Figura 7.11: Creando una red desde Horizon.

tancia y es el router SDN el que se asegura de que todo el tráfico que llega, por ejemplo, a la IP 91.208.16.144, se reenvíe a la 192.168.0.2.

Hay una parte interesante en la configuración de todo esto y es que la dirección IP flotante no es conocida por la instancia como ya hemos comentado, sino que es conocida únicamente por el router SDN que la usa para llegar a la instancia.

7.3.5. Creación de redes

Ya tenemos configurado un entorno de proyecto con un administrador de proyecto. El siguiente paso es comenzar a trabajar con instancias, Pero antes de que podamos comenzar a trabajar con instancias debemos encargarnos de definir las redes.

Desde dentro de Horizon, en el apartado "Proyecto->Red->Redes", el primer paso será crear una nueva red definida por software. Para hacerlo, hacemos clic en "Crear red" y le ponemos el nombre "horizon-private-net" (Fig.7.11). Esta será nuestra red privada o interna.

Al configurar una SDN, es muy importante hacer una distinción entre la red interna y la red externa. La red interna es a la que se conectarán las máquinas virtuales que creemos mientras que la red externa es la que nos permitirá conectarnos con nuestra infraestructura de red física.

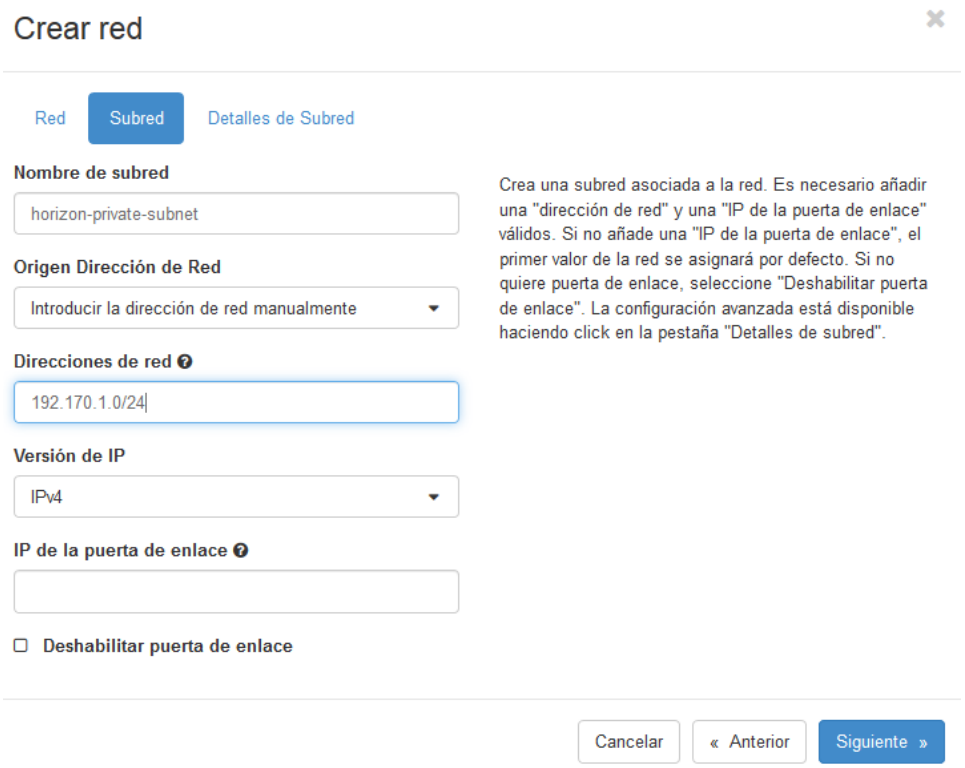


Figura 7.12: Creando una subred desde Horizon.

El siguiente paso es crear una subred desde la pestaña de la misma ventana "Subred". Vamos a llamarla "horizon-private-subnet". Esta subred necesita una dirección de red, usaremos 192.170.1.0/24 como vemos en la Fig.7.12. Se puede usar cualquier dirección IP que deseemos siempre que sea una dirección IP privada. Esta es la subred que usarán internamente solo las máquinas virtuales y permitirá que las máquinas virtuales en la nube se comuniquen entre sí.

En nuestro rango de red IP, es probable que deseemos tener una puerta de enlace o *gateway* también. Podemos especificar la puerta de enlace, pero si no lo hacemos, obtendremos una dirección de puerta de enlace predeterminada, ya que en cualquier red para que las máquinas se comuniquen con el mundo exterior, necesitamos pasarelas en todo momento. En caso de que no queramos una puerta de enlace, podemos hacer clic específicamente en "Deshabilitar puerta de enlace", pero eso normalmente no es algo que deseemos hacer.

En la siguiente pestaña tenemos el bloque de "Detalles de Subred". En los detalles de la subred podemos especificar el resto de elementos necesarios

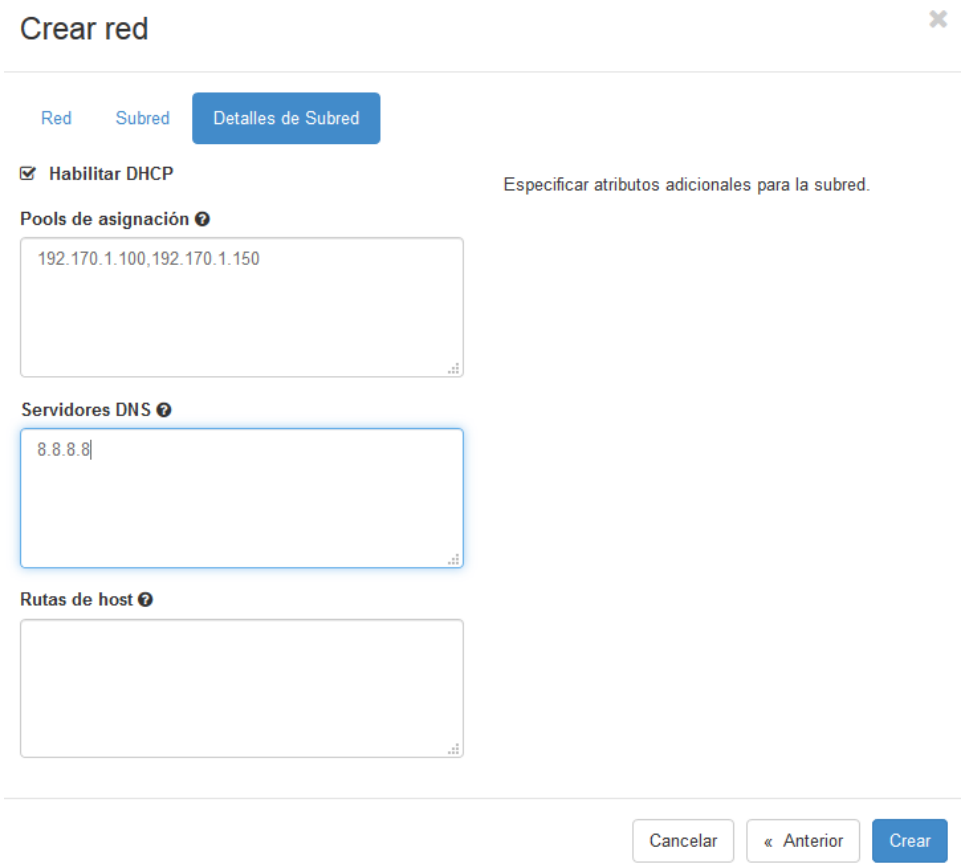
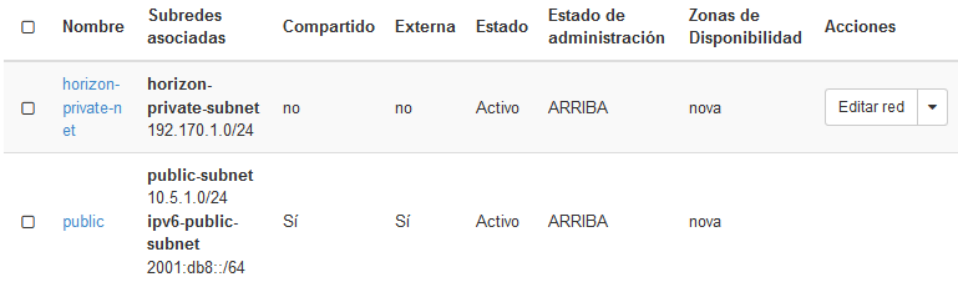


Figura 7.13: Configuración de DHCP y DNS.

para especificar nuestra subred. Básicamente esto se reduce a la configuración de DHCP. DHCP se asegurará de que nuestras instancias obtengan una dirección IP por defecto en el rango de subred especificado. En el campo "Pools de asignación", necesitamos crear una lista de direcciones IP separada por comas con la primera dirección en el grupo de asignación y la última dirección en el grupo de asignación, o una dirección por línea si así lo queremos. En este caso, crearemos un rango de direcciones DHCP válidas dentro de la subred que irá desde la 192.170.1.100 a la 192.170.1.150. Como evidencia mostramos la Fig.7.13.

También podemos especificar el o los servidores DNS en el campo "Servidores DNS", asignando así un servidor de nombres a las instancias y también podemos crear rutas a host específicos en el campo "Rutas de host". Normalmente, esto no será necesario ya que la gateway creada por defecto se encargará de ello. Haciendo clic en "Crear" tendremos la red a punto.

En este punto, debemos asegurarnos de que haya también una red públi-



Nombre	Subredes asociadas	Compartido	Externa	Estado	Estado de administración	Zonas de Disponibilidad	Acciones
horizon-private-net	**horizon-private-subnet** 192.170.1.0/24	no	no	Activo	ARRIBA	nova	Editar red
public	**public-subnet** 10.5.1.0/24 **ipv6-public-subnet** 2001:db8::/64	Sí	Sí	Activo	ARRIBA	nova	

Figura 7.14: Panel de redes creadas.

ca. La red pública podríamos crearla de forma análoga clicando de nuevo en "Crear red" y habilitándola como red pública después desde el tenant de administrador, pero está red ya se creo en la instalación inicial de manera automática en base a los parámetros de configuración introducidos en el archivo *local.conf* (ver apéndice B).

Una vez tenemos ambas redes nos aparecerán tal cual se muestra en la Fig.7.14 en nuestro panel.

7.3.6. Creación de routers

El siguiente paso será establecer una conexión entre la red pública y la red privada para lo cual necesitaremos crear un router. En la pestaña "Proyecto->Red->Routers" cliclamos en "Crear Router" y nombramos el router a crear como "horizon-router".

Además tendremos que seleccionar una red externa que será la red pública creada nombrada como "public". Haciendo clic en "Crear Router" tendremos el router creado y en la topología de red que vemos en la Fig.7.15 aparecerá conectado a la red pública. Si pasamos el cursor sobre el icono "horizon-router", como vemos en la Fig.7.16 tendremos diferentes opciones. De ellas seleccionamos "Añadir interfaz" para agregar una nueva interfaz que nos permitirá conectar nuestro router también con la red interna "horizon-private-net", al seleccionarla, automáticamente conecta el router con esa red creando una interfaz que tiene asignada la dirección IP 192.170.1.1, dentro del rango de direcciones de la subred que creamos.

7.3.7. Implementar una instancia desde Horizon

Ahora que se ha configurado la red, podemos encargarnos del resto. Para empezar, tendremos que crear un grupo de seguridad o *security group*. El grupo de seguridad es un conjunto de reglas de firewall que se aplicarán

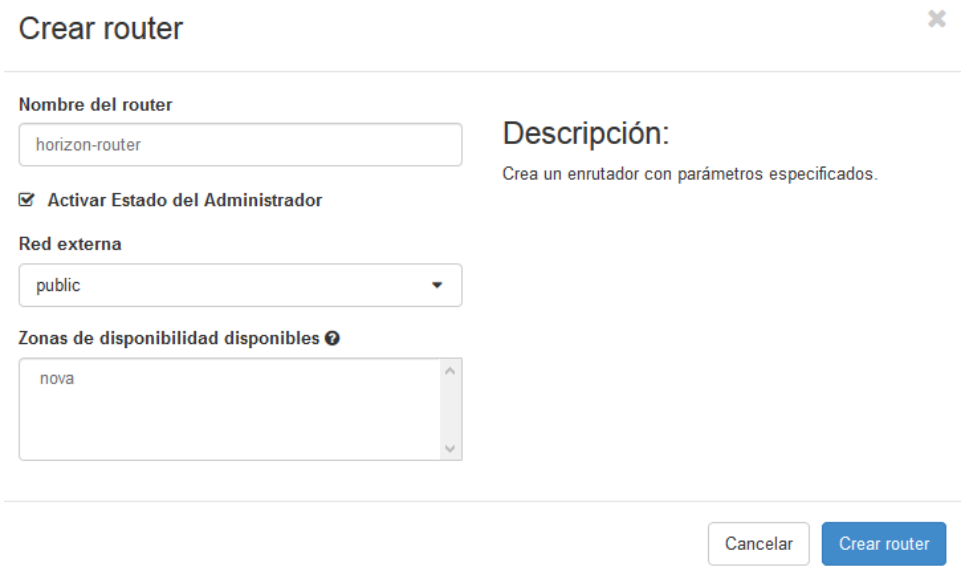


Figura 7.15: Creación de un router.

desde la nube a una instancia. La idea es que la instancia se genere desde una imagen de Glance. En una imagen de Glance, no se pueden definir las reglas de firewall que deben aplicarse y es por eso que tenemos que definirlas a través de la creación de un grupo de seguridad en la nube.

El siguiente paso por tanto es crear un par de claves SSH (*SSH Key Pair*). SSH *keypair* consiste en un conjunto de clave pública-privada. La clave privada se mantiene en la estación de trabajo y la clave pública en la instancia, por tanto, el par de claves SSH nos sirve para permitir una conexión segura a la instancia. La otra opción sería acceder a la instancia utilizando contraseñas predeterminadas y nombres de usuario predeterminados lo cual es muy inseguro por lo que utilizando *SSH Key Pairs*, aumentamos el nivel de seguridad. Estas claves generadas serán las que usemos más tarde para conectarnos a las instancias creadas a través de un terminal.

También necesitamos la imagen de Glance. Una imagen de Glance es como un *live* CD de Linux. Veremos cómo podemos descargar una imagen usando de Cirros desde la nube.

Por último tenemos el flavor que también tendremos que seleccionar. Un flavor es básicamente el perfil de hardware que elegiremos para nuestra instancia. De este modo, si creamos una instancia de una imagen de Glance, determinaremos cuánta RAM, cuánto espacio en disco, etc., va a usar dicha instancia. Una vez que hayamos seguido este proceso, podremos arrancar la instancia.

Figura 7.16: Añadir interfaz a un router.

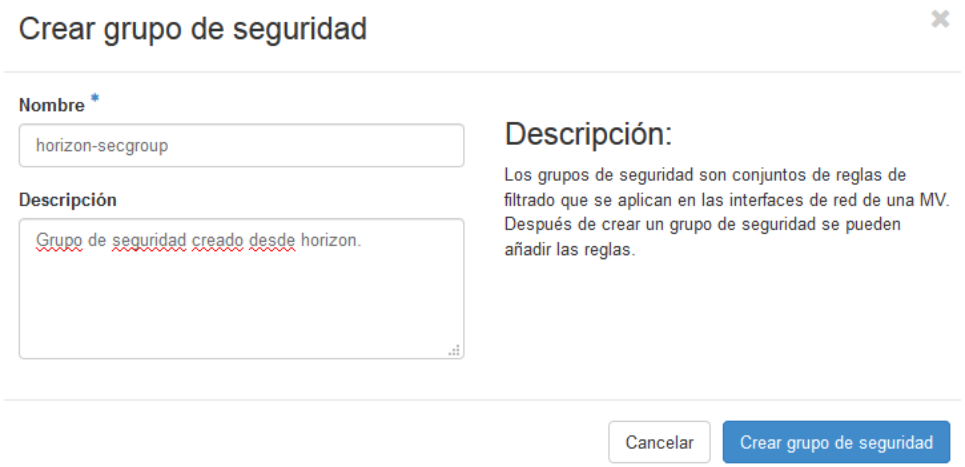


Figura 7.17: Creación de un grupo de seguridad.

En este punto, tenemos una red operativa, lo cual significa que cumplimos con todos los requisitos para implementar instancias desde Horizon. En primer lugar, tendremos que entrar en Horizon como tenant "horizon-user", un usuario cualquiera, ya que el administrador de la nube no va a derivar instancias para los inquilinos si no que lo harán ellos mismos.

7.3.8. Creación de grupos de seguridad

Lo primero que vamos a crear es un grupo de seguridad. Un grupo de seguridad es un *firewall*. La razón por la que estamos trabajando con grupos de seguridad en entornos de nube es que, si se derivan las instancias de la nube de forma dinámica, no es posible crear *firewalls* diferenciados dentro de las instancias de la nube, y es por eso que vamos a definir el firewall desde la nube y asignarlo a las instancias. Para hacerlo, dentro de "Red", en el apartado de "Grupos de seguridad". Le daremos el nombre de "horizon-secgroup" tal como vemos en la Fig.7.17.

Una vez creado haciendo clic en el "Administrar reglas" del campo "Acciones" del grupo de seguridad creado vamos a definir algunas reglas de seguridad. Las reglas de seguridad funcionan de manera similar a como se hacen creando listas de acceso en un router de Cisco por ejemplo. Por defecto, tal como aparece en la Fig.7.18, tenemos permitido tráfico saliente permitido en IPv4 a cualquier parte, y tenemos tráfico saliente permitido en IPv6 en cualquier lugar, además, de manera predeterminada el tráfico entrante no está permitido.

Podemos crear reglas nuevas pulsando en "Agregar regla". En primer

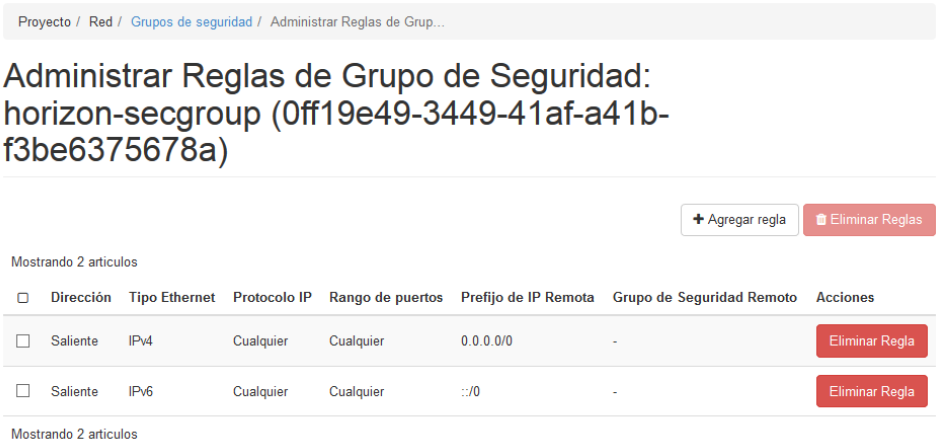


Figura 7.18: Reglas por defecto y creación de nuevas reglas.

lugar agregaremos una regla del tipo "Todos los ICMP" cuyos parámetros vemos en la Fig.7.19. Esta regla especificamos en el campo "Dirección", "Entrante", para indicar que se trata de una regla acerca del tráfico ICMP de entrada, en el campo "Remoto" seleccionamos "CIDR" y como CIDR (*Classless Inter-Domain Routing*) 0.0.0.0/0, indicando así que permitimos cualquier dirección IP, permitiendo de este modo que las instancias creadas puedan ser alcanzadas por cualquiera.

Creamos una regla más del mismo modo. Esta vez vamos a permitir el tráfico HTTP para lo que en "Regla" marcamos "HTTP" con "CIDR", 0.0.0.0/0, y una última, sobre el tráfico SSH con los mismos parámetros, la cual es de vital importancia ya que a las instancias, de manera predeterminada, accederemos como usuarios vía SSH y tenemos que permitirlo en el *firewall*. Una creadas todas se mostrarán en el panel como se aprecia en la Fig.7.20.

7.3.9. Creación de IPs flotantes

En el siguiente paso trabajaremos con direcciones IP flotantes. Una dirección IP flotante es una dirección IP que se utilizará para hacer que la instancia sea accesible desde el exterior. La razón por la que las necesitamos es que por defecto, las instancias son visibles sólo en la red privada. Si solo están en la red privada, nadie externo podrá acceder a ellas, y es por eso que debemos usar direcciones IP flotantes.

Por tanto, una dirección IP flotante es una dirección IP en la nube, y todo lo que entra en la dirección IP flotante se reenviará a la dirección IP de la instancia. Esto significa que la dirección IP flotante debe estar disponible

Figura 7.19: Creación de una regla para el tráfico ICMP.

Figura 7.20: Panel de grupos de seguridad con las reglas creadas.

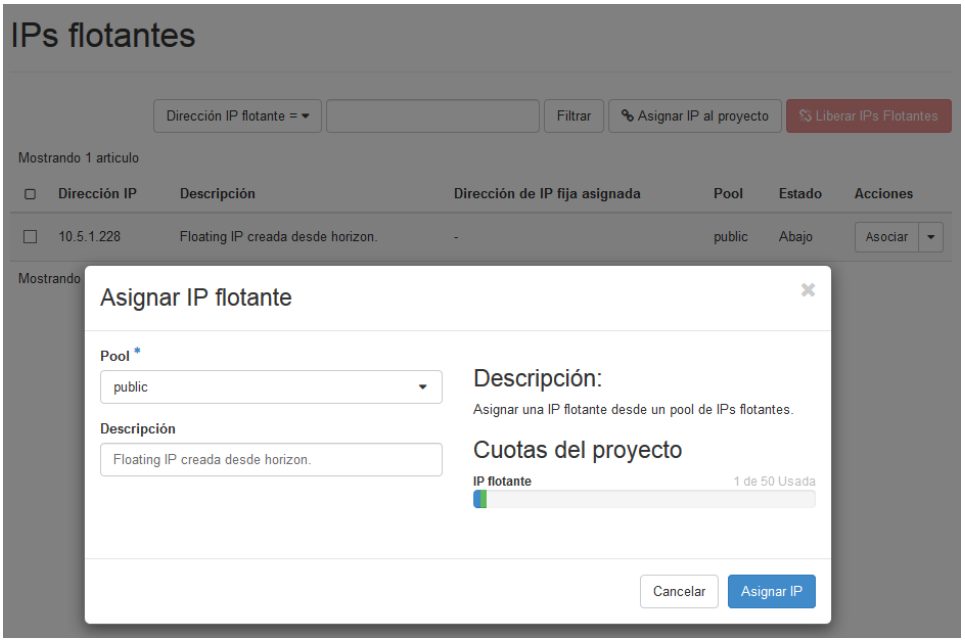


Figura 7.21: Creación de una IP flotante.

en la red externa, que es la red pública que creamos como "public", lo que implica que debemos asegurarnos de que cuando se está construyendo la nube, la red pública tenga una cantidad significativa de direcciones IP disponibles. Por esto motivo, en la preparación de los servidores creamos una interfaz virtual de la tarjeta de red disponible, ya que al sólo tener una, sólo podíamos disponer de una dirección IP flotante. De este modo conseguimos tener un *pool* de direcciones IPs flotantes disponibles para asignar a las instancias.

Comenzamos moviéndonos al apartado "Proyecto->Red->IPs flotantes" y clicamos en "Asignar IP al proyecto" para asignar una IP flotante al proyecto, que se creará automáticamente del *pool* que habíamos creado como vemos en la Fig7.21. Necesitaremos una IP flotante por cada instancia que creemos, luego siguiendo el mismo proceso podemos crear tantas como nuestra cuota nos permita.

Como podemos ver en la Fig.7.22, la IP flotante creada, 10.5.1.228, está dentro del *pool* de IPs especificado en el archivo *local.conf* que iba de la 10.5.1.224 a la 10.5.1.254.

Figura 7.22: IP flotante creada.

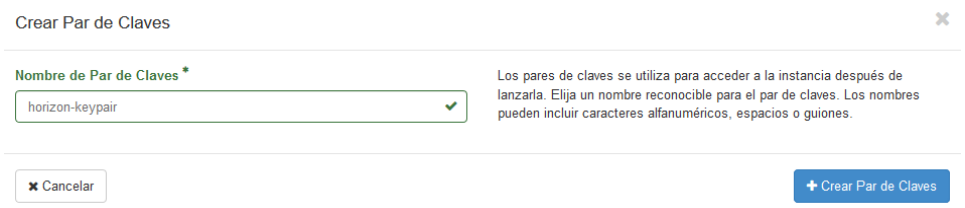


Figura 7.23: Creación de un par de claves.

7.3.10. Creación de pares de claves público-privada

Para llegar al paso de la Fig.7.23 iremos ahora al apartado "Proyecto->Compute", donde vamos a encargarnos de las instancias. Antes de que podamos derivar una instancia, necesitamos un par de claves SSH, por tanto en primer lugar clicamos en "Pares de claves" y dentro del apartado "Crear Par de de Claves". Tenemos que darle un nombre que será "horizon-keypair" y pulsar en "Crear Par de Claves" para que se genere.

Una vez creada podremos descargarla para usarla después. Esta clave privada es una clave que debe estar disponible y por tanto almacenada en la estación de trabajo del usuario final.

Si nos encontramos en el PC que vaya a hacer de estación de trabajo, en mi caso mi ordenador personal, vamos al directorio ".ssh" y copiaremos la clave privada generada para alojarla en la estación de trabajo:

```
atj@wimunet4:~/devstack$$ ls -l | grep key
-rw-rw-r-- 1 atj atj 1679 sep  6 10:52 horizon-keypair
atj@wimunet4:~/devstack$ chmod 400 horizon-keypair
atj@wimunet4:~/devstack$ ls -l | grep MyKeyP
-r-------- 1 atj atj 1679 sep  6 10:52 horizon-keypair
atj@wimunet4:~/devstack$ cp horizon-keypair /home/atj/.ssh
```

"ls -l" muestra los permisos que necesitamos. Esta clave privada será utilizada por el usuario final para conectar a los servidores que tienen una

copia de la clave pública disponible, necesario para entrar a las instancias que se implementarán en OpenStack. También hemos cambiado el tipo de permisos ya que inicialmente no era el correcto. La clave privada debe ser altamente segura, por lo que es aconsejable cambiarle los permisos.

7.3.11. Creación de imágenes

El siguiente paso está encaminado a la creación de imágenes. Necesitamos las imágenes porque las instancias de la nube de OpenStack no están instaladas, sino que se implementan. La instalación funciona para un centro de datos tradicional, donde un administrador pasa por todas las diferentes opciones para instalar una máquina virtual. Eso no es lo que se hace en OpenStack. En OpenStack, la implementamos utilizando una imagen lista para ser usada.

En nuestro caso vamos a usar la imagen de Cirros, que es más ligera y por motivos de capacidad la más aconsejable. Antes de poder usarla necesitamos realizar la descarga de la imagen desde la máquina en la que se esté ejecutando Horizon, en nuestro caso nuestro servidor. Para iniciar la descarga en caso de que no hubiésemos incorporado la misma en el script de configuración inicial o quisiéramos otra podemos escribir desde una consola dentro de la máquina que aloja OpenStack:

```
stack@wimunet4:~/devstack$ wget http://download.cirros-cloud.net
    /0.4.0/cirros-0.4.0-x86_64-disk.img
```

Cirros es una imagen de nube que nos permitirá probar la implementación de instancias en una nube de OpenStack o en cualquier otra nube. Esta imagen no sería la que usaríamos en un entorno cloud de producción, pero es altamente aconsejable para nuestro propósito debido a que las imágenes de cirros son realmente pequeñas, por lo que no supondrá una gran capacidad de los recursos que tenemos disponibles en nuestra máquina. Por supuesto, existen también otras imágenes en la nube disponibles pero estas requieren más requisitos hardware y suelen modificarse para usarse en entornos de producción.

Una vez que tenemos la imagen en la nube disponible, podemos volver a Horizon para crear nuestra imagen con los parámetros de la Fig.7.24, y dentro de “Proyecto->Compute->Imágenes” hacer clic en “Crear imagen”. Vamos a llamar “horizon-cirros” y especificamos el tipo de formato, que será QCOW2. QCOW2 es un formato de imagen de nube común, tenemos otros formatos disponibles, también, pero de nuevo no es recomendable usar ninguno de ellos ya que son para entornos específicos que no deseamos usar por el momento. Tenemos una opción en el campo “Compartir imagen” interesante, ya que permite establecer una imagen como protegida o no.

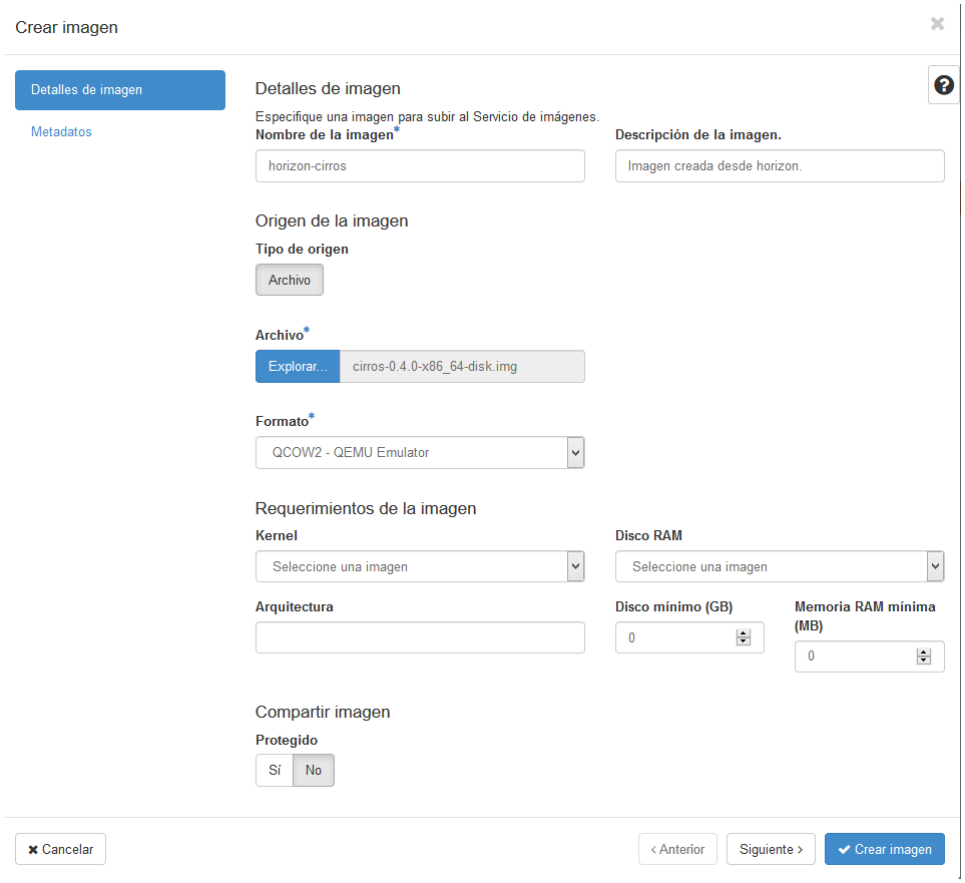


Figura 7.24: Creación de una imagen.

Puesto que es una imagen que como usuario tenant, queremos que esté disponible para todos los demás, marcaremos la opción “No”. Después de hacerlo, haciendo clic en “Crear imagen” importaremos la imagen a nuestro entorno de nube OpenStack.

7.3.12. Creación de instancias

Una vez que la imagen se ha importado con éxito, vamos a “Proyecto->Compute->Instancias” y seleccionamos “Lanzar instancia”. Esto nos lleva a una lista de diferentes ítemes que podemos configurar al iniciar una instancia. Todo lo que vemos en la Fig.7.25 está marcado con un asterisco es obligatorio y el resto opcional como podemos ver en, donde en primer lugar le damos un nombre a la instancia, “horizon-instance-1”. La opción “Count” permite lanzar más instancias al mismo tiempo. Haciendo clic en “Siguiente” pasaremos a especificar el origen de arranque, que será nuestra imagen

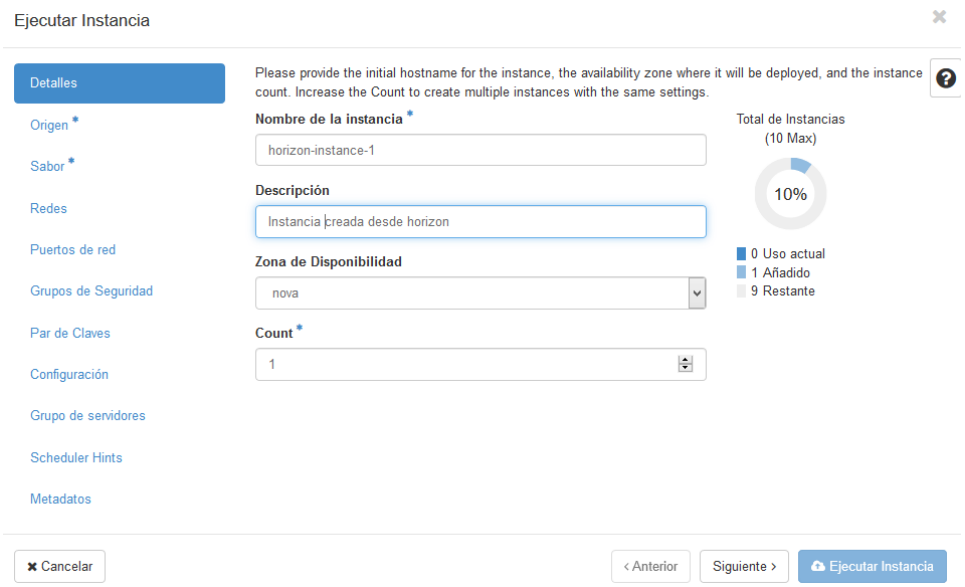


Figura 7.25: Creación de una instancia: Detalles

importada También nos da la opción de elegir crear o no un volumen nuevo y si mantenerlo o no una vez eliminada la instancia. Este volumen no hace referencia al almacenamiento o *storage* si no a la unidad de volumen destinada al arranque de la imagen. Finalmente tendremos la configuración de la Fig.7.26.

Pulsamos "Siguiente" y seleccionamos el "Sabor" (del inglés *flavor*) para especificar el perfil de hardware que va a utilizar nuestra instancia. Podríamos crear nuestros propios flavors si así lo queremos.

De hecho, para tener un flavor que se ajuste más a los requisitos que necesite nuestra máquina vamos a crear el que usaremos para la creación de instancias. Para ello nos autenticamos como usuario administrador en Horizon y vamos al apartado "Administrador->Compute->Sabores". Dentro de este pincharemos en la opción "Crear Sabor" e introduciremos los parámetros de la Fig.7.27.

El motivo de autenticarnos antes como usuario administrador del entorno es que este flavor pueda estar disponible para cualquier usuario ya que si lo hacemos desde un tenant específico, como por ejemplo el que estamos usando, horizon-user, sólo estaría disponible para el proyecto horizon-project. De acuerdo a la configuración introducida, la imagen que asociemos a este flavor tendrá una RAM de 512 MB, 4 GB de disco reservada para la partición /root, una única CPU virtual y como nombre *m1.little*.

Una vez creado, podemos continuar con la creación de la instancia y selec-

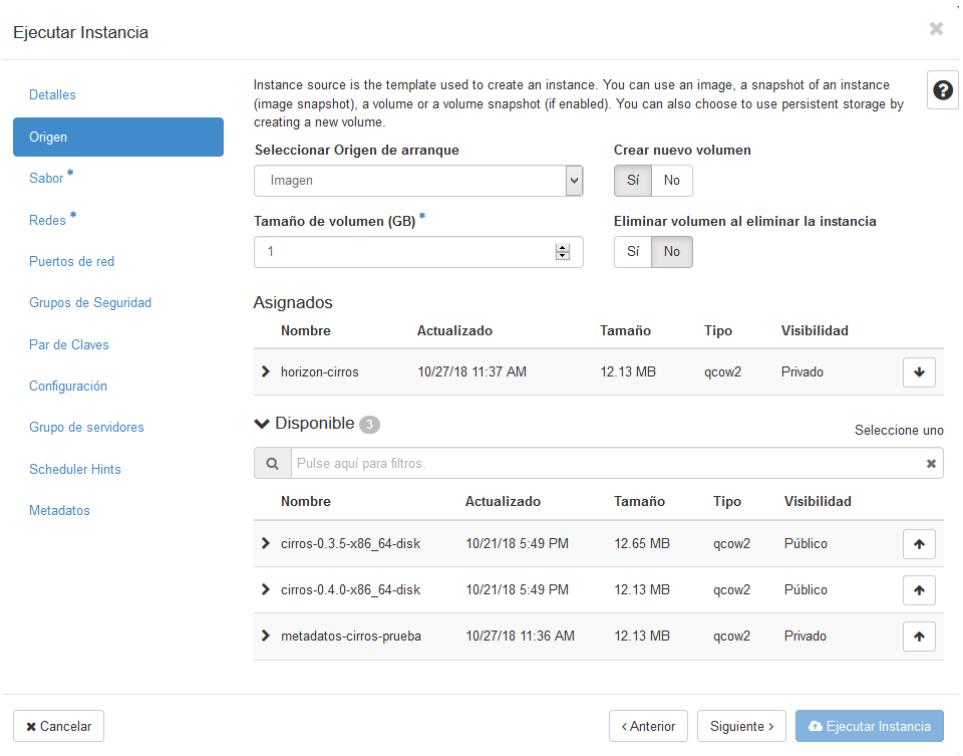


Figura 7.26: Creación de una instancia: Origen.

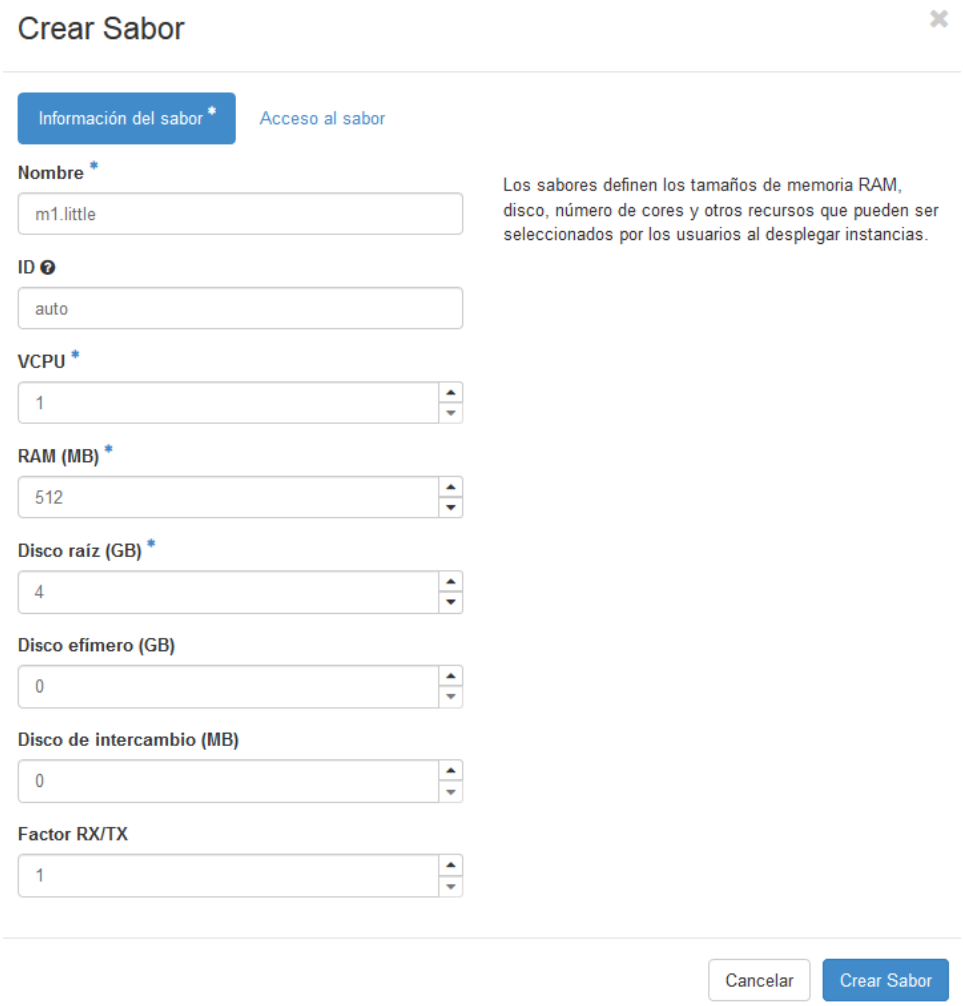


Figura 7.27: Creación de un flavor personalizado.

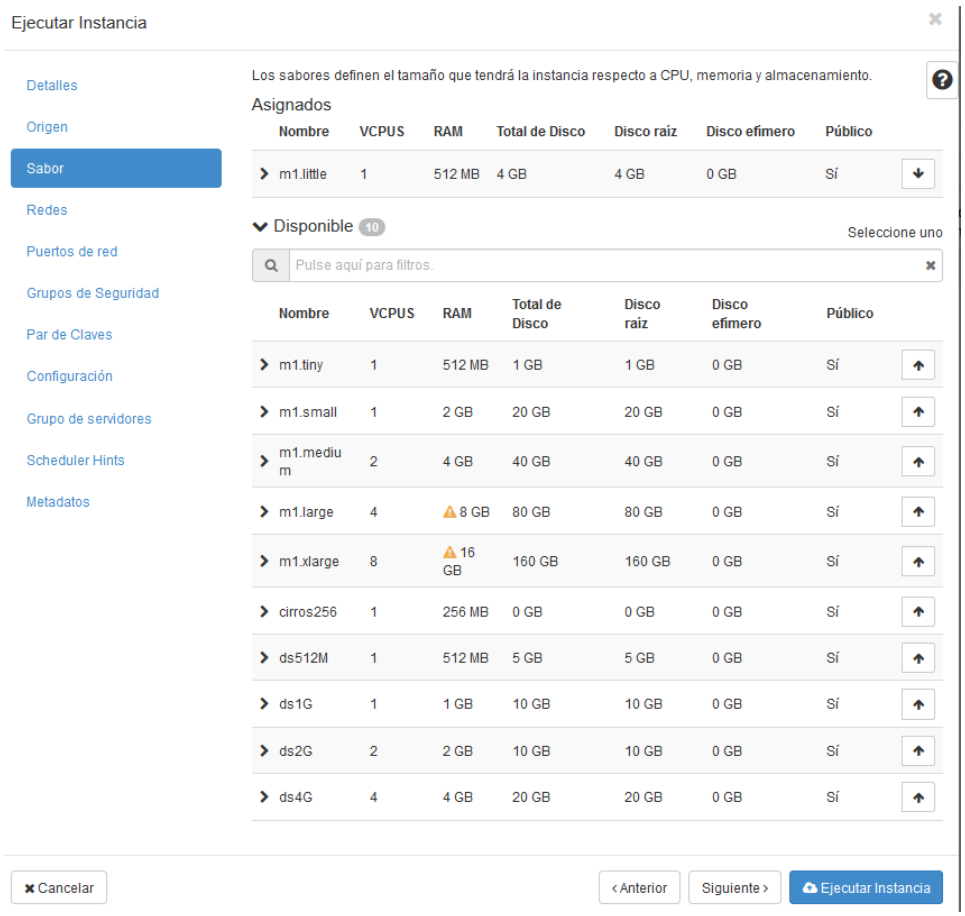


Figura 7.28: Creación de una imagen: Sabor.

cionarlo dentro de las opciones disponibles en el apartado flavor (Fig.7.28).

Con esta configuración ya podríamos crear la instancia pero vamos a especificar algunos parámetros de configuración más. Ahora, como vemos en la Fig.7.29, iremos al apartado "Redes" dentro de las opciones de creación de la instancia veremos las redes que se han creado. Necesitamos que esté disponible en "horizon-private-net" para asignar las instancias que creamos a la red interna y luego hacerlas accesibles a través de una dirección IP flotante.

No necesitamos asignar puertos por lo que pasamos a "Grupos de Seguridad". Es importante asignarle el grupo creado, pues de lo contrario tendremos una instancia que no será accesible en absoluto. Hacemos clic en la flecha hacia arriba para asignar el grupo "horizon-secgroup" que aparece en la Fig.7.30 como grupo de seguridad a la instancia.

Finalmente en el apartado "Par de Claves" elegimos "horizon-keypair" y

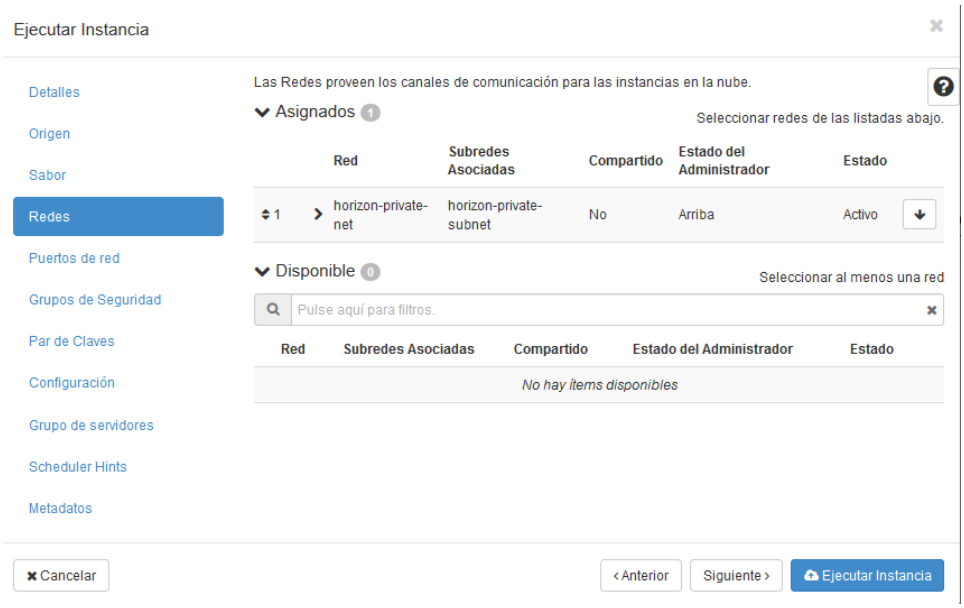


Figura 7.29: Creación de una imagen: Redes.

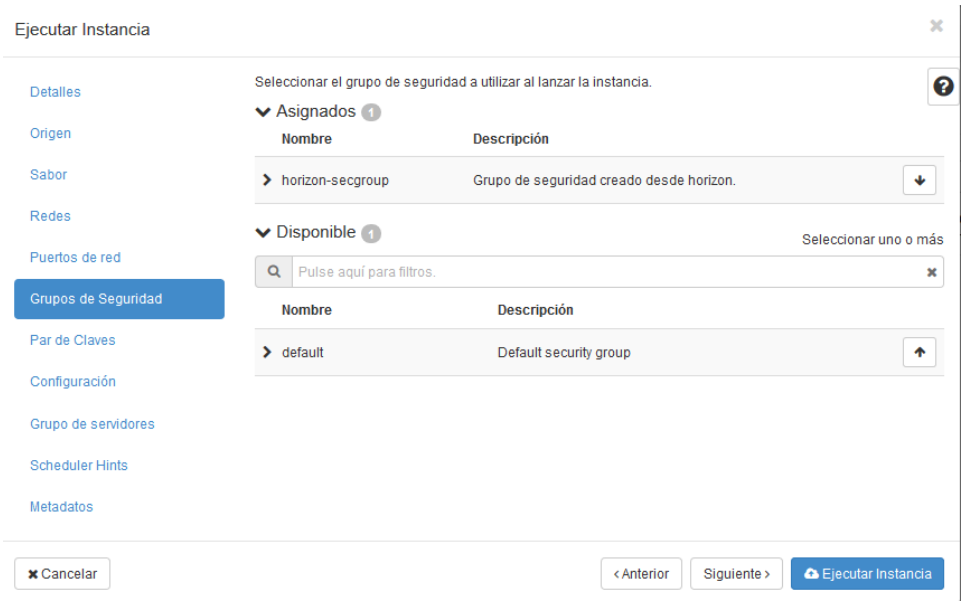


Figura 7.30: Creación de una imagen: Grupos de Seguridad.

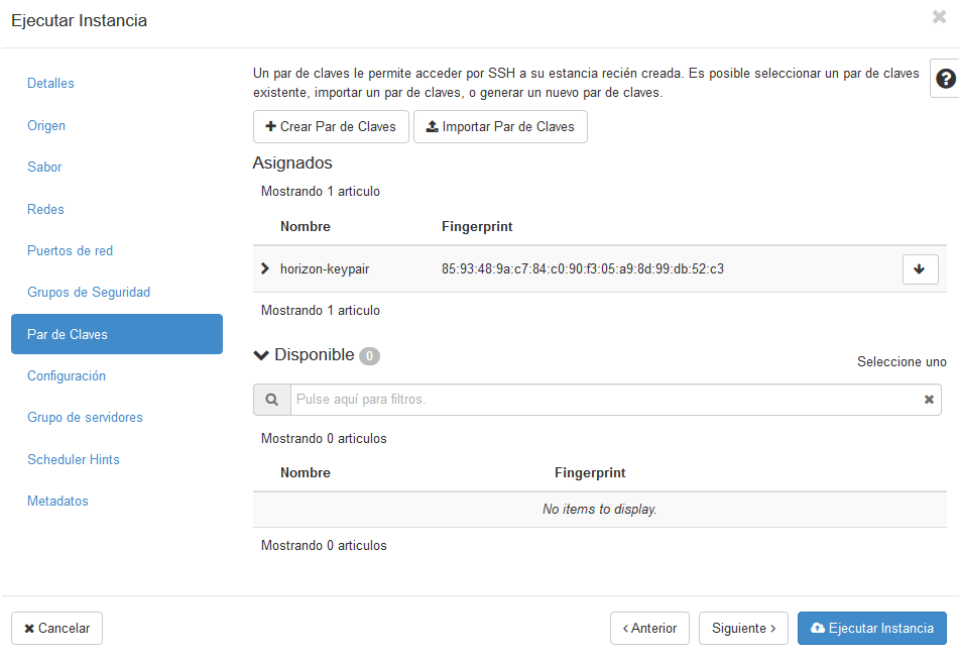


Figura 7.31: Creación de una imagen: Par de Claves.

ya podremos cargar la instancia pulsando en el botón "Ejecutar Instancia" de la Fig.7.31.

7.3.13. Creación de volúmenes de almacenamiento para las instancias

Ya creada nuestra instancia podríamos necesitar algo más. Las instancias, por defecto, no se ocupan del almacenamiento. El almacenamiento en OpenStack es efímero, lo que significa que todo lo que hagamos dentro de nuestra instancia se borrará una vez la derribemos.

Para resolver este inconveniente vamos a crear un volumen accediendo al apartado "Proyecto->Volúmenes->Volúmenes" y tras seleccionar "Crear volumen", nos aparecerá el menú de la Fig.7.32 en el que hemos puesto como nombre del volumen "horizon-volume" que tendrá un tamaño de 1 GiB y como tipo de volumen "lvmdriver-1" un driver para manejar LVM (*Logical Volume Manager*), una implementación de volúmenes lógicos para el kernel Linux.

Una vez creado desde "Proyecto->Compute->Instancias", clicando en la instancia deseada, por ejemplo "horizon-instance-1", podremos asociar el volumen creado a la instancia seleccionando "Asociar volumen" dentro del

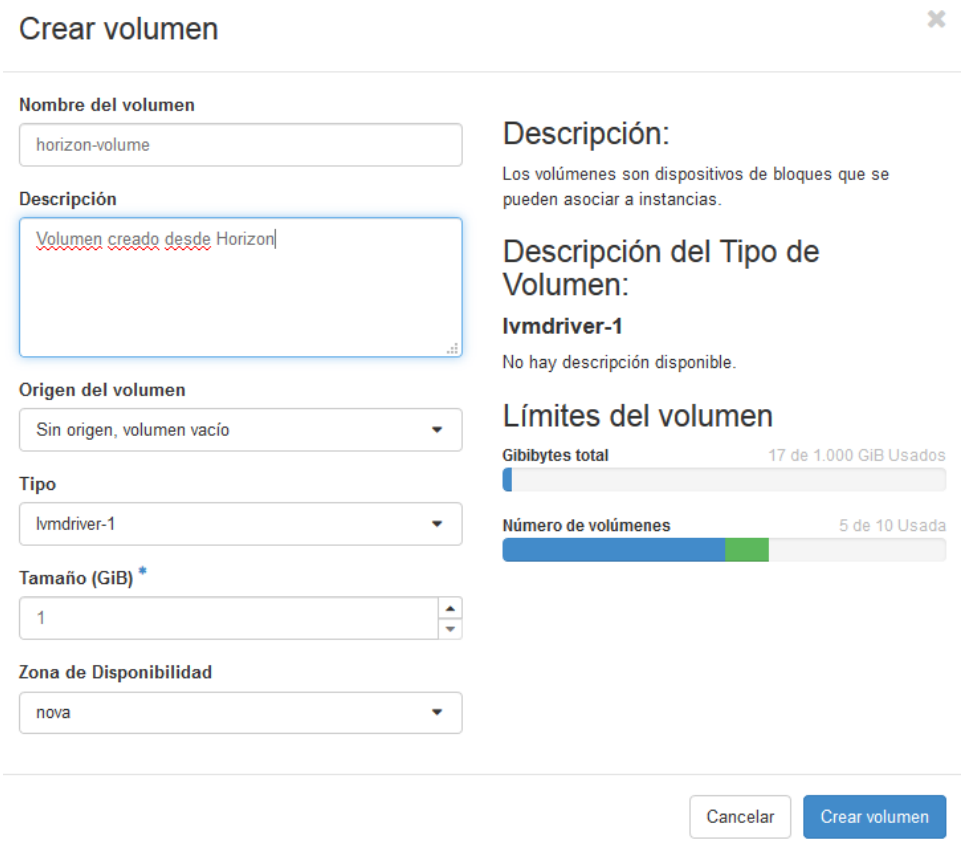


Figura 7.32: Creación de un volumen

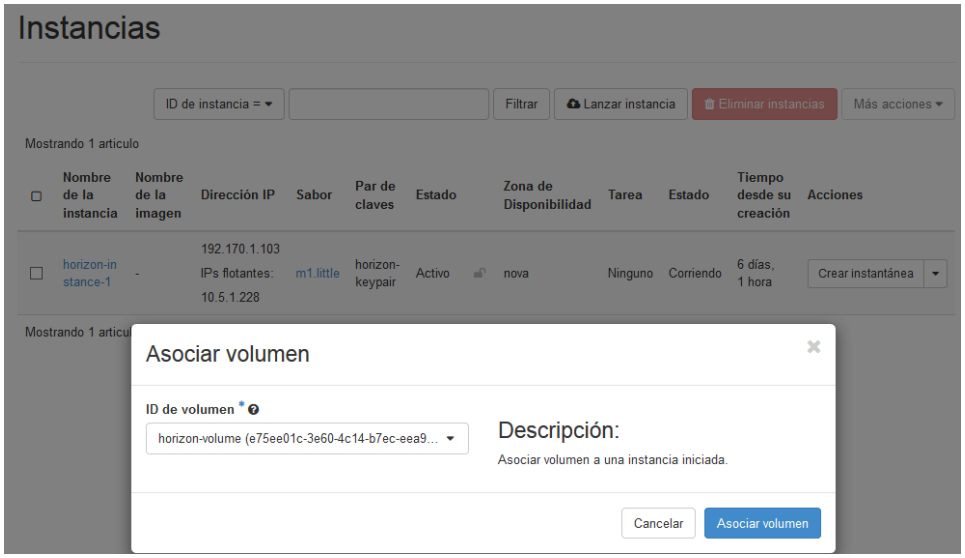


Figura 7.33: Asociar un volumen a una instancia

campo “Acciones” de la instancia y seleccionando el volumen creado como vemos en la Fig.7.33. Tras pulsar en “Asociar volumen”, podremos disponer de este desde la instancia.

7.3.14. Asignando IPs flotantes a las instancias

Por último, será necesario para hacer las instancias accesibles desde fuera de nuestra red, la asociación de estas con una IP flotante que nos ayude a conseguir dicho fin.

Para ello, desde “Proyecto->Red->IPs Flotantes”, dentro del campo “Acciones” de la IP flotante creada desde Horizon, seleccionamos la opción “Asociar” y como puerto a asociar elegimos la instancia “horizon-instance-1”, clicando en “Asociar” de nuevo (ver Fig.7.34), habremos establecido la unión de ambas.

7.4. Configuración de un escenario mediante la CLI

En este apartado vamos a ver como se puede crear también un escenario como en el caso anterior, pero esta vez desde línea de comandos. Puesto que hemos ido explicando y motivando cada paso en el apartado anterior, vamos a ir directos aquí a las pasos a seguir para alcanzar nuestro objetivo, listando los comandos necesarios para la consecución del acceso a las instancias.

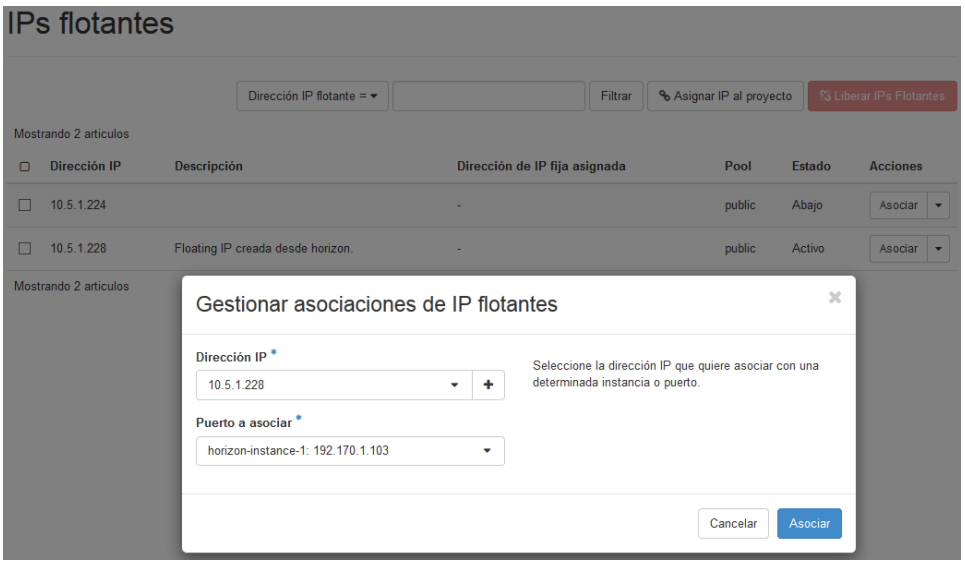


Figura 7.34: Asociar IP flotante a una instancia

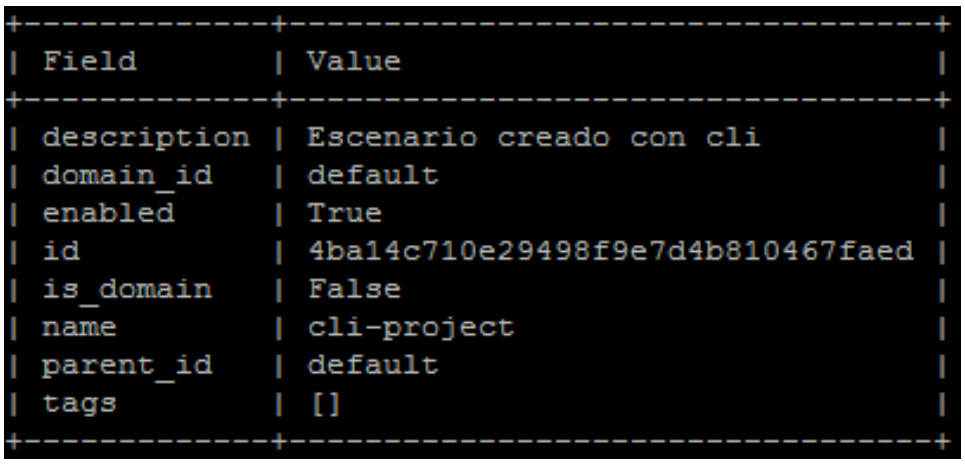
La lista de comandos de OpenStack para administrar la CLI se puede consultar en la documentación pertinente.[62] En ella podemos ver una interfaz genérica para gestionar diversos proyectos, como ejemplos:

- **openstack network**. Para gestionar la conectividad en Neutron.
- **openstack projet**. Gestión de los proyectos con Keystone.
- **openstack server**. Gestión de instancias en Nova.
- **openstack stack**. Gestión de pilas o *stacks* en Heat.
- **openstack volume**. Gestión de volúmenes en Cinder.

Para automatizar el proceso que aquí veremos y ver así la evolución en la administración de OpenStack de la que hablábamos en la sección 5.6 hemos creado un script en bash con el nombre *cli_creacion_escenario.sh*. Este script podremos verlo en el apéndice C.

7.4.1. Credenciales

Lo primero que debemos hacer al acceder a la CLI, es autenticarnos con las credenciales de OpenStack de aquel usuario tenant con el que queramos acceder a nuestro entorno. Cuando creamos un usuario, Keystone almacena en su base de datos las credenciales del mismo. Para acceder a ellas y autenticarnos, en este caso como usuario administrados, en DevStack se utiliza el



Field	Value
description	Escenario creado con cli
domain_id	default
enabled	True
id	4ba14c710e29498f9e7d4b810467faed
is_domain	False
name	cli-project
parent_id	default
tags	[]

Figura 7.35: Salida de CLI al crear proyecto.

archivo *openrc*, con el cual se indica el nombre del proyecto y el usuario que tenemos:

```
stack@wimunet4:~/devstack$ source openrc admin admin
```

Para establecer las variables de entorno requeridas para los clientes desde la CLI de OpenStack, se debe crear un archivo de entorno llamado archivo *openrc.sh* que en nuestro caso se proporciona al instalar DevStack. Este archivo de entorno específico del proyecto contiene las credenciales que utilizan todos los servicios de OpenStack. De este modo, las variables de entorno se establecen para nuestro *shell* actual. Las variables permiten que los comandos del cliente OpenStack se comuniquen con los servicios OpenStack que se ejecutan en la nube.[63]

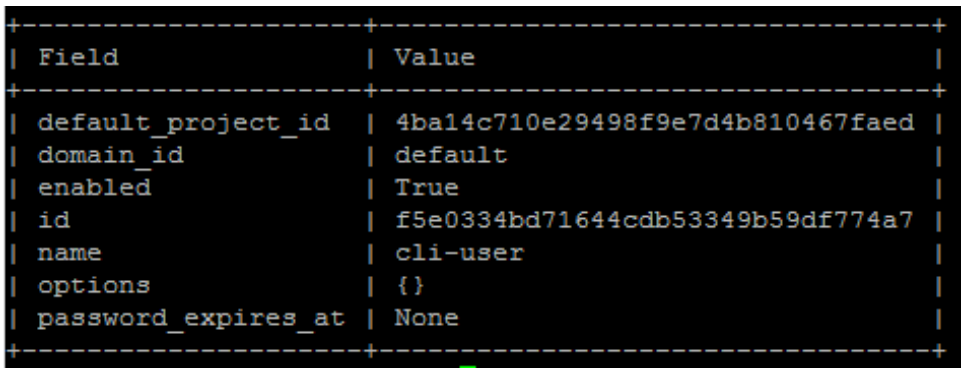
7.4.2. Creación de proyectos o tenants

En lo sucesivo, salvo que se indique lo contrario usaremos para la creación de los recursos que siguen el tenant y proyecto que crearemos en esta sección y siguiente.

La forma de crear un proyecto desde la CLI:

```
stack@wimunet4:~/devstack$ openstack project create --description '
    Escenario creado con cli' cli-project
```

Donde sólo tenemos que indicar el nombre del proyecto, *cli-project*, la descripción es opcional. La salida del comando podemos verla en la Fig.7.35.



Field	Value
default_project_id	4ba14c710e29498f9e7d4b810467faed
domain_id	default
enabled	True
id	f5e0334bd71644cdb53349b59df774a7
name	cli-user
options	{}
password_expires_at	None

Figura 7.36: Salida de CLI al crear un usuario.

Por defecto el proyecto se crea con RAM e instancias ilimitadas, lo cual no es real y además resulta ser el aspecto más crítico en cuanto a la gestión de los recursos disponibles por lo que vamos a limitar la cuota del proyecto en estos dos elementos:

```
stack@wimunet4:~/devstack$ openstack quota set cli-project --instances
    10
stack@wimunet4:~/devstack$ openstack quota set cli-project --ram 4096
```

7.4.3. Creación de usuarios

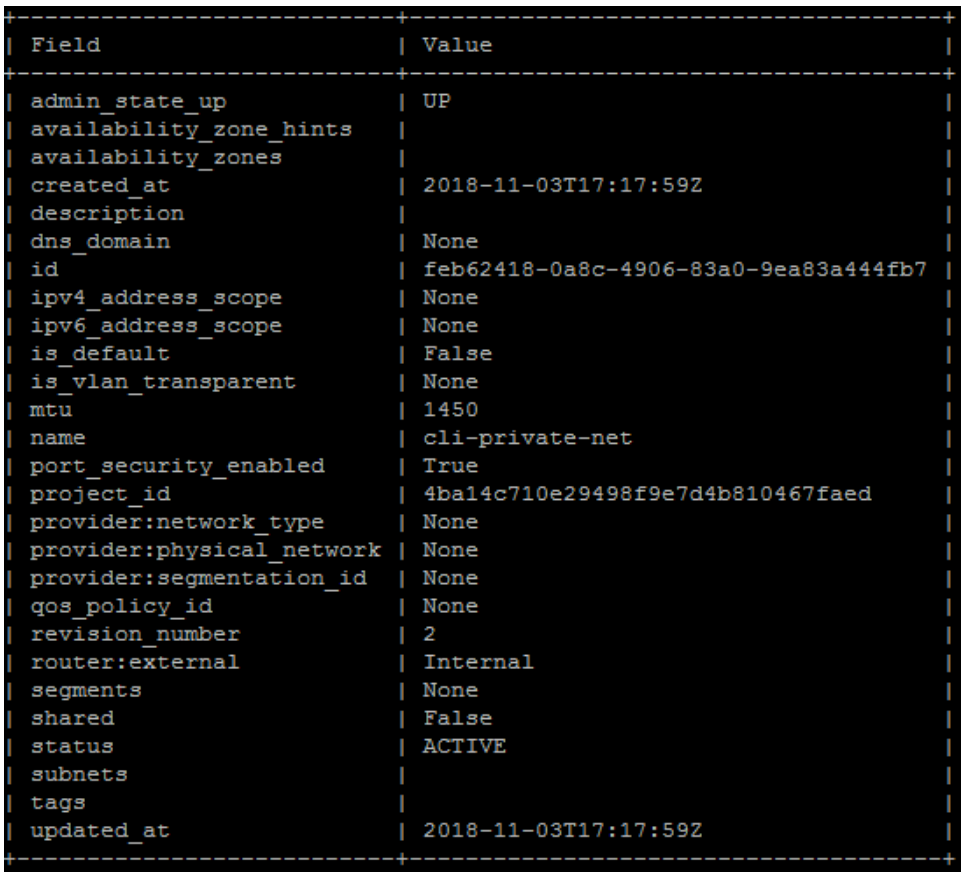
Del mismo modo que hicimos desde Horizon, podemos crear un usuario desde la CLI y asociarlo al proyecto que acabamos de crear escribiendo:

```
stack@wimunet4:~/devstack$ openstack user create --project cli-project
    --password temporary cli-user
```

Donde indicamos que el usuario se llame *cli-user*, el nombre del proyecto al que queremos asociarlo con el parámetro –cli-project, y la password que usará. La salida la tenemos en la Fig.7.36

Además vamos a especificar que el usuario creado asignado al proyecto sea de tipo *Member*.

```
stack@wimunet4:~/devstack$ openstack role add --user cli-user --
    project cli-project Member
```



Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-11-03T17:17:59Z
description	
dns_domain	None
id	feb62418-0a8c-4906-83a0-9ea83a444fb7
ipv4_address_scope	None
ipv6_address_scope	None
is_default	False
is_vlan_transparent	None
mtu	1450
name	cli-private-net
port_security_enabled	True
project_id	4ba14c710e29498f9e7d4b810467faed
provider:network_type	None
provider:physical_network	None
provider:segmentation_id	None
qos_policy_id	None
revision_number	2
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	
tags	
updated_at	2018-11-03T17:17:59Z

Figura 7.37: Salida de CLI al crear una red.

7.4.4. Creación de redes

Vamos a ver en este apartado como crear elementos de conectividad de red, en concreto redes y subredes desde la CLI.

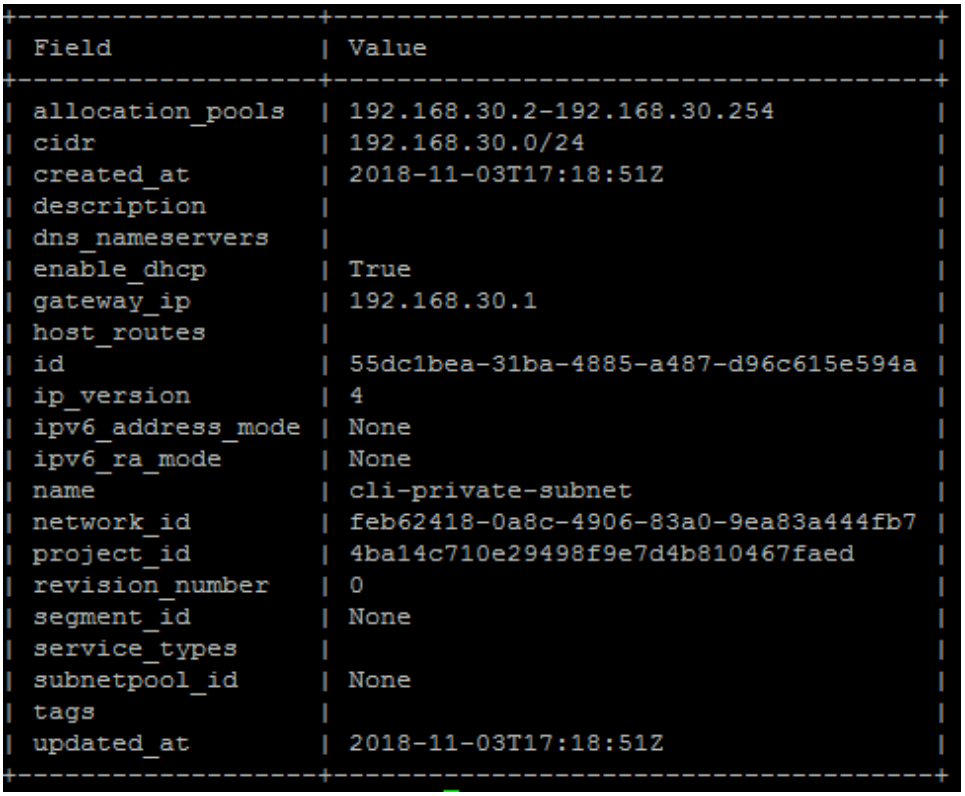
Para crear una red escribimos el siguiente comando:

```
stack@wimunet4:~/devstack$ openstack network create cli-private-net
```

Que nos devolverá la salida de la Fig.7.37. De este modo crearemos una red llamada *cli-private-net*. No necesitamos propiedades adicionales puesto que estas se declaran en la subred. Así pues este será el siguiente paso:

```
stack@wimunet4:~/devstack$ openstack subnet create --network cli-
    private-net --subnet-range 192.168.30.0/24 cli-private-subnet
```

Donde los parámetros que aparecen indican:



Field	Value
allocation_pools	192.168.30.2-192.168.30.254
cidr	192.168.30.0/24
created_at	2018-11-03T17:18:51Z
description	
dns_nameservers	
enable_dhcp	True
gateway_ip	192.168.30.1
host_routes	
id	55dc1bea-31ba-4885-a487-d96c615e594a
ip_version	4
ipv6_address_mode	None
ipv6_ra_mode	None
name	cli-private-subnet
network_id	feb62418-0a8c-4906-83a0-9ea83a444fb7
project_id	4ba14c710e29498f9e7d4b810467faed
revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2018-11-03T17:18:51Z

Figura 7.38: Salida de CLI al crear una subred.

- –network cli-private-net. La red a la que queremos asociar la subred.
- –subnet-range 192.168.30.0/24. El rango de la subred.
- cli-private-subnet. Nombre que le hemos dado a la subred.

Con esto se nos devolverán los parámetros para la subred creada de la Fig.7.38, donde podemos ver todas las propiedades de la subred *cli-private-subnet*:

7.4.5. Creación de routers

Para hacer las redes creadas accesibles necesitamos crear un router. Desde la CLI escribimos:

```
stack@wimunet4:~/devstack$ openstack router create cli-router
```

Para tener acceso a la subred *cli-private-subnet* necesitamos conectar el router que acabamos de generar a esta subred.

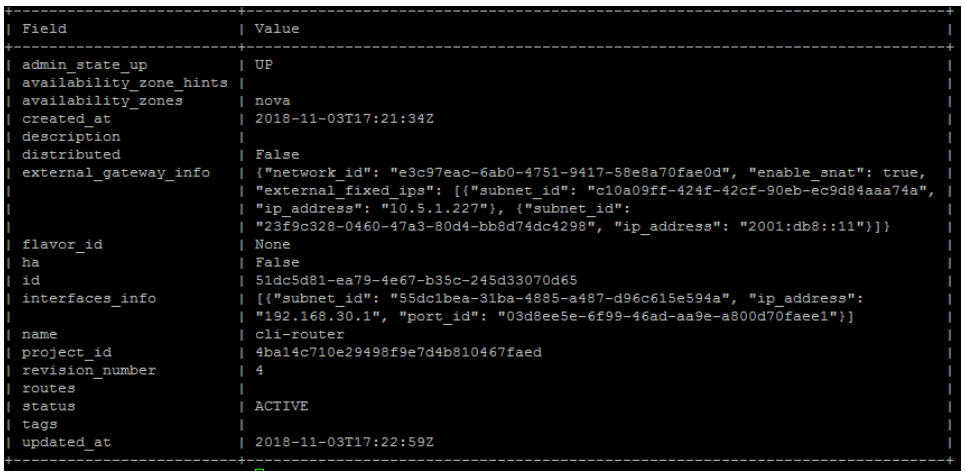


Figura 7.39: Salida de CLI con los detalles del router creado, *cli-router*.

```
stack@wimunet4:~/devstack$ openstack router add subnet cli-router cli-
    private-subnet
```

Y también a la red pública que ya teníamos creada cuyo nombre es *public* con el siguiente comando, añadiendo el parámetro –external-gateway para indicar que se trata de la puerta de enlace a la red externa.

```
stack@wimunet4:~/devstack$ openstack router set cli-router --external-
    gateway public
```

Podemos una vez realizada la configuración de los parámetros de red, ver como queda la configuración del router mediante el siguiente comando:

```
stack@wimunet4:~/devstack$ openstack show --fit-width cli-router
```

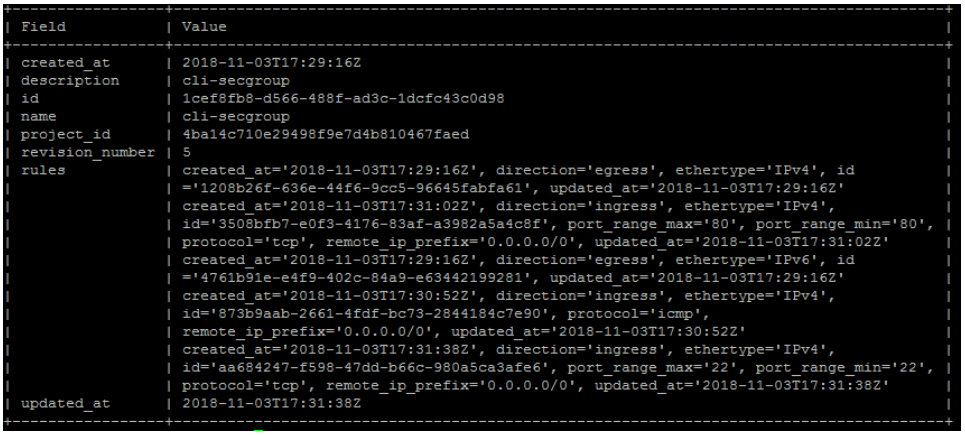
Que da como salida la Fig.7.39

7.4.6. Creación de grupos de seguridad

Ahora vamos a ver como crear un grupo de seguridad que como sabemos hará las veces de firewall para las instancias:

```
stack@wimunet4:~/devstack$ openstack security group create cli-
    secgroup
```

Con esto habremos creado un grupo de seguridad llamado *cli-secgroup*. Para que actúe como nosotros queremos, crearemos una serie de reglas que



Field	Value
created_at	2018-11-03T17:29:16Z
description	cli-secgroup
id	1cef8fb8-d566-488f-ad3c-1dcfc43c0d98
name	cli-secgroup
project_id	4ba14c710e29498f9e7d4b810467faed
revision_number	5
rules	created_at='2018-11-03T17:29:16Z', direction='egress', ethertype='IPv4', id ='1208b26f-636e-44f6-9cc5-96645fabfa61', updated_at='2018-11-03T17:29:16Z' created_at='2018-11-03T17:31:02Z', direction='ingress', ethertype='IPv4', id='3508bfb7-e0f3-4176-83af-a3982a5a4c8f', port_range_max='80', port_range_min='80', protocol='tcp', remote_ip_prefix='0.0.0.0/0', updated_at='2018-11-03T17:31:02Z' created_at='2018-11-03T17:29:16Z', direction='egress', ethertype='IPv6', id ='4761b91e-e4f9-402c-84a9-e63442199281', updated_at='2018-11-03T17:29:16Z' created_at='2018-11-03T17:30:52Z', direction='ingress', ethertype='IPv4', id='873b9aab-2661-4fdf-bc73-2844184c7e90', protocol='icmp', remote_ip_prefix='0.0.0.0/0', updated_at='2018-11-03T17:30:52Z' created_at='2018-11-03T17:31:38Z', direction='ingress', ethertype='IPv4', id='aa684247-f598-47dd-b66c-980a5ca3afe6', port_range_max='22', port_range_min='22', protocol='tcp', remote_ip_prefix='0.0.0.0/0', updated_at='2018-11-03T17:31:38Z'
updated_at	2018-11-03T17:31:38Z

Figura 7.40: Salida de CLI con los detalles del grupo de seguridad creado.

permitan el tráfico que deseamos y el resto no, como si de listas de acceso se tratara. Por defecto, todo el tráfico saliente esta permitido. Para el entrante:

```
stack@wimunet4:~/devstack$ openstack  security group rule create --
    remote-ip 0.0.0.0/0 --protocol tcp --dst-port 22  --ingress cli-
    secgroup
stack@wimunet4:~/devstack$ openstack  security group rule create --
    remote-ip 0.0.0.0/0 --protocol icmp --ingress cli-secgroup
stack@wimunet4:~/devstack$ openstack  security group rule create --
    remote-ip 0.0.0.0/0 --protocol tcp --dst-port 80:80  --ingress cli
    -secgroup
```

Donde hemos creado tres reglas de seguridad asignadas al grupo de seguridad *cli-group* que permiten en tráfico ICMP, SSH y HTTP respectivamente desde cualquier dirección.

Podemos consultar si las reglas del grupo se han creado con éxito escribiendo:

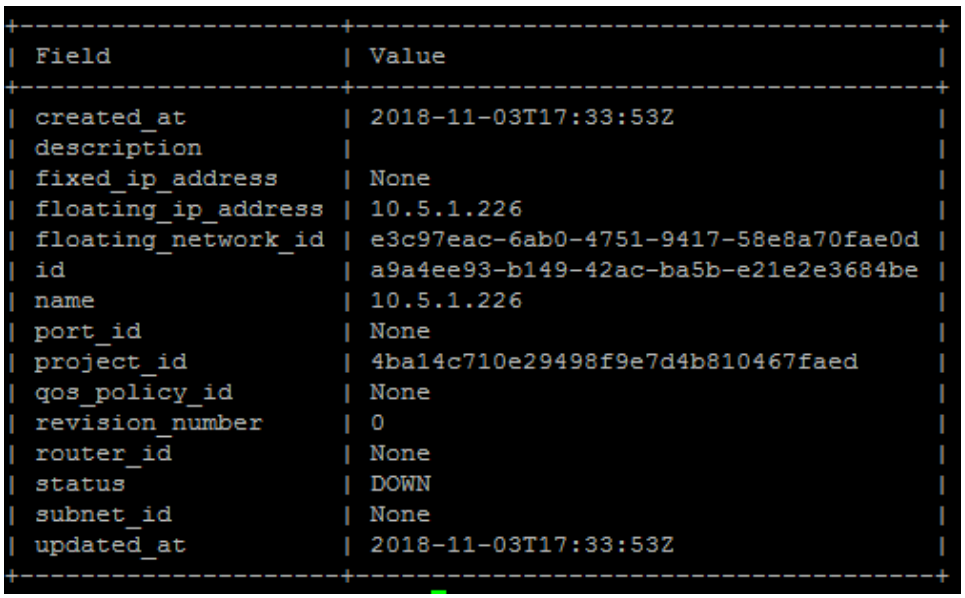
```
stack@wimunet4:~/devstack$ openstack security group show --fit-width
    cli-secgroup
```

Cuya salida mostramos en la Fig.7.40

7.4.7. Creación de IPs flotantes

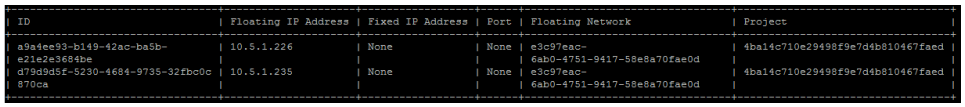
Otro paso necesario como sabemos, es la creación de IPs flotantes para poder acceder remotamente a las instancias. Para ello escribimos:

```
stack@wimunet4:~/devstack$ openstack floating ip create public
```



Field	Value
created_at	2018-11-03T17:33:53Z
description	
fixed_ip_address	None
floating_ip_address	10.5.1.226
floating_network_id	e3c97eac-6ab0-4751-9417-58e8a70fae0d
id	a9a4ee93-b149-42ac-ba5b-e21e2e3684be
name	10.5.1.226
port_id	None
project_id	4ba14c710e29498f9e7d4b810467faed
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-11-03T17:33:53Z

Figura 7.41: Salida de CLI al crear una IP flotante*cli-router*.



ID	Floating IP Address	Fixed IP Address	Port	Floating Network	Project
a9a4ee93-b149-42ac-ba5b-e21e2e3684be	10.5.1.226	None	None	e3c97eac-6ab0-4751-9417-58e8a70fae0d	4ba14c710e29498f9e7d4b810467faed
d79d9d5f-5230-4684-9735-32fbc0c870ca	10.5.1.235	None	None	e3c97eac-6ab0-4751-9417-58e8a70fae0d	4ba14c710e29498f9e7d4b810467faed

Figura 7.42: Salida de CLI con listado de IPs flotantes*cli-router*.

Que nos devolverá los detalles de la IP flotante creada, 10.5.1.226, alojada dentro del pool creado de inicio en *local.conf* y que podemos ver en la Fig.7.41.

Este paso lo repetiremos las veces que sea necesaria, tantas como IPs flotantes necesitemos. Un listado de todas las IPs flotantes que tenemos podemos obtenerlo escribiendo en la CLI:

```
stack@wimunet4:~/devstack$ openstack floating ip list
```

En la Fig.7.42 vemos como salida el listado devuelto con las IPs flotantes disponibles.

7.4.8. Creación de pares de claves público-privada

Vamos a crear ahora una clave SSH para el acceso a las instancias que llamaremos *cli-keypair* y guardaremos como *cli-keypair.pem* para su posterior uso cuando queramos acceder a las instancias a las que se les haya

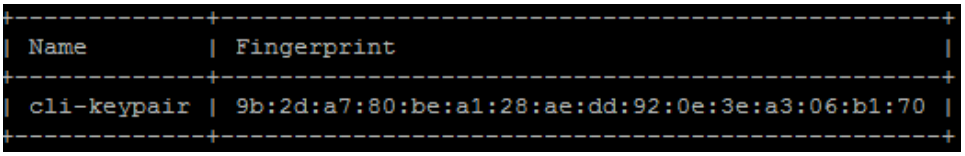


Figura 7.43: Salida de CLI con listado de claves público-privadas *cli-router*.

asignado esa clave:

```
stack@wimunet4:~/devstack$ openstack keypair create cli-keypair > cli-
    keypair.pem
stack@wimunet4:~/devstack$ chmod 0600 cli-keypair.pem
stack@wimunet4:~/devstack$ ssh-add cli-keypair.pem
```

Además hemos modificado los permisos de la clave creada para que sea más segura, pues no es algo a lo que queramos que todos los usuarios tengan acceso y la hemos añadido a nuestro agente de claves ssh.

Para comprobar que se ha creado correctamente podemos escribir:

```
stack@wimunet4:~/devstack$ openstack keypair list
```

Y veremos (Fig.7.43) como aparece disponible con su correspondiente huella digital o *fingerprint*.

7.4.9. Creación de imágenes

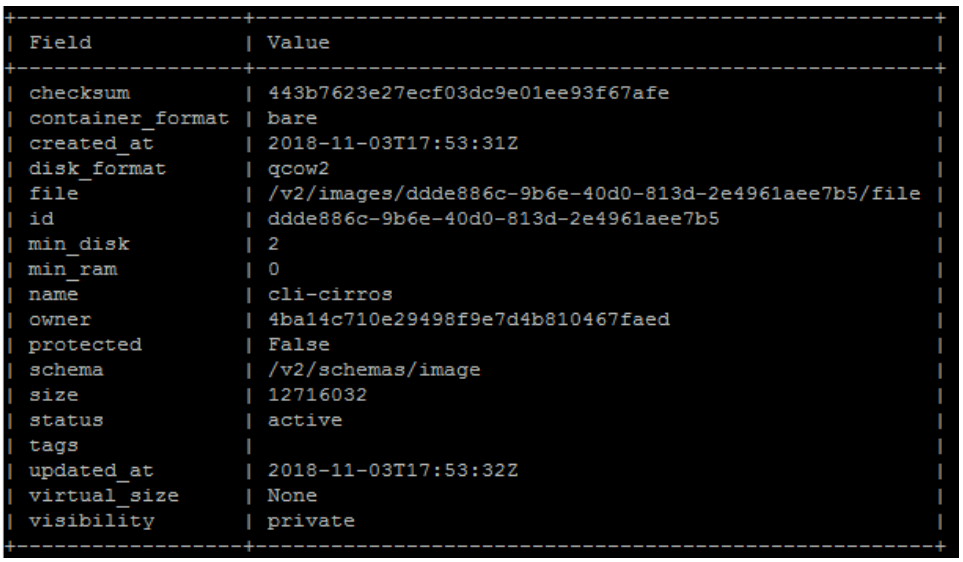
Para desplegar instancias en OpenStack desde la línea de comandos necesitamos obtener primero una imagen de Glance. Almacenaremos todas la imágenes que queramos descargar en la carpeta *images* creada para tal fin:

```
stack@wimunet4:~/devstack$ mkdir image
```

Para preparar la imagen de Cirros la descargamos del repositorio de imágenes deseado como ya hicimos en el apartado 7.3.11 con el comando *wget*. En nuestro caso este paso no es estrictamente necesario al no ser que necesitemos otra distribución o imagen, puesto que en el archivo de configuración *local.conf* que recordemos podemos ver en el apéndice B ya impusimos la descarga automática de la misma.

Para crear la imagen escribimos:

```
stack@wimunet4:~/devstack$ openstack image create --disk-format qcow2
    --min-disk 2 --file /opt/stack/devstack/images/cirros-0.4.0-x86_64
    -disk.img  --private cli-cirros
```



Field	Value
checksum	443b7623e27ecf03dc9e01ee93f67afe
container_format	bare
created_at	2018-11-03T17:53:31Z
disk_format	qcow2
file	/v2/images/ddde886c-9b6e-40d0-813d-2e4961aee7b5/file
id	ddde886c-9b6e-40d0-813d-2e4961aee7b5
min_disk	2
min_ram	0
name	cli-cirros
owner	4ba14c710e29498f9e7d4b810467faed
protected	False
schema	/v2/schemas/image
size	12716032
status	active
tags	
updated_at	2018-11-03T17:53:32Z
virtual_size	None
visibility	private

Figura 7.44: Salida de la CLI al crear imagen.

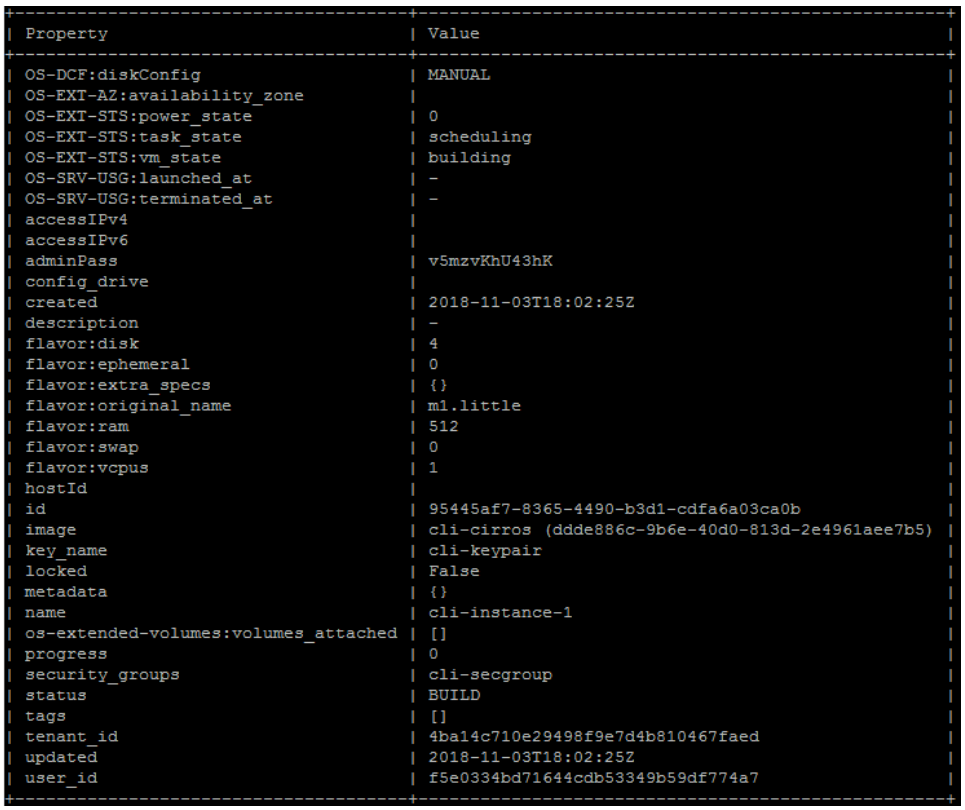
Con estemos comando estamos indicando que queremos crear una imagen de las siguientes características:

- –disk-format. Formato *qcow2* que es el predeterminado para los entornos cloud.
- –min-disk 2. Creando así un disco de al menos 2 GB.
- –file /opt/stack/devstack/images/cirros-0.4.0-x86_64-disk.img. Para indicar la ruta en la que se encuentra la imagen de cirros.
- –private. Indicamos que la imagen sea privada y por tanto sólo disponible para este proyecto.
- cli-cirros. Nombre que damos a la imagen creada.

Después de introducir el comando podremos ver en la Fig.7.44 sus propiedades. De este modo OpenStack a creado nuestra imagen con el servicio de Glance y está ya lista para ser usada.

7.4.10. Creación de instancias

Para la creación de nuestra instancia, tenemos ya un flavor personalizado que creamos en la sección 7.3.12. Del mismo modo podríamos haberla creado desde la CLI escribiendo:



Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	
OS-EXT-STS:power_state	0
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	-
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
adminPass	v5mzvKhU43hK
config_drive	
created	2018-11-03T18:02:25Z
description	-
flavor:disk	4
flavor:ephemeral	0
flavor:extra_specs	{}
flavor:original_name	m1.little
flavor:ram	512
flavor:swap	0
flavor:vcpus	1
hostId	
id	95445af7-8365-4490-b3d1-cdfa6a03ca0b
image	cli-cirros (ddde886c-9b6e-40d0-813d-2e4961aee7b5)
key_name	cli-keypair
locked	False
metadata	{}
name	cli-instance-1
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	cli-secgroup
status	BUILD
tags	[]
tenant_id	4ba14c710e29498f9e7d4b810467faed
updated	2018-11-03T18:02:25Z
user_id	f5e0334bd71644cdb53349b59df774a7

Figura 7.45: Salida de la CLI al crear una instancia.

```
stack@wimunet4:~/devstack$ source openrc admin admin
stack@wimunet4:~/devstack$ openstack flavor create --ram 512 --disk 4
    --vcpus 1 m1.little
```

Para crear la instancia cuya salida de la CLI vemos en la Fig.7.45 escribiremos:

```
stack@wimunet4:~/devstack$ nova boot --flavor m1.little --image cli-
    cirros --key-name cli-keypair --security-groups cli-secgroup --nic
     net-name=cli-private-subnet cli-instance-1
```

En este comando estamos usando Nova, que como sabemos se encarga de la gestión de la instancias para que cree en una acorde a los parámetros introducidos que explicamos a continuación:

- –flavor. Con este parámetro indicamos el sabor o flavor, que en este caso será el ya creado *m1.litlle*.
- –image. Indicamos la imagen de Glance que vamos a usar. Como vemos

hemos sellecionado la imagen de Cirros personalizada desde la CLI, *cli-image*.

- –key-name. Par de claves usado, *cli-keypair*.
- –security-groups. Grupo de seguridad que definirá el tráfico permitido para esta instancia, *cli-secgroup*.
- –nic. Con este parámetro indicamos que el siguiente, net-name, vendrá definido por el nombre con el que se creo, en lugar de su ID, lo cuál lo hace más fácil de gestionar. Indicamos que la instancia se cree dentro de la subred privada creada desde la linea de comandos, *cli-private-subnet*.
- cli-instance-1. Por último, escribimos el nombre que queremos que tenga nuestra instancia.

Una forma sencilla de saber si se ha creado con éxito la instancia sería escribir:

```
stack@wimunet4:~/devstack$ openstack server list
```

Y buscar si la nuestra está activa. Existe otro modo de hacerlo que además nos servirá para ver de que tipo es la imagen creada y si está accesible de un modo sencillo, cuya salida tenemos en la Fig.7.46 y se corresponde con el *login prompt* de la instancia.

```
stack@wimunet4:~/devstack$ nova console-log cli-instance-1
```

7.4.11. Creación de volúmenes de almacenamiento para las instancias

Como hemos indicado anteriormente, el almacenamiento en OpenStack de las instancias es efímero por defecto. Para crear un volumen de storage en caso de necesitarlo escribiremos:

```
stack@wimunet4:~/devstack$ openstack volume create --image cli-cirros
    --size 8 --availability-zone nova cli-volume
```

De este modo crearemos un volumen llamado *cli-volume* donde –image hace referencia al origen del volumen, es decir, a la imagen con la que se pretende que esté relacionado dicho volumen. Con –size, indicamos el tamaño del mismo, que será de 8 GiB y el parámetro –availability-zone nova, nos

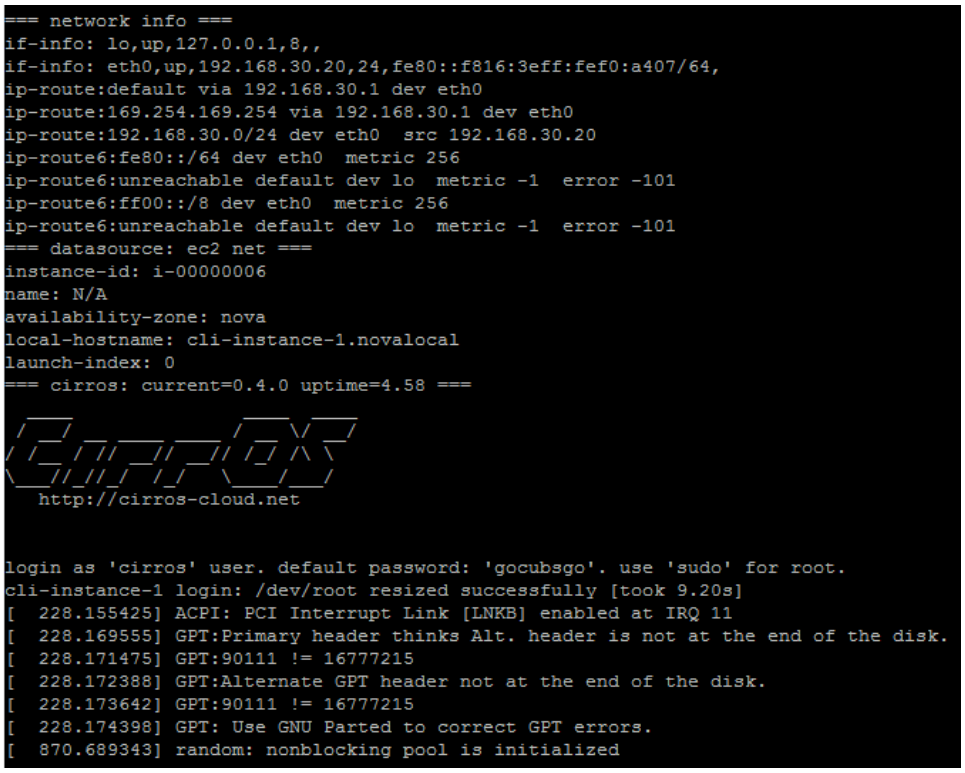


Figura 7.46: Login prompt de la instancia *cli-instance-1*.

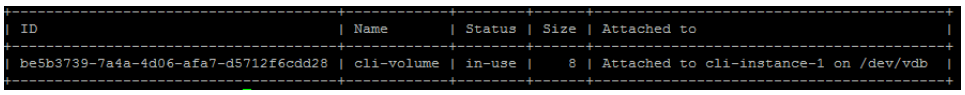


Figura 7.47: Salida por CLI del listado de volúmenes disponible.

permite introducir el volumen como parte de la gestión de instancias de Nova.

Una vez creado el volumen podemos asociarlo valiéndonos de Nova con el siguiente comando:

```
stack@wimunet4:~/devstack$ openstack server add volume openstack
    server add volume cli-instance-1 cli-volume --device /dev/vdb
```

Así indicamos que el volumen creado se asocie a la instancia *cli-instance* como una partición en /dev/vdb.

Para asegurarnos de que se ha creado y asociado con éxito:

```
stack@wimunet4:~/devstack$ openstack volume list
```

7.5. Orquestación de VNFs con Heat

Estando en posesión de los conocimientos necesarios para comprender las tecnologías de las que OpenStack esta rodeada, como nuestra infraestructura está creada y la forma que tenemos de interactuar con OpenStack, en lo que resta de este capítulo vamos a proceder a mostrar de forma más práctica la creación de VNFs.

En primer lugar veremos la HOT creada para el auto-escalado de instancias para a continuación mostrar la forma en que monitorizaremos las instancias creadas para que en caso de caída de una interfaz de red virtual de las mismas o cualquier otra incidencia que derive en la pérdida de conexión a la VM creada, seamos capaces mediante un *script* en Python tanto de realizar dicha monitorización como de recuperar la máquina cuya conexión se pierda.

Para esta sección y la última de este capítulo 7.6 se ha creado un repositorio en GitHub [64] donde podremos ver las *templates* creadas que usaremos para la creación de los escenarios restantes para crear VNFs.

7.5.1. Auto-escalado de instancias

En este apartado vamos a describir la HOT para el auto-escalado o *autoscaling* de instancias que usaremos en la siguiente sección:

```
heat_template_version: 2015-04-30

description: >
  This is a template that illustrates the basic features
  of OS::Heat::AutoScalingGroup when the scaled resource is an
  OS::Nova::Server.
parameters:
  key_name:
    type: string
    description: Name of an existing key pair to use for the instances
    default: prueba-keypair
    constraints:
      - custom_constraint: nova.keypair
        description: Must name a public key (pair) known to Nova
  flavor:
    type: string
    description: Flavor for the instances to be created
    default: m1.little
    constraints:
      - custom_constraint: nova.flavor
        description: Must be a flavor known to Nova
  image:
    type: string
    description: >
      Name or ID of the image to use for the instances.
    default: prueba-cirros
    constraints:
      - custom_constraint: glance.image
        description: Must identify an image known to Glance
  network:
    type: string
    description: The network for the VM
    default: net_mgmt

resources:
  asg:
    type: OS::Heat::AutoScalingGroup
    properties:
      resource:
        type: OS::Nova::Server
        properties:
            key_name: { get_param: key_name }
            image: { get_param: image }
            flavor: { get_param: flavor }
            networks: [{network: {get_param: network} }]
      min_size: 1
      desired_capacity: 3
      max_size: 5

  scale_up_policy:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: {get_resource: asg}
      cooldown: 60
```

```
      scaling_adjustment: 1

  scale_down_policy:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: {get_resource: asg}
      cooldown: 60
      scaling_adjustment: '-1'

outputs:
  scale_up_url:
    description: >
      This URL is the webhook to scale up the group.  You can invoke
      the scale-up operation by doing an HTTP POST to this URL; no
      body nor extra headers are needed.
    value: {get_attr: [scale_up_policy, alarm_url]}
  scale_dn_url:
    description: >
      This URL is the webhook to scale down the group.  You can invoke
      the scale-down operation by doing an HTTP POST to this URL; no
      body nor extra headers are needed.
    value: {get_attr: [scale_down_policy, alarm_url]}
  asg_size:
    description: >
      This is the current size of the auto scaling group.
    value: {get_attr: [asg, current_size]}
  server_list:
    description: >
      This is a list of server names that are part of the group.
    value: {get_attr: [asg, outputs_list, name]}
  networks:
    description: >
      This is a map of server resources and their networks.
    value: {get_attr: [asg, outputs, networks]}
  server_ips:
    description: >
      This is a list of first ip addresses of the servers in the group
      for a specified network.
value: {get_attr: [asg, outputs_list, networks, {get_param: network},
    0]}
```

En esta plantilla, estamos definiendo como parámetros poder defecto el *keypair*, *flavor*, la imagen y la red que vamos a usar para crear nuestras VM.

Se crea un recurso de tipo OS::Heat::AutoSalingGroup que utilizamos para crear un número determinado de instancias, en concreto 3, las que se indican en el la propiedad "desired_capacity". Además creamos dos políticas, OS::Heat::ScalingPolicy, que podremos usar si así lo queremos para aumentar o disminuir el número de VM del grupo en una cada vez que lancemos la url que se devuelve como salida al crear el stack, "scale_up_policy" para aumentar en una y "scale_down_policy" para disminuirla. Como nota a tener en cuenta, la red a la que se conectan por defecto se llamará "net_mgmt", cuyo origen veremos en la siguiente sección.

7.5.2. Script para la monitorización y recuperación de VNFs

En este apartado vamos a trabajar con un usuario y un proyecto creado para tal fin que hemos llamado "prueba-user" y "prueba-project" respectivamente.

Para monitorizar la conectividad a las instancias o VMs mediante *ping* y comprobar si estas están o no activas además de lanzar la propia pila que las creará, vamos a crear un script ejecutable en Python con el nombre "respawn.sh".

Este script creará en primer lugar las instancias introduciendo el siguiente comando una vez autenticados como usuario "prueba-user":

```
stack@wimunet4:~/devstack$ source openrc prueba-user prueba-project
stack@wimunet4:~/devstack$ ./respawn.sh https://raw.githubusercontent.
    com/ourtelcloudthings/openstack-tfg-poc/master/Heat%20templates/
    autoscaling-VNFs-VMs.yaml autoscaling
```

Donde estamos indicando el script a ejecutar. El primer parámetro que recibe tendrá que ser el directorio o en este caso la url de nuestra repositorio de GitHub donde está alojada la HOT y como segundo parámetro el nombre que le daremos a la pila.

Una vez lanzado, el script creará un fichero llamado "server_ips.txt" que no es más que la salida del comando:

```
stack@wimunet4:~/devstack$ openstack stack output show -f yaml -c
    output_value autoscaling server_ips > server_ips.txt
```

En este comando le estamos indicando que nos muestra en formato "yaml", la columna "output_value" de la pila llamada "autoscaling" en cuya *template* definimos una salida llamada "servers_ips" que devuelve las IPs de las VM creadas. El script se encarga de modificar este fichero de texto para que las IPs puedan ser utilizadas al realizar pruebas de conectividad.

Una vez creada la pila, irá haciendo ping cada cierto tiempo a cada una de las IPs de las VM creadas. En caso de detectar alguna en la que hubiésemos perdido la conexión, lanzará un *update* de la plantilla del apartado anterior que automáticamente recuperara aquella instancia no alcanzable, dotando así a la configuración de mecanismos de auto-recuperación también conocidos como *autohealing* o *respawn*. El script explicado es el siguiente:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import time
import os
import subprocess
import sys
```

```
# Stack will be the  HOT passed as first argument when running the
    script
stack = sys.argv[1]
# Stack name will be the second argument passed when running the
    script
stack_name = sys.argv[2]
# Launch the stack
os.system("openstack stack create -t " + stack + " " +  stack_name)
# Wait until stack is running.
time.sleep(30)
# Create a file to archive the serves instances ips
os.system("openstack stack output show -f yaml -c output_value " +
    stack_name + "  server_ips > server_ips.txt")
# Waitl until file will created.
time.sleep(20)
print("Stack is ready to use")

while(1):
        subprocess.call(['sed','-i','/.*output_value:.*/d','server_ips
            .txt'])
        server_ip = open("server_ips.txt", "r")
        for linea in server_ip.readlines():
                ip = linea.replace("-","")
                response = os.system("ping -c 1 " + ip)
                if response == 0:
                        print("ACK")
                else:
                        print("NACK")
                        os.system("openstack stack update -t " + stack
                             + " " + stack_name)
                        time.sleep(60)
                        os.system("openstack stack output show -f yaml
                             -c output_value " + stack_name + "
                            server_ips > server_ips.txt")
                        subprocess.call(['sed','-i','/.*output_value
                            :.*/d','server_ips.txt'])
                        time.sleep(30)
                        print("Stack active again")
                time.sleep(30)
```

7.6. Orquestación de VNFs con Tacker

En esta sección nos apoyaremos en la Fig.7.48 que hemos creado para ilustrar el funcionamiento de Tacker de un modo general, pero adaptado a las funcionalidades y recursos que vamos a crear en estos escenarios.

En primer lugar veremos la sencillez con la que podemos registrar nuestra IaaS como una VIM en Tacker. Después, de un modo similar a como hicimos en la sección anterior, veremos como podemos escalar VNFs que serán instancias de VM con Tacker.

A continuación, crearemos una instancia de una VM que será capaz de recuperarse por si misma ante la caída de un enlace o la perdida de conexión sin necesidad de hacer ningún tipo de *scripting*, directamente con las

funcionalidades de *monitoring* que incluye Tacker. Además, veremos también como podríamos incorporar a nuestra VM configuración adicional que pudiera ser necesaria mediante inyección de datos.

Por último, crearemos una VNF que hará de router y firewall. Todas las VNF que vamos a crear se reflejan dentro de la VIM en la Fig.7.48, para lo que antes tendremos que registrar las VNFD en el catálogo para que estén disponibles a la hora de crear VNFs.

Para el resto de apartado que quedan vamos a trabajar sobre el proyecto y con el usuario con el rol de administrador. Para autenticarnos como tal:

```
stack@wimunet4:~/devstack$ source openrc admin admin
```

Para crear VNFD y VNF usaremos los siguientes comandos respectivamente:

- tacker vnfd-create –vnfd-file <nombre de la plantilla><nombre para el descriptor>
- tacker vnf-create –vnfd-name <nombre del descriptor><nombre de la VNF>–vim-name VIM0

Podríamos haber puesto por defecto que la única VIM que tenemos fuese la de referencia, pero de esta forma ilustramos lo sencillo que sería crear VNFs en una u otra VIM sólo indicando su nombre.

Hay otra parte importante en la creación de los escenarios que aún no hemos comentado y que ya utilizamos en el capítulo anterior. Cuando añadimos al inicio de la instalación en nuestro archivo de configuración "local.conf" las líneas necesarias para instalar Tacker, se instalan por defecto tres redes.

Como sabemos, nuestra instalación está basada en un todo en uno que da cabida al conjunto de proyectos que necesitamos. Idealmente, la herramienta encarga de gestionar las distintas VIM estaría en una ubicación diferente a estas, de forma que se pudiesen controlar de forma centralizada. Al optar por la instalación *all-in-one* estamos instalando este servicio en el mismos servidor. En dicha instalación se crearan las redes que podemos ver en la Fig.7.49. Tenemos tres:

- net_mgmt.
- net0.
- net1.

La naturaleza de creación de estas redes es la gestión de las VNFs que se añadirán en una u otra atendiendo a los criterios de diseño. Todas se

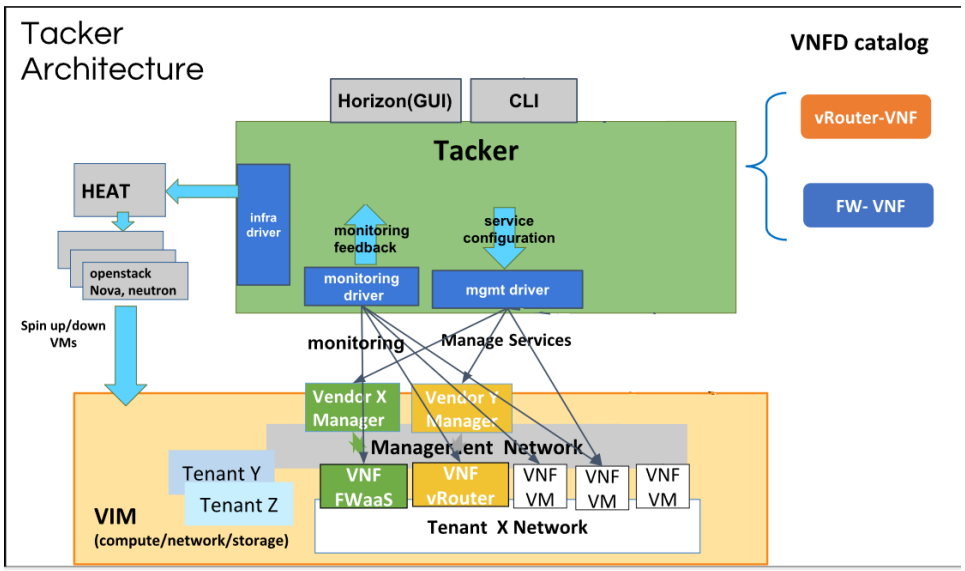


Figura 7.48: Arquitectura de Tacker para el escenario propuesto.

suelen conectar a la red net_mgmt para que cualquier VNF que tengamos este accesible para ser gestionada como usuarios administradores desde esta red. Las otras dos redes, se utilizarán como redes de servicio en las que se explotará la funcionalidad que aporte la VNF creada.

7.6.1. Registrando una VIM

Para registrar nuestra infraestructura como una VIM y poder así gestionarla desde Tacker, necesitamos en primer lugar, si decidimos hacerlo desde la CLI, crear un archivo de configuración que hemos llamado "vim_config.yaml":

```
auth_url: 'http://10.5.1.201/identity'
username: 'admin'
password: 'temporary'
project_name: 'admin'
project_domain_name: 'Default'
user_domain_name: 'Default'
```

Este archivo con tiene la información necesaria para crear un registro en Tacker de la infraestructura. El parámetro "auth_url" es la url donde el servicio de identidad de Keystone está operativo y será necesario para poder registrarnos en su base de datos y acceder al resto de servicios que oferte nuestra cloud para crear VNFs. A continuación se indica el nombre de usuario y la contraseña del tenant que vaya a gestionar la VIM, que en nuestro caso será el administrador, el nombre del proyecto, y el dominio de

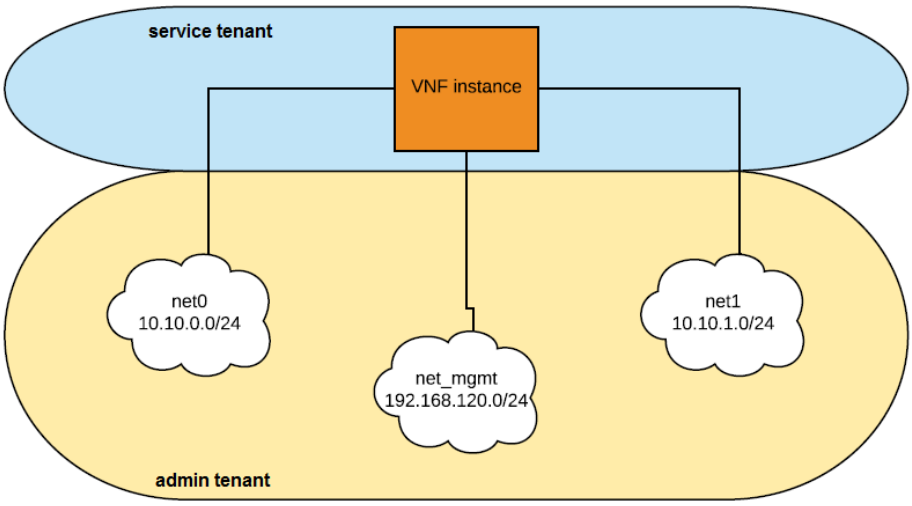


Figura 7.49: Redes creadas al añadir Tacker en OpenStack.

usuario y del proyecto que se dejaremos el que tenga por defecto.

Una vez creado este archivo podremos registrar ya nuestra VIM desde la CLI con el siguiente comando:

```
stack@wimunet4:~/devstack$ openstack vim register --config-file
    vim_config.yaml --description 'vim Regione One admin Project'
    VIM0
```

Donde se pasa como parámetro el archivo de configuración explicado, una descripción del proyecto, y el nombre que queremos para la VIM, que en nuestra caso será "VIM0". En este punto tendríamos ya el equivalente en la Fig.7.48 a la VIM que abarca todos los componentes del cuadro inferior en amarillo.

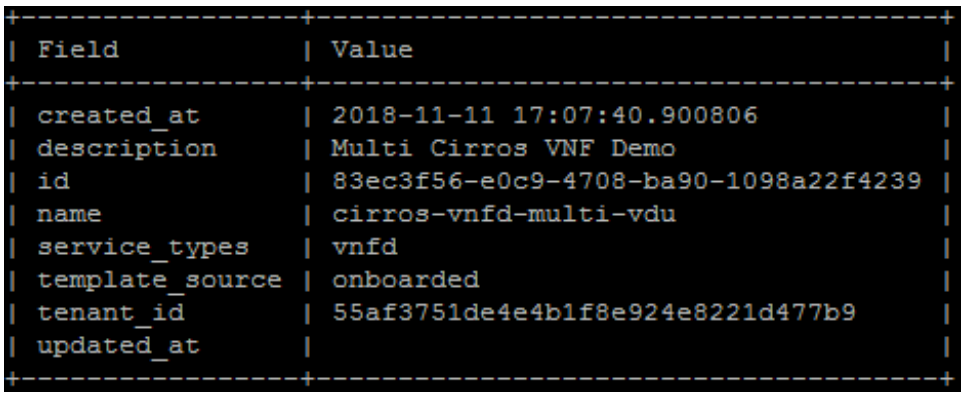
7.6.2. Autoescalado de instancias

En primer lugar tendremos que crear el descriptor VNFD mediante una plantilla de TOSCA tipo YAML en la que especificaremos los recursos a crear y que formarán nuestra VNF:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Multi Cirros VNF Demo
```

```
metadata:
  template_name: tosca-multi-vnf-cirros

topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
        image: admin-cirros
        flavor: m1.little
        availability_zone: nova
        mgmt_driver: noop
        config: |
          param0: key1
          param1: key2
    CP11:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1

    CP12:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL2
        - virtualBinding:
            node: VDU1

    CP13:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL3
        - virtualBinding:
            node: VDU1

    VDU2:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
        image: admin-cirros
        flavor: m1.little
        availability_zone: nova
        mgmt_driver: noop
        config: |
          param0: key1
          param1: key2
    CP21:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
      requirements:
        - virtualLink:
```

```
          node: VL1
      - virtualBinding:
          node: VDU2

  CP22:
    type: tosca.nodes.nfv.CP.Tacker
    requirements:
      - virtualLink:
          node: VL2
      - virtualBinding:
          node: VDU2

  CP23:
    type: tosca.nodes.nfv.CP.Tacker
    requirements:
      - virtualLink:
          node: VL3
      - virtualBinding:
          node: VDU2

  VDU3:
    type: tosca.nodes.nfv.VDU.Tacker
    properties:
      image: admin-cirros
      flavor: m1.little
      availability_zone: nova
      mgmt_driver: noop
      config: |
        param0: key1
        param1: key2
  CP31:
    type: tosca.nodes.nfv.CP.Tacker
    properties:
      management: true
    requirements:
      - virtualLink:
          node: VL1
      - virtualBinding:
          node: VDU3

  CP32:
    type: tosca.nodes.nfv.CP.Tacker
    requirements:
      - virtualLink:
          node: VL2
      - virtualBinding:
          node: VDU3

  CP33:
    type: tosca.nodes.nfv.CP.Tacker
    requirements:
      - virtualLink:
          node: VL3
      - virtualBinding:
          node: VDU3

  VL1:
    type: tosca.nodes.nfv.VL
    properties:
      network_name: net_mgmt
      vendor: Tacker
```



Field	Value
created_at	2018-11-11 17:07:40.900806
description	Multi Cirros VNF Demo
id	83ec3f56-e0c9-4708-ba90-1098a22f4239
name	cirros-vnfd-multi-vdu
service_types	vnfd
template_source	onboarded
tenant_id	55af3751de4e4b1f8e924e8221d477b9
updated_at	

Figura 7.50: Creación de un VNFD multi VDU.

```
    VL2:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net0
        vendor: Tacker

    VL3:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net1
        vendor: Tacker
```

Guardamos la *template* con el nombre "tosca-cirros-vnfd-multi-vdu.yaml" y le damos permisos de ejecución. Una vez realicemos este paso registraremos en Tacker el descriptor del siguiente modo:

```
stack@wimunet4:~/devstack$ chmod +x tosca-cirros-vnfd-multi-vdu.yaml
stack@wimunet4:~/devstack$ tacker vnfd-create --vnfd-file tosca-cirros
    -vnfd-multi-vdu.yaml cirros-vnfd-multi-vdu
```

Como vemos, únicamente tenemos que usar el comando Tacker establecido para la creación de un VNFD y pasarle como parámetro la *template* y el nombre que le queramos dar, en este caso "cirros-vnfd-multi-vdu". Cuando la creemos con éxito se nos mostrará por consola la salida dela Fig.7.50.

En este punto pasaremos a crear nuestra VNF a partir del descriptor cerrado desde la CLI:

```
tacker vnf-create --vnfd-name cirros-vnfd-multi-vdu cirros-vnf-multi-
    vdu --vim-name VIM0
```

Cuya salida tenemos en la Fig.7.51. Cuando se crea esta VNF, podemos ver desde la topología de red dentro de Horizon, como las distintas VMs creadas se conectan tanto a la rede gestión, como a la net0 y net1 (Fig.7.52).

Field	Value
created_at	2018-11-11 17:13:20.091880
description	Multi Cirros VNF Demo
error_reason	
id	46b07f38-8592-4de3-a7cf-b58487315a44
instance_id	55e342f5-7193-4eae-bece-46ea655d3a87
mgmt_url	
name	cirros-vnfd-multi-vdu
placement_attr	{"vim_name": "VIM0"}
status	PENDING_CREATE
tenant_id	55af3751de4e4b1f8e924e8221d477b9
updated_at	
vim_id	34faaa1a-cbcc-4fd9-83c1-fcb9244af4a2
vnfd_id	83ec3f56-e0c9-4708-ba90-1098a22f4239

Figura 7.51: Creación de un VNF multi VDU.

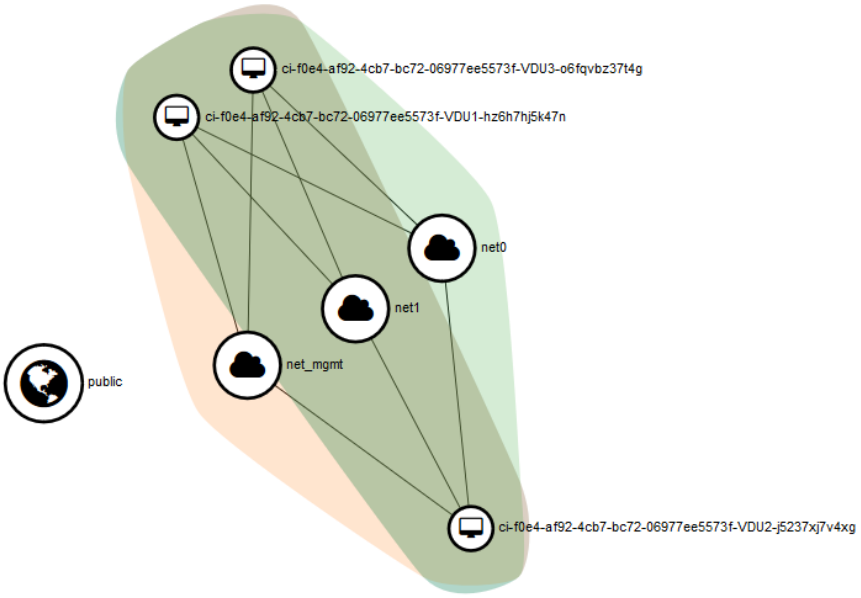


Figura 7.52: Topología de red mostrada des Horizon tras la creación de la VNF.

7.6.3. Recuperación automática de una instancia

La siguiente VNF que vamos a crear consistirá en una VM a la que introduciremos un fichero con datos del propio host para ilustrar como desde Tacker podríamos modificar la imagen, que además tendrá capacidad de *auto-healing* o recuperación automática en caso de perder la conectividad con ella.

Los pasos de la creación de la VNFD y de la VNF son idénticos a los del apartado anterior, con la salvedad del nombre que le demos al archivo de configuración, que en este caso será "tosca-cirros-vnfd-user-data-configure-monitor.yaml", a la VNFD, que nombraremos como "cirros-vnfd-user-data-configure-monitor" y a la VNF que llamaremos "cirros-vnf-user-data-configure-monitor". Por tanto nos centraremos en la *template* ya que representa la parte cambiante con respecto a los pasos a realizar de cada creación de una VNF.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Cirros Monitor Demo User Data

metadata:
  template_name: tosca-cirros-vnfd-userdata-configure-monitor

topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
        image: admin-cirros
        flavor: m1.little
        user_data_format: RAW
        user_data: |
          #!/bin/sh
          i=$(ifconfig eth0|grep 'inet addr'|awk -F: '{print $2}'| awk
               '{print $1}')
          w=$(hostname)
          y=$i"  "$w
          sudo sed -i -e "\$a$y" /etc/hosts
          sudo sed -i -e "s/cirros/$w/g" /etc/hosts
        availability_zone: nova
        mgmt_driver: noop
        config: |
          param0: key1
          param1: key2
        monitoring_policy:
          name: ping
          parameters:
            monitoring_delay: 45
            count: 3
            interval: 1
            timeout: 2
          actions:
            failure: respawn

    CP1:
```

```
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1

    CP2:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL2
        - virtualBinding:
            node: VDU1

    CP3:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL3
        - virtualBinding:
            node: VDU1

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net_mgmt
        vendor: Tacker

    VL2:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net0
        vendor: Tacker

    VL3:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net1
        vendor: Tacker
```

Como vemos en la definición de la plantilla, en el aparado "user_data" estamos creando un script en bash mediante el cual crearemos un archivo en el que registra las IP y los *hostnames* de la VM.

El otro punto fuerte de esta plantilla es el campo "monitoring_policy", en el que indicamos que Tacker monitorice mediante ping la VNF creada y ante un fallo ejecute la acción de *respawn* o auto-recuperación.

Una vez creada, para asegurarnos que se ha construido con éxito, vamos esta vez a ver el aspecto que tiene desde Horizon. Dentro del *dashboard* iremos al apartado "NFV->VNF Management->VNF Manager" para acceder

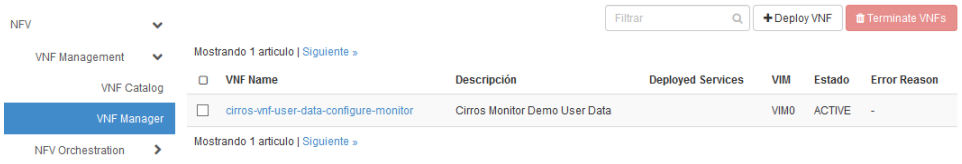


Figura 7.53: VNF Manager Dashboard.

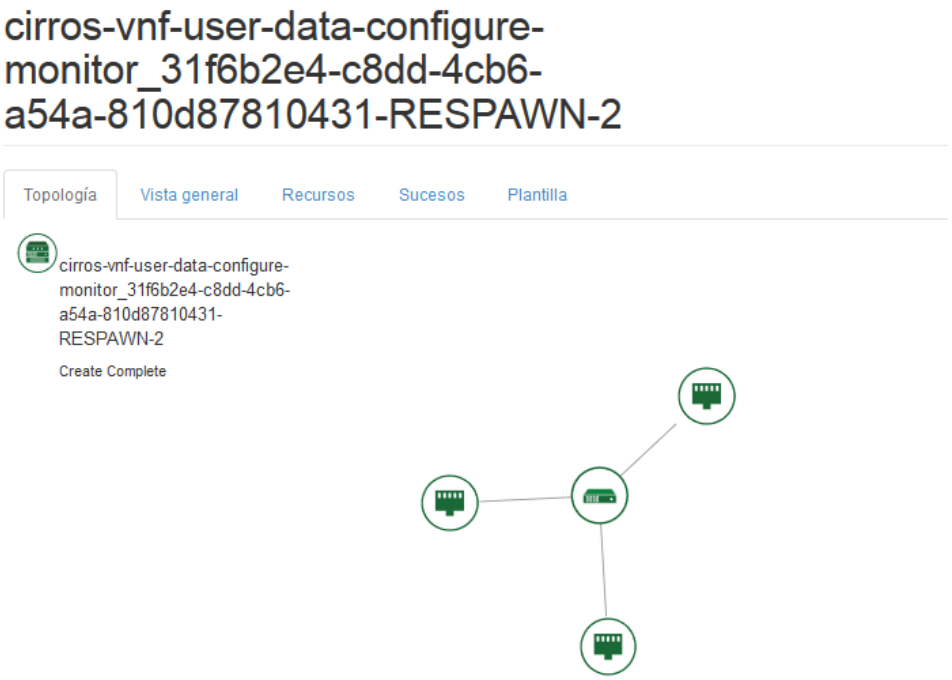


Figura 7.54: Pila creada por Heat al construir nuestra VNF en Tacker.

al VNFM. Al ir, veremos el panel de la Fig.7.53 podremos ver como nuestra VNF está activa y en VIM0.

Como explicamos en en capítulo introductorio de Tacker 5.7, cuando creamos una VNF, Heat se encarga de traducirla a su lenguaje para interactuar con la VIM creada y todos los elementos que impliquen de nuestra IaaS, tal como se esquematiza en la parte superior izquierda de la Fig.7.48. Por ello, si después de crear nuestra VNF vamos al apartado dentro de Horizon "Proyecto->Orquestación->Pilas" veremos que hay una pila creada con un nombre similar al que nosotros le dimos a la VNF y una serie de caracteres. Pulsando en ella podremos ver los recursos que Heat a creado para en nuestra VIM para que la VNF funcione como esperamos (Fig.7.54). Lo mismo ocurre para cualquier VNF que creemos.

7.6.4. VNF Router

Como última VNF vamos a crear un router para ver como podemos modificar las características de este. Hasta ahora, cuando creábamos un router en OpenStack, para nosotros era totalmente trasparente en el sentido de que hacía las funciones de enrutamiento pero no sabíamos como ni podíamos controlar esta parte. Además, en cuanto al tráfico permitido o no, este se tenía que gestionar a través de los grupos de seguridad que son un tanto rígidos en comparación con las opciones que presenta un router.

OpenWRT es un firmware basado en una distribución de Linux empotrada en dispositivos tales como routers personales. Tiene soporte para fabricante como Netgear, D-Link o Asus y se pueden implementar sobre el servicios como QoS, VPN o firewall [65]. De este último veremos una implementación.

Para poder usar esta imagen lo primero es disponer de ella en Glance al igual que en el resto de casos. OpenStack nos ofrece sugiere un repositorio para descargarla:

```
stack@wimunet4:~/devstack$ wget https://anda.ssu.ac.kr/~openwrt/
    openwrt-x86-kvm_guest-combined-ext4.img.gz
```

Una vez la ubiquemos en el directorio deseado vamos a descomprimirla y a crear una imagen como ya hicimos de varios formas con la imagen de CirrOS:

```
stack@wimunet4:~/devstack$ gunzip openwrt-x86-kvm_guest-combined-ext4.
    img.gz
stack@wimunet4:~/devstack$ openstack image create OpenWRT --disk-
    format qcow2 \
                               --container-format bare \
                               --file /opt/stack/devstack/tacker-test/
                                   OpenWRT/openwrt-x86-kvm_guest-
                                   combined-ext4.img \
                               --public
```

Para ver si tenemos ya la imagen disponible podemos listar las imágenes que tenemos en Glance:

```
stack@wimunet4:~/devstack$ openstack image list
```

Y veremos (Fig.7.55) como ya aparece la imagen OpenWRT. Una vez disponible estamos en condiciones de al igual modo que en los apartados anteriores registrarla en nuestro catálogo de VNFD y crear una VNF a partir de este.

```
# Definir VNFD
```

```
stack@wimunet04:~/devstack$ openstack image list
+--------------------------------------+--------------------------+--------+
| ID                                   | Name                     | Status |
+--------------------------------------+--------------------------+--------+
| 3cbba24c-bff7-40d5-b3da-679b1deecfcd | OpenWRT                  | active |
| 58b10ea5-7618-4586-900f-9dab98014b1d | admin-cirros             | active |
| 84b49d86-aaee-402a-9a25-4e6d736d857b | cirros-0.3.5-x86_64-disk | active |
| 7f07e61c-54cc-4498-8e46-9218011faa39 | cirros-0.4.0-x86_64-disk | active |
| ddde886c-9b6e-40d0-813d-2e4961aee7b5 | cli-cirros               | active |
| 268ff4f6-47ba-4ecc-8aa0-7284d728a3b0 | horizon-cirros           | active |
| 537f54af-5145-4f65-822f-e46abd394584 | metadatos-cirros-prueba  | active |
| c88d2908-bebe-4434-b1cd-38b3668daf9b | prueba-cirros            | active |
+--------------------------------------+--------------------------+--------+
```

Figura 7.55: Listado de imagen de Glance.

```
stack@wimunet4:~/devstack$ tacker vnfd-create --vnfd-file tosca-
    openwrt-vnfd.yaml openwrt-vnfd
# Crear VNF
stack@wimunet4:~/devstack$ tacker vnf-create --vnfd-name openwrt-vnfd
    openwrt-vnf --vim-name VIM0
```

La TOSCA *template* a partir de la cual se crea esta VNFD es la que sigue:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: OpenWRT with services

metadata:
  template_name: OpenWRT

topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 512 MB
            disk_size: 1 GB
      properties:
        image: OpenWRT
        config:
          firewall: |
            package firewall

            config defaults
                option syn_flood '1'
                option input 'ACCEPT'
                option output 'ACCEPT'
                option forward 'REJECT'

            config zone
                option name 'lan'
                list network 'lan'
                option input 'ACCEPT'
```

```
                option output 'ACCEPT'
                option forward 'ACCEPT'

            config zone
                option name 'wan'
                list network 'wan'
                list network 'wan6'
                option input 'REJECT'
                option output 'ACCEPT'
                option forward 'REJECT'
                option masq '1'
                option mtu_fix '1'

            config forwarding
                option src 'lan'
                option dest 'wan'

            config rule
                option name 'Allow-DHCP-Renew'
                option src 'wan'
                option proto 'udp'
                option dest_port '68'
                option target 'ACCEPT'
                option family 'ipv4'

            config rule
                option name 'Allow-Ping'
                option src 'wan'
                option proto 'icmp'
                option icmp_type 'echo-request'
                option family 'ipv4'
                option target 'ACCEPT'
        mgmt_driver: openwrt
        monitoring_policy:
          name: ping
          parameters:
            count: 3
            interval: 10
          actions:
            failure: respawn

    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net_mgmt
        vendor: Tacker
```

Una vez creada, para comprobar que tenemos acceso a ella vamos accediendo vía ssh a ella a través de la red de gestión y ver que tenemos conexión

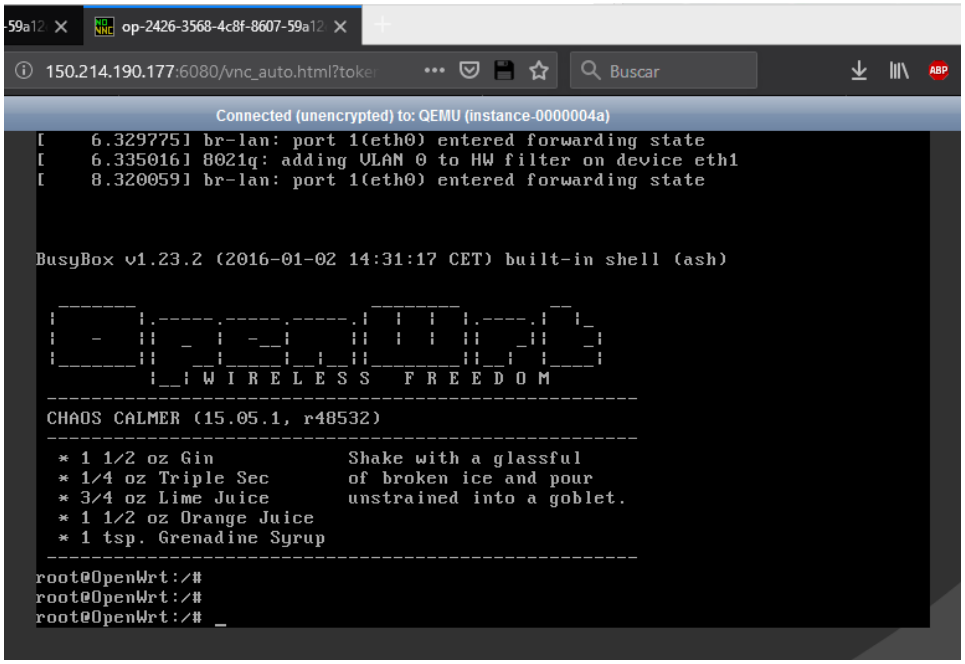


Figura 7.56: Acceso a la consola de OpenWRT.

y que se ha creado con éxito (Fig.7.56).

La gestión de Tacker tiene una funcionalidad muy interesante. Para una VNF creada, en nuestra caso el router OpenWRT, pero podrá ser cualquier otra, podemos hacer un *update* de sus parámetros en tiempo real de forma que cambiemos la configuración de la VNF en algún sentido.

Esta es la forma en que vamos a agregar nuevas funcionalidades de firewall a nuestro router. Dentro del mismo podemos ver algunas de las reglas que tiene actualmente definidas algunas de las cuales mostramos en la Fig.7.57. Una vez hayamos actualizado dichas reglas mediante:

```
stack@wimunet4:~/devstack$ tacker vnf-update --config-file tosca-
    config-openwrt-with-firewall.yaml openwrt-vnf
```

Veremos como las reglas que ahora aparecen cambian (Fig.7.58). Para realizar dicha actualización de manera análoga a como se trabaja en el resto de creación de VNFs, hemos tenido que definir una *template* que hemos llamado “tosca-config-openwrt-with-firewall.yaml”, además hay que indicar la VNF que queremos actualizar con su nombre (“openwrt-vnf”). La descripción del archivo YAML es el que sigue:

```
vdus:
  VDU1:
```

```
    config:
      firewall: |
        package firewall
        config defaults
            option syn_flood '1'
            option input 'ACCEPT'
            option output 'ACCEPT'
            option forward 'REJECT'
        config zone
            option name 'lan'
            list network 'lan'
            option input 'ACCEPT'
            option output 'ACCEPT'
            option forward 'ACCEPT'
        config zone
            option name 'wan'
            list network 'wan'
            list network 'wan6'
            option input 'REJECT'
            option output 'ACCEPT'
            option forward 'REJECT'
            option masq '1'
            option mtu_fix '1'
        config forwarding
            option src 'lan'
            option dest 'wan'
        config rule
            option name 'Allow-DHCP-Renew'
            option src 'wan'
            option proto 'udp'
            option dest_port '68'
            option target 'ACCEPT'
            option family 'ipv4'
        config rule
            option name 'Allow-Ping'
            option src 'wan'
            option proto 'icmp'
            option icmp_type 'echo-request'
            option family 'ipv4'
            option target 'ACCEPT'
        config rule
            option name 'Allow-IGMP'
            option src 'wan'
            option proto 'igmp'
            option family 'ipv4'
            option target 'ACCEPT'
        config rule
            option name 'Allow-DHCPv6'
            option src 'wan'
            option proto 'udp'
            option src_ip 'fe80::/10'
            option src_port '547'
            option dest_ip 'fe80::/10'
            option dest_port '546'
            option family 'ipv6'
            option target 'ACCEPT'
        config rule
            option name 'Allow-MLD'
            option src 'wan'
            option proto 'icmp'
            option src_ip 'fe80::/10'
            list icmp_type '130/0'
```

```
        list icmp_type '131/0'
        list icmp_type '132/0'
        list icmp_type '143/0'
        option family 'ipv6'
        option target 'ACCEPT'
    config rule
        option name 'Allow-ICMPv6-Input'
        option src 'wan'
        option proto 'icmp'
        list icmp_type 'echo-request'
        list icmp_type 'echo-reply'
        list icmp_type 'destination-unreachable'
        list icmp_type 'packet-too-big'
        list icmp_type 'time-exceeded'
        list icmp_type 'bad-header'
        list icmp_type 'unknown-header-type'
        list icmp_type 'router-solicitation'
        list icmp_type 'neighbour-solicitation'
        list icmp_type 'router-advertisement'
        list icmp_type 'neighbour-advertisement'
        option limit '190/sec'
        option family 'ipv6'
        option target 'REJECT'
```

En el se crean una serie de restricciones en el tráfico de red en el que no vamos a entrar pues no es el objetivo, aunque el lenguaje, al igual que el resto de plantilla es intuitivo de lo que simula.

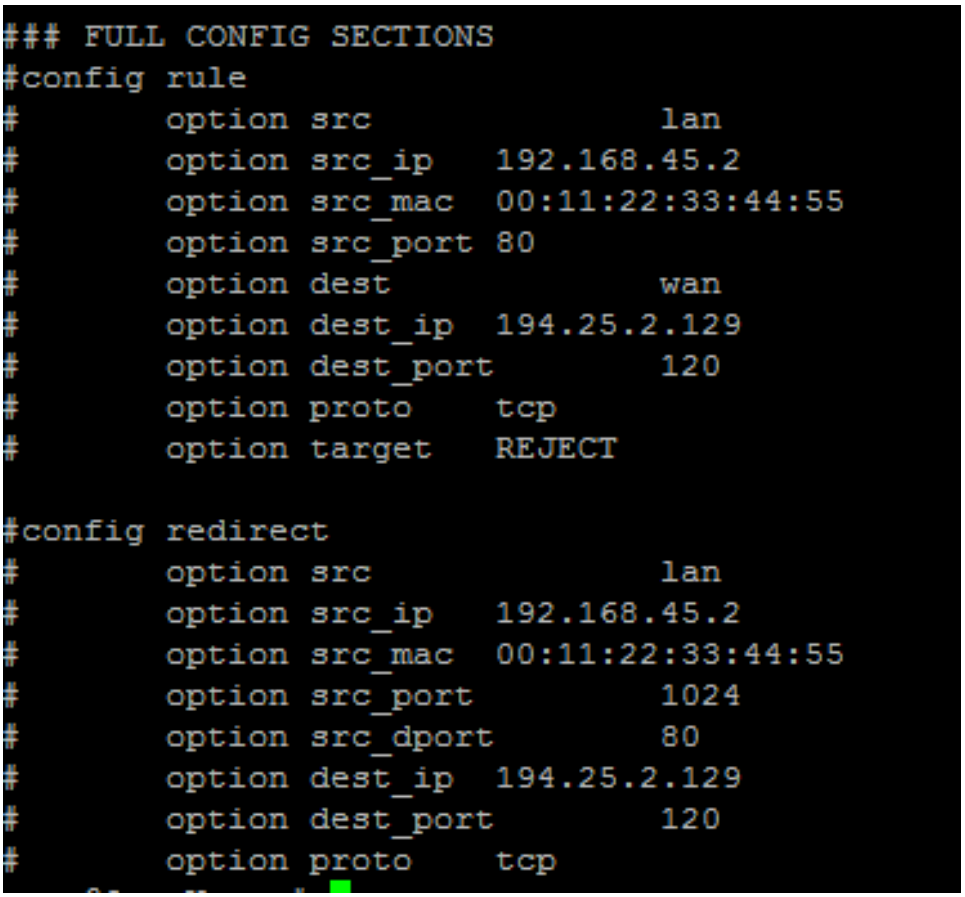


Figura 7.57: VNF con función de router antes de actualizar firewall.

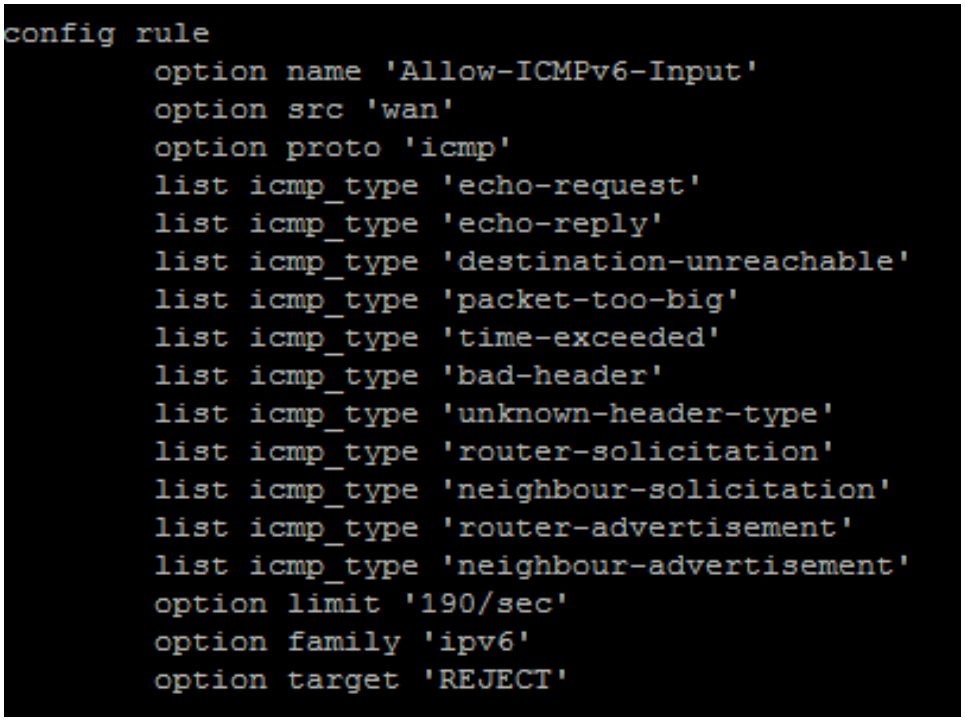


Figura 7.58: VNF con función de router y el firewall actualizado.

Capítulo 8

Pruebas y resultados

Este capítulo tratara de hacer algunas comprobaciones para asegurarnos de que los recursos y VNFs creadas en los distintos apartados del capítulo anterior funcionan como se espera.

Haremos algunas prueba de acceso para demostrar de que forma podemos acceder a las instancias creadas desde el dashboard de Horizon o las que construimos desde la CLI.

A continuación, veremos el correcto funcionamiento y despliegue del entorno de auto-escalado creado con Horizon, para por último ver como podemos acceder de forma más sencilla a través de la red de gestión que se creó al incorporar el proyecto Tacker a nuestra IaaS y para el caso concreto de la VNF creada en la sección 7.6.3, veremos si cuando se pierde la conexión de la instancia somos capaces de recuperarla automáticamente sin intervenir en absoluto y si se creó o no la configuración adicional introducida.

8.0.1. Acceso a las instancias creadas desde Horizon o CLI

Una vez que hemos conseguido levantar una instancia, tenemos varias formas de comprobar si esta es accesible y por tanto se ha creado con éxito.

Si estamos dentro de horizon dentro de "Proyecto->Compute->Instancias" podremos ver por ejemplo, la instancia creada desde horizon con el nombre "horizon-user" con una dirección IP, un flavor, un par de claves, etc. Para asegurarnos de que podemos conectarnos a ella tenemos varias opciones. La forma más sencilla es abrir una consola de la instancia desde Horizon, seleccionando "Consola" dentro del campo "Acciones" como mostramos en la Fig.8.1.

Y vemos en la Fig.8.2 como se abre una consola con la instancia creada en la que para ver que efectivamente se trata de la nombrada como "horizon-

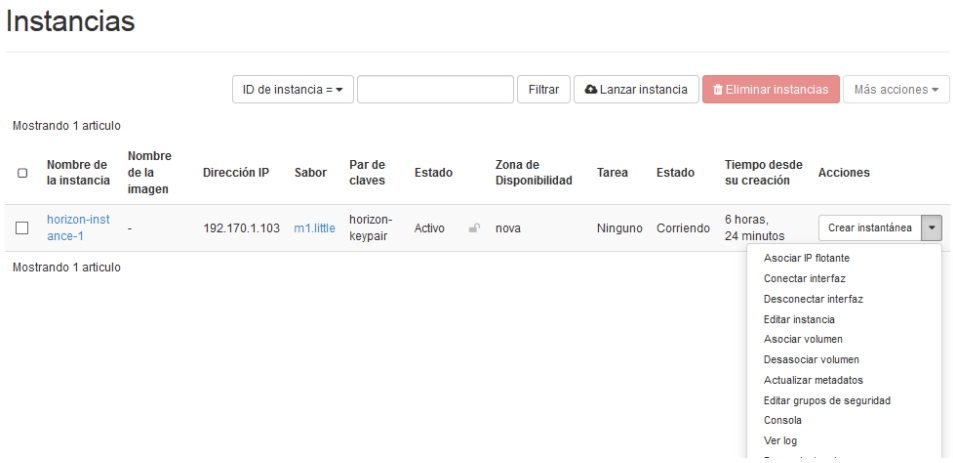


Figura 8.1: Abriendo consola de la instancia.

user" hemos puesto el comando *ip a* para ver que se trata de las instancia con IP 192.170.1.103/24.

A pesar de ser una forma fácil de acceder a la instancia no es la más útil desde el punto de vista del usuario final. Por ello, vamos ahora a ver otra opción de acceso y comprobación de que se puede alcanzar de forma externa.

```
stack@wimunet4:~/devstack$ ip netns show
```

El comando ip netns show ejecutado desde una consola de alguna máquina en la que tengamos instalado OpenStack, nos muestra los distintos *namespaces*, o espacios de nombres que se utilizan para implementar redes definidas por software.

La salida del comando podemos verla en la Fig.8.3. El espacio de nombres "qrouter" representa a los routers que hayamos creado. La utilidad de usar namespaces reside en que se crea una red completamente aislada en el nivel de Linux. Así sí escribiéramos *ip a*, podríamos ver que existen distintas interfaces de red, como por ejemplo "enp3s0", con una IP determinada, pero no veremos nada de lo que está sucediendo dentro del espacio de nombres de "qrouter".

Por tanto, para poder hacer ping o conectarnos a la instancia creada, necesitamos ejecutar comandos en el espacio de nombres de "qrouter". Para ello escribimos:

```
stack@wimunet4:~/devstack$ sudo ip netns exec qrouter-45654247-1b47
    -457e-a2de-b9aae46595dc ip a
```

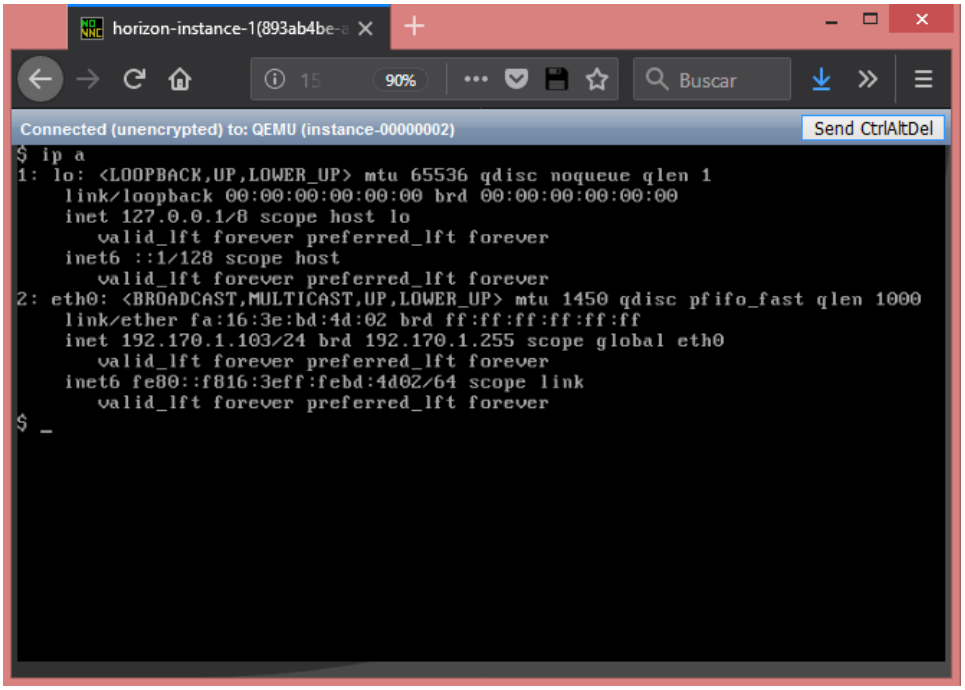


Figura 8.2: Consola de la instancia de cirros horizon-instance-1.

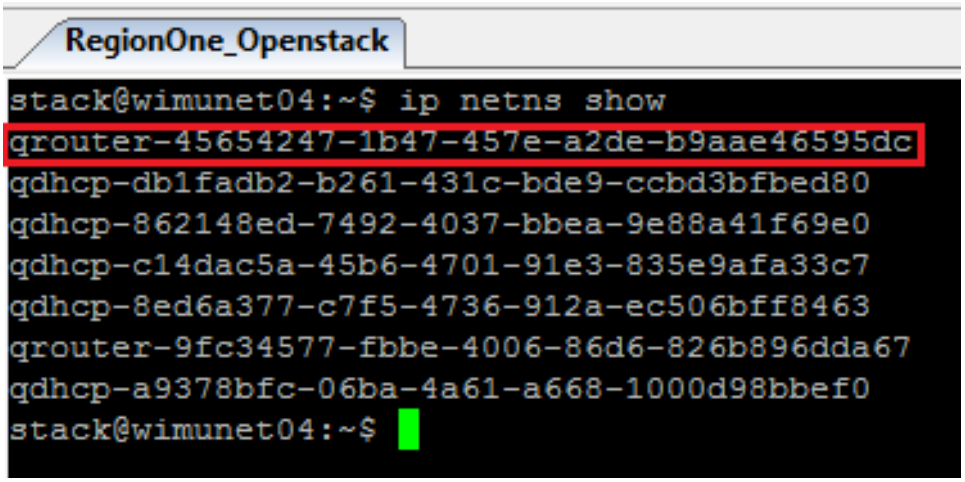


Figura 8.3: Openstack namespaces.

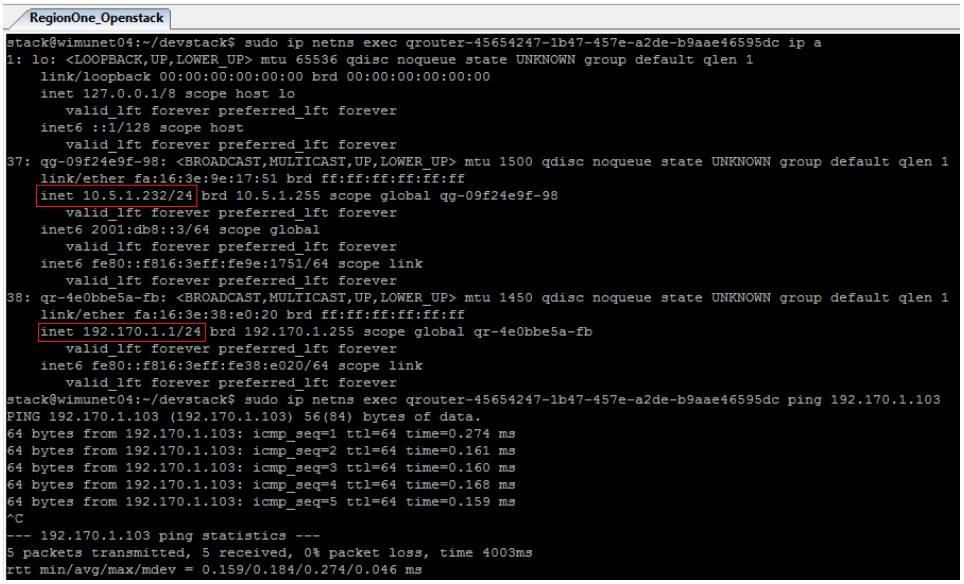


Figura 8.4: Interfaz y ping a la instancia mediante namespaces.

Con este comando, estamos ejecutando *ip a* dentro del espacio de nombres del router que conecta nuestra instancia con la red externa, obteniendo como salida la mostrada en la Fig.8.4 en la que ahora sí, podemos ver la red externa (10.5.1.232/24) e interna (192.170.1.1/24) de la instancia creada. Como se aprecia en la figura también podremos hacer ping con utilizando el espacio de nombres del router y la IP de la instancia creada:

```
stack@wimunet4:~/devstack$ sudo ip netns exec qrouter-45654247-1b47
    -457e-a2de-b9aae46595dc ping 192.170.1.103
```

Por tanto, ya sabemos que podemos hacer ping a la instancia, pero también deseamos verificar si tenemos acceso a la misma. Para ello, vamos a usar de nuevo *ip netns exec* pero esta vez ejecutando el comando ssh para conectarnos a la instancia usando la clave privada asociada a la instancia:

```
stack@wimunet4:~/devstack/.ssh$ sudo ip netns exec qrouter-45654247-1
    b47-457e-a2de-b9aae46595dc ssh -i horizon-keypair cirros@192
    .170.1.103
```

En este comando estamos usando de nuevo el espacio de nombres de "qrouter" y ejecutamos el comando *ssh -i horizon-keypair* en dicho espacio donde como sabemos *horizon-keypair* es el arichivo de clave privada que contiene las credenciales para iniciar sesión en la instancia. Accedemos como usuario *cirros* a la IP 192.170.1.103, la asignada a la instancia *horizon-instance-1*. Como resultado, en la Fig.8.7 podemos ver que nos he-

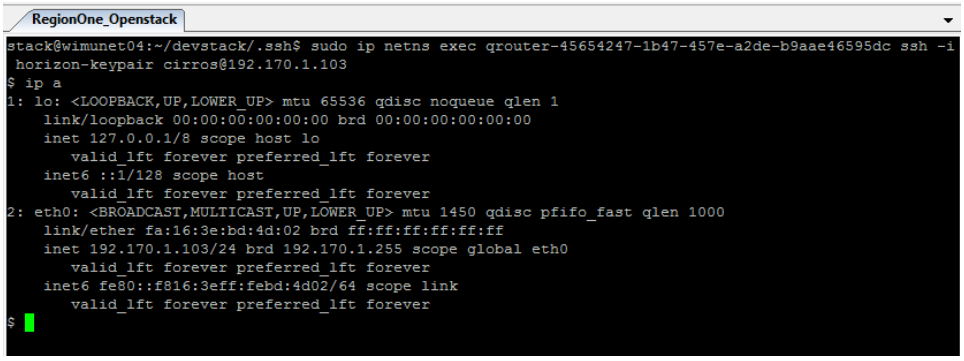


Figura 8.5: Acceso vía ssh usando namespaces.

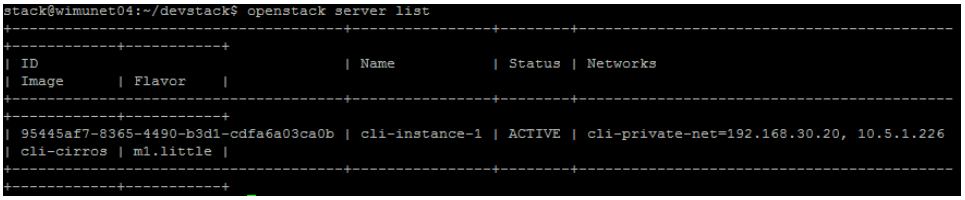


Figura 8.6: Instancia creada desde CLI.

mos conectado a la instancia deseada vía ssh donde además hemos escrito *ip a* para comprobar que efectivamente estamos dentro de esa máquina, con IP 192.170.1.103.

Por último vamos a ver la forma más interesante de acceder a una instancia de cara a un usuario externo a la red mediante su IP flotante. Cuando hablamos de IP flotante, en realidad no tenemos IPs públicas disponibles por lo que no se puede acceder directamente desde el exterior. La forma que tenemos en nuestro de comprobar que están funcionando es la siguiente. Vamos a acceder, por ejemplo, al proyecto creado desde la línea de comandos:

```
stack@wimunet4:~/devstack$ source openrc cli-user cli-project
```

Una vez dentro listaremos las instancias disponibles:

```
stack@wimunet4:~/devstack$ openstack server list
```

Y veremos la instancia que creamos desde la CLI de la Fig.8.6. En ella podemos ver que tiene una IP flotante asignada, la 10.5.1.226. Vamos ahora a hacer ping a esta IP para ver que ocurre:

```
stack@wimunet4:~/devstack$ ping 10.5.1.226
```

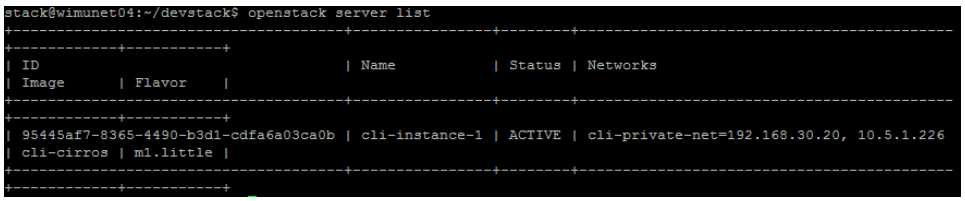


Figura 8.7: Listado de instancias disponibles.

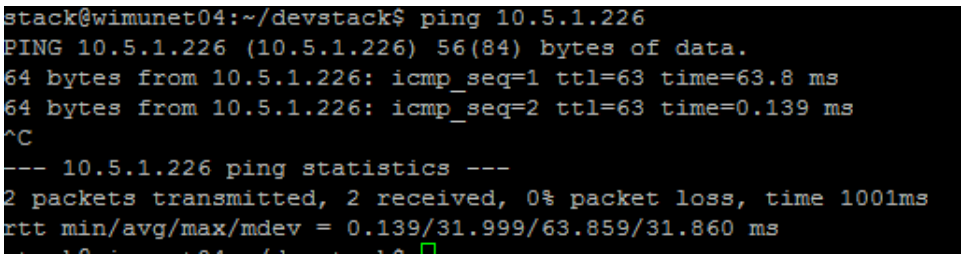


Figura 8.8: Ping a la IP flotante de la instancia.

Como vemos en la Fig.8.8 nos está respondiendo, a diferencia de lo que ocurría cuando hacíamos ping a la IP privada, donde teníamos que recurrir a los *namespaces* para ver si estaba accesible. Esta es la forma que tenemos de comprobar que efectivamente, a través de la IP flotante se puede acceder desde fuera de la red, incluso si no estuviésemos autenticados con el usuario que creo la instancia.

8.1. Comprobación de escenario en Heat

Esta sección vamos a dedicarla a comprobar si el script que creamos en la sección 7.5.2 crea la pila de manera correcta, realiza las comprobaciones oportunas, y en caso de caer una VM es capaz de recuperarla.

Empezaremos comprobando que el *stack* se creó con éxito al lanzar el script:

```
stack@wimunet4:~/devstack$ openstack stack list
```

Como vemos en la Fig.8.9 la pila con el nombre “autoscaling” está activa. Comprobemos ahora si se han creado las tres instancias que queríamos:

```
stack@wimunet4:~/devstack$ openstack server list
```

Así pues vemos en la Fig.8.10 que las VM están activas y se han creado

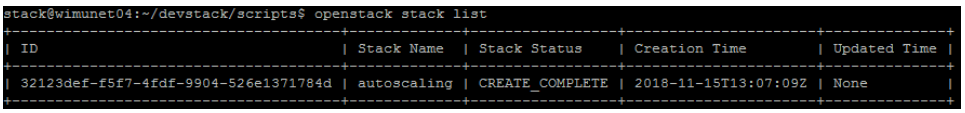


Figura 8.9: Comprobación del estado de la pila.

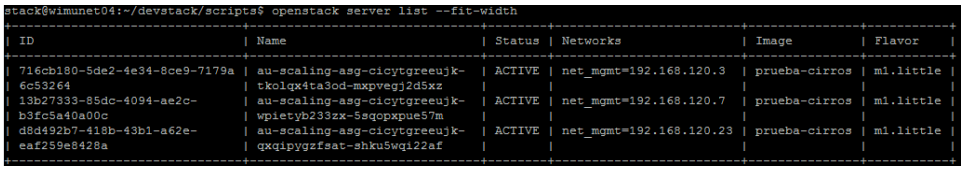


Figura 8.10: Comprobación del estado de las VMs.

con éxito.

Una vez que el mediante el script hemos creado la pila y esa las VMs, empezará de manera automática a comprobar la conectividad de las mismas, lanzando un mensaje "ACK" en caso de que tengamos conectividad con la VM que este comprobando en ese momento (Fig.8.11).

Vamos a ver ahora que ocurre cuando borramos una VM para simular la perdida de conectividad de la misma:

```
stack@wimunet4:~/devstack$ openstack server delete au-scaling-asg-
    cicytgreeujk-tkolqx4ta3od-mxpvegj2d5xz
```

Al borrar esta VM, cuando compruebe la conectividad con dicha máquina, en esta prueba aquella con IP 192.168.120.3, nos devolverá un mensaje "NACK" (Fig.8.13 y automáticamente lanzará un *update* de la pila con la que generamos la VNF para recuperarla. Este hecho se evidencia en la Fig.8.12, en la que vemos el mensaje "stack_status" donde nos indica que la actualización de esta está en proceso ("UPDATE_IN_PROGRESS").

Para comprobar que el proceso de actualización ha finalizado podemos escribir en la cLI:

```
stack@wimunet4:~/devstack$ openstack stack list
```

Y veremos como el proceso de actualización ha finalizado (Fig.8.14) tras como nos indicaba el script.

Finalmente, escribiendo de nuevo "openstack server list", veremos la salida que teníamos en la Fig.8.10 donde podemos comprobar que se ha vuelto a crear la instancia cuya conexión habíamos perdido.

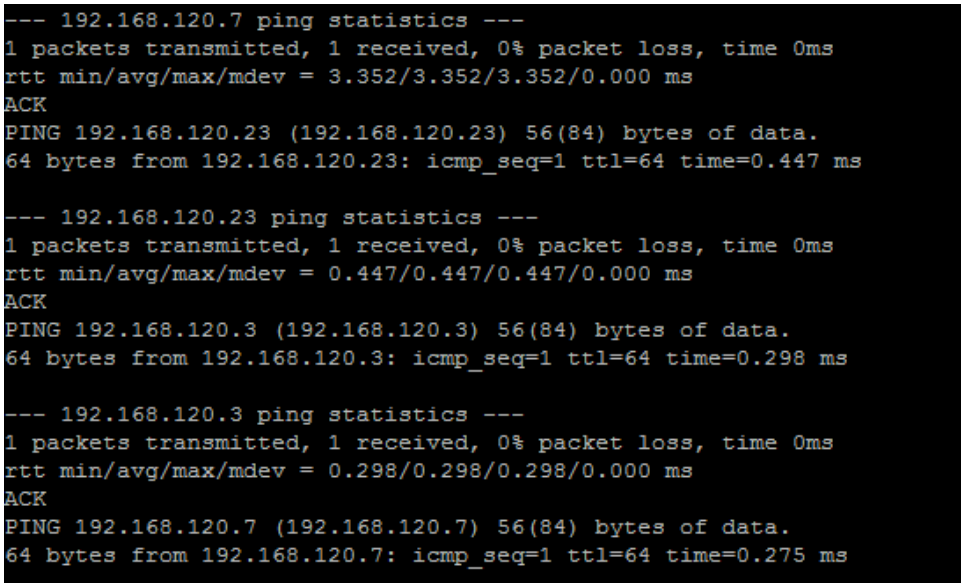


Figura 8.11: Comprobación de conectividad con las VMs.

8.2. Comprobación de escenarios en Tacker

Para la terminar el capítulo de comprobaciones, en primer lugar comentar algo que ya está evidenciado. A la hora de acceder a una instancia que se encuentra dentro la red de gestión "net_mgmt", creada al incorporar Tacker a nuestra IaaS, para hacer ping lo podemos hacer sin necesidad de usar *namespaces* como hemos visto en la sección anterior al comprobar la conectividad de las instancias creadas en Heat.

Por tanto, vamos a pasar a realizar la comprobación de la capacidad de *auto-healing* de la instancia creada. Para ello, en primer lugar vamos a acceder a ella vía SSH desde la CLU una vez conocida su IP:

```
stack@wimunet4:~/devstack$ ssh cirros@192.168.120.10
```

Una vez que hemos accedido a la consola, vamos a comprobar que se ha creado el archivo "hosts" que especificamos en la TOSCA template. Para ello dentro de la máquina escribimos "cat /etc/hosts" que efectivamente nos devolverá como salida las IPs y el nombre de la máquina que queríamos tener en el fichero (Figl8.15). Aunque este es un ejemplo sencillo de inserción de datos nos sirve para ilustrar la potencia de esta funcionalidad.

Vamos ahora a la comprobación más interesante de esta sección. El objetivo de la misma, es realizar un proceso de *auto-healing* similar al que hacíamos con el script en Python pero con una clara diferencia, y es que no

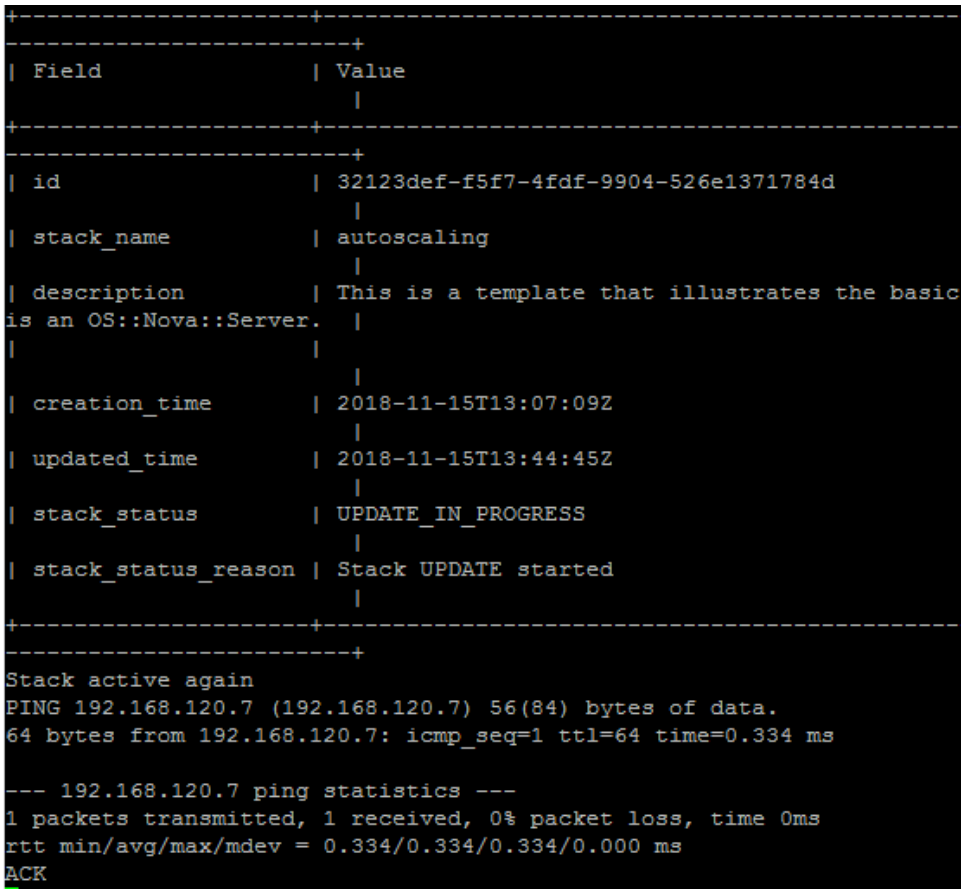


Figura 8.12: Realizando *update* de la pila.

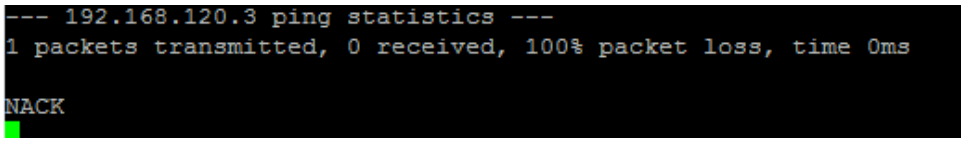


Figura 8.13: Detección de pérdida de conectividad con la VM.

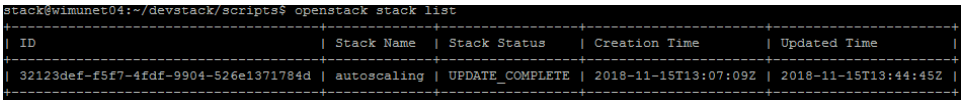


Figura 8.14: Evidencia de la finalización del *Update* de la pila.

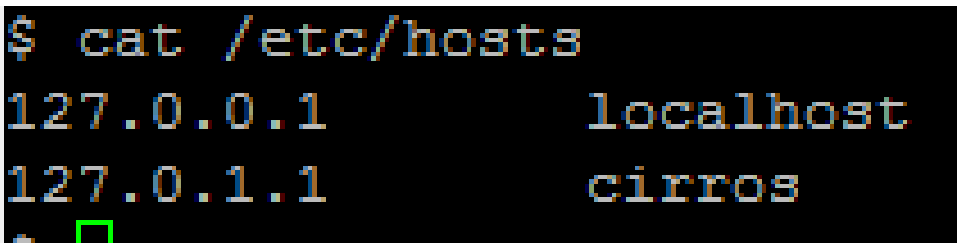


Figura 8.15: Salida del fichero de datos creado en la VM.

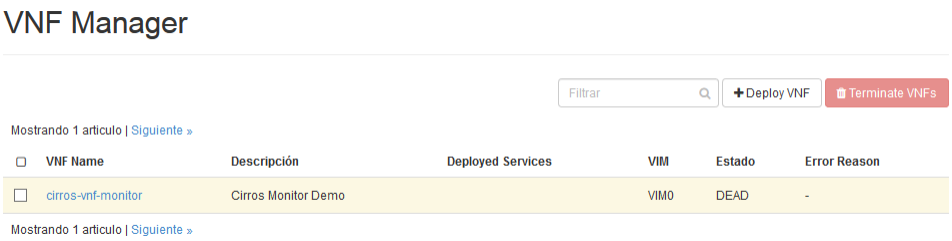


Figura 8.16: Detección de fallo en la VNF.

tenemos que crear dicho script ni lanzarlo, si no que de manera automatiza la plantilla definida en Tacker, comprobará si existe o no conexión con la VM que definimos en nuestra VNF y en caso de no ser así la recuperará sin que nosotros tengamos que intervenir de ningún modo.

Para ver lo que ocurre en este proceso, iremos al VNF Manager de Horizon. En la Fig.7.53 ya vimos como la VNF creada se encuentra activa. Con la información necesaria para realizar la prueba, vamos a proceder a eliminar la instancia creada por la VNF cuyo nombre veíamos en la Fig.7.54:

```
stack@wimunet4:~/devstack$ openstack server delete ci-be1f-ea24-4ef3
    -956f-44f90d640fe8-VDU1-gbhffxqhk3um
```

Cuando la eliminamos, dicha instancia, vemos como el estado de la VNF dentro del VNF Manager cambia a “DEAD” (Fig.8.16). Pasados unos segundos, la VNF se recupera de manera automática y vuelvo al estado de “ACTIVE” que veíamos en la Fig.7.53.

Capítulo 9

Conclusiones y líneas futuras

Durante el desarrollo de la presente memoria hemos ido tratando distintos puntos:

- Evaluar distintas opciones a las presentadas en la memoria para realizar el despliegue.
- Visión de los principales proyectos de OpenStack.
- Despliegue de la infraestructura.
- Gestión de la plataforma vía web a través del dashboard de Horizon, de la CLI.
- Creación de instancias y asignación de recursos a las mismas.
- Orquestación de funciones de red virtuales, VNFs, con Heat y Tacker.

De este modo hemos ido ilustrando no sólo el uso de la herramienta, si no la evolución que han sufrido en un corto espacio de tiempo la gestión de la infraestructura de los centros de datos y de telecomunicaciones.

Además se aprecia la clara evolución y tendencia a que todos los procesos se puedan realizar cada vez de manera más automatizada, para hacerlos así más escalables y creando mecanismos que nos permiten una gestión más eficiente de los recursos de nuestra IaaS o cloud.

De este modo se han cubierto todos los objetivos que se pretendían al inicio del proyecto y que nos han permitido crear nuestra propia cloud operativa y dotarla de mecanismos de autoservicio, orquestación y automatización además de mostrar todas las bondades que ofrecen las VNFs y la elección de OpenStack como herramienta para alcanzar nuestro fin.

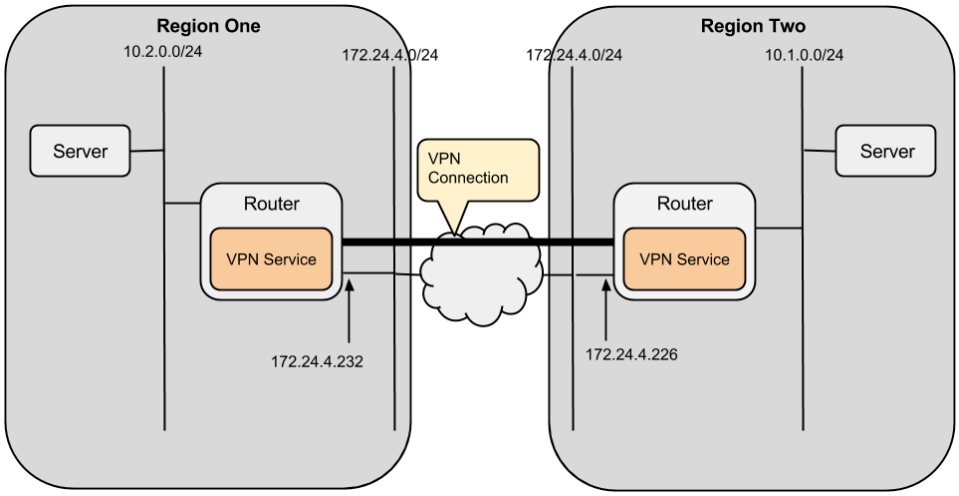


Figura 9.1: Esquema de un entorno multi-región en producción.

Fuente: Openstack Wiki

9.1. Líneas futuras

En esta sección se proponen tres líneas futuras ligadas entre sí y que unidas conllevan a la habilitación del core de una red de telefonía móvil tradicional lista para su puesta en producción.

Para ello en primer lugar proponemos una vez realizado este proyecto y adquiridos conocimientos profundos sobre OpenStack, se propone el despliegue de un entorno en producción basado en esta herramienta.

A continuación se plantea la implantación de un EPC (*Evolved Packet Core*) como servicio que dará el entorno en producción creado.

En el último apartado y como última línea veremos una descripción sobre algunas de las funcionalidades y ventajas de OSM (*Open Soruce Mano*) como herramienta para orquestación de NFVs sobre el entorno creado.

9.1.1. Despliegue de una infraestructura en producción

En este primer apartado se propone el despliegue de una infraestructura IaaS en producción para su uso corporativo. Una vez que hemos visto todas las posibilidades que OpenStack ofrece y hemos sido capaces de desplegar un entorno basado en el paquete all-in-one DevStack, el siguiente paso lógico es el despliegue de una nube privada para crear un entorno productivo.

En la sección 7.1.1 veíamos que una solución para la implementación de nuestra IaaS era el uso de una herramienta de despliegue a gran escala. Entre

ellas destacamos TripleO y Director. Para este desarrollo en producción se propone el uso de Triple0 el lugar de Director debido a que TripleO es la solución que ha creado la inmensa comunidad de desarrolladores y empresas que hay detrás de OpenStack, lo cual ya hemos visto las numerosas ventajas que presenta, mientras que Director es una solución de RedHat.

TripleO es un proyecto destinado a instalar, actualizar y operar nubes OpenStack utilizando las propias soluciones de la comunidad OpenStack que representan los cimientos de la misma, como son Nova, Neutron o Heat, para automatizar la gestión y el escalado de centro de datos. Todo el material para conseguir tal fin, incluido el desarrollo y la implementación, puede encontrarse en la documentación de TripleO.[66]

Para paliar las limitaciones de disponibilidad de equipos e IPs públicas de las que hemos hablado en distintas secciones de la memoria, proponemos esta vía como línea futura y además sugerimos el despliegue de un entorno soportado por la infraestructura de red que aparece en la Fig.9.1.

En dicha figura se refleja la infraestructura de red mínima necesaria para poder crear un entorno en producción que posteriormente, de ser necesario, sería fácilmente escalable. Tenemos dos regiones, y estas dos regiones estarían realmente separadas en dos localizaciones diferentes e interconectadas entre ella a través de una red VPN creada para tal fin. En cada una de ellas se alojaría un router que implemente funcionalidades de Firewall y los switches convenientes, que no aparecen en la figura puesto que la representación es del direccionamiento IP. De este modo crearíamos una pequeña red corporativa fácilmente escalable.

Con esto, podríamos conectar a dichos switches los servidores necesarios que implementaran las funcionalidades de los distintos nodos vistos en la sección 7.1.2 que se estimen oportunos, creando así nuestro entorno en producción.

9.1.2. EPC virtualizado como servicio

Virtual Evolved Packet Core (vEPC) es un marco para virtualizar las funciones necesarias para la convergencia de voz y datos en redes 4G *Long-Term Evolution* (LTE). vEPC mueve los componentes individuales de la red principal que tradicionalmente se ejecutan en hardware dedicado al software que opera en servidores comerciales de bajo costo (COTS).

Al virtualizar la funcionalidad del core de la red, los proveedores de servicios móviles pueden teóricamente personalizar redes para cumplir con los requisitos únicos de clientes individuales, mezclando y combinando componentes de red individuales según sea necesario. vEPC también puede reducir Capex y Opex al reducir la dependencia del hardware especializado, a la vez

que acelera la entrega de servicios y permite la escalabilidad bajo demanda y la respuesta a las condiciones de la red en tiempo real y las necesidades del usuario.

El objetivo de esta línea no sería desarrollar un vEPC. El departamento de Telemática de la ETSIIT ha trabajado ya en esta vía implementando un vEPC en *VirtualBox*, por lo que este proyecto futuro consistiría en la preparación y adaptación de esta imagen para hacerla compatible con Glance y poder crear instancias a partir de ella adaptando a nuestras necesidades y poniendo de este modo los recursos de nuestra IaaS para ofertar este servicio.

9.1.3. Orquetación de NFV con Open Source Mano

Como sabemos, OpenStack es principalmente conocido por ser el grupo más grande de proyectos de código abierto que forman la plataforma de software para la infraestructura de computación en la nube. Esta infraestructura se utiliza ampliamente en casos de uso de nubes privadas por muchas empresas. Después de una presentación de NFV por ETSI, OpenStack se ha convertido en una plataforma de infraestructura clave para NFV. En la mayoría de las implementaciones de NFV, OpenStack se utiliza en la capa VIM para proporcionar una interfaz estandarizada para administrar, supervisar y evaluar todos los recursos dentro de la infraestructura de NFV.

Varios proyectos de OpenStack (como Tacker, Neutron o Nova entres otros) son capaces de gestionar componentes de infraestructura virtualizada del entorno NFV. Como ejemplo, la tratada en el proyecto, Tacker, se utiliza para construir VNF Manager genérico (VNFM) y NFV Orchestrator (NFVO), que ayuda en el despliegue y operación de VNF dentro de la infraestructura de NFV. La integración de proyectos de OpenStack introduce varias características a la infraestructura de NFV. Las características incluyen características de rendimiento, fijación de CPU, segmentación de redes, escalabilidad, alta disponibilidad, flexibilidad y habilitación *multi-site*.

Los proveedores de servicios de telecomunicaciones y las empresas han implementado su entorno NFV con OpenStack: AT & T, China Mobile, SK Telecom, Ericsson, Deutsche Telekom, Comcast, Bloomberg, etc.

La capa MANO es responsable de la orquestación y la administración completa del ciclo de vida de los recursos de hardware y las funciones de red virtual (VNF). En otras palabras, la capa MANO coordina los recursos de la Infraestructura de NFV (NFVI) y los asigna eficientemente a varios VNF. Hay varias opciones disponibles como pila de software tridimensional para MANO, pero la OSM (*Open Source Mano*) alojada en ETSI es en gran medida la preferida debido a la gran actividad a nivel de la comunidad, el marco altamente maduro, la preparación para la producción, la facilidad de inicio y la alimentación constante de casos de uso por parte de los miembros.

Esto unido a que OSM proporciona un método para invocar la operación de actualización de VNF con un impacto mínimo en el servicio de red en ejecución es el motivo de la propuesta de estar herramienta de orquestación como línea futura.

Con el continuo apoyo y participación de la comunidad para la innovación de características, OSM ahora ha evolucionado para llevar el marco de CI / CD (Integración Continua y Entrega Continua) a la capa MANO. La última versión OSM ha aportado un gran conjunto de características y mejoras al marco OSM que ha impactado la funcionalidad, la experiencia del usuario y la madurez, lo que permite varias mejoras para NFV MANO desde la perspectiva de usabilidad e interoperabilidad.

OSM ha adoptado constantemente los principios nativos de la nube y puede implementarse fácilmente en la misma, ya que la instalación se basa en contenedores y se ejecuta con la ayuda del motor de orquestación de contenedores. El monitoreo y las capacidades de circuito cerrado también se han mejorado.

Se espera que la próxima versión, la versión 5 de OSM se lance en Noviembre de 2018 y se agregará con más funciones relacionadas con 5G, Network Slicing y VNF basadas en contenedores.

¿Por qué OpenStack con Open Source MANO para capa MANO en NFV? Tanto OpenStack como OSM tienen una gran comunidad que tiene un ritmo rápido para la innovación de NFV y la mayor contribución de todas las compañías inmersas en mejorar las características actuales y desarrollar nuevas capacidades para los principales proyectos bajo ella.

Para el caso de NFV, OpenStack estandarizó las interfaces entre los elementos e infraestructura de NFV. OpenStack se utiliza para ofertas de soluciones comerciales de compañías como Canonical / Ubuntu, Cisco, Ericsson, Huawei, IBM, Juniper, Mirantis, Red Hat, Suse, VMware y Wind River. Un gran porcentaje de las implementaciones de VIM se basa en OpenStack debido a la simplicidad en el manejo y operación de varios proyectos orientados a proporcionar almacenamiento, cómputo y creación de redes para NFVI.

Así pues, como última línea futura se pretende implementar a nuestro despliegue en producción, OSM. De este modo, ante una infraestructura que tienda al crecimiento, podemos lograr la obtención de todos los beneficios de la integración de NFV MANO utilizando OSM y OpenStack debido a una administración y una implementación eficiente, ligera y simple.

Bibliografía

[1] "OpenStack-AnnualReport2017.pdf." [Online]. Available: https://www.openstack.org/assets/reports/OpenStack-AnnualReport2017.pdf

[2] T. L. Foundation, "New 2018 Open Source Technology Jobs Report Released Today: Rapid Growth in Demand for Open Source Tech Talent," Jun. 2018. [Online]. Available: https://www.linuxfoundation.org/press-release/2018/06/new-2018-open-source-technology-jobs-report-released-today-rapid-growth-in-demand-for-open-source-tech-talent/

[3] "Open source software for creating private and public clouds." [Online]. Available: https://www.openstack.org/

[4] B. Silverman and M. Solberg, "What is NFV?" in *OpenStack for Architects - Second Edition*, 2nd ed. Packt Publishing, May 2018. [Online]. Available: http://proquest.safaribooksonline.com/book/operating-systems-and-server-administration/virtualization/9781788624510/nfv-architecture/146ebf1c_d2b1_48a7_b901_6219edebbce4_xhtml

[5] ——, "The difference between NFV and Software-Defined Networking (SDN)," in *OpenStack for Architects - Second Edition*, 2nd ed. Packt Publishing, May 2018. [Online]. Available: http://proquest.safaribooksonline.com/book/operating-systems-and-server-administration/virtualization/9781788624510/nfv-architecture/146ebf1c_d2b1_48a7_b901_6219edebbce4_xhtml

[6] "ETSI - NFV." [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv

[7] "AWS | Elastic compute cloud (EC2) de capacidad modificable en la nube." [Online]. Available: https://aws.amazon.com/es/ec2/

[8] "Virtualización de VMware." [Online]. Available: https://www.vmware.com/es/solutions/virtualization.html

[9] "vmware-vsphere-datasheet.pdf." [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vsphere/vmware-vsphere-datasheet.pdf

[10] "Apache Cloudstack." [Online]. Available: https://cloudstack.apache.org/about.html

[11] "About the Project – OpenNebula." [Online]. Available: https://opennebula.org/about/project/

[12] "Eucalyptus, CloudStack, OpenStack and OpenNebula: A Tale of Two Cloud Models – OpenNebula." [Online]. Available: https://opennebula.org/eucalyptus-cloudstack-openstack-and-opennebula-a-tale-of-two-cloud-models/

[13] Q. Jiang, "CY12-Q4 Community Analysis — OpenStack vs OpenNebula vs Eucalyptus vs CloudStack," 2013. [Online]. Available: http://www.qyjohn.net/?p=2733

[14] A. Shrivastwa, S. Sarat, K. Jackson, C. Bunch, E. Sigler, and T. Campbell, "Vendor offerings," in *OpenStack: Building a Cloud Environment*. Packt Publishing, Sep. 2016. [Online]. Available: http://proquest.safaribooksonline.com/9781787123182/ch09lvl2sec94_html

[15] "Integrated OpenStack | API de OpenStack | VMware." [Online]. Available: https://www.vmware.com/es/products/openstack.html

[16] "Choosing a Cloud Platform | Managed Cloud by Rackspace." [Online]. Available: https://www.rackspace.com/es/cloud

[17] "Software en la nube HPE Helion OpenStack." [Online]. Available: https://www.hpe.com/es/es/product-catalog/detail/pip.hpe-helion-openstack-cloud-software.1010838414.html

[18] "Cisco Solutions: OpenStack." [Online]. Available: https://www.cisco.com/c/en/us/solutions/data-center-virtualization/openstack-at-cisco/index.html

[19] "Mirantis | Software | OpenStack." [Online]. Available: https://www.mirantis.com/software/openstack/

[20] "SwiftStack." [Online]. Available: https://www.swiftstack.com/

[21] K. Peterson2700042WJC, "IBM Cloud Manager with OpenStack : IBM Cloud Manager with OpenStack," Oct. 2009. [Online]. Available: https://www.ibm.com/developerworks/community/wikis/home?lang=es#!/wiki/W21ed5ba0f4a9_46f4_9626_24cbbb86fbb9

[22] “SUSE | OpenStack Cloud.” [Online]. Available: https://www.suse.com/es-es/solutions/cloud/openstack/

[23] “Releases: OpenStack Releases.” [Online]. Available: https://releases.openstack.org/

[24] “Fortune U.S. 500 (100) - 2018 (Fortune) | Ranking The Brands.” [Online]. Available: https://www.rankingthebrands.com/The-Brand-Rankings.aspx?rankingID=131&year=1212

[25] “Statistic | OpenStack market size worldwide 2014-2021.” [Online]. Available: https://www.statista.com/statistics/498552/openstack-market-size/

[26] A. Shrivastwa and S. Sarat, “OpenStack in action,” in *Learning OpenStack*. Packt Publishing, Nov. 2015. [Online]. Available: http://proquest.safaribooksonline.com/book/operating-systems-and-server-administration/virtualization/9781783986965/9dot-looking-ahead/ch09s02_html

[27] “devstack: System for quickly installing an OpenStack cloud from upstream git for testing and development,” Aug. 2018, original-date: 2011-11-16T19:11:08Z. [Online]. Available: https://github.com/openstack-dev/devstack

[28] “BOE.es - Documento BOE-A-2007-12946.” [Online]. Available: https://www.boe.es/buscar/doc.php?id=BOE-A-2007-12946

[29] “ENDESA CLIENTES | El precio de la electricidad en el mercado regulado (tarifa PVPC) |.” [Online]. Available: https://www.endesaclientes.com/precio_electricidad

[30] “Foundation | The OpenStack Foundation.” [Online]. Available: https://www.openstack.org/foundation/

[31] “Companies | OpenStack Foundation supporting companies.” [Online]. Available: https://www.openstack.org/foundation/companies/

[32] “Nova - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Nova#Nova

[33] “Neutron | OpenStack Docs: Welcome to Neutron’s documentation!” [Online]. Available: https://docs.openstack.org/neutron/latest/

[34] “Swift | OpenStack Docs: Welcome to Swift’s documentation!” [Online]. Available: https://docs.openstack.org/swift/latest/

[35] “Glance | OpenStack Docs: Welcome to Glance’s documentation!” [Online]. Available: https://docs.openstack.org/glance/latest/index.html

[36] “Get images | OpenStack Docs: Get images.” [Online]. Available: https://docs.openstack.org/image-guide/obtain-images.html

[37] “Cinder | OpenStack Docs: The OpenStack Block Storage Service.” [Online]. Available: https://docs.openstack.org/cinder/latest/

[38] “Keystone | OpenStack Docs: Keystone, the OpenStack Identity Service.” [Online]. Available: https://docs.openstack.org/keystone/latest/

[39] “Celiometer | OpenStack Docs: Welcome to Ceilometer’s documentation!” [Online]. Available: https://docs.openstack.org/ceilometer/latest/

[40] “Trove - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Trove

[41] “Sahara - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Sahara

[42] “Ironic - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Ironic

[43] “Zaqar - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Zaqar

[44] “Manila - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Manila

[45] “Designate - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Designate

[46] “Barbican - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Barbican

[47] “Magnum - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Magnum

[48] “Murano - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Murano

[49] “Congress - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/Congress

[50] “curl.” [Online]. Available: https://curl.haxx.se/

[51] "Horizon | OpenStack Docs: Horizon: The OpenStack Dashboard Project." [Online]. Available: https://docs.openstack.org/horizon/latest/

[52] M. Dorn, "Heat architecture," in *Preparing for the Certified OpenStack Administrator Exam.* Packt Publishing, Aug. 2017. [Online]. Available: http://proquest.safaribooksonline.com/book/certification/9781787288416/heat-orchestration-service/3af8f994_f356_4a18_a9eb_0bdc23e32891_xhtml

[53] ——, "Horizon dashboard," in *Preparing for the Certified OpenStack Administrator Exam.* Packt Publishing, Aug. 2017. [Online]. Available: http://proquest.safaribooksonline.com/book/certification/9781787288416/exam-objective-managing-stacks/a118a8f8_fa15_40aa_950e_93d68963e50e_xhtml

[54] "Tacker - OpenStack." [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[55] "TOSCA Simple Profile in YAML Version 1.1." [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/csprd02/TOSCA-Simple-Profile-YAML-v1.1-csprd02.html

[56] "VNFD Descriptor Template in YAML." [Online]. Available: https://docs.openstack.org/tacker/ocata/devref/vnfd_template_description.html

[57] "Install | OpenStack Docs: OpenStack Installation Guide." [Online]. Available: https://docs.openstack.org/install-guide/

[58] "TripleO | OpenStack Docs: Welcome to TripleO documentation." [Online]. Available: https://docs.openstack.org/tripleo-docs/latest/

[59] "Director Installation and Usage - Red Hat Customer Portal." [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_openstack_platform/7/html/director_installation_and_usage/index

[60] "Queens | OpenStack Releases: Queens." [Online]. Available: https://releases.openstack.org/queens/index.html

[61] "Floating IP for Networking in OpenStack Public and Private clouds," Aug. 2012. [Online]. Available: https://www.mirantis.com/blog/configuring-floating-ip-addresses-networking-openstack-public-private-clouds/

[62] "Command List | OpenStack Docs: Command List." [Online]. Available: https://docs.openstack.org/python-openstackclient/pike/cli/command-list.html

[63] "OpenRC | OpenStack Docs: Set environment variables using the OpenStack RC file." [Online]. Available: https://docs.openstack.org/zh_CN/user-guide/common/cli-set-environment-variables-using-openstack-rc.html

[64] "GitHub | OpenStack Repository: openstack-tfg-poc." [Online]. Available: https://github.com/ourtelcloudthings/openstack-tfg-poc

[65] "OpenWRT | Wireless Freedom: Welcomo to the Project." [Online]. Available: https://openwrt.org/

[66] "TripleO - OpenStack." [Online]. Available: https://wiki.openstack.org/wiki/TripleO

Apéndice A

Arquitectura lógica de OpenStack

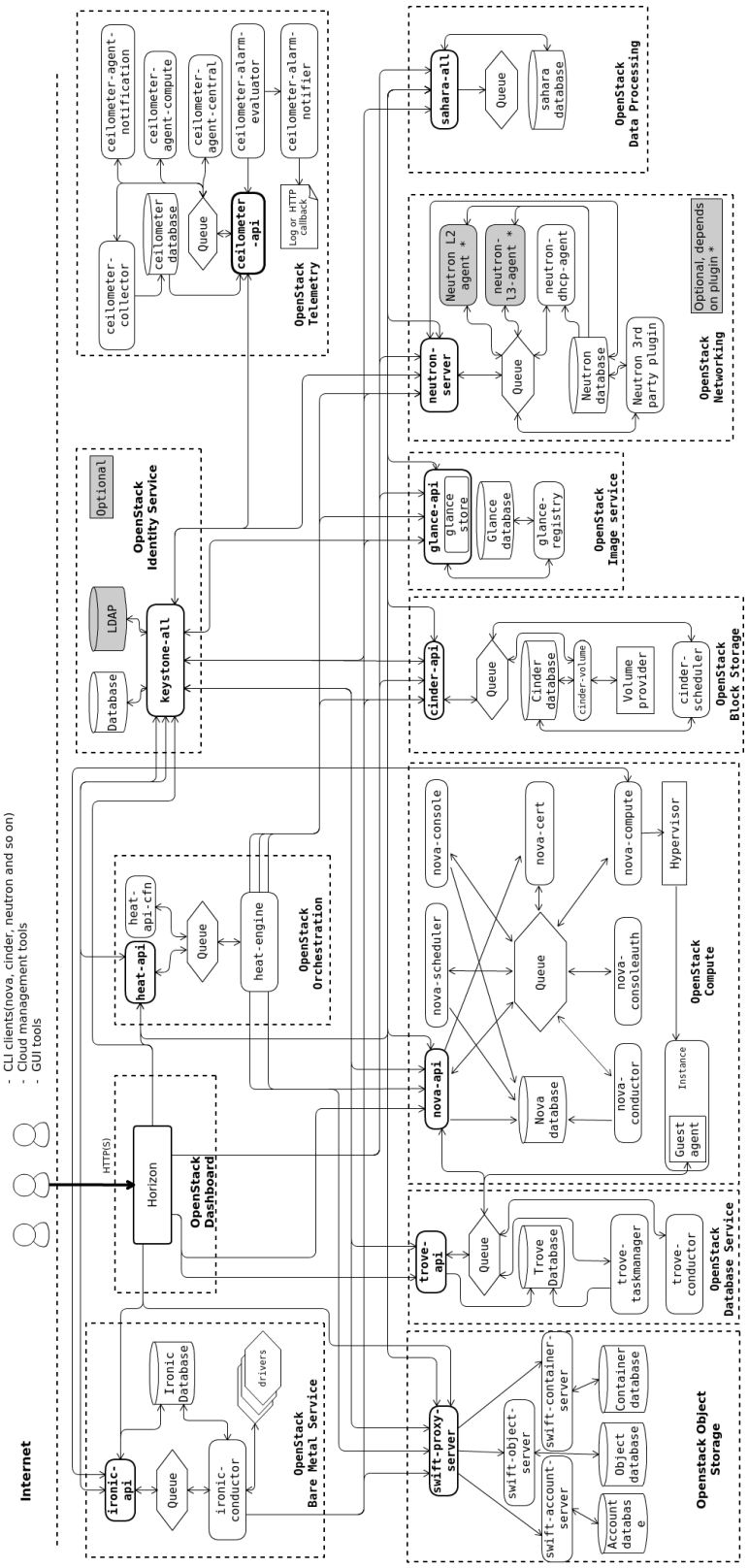


Figura A.1: Arquitectura lógica de OpenStack.

Fuente: OpenStack

Apéndice B

Archivo de configuración para la instalación

En esta sección del apéndice mostramos el archivo de configuración *local.conf* que contiene la configuración necesaria para la instalación de OpenStack en el servidor junto con la descripción de los parámetros introducidos.

A continuación se muestra dicho archivo, *local.conf*:

```
stack@wimunet4:~/devstack$ vi local.conf

[[local|localrc]]
############################################################
# Configurando el entorno a crear
############################################################
PUBLIC_INTERFACE=enp3s0
HOST_IP=10.5.1.201
FLOATING_RANGE=10.5.1.224/24
FLAT_INTERFACE=enp3s0:1
Q_FLOATING_ALLOCATION_POOL=start=10.5.1.224,end=10.5.1.254

# Claves de acceso para los distintos servicios
ADMIN_PASSWORD=temporary
MYSQL_PASSWORD=temporary
RABBIT_PASSWORD=temporary
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=temporary

# Configuramos el primer servidor como la primera region
REGION_NAME=RegionOne

############################################################
# Customizando entorno
############################################################
# Permitir el uso de Pip para manejo de paquetes
PIP_USE_MIRRORS=False
USE_GET_PIP=1

#OFFLINE=False
```

```
# Reclone nos permite reconstruir los escenarios creados en caso de
    reinicio del servidor
RECLONE=True

# Habilitar log para reportes de fallos y otros.
LOGFILE=$DEST/logs/stack.sh.log
VERBOSE=True
ENABLE_DEBUG_LOG_LEVEL=True
ENABLE_VERBOSE_LOG_LEVEL=True

# Neutron ML2 with OpenVSwitch
Q_PLUGIN=ml2
Q_AGENT=openvswitch

# Habilitar grupos de seguridad
Q_USE_SECGROUP=False
LIBVIRT_FIREWALL_DRIVER=nova.virt.firewall.NoopFirewallDriver

# Habilitar Heat
enable_plugin heat https://git.openstack.org/openstack/heat stable/
    queens
# Habilitar Heat para su uso desde el Dashboard
enable_plugin heat-dashboard https://git.openstack.org/openstack/heat-
    dashboard stable/queens
# Habilitar funciones de red
enable_plugin networking-sfc git://git.openstack.org/openstack/
    networking-sfc stable/queens

# Descarga de automatica de imagenes para el servicio de Heat.
IMAGE_URL_SITE="http://download.cirros-cloud.net"
IMAGE_URL_PATH="/0.4.0/"
IMAGE_URL_FILE="cirros-0.4.0-x86_64-disk.img"
IMAGE_URLS+=","$IMAGE_URL_SITE$IMAGE_URL_PATH$IMAGE_URL_FILE

# Habilitar Ceilometer
enable_plugin ceilometer https://git.openstack.org/openstack/
    ceilometer stable/queens
# Habilitar Tacker
enable_plugin tacker https://git.openstack.org/openstack/tacker stable
    /queens
# Habilitar Tacker para su uso desde Horizon
https://git.openstack.org/cgit/openstack/tacker-horizon stable/queens

# Servicios adicionales para Nova
enable_service n-novnc
enable_service n-cauth
disable_service tempest
[[post-config|/etc/neutron/dhcp_agent.ini]]
[DEFAULT]
enable_isolated_metadata = True
```

- Veamos la descripción de los parámetros principales del archivo de configuración *local.conf* no descritos en el mismo:

  - Cabecera del archivo, es obligatoria. [[local—localrc]].
  - PUBLIC_INTERFACE=enp3s0. Interfaz de red física que servirá como gateway hacia la red pública.

- HOST_IP=10.5.1.201. Dirección IP del host que alojará OpenStack.
- FLOATING_RANGE=10.5.1.224/24. Rango de direcciones IPs flotantes.
- FLAT_INTERFACE=enp3s0:1. Interfaz de red que conecta el entorno creado a nuestra máquina.
- Q_FLOATING_ALLOCATION_POOL=start=10.5.1.224,end=10.5.1.254. A pesar de haber creado un rango de IPs flotantes, podemos establecer únicamente una cantidad de direcciones en ese rango de las que disponer.
- Claves de acceso para los distintos servicios. Después de estos parámetros, en el archivo se usan distintas contraseñas: una para el administrador del entorno cloud creado con acceso a la configuración completa, otra para la base de datos que se especifica que sea MySQL, otra para la cola de mensajes RabbitMQ incorporada por defecto en este paquete y por ultimo la contraseña de servicio predeterminada para la autenticación en Keystone de los servicios.

El resto de parámetros se describen en el propio archivo. Como se puede ver se especifica que este Servidor haga las veces de la *Region One*, que es el nombre que tendría este *site* pudiendo haber así gestionar por nombre otras regiones.

Apéndice C

Creación de un escenario mediante scripting en bash

Para la automación del escenario, crear el script *cli_creacion_escenario.sh* que sigue y ejecutarlo:

```
#!/bin/bash
source openrc admin admin
openstack project create --description 'Escenario creado con cli' cli-
    project --domain default
openstack quota set cli-project --instances 10
openstack quota set cli-project --ram 4096
openstack user create --project cli-project --password temporary cli-
    user
openstack role add --user cli-user --project cli-project Member
source openrc cli-user cli-project
openstack network create cli-private-net
openstack subnet create --network cli-private-net --subnet-range
    192.168.30.0/24 cli-private-subnet
openstack router create cli-router
openstack router set cli-router --external-gateway public
openstack router add subnet cli-router cli-private-subnet
openstack floating ip create public
openstack floating ip create public
openstack keypair create cli-keypair > cli-keypair.pem
chmod 0600 cli-keypair.pem
eval "$(ssh-agent)"
ssh-add cli-keypair.pem
openstack security group create cli-secgroup
openstack  security group rule create --remote-ip 0.0.0.0/0 --protocol
     tcp --dst-port 22  --ingress cli-secgroup
openstack  security group rule create --remote-ip 0.0.0.0/0 --protocol
     icmp --ingress cli-secgroup
openstack  security group rule create --remote-ip 0.0.0.0/0 --protocol
     tcp --dst-port 80:80  --ingress cli-secgroup
# Esta carpeta hay que crearla antes
openstack image create --disk-format qcow2 --min-disk 2 --file /opt/
    stack/devstack/images/cirros-0.4.0-x86_64-disk.img  --private cli-
    cirros
# openstack flavor create --ram 512 --disk 4 --vcpus 1 m1.little
```

```
nova boot --flavor m1.little --image cli-cirros --key-name cli-keypair
     --security-groups cli-secgroup --nic net-name=cli-private-net cli
    -instance-1
openstack volume create --image cli-cirros --size 8 --availability-
    zone nova cli-volume
openstack server add volume cli-instance-1 cli-volume --device /dev/
    vdb
```