UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MASTER

INGENIERÍA DE TELECOMUNICACIÓN

Diseño e implementación de un entorno para servicios de red en infraestructuras virtualizadas

Autor
Alejandro García Soria

Directores
Juan José Ramos Muñoz
Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Septiembre de 2020

UNIVERSIDAD DE GRANADA

Diseño e implementación de un entorno para servicios de red en infraestructuras virtualizadas

Autor
Alejandro García Soria

Directores
Juan José Ramos Muñoz
Jorge Navarro Ortiz

Departamento de Teoría de la Señal, Telemática y Comunicaciones

—

Granada, Septiembre de 2020

Diseño e implementación de un entorno para servicios de red en infraestructuras virtualizadas

Alejandro García Soria

Palabras clave: Open Source Mano, OpenStack, NFV, VNF, NS, orquestación, virtualización, contenedores, máquinas virtuales

Resumen

En la actualidad, la sociedad está muy influida por los avances tecnológicos, bien sea por el perfeccionamiento de algunas tecnologías o por el desarrollo de otras nuevas. Asimismo se persigue proporcionar un nivel de bienestar cada vez más sofisticado y globalizado. La investigación actual sobre la tecnología 5G y otras altamente relacionadas, como la virtualización de servidores o funciones de red, permiten imaginar un futuro cada vez más conectado y con tiempos de respuesta mucho menores.

Con el paso del tiempo la virtualización está siendo protagonista de múltiples modificaciones en el paradigma digital, apoyada por su utilización por las empresas más grandes en todos los sectores mercantiles.

Si nos centráramos en algún tipo concreto de virtualización, sería sin duda alguna, aquellas relacionadas con infraestructuras y redes debido al gran beneficio que aportan a las empresas que las utilizan reduciendo tanto el CAPEX como el OPEX.

El presente proyecto trata de ejemplificar un escenario probable en cloud computing o edge computing, mediante la creación de servicios de manera automatizada en dispositivos de pocos recursos. Para realizarlo, se emplean plataformas altamente apoyadas por la comunidad tecnológica, como son OpenStack y Open Source Mano.

Gracias a ellas se podrán realizar algunas pruebas de despliegue de servicios de red, haciendo uso de virtualización de funciones de red, máquinas virtuales y contenedores. Tras los resultados de las pruebas se puede entender el gran auge de tecnologías de contenedores para dar servicios.

Ambas herramientas tienen el respaldo de grandes empresas como IBM, Oracle, Canonical o Telefónica. El uso conjunto de estos dos proyectos podría suponer un gran ahorro en infraestructura y sistemas de monitorización de recursos. Esto es muy importante para empresas que provean algún servicio como Walmart o Rakuten.

Al fin y al cabo, no deja de ser un avance tecnológico que evoluciona acorde a la investigación e innovación en las herramientas que incorpora, permitiendo ser cada vez más eficiente e incluir nuevas tecnologías de implementación de servicios en el menor tiempo posible, como puede ser el uso de contenedores.

Project Title: Project Subtitle

Alejandro García Soria

Keywords: Open Source Mano, OpenStack, NFV, VNF, NS, orchestration, virtualization, containers, virtual machines

Abstract

Our society is very influenced by technological advances, due to the improvement of some technologies or the development of new ones. Additionally, it is intended to offer an increasingly sophisticated and globalized level of well-being. The current research on the 5G technology and other highly related technologies, such as server virtualizations or virtual network functions, make it possible to imagine an increasingly connected future with shorter response times.

Over the time, virtualization has become one of the main key for digital change, which is supported by large tech enterprises across sectors and industries

It is worth to highlight the importance of infrastructure and network virtualization in the improvement of the CAPEX and OPEX

This project tries to illustrate a likely topology in cloud computing or edge computing, by launching services with an automated way in low resource devices. In order to do this, platforms which are highly supported by the technology community are used, such as OpenStack and Open Source.

Thanks to them, it will be possible to carry out some tests of network services deployment, make use of virtualization of network functions, virtual machines and containers. After the results of the tests it is possible to understand the great boom of container technologies to provide services.

Both programs are supported by largest tech companies such IBM, Oracle, Canonical or Tenefónica. By using these platforms combinated, great savings in infrastructure and resource monitoring systems might occur. This is very important for companies that provide some services like Walmart or Rakuten.

After all, it is a technological advance that evolves by incorporating new technologies, which allows it to increase efficient and to include new

technologies to implement services throughout the internet as quickly as possible, such as the use of containers.

Yo, **Alejandro García Soria**, alumno de la titulación Máster en Ingeniería de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con [redacted] autorizo la ubicación de la siguiente copia de mi Trabajo n e aster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alejandro García Soria

Granada a 10 de Septiembre de 2020 .

D. **Juan José Ramos Muñoz**, Profesor del Área de Ingeniería Telemática del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

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

Informan:

Que el presente trabajo, titulado ***Diseño e implementación de un entorno para servicios de red en infraestructuras virtualizadas***, ha sido realizado bajo su supervisión por **Alejandro García Soria**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 10 de Septiembre de 2020 .

Los directores:

Juan José Ramos Muñoz **Jorge Navarro Ortiz**

Agradecimientos

Durante todo el tiempo que he estado en proceso de aprendizaje y preparación profesional, que únicamente comienza aquí, tengo y quiero agradecer a todas aquellas personas que me han ayudado a conseguir mis objetivos, mostrando apoyo y confianza en mi persona.

Pero sin duda tengo que agradecer a mi familia, que son los que más han soportado mis momentos de frustración y me han ayudado a seguir adelante y no rendirme en ningún momento.

A mis tutores para darme la oportunidad de realizar este proyecto y gracias a él, abrirme los ojos de un tema bastante desconocido hasta el momento.

A todo y cada uno de vosotros, gracias por acompañarme en este camino.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo se recogerá una introducción de los conceptos más relevantes para poder entender la memoria y el presente trabajo en su totalidad. Es necesario conocer la gran importancia que están adquiriendo todos los conceptos referentes a la tecnología 5G y la virtualización. Cada vez más empresas están incorporando mecanismo de virtualización a sus servidores, servicios o, en general, infraestructura informática.

Se recogerán los objetivos principales que han promovido la realización de este proyecto en temática y herramientas empleadas.

Para finalizar el capítulo, se indicará la estructura de la presente memoria, incluyendo un breve resumen de los diferentes temas tratados, y algunas de las principales referencias consultadas tanto para la realización de la memoria como para la instalación y configuración de los distintos servicios.

Este proyecto surge por interés tanto propio como de los tutores en la investigación de nuevas tecnologías y/o posibles soluciones basadas en virtualización de infraestructuras o funciones de red. Este ámbito lleva algún tiempo experimentando un gran auge entre las grandes empresas tecnológicas u otras punteras en otros mercados, como automoción o finanzas.

1.1. Motivación y Contexto

Desde hace algún tiempo se lleva debatiendo sobre el futuro de la virtualización y el cambio de paradigma que podría acarrear en el ámbito tecnológico de manera directa, pero también en el ámbito global, porque muchas empresas comenzaron a emplearlo para proporcionar sus servicios reduciendo

costes y mejorando el servicio de cara al usuario.

Pero antes de nada, es recomendable saber cómo surge este concepto y qué es, a día de hoy, la virtualización.

Se cree que el concepto de virtualización tiene su origen [1] en los "mainframes" de finales de los años 60 cuando IBM apostó, dedicando una gran esfuerzo y recursos, al desarrollo de la compartición de recursos de computación entre una gran cantidad de usuarios, incrementando de esta manera la eficiencia de los grandes computadores de los que disponía. Este modelo representó un gran avance, pudiendo diferentes organizaciones, e incluso personas, utilizar los recursos de un ordenador como si dispusieran del suyo propio. Este concepto en la actualidad se conocería como "Time-sharing".

El modelo es similar a lo que hoy en día se conoce como virtualización, entendiéndolo como la capacidad de ejecutar varios sistemas independientes al mismo tiempo mediante abstracciones que utilizan el hardware físico para crear agregados lógicos de recursos (CPU, memoria, aplicaciones, red, etc). Sabiendo, más o menos, del concepto en el que está basada la virtualización, vamos a ver algunas definiciones de virtualización por empresas punteras del sector.

• Definición de Microsoft: "La virtualización crea un entorno informático simulado, o virtual, en lugar de un entorno físico. A menudo, incluye versiones de hardware, sistemas operativos, dispositivos de almacenamiento, etc., generadas por un equipo. Esto permite a las organizaciones particionar un equipo o servidor físico en varias máquinas virtuales. Cada máquina virtual puede interactuar de forma independiente y ejecutar sistemas operativos o aplicaciones diferentes mientras comparten los recursos de una sola máquina host [2]."

• Definición de Red HAT: "La virtualización es una tecnología que permite crear servicios de TI útiles mediante recursos que están ligados tradicionalmente al hardware. Además, distribuye sus funcionalidades entre diversos usuarios o entornos, lo que permite utilizar toda la capacidad de una máquina física [3]."

• Definición de IBM: " La virtualización es un proceso que permite una utilización más eficiente del hardware físico de la computadora y es la base del cloud computing. La virtualización utiliza software para crear una capa de abstracción sobre el hardware de la computadora que permite que los elementos físicos de un solo PC - procesadores, memoria, almacenamiento

y más - se dividan en múltiples máquinas virtuales. Cada una ejecuta su propio sistema operativo y funciona como un PC independiente, a pesar de que se ejecuta en sólo una parte del hardware de una máquina física [4]."

• Definición de Citrix: "La virtualización es una tecnología que simula la funcionalidad de hardware para crear servicios de TI basados en software como servidores de aplicaciones, almacenamiento y redes.

La virtualización crea varias máquinas virtuales (VM) a partir de una máquina física utilizando un software que se llama hipervisor. Las VMs se comportan igual que las máquinas físicas mientras que solo dependen de los recursos informáticos en una única máquina (CPU, memoria, almacenamiento...). El hipervisor asigna los recursos informáticos a cada máquina virtual según los necesite. Esto hace que las operaciones de TI sean mucho más eficientes y económicas [5]."

Si vemos noticias sobre la tecnología de virtualización podremos ver cómo está siendo empleada principalmente para mejorar la capacidad en los centros de datos.

Según un informe de *Uptime Institute*, donde han entrevistado a multitud de ejecutivos y gerentes de empresas IT, señala que los centros de datos utilizan mecanismos de virtualización para liberar capacidad de sus infraestructuras IT. Un 51 % de las empresas creen que la virtualización es la tecnología más contribuyente a reducir la demanda de capacidad en sus instalaciones, mientras que el 32 % dice que es la nube pública. Por otro lado, un 19 % dice que las tecnologías con mayor impacto en la reducción de demanda de capacidad son las capacidades avanzadas de computación de los servidores [6].

"¿Cuál es el futuro de la virtualización de servidores? La virtualización de servidor puede ayudar a combatir la dispersión de servidores, hacer un mejor uso de la potencia informática, reducir las facturas de energía y mejorar la agilidad y flexibilidad del centro de datos [7]."

En un primer momento podemos pensar que la virtualización solo está pensada para operar en servidores pero no es así, hay muchos elementos de una infraestructura IT que pueden ser virtualizados para proporcionar algunas ventajas. Algunos tipos de virtualización son:

- Virtualización de escritorio
- Virtualización de red

- Virtualización de almacenamiento

- Virtualización de aplicación

- Virtualización en la “nube”

- Virtualización en Contenedores

En la actualidad, las empresas del sector IT están utilizando cada vez más estos tipos de virtualización para apoyar sus infraestructuras y mejorar los servicios que proporcionan, pero sobre todo están cobrando especial importancia las relacionadas con contenedores, la “nube” y la red.

Para entender el funcionamiento de la virtualización en contenedores y porqué desde hace algún tiempo se cree que será la tecnología capaz de desbancar a la “virtualización”, entendida como uso de máquinas virtuales, primero definiremos el concepto de contenedor.

Esta tecnología es un método para empaquetar una aplicación de tal forma que se pueda ejecutar con sus dependencias y de manera independiente al resto de procesos. Esta tecnología surge ante la problemática de mover software desde el PC de un desarrollador a un servidor de prueba, o de un centro de datos de una empresa a un servidor en la nube. Cuando se hacía esta migración habitualmente podían surgir errores de incompatibilidades de sistemas operativos, bibliotecas necesarias para la ejecución, almacenamiento, seguridad o topología de red [7].

¿Por qué se virtualiza empleando contenedores en lugar de máquinas virtuales? El concepto de contenedorización (virtualización en contenedores) permite que las diferentes instancias virtuales compartan un único sistema operativo host y binarios, bibliotecas o controladores relevantes. Este nuevo enfoque de virtualización reduce el desperdicio de recursos por cada contenedor [8].

Con el auge del uso de contenedores en la actualidad, la complejidad de las aplicaciones y la gran flexibilidad y ligereza que presentan los contenedores han surgido nuevas arquitecturas que empaquetan los diferentes servicios en varios contenedores. Por ello para simplificar el empleo de contenedores y mejorar su funcionalidad surge un nuevo concepto, la orquestación de contenedores.

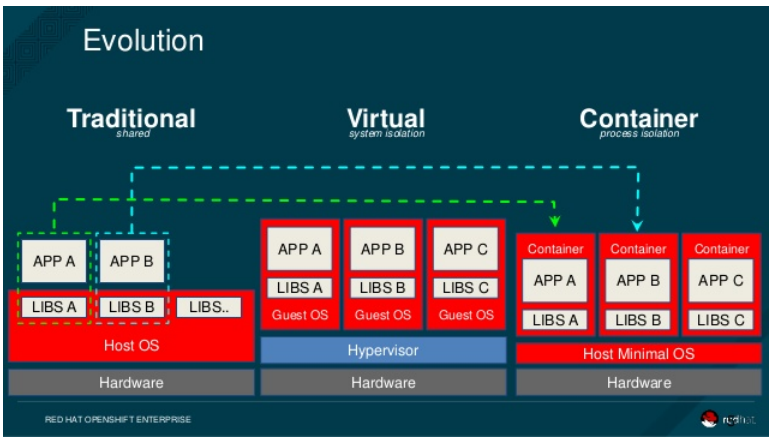

Figura 1.1: Evolución de la virtualización (RedHat)[9]

La orquestación de contenedores es una herramienta o sistema que permite automatizar el despliegue, gestión, escalado, interconexión y disponibilidad de las aplicaciones basadas en contenedores. Con el empleo de un orquestador de contenedores podemos realizar configuraciones automáticas, desplegar automáticamente los servicios basados en contenedores, realizar balanceado de carga, auto-escalado y auto-reinicio en caso de fallo o control de salud.

El uso de orquestación de contenedores facilita mucho el empleo de estos y cuando grandes empresas tecnológicas como Google, Microsoft, IBM y Red Hat, se unen tras un mismo proyecto para apoyar un mismo proyecto, Kubernetes, se puede prever un futuro prometedor para este tipo de tecnología.

Por otro lado, tenemos la virtualización de red [10] que emplea software para generar una "vista" de la red que el administrador puede utilizar para gestionar toda la red desde una consola. De esta manera, se agrupan los recursos de la red física, elementos y funciones de hardware (por ejemplo, conexiones, conmutadores, enrutadores, etc) en un software que se ejecuta en el hipervisor. El administrador de red tiene la capacidad de modificar y controlar estos elementos sin necesidad de tocar los componentes físicos, simplificando en gran medida la gestión de red. Además, gracias a la virtualización de red también es posible automatizar muchas tareas reduciendo los errores manuales y el tiempo de aprovisionamiento, aumentando la productividad y la eficiencia de la red.

Hay varios tipos de virtualizaciones de red [4]:

- Redes definidas por software, SDN: basadas en los conceptos de abstracción y modularidad proponen un nuevo paradigma para la evolución de las redes. Virtualizan el hardware que controla el enrutamiento del tráfico de la red, conocido como "plano de control"

- Virtualización de la función de red, NFV: virtualiza uno o más dispositivos hardware que proporcionan una función de red específica - por ejemplo, un cortafuegos, un balanceador de carga o un analizador de tráfico - facilitando la configuración, el abastecimiento y la gestión de todos los dispositivos.

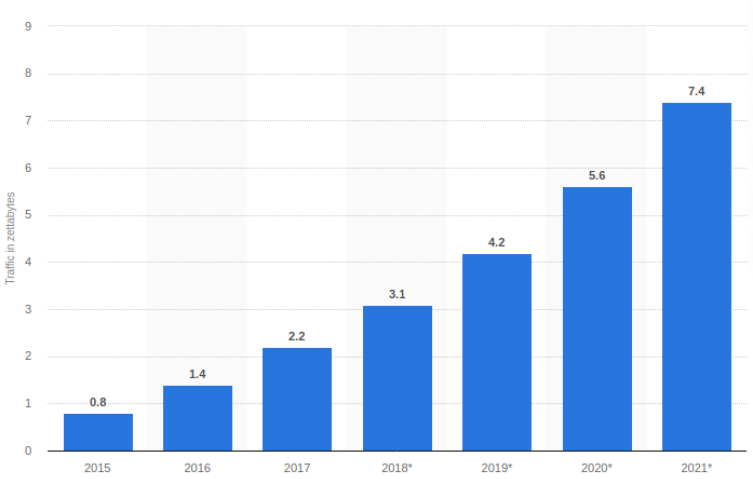

Figura 1.2: Cantidad de tráfico de SDN y NFV en los centros de datos del mundo, 2015 a 2021 [11].

La gráfica de la figura 1.2 proporciona un pronóstico del tráfico de redes definidas por software y de virtualizaciones de red dentro de centros de datos en todo el mundo en el intervalo de 2015 a 2021. Se puede observar que a finales de 2020, 5.6 zettabytes del tráfico que haya en centros de datos tendrán de origen funciones SDN y NFV.

IDC (*International Data Corporation*) [12] estima que el mercado mundial de SDN de centros de datos tendrá un valor de más de 12 mil millones de dólares en el año 2022, registrando un tasa compuesta anual durante los años entre 2017 y 2022. Teniendo en cuenta que el mercado ingresó casi 5.150 millones de dólares en 2017, suponiendo un incremento del 32.2 % respecto de 2016.

Según Cisco [13], el futuro de las redes SDN y el Cloud Computing es trabajar conjuntos, por los grandes beneficios y facilidades que proporcionan este tipo de redes, adaptándose automáticamente a las cargas de trabajo, mejorando la seguridad y facilitando su gestión, al fin y al cabo, se trata de operar con redes cada vez más optimizadas que varían en función de las variaciones en las aplicaciones y en los requisitos de las redes.

Se prevé un futuro prometedor para esta unión de tecnologías y es fácil describir por qué:

- La integración de entre las distintas SDN empleadas en las empresas y las nubes sigue creciendo año tras año. Teniendo en cuenta que los proveedores de Cloud quieren que cada vez se use más su infraestructura, y cuanto más se adentre en la infraestructura, más integrada y más rígida será la plataforma utilizada en la empresa. Buscando mejorar los servicios existentes, e incluso añadir nuevos, teniendo una mejor coordinación de los distintos elementos de la red, mejorando la seguridad y reduciendo los posibles problemas físicos en la infraestructura.

- Se tiene una supervisión y gestión de red integrada, dando toda la información que se quiere monitorizar tanto de la nube pública y las redes locales en un sistema homogéneo. Además se añade la probabilidad de que terceros realicen una supervisión de red integrada, análisis y resolución reactiva de problemas.

- Capacidad de habilitar aprendizaje automático. Se puede incluir el uso de aprendizaje automático con la finalidad de encontrar las configuraciones de red óptimas en aquellas redes que por su naturaleza cambian con alta frecuencia. Conforme se mejoren las capacidades de aprendizaje automático e inteligencia artificial, se irán incluyendo capacidades de autocontrol, autocuración y autoreparación continua de la red.

Se ha mencionado anteriormente el término "*cloud computing*" pero a qué se refiere este tipo de tecnología. Consiste en ofrecer servicios a través de la conectividad y gran escala de Internet, desde aplicaciones hasta centro de datos completos mediante un modelo de negocio basado en el pago por uso. Para ello se emplean una asignación de recursos flexible, aumenta o reduce los recursos rápidamente en función de la demanda; servicio medido para que solo se pague por aquello que se utiliza.

Según Salesforce [14], un aspecto importante en la computación en la nube es que ofrece tanto a individuos como empresas de todos los tamaños la

capacidad de un pool de recursos de computación con un buen mantenimiento, seguro, de fácil acceso y baja demanda, como servidores, almacenamiento de datos y soluciones de aplicaciones. Esto permite que las empresas posean una mayor flexibilidad en relación a sus datos e informaciones, que se pueden acceder desde cualquier lugar y hora, siendo algo primordial en empresas con sedes alrededor del mundo.

Figura 1.3: El valor del cloud computing en el sector.

En el sector tecnológico, el empleo de *cloud computing* está provocando una transformación importante en las grandes empresas. Está viéndose afectado tanto las estrategias de trabajo como las relaciones con los consumidores, llegando a no ser nunca igual que antes.

En España [15], un estudio realizado por el estudio Quint Group, reveló que aproximadamente un 70 % de las organizaciones coinciden en el uso de IaaS y PaaS - Infraestructure-as-a-Service y Platform-as-a-Service, respectivament - como un elementos estratégicos de los servicios que proporcionan.

Forrester [16], una empresa de investigación de mercados especializada en el impacto de la tecnología realizó un informe donde especulaba con un contexto más que prometedor, siendo un año estratégico, para esta tecnología en el año 2019 teniendo claras algunas ideas:

- Aumento de la inversión y crecimiento de los proveedores de nube pública.
- Reestructuración de los contenedores.
- Hibridación entre la nube pública y privada.
- Automatización de procesos gracias a la inteligencia artificial.

Pero, ¿y si se lleva la computacion al borde de la red?. En este contexto surge el conocido *Edge computing* con el cual es posible ofrecer servicios contenido e inteligencia en milisegundos con necesidades de baja latencia, altas velocidades y calidad, e incluso escasos recursos por la naturaleza de los dispositivos en los que se pretende implementar.

Para entender correctamente lo que es el *Edge Computing* [17] deberemos de conocer cómo se encuentra el mundo digital en la actualidad. Actualmente se navega a través de una gran cantidad de información y cómputo, con la llegada de tecnologías de IoT y sus innumerables sensores y dispositivos conectados, la cantidad de datos que fluyen a través de la red crece a gran escala. Según Cisco, en 2020 habrá más de 50.000 millones de dispositivos inteligentes conectados a la red. Esto significa una ingente cantidad de datos fluyendo por internet que necesita ser gestionada, analizada o computada, en general.

Toda esta información generada normalmente se gestiona en el cloud. Sin embargo, se puede gestionar, analizar o procesar antes, en el borde, dando lugar al *Edge Computing*

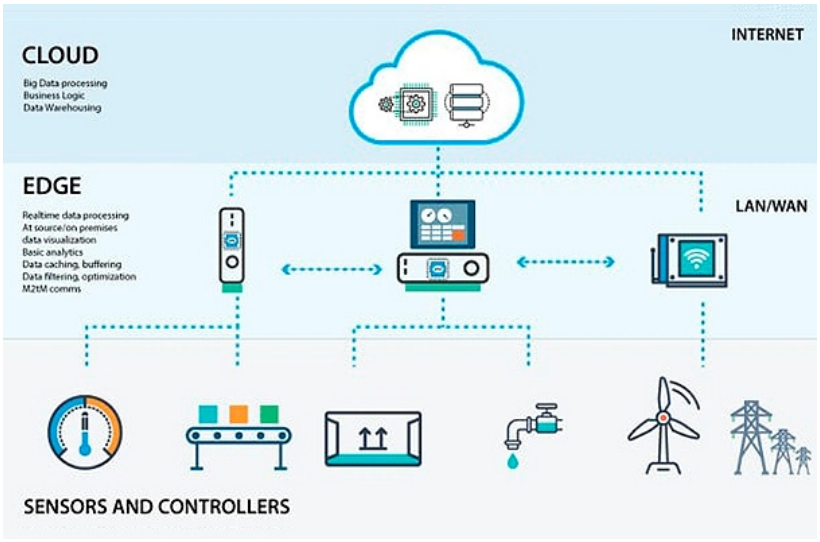

Figura 1.4: Edge Computing, Cloud y sensores IoT [17].

De esta forma, cuando se emplea "Edge Computing" se descarga un poco la red y se procesan los datos en zonas más cerca de donde se generan en lugar de enviarlos directamente a través de la red con la finalidad de procesarlos en data centers y la nube.

En muchas ocasiones, incluso se procesan en los mismo sensores o controladores, por ello se necesitan que los servicios o aplicaciones se ejecuten en entornos con pocos recursos.

En este contexto de funcionamiento o procesamiento de datos en la nube, teniendo en cuenta el cloud computing y el edge computing, apoya el desarrollo de la tecnología hacia un entorno caracterizado por nuevas tecnologías como 5G más potentes y que pueden mejorar en muchos casos las calidades de servicio y experiencia vistas desde los usuarios.

Si se quiere saber hacia dónde se dirige la tecnología basta con realizar una rápida búsqueda en internet para darse cuenta de que 5G está adquiriendo la mayor parte de la atención desde todos los estamentos de la sociedad para bien y mal, pero sobre todo para conseguir proyectos con la mentalidad de encabezar el desarrollo o la investigación de este nuevo área que según las predicción de lo que puede suponer mejoraría muchos aspectos cotidianos de la vida.

Sin ir más lejos, en España hay cada vez más proyectos destinados al desarrollo de esta tecnología con la finalidad de mejorar y realizar una transformación digital de la sociedad.

"El Ministerio de Asuntos Económicos y Transformación Digital ha adjudicado recientemente ocho nuevos proyectos piloto 5G, la quinta generación de redes móviles que permitirá una navegación hasta 10 veces más rápida de que la actual." [18]

Finalizando, si unificamos las distintas tecnologías que se han tratado hasta ahora podremos imaginar hacia dónde va dirigida la nueva realidad digital de todos en prácticamente todos los aspectos, tanto personales como profesionales, y la manera en que se está intentando facilitar e incluso predecir posibles comportamientos para actuar sobre ellos.

1.2. Objetivos

En esta sección se expondrán los principales objetivos que han motivado la realización del presente proyecto.

El objetivo principal es realizar el diseño y despliegue de un entorno preparado para la instanciación rápida de servicios o aplicaciones críticas en el menor tiempo posible, dado que en estos casos, el despliegue no puede tardar minutos. Se tratará de realizar un entorno controlado y optimizado basado en MANO (*Management and orchestration*).

Para la realización de dicho entorno utilizaremos únicamente software open-source. Los PCs, que harán de servidores utilizarán distribuciones de Linux. Se ha de tener en cuenta que se necesita un software orquestador de NFV, en nuestro caso utilizaremos Open Source MANO (OSM), y por otro lado necesitaremos un VIM (Virtualized Infraestructure Manager), para este caso utilizaremos OpenStack.

La finalidad principal es poder desplegar VNFs desde OSM en la red virtualizada por OpenStack, ya sea como instancias de máquinas virtuales o como contenedores. Además de comparar el tiempo de despliegue entre crear los servicios en máquinas virtuales desde OpenStack y OSM, o incluso en contenedores.

1.3. Estructura de la memoria

En esta parte del capítulo se llevará a cabo un breve resumen de la estructura que compone la presente memoria, para cada uno de los apartados o capítulos se expondrá una síntesis de las ideas principales expuestas.

La memoria está compuesta por ochos capítulos, en donde se explicarán los conceptos teóricos necesarios para entender las herramientas empleadas en la solución en este proyecto. Para finalizar, se incluirá un apartado que recoge toda la bibliografía consultada.

Adicionalmente, se incluirá un anexo con un breve manual de usuario que detalle algunos mecanismos de uso del escenario planteado.

- Capítulo 1. Introducción:

  En este capítulo se recogerá cómo ha evolucionado con el paso de los años el mundo relacionado con la virtualización y las tecnologías de telefonía móvil, analizando el panorama actual de ambas tecnologías y el impacto que están teniendo en la digitalización de la vida cotidiana.

  Se expondrán los principales objetivos planteados en la realización del presente proyecto, se explicará la estructura que compone la memoria y, para finalizar, se recogerán algunas de las referencias bibliográficas de mayor relevancia para el estudio y desarrollo del trabajo fin de máster.

- Capítulo 2. Estado del arte:

  En este apartado se presentará pequeño apartado introduciendo algunos conceptos necesarios para entender el resto del capítulo y un estudio de las diferentes alternativas disponibles en el mercado, explicando algunas de las características más relevantes de cada una de ellas y comparándola con la solución elegida para el proyecto.

- Capítulo 3. Especificación de requisitos:

  En este capítulo se realizará una introducción básica de la solución elegida y explicada en este informe, mediante la exposición de los requisitos y decisiones previas necesarias para el correcto funcionamiento y conexión entre los distintos servidores.

- Capítulo 4. Planificación y estimación de coste:

  En esta sección se expondrá la planificación temporal del proyecto, prosiguiendo con una escueta explicación de los recursos necesarios y/o imprescindibles para la instalación y configuración. Además, se recogerá una estimación del coste total de la realización del proyecto, teniendo en cuenta cada uno de los recursos utilizados y las diferentes fases de desarrollo que lo componen.

- Capítulo 5. Tecnologías y herramientas utilizadas:

  En esta parte de la memoria se recogerá un estudio teórico detallado de las diferentes tecnologías y términos con una alta relevancia en el proyecto. Además, se explicarán, de forma extensa, las herramientas (software) empleado para plantear el escenario del trabajo.

- Capítulo 6. Diseño e Implementación:

  En este capítulo se expondrá en detalle el procedimiento seguido para diseñar e implementar el escenario de MANO y OpenStack sus respectivos servidores, así como la configuración para que sea posible la comunicación entre ambos.

- Capítulo 7. Fase de pruebas:

  Este apartado recogerá algunas de las pruebas realizadas en el escenario montado, haciendo una breve descripción del mismo, de modo que se compruebe su correcto funcionamiento y comunicación entre servicios desplegados en el proyecto.

- Capítulo 8. Conclusiones:

  Con este apartado se concluye el estudio realizado para el trabajo fin de master. En él se recogerán los aspectos más relevantes del proyecto. Se expondrá un breve resumen de algunas ideas o conocimientos adquiridos tras el trabajo realizado sobre el escenario tecnológico desarrollado.

  Posteriormente, se recogerán posibles mejoras o líneas de trabajo que pueden ser explotadas para mejorar el funcionamiento o incluso para preparar el escenario a un ámbito de producción.

- Bibliografía:

  En esta parte de la memoria se recogerán todos los documentos y/o enlaces bibliográficos consultados durante la realización y estudio del presente proyecto.

- Anexo:

  Como última parte de la memoria, se recogerán los ficheros de configuración y plantillas de despliegue de servicios utilizados en el proyecto.

1.4. Principales referencias utilizadas

En este apartado se recogerán las referencias bibliográficas más consultadas para el estudio teórico e instalación de las diferentes herramientas y conexiones, y la posterior configuración en los servidores:

- OpenStack: Web oficial del proyecto de computación en la nube open-source, OpenStack, encargado de proveer una infraestructura como servicio. Se puede encontrar toda la información referente a los distintos módulos o servicios que componen esta solución *"cloud"*, así como, toda la documentación útil y necesaria para entender su comportamiento e instalación.
  Disponible en : **https://www.openstack.org/**

- Open Source MANO: Página oficial del proyecto open-source para proporcionar una pila de gestión y orquestación basada en los estándares ETSI NFV. Se trata de un sitio web donde encontrar tutoriales y documentación necesaria para entender e instalar el software.
  Disponible en : **https://osm.etsi.org/**

Capítulo 2

Estado del arte

En este capítulo de la memoria se realizará un estudio del estado del arte, para ello se expondrán algunas de las características más importantes y el coste, si fuera posible, de cada una de las alternativas disponibles en el mercado tecnológico actual, para realizar un escenario similar al de este proyecto.

Además, para facilitar su entendimiento se incluirá una introducción de algunos conceptos especialmente relevantes en el desarrollo del proyecto.

Posteriormente, se recogerán algunas de las características de las alternativas elegidas, con lo que poder realizar una breve comparativa de pros y contras de las diferentes opciones.

Hoy en día, hay que prestar especial atención al avance que está experimentando el sector tecnológico en prácticamente todos sus ámbitos de actuación, aunque nosotros nos centraremos en la tecnología de virtualización general y, además, en la virtualización de funciones de red. Cada vez se escucha más a menudo que las empresas que proporcionan servicios en internet, e incluso otras menos relacionadas con ellos, están optando por virtualizar algunos de sus sistemas para mejorar la calidad de servicio y de experiencia de sus usuarios mediante el uso de la *"nube"*, cloud computing.

En la actualidad, hay muchas alternativas para acercarse a estas soluciones e implementar dichas tecnologías en el funcionamiento de los servicios. Si bien es cierto, las más utilizadas y conocidas dependen de las grandes empresas tecnológicas, pero no son las únicas y se expondrán en mayor o menor medida a continuación.

2.1. Conceptos previos.

Para facilitar el entendimiento y poder seguir con facilidad el desarrollo del proyecto se expondrán una serie de conceptos importantes relacionados con las tecnologías o herramientas utilizadas.

Para comenzar, y aunque se haya descrito un poco en el apartado anterior se definirá el concepto de virtualización y se darán a conocer algunos de los tipos de virtualización más importantes actualmente, tras lo cual nos centraremos en aquellos que nos incumbe.

- **Virtualización.**

La virtualización consiste en realizar una abstracción de recursos físicos de IT normalmente ligados al hardware de máquinas físicas y mediante la aplicación de software la creación de un entorno informático simulado o virtual, pudiéndose incluir sistemas operativos, servicios de almacenamiento, memoria, red, entre otros.

Al crearse todos los servicios en un mismo equipo las empresas o individuos que lo emplean consigue una mayor escalabilidad y cargas de trabajo, entendiéndose como una facilidad para adecuar las capacidades de las máquinas virtuales o incluso los anfitriones (servidores) en función de los requerimientos en cada momento.

- **Tipos de virtualización.** [4]

La virtualización presente varios tipos, algunos más habituales y otros un poco más desconocidos pero no menos importantes:

- Virtualización de escritorio: permite ejecutar múltiples sistemas operativos de escritorio, cada uno en su propia máquina virtual desde un mismo PC.

  Hay varios tipos de virtualización de escritorio:

  - Infraestructura de escritorio virtual (VDI): se ejecutan varios escritorios en máquinas virtuales en un servidor central y los transmite a los usuarios finales que inician sesión en dispositivos con un cliente liviano.

  - Virtualización de escritorio local: ejecuta un hipervisor en un PC local, permitiendo al usuario ejecutar uno o más sistemas operativos sin necesidad de modificar nada en el sistema operativo anfitrión.

- Virtualización de red: este tipo de virtualización emplea software para crear una abstracción de los elementos y funciones de hardware que componen una red pudiendo controlarlos todos desde una misma consola sin necesidad de manipulación física de los diferentes componentes.

  Los distintos tipos de virtualización de red incluyen las redes definidas por software (SDN, software-defined networking) y la virtualización de funciones de red (NFV, network function virtualization)

- Virtualización de almacenamiento: permite que se pueda acceder a todos los dispositivos de almacenamiento de la red, tanto servidores individuales como unidades independientes de almacenamiento, pudiendo administrarlos como si se tratara de un único dispositivo de almacenamiento.

- Virtualización de datos: la mayoría de empresas almacenan datos de multitud de aplicaciones, empleando múltiples formatos, en múltiples lugares, desde la nube hasta sistemas físicos en sus instalaciones. Mediante la virtualización de datos se permite a cualquier aplicación acceder a todos los datos de manera independiente a la fuente, el formato o la ubicación.

- Virtualización de aplicación: permite ejecutar software de aplicación sin instalarlo directamente en el sistema operativo, es distinto a la virtualización de escritorio porque en este caso solo se ejecuta la aplicación en un entorno simulado.

  Existen varios tipos de virtualización de aplicaciones:

  - Virtualización de aplicaciones locales: la aplicación al completo se ejecuta en un dispositivo final, pero sobre un sistema de tiempo de ejecución en lugar del hardware físico.

  - Aplicaciones en “streaming”: la aplicación se aloja en un servidor que envía los paquetes necesarios para que se pueda ejecutar por el usuario final en el momento que lo requiera.

  - Virtualización de aplicaciones basadas en el servidor: la aplicación se ejecuta al completo en el lado del servidor, que se encarga de enviar únicamente una interfaz de usuario al cliente.

- Virtualización de data-cente: abstrae la mayor parte de elementos hardware de un centro de datos en software, permitiendo de esta forma que el administrador pueda dividir el centro de datos físicos en varios centros de datos virtuales preparados con los recursos necesarios para cada cliente.

- Virtualización de CPU: es una tecnología fundamental para que puedan funcionar los hipervisores, las máquinas virtuales y los sistemas operativos. Permite dividir una sola CPU en múltiples virtuales para ser utilizadas por múltiples máquinas virtuales.

- Virtualización de GPU: sabiendo que una GPU está compuesta por varios núcleos empleados para mejorar el rendimiento general de la computación haciéndose cargo del procesamiento gráfico o necesidades de cómputo de gran potencia. La virtualización de GPU brinda la capacidad de que varias máquinas virtuales empleen de forma total o parcial el potencial de una sola GPU para acelerar el video, emplear inteligencia artificial u otras aplicaciones de alta carga computacional.

- Virtualización de Linux: el sistema operativo Linux posee su propio hipervisor integrado, denominado kernel-based virtual machine (KVM), soporta extensiones del procesador para virtualización de Intel y AMD, con la finalidad de poder crear máquinas virtuales desde un sistema operativo anfitrión Linux.

- Virtualización en la nube: como se ha mencionado anteriormente, el modelo de cloud computing depende directamente de la virtualización. Con la virtualización de servidores, almacenamiento, red y centros de datos, los proveedores de cloud computing pueden ofrecer una serie de distintos servicios a sus clientes:

- **Beneficios de la virtualización.**

El empleo de virtualización en sus distintos aspectos conlleva una serie de beneficios:

- Eficiencia de recursos: antes de virtualizar cada sistema necesitaba de sus recursos hardware específicos, como CPU dedicada. El administrador era el encargado de monitorizar y configurar los servidores independientes necesarios para cada aplicación o servicio que se deseaba ejecutar. Mediante la virtualización se permite ejecutar cada aplicación

en su propia máquina virtual con su sistema operativo independiente, aprovechando de mejor modo los recursos totales del equipos físico.

- Control más sencillo: la sustitución de equipos físicos por múltiples máquinas virtuales que se pueden gestionar desde una misma consola facilitando en gran parte la gestión y permitiendo crear flujos de trabajo automatizados para la gestión de servicios IT. Un ejemplo de ello es que los administradores poseen herramientas de configuración e implementación automatizadas con las que pueden desplegar plantillas software que realicen todo el trabajo de despliegue.

- Tiempo de inactividad mínimo: los errores en los sistemas operativos y en las aplicaciones pueden provocar tiempos de inactividad e interrumpir el servicio que se estaba prestando, disminuyendo la productividad de los usuarios. Los administradores, mediante virtualización, pueden desplegar máquinas redundantes y realizar conmutación de equipos si surge algún error en alguna de ellas.

- Aprovisionamiento más rápido y eficiente: comprar, instalar y configurar hardware para cada aplicación requiere mucho tiempo. En aquellos casos en los que el hardware esté preinstalado se puede conseguir una reducción considerable del tiempo de aprovisionamiento en las máquinas virtuales sobre las que correr aplicaciones significativamente más rápido.

- **Virtualización en Contenedores vs VMs.**

En la virtualización con máquinas virtuales se reproduce lo que sería un equipo hardware completo, donde instala un sistema operativo sobre el que lanzar una aplicación. Esto hace que sea más eficiente que no aplicar virtualización, pero también hay que decir que se duplican elementos innecesarios, que consumen recursos para cada aplicación que se quiera ejecutar.

Una alternativa con buen rendimiento son los contenedores. Comparten el kernel del sistema operativo anfitrión y únicamente necesitan instalar las cosas de las que depende la aplicación que se desea ejecutar, como librerías o variables de entorno.

Por consiguiente, hace que el uso de contenedores sea más simple, rápido y portable en cuanto al despliegue.

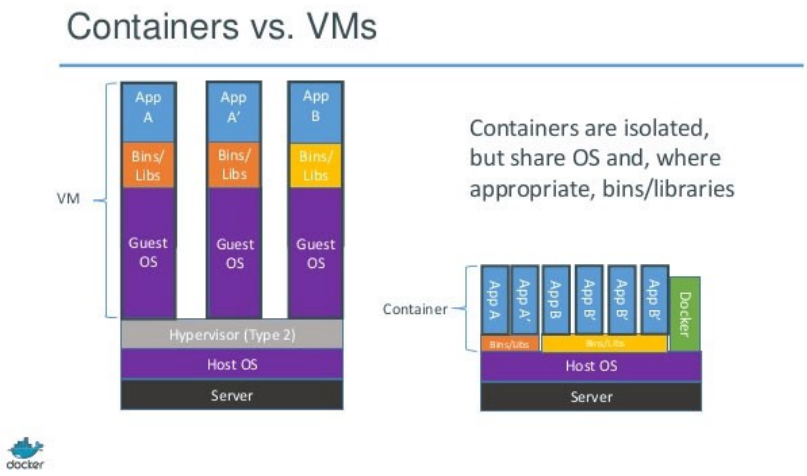

Figura 2.1: Máquinas virtuales frente a contenedores.[19]

Los contenedores son como una instancia de ejecución de una imagen del sistema operativo, pero vista de un modo efímero dado que se lanzan para realizar algunas tareas y se eliminan. Por esta naturaleza efímera, los usuarios pueden llegar a lanzar una mayor cantidad de instancias de contenedores en comparación a las máquinas virtuales.

Además, para poder crear aislamiento entre los contenedores que se ejecutan a la vez sobre un mismo equipo se emplean características propias del kernel de Linux: los espacios de nombre y los cgroups. De este modo, para aislar un contenedor del resto de recursos del sistema, se crea un espacio de nombres para cada recurso. Los grupos de control (cgroups) son empleados monitorear y limitar los recursos del sistema: cpu, memoria, E/S de disco, red, etc [20].

- **Virtualización de red (sdn y NFV).**

Como se ha descrito en el apartado 1.1. la virtualización de red está basada en el empleo de software para crear una vista global de la red y que un administrador pueda controlar todos los elementos desde una misma consola, agrupando en un mismo lugar tanto la red física como virtual de la infraestructura. Actuando de esta forma, se consigue una capacidad de modificación, configuración, monitorización global sin necesidad de actuar directamente sobre los elementos físicos que componen la red, además de proporcionar mecanismo de automatización de despliegue o escalado, mejorando tareas de aprovisionamiento de recursos o mejorando la productividad y eficiencia de la red.

Los tipos de virtualización de red más conocidos en este sentido son las SDN y NFV.

Las SDN, redes definidas por software, se basan en el concepto de abstracción por capas y modularidad, separando el plano de control de una red del plano de datos encargado del reenvío de tráfico a través de la red, con la finalidad de presentar un nuevo paradigma en la evolución de las redes con las nuevas tecnologías. El objetivo principal es crear una red que gestione de forma centralizada y que a la vez sea programable.

Las NFV, virtualización de funciones de red, se encargan de virtualizar uno o más dispositivos hardware para proporcionar una función específica de red - por ejemplo, un cortafuegos, un balanceador de carga, un analizador de tráfico - facilitando en gran medida la configuración, aprovisionamiento y gestión de todos los elementos que componen una red.

La intención inicial de utilizar NFV es trasladar las funciones de cada elemento de la red, separándolo de su hardware físico, a una máquina virtual independiente, pudiendo desarrollarse una red completa en un único sistema físico.

En muchas ocasiones pueden confundirse los términos SDN y NFV, pero realmente no son lo mismo. Las SDN fueron desarrolladas con la finalidad de crear redes más flexibles y reducir los costes de implementación. Para ello emplea un software que funciona como controlador de los distintos elementos de la red, firewalls, routers y switches.

Sin embargo, las NFV, como su nombre indica, proporciona funciones de red virtuales - firewalls virtuales, balanceadores, IDS, etc - y se desarrollaron con la intención de poder ofrecer servicios de red abstrayendo el hardware y pudiendo modificar la función en cualquier momento. Estas funciones reciben el nombre de "funciones de red virtuales" (VNF, virtual network function).

Esto quiere decir que en el caso de emplear ambas tecnologías en paralelo, SDN sería la encargada de crear una superposición virtual que ayuda a abastecer y administrar las distintas funciones de red virtuales con NFV [21].

Funcionando de esta manera, se puede hacer que un proveedor de servicios active o desactive los sistemas virtuales que realizan las funciones de red dependiendo de sus necesidades, consiguiéndose un ahorro en el abaste-

cimiento de elementos de red y reduciendo diferentes costes, tanto de implementación (CAPEX) como de operación (OPEX).

En los últimos años se ha visto un gran aumento en el uso de NFV y esto viene apoyado por algunos beneficios que trae consigo su utilización como pueden ser; la reducción del coste, gracias a la centralización de los servicios en un mismo equipo; eficiencia operacional, permite una mayor rapidez en la implementación de nuevos servicios de red, aumentando la escalabilidad o la disponibilidad; innovación, al no usarse soluciones propietarias se tiene una mayor libertad para el desarrollo de aplicaciones o servicios nuevos.

- **ETSI NFV – MANO.** [22]

La ETSI es una organización europea de estandarización independiente, sin ánimo de lucro, encargada de las telecomunicaciones, la radiodifusión y otras redes y servicios de comunicaciones electrónicas.

En Noviembre de 2012, siete de los principales operadores de red del mundo fundaron el grupo ETSI ISG NFV (ISG, grupo de estandarización de la industria), convirtiéndose en la sede de la definición y consolidación de las tecnologías de virtualización de funciones de red.

Con el paso de los años la organización ha ido evolucionando a través de varias fases, los primeros esfuerzos en pruebas de concepto han evolucionado dando lugar a una serie de eventos de interoperabilidad. La comunidad sigue trabajando intensamente para desarrollar las normas necesarias para la transformación de NFV incorporando las últimas tecnologías, así como para compartir, de manera pública, las experiencias de implementación y pruebas de NFV en entornos con múltiples operadores.

La ETSI proporciona un marco para la arquitectura NFV, que se aleja del hardware patentado especialmente diseñado para un único servicio, hacia funciones de red que se entregan a través de la virtualización de software como funciones de red virtual (VNF). Este marco (véase figura 2.2) puede dividirse en bloques bien diferenciados [23]:

- Infraestructura NFV (NFVI): consiste en la base de la arquitectura general de NFV. Proporciona el hardware físico de computación, almacenamiento y redes que aloja los VNF. El hipervisor proporciona una capa de virtualización permitiendo cargas de trabajo que son in-

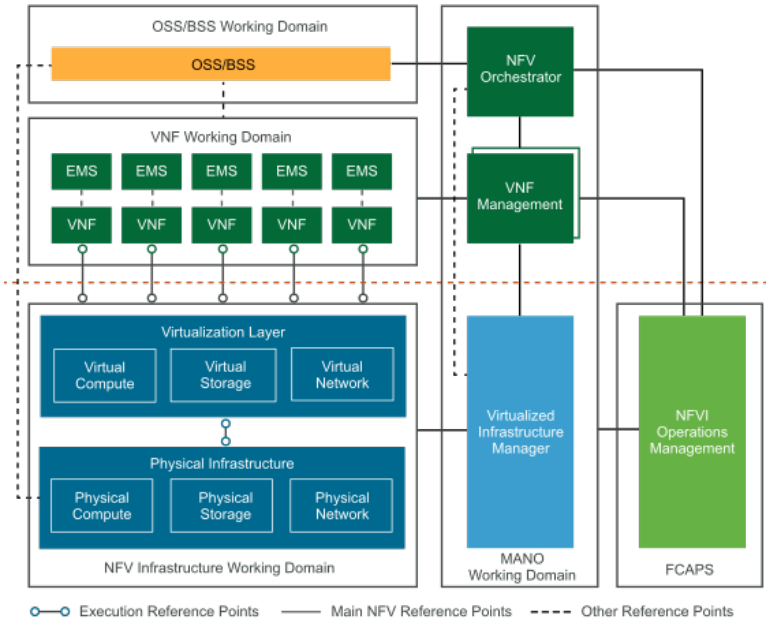

Figura 2.2: Marco de la arquitectura NFV según ETSI.[23]

dependientes del hardware subyacente. Esto permite la existencia de interoperabilidad entre los distintos componentes y las virtualizaciones de red.

- Gestión y orquestación (MANO): el bloque de gestión y orquestación (MANO) es el responsable de gestionar todos los recursos de la infraestructura, además de realizar la orquestación y gestión de los ciclos de vida de las VNF. Siendo capaz de interactuar con NFVI y con los VNF, es el encargado de gestionar los recursos físicos y/o de software de la virtualización.

- Funciones de red virtualizadas (VNF): consiste en el bloque que se encarga de utilizar las máquinas virtuales ofrecidas por el bloque NFVI, con la finalidad de construir sobre ellas las funciones de red virtualizadas, agregando el software que necesiten.

- Sistema de apoyo a operaciones y sistema de apoyo empresarial (OSS/BSS): se trata de una API que se puede consumir desde uno o varios sistemas de soporte de operaciones y sistema de soporte empresarial.

2.2. Administrador de Infraestructura Virtualizada (VIM).

El administrador de infraestructura virtualizada (VIM) es uno de los elementos clave del marco para NFV de la ETSI, siendo el responsable de gestionar los recursos de hardware y software que el proveedor de servicios emplea para crear cadenas de servicios y distribuir los servicios de red a sus clientes.

El VIM [24] es responsable de controlar y administrar los recursos de una NFVI - computación, almacenamiento, y recursos de red - basada en NFV. Es el bloque VIM es encargado de las siguientes funciones:

- Es el encargado de mantener un inventario de asignación de recursos virtuales, esto le permite realizar la orquestación de la asignación, actualización, liberación y recuperación de recursos en la NFVI y realizar una optimización sobre los mismos.
- Realiza organización de enlaces, redes, subredes y puertos virtuales.
- Administra las políticas de los grupos de seguridad para garantizar el control de acceso.
- Administra el repositorio de recursos hardware de la NFVI - computación, almacenamiento, redes - y recursos de software (hipervisores). Además, tiene capacidades para realizar optimización en su uso.
- Otras funciones que puede realizar un VIM pueden ser: recopilar información de rendimiento y errores, administrar imágenes de software, administrar catálogos de recursos virtualizados.

En el mercado actual hay muchos proveedores que ofrecen un VIM, sin embargo, hay que destacar que la mayoría de soluciones están basadas en la tecnología VIM de VMware u OpenStack. A continuación, se listarán algunos de los VIMs disponibles en el mercado.

A continuación, se listan algunos de los VIMs disponibles en el mercado.

- VMware vSphere [25].

  VMware vSphere se trata de un VIM patentado pero muy probado, líder en el sector para construir infraestructuras de cloud. Permite a

los usuarios ejecutar aplicaciones críticas para su negocio con mayor confianza y rapidez ante la variación en las necesidades empresariales.

vSphere está acelerando el cambio del sector IT hacia el cloud computing para los centros de datos existentes, además de ser uno de los sustentos de la nube pública. De esta forma, constituye el único modelo híbrido del sector.

Algunas ventajas de su uso:

- Eficiencia gracias a la utilización y a la automatización: consigue una mejora de la utilización del hardware hasta un 80 % sin perjudicar el rendimiento.
- Reducción drástica de los costes de IT: disminuye los gastos de propiedad en hasta un 70 % y los costes operativos en un 30 %.
- Agilidad con control: permite responder con mayor rapidez ante las necesidades empresariales en constante cambio sin sacrificar la seguridad ni el control, y proporciona una infraestructura sin contacto con disponibilidad, escalabilidad y rendimiento integrados y garantizado para las aplicaciones críticas.
- Libertad de elección: use una plataforma común basada en estándares para sacar partido a los activos existentes de TI junto con los servicios de TI de próximas generaciones

Por otro lado, tenemos las principales funciones y componentes de vSphere, mostradas en la figura 2.3.

Algo necesario que destacar es que VMware posee otra solución parecida pero que actúa sobre vSphere, denominado VMware Cloud Foundation [26]. Esta es una plataforma de nube híbrida para la gestión de máquinas virtuales y coordinación de contenedores, gracias a esto permite a las empresas aumentar la agilidad y flexibilidad de la empresa al tratar entornos de nube públicas y privadas. Sus ventajas son: gestión simplificada, arquitectura estandarizada, opciones de implementación flexibles, máxima escalabilidad y eficiencia del entorno IT, mejora del rendimiento de las aplicaciones y seguridad intrínseca.

Al fin al cabo VMware tiene alternativas muy parecidas pero con distinto enfoque vSphere, vCloud Suite o VMware Cloud Foundation.

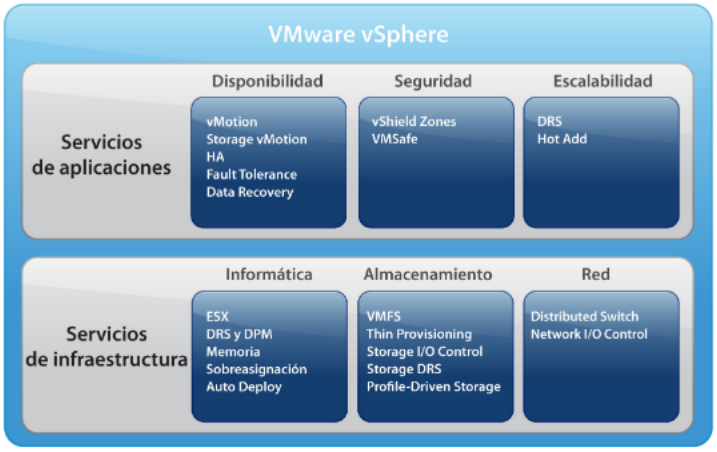

Figura 2.3: Servicios proporcionados por VMware vSphere.

- OpenStack. [27] [28]

  OpenStack es una plataforma de tecnología open source que utiliza recursos virtuales agrupados para diseñar y gestionar tanto nubes públicas como privadas. Está pensado para el control de grandes grupos de recursos informáticos, almacenamiento y redes en un centro de datos, todo ello gestionado y abastecido a través de APIs con mecanismos de autenticación. También incorpora un panel que habilita un control administrativo mientras que permite a sus usuarios suministrar recursos a través de una interfaz web.

  Además, del funcionamiento estándar de infraestructura como servicio, tiene componentes adicionales que proporcionan servicios de orquestación, administración de errores y gestión de servicios, gracias a ello, puede dar un servicio de alta disponibilidad de las aplicaciones de usuarios.

  OpenStack permite controlar una infraestructura como servicio, donde se pueden utilizar servidores bare-metal, máquinas virtuales o contenedores de manera indistinta, todo dependerá de los proyecto o componentes con los que se despliegue. Los “proyectos” es como se refiere a las diferentes herramientas que pueden desplegarse en OpenStack para proporcionar servicios de Cloud Computing.

  Funcionando de tal forma se consigue tener servicios on-premise, nube pública o incluso en el borde.

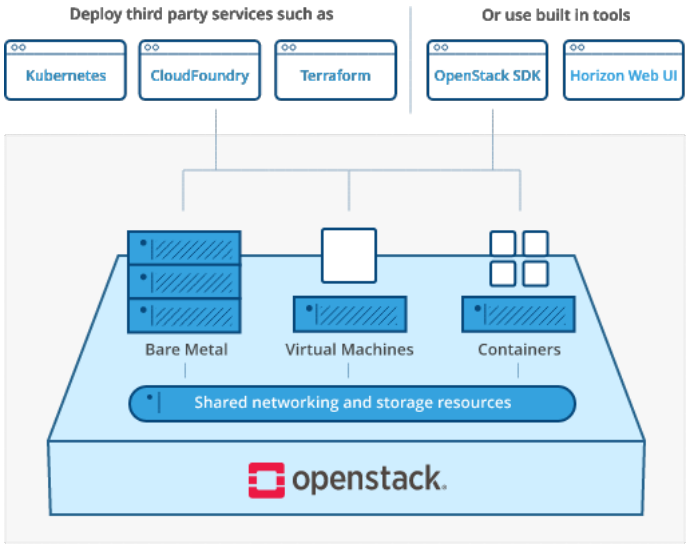

Figura 2.4: OpenStack.

Esta tecnología lleva muchos años desarrollándose a la par que evoluciona el sector tecnológico y por ello posee una amplia variedad de posibilidades de implementación. Hay empresas que venden sus servicios en la nube utilizando OpenStack y se encargan de mantener la infraestructura en las mejores condiciones de uso y optimizada, como Mirantis o RackSpace.

Pero si queremos montar un servicio nosotros, vemos que se puede realizar mediante Ansible, Tripleo, Chef o Charms y Juju, entre otros. Además, posee opciones de implementación para que los desarrolladores puedan realizar pruebas en una misma máquina física, DevStack o PackStack.

- Amazon Web Services. [29]

  Amazon Web Service, AWS, es la plataforma en la nube más adoptada y completa del mundo, dado que ofrece hasta 175 servicios integrales de centros de datos a nivel global. Millones de clientes están utilizando AWS para reducir los costes, aumentar su agilidad y fomentar la innovación de forma más rápida.

  AWS cuenta con una gran cantidad de servicios y de características incluidas que supera la de cualquier proveedor, ofreciendo desde tecnologías de infraestructura como cómputo, almacenamiento y base de datos hasta tecnologías emergentes como aprendizaje automático e

inteligencia artificial. Esto hace que llevar aplicaciones existentes a la nube sea más rápido, fácil y rentable, permitiendo crear casi cualquier cosa posible de imaginar.

AWS posee la funcionalidad más completa del mercado ofreciendo un amplio abanico de posibilidades dentro de cada una de los servicios que provee, de modo que resulta sencillo utilizar aquellas soluciones que nos resulten más adecuadas para nuestras necesidades para reducir costo y rendimiento.

Otras característica que dispone son la amplia comunidad de clientes y socios, la seguridad del entorno en el que se desarrolla, el ritmo de innovación y sobre todo la experiencia operacional que brinda.

El servicio de Amazon Web Services que nos brinda el servicio más semejante a VIM es Amazon Elastic Compute Cloud, Amazon EC2, un servicio web que proporciona la capacidad informática en la nube segura y de tamaño modificable, diseñado para simplificar el uso de la informática por parte de los desarrolladores.

- Cisco VIM. [30]

  Es una plataforma de Cisco para administrar el ciclo de vida completo del software y el hardware que componen la infraestructura NFV, manteniendo un inventario en tiempo real y un plan de asignación de recursos tanto físicos como virtuales.

  Entre las capacidades que ofrece están:

  - Gestión del ciclo de vida de la infraestructura virtualizada, tanto hardware como software, manteniendo un inventario con los recursos disponibles y controlando la capacidad de cómputo, almacenamiento y recursos de red
  - Optimización del rendimiento, optimizando los recursos y mejorando el rendimiento.
  - Diseño seguro, minimiza el área que puede ser atacada y valida los despliegues y operaciones.
  - Posee la capacidad de integrar aplicaciones de código abierto para registros y visualizaciones, observar el rendimiento del plano de datos, con los datos y el rendimiento de las máquinas virtuales y para controlar la "*salud*" de la nube.

- OpenShift. [31]

  Red Hat OpenShift es una plataforma de contenedores de Kubernetes con operaciones automatizadas integrales para gestionar implementaciones de nube híbrida y multicloud. Red Hat OpenShift está pensada y optimizada para mejorar la productividad de los desarrolladores y promover la innovación.

  Las ventajas que presenta OpenShift es el tiempo de ejecución de contenedores, conexiones de red, supervisión, registro y soluciones de autenticación y autorización. Es posible automatizar la gestión de los ciclos de vida obteniendo de esta forma una mayor seguridad, proporcionar operaciones de clúster fáciles de gestionar y dar portabilidad de las aplicaciones.

  Hace uso de Kubernetes de forma compatible y centrada en la seguridad incluyendo una base en Red Hat Enterprise Linux. Tiene la capacidad de poder desplegar software empresarial certificado tanto propietario como open source, gracias a su "*Marketplace*".

  En último lugar tiene cuatro posibles planes de contratación dependiendo de donde esté alojado el sistema: Red Hat Openshift Dedicado (alojado por Red Hat), Red Hat Microsoft Azure (Openshift alojado en Azure), Red Hat IBM (Openshift en la nube pública de IBM) y Red Hat OpenShit Container Platform (desarrollado sobre kubernetes en una infraestructura propia)

- CloudStack. [32]

  Apache CloudStack es un software open source diseñado para el despliegue y administración de grandes redes de máquinas virtuales, como una plataforma de computación en la nube de infraestructura como servicio. Múltiples proveedores de servicios emplean CloudStack para ofrecer servicios de nube pública y muchas empresas para proporcionar una oferta de nube local, privada, o como parte de una solución de nube híbrida.

  CloudStack es una solución preconfigurada que incluye todas las características principales que la mayoría de organizaciones desea para su nube IaaS (Infraestructure-as-a-Service): orquestación y contabilidad de recursos de red, computación y almacenamiento; redes como

servicio, administración de usuarios y cuentas, una API nativa completa y abierta, contabilidad de recursos y una interfaz de usuario.

Otras funcionalidades interesantes pueden ser la orquestación de servicios de red desde la capa de enlace (L2) hasta algunos servicios de la capa de aplicación (L7), como puede ser DHCP, NAT, Firewall, VPN, etc.

Una de las características más relevantes de CloudStack es su compatibilidad con los hipervisores más populares del mercado: VMware, KVM, Citrix XenServer, Xen Cloud Platform (XCP), Oracle VM Server y Microsoft Hyper-V. Además, posee una API compatible con AWS EC2 y S3 (servicios de Amazon Web Services) para organizaciones que deseen una nube híbrida.

- OpenVIM. [33]

  OpenVIM es una parte de OpenMANO y es una implementación ligera de un VIM que soporta funciones EPA, interactúa con los nodos de computación de la infraestructura NFV y un controlador de OpenFlow para proporcionar capacidades de computación y redes cuyo fin es desplegar máquinas virtuales. Además ofrece una interfaz "*northbound*", basada en REST y denominada "openvim API". Esta API ofrece servicios en la nube mejorados incluyendo la creación, eliminación y gestión de imágenes, sabores, instancias y redes.

- OpenNebula. [34] [35]

  OpenNebula es un entorno completo IaaS - basado en la externalización de las máquinas de procesado de datos y almacenamiento - open source soportado por una comunidad de desarrolladores. Surgió como proyecto de un grupo de investigación de la Universidad Complutense de Madrid y enfocado a la computación distribuida, visualización y plataformas de IaaS.

  Es una plataforma preparada para empresas que le ayuda a crear una nube privada y elástica. Hace uso de Kubernetes y Docker Hub para desplegar aplicaciones y se puede combinar con máquinas virtuales VMware y KVM con la finalidad de obtener nubes totalmente virtualizadas. Además, se pueden emplear recursos de infraestructura de AWS, Microsoft Azure y Packet, consiguiéndose una implantación de tecnología híbrida en la nube.

Dos características principales resultan al ser open source presenta una completa interoperatividad con los componentes de infraestructuras actuales y no depender exclusivamente de ningún tipo de hipervisor.

Una vez hemos visto las principales áreas de funcionamiento de cada VIM y hasta dónde pueden llegar, en cuanto a servicios se refiere podemos realizar una breve comparativa entre las distintas opciones para saber cuales son las más idóneas.

En primer lugar, no es de extrañar que desde un principio se haya comentado que muchos proveedores de cloud computing den sus servicios basados en las herramientas de VMware vSphere y OpenStack, puesto que junto a Amazon son las opciones con mayor capacidad de servicios disponibles. Ahora bien, Amazon tiene una infraestructura muy grande sobre la que proporcionar su servicios en la nube y su propuesta con Amazon Web Service engloba todos los servicios de los que dispone en la nube, siendo Amazon EC2 la más parecida a un VIM, pero no deja ser un servicio de pago y para un proyecto como se está desarrollando este no es una alternativa viable.

Más adelante se terminará de comentar la comparación entre vSphere y OpenStack, antes se explicará el resto de opciones.

En segundo lugar, tenemos distintas alternativas para ejercer de VIM como OpenVIM, OpenShift, OpenNebula, Cisco VIM, CloudStack. Estas se han descartado casi desde un principio porque son las que menos opciones de servicio ofrecen, es decir, no nos brindan todas las oportunidades de servicios o potenciales aplicaciones que pueden ser desplegados en alternativas como OpenStack. Puede deberse a que algunas son propietarias y no están destinadas a abarcar tanto terreno, o incluso hacen uso de otras alternativas como Cisco VIM hace uso de OpenStack, o porque lleven en el mercado menos años de desarrollo y prestando servicios.

Para finalizar el apartado, vamos a tener que elegir entre vSphere u OpenStack, y aunque uno de los principales elementos decisivos para quedarnos con OpenStack es su característica de software libre más grande del mercado. Tanto es así que la propia desarrolladora de vSphere, VMware, ha realizado su propia distribución de OpenStack para poder integrar en vSphere.

vSphere es una herramienta más destinada al ámbito de nube privada y se queda un poco corta en cuanto al mercado que puede llegar a abarcar una solución bien integrada de OpenStack que además tiene una característica de alta modularidad pudiendo integrar múltiples servicios distintos en función de las necesidades que se tengan. Es por ello que para este proyecto se ha elegido la opción de emplear OpenStack y esta decisión puede respaldarse viendo distintas empresas de alta repercusión que hacen uso de OpenStack: Walmart, Blizzard, Rakuten, CERN, IBM, BBVA, Santander, etc.

2.3. Administración y orquestación de virtualización de funciones de red (NFV MANO)

NFV MANO (management and orchestration) [36] es un marco definido por la ETSI - desarrollado por el grupo de trabajo ETSI ISG NFV- para la administración y orquestación de todos los recursos de red en un ámbito cloud que incluye recursos de cómputo, redes, almacenamiento y máquinas virtuales. El principal objetivo de NFV MANO es permitir una incorporación flexible y evitar el caos que puede surgir por la incorporación y rápida puesta en marcha de nuevos componentes de red.

NFV MANO está compuesto por tres bloques funcionales:

- Administrador de infraestructura virtualizada (VIM): es un bloque funcional de MANO y es el responsable de controlar, administrar y monitorizar el hardware de red, almacenamiento y computación de NFVI, el software de la capa de virtualización y los recursos virtualizados. También es el encargado de gestionar la asignación y liberación de recursos virtuales y la asociación de virtuales a físicos, incluyendo la optimización de recursos.

- Administrador de funciones de red virtual (VNFM) [37] : es el encargado de gestionar el ciclo de vida de las instancias de VNF, siendo el encargado de escalar, cambiar operaciones, agregar nuevos recursos y comunicar los estados de los VNF a la NFVO.

- Orquestador de virtualización de funciones de red (NFVO) [38] : es el bloque funcional encargado de realizar la incorporación de nuevos paquetes de VNF y Network Services (NS), la gestión del ciclo de vida

de los NS, la gestión de recursos y la gestión de la política para las instancias de NS.

Su funcionamiento es primordial para estandarizar las funciones de las redes virtuales con la finalidad de aumentar la interoperabilidad de recursos y la orquestación de servicios de red.

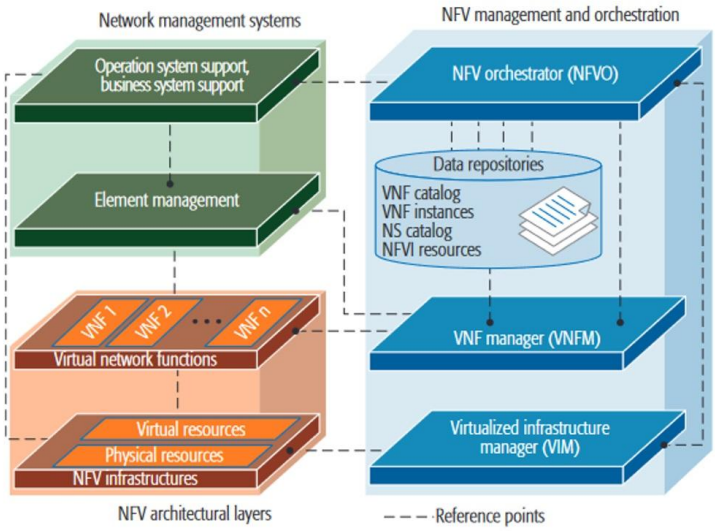

Figura 2.5: Marco NFV MANO. [39]

Conociendo el mercado en el que nos vamos a mover, a continuación pasaremos a ver algunas alternativas de herramientas que se pueden utilizar:

- OpenMano. [40]

  Es un proyecto de código abierto de Telefónica que proporciona una implementación práctica de la arquitectura de referencia para orquestación y gestión NFV propuesta por ETSI.

  Se trata de un modelo innovador que permite la creación sencilla y el desarrollo de complejos escenarios de red y que ha sido validado en entornos controlados de laboratorio.

  Telefónica apuesta por el código abierto favoreciendo a que tanto industria como desarrolladores investiguen los ámbitos de NFV.

  OpenMano proporciona tres módulos software:

- Openvim: es una implementación de referencia de un gestor de infraestructura virtualizada NFV VIM con soporte para un rendimiento alto y predecible. Posee una interfaz con los nodos de computación de la infraestructura NFV y con el controlador OpenFlow para proporcionar capacidades de computación , enrutado y despliegue de máquinas virtuales.

- Openmano: es una implementación de un orquestador de funciones virtualizadas de red, NFVO. Permite la creación de escenarios de red complejos. Posee una interfaz con el NFV VIM a través de su API y ofrece una interfaz "northbound" a través de la que ofrecer los servicios NFV incluyendo creación y borrado de servicios de red, VNFs.

- Openmano-gui: web interfaz gráfica de usuario que interacciona con la API de openmano de manera amigable. Además de esta opción también se dispone de una interfaz de línea de comandos.

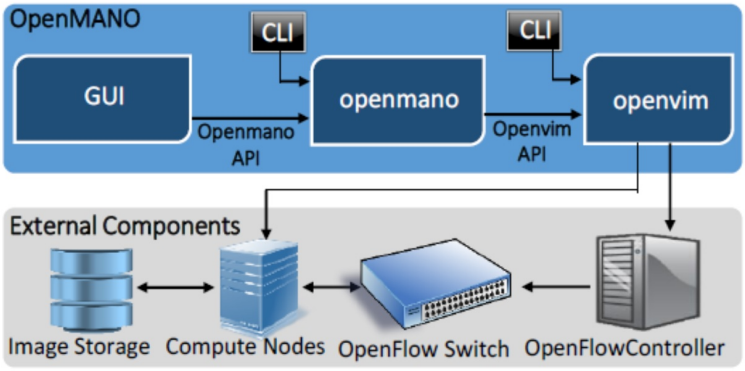

Figura 2.6: Arquitectura OpenMANO Telefónica. [39]

El proyecto de Telefónica OpenMANO se ha unido al proyecto Open Source MANO de la ETSI. Por lo que ha día de hoy es recomendable hacer uso de las versiones del nuevo proyecto.

- Open Source Mano. [41] [42]

  Open Source Mano es una iniciativa alojada por ETSI para desarrollar una pila de software Open Source NFV Management and Orchestration (MANO) fundamentada en ETSI NFV. OSM tiene como objetivo permitir que un ecosistema de proveedores de soluciones NFV entregue soluciones a sus usuarios de manera rápida y rentable.

ETSI OSM complementa el trabajo de ETSI NFV y viceversa. En particular, ETSI OSM brinda la oportunidad de capitalizar la sinergia entre la estandarización de los enfoques de código abierto al acceder a un conjunto mayor y más diverso de contribuyentes y desarrolladores de lo que normalmente sería posible.

Con este modo de actuación se maximiza la innovación, la eficiencia y el tiempo de comercialización y asegura una serie de continuas implementaciones de referencia en el sector.

El proyecto de OSM está siendo apoyado por grandes empresas tecnológicas líderes en el sector que ven el potencial que puede llegar a conseguir su infraestructura si utilizan la herramienta - Amazon, Telefónica, Intel, Whitestack, VMware, Canonical, Oracle, etc.

Al adoptarse el proyecto OpenMANO de Telefónica la arquitectura de OSM es la misma, compuesta por:

- VIM: encargado de gestionar la infraestructura NFV con soporte para operar en un alto rendimiento, optimizada y de manera optimizada.
- Open Source MANO: es el encargado de conectarse con el VIM para gestionar y desplegar VNFs dentro de la infraestructura, posee el catálogo y manejo de los VNFs que se pueden implementar y posee una comunicación directa con la interfaz API "*nortbound*".
- Open Source MANO UI: es una interfaz web para que el usuario pueda interactuar con la herramienta de forma gráfica. Adicionalmente, posee un cliente de línea de comandos para usuarios más experimentados.

Además, hay disponibles diferentes formas de instalación para simplificar la adopción de la herramienta e incorpora múltiples tecnologías punteras para realizar monitorización tanto de VNFs como de los VIMs que se tengan conectados.

Para instalarlo en una misma máquina emplea tecnologías de contenedores, Docker Swarm o Kubernetes, y por otro lado se puede instalar una distribución de Canonical empleado Charms y Juju.

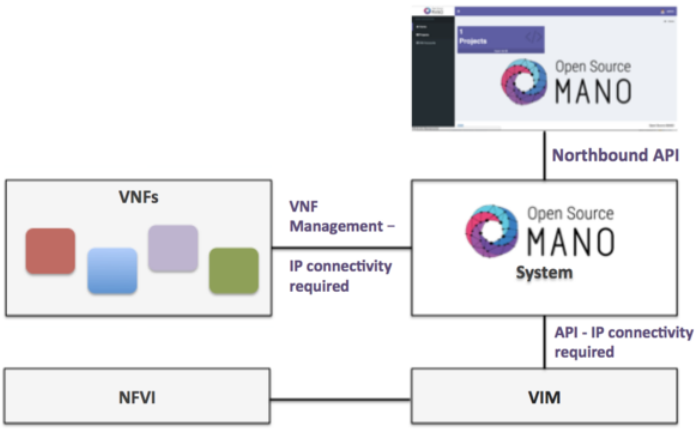

Figura 2.7: Arquitectura ETSI OSM.

- ONAP - Cloudify [43] [44]

  Open Network Automation Platform (ONAP) es una plataforma software de código abierto alojado por la fundación Linux.

  ONAP es una plataforma integral para la orquestación, gestión y automatización de servicios de redes de computación en el borde para operadores de red, proveedores de nubes y empresas, en tiempo real y basada en políticas. de esta forma permite automatizar rápidamente la instanciación y configuración de las funciones de red física y virtual y apoyar las actividades de gestión del ciclo de vida completo que es fundamental para las redes 5G y de próxima generación.

  Su objetivo principal es abordar las crecientes necesidades de una plataforma común con la capacidad de proporcionar una gestión de infraestructura eficiente y de extremo a extremo.

  ONAP consta de dos marcos arquitectónicos principales: entorno de tiempo de diseño y entorno de tiempo de ejecución, con numerosos subsistemas que operan dentro de ellos. El entorno de tiempo de diseño se utiliza como entorno de desarrollo con todas las funciones y bibliotecas necesarias para el desarrollo de nuevas capacidades. Por otro lado, el entorno de tiempo de ejecución, es un marco que se utiliza para ejecutar las políticas y reglas que se han preparado en el entorno de tiempo de diseño. Dichas políticas y reglas son responsables de una gran variedad de cosas, como puede ser la recopilación de datos, el análisis y el inventario de recursos.

- OpenBaton [45]

  Open Baton es una plataforma de código abierto resultado de un proceso de diseño ágil que tiene como principal objetivo el desarrollo de una implementación integral, extensible y personalizable capaz de orquestar servicios de red a través de infraestructuras NFV y siguiendo la especificación de la ETSI, NFV MANO.

  La versión actual ha incorporado múltiples funcionalidades nuevas. Puede gestionar un ecosistema diverso de VNFs, a través de su EMS genérico y VNFM genérico, componiéndolos en tiempo de ejecución en cualquier tipo de servicios de red. Se puede integrar con los VNFM existentes siguiendo un modelo *plug-and-play*.

- OPNFV[46]

  Open Platform for NFV - OPNFV - es una plataforma colaborativa de código abierto para la virtualización de funciones de red. Reduce el tiempo de integración e implementación de la infraestructura NFV y los VNF y CNF incorporados para los proveedores de componentes y operadores de estas plataformas.

  Esto se consigue a través de la integración, despliegue y pruebas a nivel de sistema, de varios ecosistemas de código abierto.

  OPNFV crea una plataforma de NFV de referencia para acelerar la transformación de las redes de empresas y proveedores de servicios.

Como se ha podido estudiar en este apartado hay variedad de herramientas que ofrecen soluciones para tecnologías NFV MANO, pero sin duda la más apoyada por la comunidad tanto en desarrollo como en clientes importantes del sector que hagan uso de ella es OSM. Cabe destacar que aunque no se haya mencionado, OpenStack posee un módulo para orquestación de NFV totalmente integrado, Tacker, pero tiene un rango de actuación limitado en comparación con el resto.

Por ello en este proyecto, y apoyando el uso de software de código abierto, se hará uso de ella para la interoperatividad con el resto de herramientas que se emplean.

Capítulo 3

Especificación de requisitos

En este capítulo se pretende realizar un análisis del escenario propuesto como solución en el presente proyecto para los objetivos planteados. Se expondrá, de la forma más detallada posible, los requerimientos fundamentales necesarios en las posteriores etapas de diseño e implementación de los servicios utilizados.

En esta fase del proyecto es necesario tener muy presente los diferentes objetivos que se persiguen con el proyecto. Estos nos darán una visión aproximada del escenario que se desea para actuar acorde a ello en el planteamiento de los requisitos.

A continuación, se pormenorizarán los requisitos de la solución y algunas decisiones previas tomadas al inicio del proyecto y que afectarán directamente a la realización del mismo.

3.1. Requisitos. Requisitos funcionales.

Resulta especialmente importante tener en cuenta las características necesarias para que las implementaciones de ambos servicios funcionen correctamente. Hay que prestar especial atención a la finalidad del proyecto, las condiciones iniciales de funcionamiento, etc.

- Especificación equipos servidores: Los equipos utilizados para instalar los diferentes servidores del escenario planteado necesitan una serie de recursos mínimos bien definidos. A continuación, se diferenciarán los requerimientos para cada uno de los PCs empleados.

  - Servidor OpenStack: Son necesarios al menos 2 procesadores, 8 GB de RAM y 60 GB de almacenamiento. Se instalará sobre una instalación mínima de Ubuntu 18.04 con conexión a internet, al menos una interfaz de red. Para realizar la instalación se requiere un usuario con permisos *"sudo"*.

  - Servidor OSM: Partiremos de los requisitos recomendados en la documentación oficial. Se necesita Ubuntu 18.04 en su versión de 64 bits, al menos 2 procesadores, 8 GB de RAM y 40 GB de almacenamiento, además de una interfaz de red con acceso a internet.

- Funcionamiento servidores: Por la naturaleza de los distintos servicios y el funcionamiento de los servidores será necesario tener los equipos en alta disponibilidad, es decir, encendidos y conectados de manera continua para poder realizar operaciones en ellos en cualquier momento.

- Conexión entre equipos: En el escenario diseñado será estrictamente necesario que los diferentes servidores empleados estén comunicados entre ellos de manera constante. El servidor de OSM deberá de tener asociado el servidor de OpenStack porque lo utiliza como VIM para desplegar las VNFs.

- Autenticación y autorización de clientes: Los clientes se tienen que autenticar antes de utilizar los servicios de OpenStack, de otro modo, no podrá realizar ninguna acción que requiera la creación de nuevos servicios.

- Monitorización de los servicios: Es necesario que los servicios implementados sean monitorizados, ya que principalmente se quieren levantar servicios en el menor tiempo posible cuando haya algún problema en alguno que se encuentre activo. Además, sería interesante que el servidor que tiene OSM posea la capacidad de controlar la disponibilidad del servidor OpenStack, entiéndase como monitorizar su funcionamiento para saber si puede seguir utilizándolo como VIM o no.

3.2. Decisiones previas. Requisitos no funcionales.

Este apartado expondrá algunas decisiones previas tomadas para el diseño del escenario, se incluirán aquellas características que agregan un atributo o imponen una restricción específica a la implementación de los servidores.

Se tratará la elección de las diferentes soluciones software para cada uno de los servicios que se desean desplegar en el marco del proyecto.

- OpenStack: El uso de OpenStack como VIM será elegida en este proyecto, esto nos pone una gran parte de requisitos funcionales anteriores, pero también una gran ventaja dada su gran modularidad gracias a la separación de los distintos servicios disponibles en módulos o elementos.

- Open Source MANO: El servicio de NFV MANO empleado es Open Source MANO, dado que sigue los estándares de la ETSI y está altamente apoyado por grandes potencias tecnológicas del sector. Además, su gran grado de adecuación a un escenario heterogéneo es idóneo para el futuro que se espera en el ámbito de la virtualización y las comunicaciones futuras.

- Ubuntu 18.04.4 LTS: El sistema operativo elegido es Ubuntu 18.04.4, esta decisión ha sido necesaria porque la documentación de OpenStack, en su versión DevStack, nos indica que es la versión más probada hasta el momento de manera oficial y, por otro lado, en el caso de OSM es un requisito para el correcto funcionamiento del servidor.

- Pocos recursos: Dada la naturaleza y el escenario que se desea plantear en el proyecto debemos tener en cuenta un escenario de recursos altamente limitados donde desplegar todos las herramientas y aplicaciones.

- Escalabilidad: El sistema se desarrollará de tal manera que sea posible esca-larlo en caso de que fuera necesario. Un ejemplo de ello es que OSM pue-de operar con varios VIM a la vez si están definidos y este tiene conexión con ellos.

- Interoperabilidad: El sistema desarrollado tiene que conectar los dos servicios y en la medida de lo posible tiene que ser un servicio que se pueda consultar y operar desde otros dispositivos con conexión a ellos siempre que se autentiquen en los servicios. Se podrá probar que se puede acceder desde otros equipos para ver que efectivamente estos servicios se pueden lanzar por ejemplo desde un móvil o tablet.

- Visibilidad: En conexión con el punto anterior, al tratarse de un servicio que se quiere monitorizar u actuar sobre el en el menor tiempo posible es necesario que sea visible para otros equipos de la red y que se pueda acceder a las distintas GUI en los servidores web.

- Usabilidad: Se deberá de facilitar la utilización de los diferentes elementos del sistema por los usuarios potenciales, para ello se realizará un manual de usuario como Anexo de la memoria y en la medida que se requiera se implementarán script que simplifiquen las implementación de servicios o funciones en los servidores.

Capítulo 4

Planificación y estimación de costes

En este capítulo se recoge toda la información referente a la planificación realizada para la consecución de los objetivos planteados en el trabajo. Para mejorar su entendimiento se puntualizarán las distintas fases que lo han compuesto.

Se tratará de representar todos los recursos que han sido necesarios para la realización de la instalación, configuración, puesta en marcha, operación y desarrollo de la memoria.

Finalizando con una estimación del coste asociado a la ejecución de un proyecto de este tipo, teniendo en cuenta todas las fases de desarrollo y los distintos recursos necesarios.

4.1. Fases de desarrollo

En esta sección del capítulo se recogerá un análisis de las diferentes fases que componen el proyecto, exponiendo los aspectos más significativos de cada una de ellas y el tiempo necesario para su consecución y estimación del coste aproximado de la totalidad del proyecto. Además, se representarán un diagrama de Gantt para visualizar una planificación aproximada de cada una de las fases.

A continuación, se realizará la enumeración de las diferentes fases con una breve descripción:

- Fase 1: Definición del proyecto.

  En esta primera fase se intentará realizar una definición completa del trabajo que se desea realizar, para ello será necesario plantear los distintos objetivos que se quieren satisfacer con la elaboración del mismo, estudiar de forma superficial las posibilidades que nos brinda el actual mercado tecnológico para elegir las posibles herramientas o tecnologías. Por lo tanto, es en esta fase en la que se acabará planteando cuáles serán los diferentes elementos de nuestra solución, OpenStack como VIM y Open Source MANO como orquestador de NFVs.

- Fase 2: Especificación de requisitos.

  Cuando ya se han planteado todos los objetivos que se desean conseguir con el presente proyecto, tendremos que fijar una serie de requisitos para que todo funcione correctamente y como es esperado. Estos los tendremos que diferenciar entre requisitos funcionales y no funcionales, de esta forma, en el momento que se hayan concluido tanto esta etapa como la anterior podremos decir que han quedado fijadas las bases del trabajo fin de máster sobre las que empezar a trabajar de un manera más sólida.

- Fase 3: Estudio de alternativas disponibles

  Esta fase es una de las más importantes y ya se ha tenido una toma de contacto con ella en la fase 1, mediante el estudio del mercado para analizar las posibles alternativas disponibles que proveen un servicio similar al que se pretende exponer en esta memoria. Durante este análisis se realizará una comparación de las posibles soluciones teniendo en cuenta las ventajas e inconvenientes presentes en cada una. En el caso de que fuera necesario y/o posibles se verá el coste asociado a estas opciones.

- Fase 4: Familiarización con las herramientas utilizadas.

  En esta fase se desarrollará un estudio exhaustivo de las diferentes herramientas empleadas. En primer lugar, se empezará estudiando la documentación oficial de cada una de las herramientas que serán utilizadas y posteriormente, se intentará realizar una serie de pruebas para tener una toma de contacto con las soluciones que ofrece ante la problemática que tenemos como objetivo.

- Fase 5: Diseño.

  Esta fase es importante para realizar el diseño de manera correcta de un contexto que pueda cumplir los objetivos que tenemos planeados. Es primordial establecer los distintos elementos que compondrán el escenario de la solución propuesta y sobre la que tendremos que realizar una serie de pruebas y comprobar que efectivamente se están cumpliendo todas las premisas planteadas.

- Fase 6: Implementación.

  Esta es una de las fases del proyecto más costosa, en términos de tiempo, porque consistirá en el desarrollo en sí del escenario que se ha planteado en etapas anteriores. Estará compuesta por las tareas de instalación y configuración de cada una de las herramientas, OpenStack y Open Source MANO, en los dos equipos que se utilizarán.

  Además, de la instalación será necesario hacer que ambos equipos con sus respectivos servicios interactúen entre ellos, y hay que ir verificando poco a poco que cada elemento funciona correctamente antes de seguir con la configuración para que se comuniquen entre servicios, de esta forma se enlaza de alguna forma con la siguiente fase.

- Fase 7: Pruebas y validación.

  Esta etapa conlleva la realización de una batería de pruebas para cerciorarse del correcto funcionamiento de los diferentes elementos que conforman la implementación.

  Algunas de ellas, es necesario realizarlas conforme se van instalando y configurando los distintos servicios y elementos. Estas consistirán principalmente en comprobar que los distintos equipos son capaces de comunicar sus servicios dentro de la red interna que se está utilizando.

  La batería de pruebas será realizada cuando el escenario se encuentre completo, donde se verá el funcionamiento de la funcionalidad completa de la alternativa propuesta en este proyecto.

- Fase 8: Redacción de la memoria y revisión bibliográfica.

  La última fase antes de finalizar el proyecto es la realización de la memoria, aunque es una etapa que se extiende desde el inicio de la primera fase junto con la revisión bibliográfica.

Se recogerán los aspectos más relevantes para facilitar el entendimiento de cada una de las decisiones tomadas y los pasos dados durante la realización de todo el proyecto. Además, se expondrá una contextualización teórica para situar al lector, un apartado con la realización práctica de todas las configuraciones realizadas y todas las referencias bibliográficas consultadas.

Habiendo concluido breve explicación de todas las fases que han compuesto este proyecto, se procederá con presentación de la planificación de cada una de las fases. Con ella se podrá estimar el tiempo empleado en la elaboración de cada una. Esta planificación puede apreciarse gráficamente en un diagrama de Gantt. (Véase figura 4.1).

En el diagrama de Gantt se puede observar la cantidad de horas necesarias para realizar cada una de las tareas a realizar en el presente trabajo. Algunas de ellas se han prolongado durante, prácticamente, todo el proyecto teniendo que compaginarse con otras tareas. Partiendo de esto, podemos esperar que las más relevantes son la redacción de la memoria y la familiarización con las herramientas que quieren emplear.

En la tabla (4.1) se puede contemplar el número de horas asignadas a cada una de las diferentes tareas que compone el presente proyecto.

Asignación horaria de las etapas del proyecto		
Fase de realización	Descripción	Tiempo aproximado
Fase 1	Definición del proyecto	10 Horas
Fase 2	Especificación de requisitos	15 Horas
Fase 3	Estudio de alternativas disponibles	30 Horas
Fase 4	Familiarización con las herramientas utilizadas	80 Horas
Fase 5	Diseño	35 Horas
Fase 6	Implementación	65 Horas
Fase 7	Pruebas y validación	35 Horas
Fase 8	Redacción de la memoria y revisión bibliográfica	220 Horas

Tabla 4.1: Estimación temporal del proyecto.

La totalidad de horas dedicadas a la realización del presente Trabajo Fin de Máster, teniendo en cuenta todas las fases que lo componen y con un recuento aproximado asciende a 490 horas.

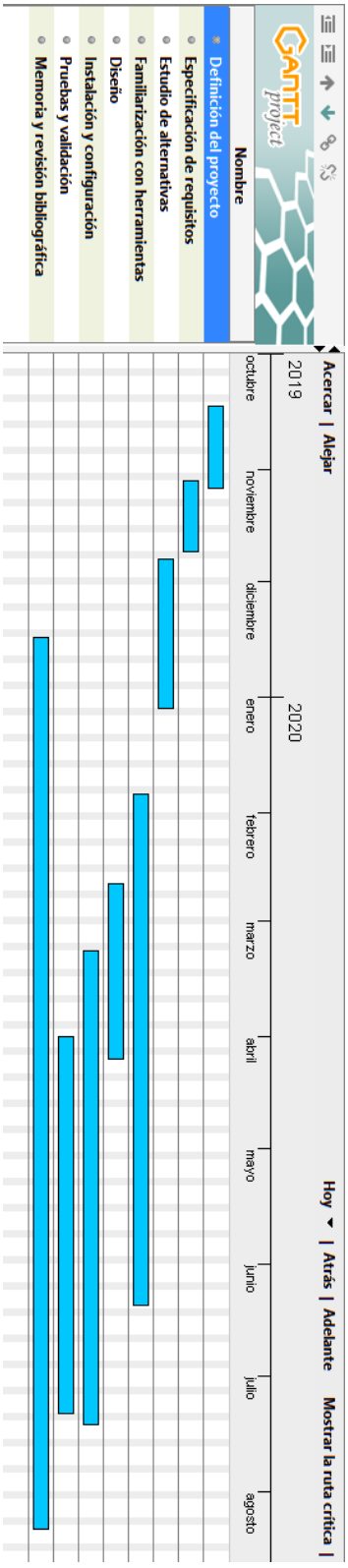

Figura 4.1: Diagrama de Gantt.

4.2. Recursos

Este apartado contendrá una exposición de todos los recursos utilizados durante la realización del presente estudio, abarcará tanto recursos humanos como recursos hardware y software.

4.2.1. Recursos humanos.

- D. Juan José Ramos Muñoz, en calidad de tutor del proyecto y profesor contratado doctor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

- D. Jorge Navarro Ortiz, en calidad de tutor del proyecto y profesor contratado doctor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

- Alejandro García Soria, alumno del Grado en Ingeniería de Tecnologías de Telecomunicación y autor del presente proyecto.

4.2.2. Recursos hardware.

- HP Envy 15 j006ss, dispone de un procesador Intel Core i5-3230M (2,6 GHz, 3 MB L3 de caché), 8 GB de memoria RAM y 750 GB de disco duro.

- MSI GL65 9SEK, dispone de un procesador Intel Core i7-9750H+HM370 (4,5 GHz, 12 MB de caché), 16 GB de memoria RAM y 512 GB de disco duro SSD.

4.2.3. Recursos software.

- Sistema operativo Ubuntu 18.04.4 LTS, en ambos equipos. Empleado para la instalación, configuración y comprobación de los servicios de OpenStack y Open Source Mano.

- OpenStack, en concreto la Release Ussuri de la distribución DevStack. Utilizado como Virtualized Infraestructure Manager (VIM) en la nube para desplegar máquinas virtuales.

- Open Source Mano, release SEVEN. Se emplea como orquestador de NFVs y estará comunicado con OpenStack.

- Docker. Es utilizado para lanzar aplicaciones en contenedores automatizados para el despliegue de aplicaciones.

- Kubernetes. Se emplea para realizar clústeres de contenedores Docker con la finalidad de desplegar aplicaciones y tener control de réplica.

- Grafana. Programa empleado para la visualización y monitorización de métricas.

- Timeshift. Se emplea como programa para realizar backup o snapshots del sistema operativo para ir controlando errores con puntos de restauración.

- Creately. Servicio web online para crear diagramas de red.

- Sublime Text 3, como editor de texto para modificación de algunos scripts.

- Overleaf. Editor online de texto en LaTeX.

4.3. Coste del proyecto

En esta sección se realizará un análisis del coste asociado al proyecto, teniendo en cuenta el estudio, la implementación y la reproducción bibliográfica. Será necesario tener en cuenta cada uno de los elementos que lo componen y las diferentes fases de desarrollo.

En primer lugar, nos centraremos en los recursos humanos empleados:

- Valor del trabajo autónomo de un ingeniero del máster en Ingeniería de Telecomunicaciones, aproximadamente 25 euros/hora.

- Trabajo de un profesor Doctor en la Universidad de Granada, aproximadamente euros/hora.

En la 4.2 se recoge detalladamente una estimación del coste de cada una de las fases del proyecto, teniendo en cuenta las horas indicadas en apartado 4.1.

Tareas	Tiempo empleado	Coste
Definición del proyecto	10 Horas	250 €
Especificación de requisitos	15 Horas	375 €
Estudio de alternativas disponibles	30 Horas	750 €
Familiarización con las herramientas utilizadas	80 Horas	2000 €
Diseño	35 Horas	875 €
Implementación	65 Horas	1625 €
Pruebas y validación	35 Horas	875 €
Redacción de la memoria y revisión bibliográfica	220 Horas	5500 €
Tutorias	20 Horas	2000 €
	Coste total:	14250 €

Tabla 4.2: Coste asociado a los recursos humanos.

En segundo lugar, nos centraremos en el coste debido a los recursos hardware y software, aunque en este punto es necesario indicar que el software utilizado en el proyecto es open-source y las versiones empleadas son gratuitas. Además, al haberse utilizado Ubuntu como sistema operativo el coste asociado al proyecto es inferior.

En la tabla 4.3 se recoge el precio asociado a cada uno de los recursos empleados en la implementación del proyecto. En ella veremos que, exclusivamente, supone una cuantía definida el hardware asociado al desarrollo práctico de los distintos servidores.

Recursos	Coste	Ciclo de vida	Coste
Ordenador HP Envy 15	800 €	60 meses	134 €
Ordenador MSI GL65 9SEK	1600 €	6 meses	1600 €
		Coste total:	1734 €

Tabla 4.3: Coste asociado a los recursos hardware y software.

Para concluir este capítulo, se realizará el cálculo del presupuesto total necesario para la realización íntegra del proyecto. Tendremos que sumar la cuantía que suponen los recursos humanos y los recursos del hardware y software, detallado en el apartado 4.2.

Por lo tanto, el presupuesto final del proyecto sabiendo que el valor de los recursos humanos es de 14250 euros y de hardware de 1734 euros, asciende a un montante de 15984 euros.

Capítulo 5

Tecnologías y herramientas utilizadas.

En este capítulo se llevará a cabo un estudio teórico de las diferentes herramientas y tecnologías empleadas en las fases posteriores de diseño, instalación y configuración y pruebas de validación. Se abarcarán una explicación de las tecnologías o software utilizado para funcionar con virtualización de equipos, empleo de contenedores, para desplegar ciertas funciones o servicios, y orquestar y controlar acorde a los modelos de información del ETSI NFV.

Se expondrán los datos que se consideren más relevantes sobre OpenStack y Open Source MANO para facilitar la comprensión de cómo funcionan y están estructurados cada uno. Es de mencionar, que estos entornos funcionan conectados. Se utilizará OpenStack para instanciación de infraestructuras de máquinas virtuales y contenedores porque presenta grandes ventajas para poder monitorizar, actuar y gestionas, en general, todos los recursos hardware disponibles para proveer el servicio.

Por otro lado, para gestionar configuraciones y despliegues de servicios de red y diferentes funciones de red virtuales, se empleada Open Source MANO porque tiene una serie de ventajas compartidas con la herramienta anterior y los distintos servicios de esta herramienta pueden ser gestionados por OpenStack. Además, está consiguiendo un gran apoyo por la comunidad tanto pública como privada desde hace algunos años.

5.1. Open Source MANO

Open Source Mano (OSM) [45][46] es una comunidad de código abierto apoyada por el ETSI y regido por los estándares ETSI NFV, ofrece una pila de MANO con calidad para emplear en producción para NFV, capaz de consumir modelos de información publicados abiertamente, adecuados para todos los VNF, operacionalmente significativos e independientes de VIM.

OSM ha ido lanzando versiones diferentes, incorporando nuevas funcionalidades y mejorando y optimizando el funcionamiento de las existentes. En la actualidad se ha presentado la versión ocho.

Dada la alta repercusión y apoyo mostrado en la comunidad se está erigiendo como la alternativa más completa e importante del mercado tecnológico para la administración y orquestación de servicios en Infraestructuras NFV.

La arquitectura de OSM se basa en una serie de principio seguidos desde la fundación del proyecto y su evolución:

- Separación en capas: es necesaria una clara delimitación de las capas o modelos que forman cada capa estando siempre fundamentadas en los modelos de ETSI NFV y las distintas recomendaciones que realizan. esto favorece la posible sustitución de elementos en las capas (SO, RO, VIM, etc.), haciendo que ninguna sea dependiente de un elemento concreto.

- Abstracción: debido a que se basa en los estándares de ETSI-NFV, se debería ofrecer una clara diferenciación en los niveles de abstracción y detalle presentados en el modelo de información y su uso dentro de las capas.

- Modularidad: se prefiere una característica clara de modularidad basada en el modelo plug-in, para facilitar la sustitución de los módulos o ir incorporando nuevos con el desarrollo y la evolución.

- Simplicidad: la solución que se desarrolle deberá de tener la mínima complejidad posible para implementar los modelos de información. La pila de OSM deberá de evolucionar a la par que lo hace el modelo de información, impulsando un control adecuado sobre la arquitectura.

5.1.1. Arquitectura de OSM.

Con el lanzamiento de cada versión nueva de OSM, su arquitectura principal puede verse modificada, buscando ser cada vez una solución más abierta y disponer de un proceso de integración de nuevos módulos más simple. La arquitectura de las últimas versiones es mostrada en la figura 5.1.

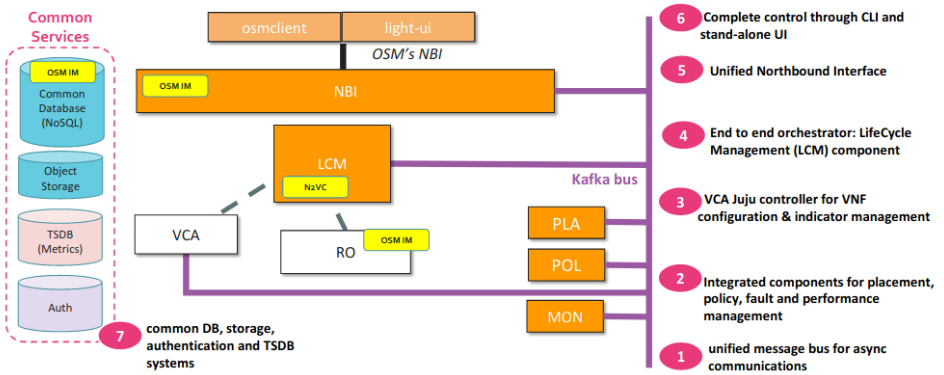

Figura 5.1: Arquitectura OSM Release SEVEN. [45]

Si estudiamos la vista de la arquitectura de OSM, observaremos que se compone de:

- Interfaz de Usuario: nos proporciona la posibilidad de interactuar de manera gráfica con el entorno de OSM, pudiendo usarse para controlar las operaciones del ciclo de vida en VNFs y servicios de red. Esta interfaz nos da información en tiempo real de VNFs, servicios de red y vistas detalladas de las topologías y nos permite actuar sobre el catálogo de paquetes VNF y NS (*network service*) que tenemos disponibles para su despliegue en nuestra instalación de OSM.

  Adicionalmente, se incorpora un cliente en línea de comandos para interactuar con la *API Rest Northbound* de OSM.

- Orquestador de servicios (SO, *Service Orchestrator*): es el módulo responsable de todos los aspectos relacionados a la orquestación de servicios, incluyendo la gestión del ciclo de vida y la ejecución de primitivas sobre el servicio. Se trata del componente "maestro del sistema que rige el flujo de trabajo de todo OSM.

  Es el responsable de los proyectos, usuarios, de hacer cumplir los controles de acceso basándose en los roles definidos. El almacén de datos

es el encargado de mantener persistentemente el estado de SO.

Además, posee el motor de composición de servicios de red, encargado la composición y validación de los descriptores tanto de VNF como de servicios de red (NS)

- Network Service to VNF Communication (N2VC): es responsable del marco de plugins entre el orquestador de servicios, SO, y la capa de configuración y abstracción del VNF, VCA.

- Configuración y Abstracción VNF (VCA): es la capa responsable de habilitar las configuraciones, acciones y notificaciones hacia o desde los VNF. En el caso de implementar Juju, se facilita la posibilidad de crear VNFM genéricos o específicos. (VCA, *VNF Configuration and Abstraction*)

- Orquestador de recursos (RO, *Resource Orchestrator*): este módulo se encarga de procesar la asignación de recursos del VNF según se indique en los descriptores y de administrar y coordinar las asignaciones de recursos a través de múltiples VIMs, que pueden estar geodistribuidos y disponer de múltiples controladores SDN. Los plugins para VIM y SDN son encargados de conectar el RO con la interfaz específica para cada VIM y controlador SDN.

- Monitorización (MON): se trata de un módulo que debe interactuar con los sistemas de control y vigilancia existentes y tener capacidad de adaptarse a nuevos. Una de las principales características y de mayor potencia que ofrece el módulo de monitorización de OSM es la habilidad de correlacionar telemetrías relativas a los VMs y VNFs con servicios de red relevantes. Un avance más sería utilizar la correlación automatizada para proporcionar una mejora de la experiencia de los usuarios de OSM y conseguir un aumento de la eficiencia de los operadores o proveedores en el entorno de las telecomunicaciones.

- Modelos de información OSM (OSM IM): fue desarrollado para ser el único punto de autoridad en el modelo de datos OSM que es aprovechado por los diferentes componentes. Esto favorece a que los dos modelos de datos importantes, descriptor VNF (VNFD) y de servicios de red (NSD), puedan ser compartidos entre los componentes.

- API Northbound (NBI): se trata de una interfaz RESTful, sigue el estándar "*ETSI SOL005 standard*". Admite formatos YAML y JSON.

Es una manera de comunicar los componentes facilitando la interoperabilidad. Interactúa con la interfaz de usuario, que se comunica con ella de forma remota.

5.1.2. Tecnologías soportadas.

En las últimas versiones se han incorporado tecnologías de contenedores como Docker o Kubernetes con la finalidad de facilitar la implementación y despliegue de OSM, además de incorporar la posibilidad de desplegar los VNF y NS en contenedores en lugar de máquinas virtuales, denominándose VNF basados en contenedores (CNF) o VNF basado en Kubernetes (KNF).

El empleo de estas tecnologías se realiza con la finalidad el tiempo de despliegue de los distintos servicios de red, pensando en aquellas aplicaciones críticas que no puedan tardar minutos en desplegarse o necesiten ejecutarse en un entorno de pocos recursos como *Edge computing*.

Por otro lado, en el caso de la monitorización, OSM emplea Apache Kafka como bus de comunicación, se trata de un sistema tolerante a fallos y basado en el modelo publicación-suscripción.

Adicionalmente para las tareas de monitorización se emplea Prometheus como TSDB (*base de datos de series temporales*), la información registrada por esta herramienta es mostrada por Grafana, que proporciona una interfaz gráfica para facilitar el control del sistema. En la figura 5.2 se puede ver la interconexión de estas herramientas para su correcto funcionamiento.

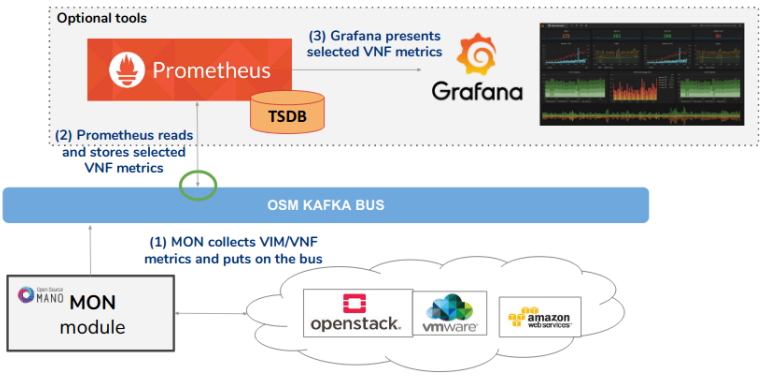

Figura 5.2: Interconexión de herramientas de monitorización de OSM.

5.2. Openstack

OpenStack [47] es un proyecto de computación en la nube que controla grandes grupos de recursos informáticos, almacenamiento y recursos de red en centros de datos, todo esto administrado y abastecido a través de una API que posee mecanismo comunes de autenticación.

Proporciona un panel de control para administradores y usuarios finales mediante una interfaz web.

Se trata de un proyecto pensado para proporcionar IaaS (*infraestructura como servicio*), pero incorpora una serie de servicios adicionales que permiten realizar orquestación, monitorización, gestión de fallos y control de recuperación ante catástrofes para garantizar servicios de alta disponibilidad.

OpenStack es una herramienta que se divide en servicios diferentes y permite conectarlos en función de las necesidades. La figura 5.3 muestra el mapa de los diferentes módulos que engloba OpenStack y cómo pueden trabajar juntos.

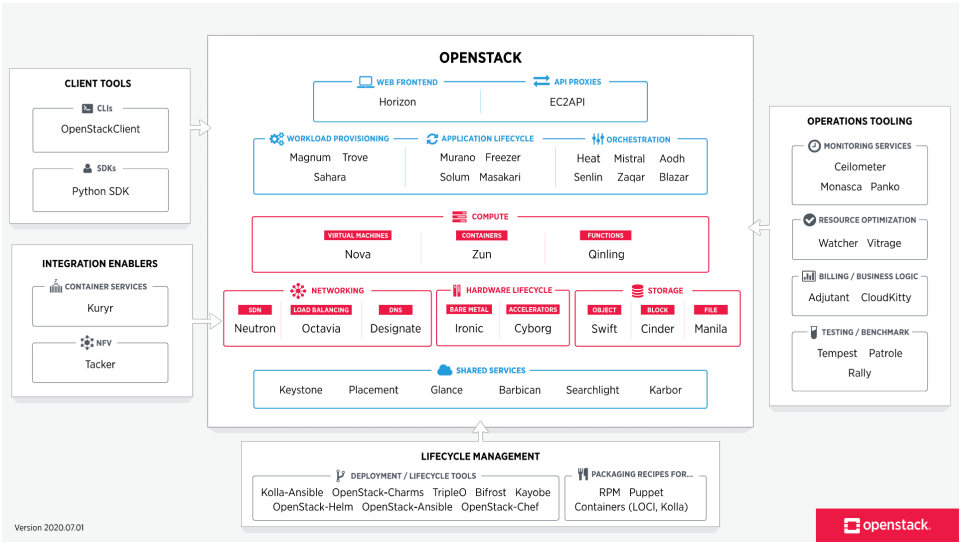

Figura 5.3: Arquitectura servicios OpenStack.

Algunos módulos mostrados en la arquitectura son requeridos para un correcto funcionamiento del sistema OpenStack. A modo de resumen se pueden mencionar algunos importantes. Los distintos servicios hacen uso del

módulo de autenticación llamado "*keystone*". Nova es el encargado de proveer los recursos para las máquinas virtuales, glance contiene las imágenes de sistemas operativos, cinder proporciona los volúmenes definidos para almacenamiento de los datos de instancias, swift se encarga de almacenar los volúmenes en forma de objetos, neutron, uno de los más importantes, es el encargado de gestionar todos los servicios de conectividad para la capa de red y en último lugar podemos hacer uso de ceilometer para realizar monitorizaciones de recursos.

Cada uno de estos servicios provee una API mediante las que se comunican y mantiene los datos relevantes en una base de datos. Adicionalmente, se incorpora un cliente de línea de comandos, realizado en Python, encargado de operar sobre las APIs de los servicios y una interfaz web que permite gestionar toda la infraestructura y los servicios de manera gráfica.

OpenStack es una herramienta con un amplio recorrido en el sector del cloud computing por ellos tiene múltiples versiones a las que han ido añadiendo funcionalidades diferentes acorde a la evolución de la tecnología.

Cabe destacar que posee una amplia variedad de distribuciones diferentes para realizar la implementación y despliegue en nuestro entorno, tanto propietarios y, por lo tanto, de pago como gratuitas pero empleando diferentes herramientas para su despliegue.

Es posible implementar el entorno de OpenStack tanto en diferentes máquinas como en una misma, para ello podremos emplear herramientas basadas Helm, Ansible, Charms y Juju o de manera contenerizada en equipos bare-metal.

Para poder realizar un despliegue rápido, pensado para que los desarrolladores puedan realizar pruebas, existen las versiones DevStack que funciona en distribuciones de Ubuntu y PackStack, funciona en CentOS.

A continuación, para definir y entender mejor el funcionamiento de OpenStack y sus servicios, se va a realizar una explicación de cada uno de los módulos que forma parte de esta herramienta y que se han considerado interesantes para el proyecto presente. Con la finalidad de facilitar su comprensión se va a dividir dicha explicación en dos apartados, refiriéndonos a aquellos servicios que estarían disponibles en una instalación por defecto y otros que se han considerado apropiado para añadirlos en el presente trabajo fin de máster, ambos casos referidos a la opción de despliegue DevStack.

5.2.1. Instalación por defecto.

La instalación más simple de Openstack, como DevStack, está formada por los servicios Nova, Neutron, Keystone, Cinder, Glance, Horizon, Placement, y aunque no sea incluido por defecto, en la mayoría de instalaciones de prueba y plantilla de ejemplo de la documentación se añade Swift.

- **Keystone:** es un servicio de OpenStack que proporciona una autenticación de clientes API, descubrimiento de servicios y autorización distribuida mediante una API de identidad. Mediante la autenticación permite confirmar la identidad del usuario que está intentando acceder y mediante la autorización se pretende controlar qué acciones tiene permitidas y cuáles no. Hay diferentes formas de interactuar con los servicios de OpenStack y por lo tanto de autenticarse, mediante interfaz web, por línea de comandos o directamente haciendo uso de la API. Una de las propiedades principales que se utilizan en este servicio es el rol de cada usuario para saber si se trata de un usuario normal o se trata de un administrador.

  Para poder interactuar con Openstack desde otro servicio o sistema es necesario que se pueda alcanzar el servicio de indentidad Keystone, de otro modo no se podrá realizar ninguna operación en el entorno implementado.

  Cabe destacar que para mantener toda la información relativa a los usuarios y demás recursos de Keystone (dominios, proyectos, usuarios, roles, grupos o listado de servicios), se emplea una base de datos que habitualmente suele ser "*MariaDB*".

- **Nova:** es el servicio encargado de proporcionar una manera de aprovisionamiento de instancias de computación, servidores virtuales. Nova soporta la creación de máquinas virtuales o bare-metal, aunque para este último caso necesita de otro servicio de OpenStack, Ironic, que en este proyecto no ha sido considerado. Además, en las últimas versiones se le ha incorporado una funcionalidad limitada para sistemas de contenedores.

  El servicio Nova ha sido ideado para la creación, gestión, configuración y eliminación del conjunto de recursos de computación que puede desplegarse dentro de la Infraestructura como servicio que proporciona OpenStack. Por defecto, se emplea el hipervisor basado en el kernel de Linux, KVM.

Para que Nova funcione correctamente necesita de otros servicios de OpenStack: Keystone, Glance, Neutron, Placement.

Emplea el servicio de Keystone para realizar la autenticación de los usuarios, Placement para seguimiento y selección de recursos del inventario, Glance para las imágenes de sistema operativo empleado en los servidores, Neutron para realizar la conexión de las instancias virtuales tanto a las redes privadas como al exterior. También cabe la posibilidad de emplear Horizon para realizar un aprovisionamiento y gestión de forma gráfica a través de la interfaz web.

Nova se ejecuta como un conjunto de demonios, lanzados sobre el servidor Linux que lo contiene.

- Nova-api: es el servicio encargado de aceptar y responder las llamadas de los usuarios finales a la API de nova, es decir, el punto de conexión de cualquier servicio o usuario con el módulo de Nova. Hacer cumplir políticas e iniciar la mayoría de las actividades de orquestación como lanzar una instancia.

- Nova-api-metadata: acepta peticiones de metadatos desde las instancias. Se utiliza para que las instancias puedan recuperar datos específicos de instancias. Forma parte del funcionamiento de "*Nova-api*".

- Nova-compute: consiste en un demonio trabajador encargado de crear y eliminar instancias de máquinas virtuales a través de APIs de los hipervisores, XenAPI para XenServer, libvirt para KVM o QEMU y VMwareAPI para VMware.

  Se trata de un procesamiento complejo, el demonio acepta las acciones de una cola y ejecuta un conjunto de órdenes para lanzar una instancia, eliminarla o actualizar su estado en la base de datos.

- Nova-scheduler: es un demonio encargado de registrar una solicitud sobre una instancia y obtener el servidor en donde se ejecuta. Evalúa y filtra los nodos de computación disponibles (servidores con nova) para elegir la mejor opción donde desplegarlas las VMs.

- Nova-conductor: es un demonio lanzado con la finalidad de mediar en las comunicaciones entre los servicios de nova-compute y la base de datos. Principalmente por temas de seguridad y escalabilidad, toda comunicación hacia la base de datos pasa por nova-conductor.

- Nova-novncproxy: es un demonio encargado de proporcionar un proxy de acceso a las instancias virtuales a través de conexiones VNC.

- **Neutron:** se trata de un proyecto de red SDN centrado en proporcionar redes como servicio (NaaS,*network-as-a-Service*) entre interfaces virtuales de dispositivos gestionados por otros servicios de OpenStack, como nova. En general, es el encargado de todo lo relacionado con la conectividad de las instancias, realizando la administración de las redes virtuales, las subredes, las direcciones IP, los routers, las reglas de seguridad aplicadas a los firewalls, etc.

  Es el servicio que gestiona las direcciones IP internas y la conectividad de todas las instancias virtuales que se despliegan en la infraestructura. Sin embargo, también es responsable de la asignación de direcciones IP externas, IP flotantes de la red pública de la infraestructura con la finalidad de hacer accesibles las aplicaciones o servicios que se desplieguen dentro de la infraestructura virtual.

  Se pueden modificar gracias a él,todas características de las redes: direccionamiento DHCP, servidores DNS, etc. Además, de permitir aplicar grupos de seguridad para limitar las conexiones mediante reglas de cortafuegos.

  Al igual que para Nova, Neutron está formado por una serie de servicios ejecutándose en el server Linux principal en forma de demonios:

  - Neutron-server: se trata de la API del servicio de Neutron, es el mecanismo de comunicación con este módulo por parte de un usuario o servicio para crear, enumerar, administrar o eliminar algún dato de una red o la red por completo.

  - Neutron-DHCP-agent: este agente es el encargado de hacer las labores de DHCP (*Dynamic Host Configuration Protocol*), para asignar direcciones IP a las diferentes máquinas virtuales que se desplieguen dentro del entorno, y RADVD (*Router Advertisement Daemon*), se encarga de anunciar las direcciones y rutas del protocolo IPv6. Hay que remarcar que se necesita el agente de capa 2 funcionando en el mismo nodo.

    Para realizar la tarea de DHCP emplea el software "*dnsmasq*", incorporando en algunos casos servicio de DNS en las instancias.

- Neutron-L3-agent: este agente se encarga de proveer servicios avanzados de capa 3, como routers o IPs flotantes. Cabe destacar la necesidad de una ejecución en paralelo del agente de capa 2.

- Neutron-metadata-agent: este agente permite a las instancias acceder a metadatos de cloud-init y datos de usuario a través de la red. Es necesaria una ejecución del agente de capa de enlace, L2, en el mismo nodo.

- Neutron-openvswitch-agent: este es un demonio de capa 2 para realizar la conectividad en capa de enlace a los diferentes recursos de OpenStack. Normalmente este servicio se ejecuta tanto en los nodos de red como en los nodos de computación.

  De este tipo de agentes, L2, hay varios, aunque los utilizados con mayor frecuencia "*Open vSwitch Agent*" y "*Linux Bridge Agent*", ambos poseen el mismo funcionamiento, únicamente varía el proveedor (programa).

- **Cinder:** es un sistema de almacenamiento en bloque desarrollado para OpenStack. Se encarga de virtualizar la gestión de los dispositivos de almacenamiento en bloque y proporciona a los usuarios finales una API que pueden emplear para solicitar y consumir esos recursos sin necesidad de saber la localización del almacenamiento o el tipo de dispositivo.

  Proporciona volúmenes a las instancias virtuales de Nova, a host bare-metal de Ironic, a los contenedores, etc. Algunos de los objetivos de Cinder son:

  - Arquitectura basada en componentes: permite añadir rápidamente nuevos comportamientos.

  - Muy alta disponibilidad: Escalable a cargas de trabajo muy altas.

  - Tolerante a fallos: mediante el aislamiento de procesos evita los fallos en cascada.

  - Recuperable: los fallos deben ser fáciles de diagnosticar, depurar y arreglar.

  - Estándares abiertos: debe de ser una aplicación de referencia para una API apoyada e impulsada por la comunidad.

Cinder funciona con tres conceptos clave, volumen, instantánea y backup.

- Volumen: es un dispositivo de almacenamiento desmontable semejante a un USB, se puede asociar a una instancia a la vez.
- Instantánea: consiste en una copia de un punto temporal de los datos que contiene el volumen, tiene propiedades de solo lectura.
- Backup: es una copia completa de un volumen almacenado en un servicio externo. Este servicio tiene que ser configurado para utilizarlo y únicamente es compatible con el servicio Swift.

El módulo de Cinder está compuesto por varios demonios lanzados como servicios, cinder-api, cinder-volume, cinder-scheduler.

- Cinder-api: es la API de cinder, actúa como pasarela de conexión con el resto de usuarios y servicios. Es necesario para crear, enumerar, administrar y eliminar volúmenes, instantáneas o backup.
- Cinder-volume: está configurado en los nodos de almacenamiento y es el responsable de la creación y eliminación de los volúmenes.
- Cinder-scheduler: este demonio se encarga de evaluar y filtrar los diferentes nodos de almacenamiento utilizados en el entorno de Openstack para decidir cuál es el más idóneo para utilizar en la creación de un nuevo volumen.

- **Glance:** es un servicio que incluye el descubrimiento, registro y recuperación de imágenes de máquinas virtuales. Tiene una API RESTful con la que consultar metadatos de las imágenes de máquinas virtuales o recuperar las imágenes originales.

  Las imágenes disponibles en el catálogo de Glance pueden ser almacenadas en múltiples lugares, desde sistemas de ficheros simples o sistemas de almacenamiento de objetos proporcionados por el servicio Swift.

  Una imagen de Glance es un fichero que contiene una imagen del sistema operativo preinstalada y con una configuración previa realizada para que sea compatible con la nube, funcionando como plantilla de disco duro con sistema operativo instalado para las instancias que se desplieguen.

Las imágenes de Glance almacenan todo el contenido del sistema operativo y pueden presentarse en varios formatos:

- RAW: formato simple y sin comprimir, es soportado de manera nativa en hipervisores KVM y XEN.
- QCOW2 (*QEMU copy-on-write*): se trata de un formato de archivo de imagen utilizado por hipervisores QEMU y recibe su nombre por la manera en la que optimiza el almacenamiento, únicamente lo asigna cuando es estrictamente necesario. Por esto las imágenes que tiene este formato tienen generalmente un peso menor.
- VMHDK: inicialmente fue diseñado para su empleo en hipervisores VMware, pero en la actualidad es compatible con otros.
- VHD: es un formato de imagen relacionado con productos de Microsoft, como Hyper-V o Azure.
- VDI: es el tipo de formato más empleado en Virtualbox.
- ISO: siendo uno de los formatos más comunes del mercado, es un archivo de disco óptico que contiene todos los datos del sistema operativo y sistema de ficheros.

Glance está compuesto por varios demonios lanzados como servicios para realizar las operaciones que permite, aunque por defecto solo se ejecuta "glance-api", encargado del almacenamiento y la recuperación de de imágenes de disco.

- **Horizon:** consiste en una implementación canónica de un panel de control de Openstack, proporciona una interfaz gráfica para el usuario basada en web. Es extensible en función de los servicios que se instalen en el entorno Openstack: Swift, Heat, Cinder, etc.

  Las características más relevantes de Horizon son:

  - soporte del núcleo, tiene un soporte inmediato para todos los proyectos.
  - extensible, cualquier persona puede añadir un nuevo componente.
  - manejable, el código base del núcleo debe ser simple y fácil de navegar.
  - consistente, los paradigmas visuales y de interacción se mantienen en todo momento.

- estable, una API confiable que tenga muy en cuenta la retrocompatibilidad.
- utilizable, proporcionando una interfaz llamativa que los usuarios quieran utilizar.

Horizon es un servicio web sobre Apache, y desarrollado en Python y su framework Django.

La interfaz web permite a los usuarios acceder y administrar los servicios con la finalidad de proveer y automatizar el reparto de recursos disponibles. Se pueden incorporar otras funcionalidades como monitorización, facturación, etc.

En función del tipo de usuario que se tenga se podrán realizar unas acciones u otras, no será lo mismo se usuario administrador que obtiene una vista general del sistema, pueda crear usuarios, asignarlos a proyecto, grupos, crear proyecto, configurar permisos de los usuarios.

O ser un usuario normal que tiene permitido el aprovisionamiento de recursos de entre los definidos para la quota de su proyecto, como almacenamiento, instanciación de máquinas virtuales, creación de redes privadas, etc.

- **Placement:** se trata de un servicio que proporciona una API HTTP para realizar el seguimiento de los inventarios y uso de recursos de la nube, de tal forma que ayuda a otros servicios a gestionar y asignar los recursos de forma eficaz y optimizada.

  Permite controlar el uso de recursos por parte de cada proveedor. Por ejemplo, una instancia creada por un nodo de computación puede consumir recursos de RAM y CPU de un proveedor nova, el disco de un proveedor de almacenamiento y las direcciones de red de un proveedor de recursos de IP, neutron.

  En sus inicios, el servicio de Placement surgió con el proyecto Nova, cuyos requisitos propiciaron la mayor parte de las funcionalidades disponibles en Placement, pero se diseñó lo suficientemente genérico para que pudiera ser utilizado por cualquier otro servicio que necesitará realizar selección y consumo de recursos.

- **Swift:** es un almacén de objetos altamente disponible, distribuido y eventualmente consistente. Swift te permite crear, modificar y obtener objetos y metadata mediante el empleo de la API de swift, implementada como servicio web REST.

  Las organizaciones suelen emplear Swift para almacenar muchos datos de manera eficiente, segura y barata. Está construido para escalar y optimizado para proporcionar durabilidad, disponibilidad y concurrencia en todo el conjunto de datos. Swift tiene un funcionamiento perfecto para almacenar datos no estructurados.

  Se trata del servicio perfecto para almacenar datos estáticos de webs, realizar control de versiones, además permite listas de control de acceso.

  Swift almacena los objetos en distintas unidades de disco distribuidas en distintos servidores, realiza una replicaciones de datos buscando tener integridad en todo el cluster y ser tolerante a fallos.

  El sistema de almacenamiento de objetos organiza los datos en una jerarquía que afecta a la forma de interactuar con la API de Swift y se clasifica de la siguiente forma:

  - Cuenta: es el nivel superior. Un proveedor crea una cuenta y asigna una serie de recursos. La cuenta define un espacio de nombre para contenedores. En el entorno Openstack, cuenta se reconoce como proyecto.

  - Contenedor: define un espacio de nombres para objetos. Aunque un objeto tenga el mismo nombre si está en contenedores diferentes se consideran objetos distintos. Además de contener objetos, puede utilizar contenedores para controlar el acceso a los objetos mediante listas de control de acceso (ACLs), o incluso configurar y controlar otras funciones como el control de versiones de objetos.

    Una característica importante es que se pueden eliminar contenedores masivamente en una sola solicitud a la API.

  - Objeto: almacena contenido de datos, como documentos, imágenes, etc. Incluso puede almacenar metadatos como objeto.

    El empleo de la API permite almacenar una cantidad ilimitada de objetos, cargar y almacenar objetos de cualquier objeto permitiendo la creación de objetos grandes, emplear uso compartido

de recursos para administrar seguridad, comprimir archivos empleando metadatos, programar eliminación de objetos, eliminación masiva de objetos (hasta 10.000) con una solicitud, extraer automáticamente los archivos de almacenamiento, generar una URL para acceso temporal, cargar objetos mediante un middleware para formularios POST o incluso crear enlaces simbólicos a otros objetos.

Se puede ver cómo la API de Swift proporciona una amplia funcionalidad y para ello necesita que estén lanzados una serie de servicios como demonios hasta catorce en función de las necesidades del entorno Openstack utilizado. Aunque por defecto únicamente emplea cuatro que son esenciales para su arquitectura.

- Swift-proxy-server: es el servicio responsable de unir el resto de servicios de la arquitectura de Swift y de actuar como API de conexión al módulo de almacenamiento de objetos. Los usuarios que deseen interactuar con Swift tendrá que comunicarse con este demonio para crear contenedores, cargar objetos, establecer lista de control de acceso o habilitar capacidades de control de versiones.

- Swift-object-server: es un servidor de almacenamiento de objetos muy simple que permite almacenar, recuperar y eliminar objetos almacenados en dispositivos locales. Los objetos se almacenan como archivos binarios en el sistema de archivos en forma de metadatos almacenados en los atributos extendidos del archivo (*xattrs*).

  Cada objeto se almacena en una ruta derivada del hash obtenido del nombre del objeto y la marca de tiempo de la operación. Cuando se elimina un objeto también se trata como una versión del objeto permitiéndose una replicación correcta.

- Swift-container-server: es el servicio demonio que administra las lista de objetos dentro de un contenedor. Los listados se almacenan como archivos de base de datos sqlite y se replican en el cluster de manera similar a los objetos. Adicionalmente, se encarga de realizar un seguimiento de estadísticas del número de objetos y de su uso.

- Swift-account-server: este servicio demonio tiene un funcionamiento muy similar al "*Swift-container-server*" con la salvedad de que es responsable de listas de contenedores en lugar de objetos.

5.2.2. Servicios adicionales.

En adición a los servicios desplegados para el caso anterior, se considera de especial relevancia para el escenario que se desea desarrollar en el presente proyecto añadir una serie de servicios o módulos adicionales que nos confieren algunas opciones de desarrollo o implementación más que interesantes para una herramienta de cloud computing como OpenStack.

- **Ceilometer:** es un servicio para recopilación de datos que proporciona la capacidad de normalizar y transformar los datos de todos los componentes. Consiste en un proyecto de telemetría, los datos registrados pueden ser empleados para realizar la facturación a los clientes, hacer un seguimiento de recursos y configurar alarmas para los componentes centrales de Openstack.

  Al fin y al cabo su finalidad principal es recoger, normalizar y transformar eficientemente los datos producidos por los distintos servicios de Openstack.

  En el inicio de este proyecto fue diseñado para proporcionar una infraestructura capaz recopilar cualquier información necesaria sobre los proyecto de Openstack. A medida que fue cogiendo fuerza se fueron incorporando otros proyectos medidores terminando en convertirse en una forma estándar de medir independiente del propósito, posteriormente se clasificarían los datos.

  El proceso de medición sigue tres pasos: medida, clasificación y facturación. Desde el principio ceilometer se ha centrado en el primer paso la medición puesto que los otros dos abría un amplio abanico de posibilidades.

  En la figura 5.4 se muestra un resumen general de la arquitectura lógica de Ceilometer. Hay que decir que cada componente de Ceilometer está preparado para escalar horizontalmente y se pueden añadir nodos en función de la carga. En arquitectura lógica podemos apreciar que ceilometer ofrece dos servicios:

  - Polling Agent (*agente de sondeo*): se trata de un demonio creado para sondear los diferentes servicios implementados en el entorno Openstack y recopilar información.
  - Notifications Agent (*agente de notificación*): es un demonio diseñado para escuchar notificaciones en la cola de mensajes, convertirlas en eventos y muestras, y aplicar acciones de canalización.

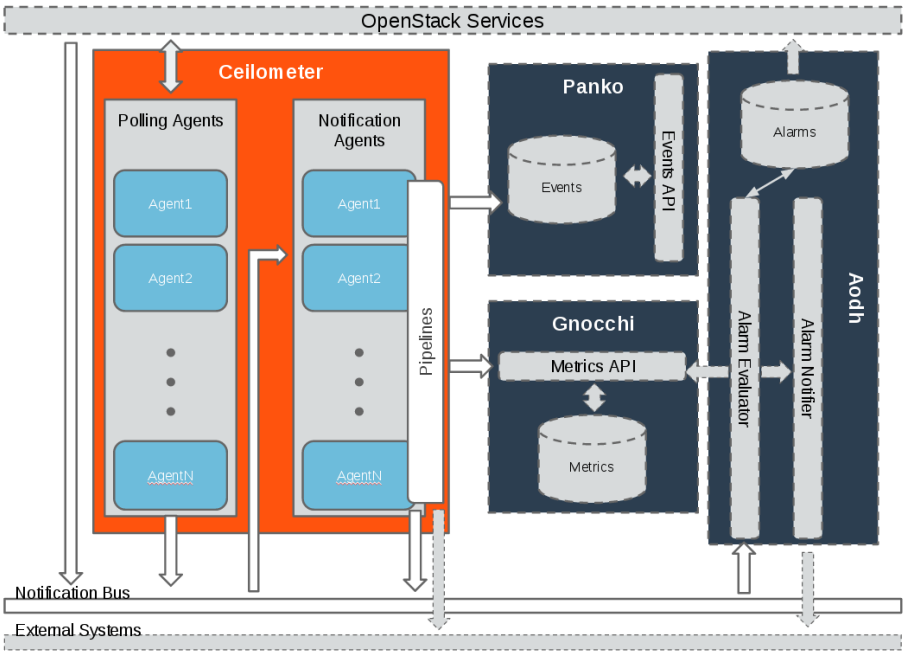

Figura 5.4: Arquitectura lógica de Ceilometer.

Los distintos datos que son normalizados y recolectados por Ceilometer se pueden enviar a diferentes objetivos. Gnocchi desarrollado para capturar datos de medición en un formato de serie de tiempo con la finalidad de optimizar el almacenamiento y las consultas. Aodh, es un sistema de alarma que alerta de casos donde se rompen algunas reglas definidas por los usuarios.

En último lugar, tenemos Panko, un servicio de almacenamiento de eventos diseñado para capturar datos orientados a documentos, como registros y eventos del sistema. Los dos últimos servicios se explicarán un poco más a continuación.

- **Aodh:** proporciona un servicio que permite la capacidad de desencadenar acciones basadas en reglas definidas contra algunos datos métricos o de eventos recogidos por los servicios de Ceilometer o Gnocchi. Es decir, Aodh proporciona un servicio de alarma ante distintos eventos.

- **Panko:** es un componente del proyecto global de telemetría de Openstack, es un servicio de almacenamiento de eventos que ofrece la posibilidad de almacenar y consultar los datos de eventos generados por Ceilometer.

Panko fue diseñado para proporcionar un servicio de indexación de metadatos y almacenamiento de eventos que permite a los usuarios capturar información de estado de los recursos de OpenStack en un momento dado. Su objetivo principal es poder tener un medio escalable de almacenamiento de datos telemétricos para poder llevar a cabo auditorías o depuración del sistema.

- **Heat:** es un servicio desarrollado para orquestar aplicaciones en la nube usando plantillas declarativa, en forma de texto que posteriormente se trata como código,a través de una API REST.

  Heat también proporciona un servicio de autoescalado que se integra con los datos de telemetría obtenidos por los servicios de Ceilometer y compañía.

  El objetivo principal de Heat es ayudar a los usuarios a modelar, configurar y automatizar la creación y administración de recursos de Openstack. Su funcionamiento está basado en la definición de plantillas en formato JSON que se encargan de describir todos los detalles sobre los recursos o elementos que componen el escenario que se quiere desplegar de manera automatizada. Además, como se ha comentado anteriormente, permite incluir mecanismo de control y monitorización de diversas características con los que escalar automáticamente el escenario en función de los requisitos.

  El funcionamiento de Heat está compuesto por tres componentes ejecutados como demonios:

  - Heat-api: este componente se encarga de proveer una API REST nativa de Openstack que procesa las solicitudes y las reenvía a Heat-engine. Los usuarios deben de interactuar con esta API para poder trabajar con las pilas del servicio.

  - Heat-api-cfn: conforma una API de consultas AWS compatible con AWS CloudFormation, que procesa las solicitudes y las redirecciona a Heat-engine.

  - Heat-engine: es un componente de Heat cuya principal responsabilidad es orquestar el lanzamiento de plantillas, administrar los recursos de las plantillas y proporcionar eventos a los clientes de la API.

- **Zun:** es un servicio de contenedores de Openstack, su objetivo es proporcionar un servicio de API para poder ejecutar contenedores y lanzar aplicaciones en ellos sin necesidad de utilizar servidores virtuales o clústeres.

  A diferencia de Magnum, Zun es un elemento para tratar contenedores como recurso gestionado por OpenStack. Esto hace que sea extremadamente importante que este módulo se integre bien con otros servicios, como la red de Neutron o los volúmenes de Cinder.

  Zun para funcionar necesita de otros servicios de Openstack como Keystone, Neutron y Kuryr-libnetwork (un driver de red de Docker empleado por Neutron para proporcionar servicios de red a los contenedores). Además, puede integrarse con otros servicios como Cinder, Heat y Glance.

  El servicio Zun está compuesto por una serie de demonios que hace que rinda correctamente:

  - Zun-api: consta de una API Rest nativa de Openstack que procesa las solicitudes de los usuarios y las envía a zun-compute. Es necesario comunicarse con esta API para poder interactuar con los contenedores dentro de Openstack, crear, gestionar o eliminar.
  - Zun-compute: es un demonio trabajador encargado de crear y terminar los contenedores o cápsulas (conjunto de contenedores) a través de la API de Zun. Gestiona los contenedores, las cápsulas y los recursos de computación que hay disponibles en el host local.
  - Zun-wsproxy: este servicio proporciona un proxy para acceder y gestionar los contenedores a través de una conexión web.
  - Zun-cni-daemon: proporciona un servicio demonio CNI que permite la implementación del plugin de CNI de Zun.

  Opcionalmente, se puede utilizar python-zunclient, como cliente de línea de comandos para crear, administrar y eliminar contenedores, y zun-ui, un plugin que se añade a Horizon para gestión gráfica a través de la interfaz web.

- **Magnum:** magnum pone a disposición de OpenStack motores de orquestación de contenedores como Docker Swarm, Kubernetes y Apache Mesos, siendo recursos de primera clase. Magnum emplea Heat para poder orquestar una imagen del sistema operativo que contiene Docker

y Kubernetes y ejecuta esa imagen en máquinas virtuales y servidores bare-metal con la finalidad de configurar un clúster de contenedores.

Magnum tiene diferentes tipos de objetos:

- Clúster: una colección de objetos de nodo donde se programa el trabajo.
- Cluster template: un objeto almacena información de plantilla sobre el clúster que se usa para crear nuevos clústeres de manera consistente.

Para que funcione el módulo de Magnum se ejecutarán dos servicios:

- Magnum-api: se trata de un servidor API REST, este servidor puede ejecutar en uno o varios procesos. Cuando se envía una solicitud a la API del cliente se reenvía al proceso magnum-conductor, que actualmente se encuentra limitado a un proceso.

Las características principales de Magnum son:

- Abstracciones para clústeres.
- Integración con Kubernetes, Docker Swarm y Mesos como tecnologías de contenedores.
- Integración con Keystone para seguridad multi-proyecto.
- Integración de Neutron para seguridad de red multi-proyecto de Kubernetes.
- Integración con Cinder para proveer servicios de volúmenes para los contenedores.

- **Kuryr:** es un componente de Openstack que funciona como puente entre los modelos de contenedores y la abstracción de las redes de Openstack.

Es un plugin de Docker para poder proporcionar servicio de red a contenedores mediante Neutron.

De este proyecto forma parte Kuryr-libnetwork, que es un driver de Docker que utiliza Neutron para proveer recursos de red a los contendores. Presenta como características:

- Control remoto del driver de red de Docker.
- Driver IPAM del libnetwork de Docker.
- Soporte para Linux Bridge, Open vSwitch, Midonet y enlaces de puertos IOvisor.
- Apoya el uso de las redes Neutron existentes.
- Apoya el uso de puertos de Neutron existentes y permite la opción de exponer puertos de Docker.

- **Barbican:** es un servicio de administración de llaves, realiza un almacenamiento seguro, provisión y gestión de datos secretos, como contraseñas, cifrados, claves simétricas y asimétricas, certificados X.509 y datos binarios en bruto.

- **Trove:** es un módulo que ofrece base de datos como servicio mediante aprovisionamiento de bases de datos relacionales y no relacionales. Su objetivo principal es permitir a los usuarios una utilización rápida y fácil de las características de las bases de datos por parte de los usuarios finales. Los usuarios de la nube y los administradores de la base de datos pueden aprovisionar y gestionar múltiples instancias de la base de datos según sea necesario.

- **Freezer:** es una plataforma de copia de seguridad distribuidas, restauración y recuperación ante desastres como servicio. Está diseñada para ser compatible con múltiples sistemas operativos, enfocado a proporcionar eficiencia y flexibilidad para copias de seguridad basadas en bloques, copias de seguridad incrementales basadas en archivos, acciones de punto temporal, sincronización de trabajos (entiéndase como sincronización de backups en múltiples nodos). Busca ser un servicio útil para cualquier entorno de cloud.

- **Masakari:** es un módulo que proporciona un servicio de alta disponibilidad para las instancias de la nube Openstack mediante recuperación automática de instancias con errores. En la actualidad, Masakari puede recuperar VMs basadas en KVM de eventos de fallos, como caída de procesos de la máquina virtual, caída del abastecimiento o el fallo del nodo de nova-compute.

  Masakari también proporciona una API que permite gestionar y controlar los mecanismos de rescate automatizados.

- **Octavia:** se trata de un proyecto de código abierto para ofrecer balanceadores de carga como servicio en entorno Openstack. Surgió como bifurcación del proyecto Neutron LBaaS hasta que se convirtió en una implementación de referencia dentro de los componentes de OpenStack.

  Octavia consigue realizar habilidades de balanceo mediante la gestión de un conjunto de máquinas virtuales, contenedores y servidores bare-metal - conocidos como ánforas - en función de la demanda. Esta característica que le permite escalar horizontalmente le confiere una gran diferencia con respecto al resto de alternativas, confirmando como una solución más que adecuada para entornos ejecutados en la nube.

- **Designate:** es un componente de Openstack multi-proyecto que proporciona DNS como servicio. Proporciona una API REST con autenticación de Keystone integrada. Posee la posibilidad de configurar el registro automático en función de las acciones de Nova y Neutron. Designate es compatible con una amplia variedad de servidores DNS, entre los que destacan Bind9 y PowerDNS 4.

- **Monasca:** consiste en un proyecto de código abierto multi-proyecto altamente escalable, de alto rendimiento y tolerante a fallos, integrado en OpenStack. Emplea una API REST para procesamiento y consulta de métricas de alta velocidad y tiene un sistema de alarma y notificación en tiempo real.

- **Manila:** es un servicio que proporciona un acceso coordinado a sistemas de ficheros compartidos o distribuidos. Algunos objetivos que tiene presentes este proyecto son:

  - Arquitectura basada en componentes para agregar rápidamente nuevas funcionalidades.
  - Alta disponibilidad: pretende funcionar correctamente ante grandes cargas de trabajo.
  - Tolerante a fallos: al tratar las funcionalidades mediante componentes separados evita fallos en cascada.
  - Recuperable: los errores originados deben de ser fáciles de diagnosticar, depurar y rectificar.
  - Estándares abiertos: debe de ser una implementación de referencia para una API impulsada por la comunidad, tanto desarrolladores como clientes.

Capítulo 6

Diseño e Implementación. Instalación y Configuración.

En este capítulo se expondrá un recopilatorio de todos los pasos necesarios para realizar la instalación, configuración y posibles modificaciones necesarias post-configuración de todas las tecnologías o herramientas empleadas en el presente proyecto.

Primero de todo, será necesario realizar una pre-configuración del sistema operativo elegido, en nuestro caso Ubuntu 18.04.4 LTS. En algunos casos será necesario realizar ciertas modificaciones de los ficheros de instalación de las herramientas porque se detectan errores y es necesario solventarlos para que la instalación y configuración de los diferentes módulos resulte satisfactoria.

Una vez se tiene todo preparado será necesario instalar OpenStack en el equipos con mayores prestaciones, son necesarias para poder lanzar máquinas virtuales. En el otro equipos se instalará Open Source MANO, por ello puede haber alguna diferencia en la configuración del sistema operativo.
Además, para que funcionen algunos servicios como se desea será necesario realizar ciertas modificaciones de las configuraciones de los equipos, estas concluirán el capítulo.

6.1. Diseño del escenario.

El escenario que se va a plantear en el presente proyecto está conformado por una plataforma de orquestación de NFV, Open Source Mano (OSM), se encargará de las tareas de administración y orquestación de algunos escena-

rios de prueba lanzado sobre la infraestructura virtual de cloud computing creada por OpenStack - encargada de proveer los servicios de máquinas virtuales a los Network Service lanzados por OSM.

Además de OSM como medio de orquestación, OpenStack se instalará con el módulo Heat para emplear el servicio de orquestación propio de OpenStack y comprobar resultados.

Para ir comprobando el funcionamiento de cada uno de los elementos del escenario se irán lanzando instancias o servicios, y se concluirá con el despliegue del recursos deseado, un servidor web con uso de base de datos.

• **Servicio Wordpress:** consiste en un servicio web ejecutado sobre un servidor de Apache, configurado de tal manera que emplee para funcionar la base de datos del otro servicio, MariaDB.

• **Servicio MariaDB:** es un sistema de gestión de bases de datos derivado de MySQL utilizado como base de datos del servidor web.

Con el uso de Heat en Openstack se permite la automatización del despliegue y administración de los servicios, ya sea desplegados como máquinas virtuales o como contenedores. Por otro lado, tenemos OSM, cuya funcionalidad nos permitirá automatizar el despliegue de los servicios definiendo las VNFs y NS a utilizar. Además, nos permite escalar fácilmente el escenario.

A continuación, estudiaremos los dos posibles escenarios que nos vamos a encontrar en el desarrollo del proyecto.

En la figura 6.1 podemos ver los diferentes elementos que forman parte del proyecto y su relación. OSM será el encargado de gestionar la definición de VNFs mediante la que lanzar nuevos NSs (Network Services). Como vimos en capítulos anteriores un orquestador de mano necesita de un NFVI, aquí entra juego OpenStack encargado de gestionar y administrar los recursos de la infraestructura para NFV.

Por otro lado, tenemos la figura 6.2 que nos muestra la conexión entre los diferentes entornos y permite lanzar tanto máquinas virtuales como contenedores dentro de la infraestructura virtual generada con la plataforma OpenStack.

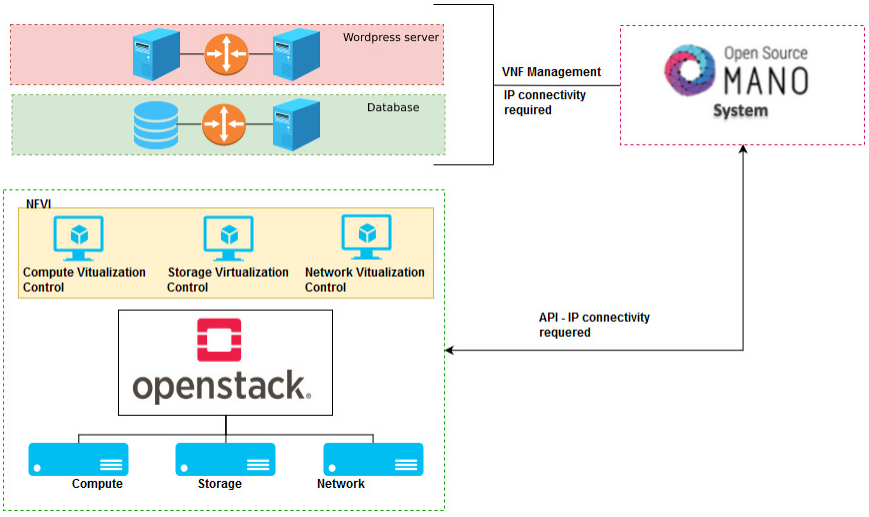

Figura 6.1: Arquitectura OSM-Openstack para NFV.

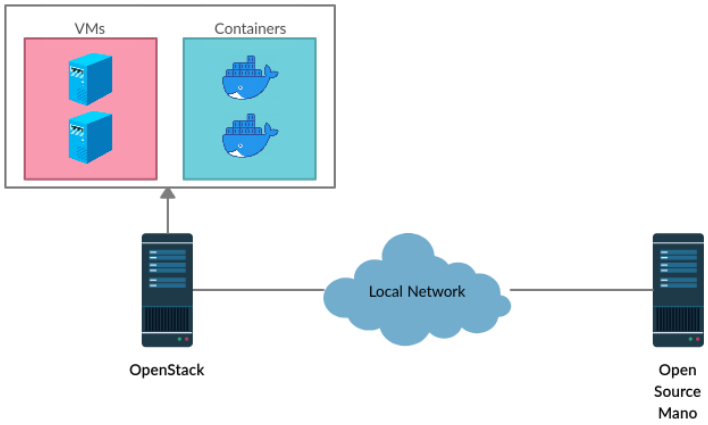

Figura 6.2: Arquitectura OSM-OpenStack con Heat.

6.2. Instalacion y configuracion de Openstack.

Este apartado recogerá todo el procedimiento de preparación del sistema operativo, instalación y configuración del entorno OpenStack y posteriores configuraciones realizadas para poder funcionar correctamente. Se incluirá los problemas que se han ido dando durante todo el proceso, y evidenciando las razones que han llevado a realizar la instalación de la plataforma como se tiene en la actualidad.

El ordenador utilizado para este entorno deberá tener más recursos que el de OSM, para poder lanzar instancias virtuales y poder realizar pruebas de un entorno controlado de cloud computing. Por ello se utilizarán un PC con las siguientes características:

- Sistema operativo Ubuntu 18.04.4 LTS.
- Procesador Intel Core i7-9750H+HM370.
- Memoria RAM de 16 Gb.
- Disco duro SSD de 512 Gb.

La versión de OpenStack instalada es Ussuri, la última estable hasta el momento liberada por la organización. La distribución utilizada de esta versión es DevStack, porque está pensada para que funcione en un único nodo, como es nuestro caso.

Se han realizado varios intentos de despliegue, en cuanto a componentes se refiere y se han ido amoldando los componentes que se instalan en la versión final, en función de la funcionalidad y los recursos disponibles.

Antes de comenzar con los intentos de instalación realizados, se van a exponer los pasos realizados para configurar el sistema operativo con la finalidad de dejarlo preparado para las instalaciones de OpenStack. Es necesario destacar que una vez configurado el equipo se creó un punto de restauración con el programa "*Timeshift*" y de este modo se ahorra tiempo entre intentos de instalación y siempre en las mismas condiciones.

La configuración inicial realizada consiste en configurar las interfaces de red del PC e instalar y desinstalar algunas herramientas que evitan errores en la instalación de OpenStack y facilitará su utilización posteriormente.

En primer lugar, tendremos que configurar una dirección IP estática en una de las interfaces, para evitar posibles errores de microcortes de conexión durante las instalaciones se ha elegido configurar la interfaz WiFi y dejar la interfaz Ethernet como conexión por defecto a internet. En el caso de OpenStack es estrictamente necesario fijar una dirección IP porque además de ser un servicio que debe de ser accedido con una URL, todos los componentes tienen que definir el endpoint del servicio de identidad en sus ficheros de configuración, entonces no puede variar la dirección del servicio.

Para realizar la configuración de la dirección IP fija tenemos dos opciones, mediante la configuración gráfica en el menú de ajustes del sistema operativo, donde elegiremos la opción "*Inalámbrica*" y seleccionaremos la red a la que queremos conectarnos y una vez estemos conectados, seleccionaremos el botón del engranaje. En la opción de IPv4, seleccionaremos "*Manual*" y rellenaremos los datos de IP, máscara de red, pasarela y servidores DNS.

Existe otro método para definir la configuración de las interfaces de red, pero mediante línea de comandos y será explicado en el apartado de configuración de OSM.

Una vez hemos asignado la dirección IP que deseamos, podemos proceder con la preparación del sistema instalando el editor de textos por línea de comandos, "*Vim*", el gestor de interfaces de red, "*net-tools*", y la herramienta "*traceroute*". Para ello actualizamos los repositorios de APT y lanzamos el comando de instalación:

```
$ sudo apt update
$ sudo apt install -y vim net-tools traceroute
```

Adicionalmente, será necesario eliminar dos complementos de Python porque durante la instalación de OpenStack se detecta una versión existente y no puede ser modificada, se trata de "*pexpect*" y "*simplejson*".

```
$ sudo apt purge python3-pexpect python3-simplejson
```

En este punto podemos pasar a comentar cada intento de instalación de OpenStack, pero no sin antes indicar que algunos errores se han dado en todos los intentos y es por ello que será tratados en la instalación final junto con los pasos comunes de preconfiguración para lanzar el instalador de DevStack (véase apartado 6.2.3).

Para el caso de los intentos fallidos se tratarán las modificaciones de ficheros necesarias y configuraciones realizadas para que puedan ser replicadas en cualquier momento, informando de aquellos errores y soluciones que se han realizado para poder continuar con la instalación.

6.2.1. Intento de instalación de OpenStack completo.

La primera intención fue realizar una instalación de OpenStack lo más completa posible, con una gran número de componentes que se vería truncada por los recursos disponibles en el PC utilizado.

Este intento de instalación estuvo conformado con los componentes de OpenStack por defecto, Nova, Neutron, Keystone, Horizon, Cinder, Glance y Swift, y como componentes adicionales, Heat, Ceilometer, Panko, Aodh, Monasca - con la opción de implementada junto con Ceilometer, denominada Ceilosca - Manila, Masakari, Freezer, Trove, Barbican, Octavia, Designate, Zun, Magnum y Kuryr.

Las expectativas eran demasiado altas al pensar que un PC con buenas prestaciones pero comercial podría correr todos los servicios listados más arriba. Antes de darnos cuenta que no se tenían recursos suficientes para todos los servicios, se intentó realizar la instalación de todos los componentes. En algunas ocasiones, al saturarse el PC se quedaba colgado y "*cerraba*" la sesión de usuario sin poder volver a iniciarla quedando inservible porque tampoco se sabía si la instalación concluía o no.

El primer fallo que nos encontramos fue relacionado con el módulo Monasca, el error indicaba que durante la ejecución de una función de instalación de "*monasca-transform*", un directorio no se encontraba vacío y una operación del comando "*mv*" no podía llevarse a cabo. La solución a este error consiste en modificar el comando "*mv*" por "*rsync*" en la función que falla. Deberemos de modificar el fichero con ruta "*/opt/stack/monasca-transform/devstack/plugin.sh*", buscaremos la función "*install_spark_jars*" y modificaremos el comando "*mv*" por "*rsync -a*", dejándolo como:

```
rsync -a /opt/spark/current/jars/ /opt/spark/current/jars_original
```

Otro fallo que surgió, también referido al módulo Monasca, pero esta vez para el caso de eliminar la instalación para poder lanzar el instalador desde cero y quitar la instalación anterior terminada con errores, fue que no se encontraba definida una función para eliminar una función de logs de monasca-api. Para este caso tuvimos que ver el script de instalación de monasca-api (también contiene las funciones de desinstalación) y buscar el apartado donde debería de estar.

Al no encontrarse definida se realizó una búsqueda en el repositorio de

OpenStack para saber si algún otro fichero tenía definida la función para poder copiarla en donde necesitábamos. Se encontró en el fichero "*plugin.sh*" del repositorio "*monasca-log-api*" (*monasca-log-api/devstack/plugin.sh*). Se tuvo que copiar la función "*clean_monasca_log_api*", pero esta necesitaba de otras dos funciones que tampoco estaban definidas en el fichero que nos estaba dando errores, hubo que copiar las tres funciones en el fichero de nuestro instalador.

El fichero que nos daba a nosotros error está en la ruta "*/opt/stack/monasca-api/devstack/lib/monasca-log.sh*", a este fichero se le tiene que añadir las funciones "*clean_monasca_log_api*", "*clean_monasca_log_api_wsgi*", "*stop_monasca_log_api*".

```
668 function clean_monasca_log_api {
669     if is_service_enabled monasca-log-api; then
670         echo_summary "Cleaning monasca-log-api"
671 
672         sudo rm -f $MONASCA_LOG_API_CONF || true
673         sudo rm -f $MONASCA_LOG_API_PASTE  || true
674         sudo rm -f $MONASCA_LOG_API_LOGGING_CONF || true
675         sudo rm -rf $MONASCA_LOG_API_CACHE_DIR || true
676         sudo rm -rf $MONASCA_LOG_API_CONF_DIR || true
677 
678         sudo rm -rf $MONASCA_LOG_API_DIR || true
679 
680         if [ "$MONASCA_LOG_API_USE_MOD_WSGI" == "True" ]; then
681             clean_monasca_log_api_wsgi
682         fi
683     fi
684 }
685 
686 function clean_monasca_log_api_wsgi {
687     sudo rm -f $MONASCA_LOG_API_WSGI_DIR/*
688     sudo rm -f $(apache_site_config_for monasca-log-api)
689 }
690 
691 function stop_monasca_log_api {
692     if is_service_enabled monasca-log-api; then
693         if [ "$MONASCA_LOG_API_DEPLOY" == "mod_wsgi" ]; then
694             disable_apache_site monasca-log-api
695             restart_apache_server
696         else
697             stop_process "monasca-log-api"
698             if [ "$MONASCA_LOG_API_DEPLOY" == "uwsgi" ]; then
699                 remove_uwsgi_config  "$MONASCA_LOG_API_UWSGI_CONF" "$MONASCA_LOG_API_WSGI"
700             fi
701         fi
702     fi
703 }
704 
```

Figura 6.3: Funciones incluidas en el fichero monasca-log.sh.

En algunas ocasiones ha surgido otro problema referente a Monasca, en concreto un error de Kafka mientras se instala "*monasca-transform*". Simplemente volviendo a lanzar el instalador de Devstack se solucionaba.

Una vez se tuvieron todos los errores bajo control para que la instalación pudiera continuar surgió otro problema nuevo, el servicio Kuryr necesita de Octavia como balanceador para funcionar pero no era capaz de configurarlo correctamente y la instalación volvía a fallar. Se decidió eliminar la opción de

instalar Kuryr desde el principio, puesto que por la documentación referente a este módulo podía desplegarse posteriormente sobre una instalación ya activa.

Tras este cambio en el fichero de configuración de la instalación se realizó la instalación completa de forma satisfactoria, pero el ordenador está utilizando de manera constante casi el 94 % de memoria RAM, esto impedía el uso de cualquier instancia virtual dentro de la infraestructura virtual de OpenStack. En la figura 6.4 se muestran los servicios instalados y los recursos que consume el PC con la instalación de OpenStack funcionando. El servicio de kuryr-libnetwork actúa como driver de red para el uso de contenedores.

Por último, hay que indicar que esta instalación, aunque tiene el servicio habilitado, no se realizó la configuración de la interfaz web para gestionar de forma gráfica OpenStack. Se intentó su instalación de manera manual pero no surtió efecto.

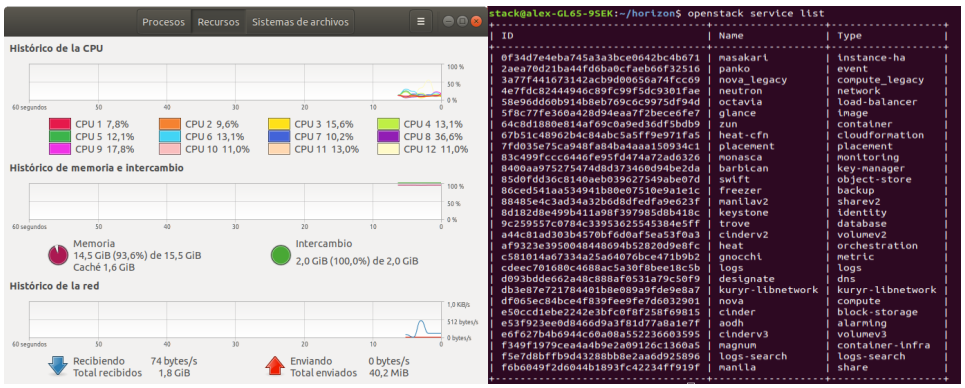

Figura 6.4: Servicio OpenStack instalados y recursos consumidos.

Otra prueba de instalación que se realizó fue incluyendo Kuryr, pero eliminando las opciones de Octavia y Designate, en este caso surgieron varios errores y se procedió con otras pruebas, puesto que no se encontró información ni solución.

6.2.2. Intento de instalación de OpenStack aligerado.

Ante el problema del consumo de recursos, se restauró el equipo al punto creado con la configuración previa del sistema operativo realizada y se lanzó un nuevo intento de instalación de OpenStack. Sin embargo, esta vez se quitaron algunos servicios para ver cómo se veían afectados los recursos consumidos. Se eliminaron las opciones de instalación de Kuryr - como se había mencionado anteriormente -, Octavia, Designate y Monasca con el plugin de Ceilosca.

En esta ocasión se pudo comprobar que Horizon no se había instalado en el caso anterior por algún problema originado por el módulo Monasca y visto que se habían tenido que modificar algunos ficheros de la instalación de este componente, no era algo para sorprenderse.

La instalación de OpenStack finalizó correctamente, pero esta vez sí se tenía disponible el componente de Horizon con el panel de control web para manejar la infraestructura de OpenStack de manera visual. En la figura 6.5 se puede observar la lista proporcionada por el cliente de línea de comandos de OpenStack de los servicios instalados.

En este despliegue se intentó realizar algunas pruebas de creación de instancias virtuales, las cuales funcionaban. Sin embargo, como se consumían prácticamente todos los recursos del PC con el funcionamiento de OpenStack, las máquinas virtuales no tenían recursos demasiados recursos para operar y cuando el sistema detectaba que el host anfitrión se estaba saturando mandaba una petición interna de apagar las VMs, lo cual impedía realizar cualquier tipo de configuración o prueba.

Para ir sobre seguro, se decidió ir parando servicios de forma manual con la finalidad de ver qué cantidad de recursos necesitaba cada uno. Los servicios de los componentes de DevStack se pueden controlar con "*systemctl*" y son nombrados como "*devstack@<servicio>*", siendo *<servicio>* los distintos nombres que reciben los componentes de cada módulo.

Para listar los nombres y el estado de los servicios que hay funcionando podemos utilizar el comando:

```
$ sudo systemctl status devstack@*
```

```
+----------------------------------+------------------+------------------+
| ID                               | Name             | Type             |
+----------------------------------+------------------+------------------+
| 0bf70da00d4e430fb2515dbc922c2a0e | freezer          | backup           |
| 1007f3b790ef4e59a770bfa2bdc36334 | neutron          | network          |
| 207171dd02be45838e7d3803fe849294 | aodh             | alarming         |
| 208e716e317c47719673aaaa8a217f04 | nova_legacy      | compute_legacy   |
| 33d3a989e862414a91c37066f21911a3 | manilav2         | sharev2          |
| 341f2a8e083a469ba8360bcd0d541b62 | swift            | object-store     |
| 454fca799f424d92819d78c17411ac3b | panko            | event            |
| 467add8144ed4d57ab2b96e30c2a8f10 | cinder           | block-storage    |
| 4e1b85741175429 39d09d14687f47527 | heat             | orchestration    |
| 620e26681cd045fcb00de239a19b4e13 | trove            | database         |
| 6568f350b8ee4c69bbc204b24f1bf035 | magnum           | container-infra  |
| 7a2035d792cf4e80be0ed52bd4387cb0 | heat-cfn         | cloudformation   |
| 7cbb5934f2da4faea26c29485e41caef | keystone         | identity         |
| 8d23dfcd6bd14afdb59709bb06e4a284 | cinderv3         | volumev3         |
| 8ee59fe34a6b4276b491fc6584793e90 | nova             | compute          |
| 92b38863c3f64024938b5b08319dd7cd | kuryr-libnetwork | kuryr-libnetwork |
| 9de643d3d6d846e892c991ab977318bc | masakari         | instance-ha      |
| aa1b1702956b48b189526e426b339966 | zun              | container        |
| adac010f9f294951a0ab51a82da59b28 | barbican         | key-manager      |
| bcae54928a784c33b8a47a497a81a453 | glance           | image            |
| d7ca28a0e6be45b2801eafb1936a7262 | placement        | placement        |
| f22c821e8a4644aba0f608b89e849d49 | manila           | share            |
| f2e41b6c057f457d8af4d69ddefc0195 | cinderv2         | volumev2         |
| f604690ecf9142f382f0137d4404468b | gnocchi          | metric           |
+----------------------------------+------------------+------------------+
```

Figura 6.5: Servicio instalados de OpenStack.

Con todos los servicios funcionando se consumían en torno a 14.9-15 Gb de memoria RAM, pero se tenía una instancia virtual de Ubuntu 16.04 funcionando con los mínimos recursos posibles para que pudiera ser levantada, sin contar los recursos de dicha VM se consumían 14.5 Gb, practicamente los mismo recursos que en el caso anterior.

Se procedió con la parada manual de los servicios obteniendo los siguientes resultados (servicio parado RAM consumida sin el servicio):

- Manila → 14.1 Gb
- Masakari → 13.7 Gb
- Trove → 13.1 Gb
- Zun → 13.0 Gb
- Barbican → 12.9 Gb
- Freezer → 11.7 Gb
- Magnum → 11.1 Gb

Ante la problemática de los recursos y viendo que muchos de ellos no iban a ser utilizados en el escenario de prueba se procedió a la restauración del sistema operativo al punto de restauración creado para hacer una instalación definitiva con los servicios mínimos necesarios para el escenario de pruebas deseado para el presente proyecto.

6.2.3. Instalación final.

Como última opción de instalación y viendo que los problemas de recursos persistían se tuvo que decidir sacrificar funcionalidades y quedarnos únicamente con los módulos de OpenStack necesarios para el escenario de pruebas que se iba a emplear.

En esta ocasión, la instalación de OpenStack sería lo más básica posible para las funcionalidades y pruebas que se van a desarrollar con contenedores orquestados y contenedores. La instalación contaría con los módulos por defecto: Nova, Neutron, Keystone, Horizon, Cinder, Glance y Swift, y como componentes adicionales, Heat, Ceilometer, Panko, Aodh, Barbican, Zun y Magnum.

En este apartado se recogerá todo el proceso de instalación necesario para desplegar el entorno de OpenStack en una única máquina mediante DevStack, incluyendo las soluciones a los errores que se han dado en todos los intentos de instalación. Se incluyen en este apartado para que el proceso de instalación se pueda seguir de manera más clara.

Antes de comenzar con la instalación, se vuelve a remarcar que se está utilizando el sistema operativo Ubuntu 18.04.4 como instalación mínima. En cuanto a la configuración de red realizada y dado que se tiene que poder utilizar direcciones IP flotantes se emplea una subred que no afecte al servicio DHCP de la red local.

La interfaz de red utilizada para el despliegue de OpenStack tiene la dirección IP *192.168.0.200* y se utiliza como subred, *192.168.0.224/27*.

Para instalar DevStack es necesario crear un usuario que tenga privilegios sudo para poder hacer cambios en el sistema.

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

Una vez se ha creado el usuario podemos proceder a loguearnos y comenzar con la descarga del repositorio de DevStack que será instalado. Mediante el comando siguiente nos logueamos y dirigimos al directorio del usuario.

```
$ sudo su - stack
```

Cuando estamos en el directorio de usuario nuevo y con la sesion iniciada, podemos proceder a descargar el repositorio con Git (si la herramienta no está instalada se puede instalar fácilmente):

```
$ sudo apt-get install git -y  #instalamos la herramienta Git
$ git clone https://opendev.org/openstack/devstack
$ cd devstack
```

Estando en el repositorio clonado, tendremos que cambiar de rama a la versión estable que se va a instalar, Ussuri:

```
$ git checkout stable/ussuri
```

Una vez hemos cambiado de rama podemos comenzar a configurar los ficheros necesarios para la instalación, crearemos el fichero "*local.conf*" que contiene todos los datos necesarios para la instalación de DevStack, configuración de red, contraseñas de servicios, componentes instalados en DevStack adicionales a los mínimos por defecto.

Hay un ejemplo de este fichero en la ruta "*devstack/samples/local.conf*", copiaremos ese fichero en el directorio "*devstack*" y lo modificaremos introduciendo la información necesaria para nuestro entorno.

Para configurar las contraseñas de los servicios de OpenStack se han incluido las siguientes líneas (a partir de la sección "*[[local — localrc]]*"):

```
ADMIN_PASSWORD=openstack
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
```

```
SERVICE_TOKEN=$ADMIN_PASSWORD
# Enable Keystone v3
IDENTITY_API_VERSION=3
```

Para configurar la interfaz que se desea utilizar como puente entre la infraestructura virtual interna y la externa, empleando la interfaz wifi y asignando un pool de red como IPs flotantes.

```
PUBLIC_INTERFACE=wlo1
HOST_IP=192.168.0.200
FLOATING_RANGE=192.168.0.224/27
```

En este punto, estaríamos frente a una instalación básica de OpenStack. Ahora pasaremos a indicar la configuración de los distintos componentes adicionales que se han incorporado a la plataforma.

Swift:

```
# Services Swift
enable_service s-proxy s-object s-container s-account
```

Heat y su dashboard en Horizon:

```
enable_plugin heat https://opendev.org/openstack/heat stable/ussuri
enable_plugin heat-dashboard https://opendev.org/openstack/heat-
    dashboard stable/ussuri
```

Mecanismos de monitorización, Ceilometer y adicionales:

```
CEILOMETER_BACKEND=gnocchi
enable_plugin panko https://opendev.org/openstack/panko stable/ussuri
enable_plugin ceilometer https://opendev.org/openstack/ceilometer
    stable/ussuri
enable_plugin aodh https://opendev.org/openstack/aodh stable/ussuri
```

Incorporación de Barbican, un servicio necesario para los contenedores con Zun:

```
### Barbican
enable_plugin barbican https://opendev.org/openstack/barbican stable/
    ussuri
```

Habilitamos Magnum como orquestador de clústeres de contenedores:

```
### Enable Magnum
enable_plugin magnum https://opendev.org/openstack/magnum
enable_plugin magnum-ui https://opendev.org/openstack/magnum-ui
```

Por último, pero no por ello menos importante y será bastante utilizado para las pruebas en el escenario diseñado, añadimos Zun con el driver de red de Kuryr. Además, se instala la API de línea de comandos para controlar y gestionar los contenedores desplegados.

```
enable_plugin zun https://opendev.org/openstack/zun stable/ussuri
enable_plugin zun-tempest-plugin https://opendev.org/openstack/zun-
    tempest-plugin
# Optional:  uncomment to enable the Zun UI plugin in Horizon
enable_plugin zun-ui https://opendev.org/openstack/zun-ui stable/
    ussuri

# This below plugin enables installation of container engine on
    Devstack.
# The default container engine is Docker
enable_plugin devstack-plugin-container https://opendev.org/openstack/
    devstack-plugin-container stable/ussuri
# This enables CRI plugin for containerd
ENABLE_CONTAINERD_CRI=True
enable_plugin kuryr-libnetwork https://opendev.org/openstack/kuryr-
    libnetwork stable/ussuri
# install python-zunclient from git
LIBS_FROM_GIT="python-zunclient"
```

Antes de continuar con la instalación, será necesario modificar un fichero de configuración de Apache dentro del repositorio de Devstack, de otro modo dará error porque no encuentra un fichero y tendremos que volver a lanzar la instalación.

La ruta del fichero que hay que modificar es "*devstack/lib*" y es fichero "*apache*". Antes de la modificación tendremos:

```
uwsgi=$(ls uwsgi*)
tar xvf $uwsgi
cd uwsgi*/apache2
```

Y deberemos de modificarlo para dejarlo así:

```
uwsgi=$(ls uWSGI*)
tar xvf $uwsgi
cd apache2
```

Una vez se ha creado el fichero "*local.conf*" y modificado el error del fichero de Apache, se puede proceder con la instalación de DevStack, no sin

antes indicar que este intento de instalación fallará, tendremos que solucionar el error y volver a lanzarla.

Esta solución no se puede hacer antes de realizar el primer intento de instalación, debido a que se trata de un problema de permisos en unos archivos y directorios que se crean y descargan durante la instalación.

Cuando nos salte el error, un problema indicando que "*pip*" no tiene permisos para realizar cambios o sobre un directorio "*.config*", tendremos que modificar el propietario y el grupo al que pertenece dos directorio de manera recursiva para que se aplique a todos los directorio y ficheros del árbol. Esta operación la conseguiremos con el uso del comando siguiente:

```
$ sudo chown -R stack:stack .cache .config
```

La modificación del directorio "*.cache*" soluciona el problema de permisos de pip y "*.config*" solventa también otro problema de permisos porque el proceso de instalación de Devstack no puede acceder.

Habiendo solucionado este problema, bastará con volver a lanzar el proceso de instalación y esperar a que termine. Es importante destacar que entre intentos de instalación se ha limpiado la instalación anterior porque de lo contrario se pueden dar más fallos debido a que algunos servicios ya están funcionando como demonios en el sistema y no se pueden realizar ciertas operaciones sobre ellos.

Para realizar cada intento de instalación se ha usado el comando siguiente, que además incluye el temporizador para saber lo que ha tardado en terminar, esta última instalación tomó casi media hora en finalizar.

```
$ time ./clean.sh && time ./stack.sh
```

El comando deberá lanzarse desde el directorio devstack, el script "*clean.sh*" se encarga de eliminar instalaciones anteriores y el script "*stack.sh*" en realizar una nueva instalación cogiendo los valores definidos en el fichero "*local.conf*".

Cuando la instalación finalice deberemos de observar una salida por el terminal como la mostrada en la figura 6.6, en donde apreciamos que la

instalación ha tardado casi 28 minutos. Por otro lado, la figura 6.7 muestra el listado de servicios instalados en el entorno definitivo de Devstack, y en esta ocasión consumen 11.3 Gb con la instalación recien finalizada.

En este punto, tenemos una instalación totalmente operativa para comenzar a realizar despliegues de instancias virtuales, contenedores o redes completas con firewalls, routers, etc.

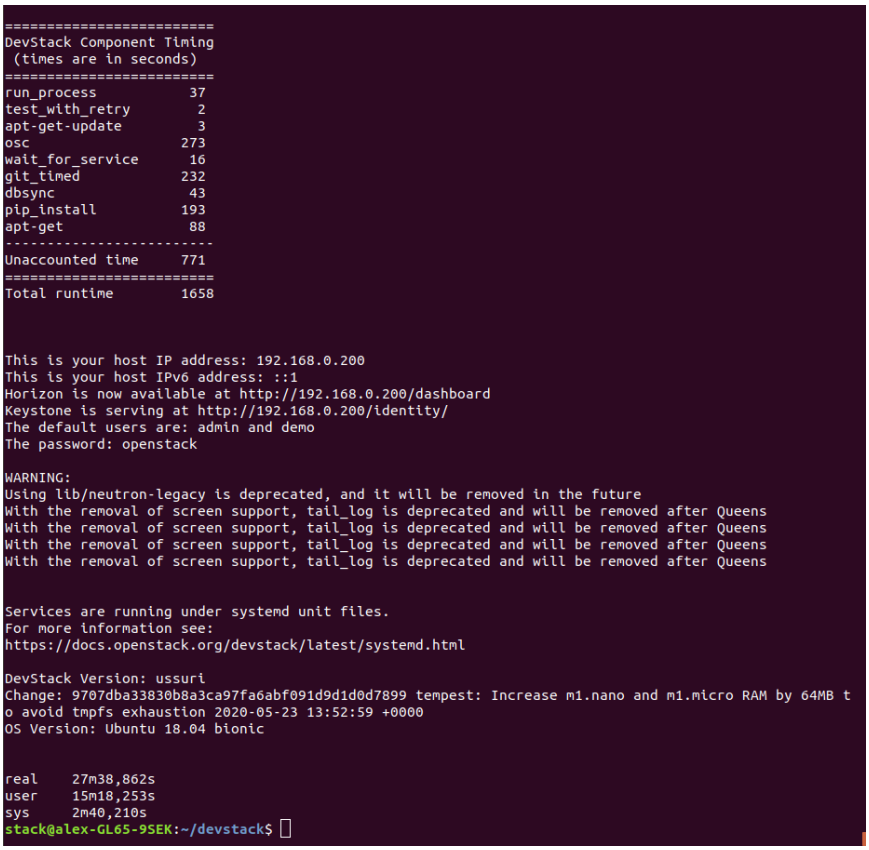

Figura 6.6: Finalización de instalación DevStack.

Es necesario indicar que para interactuar por línea de comandos con el entorno de OpenStack es necesario autenticarse y para ello se ha creado un fichero que carga la información de usuario, contraseña y proyecto en las variables de entorno del sistema, facilitando el uso del cliente CLI de OpenStack.

```
alex@alex-GL65-9SEK:~$ openstack service list
+----------------------------------+------------------+------------------+
| ID                               | Name             | Type             |
+----------------------------------+------------------+------------------+
| 0b422c86e3bf441d9d0e139168b33e32 | gnocchi          | metric           |
| 3299e50ae2c44676b72f7df652362017 | heat             | orchestration    |
| 41496ed13391415e89cba3b6208ac34d | cinderv2         | volumev2         |
| 4c5c6dd224d94d8699ea50ee6e7a6770 | placement        | placement        |
| 52ae7bb8da3843ca8c69d925098c9b17 | kuryr-libnetwork | kuryr-libnetwork |
| 5acd2dd2174849c18b555ef9d44a95a4 | glance           | image            |
| 5c3ed5fcdec14471b5afe0024b9cac3b | neutron          | network          |
| 5d343e7ec549445984331661409b3d96 | keystone         | identity         |
| 6b44842dead945719e7fdc7e7b9f72fe | magnum           | container-infra  |
| 767054863aa84eccbf8669a671821e4d | cinderv3         | volumev3         |
| 7d63174f9c314cbba27706099e81b44f | heat-cfn         | cloudformation   |
| 8fd64777c1b842008f7d630369320e02 | cinder           | block-storage    |
| ad0cbef7cc5d4acbb500c880b9ca0631 | swift            | object-store     |
| ad447ffe75044b28be80bbfce02f379b | nova             | compute          |
| b973b8ca2c99426695126e2ba22ebae0 | zun              | container        |
| c79a273855514c0f8fa1dbe92b923381 | panko            | event            |
| cd48eb7b239b46a8a297bca774244cd1 | nova_legacy      | compute_legacy   |
| d8782370e81843c693809eb8f312ecb8 | aodh             | alarming         |
| fb458c8aaf2d4738aa0375c24381dd36 | barbican         | key-manager      |
+----------------------------------+------------------+------------------+
alex@alex-GL65-9SEK:~$
```

Figura 6.7: Servicios desplegados en instalación final.

6.3. Instalacion de Open Source MANO.

Este apartado recogerá todas las acciones necesarias para realizar la preconfiguración del sistema operativo, instalación y configuración de la plataforma Open Source Mano, incluso los intentos erróneos de instalación que apoyan el uso de la versión finalmente utilizada.

El ordenador que se utilizado para la plataforma de Open Source Mano tiene las siguientes características:

- Sistema operativo Ubuntu 18.04.4 LTS.
- Procesador Intel Core i5-3230M.
- Memoria RAM de 8 Gb.
- Disco duro HDD de 750 Gb.

La versión de OSM instalada es la siete, aunque se comenzó probando con el despliegue de la última versión lanzada, versión ocho, obteniendo algunos errores durante la instalación que evitaban que funcionase correctamente toda la plataforma, por ejemplo la autenticación en la interfaz web no funcionaba y no permitía hacer uso de la GUI de OSM.

OSM presenta varias opciones de instalación, se puede realizar mediante la utilización de Docker Swarm - opción instalada finalmente -, Kubernetes o mediante Charms y Juju - distribución de Canonical.

Se empezó intentando instalar la última versión de OSM basada en docker swarm, la instalación finalizó aparentemente de forma satisfactoria pero cuando se intentaba probar su funcionamiento mediante la interfaz web nos dimos cuenta de que algo había fallado, dado que no permitía la autenticación de los usuarios y, por lo tanto, la operación sobre el entorno de orquestación de NFV.

Ante este problema se intentó realizar la instalación por otros medios, Charmed OSM de Canonical - esta alternativa emplea Juju y un cliente OSM de la tienda "snap" de Ubuntu-, apoyado por la necesidad de menos recursos se pensó como una alternativa viable. Sin embargo, en mitad del proceso de instalación nos surgió un error que evitaba definir las conexiones entre el snap de Juju y el cliente de OSM, necesario para obtener configuración para continuar la instalación.

En este punto se buscó información del error y los propios desarrolladores encargados de esta alternativa proponían la instalación basada en Charms propia de OSM. Esta alternativa también se probó pero sin mejores resultados.

Viendo las novedades que introducía la última versión de OSM, release ocho, frente a la anterior más probada, se decidió con la instalación de la "*release SEVEN*", versión utilizada actualmente.

Ahora sabiendo el procedimiento de decisión sobre la versión instalada se procederá con la explicación paso a paso de la implementación de la plataforma OSM.

En primer lugar, como se ha mencionado anteriormente, el sistema operativo utilizado en el PC es Ubuntu 18.04.4 LTS y al tratarse de un servicio que debe de estar activo en una dirección web para poder interactuar con él, será necesaria la asignación de forma manual de una dirección IP estática. La opción más sencilla es mediante la interfaz gráfica de configuración del sistema.

Elegimos la opción de "*Red* $\rightarrow$ *Cableada*" y seleccionamos el botón con forma de engranaje para abrir el panel de configuración de dicha interfaz

de red. En la ventana emergente seleccionamos la opción "*IPv4 → Método IPv4 → Manual*", rellenaremos la información y para que los efectos tengan resultado clicamos en el botón "*Aplicar*".

Figura 6.8: Configuración IP estática interfaz gráfica.

Otra opción de configuración de una dirección IP estática es mediante línea de comandos y la edición de los ficheros que contiene la información de las interfaces de red. En la versión Ubuntu 18.04 el gestor de las interfaces de red se ha visto modificado y ahora se emplea "*Netplan*". Para modificar los datos de la interfaces de red deberemos de modificar el fichero localizado en la ruta "*/etc/netplan/01-network-manager-all.yaml*", la información introducida deberá tener el siguiente formato:

```
network:
    version: 2
    renderer: NetworkManager
    ethernets:
       [DEVICE_NAME]:
          dhcp4: false
          addresses: [IP_ADDRESS/NETMASK]
          gateway4: GATEWAY_IP
          nameservers:
             addresses: [NAMESERVER_1, NAMESERVER_2]
```

Una vez se haya modificado el fichero tenemos que ejecutar un comando para que se apliquen los cambios, es recomendable utilizar la opción de depuración.

```
$ sudo netplan --debug apply
```

Llegados a este punto podemos iniciar la instalación de OSM, basada en Docker Swarm, se ha elegido esa opción sobre la de Kubernetes por el hecho de que requiere menos recursos.

Vistos los problemas anteriores en los intentos de instalación se guardará el log del proceso completo para facilitar la resolución de errores en caso de que sea necesario, algo recomendado por la propia comunidad de OSM en su documentación.

Primero deberemos descargar el instalador de OSM, un script que realiza la descarga, instalación y configuración de todos los componentes del entorno. Se le dan permisos de ejecución y se lanza guardando el log de instalación en un fichero.

```
$ wget https://osm-download.etsi.org/ftp/osm-7.0-seven/install_osm.sh
$ chmod +x install_osm.sh
$ ./install_osm.sh 2>&1 | tee osm_install_log.txt
```

Cuando se lance el proceso, para proceder con la instalación se preguntará si se desea instalar y configurar LXD, Juju, Docker CE y la inicialización de un docker swarm, tendremos que indicar "*y*", para que continúe la instalación.

Cuando termine el proceso de instalación de OSM deberemos obtener una salida parecida a la mostrada en la figura 6.9, donde se muestra que el proceso ha finalizado y el servicio está funcionando.

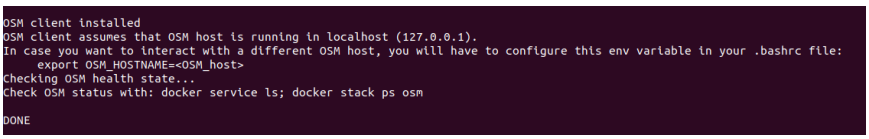

Figura 6.9: Instalación satisfactoria de OSM.

De la figura 6.9, también podemos ver que para interactuar con el cliente de de OSM desde otro host diferente, se deberá configurar una variable de entorno indicando dónde se encuentra el servicio de OSM, con el comando siguiente se puede añadir la variable de forma temporal a la sesión de

línea comandos abierta, en caso de quererlo forma permanente será necesario incluirlo en el fichero "*.bashrc*".

```
$ export OMS_HOSTNAME=<OSM_host>
```

Donde *<OSM_host>* es la dirección IP del equipo donde se ejecuta el entorno OSM.

Hay destacar que en algunas ocasiones el proceso de instalación presenta errores de permisos cuando ejecuta alguna operación con Docker, esto se debe a que por defecto Docker solo funciona utilizando su cliente de línea de comandos como usuario "*sudo*".

Para solucionarlo basta con realizar el siguiente procedimiento:

- Crear el grupo docker:

```
$ sudo groupadd docker
```

- Añadir el usuario actual al grupo:

```
$ sudo usermod -aG docker ${USER}
```

- Reevaluar la pertenencia al grupo, aplicando cambios en los grupos:

```
$ newgrp docker
```

Una vez que hemos configurado el grupo para docker tendremos que volver a intentar instalar OSM, pero si lo hacemos directamente nos toparemos con un nuevo problema, se detecta que ya existe un controlador osm y no se puede instalar. Para solucionarlo en la documentación de OSM nos indica que lancemos el comando:

```
$ juju destroy-controller osm --destroy-all-models -y
```

En nuestro caso, este comando no surte efecto y tenemos que hacerlo de otro modo, mediante el listado y eliminación de los contenedores juju y controladores.

Listamos los contenedores de Juju ejecutados sobre LXC:

```
$ lxc list
```

Del listado elegimos el contenedor que deseamos eliminar y lanzamos las directivas de parada y eliminación del contenedor (juju-98c111-0 es un nombre a modo de ejemplo).

```
$ lxc stop juju-98c111-0
$ lxc delete juju-98c111-0
```

Por último, se debe de eliminar el registro del controlador de osm y liberar los recursos asociados:

```
$ juju unregister -y osm
```

Para comprobar que se ha eliminado el controlador y se puede proceder nuevamente con la instalación desde cero, se puede lanzar un comando que lista todos los controladores registrados en Juju.

```
$ juju list-controllers
```

Una vez que finaliza la instalación con un resultado satisfactorio, podemos proceder a comprobar su funcionamiento. Primero de todo podremos probar el acceso a la interfaz web de administración, utilizando la URL: http://<IP-host>, siendo "*IP-host*" la dirección IP del equipo donde se ha instalado, o si se realiza el acceso desde el mismo equipos mediante localhost.

Para iniciar sesión se utilizarán las credenciales por defecto, usuario: admin y contraseña: admin.

Como resultado de la instalación, se crean 13 contenedores docker en el host. Para comprobar que están corriendo y no presentan errores se puede lanzar el siguiente comando (véase figura 6.11):

```
$ docker stack ps osm |grep -i running
```

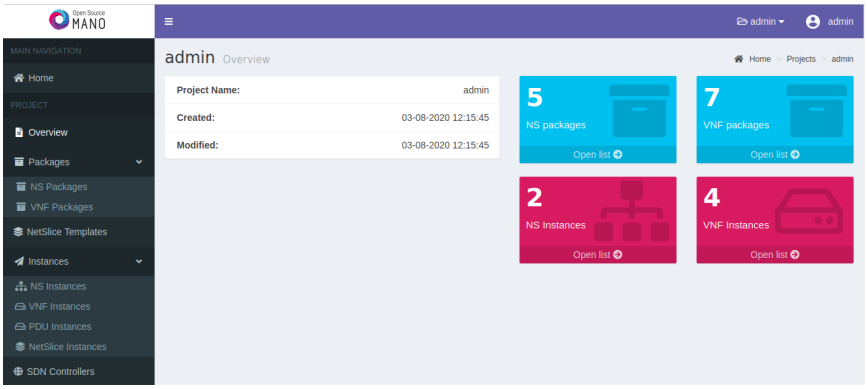

Figura 6.10: Interfaz gráfica de OSM.

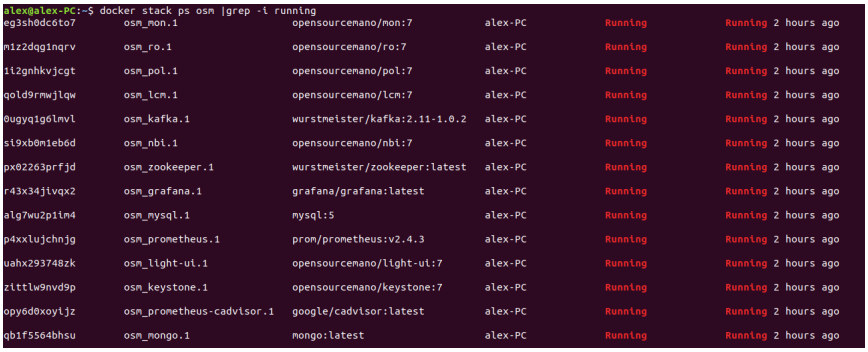

Figura 6.11: Contenedores docker lanzados tras instalación.

U otro que permite observar el control automatizado de los servicios desplegados dentro de Docker, en este caso siempre deberemos tener como mínimo una réplica en cada uno (véase figura 6.12)

```
$ docker service ls
```

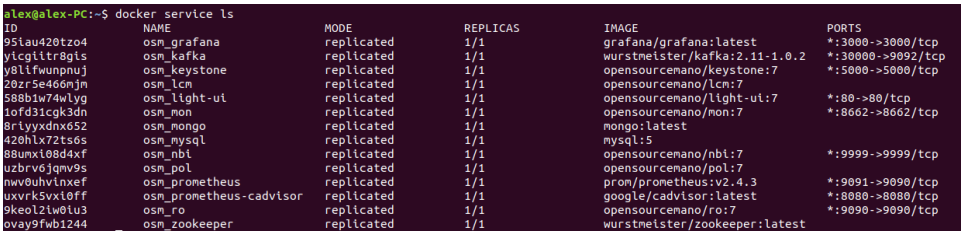

Figura 6.12: Servicios funcionando sobre docker.

Un aspecto a tener muy en cuenta en este tipo de instalación es la posibilidad de comprobar posibles errores en los logs de cada uno de los contenedores, los comandos siguientes permite observar el log del contenedor "*osm_lcm*" a modo de ejemplo:

```
$ docker service logs osm_lcm    #muestra el logs de todos los
    contenedores asociados a ese nombre, incluidos los contenedores
    parados.

$ docker logs $(docker ps -aqf "name=osm_lcm" -n 1)   #muestra el log
    del  ltimo  contenedor de LCM que ha sido parado.
```

Otro mecanismo para controlar el funcionamiento del sistema es gracias al uso de Grafana, sistema de monitorización incluido en la instalación y operación de OSM.

Para acceder a la interfaz gráfica de Grafana, tendremos que dirigirnos a la URL, http://<IP-host>:3000 , al igual que para el caso anterior IP-host es la dirección IP del host donde se encuentra instalado OSM, salvo que en este caso se le indica que utilice el puerto 3000 para la conexión, dándonos acceso al panel de control de Grafana.

Las credenciales de acceso son las mismas, por defecto, que para la interfaz de OSM. (usuario: admin y contraseña: admin).

Para ver la información que nos interesa tendremos que elegir el panel que queremos, OSM System Metrics, en la esquina superior izquierda. Este panel nos muestra los recursos consumidos del ordenador anfitrión y su estado (véase figura 6.13). Además, muestra información referente a los recursos empleados por los contenedores docker desplegados en la instalación de OSM (véase figura 6.14).

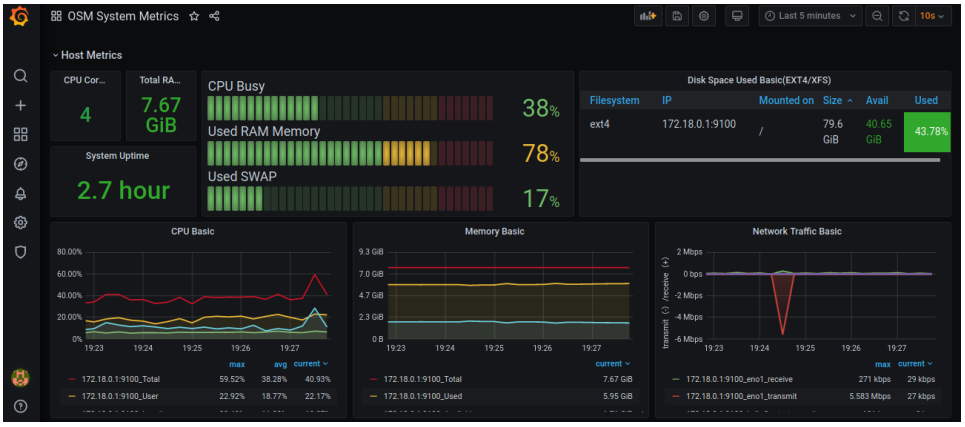

Figura 6.13: Consumo de recursos del PC anfitrión.

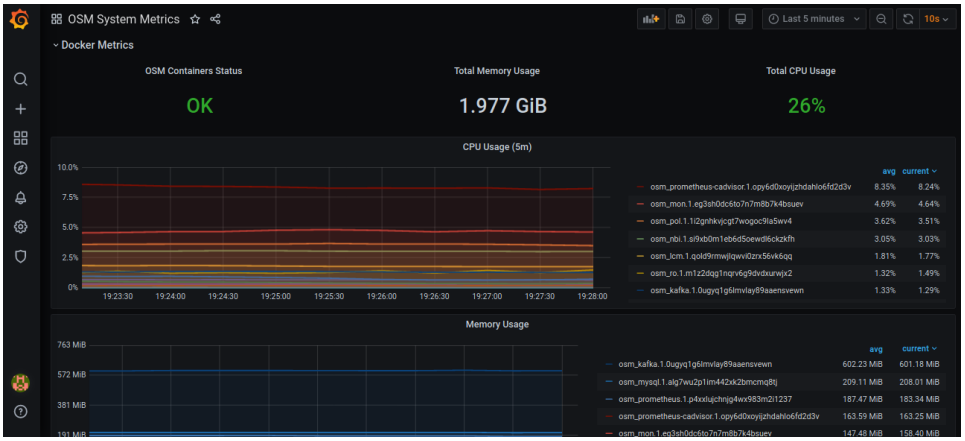

Figura 6.14: Recursos empleados por contenedores Docker.

6.4. Configuración adicional.

En este apartado se recogerán las configuraciones realizadas para crear el escenario de pruebas entre OpenStack y OSM, cómo se realiza la conexión para que definiendo una VNF y un NS en OSM , las máquinas virtuales y redes definidas se desplieguen en la infraestructura de OpenStack.

6.4.1. Configuración de entorno OpenStack.

Para poder utilizar la infraestructura virtual de OpenStack de manera idónea en nuestro proyecto, tendremos que realizar una serie de configuraciones, creación y administración de algunos elementos que nos facilitará en gran medida la operación con los componentes de OpenStack.

Partiendo de una instalación limpia de OpenStack es necesario crear algunos elementos para poder empezar a funcionar: par de claves público-privada, grupo de seguridad, asignación servidores DNS en las redes. Adicionalmente, se añadirá una nueva red para que operen las VNFs lanzadas desde OSM en el escenario de pruebas.

La creación de un par de claves es fundamental para poder gestionar las instancias virtuales que se levanten en la infraestructura a través de SSH. En caso de no utilizar el par de claves, el único medio para administrar las instancias virtuales es mediante la consola disponible en la información de la instancia, basada en VNC.

Hay que destacar que todas las operaciones que se van a describir aquí pueden ser creadas mediante dos opciones, o bien mediante la interfaz web o por línea de comandos con el cliente de openstack. Para crearlo por la

interfaz web tendremos que ir a pestaña "*Project*", abrir la opción "*Compute*", y del desplegable, seleccionar "*Key Pairs*". Seleccionaremos en la parte derecha el botón "*Create Key Pair*", le asignamos un nombre y elegimos tipo SSH, y finalizamos pulsando sobre el botón azul "*Create Key Pair*". Automáticamente, se desplegará una ventana avisandonos para guardar la clave en nuestro PC.

Antes de poder utilizar la clave para conectarnos por SSH a alguna instancia tendremos que modificar los permisos y dejarlos como lectura y escritura para sólo el propietario. Esto lo hacemos con el comando:

```
$ sudo chmod 600 mykey.pem
```

Para el caso de crearlo por línea de comandos, lo haremos con el comando siguiente del cliente de CLI de Openstack:

```
$ openstack keypair create mykey > mykey.pem
```

En este punto podríamos intentar conectarnos a alguna instancia que tengamos levantada, pero ni siquiera vamos a poder hacer ping a ella, esto se debe a que por defecto se aplica un grupo de seguridad que aplica reglas de firewall y no se aplican correctamente aquellas que permiten todo el tráfico. Por ello es necesario realizar algunas modificaciones de reglas o directamente crear un grupo de seguridad nuevo.

Para crear un grupo de seguridad tendremos que dirigirnos a "*Project - Network - Security Groups*". Pulsaremos sobre el botón "*Create Security Group*", asignamos un nombre y una descripción (opcional) y clicamos sobre el botón azul "*Create Security Group*".

Una vez creado el grupo de seguridad tendremos que administrar las reglas que se emplean. Esto se administra de forma sencilla en el botón "*Manage rules*". En el nuevo menú veremos el botón "*Add Rule*". En función de la regla que se defina tendremos una serie de opciones. En la figura 6.15 podremos ver el ejemplo de definición de una regla para permitir SSH desde cualquier red.

Para operar correctamente con las instancias, permitiendo conexiones SSH, conexiones desde la instancia hacia el exterior con resolución de nombre y permitir pruebas para saber si una instancia está levantada o no, deberemos de definir reglas para los protocolos SSH, ICMP y DNS. Además, se permiten cualquier tipo de tráfico saliente y entrante y entrante (a modo de prueba, porque no se trata de un entorno de producción, de lo contrario tendríamos un grave problema de seguridad.)

En el caso de que se quieran permitir otro puertos de red u otros protocolos deberemos de indicarlo en el grupo de seguridad creando una regla específica para ello.

Otra opción que es necesario configurar son los servidores DNS que funcionan en una red, esto se configura en la información de la subred que queremos utilizar.

En el panel lateral navegamos por "*Project - Network - Networks*", elegimos la red que deseamos y vamos a la pestaña "*Subnets*" en el panel superior. Para asignar los servidores DNS tendremos que clicar encima del botón "Edit Subnet", se abrirá una ventana emergente en mitad de la pantalla y elegiremos "*Subnet Details*". Llegados a este punto podremos ver que se puede configurar el conjunto de direcciones IP asignadas dentro de la red, los servidores DNS - que son los que nos interesan ahora mismo -, y rutas estáticas para todos los host virtuales que se conecten a dicha subred. En la figura 6.16 se puede ver esta ventana emergente.

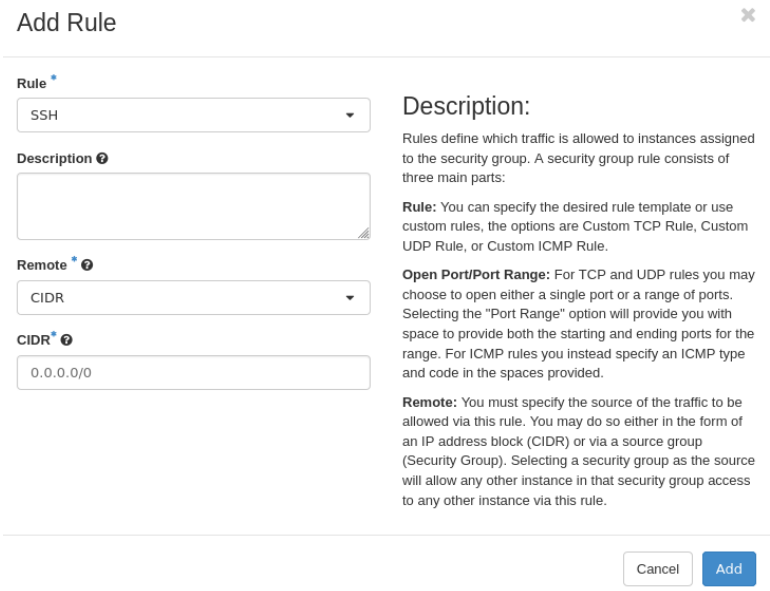

Figura 6.15: Definición de regla SSH.

A modo de preparación de la infraestructura para su uso con OSM, se creará una red nueva donde se desplegarán tanto las interfaces creadas por VNFs y router para redireccionar las peticiones o pruebas desde el host anfitrión, sin necesidad de emplear direcciones IP flotantes.

Para crear la nueva red vamos a "*Project - Network - Networks*", clicamos en "*Create Network*". Emergerá una ventana donde tendremos que rellenar los datos de información, seleccionaremos la opción "*Shared*" para que pueda utilizarse con cualquier proyecto de OpenStack. En la pestaña "*Subnet*", rellenamos todos los datos, nombre, dirección de red, puerta de enlace. Por último, tenemos que rellenar los detalles de la subred al igual que se hizo con la configuración de los servidores DNS.

Una vez se ha creado la nueva red, se deberá crear un router que conecta la red pública (*public*) con la nueva (*osm-ext*). Para crear el router vamos a "*Project - Network - Routers*", pulsamos sobre el botón "*Create Router*" y saldrá nuevamente una ventana emergente. En esta ventana tendremos que ponerle un nombre al router y elegir la red externa que queremos, en nuestro caso "*public*", finalizamos el proceso pulsando "*Create Router*".

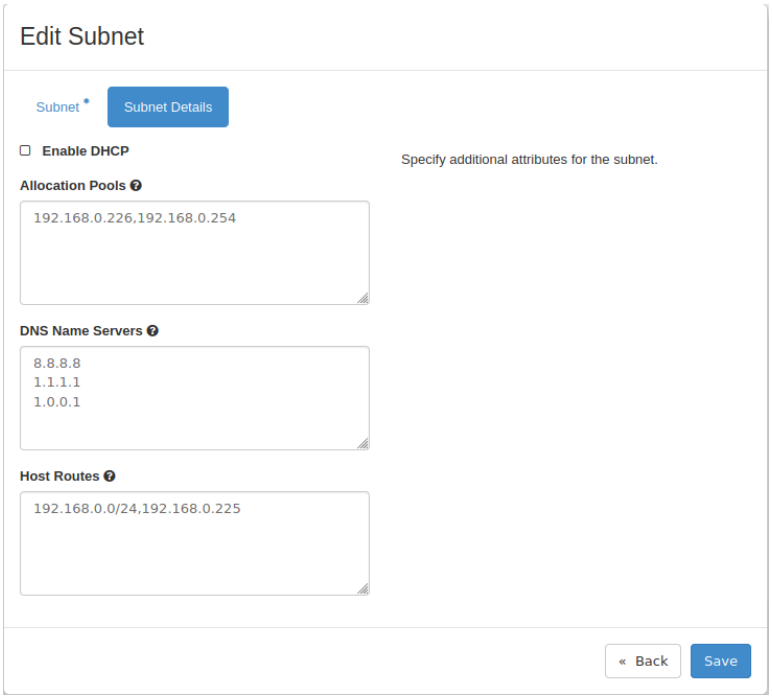

Figura 6.16: Configuración subred, asignación servidores DNS.

Bien, ya tenemos creado el router, pero ahora tenemos que configurar una nueva interfaz de red conectada a la red creada, para ello clicamos encima del nombre del router, como si fuéramos a ver las propiedades. En la parte superior, vemos las pestañas "*Interfaces*" y "*Static Routes*". Seleccionamos "*Interfaces*", "*Add Interface*" y elegimos la red que queremos (*osm-ext*) y le asignamos una dirección IP, para confirmarlo pulsamos "*Submit*".

Para poder conectar con estas interfaces es necesario deshabilitar el puerto de seguridad o por el contrario asignarle un grupo de seguridad para que aplique las reglas de firewall, por defecto está habilitado como medida anti-spoofing.

Además, para evitar tener que configurar las rutas para conectarse con los equipos de la red externa a OpenStack en las VMs, se define una ruta estática en el router recién creado indicando como siguiente salto un router que crea OpenStack (en nuestro caso tiene dirección IP, *192.168.0.225*) durante la instalación que permite conectar la red externa a la infraestructura virtual. Esto se realiza en la pestaña "*Static Routes*", pulsamos sobre el botón "*Add Static Route*" e indicamos la dirección de red destino y el siguiente salto. En la figura 6.17, se muestra un ejemplo de ello.

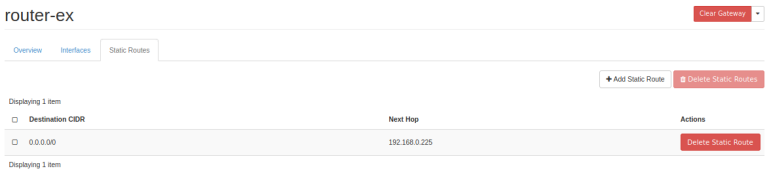

Figura 6.17: Muestra de ruta estática en router OpenStack.

Para finalizar, es necesario comprobar las rutas de red definidas en el PC desde el que vamos a realizar la conexión a las instancias. En caso de no tener reglas definidas para esas redes, será necesario crearlas. Por ejemplo, para conectar con las instancias de la red "*osm-ext*", tendremos que fijar una ruta para que el PC sepa por donde tiene que enviar el tráfico. Con el comando siguiente creamos la regla para llegar a las máquinas desplegadas en la nueva red en el PC host de OpenStack.

```
$ sudo route add -net 192.168.1.0/24 gw 192.168.0.239 dev br-ex
```

6.4.2. Conexión OSM con OpenStack.

Este apartado recoge el mecanismo para asociar OpenStack como VIM de OSM, cómo se puede monitorizar mediante Grafana y una prueba de uso con la que corroborar que efectivamente funciona.

En primer lugar, nos centraremos en la definición de OpenStack como VIM para OSM. Para realizar este proceso de configuración tenemos dos opciones sencillas, o mediante línea de comandos o por la interfaz web de OSM.

Para realizarlo a través de línea de comandos tendremos que utilizar el cliente CLI de OSM con la opción "*vim-create*", como se muestra a continuación:

```
$ osm vim-create --name openstack-site --user admin --password
    openstack --auth_url http://192.168.0.200/identity/v3 --tenant
    admin --account_type openstack --config='{security_groups: default
    , keypair: mykey}'
```

En el comando anterior indicamos el nombre del vim con el que será reconocido por OSM para desplegar NFVs. Será necesario indicar usuario, contraseña, proyecto (*tenant*) y URL de autenticación para poder autenticarse con un usuario en la API de OpenStack y poder interactuar con la infraestructura. Para finalizar se indica el tipo de VIM utilizado y una breve configuración con valores <*clave:valor*>.

Con los comandos "*osm vim-list*" y "*osm vim-show <nombre-VIM>*", se pueden listar los VIM indicados a OSM y mostrar toda la información referentes a un VIM concreto, respectivamente.

Por otro lado, se puede configurar mediante la interfaz web de OSM, para ello bastará con buscar la pestaña "*VIM Accounts*" en la lista del lateral, pulsaremos encima del botón "*New VIM*", y rellenamos la ventana emergente como se muestra en la figura 6.18.

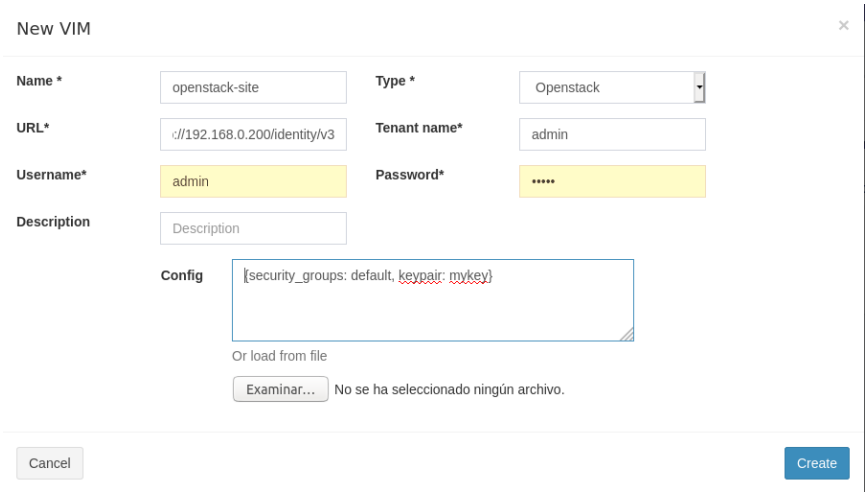

Figura 6.18: Creación de nuevo VIM en OSM.

Para saber que ha concluido con éxito la creación se puede ver que "*Operational State*" esté en "*ENABLED*" y la información del VIM puede obtenerse o mediante el comando mencionado anteriormente o pulsando sobre el botón de la "*i*".

Como medida para comprobar que funciona el despliegue de VNFs como instancias virtuales en el VIM asociado se va realizar el ejemplo de uso de la documentación de OSM, desplegando dos VNFs simples basada en el sistema operativo CirrOS y conectados por una simple VLD (*virtual link descriptor*)

Primero de todo tendremos que descargarnos la imagen de SO a utilizar (se puede obtener desde el enlace: `http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img`) y crearla en OpenStack desde línea de comandos o por la interfaz de Horizon. Mediante la utilización del siguiente se crea la imagen con el cliente CLI:

```
openstack image create --file="./cirros-0.3.4-x86_64-disk.img" --
    container-format=bare --disk-format=qcow2 cirros034
```

Para realizar la prueba será necesario descargar los ficheros comprimidos (*tar.gz*) proporcionado por OSM en el enlace: `https://osm-download.etsi.org/ftp/osm-3.0-three/examples/cirros_2vnf_ns/`

Teniendo los ficheros será necesario añadirlos a OSM. Primero será necesario añadir el paquete VNF, en el desplegable de la derecha se elige la opción "*VNF Packages*" y en se arrastra el fichero sobre el área indicada para importarlo. Del mismo modo, procedemos con el "*NS Packages*".

El último paso que queda es instanciar el NS, al igual que el resto de operaciones se puede hacer mediante el cliente CLI o por la interfaz web. Por método gráfico, se irá al apartado "*NS Package*" y en el NSD deseado de la lista procederemos a clicar encima del botón "*Instantiate NS*", nuevamente saltará una ventana emergente que rellenaremos y pulsaremos sobre el botón "*Create*".

Tras un poco de tiempo, si tenemos abierta la interfaz web de OpenStack podremos observar que se crea una red nueva (*cirros_2vnf_nsd_vld1*) y dos instancias.

Mientras tanto en OSM, en el apartado "*NS Instances*" se puede ver información del proceso de instanciación, controlando el estado de operación y configuración que debe de en verde y en "*running*" y "*configured*", respectivamente.

Para controlar que el VIM esté activo y funcionando correctamente es posible hacerlo a través de Grafana. Además, permite ver información del estado de VNFs y NSs desplegados.

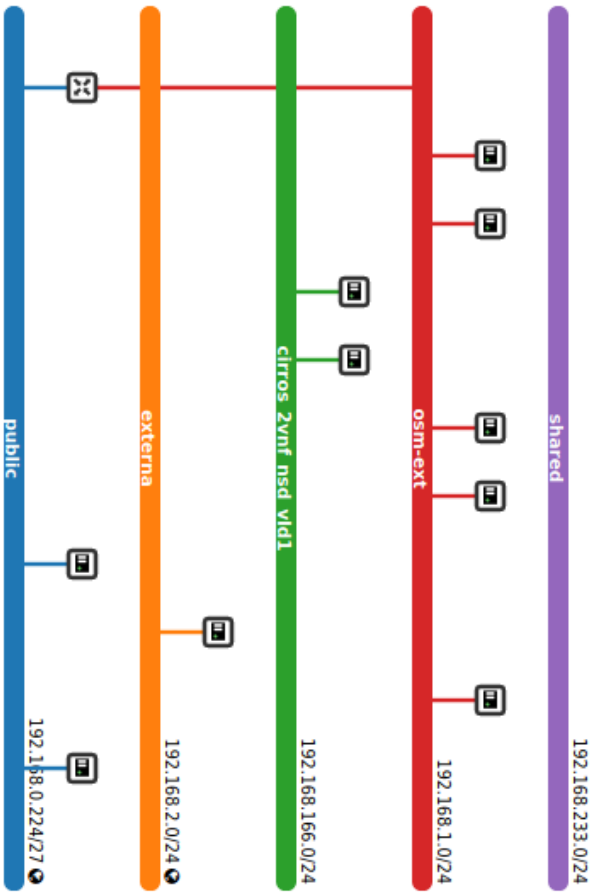

Figura 6.19: Topología de red OpenStack, con red(verde) e instancias lanzadas desde OSM.

Entramos en Grafana (*http://<direccion-IP-host-OSM>:3000*), nos autenticamos con usuario y contraseña "*admin*" y en la esquina superior izquierda elegimos el proyecto que queremos, tenemos que elegir "*OSM Project Status - admin*".

Una muestra de la información que nos brinda Grafana podemos verla en la figura 6.19.

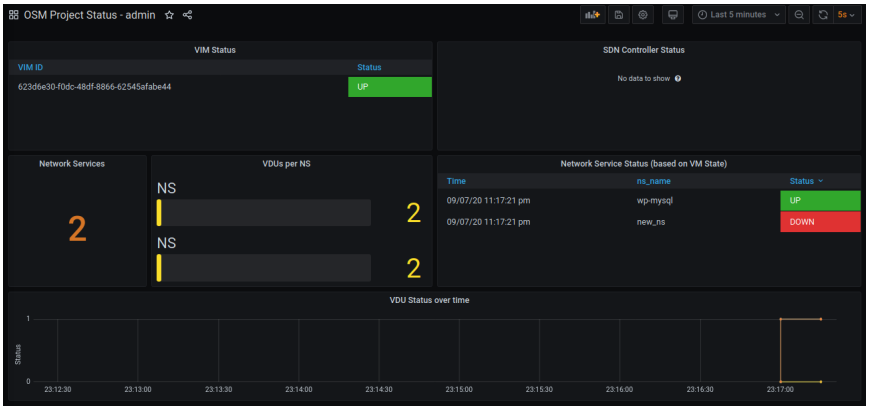

Figura 6.20: Información de VIM mostrada por Grafana.

6.4.3. Intento de instalación de Kubernetes para KNFs.

A partir de la versión siete, OSM permite la opción de definir un clúster de Kubernetes - conjunto de nodos para ejecutar aplicaciones de Kubernetes - y utilizarlo para desplegar VNFs en él. En la documentación se indican las tres maneras de instalación de Kubernetes para OSM, se han intentado instalar pero siempre han saltado errores y no ha sido posibles conseguirlo.

Además de esta opción para emplear Kubernetes, se tiene el componente Magnum de OpenStack, el cual se ha intentado utilizar también y tampoco ha sido posible.

Puesto que no se ha conseguido desplegar este servicio se expondrán las diferentes operaciones realizadas, aunque sin entrar en demasiados detalles, indicando las referencia de la documentación seguida.

En primer lugar, se comenzará con el despliegue de Kubernetes mediante los tres mecanismo indicados por OSM. Para que el clúster de Kubernetes funcione correctamente en OSM, es necesario que cumpla una serie de requisitos:

- Kubernetes tenga balanceado de carga para exponer los KNFs a la red.
- Kubernetes tenga por defecto en almacenamiento *"Storageclass"* para soportar volúmenes persistentes.
- Permisos de "*Tiller*" para clústeres de K8s.

Teniendo los requisitos claros, podemos pasar a comentar los tres métodos para crear un clúster de Kubernetes.

- Cluster de Kubernetes creado como un servicio de red (*NS*) de OSM.
- Entorno local de desarrollo basado en microK8s.
- Instalación manual del clúster basada en "*kubeadm*".

- **Cluster de Kubernetes como servicio de red (NS) de OSM**

  Según la documentación el clúster de Kubernetes se va a desplegar mediante la definición de VNFs y NS con OSM. El network service está compuesto por un nodo controlador (*k8s_jujucontroller_vnf*) y cuatro nodos trabajadores (*k8s_jujumachine_vnf*) conectados a una red. El nodo controlador y encargado de desplegar el clúster tiene un instalador de Kubernetes basado en Juju que configura los otros cuatros nodos.

  El primer problema surge al comprobar las descripciones de las VNF, donde se le asignan unos recursos inasumibles a cada una de las máquinas virtuales que serían desplegadas en el clúster. El nodo controlador tiene 4Gb de RAM y 2 CPUs y cada uno de los nodos trabajadores tiene 16Gb de RAM y 8 CPUs.

  Los descriptores se modificaron y se le asignó a cada nodo 2Gb de RAM y 2 CPUs, y en lugar de lanzar cuatro nodos trabajadores se probó tanto con dos como con uno solo.

  Para realizar la instalación de este clúster se tienen que lanzar los siguientes comandos:

```
$ wget http://osm-download.etsi.org/ftp/Packages/hackfests/
    k8s_jujucontroller_vnf.tar.gz
$ wget http://osm-download.etsi.org/ftp/Packages/hackfests/
    k8s_jujumachine_vnf.tar.gz
$ wget http://osm-download.etsi.org/ftp/Packages/hackfests/
    k8s_juju_ns.tar.gz
$ osm nfpkg-create k8s_jujumachine_vnf.tar.gz
$ osm nfpkg-create k8s_jujucontroller_vnf.tar.gz
$ osm nspkg-create k8s_juju_ns.tar.gz
```

Cuando se han añadido las descripciones de VNFs a OSM, podremos a través de la interfaz web modificar los valores de recursos asignados a cada nodo, antes de proceder con la instanciación del NS y ver si la instalación funciona.

Para instanciar el NS se realizará por el cliente CLI con el comando:

```
$ osm ns-create --ns_name k8s-cluster --nsd_name k8s_juju --
    vim_account openstack-site --config_file config.yaml --
    ssh_keys ${HOME}/.ssh/id_rsa.pub
```

Uno de los parámetros más importantes del comando anterior es "*–config-file*" al que se pasa un fichero de configuración en formato YAML, con información adicional sobre las direcciones IP de los nodos y la red de la infraestructura virtual que se va a utilizar.

Para información sobre las VNF se puede usar la opción "*vnf-list*", obteniendo la dirección IP finalmente asignada. Sabiendo la dirección IP podemos conectarnos a la instancia del nodo controlador y obtener el fichero de configuración del cluster de Kubernetes. Este fichero es necesario para poder asociar el clúster a OSM y posteriormente desplegar KNFs.

En nuestro caso, no fue así y se quedó la configuración de los nodos a medias sin entender muy bien por qué. Por lo tanto, no se pudo desplegar el clúster de Kubernetes y obtener el fichero de configuración para asociar el clúster a OSM.

- **Entorno local de desarrollo basado en microK8s**

Otra opción de instalación de Kubernetes es mediante un único paquete completo y ligero, capaz de funcionar en multitud de distribuciones distintas de Linux. Esta opción se conoce como MicroK8s.

El procedimiento a seguir fue crear una instancia en OpenStack con una distribución de Ubuntu, se configuró para que pudiera comunicarse con máquinas externas a OpenStack pensando en conectarla con el host de OSM.

Teniendo configurada la instancia virtuales, se proceder a instalar MicroK8s, configurar el usuario actual para que pueda hacer uso de

microk8s, y a ver el estado de la instalación con la ejecución de los siguientes comandos:

```
$ sudo snap install microk8s  --classic
$ sudo usermod -a -G microk8s `whoami`
$ newgrp microk8s
$ microk8s.status --wait-ready
```

El último comando espera hasta que esté lista la instalación para mostrar la información.

MicroK8s funciona a base de addons que se pueden instalar y en nuestro caso necesitamos de algunos para satisfacer los requisitos de OSM. Será necesario instalar DNS, almacenamiento y *MetalLB*, como balanceador al estar instalado fuera del host de OSM. Para habilitar estos addons bastará con ejecutar la siguiente sentencia:

```
$ microk8s.enable storage dns metallb
```

En mitad del proceso de activación de los complementos se nos pide indicar un rango de direcciones IP para el balanceador. Estando configurado el clúster debemos de proceder del mismo modo que en el caso anterior, obteniendo el fichero de configuración para asociar el clúster a OSM.

Para crear el clúster de Kubernetes en OSM, o dicho de otro modo, registrarlo, tendremos que lanzar el comando siguiente (aunque también se puede hacer a través de la interfaz web, por línea de comandos resulta más cómodo)

```
$ osm k8scluster-add --creds Escritorio/microk8s_config.yaml
    --version '1.18.6' --vim openstack-site --description "My
     K8s cluster" --k8s-nets '{"net1": "public"}' microk8s
```

En el comando anterior se indica información como el fichero de configuración del clúster en donde se refleja la URL de acceso, el VIM empleado e incluso la red donde está el clúster.

En este caso, el error surge en la asociación del clúster con OSM, no es configurado correctamente surgiendo un error, por lo que no se pueden lanzar pruebas de instanciación de KNFs.

- **Instalación manual del clúster basada en "kubeadm"**

  En este caso de instalación es necesario el uso de varias máquinas virtuales, como mínimo una como nodo controlador encargado del clúster y otra como nodo trabajador.

  La instalación se lleva a cabo descargando e instalando con APT: docker, kubelet, kubeadm, kubectl y docker.io.

  Con las funcionalidades de "*kubeadm*" y "*kubectl*" se despliega un clúster de Kubernetes en un nodo y el otro nodo se configura con las mismas herramientas pero con la finalidad de asociarlo al clúster existente como nodo trabajador.

  Tras realizar toda la instalación a mano, intentar asociar el clúster de Kubernetes a OSM, surge el mismo problema que para el caso anterior. El clúster no está configurado correctamente, dando lugar a fallos e impidiendo operar con KNFs desde Open Source Mano.

Llegados a este punto se han agotado las posibilidades de desplegar un clúster de Kubernetes desde OSM, pero en la instalación que se ha realizado de OpenStack tenemos incluido el módulo Magnum, encargado de crear un clúster de la tecnología indicada (Docker Swarm, Kubernetes o Mesos) mediante el despliegue de instancias virtuales de manera automatizada gracias al empleo de Heat.

En primer lugar, es necesario crear una plantilla que se toma de referencia en la instanciación del clúster. En esta plantilla se debe indicar el motor de orquestación de contenedores, en nuestro caso Kubernetes, las especificaciones de los nodos, tanto maestros como esclavos. Estas especificaciones incluyen la imagen a utilizar, el par de claves de seguridad, el flavor (*cantidad de recursos*), tamaño del volumen para Docker, y en último lugar los datos sobre la red a utilizar, driver de red, red externa a utilizar, servidores DNS, IP flotantes, etc.

Esta prueba de despliegue se intentó realizar con la versión Ussuri de Magnum y fallaba en el despliegue del cluster porque detectaba que no se había definido un parámetro de configuración, y a pesar de definirlo seguía detectando el fallo. Es por ello que en la instalación final, el componente Magnum se ha instalado de su rama master, esto solventa ese error.

Ahora bien, cuando intenta desplegar el clúster con la plantilla generada ahora mismo surge un fallo que no se ha podido rastrear para intentar solventarlo.

Magnum emplea Heat para realizar los despliegues de clústeres, por ello para intentar averiguar qué era lo que estaba fallando nos dirigimos a evaluar el stack de Heat que se estaba desplegando, y pudimos ver que estaba prácticamente todo el sistema desplegado a falta del componente "OS::Heat::ResourceGroup" que finalmente fallaría.

De esta manera, y al fallar el componente que fue, puede estar relacionado con la cantidad tan limitada de recursos disponibles para levantar instancias o servicios en la infraestructura virtual de OpenStack.

Capítulo 7

Experimentación y validación del sistema

En este capítulo se realizará una validación de toda la configuración y funcionamiento de los diferentes servicios que componen el escenario diseñado.

Durante el desarrollo del capítulo anterior se han ido haciendo algunas pruebas que mostraban que cada servicio operaba correctamente, antes de dar un paso más y arrastrar errores. Actuando de esta manera se ha conseguido ir localizando y acotando los distintos errores que han surgido, para solucionarlos en la medida de lo posible.

Se realizarán pruebas en un escenario diseñado para desplegar un servicio mediante las diferentes opciones disponibles en la implementación final del proyecto, cuyo objetivo es comparar el tiempo de despliegue para conocer qué opción es la más idónea para desplegar servicios críticos que necesitan tardar el mínimo tiempo posible en entornos con pocos recursos.

7.1. Planteamiento del escenario y pruebas.

Se desea realizar una serie de pruebas sobre la instalación del proyecto con el objetivo de comprobar el tiempo necesario para desplegar un servicio sobre máquinas virtuales y sobre contenedores.

La pruebas serán realizadas lanzando un servidor de Wordpress en una instancia y la base de datos, mariaDB, para que funcione en otra diferente. Ambas instancias funcionan en la misma red y para poder acceder a ellas será necesario asignarles una IP flotante.

Se ha decidido realizar las pruebas con este simple servicio debido a la falta de recursos que se viene arrastrando desde el principio por la naturaleza de los programas utilizados.

En la 7.1, se detalla en escenario experimental completo utilizado, indicando la asignación de direcciones IP de los servidores de OSM y OpenStack. Las direcciones IP de las máquinas virtuales y contenedores dependen de la red que se asigne durante la definición de las plantillas de Heat o VNF y NS de OSM.

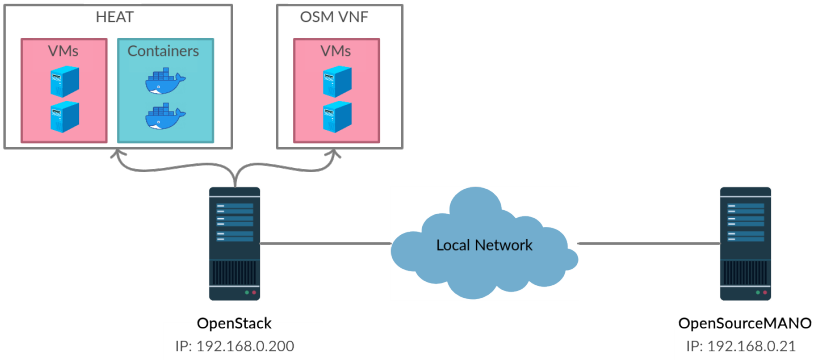

Figura 7.1: Escenario diseñado para las pruebas.

Antes de comenzar con el desarrollo de las pruebas, hay que saber ciertos aspectos condicionantes en el despliegue de las máquinas virtuales y los contenedores que apoyan la decisión de hacer el despliegue con Heat y sobre la red que se creó en el apartado 6.4.1, a pesar de tener que utilizar IPs flotantes para acceder a los servicios.

Para desplegar servicios en instancias virtuales desde OpenStack de manera automatizada es necesario emplear Heta, sin embargo para desplegar dichos servicios sobre contenedores tenemos dos opciones. La primera de ellas es utilizando las propias opciones de Zun para crear un elemento con un funcionamiento similar a los "*pods*" de Kubernetes.

Un pod es una colección de contenedores que se agrupan de forma única. Es una abstracción de alto nivel para gestionar un conjunto de contenedores que comparten volúmenes y namespaces. Se trata de la unidad mínima de despliegue en Kubernetes.

La segunda opción, consiste en utilizar Heat nuevamente para desplegar los servicios, pero esta vez en contenedores Docker.

La opción de Zun para crear conjuntos de contenedores se denomina, Capsule. Para crear este elemento es necesario el uso de plantillas. Hay que destacar que en la instalación actual de OpenStack que se ha desplegado, el elemento Capsule no despliega contenedores Docker, utiliza la herramienta "*CRICTL*" y por algún problema no se configurar bien los puertos de los contenedores ni los grupos de seguridad. Como solución se puede modificar el driver para cápsulas en el fichero "*/etc/zun/zun.conf*", cambiando el valor de la variable "*capsule_driver*" de "*cri*" a "*docker*".

Para que los cambios surtan efecto tenemos que reiniciar el servicio de Zun:

```
$ sudo systemctl restart devstack@zun-compute.service
```

Se han realizado algunas pruebas para entender su funcionamiento, entre ellas se desplegó un contenedor que levantaba un servidor web de apache y antes de configurarlo se descarga la página principal del proyecto de OpenStack La plantilla utilizada puede encontrarse en el Anexo B.

Ahora bien, cuando se intenta desplegar un servidor de Wordpress y una base de datos MySQL, los contenedores que se generan se quedan en un estado de continuo reinicio y se ha tenido que buscar otra opción para crear el servicio del escenario de pruebas utilizando contenedores de manera automatizada. En este punto entra en juego Heat con los elementos para desplegar contenedores.

Por otro lado, tenemos que tener muy en cuenta el tipo de red, externa o privada, que se utiliza en la creación de los servicios. Se ha detectado que cuando se crea una máquina virtual conectada a una red tipo externa, durante el levantamiento de las VMs, no se activan correctamente las interfaces de red asociadas y por consiguiente la tabla de rutas y los servidores DNS. Esta situación provoca que no se pueda realizar las descarga e instalación

del servicio deseado, no se pueda conectar por SSH a la instancia porque no tiene dirección IP y que el arranque tarde mucho más que en otros caso, debido a que tienen que saltar los timeout.

En la figura 7.2 se muestra el log de una instancia conectada a la red "*public*" que es externa (*izquierda*) y otra conectada a la red "*osm-ext*" (*derecha*).

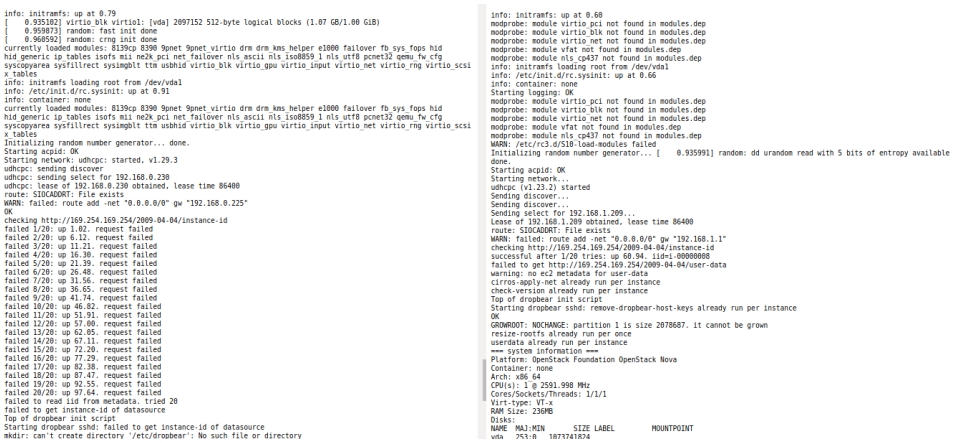

Figura 7.2: Logs de instancias conectadas a red externa y red privada.

7.2. Realización de las pruebas.

Para el desarrollo de las pruebas de instanciación se ha utilizado la imagen de Fedora Cloud Base (en caso de no tenerla disponible será necesario crearla como se indicó en el apartado 6.4.2), la red "*osm-ext*" y asignado IPs flotantes.

7.2.1. Despliegue instancias de máquina virtual con Heat.

En esta prueba de despliegue diferenciaremos dos partes, en una se le asigna a cada instancia virtual 1 Gb de RAM y en la otra, 2 Gb.

La creación de la pila se ha realizado mediante la definición de un plantilla en formato YAML para Heat (véase Anexo C). En ella se indican todos los datos relacionados con la red, tipo de instancia, parámetros de creación

de máquinas virtuales. Se definen variables relacionadas con los usuarios y contraseñas de configuración para la base de datos y, lo más importante, el script leído por "*cloud-init*" para configurar los servicios de cada instancia.

En el caso de la instancia como base de datos:

```
user_data:
    str_replace:
      template: |
            #!/bin/bash -v

            yum -y install mariadb mariadb-server
            touch /var/log/mariadb/mariadb.log
            chown mysql.mysql /var/log/mariadb/mariadb.log
            systemctl start mariadb.service

            # Setup MySQL root password and create a user
            mysqladmin -u root password db_rootpassword
            cat << EOF | mysql -u root --password=db_rootpassword
            CREATE DATABASE db_name;
            GRANT ALL PRIVILEGES ON db_name.* TO "db_user"@"%"
            IDENTIFIED BY "db_password";
            FLUSH PRIVILEGES;
            EXIT
            EOF
      params:
            db_rootpassword: { get_param: db_root_password }
            db_name: { get_param: db_name }
            db_user: { get_param: db_username }
            db_password: { get_param: db_password }
```

Por otro lado, tenemos la configuración del servidor de Wordpress, en cuya configuración se tiene que incluir la información referente a la base de datos.

```
user_data:
    str_replace:
       template: |
            #!/bin/bash -v

            yum -y install httpd wordpress

            sed -i "/Deny from All/d" /etc/httpd/conf.d/wordpress.conf
            sed -i "s/Require local/Require all granted/" /etc/httpd/
                conf.d/wordpress.conf
            sed -i s/database_name_here/db_name/ /etc/wordpress/wp-
                config.php
            sed -i s/username_here/db_user/      /etc/wordpress/wp-
                config.php
            sed -i s/password_here/db_password/  /etc/wordpress/wp-
                config.php
            sed -i s/localhost/db_ipaddr/        /etc/wordpress/wp-
                config.php
            setenforce 0 # Otherwise net traffic with DB is disabled
            systemctl start httpd.service
```

```
      params:
           db_rootpassword: { get_param: db_root_password }
           db_name: { get_param: db_name }
           db_user: { get_param: db_username }
           db_password: { get_param: db_password }
           db_ipaddr: { get_attr: [DatabaseServer, networks, osm-ext,
                0] }
```

Habiendo configurado la plantilla de Heat, para la creación de la pila tendremos que o bien realizarlo por línea de comandos o mediante la interfaz web.

En el caso gráfico, nos dirigimos a "*Project - Orchestration - Stack - Launch Stack*". Cargamos el fichero que acabamos de crear y rellenamos todos los datos que se nos solicitan, no son más que el nombre de la pila, la contraseña del usuario admin de la base de datos, el resto de variables tienen valores por defecto los configurados en la plantilla. Finalizamos el proceso pulsando en el botón "*Launch*".

Para realizar el despliegue por línea de comandos tenemos que ejecutar el siguiente comando:

```
$ openstack stack create -t Escritorio/Openstack/Heat_templates/
    WordPress_2_Instances.yaml wordpress-2instances
```

El despliegue de la pila puede controlarse clicando encima del nombre de la pila, veremos un gráfico con los elementos que la componen, cuando se encuentren en verde significa que la creación ha finalizado con éxito. En la figura 7.3 se muestra la pila de esta prueba con los elementos que la componen.

Como se ha mencionado con anterioridad se asignan IP flotantes para acceder al servicio, esto también se asigna como elemento en las pilas de Heat.

Viendo que se ha desplegado correctamente la pila (*stack*), podemos proceder a tomar los datos del tiempo de despliegue. Estos datos se encuentran en el log de las instancias creadas, para hacer el trabajo más sencillo lo realizaremos por línea de comandos.

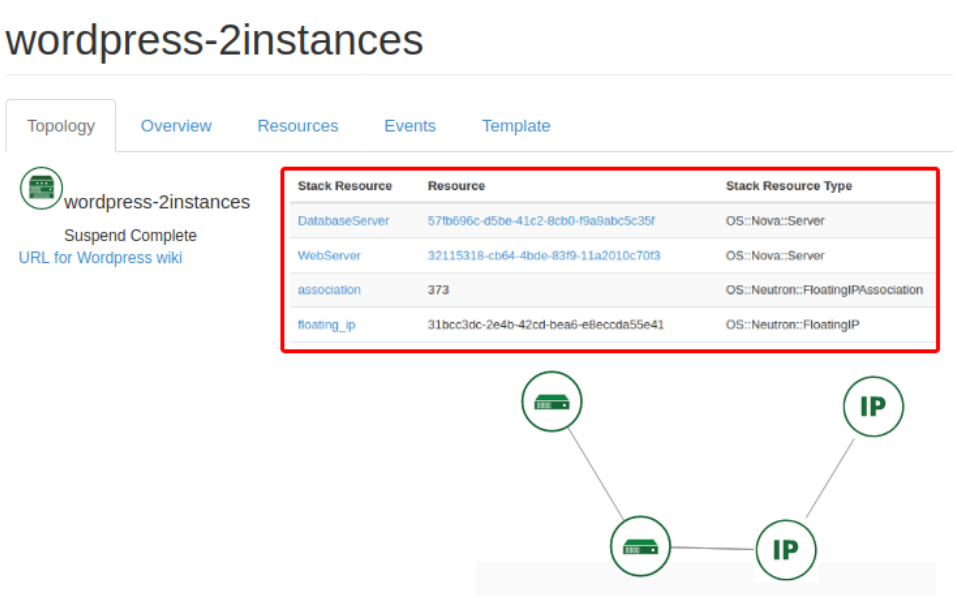

Figura 7.3: Pila de despliegue de Wordpress y MariaDB en dos instancias virtuales.

Creación del stack y registro de tiempo:

```
$ date && openstack stack create -t Escritorio/Openstack/
    Heat_templates/WordPress_2_Instances.yaml wordpress-2instances
```

Listado de instancias virtuales:

```
$ openstack server list
```

Mostramos el log de la instancia deseada:

```
$ openstack console log show <ID-server>
```

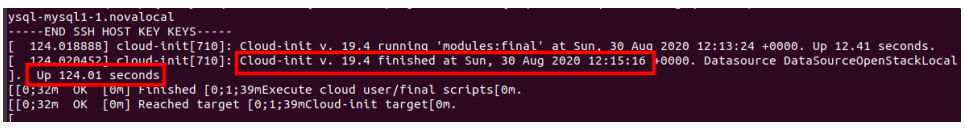

Figura 7.4: Log instancia MySQL lanzada con Heat.

Creación de instancias con Heat			
Nº	Creado	Finalizado DB-WP	Intervalo DB-WP (seg)
1 (2Gb)	16:25:32	16:27:50	111.66 – 133.90
2 (2Gb)	16:35:35	16:37:52	101.08 – 128.23
3 (2Gb)	17:07:12	17:08:40	113.32 – 76.90
4 (1Gb)	17:21:39	17:23:29	100.78 – 98.24
5 (1Gb)	18:31:18	18:34:28 – 18:33:13	184.42 – 105.21
6 (2Gb)	18:38:04	18:40:55 – 18:40:20	165.02 – 124.28
7 (2Gb)	18:47:04	18:49:48 – 18:49:13	158.37 – 118.03
8 (2Gb)	18:51:20	18:53:36 – 18:53:25	130.41 – 114.29
9 (2Gb)	18:56:28	18:58:55 – 18:59:27	139.77 – 167.68
10 (2Gb)	19:02:01	19:04:08 – 19:03:57	121.16 – 103.77

Tabla 7.1: Resultados de tiempos de despliegue en instancias con Heat.

Tras la realización reiterada del proceso podemos obtener los siguientes resultados (base de datos, DB, y wordpress, WP):

7.2.2. Despliegue instancias de contenedores con Heat.

Otra opción de despliegue trata de utilizar contenedores para crear los servicios del servidor Wordpress con la base de datos. En este caso en lugar de definir elementos "*OS::Nova::Server*" que son instancas, definirá "*OS::Zun::Container*".

En este caso de implementación se utilizarán las propias imágenes de mysql y wordpress disponibles en el repositorio de imágenes Docker HUB.

Para poder utilizar los servicios creados tenemos que cerciorarnos de que los puertos necesarios están permitidos en el grupo de seguridad que se utiliza, para ello la solución más sencilla es generar uno nuevo con las reglas que se quieran en la propia plantilla de despliegue.

Teniendo la plantilla definida (véase Anexo C), podremos lanzar la creación del stack y controlar su creación en la información de la misma. El comando para crear el stack mostrando la hora del momento en que se ejecuta:

```
$ date && openstack stack create -t Escritorio/Openstack/
    Heat_templates/zun-template.yaml wordpress-container
```

Listamos los contenedores ejecutados OpenStack:

```
$ openstack appcontainer list
```

Mostramos el log del contenedor:

```
$ openstack appcontainer log <ID-container>
```

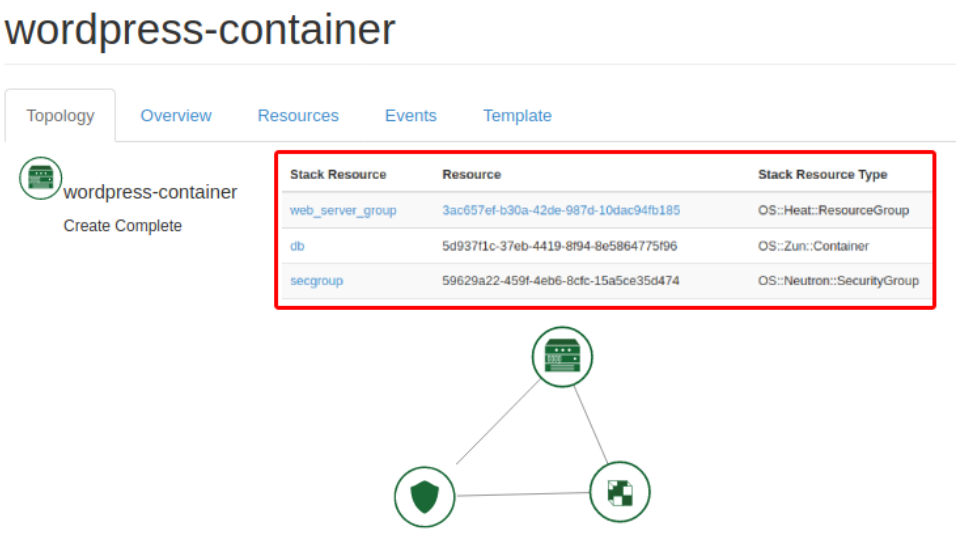

Figura 7.5: Pila de despliegue de Wordpress y MariaDB en dos contenedores.

En la figura 7.5, se aprecia el gráfico de los recursos desplegados en contenedores y en la esquina superior derecha la información sobre el tipo de recursos de que se trata. Se puede ver que el servidor web es tipo "OS::Heat::ResourceGroup", se debe a que ha sido diseñado en otra plantilla separada como recurso de Heat y la plantilla que es lanzada, zun-template.yaml, lo incorpora simplemente como un recurso de Heat. Al igual que para el caso anterior se ha realizado el despliegue de manera reiterada para obtener unos resultados más exactos, y de esta forma poder compararlos. En la tabla 7.2 se recogen los resultados del tiempo de despliegue necesario para levantar dos contenedores y configurarlos.

Creación de contenedores con Heat			
Nº	Creado	Finalizado DB-WP	Intervalo (seg)
1	15:19:26	15:19:41.85 – 15:19:43.12	17.12
2	19:17:45	19:18:04.55 – 19:18:12.93	27.93
3	19:24:40	19:24:57.02 – 19:24:59.91	19.91
4	19:29:19	19:29:34.56 – 19:29:35.22	16.22
5	19:40:12	19:40:28.44 – 19:40:30.95	18.95
6	19:46:26	19:46:42.06 – 19:46:42.29	16.29
7	19:49:35	19:49:49.79 – 19:49:51.90	16.90
8	19:51:22	19:51:40.71 – 19:51:42.00	20
9	19:52:57	19:53:13.52 – 19:53:15.06	18.05
10	19:54:43	19:54:57.68 – 19:55:00.18	17.18

Tabla 7.2: Resultados de tiempos de despliegue en contenedores con Heat.

7.2.3. Despliegue de instancias como NFV de OSM.

En último lugar, se han desplegado los servicios como instancias, pero lanzados como servicio de red desde OSM. Para ello se deberán configurar los descriptores de VNF y NS, incorporarlos a la plataforma de Open Source Mano y en último lugar, realizar la instanciación del servicio de red y monitorizar el despliegue de las máquinas virtuales en la infraestructura de OpenStack.

En primer lugar tenemos que definir los descriptores de las VNF, que serán las instancias virtuales que implementen los servicios. Se ha cogido un ejemplo simple y se le ha indicado que la configuración la coja de un fichero cloud-config. Este fichero es realmente importante, es el encargado de ejecutar los comandos de descarga, instalación y configuración de los servicios. Nuevamente los ficheros están disponibles en el anexo D.

Para el caso de la instancia de mysql tenemos como config-file:

```
#cloud-config
password: osm4u
chpasswd: { expire: False }
ssh_pwauth: True

package_update: true
packages:
- sshpass

runcmd:
- yum -y install mariadb mariadb-server
- touch /var/log/mariadb/mariadb.log
```

```
- chown mysql.mysql /var/log/mariadb/mariadb.log
- systemctl start mariadb.service

- mysqladmin -u root password admin
- cat << EOF | mysql -u root --password=admin
- CREATE DATABASE wordpress;
- GRANT ALL PRIVILEGES ON wordpress.* TO "admin"@"%"
- IDENTIFIED BY "admin";
- FLUSH PRIVILEGES;
- EXIT
- EOF
```

Para la instancia de wordpress tenemos el siguiente fichero cloud-config:

```
#cloud-config
password: osm4u
chpasswd: { expire: False }
ssh_pwauth: True

package_update: true
packages:
- sshpass

runcmd:
- yum -y install httpd wordpress
- sed -i "/Deny from All/d" /etc/httpd/conf.d/wordpress.conf
- sed -i "s/Require local/Require all granted/" /etc/httpd/conf.d/
    wordpress.conf
- sed -i s/database_name_here/wordpress/ /etc/wordpress/wp-config.php
- sed -i s/username_here/admin/      /etc/wordpress/wp-config.php
- sed -i s/password_here/admin/  /etc/wordpress/wp-config.php
- sed -i s/localhost/192.168.1.170/        /etc/wordpress/wp-config.
    php
- setenforce 0 # Otherwise net traffic with DB is disabled
- systemctl start httpd.service
```

Una vez se tienen los descriptores de VNFs, tenemos que realizar el descriptor del network service, NS. Se especificarán las conexiones de relación entre las instancias y las interfaces de red a emplear.

Como se puede observar, en la configuración de la instancia de Wordpress se le indica directamente la dirección IP de la base de datos, para saber las direcciones IP que se utilizarán se creará un nuevo fichero de configuración en formato YAML con datos sobre las IPs de los nodos y la red a utilizar, realizando un pequeño ajuste, o más bien, una especificación superior de las interfaces de red de las máquinas virtuales. De este fichero ya se habló en el apartado 6.2.3 durante el intento de instalación de Kubernetes como instancias de OSM.

Una vez tenemos definidos los descriptores tendremos que comprimirlos en formato tar.gz para poder añadirlos a OSM. La figura 7.6 muestra este proceso, que se realiza mediante los comandos:

```
$ tar -czvf wordpress_vnf.tar.gz wordpress_vnf
$ tar -czvf mysql_vnf.tar.gz mysql_vnf
$ tar -czvf wordpress_mysql_ns.tar.gz wordpress_mysql_ns
```

```
alex@alex-PC:~/Escritorio/vnf_ns$ tar -czvf wordpress_vnf.tar.gz wordpress_vnf
wordpress_vnf/
wordpress_vnf/cloud_init/
wordpress_vnf/cloud_init/cloud-init.cfg
wordpress_vnf/checksums.txt
wordpress_vnf/wordpress_vnfd.yaml
alex@alex-PC:~/Escritorio/vnf_ns$ tar -czvf wordpress_mysql_ns.tar.gz wordpress_
mysql_ns
wordpress_mysql_ns/
wordpress_mysql_ns/icons/
wordpress_mysql_ns/icons/osm.png
wordpress_mysql_ns/README.md
wordpress_mysql_ns/checksums.txt
wordpress_mysql_ns/wordpress_mysql_nsd.yaml
alex@alex-PC:~/Escritorio/vnf_ns$ tar -czvf mysql_vnf.tar.gz mysql_vnf
mysql_vnf/
mysql_vnf/cloud_init/
mysql_vnf/cloud_init/cloud-init.cfg
mysql_vnf/mysql_vnfd.yaml
mysql_vnf/checksums.txt
alex@alex-PC:~/Escritorio/vnf_ns$ date && osm ns-create --ns_name wp-mysql --nsd
_name wordpress_mysql --vim_account openstack-site --config_file config.yaml
dom ago 30 14:12:46 CEST 2020
4a6609d3-ceaf-4c6e-b0e5-21f8138507d5
```

Figura 7.6: Proceso de compresión paquetes VNF y NS.

Por el mismo procedimiento descrito en el apartado 6.4.2 se incorporan tanto los paquetes VNF como el paquete NS a catálogo de OSM.

Cuando se hayan incorporado satisfactoriamente podremos instanciar el servicio de red con el comando siguiente:

```
$ date && osm ns-create --ns_name wp-mysql --nsd_name wordpress_mysql
    --vim_account openstack-site --config_file Escritorio/vnf_ns/
    config.yaml
```

Para obtener los tiempos de despliegue de los servicios en las instancias virtuales seguiremos el mismo procedimiento que en el apartado 7.2.1.

Para proceder de la manera más rápida en la eliminación del NS para poder volver a crearlo y seguir tomando datos podremos utilizar un comando, esperar unos segundos a que se elimine y volver a lanzar el comando de creación.

Comando para eliminar el NS:

```
$ osm ns-delete wp-mysql
```

Al igual que para el caso de despliegue de instancias virtuales con Heat, se han realizado cinco despliegues del servicio con 1 Gb de RAM y otro cinco con 2 Gb.

En la tabla 7.3 se muestran los resultados obtenidos.

Creación de VNFs con OSM			
Nº	Creado	Finalizado DB-WP	Intervalo (seg)
1 (2Gb)	14:12:46	12:15:16 – 12:15:05	124.01 – 115.13
2 (2Gb)	14:46:27	12:49:43 – 12:49:38	180.61 – 174.46
3 (2Gb)	14:57:37	12:59:59 – 12:59:53	127.25 – 122.45
4 (2Gb)	15:02:15	13:05:22 – 13:04:12	172.22 – 104.37
5 (2Gb)	15:09:57	13:12:30 – 13:13:38	139.68 – 205.68
6 (1Gb)	15:16:06	13:19:41 – 13:20:11	200.98 – 228.95
7 (1Gb)	15:26:47	13:30:58 – 13:31:31	224.63 – 259.37
8 (1Gb)	15:34:08	13:36:50 – 13:39:33	146.42 – 311.49
9 (1Gb)	15:42:27	13:44:37 – 13:46:22	113.95 – 220.56
10 (1Gb)	15:48:24	13:50:51 – 13:52:46	127.73 – 243.73

Tabla 7.3: Resultados de tiempos de despliegue en instancias con OSM

7.3. Análisis de resultados.

Si analizamos detenidamente los resultados anteriores para cada una de las opciones de despliegue nos daremos cuenta muy rápido que el uso de contenedores mejora de manera más que considerable el tiempo de despliegue de los servicio, incluso consumiendo muchos menos recursos. Esto es debido a la propia naturaleza de los contenedores, sin tener que instalar un sistema operativo sino únicamente las librerías necesarias. La instanciación de máquinas virtuales ronda los 120 segundos mientras que los contenedores no llegan a 20 segundos, lo cual es una diferencia muy a tener en cuenta para aquellas aplicaciones críticas que se deban instanciar en el menor tiempo posible.

En la figura 7.7, podemos apreciar el tiempo promedio de despliegue de instancias virtuales y contenedores desde OpenStack, mediante Heat. Se representan las barras de error como la desviación estándar de los resultados.

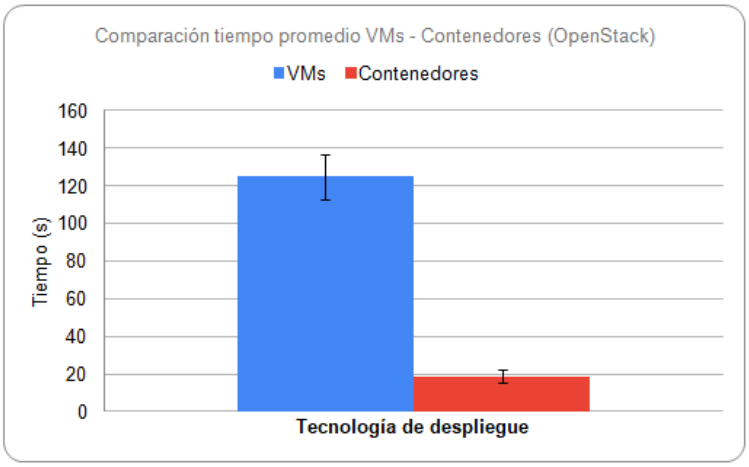

Figura 7.7: Comparación tiempo promedio VMs y contenedores.

También se observa que los tiempos de despliegue desde OSM son superiores, y desde un punto de vista lógico, tiene sentido al estar siendo solicitados los recursos desde otro PC, haciendo que tengan que realizarse más procesos.

Todos los tiempos de despliegue están fuertemente relacionados con la cantidad de procesos ejecutándose en el host de OpenStack, haciendo que si este PC está muy saturado los despliegues sean más lentos, volviendo al problema de recursos presente en el proyecto.

En la figura 7.8, observamos la gráfica de comparación entre los tiempos de despliegue de instancias desde OpenStack y OSM.

Por último, en los casos de máquinas virtuales se ha diferenciado en función de la cantidad de memoria RAM asignada a cada instancia, esto también supone variación en los tiempos de despliegue, tanto en la instalación del sistema operativo como en el despliegue final de los servicios. En la figura 7.9, se muestran los tiempos relacionados con la instanciación desde OSM y en la figura 7.10, desde OpenStack.

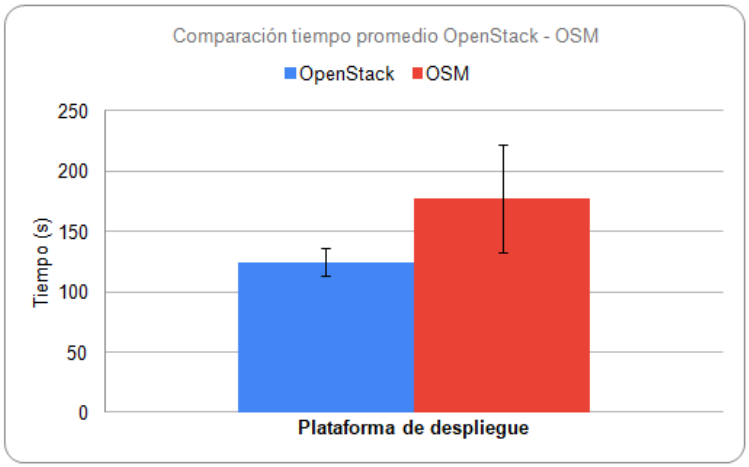

Figura 7.8: Comparación de tiempo promedio instanciando desde OpenStack y OSM.

En el caso de OSM, se aprecia una gran variación en el tiempo de despliegue del servidor Wordpress, que necesita configurarse en función de algunos parámetros de la base de datos, y por lo tanto depende directamente de que esta esté operativa cuanto antes.

Por otro lado, tenemos los datos de OpenStack. En este caso observamos poca variación general entre instancias con 1 Gb o 2 Gb de memoria, si bien es cierto que se aprecia un tiempo mayor para el despliegue de la base de datos en las máquinas virtuales de 2 Gb, esto puede ser debido a estado del sistema en alguna de las pruebas haciendo que la media ascienda considerablemente.

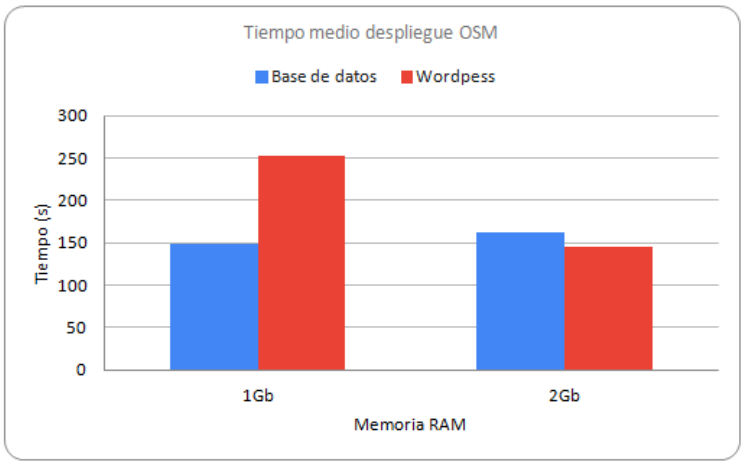

Figura 7.9: Tiempo medio de despliegue de servicios desde OSM según memoria RAM.

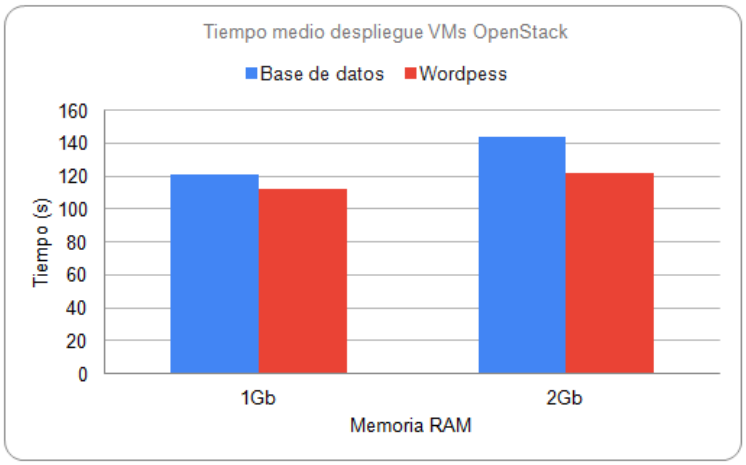

Figura 7.10: Tiempo medio de despliegue de servicios desde OpenStack según memoria RAM.

Capítulo 8

Conclusiones y Línea Futura.

En este capítulo se agruparán las conclusiones obtenidas tras la realización del proyecto completo, indicando los objetivos que se han completado o cómo se podrían completar con alguna modificación o mejora que no haya sido posible realizar. Además, se incorporarán posibles líneas de investigación abiertas o que se considera interesante de ser desarrolladas para mejorar el funcionamiento y ampliar el ámbito de aplicación de este tipo de soluciones o herramientas.

Para concluir, se expondrá una valoración personal de las temáticas abarcadas en el presente Trabajo Fin de Máster y de cómo se ve el mercado tecnológico de estas soluciones.

8.1. Conclusiones.

El presente proyecto buscaba desarrollar un escenario basado en la arquitectura estandarizada por la ETSI para NFV MANO, es por ello que se han empleado tecnologías altamente relacionadas con este estándar.

OpenStack y OSM son unas de la herramientas más apoyadas en la comunidad tecnológica para desarrollar nuevas funcionalidades, aunándolas en un mismo entorno, haciéndolo cada vez cada vez más genérico, en el sentido de poder funcionar con multitud de herramienta o tecnologías distintas, para un mismo fin. Con esto se consigue, por ejemplo, que en el mismo entorno se tenga un servicio de contenedores alojados con Docker y gestionados por Docker Swarm, y en paralelo otro gestionado por Kubernetes o Mesos.

Del mismo modo, OSM cada vez incorpora más funciones de instanciación diferentes a VNF - basadas en VMs -, como CNF - funciones de red desplegadas en contenedores, PNF - funciones de red en máquinas físicas, o KNF - funciones de red virtuales basadas en Kubernetes.

Con el desarrollo del proyecto apoyado en estas herramientas se ha podido conseguir el objetivo principal de implementar servicios de manera automatizada tanto en máquinas virtuales como en contenedores, cuyo fin era comparar los tiempos de despliegue. En el Trabajo Fin de Máster, se ha desarrollado varios escenarios: OpenStack y OSM funcionando conjuntamente, es decir, levantando las VNFs de Open Source Mano en la intraestructura virtual de OpenStack, orquestación de máquinas virtuales con Heat en OpenStack, orquestación de contenedores con Heat en OpenStack o empleo de *capsulas*, grupos de contenedores.

Esta comparativa estaba orientada a poder desplegar servicios en la nube teniendo en cuenta posibles aplicaciones críticas y situaciones de bajos recursos, como puede ser elementos de Edge Computing.

Es de mencionar, las principales problemáticas con las que se ha convivido durante la realización de este proyecto son el trabajo en un entorno de recursos muy limitados para la naturaleza de los escenarios a desarrollar y que la documentación de las herramientas utilizadas, en ciertos aspectos y funcionalidades, estaban incompletas.

En la actualidad, hay un gran apoyo hacia el desarrollo, investigación e innovación desde sectores que potencialmente podrían beneficiarse de nuevas funciones y/o mejoras de las actuales tanto en la aplicaciones de nuevas medidas de seguridad como en la modificación del paradigma de despliegue de los servicios más críticos, donde se busca proporcionar una mayor escalabilidad, disponibilidad y disminuyendo el CAPEX y el OPEX de las infraestructuras empleadas.

8.2. Línea futura.

Se han conseguido la mayoría de objetivos propuestos desde el planteamiento del Trabajo Fin de Máster, aunque sí es cierto que algunos de ellos no se han podido conseguir debido a errores en las propias herramientas utilizadas y que no han podido ser paliados.

La principal línea de investigación y desarrollo sería mejorar ciertas funcionalidades de OSM y OpenStack, desde arreglar los problemas en ficheros de configuración para la instalación hasta depurar los errores que han ido surgiendo con el empleo de ciertos componentes o funciones de OSM y OpenStack. También es cierto que hay lagunas en la documentación de estas herramientas y faltan por desarrollar, haciendo que el uso o implementación de esas funciones resulte más complicado al no haber ninguna referencia.

Otra línea de investigación a considerar es el despliegue de KNFs, dado que si se cumple el mismo comportamiento que el observado en el capítulo anterior, comparando el despliegue en máquinas virtuales y en contenedores, pero asociándolo a un entorno MANO, permitiría mejorar el despliegue, control, administración y automatización de ciertas funciones de red. Además de permitirse un mejor uso de los recursos disponibles.

En este sentido, no hay que perder de vista el módulo de orquestación de OpenStack, Heat. Este componente posee unas características muy interesantes para el despliegue y control automatizado de elementos en la infraestructura virtual creada por OpenStack. Se ha visto en capítulos anteriores que se ha utilizado para desplegar máquinas virtuales proporcionadas por Nova y contenedores por Zun, pero su funcionalidad va más allá, es capaz de crear redes completas formadas por la propia red, router, balanceador de carga, firewall, etc. Todo dependerá de las funcionalidades disponibles y proporcionadas por los componentes de OpenStack.

Una de las características más interesantes es la capacidad de monitorizar las pilas de recursos desplegadas con Heat y actuar acorde a ciertos eventos de manera automática. Por ejemplo, se puede desplegar un balanceador de carga con dos servidores y en el caso de detectar que estos servidores se están saturando poder lanzar otro idéntico y añadirlo al balanceo. Del mismo modo, en el momento que se liberan de carga los servidores, el último creado como apoyo es eliminado liberando los recursos asociados.

8.3. Valoración personal.

Para concluir la elaboración de la presente memoria se recoge una evaluación personal tanto del proyecto como de todo lo acontecido hasta este momento.

La finalización de este Trabajo Fin de Máster es una meta marcada hace algún tiempo y no por ello se ha abandonado la idea de solventar todos los problemas que se han ido presentando, es más, se ha crecido en el aspecto personal y profesional, sobre todo porque se ha ido compatibilizando el tiempo de estudio con el tiempo de trabajo.

Ambos han estado muy marcados por el desarrollo o el trabajo con proyectos relativamente innovadores, y aunque muy distanciados en cuanto al ámbito de aplicación, ambos presentan un potencial muy elevado. Este hecho me hace sentir agradecido y a la vez recompensado al mirar hacia atrás y ver cómo he ido consiguiendo cierta metas y por qué no, haber puesto mi granito de arena en estos proyectos.

Desde hace tiempo he visto la necesidad de ciertos cambios en algunos aspectos de la tecnología y poder ayudar a que se mejore transmite una gran satisfacción.

He de decir, este proyecto me ha abierto los ojos hacia unas tecnologías de la que había escuchado pero no conocía en profundidad y me han sorprendido para bien, sobretodo al conocer la gran comunidad tanto de empresas tecnológicas punteras como de clientes que apoyan su uso y solicitan ciertas mejoras que pueden ser interesantes.

Bibliografía

[1] Oracle. Brief history of virtualization. URL: https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html.

[2] Microsoft Azure. ¿qué es virtualización? URL: https://azure.microsoft.com/es-es/overview/what-is-virtualization/.

[3] Red Hat. ¿qué es la virtualización? URL: https://www.redhat.com/es/topics/virtualization/what-is-virtualization.

[4] IBM Cloud Education. Virtualization. URL: https://www.ibm.com/cloud/learn/virtualization-a-complete-guide.

[5] Citrix. ¿qué es la virtualización? - definición de virtualización. URL: https://www.citrix.com/es-es/glossary/what-is-virtualization.html.

[6] Centro de recursos IT user. La virtualización está reduciendo la demanda de capacidad en los centros de datos. URL: https://almacenamientoit.ituser.es/noticias-y-actualidad/2019/06/la-virtualizacion-esta-reduciendo-la-demanda-de-capacidad-en-los-centros-de-datos.

[7] Neal Weinberg. ¿cuál es el futuro de la virtualización de servidores? URL: https://www.networkworld.es/telecomunicaciones/cual-es-el-futuro-de-la-virtualizacion-de-servidores.

[8] Jonas DeMuro. What is container technology? URL: https://www.techradar.com/news/what-is-container-technology.

[9] Franz Kasparec. Network function cloudification. URL: https://atos.net/en/blog/network-function-cloudification.

[10] NE Digital. ¿que es la virtualización de servidores, containerización y virtualización de redes? URL: https://www.nedigital.com/es/transformacion-digital/virtualizacion/.

[11] Arne Holst. Global traffic of sdn/nfv within data centers 2015-2021. URL: https://www.statista.com/statistics/637966/worldwide-sdn-nfv-data-center-traffic/.

[12] Michael Cooney. What is sdn and where software-defined networking is going. URL: https://www.networkworld.com/article/3209131/what-sdn-is-and-where-its-going.html.

[13] David Linthicum. Sdn and cloud computing: Working in unison. URL: https://www.cisco.com/c/en/us/solutions/enterprise-networks/sdn-cloud-computing.html.

[14] Salesforce. Cloud computing - aplicaciones en un solo tacto. URL: https://www.salesforce.com/mx/cloud-computing/.

[15] ADN Cloud. 4 razones por las que cloud computing es el futuro. URL: https://www.entrepreneur.com/article/314709.

[16] Antonio Gomariz. 2019, el año clave para el futuro del cloud computing. URL: https://blogthinkbig.com/2019-clave-futuro-cloud-computing.

[17] Diego de la Torre. Edge computing, desplegar servicios, contenido e inteligencia en milisegundos. URL: https://blogthinkbig.com/explicacion-ventajas-edge-computing.

[18] ThisIsTheRealSpain. España, líder en despliegue del 5g en europa. URL: https://www.thisistherealspain.com/actualidad/espana-lider-en-despliegue-del-5g-en-europa/.

[19] Gioacchino Lonardo. Docker (container vs virtual machine). URL: https://medium.com/@lonardogio/docker-container-part-1-49ec6dbe1f34.

[20] VMware. What is container networking? URL: https://www.vmware.com/topics/glossary/content/container-networking.

[21] Alex Walton. Cuál es la diferencia entre sdn y nfv. URL: https://ccnadesdecero.es/diferencia-sdn-y-nfv/.

[22] ETSI. Network functions virtualisation (nfv). URL: https://www.etsi.org/technologies/nfv.

[23] VMware. Network functions virtualization overview. URL: https://docs.vmware.com/en/VMware-vCloud-NFV/2.0/vmware-vcloud-nfv-openstack-edition-ra20/GUID-FBEA6C6B-54D8-4A37-87B1-D825F9E0DBC7.html.

[24] SDxCentral Staff. What is the virtualized infrastructure manager (vim)? definition. URL: https://www.sdxcentral.com/networking/nfv/definitions/virtualized-infrastructure-manager-vim-definition/.

[25] VMware. Vmware vsphere. URL: https://www.vmware.com/files/es/pdf/VMware-vSphere-Entreprise-Edition-Datasheet.pdf.

[26] VMware. Vmware cloud foundation. URL: https://www.vmware.com/es/products/cloud-foundation.html.

[27] OpenSTack. Open source cloud computing infrastructure - openstack. URL: https://www.openstack.org/.

[28] Red Hat. El concepto de openstack. URL: https://www.redhat.com/es/topics/openstack.

[29] Amazon. ¿qué es aws? URL: https://aws.amazon.com/es/what-is-aws/.

[30] Cisco. Cisco virtualized infrastructure manager (vim). URL: https://www.cisco.com/c/en/us/products/cloud-systems-management/virtualized-infrastructure-manager/index.html.

[31] Red Hat. Plataforma de aplicaciones en contenedores - red hat openshift. URL: https://www.redhat.com/es/technologies/cloud-computing/openshift.

[32] Apache. Apache cloudstack: Open source cloud computing. URL: https://cloudstack.apache.org/.

[33] Gerardo García. Openvim. URL: https://github.com/nfvlabs/openvim.

[34] Fundación Parque Científico de Madrid. Opennebula. URL: https://fpcm.es/directorio-de-empresas/opennebula/.

[35] OpenNebula. Opennebula - the open source cloud management platform developed for the enterprise. URL: https://opennebula.io/.

[36] SDxCentral Staff. What is nfv mano? URL: https://www.sdxcentral.com/networking/nfv/mano-lso/definitions/nfv-mano/.

[37] SDxCentral Staff. What is a vnf manager (vnfm)? definition. URL: https://www.sdxcentral.com/networking/nfv/definitions/vnf-manager-vnfm-definition/.

[38] SDxCentral Staff. What is an nfv orchestrator (nfvo)? definition. URL: https://www.sdxcentral.com/networking/nfv/definitions/nfv-orchestrator-nfvo-definition/.

[39] R. Mijumbi, J. Serrat, J. Gorricho, S. Latre, M. Charalambides, and D. Lopez. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, 54(1):98–105, 2016.

[40] Telefónica Investigación y Desarrollo. Openmano. URL: http://www.tid.es/es/innovacion-de-largo-plazo/network-innovation/telefonica-nfv-reference-lab/openmano.

[41] ETSI. Etsi - open source mano (osm). URL: https://www.etsi.org/technologies/open-source-mano.

[42] Open Source Mano. What is osm? URL: https://osm.etsi.org/.

[43] ONAP. Open network automation platform. URL: https://docs.onap.org/en/frankfurt/.

[44] Team Cloudify. Onap vs osm – the battle for nfv management and orchestration supremacy. URL: https://cloudify.co/blog/onap-vs-osm/.

[45] OPEN BATON. Open baton: an open source reference implementation of the etsi network function virtualization mano specification. URL: https://openbaton.org/.

[46] OPNFV Project. Opnfv. URL: https://www.opnfv.org/.

[47] ETSI OSM. Osm9 hackfesthack 0: Introduction to nfv and osm. ETSI. URL: http://osm-download.etsi.org/ftp/osm-7.0-seven/OSM9-hackfest/presentations/OSM#9Hackfest-HD0.0IntroductiontoNFVandOSM.pptx.pdf.

[48] ETSI OSM Community. Osm release three. Technical report, ETSI OSM. URL: https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTHREE-FINAL.pdf.

[49] OpenStack. Openstack docs: Ussuri. URL: https://docs.openstack.org/ussuri/.

Apéndice A

Fichero configuración OpenStack.

En este anexo contiene los ficheros de configuración para realizar la instalación de OpenStack, tanto el utilizado en la instalación completa que consumía demasiados recursos para hardware disponible como el utilizado en la instalación final.

El primero muestra el fichero "*local.conf*" de la instalación completa (funcional) de OpenStack.

```
# Sample ‘‘local.conf‘‘ for user-configurable variables in ‘‘stack.sh
    ‘‘

# NOTE: Copy this file to the root DevStack directory for it to work
    properly.

# ‘‘local.conf‘‘ is a user-maintained settings file that is sourced
    from ‘‘stackrc‘‘.
# This gives it the ability to override any variables set in ‘‘stackrc
    ‘‘.
# Also, most of the settings in ‘‘stack.sh‘‘ are written to only be
    set if no
# value has already been set; this lets ‘‘local.conf‘‘ effectively
    override the
# default values.

# This is a collection of some of the settings we have found to be
    useful
# in our DevStack development environments. Additional settings are
    described
# in http://docs.openstack.org/developer/devstack/configuration.html#
    local-conf
# These should be considered as samples and are unsupported DevStack
    code.

# The ‘‘localrc‘‘ section replaces the old ‘‘localrc‘‘ configuration
    file.
```

```
# Note that if ‘‘localrc‘‘ is present it will be used in favor of this
     section.
[[local|localrc]]

ADMIN_PASSWORD=openstack
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD
# Enable Keystone v3
IDENTITY_API_VERSION=3

PUBLIC_INTERFACE=wlo1
HOST_IP=192.168.0.200
FLOATING_RANGE=192.168.0.224/27

# Services Swift
enable_service s-proxy s-object s-container s-account

enable_plugin heat https://opendev.org/openstack/heat stable/ussuri
enable_plugin heat-dashboard https://opendev.org/openstack/heat-
    dashboard stable/ussuri

CEILOMETER_BACKEND=gnocchi
enable_plugin panko https://opendev.org/openstack/panko stable/ussuri
enable_plugin ceilometer https://opendev.org/openstack/ceilometer
    stable/ussuri
enable_plugin aodh https://opendev.org/openstack/aodh stable/ussuri

enable_plugin manila https://opendev.org/openstack/manila stable/
    ussuri
enable_plugin manila-ui https://opendev.org/openstack/manila-ui stable
    /ussuri

enable_plugin freezer https://opendev.org/openstack/freezer stable/
    ussuri
enable_plugin freezer-api https://opendev.org/openstack/freezer-api.
    git stable/ussuri
enable_plugin freezer-tempest-plugin https://opendev.org/openstack/
    freezer-tempest-plugin.git
enable_plugin freezer-web-ui https://opendev.org/openstack/freezer-web
    -ui.git stable/ussuri

enable_plugin trove https://opendev.org/openstack/trove stable/ussuri
enable_plugin trove-dashboard https://opendev.org/openstack/trove-
    dashboard stable/ussuri

enable_plugin masakari https://opendev.org/openstack/masakari stable/
    ussuri
enable_plugin masakaridashboard https://opendev.org/openstack/masakari
    -dashboard stable/ussuri

### Barbican
enable_plugin barbican https://opendev.org/openstack/barbican stable/
    ussuri

### Enable Magnum
enable_plugin magnum https://opendev.org/openstack/magnum stable/
    ussuri
enable_plugin magnum-ui https://opendev.org/openstack/magnum-ui stable
    /ussuri
```

```
##############
# Enable services for Kubernetes (Zun, kuryr, magnum)
##############
enable_plugin zun https://opendev.org/openstack/zun stable/ussuri
enable_plugin zun-tempest-plugin https://opendev.org/openstack/zun-
    tempest-plugin
# Optional:  uncomment to enable the Zun UI plugin in Horizon
enable_plugin zun-ui https://opendev.org/openstack/zun-ui stable/
    ussuri

# This below plugin enables installation of container engine on
    Devstack.
# The default container engine is Docker
enable_plugin devstack-plugin-container https://opendev.org/openstack/
    devstack-plugin-container stable/ussuri
# This enables CRI plugin for containerd
ENABLE_CONTAINERD_CRI=True

enable_plugin kuryr-libnetwork https://opendev.org/openstack/kuryr-
    libnetwork stable/ussuri

# install python-zunclient from git
LIBS_FROM_GIT="python-zunclient"

# Minimal Contents
# ----------------
# While ``stack.sh`` is happy to run without ``localrc``, devlife is
    better when
# there are a few minimal variables set:

# If the ``*_PASSWORD`` variables are not set here you will be
    prompted to enter
# values for them by ``stack.sh``and they will be added to ``local.
    conf``.
#ADMIN_PASSWORD=openstack
#DATABASE_PASSWORD=$ADMIN_PASSWORD
#RABBIT_PASSWORD=$ADMIN_PASSWORD
#SERVICE_PASSWORD=$ADMIN_PASSWORD
#SERVICE_TOKEN=$ADMIN_PASSWORD
# Enable Keystone v3
#IDENTITY_API_VERSION=3

# ``HOST_IP`` and ``HOST_IPV6`` should be set manually for best
    results if
# the NIC configuration of the host is unusual, i.e. ``eth1`` has the
    default
# route but ``eth0`` is the public interface.  They are auto-detected
    in
# ``stack.sh`` but often is indeterminate on later runs due to the IP
    moving
# from an Ethernet interface to a bridge on the host. Setting it here
    also
# makes it available for ``openrc`` to include when setting ``
    OS_AUTH_URL``.
# Neither is set by default.
#HOST_IP=w.x.y.z
#HOST_IPV6=2001:db8::7
#PUBLIC_INTERFACE=wlo1
#HOST_IP=192.168.0.200
#PUBLIC_NETWORK_GATEWAY=192.168.0.1
#FLOATING_RANGE=192.168.0.192/26
#FIXED_RANGE=10.11.12.0/24
```

```
# Logging
# -------

# By default ``stack.sh`` output only goes to the terminal where it
    runs.  It can
# be configured to additionally log to a file by setting ``LOGFILE``
    to the full
# path of the destination log file.  A timestamp will be appended to
    the given name.
LOGFILE=$DEST/logs/stack.sh.log

# Old log files are automatically removed after 7 days to keep things
    neat.  Change
# the number of days by setting ``LOGDAYS``.
LOGDAYS=2

# Nova logs will be colorized if ``SYSLOG`` is not set; turn this off
    by setting
# ``LOG_COLOR`` false.
#LOG_COLOR=False

# Using milestone-proposed branches
# ---------------------------------
# Uncomment these to grab the milestone-proposed branches from the
# repos:
#CINDER_BRANCH=milestone-proposed
#GLANCE_BRANCH=milestone-proposed
#HORIZON_BRANCH=milestone-proposed
#KEYSTONE_BRANCH=milestone-proposed
#KEYSTONECLIENT_BRANCH=milestone-proposed
#NOVA_BRANCH=milestone-proposed
#NOVACLIENT_BRANCH=milestone-proposed
#NEUTRON_BRANCH=milestone-proposed
#SWIFT_BRANCH=milestone-proposed

# Using git versions of clients
# -----------------------------
# By default clients are installed from pip.  See LIBS_FROM_GIT in
# stackrc for details on getting clients from specific branches or
# revisions.  e.g.
# LIBS_FROM_GIT="python-ironicclient"
# IRONICCLIENT_BRANCH=refs/changes/44/2.../1

# Swift
# -----
# Swift is now used as the back-end for the S3-like object store.
    Setting the
# hash value is required and you will be prompted for it if Swift is
    enabled
# so just set it to something already:
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5

# For development purposes the default of 3 replicas is usually not
    required.
# Set this to 1 to save some resources:
SWIFT_REPLICAS=1

# The data for Swift is stored by default in (``$DEST/data/swift``),
# or (``$DATA_DIR/swift``) if ``DATA_DIR`` has been set, and can be
# moved by setting ``SWIFT_DATA_DIR``. The directory will be created
# if it does not exist.
```

```
SWIFT_DATA_DIR=$DEST/data
```

A continuación, se muestra el fichero "*local.conf*"definitivo de la instalación utilizada de OpenStack

```
# Sample ‘‘local.conf‘‘ for user-configurable variables in ‘‘stack.sh
    ‘‘

# NOTE: Copy this file to the root DevStack directory for it to work
    properly.

# ‘‘local.conf‘‘ is a user-maintained settings file that is sourced
    from ‘‘stackrc‘‘.
# This gives it the ability to override any variables set in ‘‘stackrc
    ‘‘.
# Also, most of the settings in ‘‘stack.sh‘‘ are written to only be
    set if no
# value has already been set; this lets ‘‘local.conf‘‘ effectively
    override the
# default values.

# This is a collection of some of the settings we have found to be
    useful
# in our DevStack development environments. Additional settings are
    described
# in http://docs.openstack.org/developer/devstack/configuration.html#
    local-conf
# These should be considered as samples and are unsupported DevStack
    code.

# The ‘‘localrc‘‘ section replaces the old ‘‘localrc‘‘ configuration
    file.
# Note that if ‘‘localrc‘‘ is present it will be used in favor of this
     section.
[[local|localrc]]
ADMIN_PASSWORD=openstack
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD
# Enable Keystone v3
IDENTITY_API_VERSION=3

PUBLIC_INTERFACE=wlo1
HOST_IP=192.168.0.200
#PUBLIC_NETWORK_GATEWAY=192.168.0.1
#FLOATING_RANGE=192.168.0.192/26
FLOATING_RANGE=192.168.0.224/27
IP_VERSION=4

# Services Swift
enable_service s-proxy s-object s-container s-account

enable_plugin heat https://opendev.org/openstack/heat stable/ussuri
enable_plugin heat-dashboard https://opendev.org/openstack/heat-
    dashboard stable/ussuri

# enable_plugin tacker https://opendev.org/openstack/tacker
```

```
CEILOMETER_BACKEND=gnocchi
enable_plugin panko https://opendev.org/openstack/panko stable/ussuri
enable_plugin ceilometer https://opendev.org/openstack/ceilometer
    stable/ussuri
enable_plugin aodh https://opendev.org/openstack/aodh stable/ussuri

### Barbican
enable_plugin barbican https://opendev.org/openstack/barbican stable/
    ussuri

### Enable Magnum
enable_plugin magnum https://opendev.org/openstack/magnum
enable_plugin magnum-ui https://opendev.org/openstack/magnum-ui

##############
# Enable services for Kubernetes (Zun, kuryr, magnum)
##############

enable_plugin zun https://opendev.org/openstack/zun stable/ussuri
enable_plugin zun-tempest-plugin https://opendev.org/openstack/zun-
    tempest-plugin
# Optional:  uncomment to enable the Zun UI plugin in Horizon
enable_plugin zun-ui https://opendev.org/openstack/zun-ui stable/
    ussuri

# This below plugin enables installation of container engine on
    Devstack.
# The default container engine is Docker
enable_plugin devstack-plugin-container https://opendev.org/openstack/
    devstack-plugin-container stable/ussuri
# This enables CRI plugin for containerd
ENABLE_CONTAINERD_CRI=True

# Optional:  uncomment to enable Kata Container
# ENABLE_KATA_CONTAINERS=True

# In Kuryr, KURYR_CAPABILITY_SCOPE is `local` by default,
# but we must change it to `global` in the multinode scenario.
KURYR_CAPABILITY_SCOPE=global
KURYR_PROCESS_EXTERNAL_CONNECTIVITY=False
enable_plugin kuryr-libnetwork https://opendev.org/openstack/kuryr-
    libnetwork stable/ussuri

# install python-zunclient from git
LIBS_FROM_GIT="python-zunclient"


# Minimal Contents
# ----------------
# While ``stack.sh`` is happy to run without ``localrc``, devlife is
    better when
# there are a few minimal variables set:

# If the ``*_PASSWORD`` variables are not set here you will be
    prompted to enter
# values for them by ``stack.sh``and they will be added to ``local.
    conf``.
#ADMIN_PASSWORD=openstack
#DATABASE_PASSWORD=$ADMIN_PASSWORD
#RABBIT_PASSWORD=$ADMIN_PASSWORD
#SERVICE_PASSWORD=$ADMIN_PASSWORD
#SERVICE_TOKEN=$ADMIN_PASSWORD
```

```
# Enable Keystone v3
#IDENTITY_API_VERSION=3

# ``HOST_IP`` and ``HOST_IPV6`` should be set manually for best
    results if
# the NIC configuration of the host is unusual, i.e. ``eth1`` has the
    default
# route but ``eth0`` is the public interface.  They are auto-detected
    in
# ``stack.sh`` but often is indeterminate on later runs due to the IP
    moving
# from an Ethernet interface to a bridge on the host. Setting it here
    also
# makes it available for ``openrc`` to include when setting ``
    OS_AUTH_URL``.
# Neither is set by default.
#PUBLIC_INTERFACE=wlo1
#HOST_IP=192.168.0.200
#PUBLIC_NETWORK_GATEWAY=192.168.0.1
#FLOATING_RANGE=192.168.0.192/26

# Logging
# -------
# By default ``stack.sh`` output only goes to the terminal where it
    runs.  It can
# be configured to additionally log to a file by setting ``LOGFILE``
    to the full
# path of the destination log file.  A timestamp will be appended to
    the given name.
LOGFILE=$DEST/logs/stack.sh.log

# Old log files are automatically removed after 7 days to keep things
    neat.  Change
# the number of days by setting ``LOGDAYS``.
LOGDAYS=2

# Nova logs will be colorized if ``SYSLOG`` is not set; turn this off
    by setting
# ``LOG_COLOR`` false.
#LOG_COLOR=False

# Using milestone-proposed branches
# ---------------------------------
# Uncomment these to grab the milestone-proposed branches from the
# repos:
#CINDER_BRANCH=milestone-proposed
#GLANCE_BRANCH=milestone-proposed
#HORIZON_BRANCH=milestone-proposed
#KEYSTONE_BRANCH=milestone-proposed
#KEYSTONECLIENT_BRANCH=milestone-proposed
#NOVA_BRANCH=milestone-proposed
#NOVACLIENT_BRANCH=milestone-proposed
#NEUTRON_BRANCH=milestone-proposed
#SWIFT_BRANCH=milestone-proposed

# Using git versions of clients
# -----------------------------
# Swift
# -----
# Swift is now used as the back-end for the S3-like object store.
    Setting the
# hash value is required and you will be prompted for it if Swift is
```

```
    enabled
# so just set it to something already:
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5

# For development purposes the default of 3 replicas is usually not
    required.
# Set this to 1 to save some resources:
SWIFT_REPLICAS=1
# The data for Swift is stored by default in (``$DEST/data/swift``),
# or (``$DATA_DIR/swift``) if ``DATA_DIR`` has been set, and can be
# moved by setting ``SWIFT_DATA_DIR``. The directory will be created
# if it does not exist.
SWIFT_DATA_DIR=$DEST/data
```

Apéndice B

Plantillas utilizadas para Capsule - Zun

Este anexo contiene la plantilla utilizada para la prueba de despliegue de un servidor web en contenedores con Capsules, haciendo preconfiguración de la web a mostrar.

```
capsuleVersion: beta
kind: capsule
metadata:
  name: init-demo
spec:
  containers:
  - image: nginx
    ports:
      - name: nginx-port
        containerPort: 80
        hostPort: 80
        protocol: TCP
    volumeMounts:
    - name: workdir
      mountPath: /usr/share/nginx/html
  # These containers are run during capsule initialization
  initContainers:
  - image: busybox
    command:
    - wget
    - "-O"
    - "/work-dir/index.html"
    - https://www.openstack.org/
    volumeMounts:
    - name: workdir
      mountPath: "/work-dir"
  volumes:
  - name: workdir
    cinder:
      size: 1
      autoRemove: True
```

Apéndice C

Plantillas de Heat utilizadas.

Este anexo contiene las plantillas utilizadas para crear servicios en instancias virtuales y contenedores de forma automatizada mediante Heat.

En primer lugar, se muestra la plantilla utilizada para crear el servicio con máquinas virtuales.

Fichero "*WordPress_2_Instances.yaml*"

```
heat_template_version: 2013-05-23

description: >
  An example Heat Orchestration Template (HOT).
  WordPress is web software you can use to create a beautiful website
  or blog. This template installs two instances: one running a
  WordPress deployment and the other using a local MySQL database to
  store the data.

parameters:

  key_name:
    type: string
    description: Name of a KeyPair to enable SSH access to the
        instance
    default: mykey

  instance_type:
    type: string
    description: Instance type for web and DB servers
    default: ds2G
    constraints:
      - allowed_values: [m1.tiny, m1.small, m1.medium, m1.large, m1.
          xlarge, ds1G, ds2G, ds4G]
        description: instance_type must be a valid instance type

  image_id:
    type: string
    description: >
```

```
    Name or ID of the image to use for the WordPress server.
    Recommended values are fedora-20.i386 or fedora-20.x86_64;
    get them from http://cloud.fedoraproject.org/fedora-20.i386.
        qcow2
    or http://cloud.fedoraproject.org/fedora-20.x86_64.qcow2 .
  default: fedora-cloud-base

db_name:
  type: string
  description: WordPress database name
  default: wordpress
  constraints:
    - length: { min: 1, max: 64 }
      description: db_name must be between 1 and 64 characters
    - allowed_pattern: '[a-zA-Z][a-zA-Z0-9]*'
      description: >
        db_name must begin with a letter and contain only
            alphanumeric
        characters

db_username:
  type: string
  description: The WordPress database admin account username
  default: admin
  hidden: true
  constraints:
    - length: { min: 1, max: 16 }
      description: db_username must be between 1 and 16 characters
    - allowed_pattern: '[a-zA-Z][a-zA-Z0-9]*'
      description: >
        db_username must begin with a letter and contain only
            alphanumeric
        characters

db_password:
  type: string
  description: The WordPress database admin account password
  default: admin
  hidden: true
  constraints:
    - length: { min: 1, max: 41 }
      description: db_password must be between 1 and 41 characters
    - allowed_pattern: '[a-zA-Z0-9]*'
      description: db_password must contain only alphanumeric
          characters

db_root_password:
  type: string
  description: Root password for MySQL
  default: admin
  hidden: true
  constraints:
    - length: { min: 1, max: 41 }
      description: db_root_password must be between 1 and 41
          characters
    - allowed_pattern: '[a-zA-Z0-9]*'
      description: db_root_password must contain only alphanumeric
          characters

net:
  type: string
  label: Selected Network
```

```
    description: Network of the instance
    default: osm-ext

  external_network:
    type: string
    default: public

resources:
  DatabaseServer:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_id }
      flavor: { get_param: instance_type }
      key_name: { get_param: key_name }
      networks:
        - network: {get_param: net}
      security_groups:
        - default
      user_data:
        str_replace:
          template: |
            #!/bin/bash -v

            yum -y install mariadb mariadb-server
            touch /var/log/mariadb/mariadb.log
            chown mysql.mysql /var/log/mariadb/mariadb.log
            systemctl start mariadb.service

            # Setup MySQL root password and create a user
            mysqladmin -u root password db_rootpassword
            cat << EOF | mysql -u root --password=db_rootpassword
            CREATE DATABASE db_name;
            GRANT ALL PRIVILEGES ON db_name.* TO "db_user"@"%"
            IDENTIFIED BY "db_password";
            FLUSH PRIVILEGES;
            EXIT
            EOF

          params:
            db_rootpassword: { get_param: db_root_password }
            db_name: { get_param: db_name }
            db_user: { get_param: db_username }
            db_password: { get_param: db_password }

  WebServer:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_id }
      flavor: { get_param: instance_type }
      key_name: { get_param: key_name }
      networks:
        - network: {get_param: net}
      security_groups:
        - default
      user_data:
        str_replace:
          template: |
            #!/bin/bash -v

            yum -y install httpd wordpress

            sed -i "/Deny from All/d" /etc/httpd/conf.d/wordpress.conf
```

```
            sed -i "s/Require local/Require all granted/" /etc/httpd/
                conf.d/wordpress.conf
            sed -i s/database_name_here/db_name/ /etc/wordpress/wp-
                config.php
            sed -i s/username_here/db_user/      /etc/wordpress/wp-
                config.php
            sed -i s/password_here/db_password/  /etc/wordpress/wp-
                config.php
            sed -i s/localhost/db_ipaddr/        /etc/wordpress/wp-
                config.php
            setenforce 0 # Otherwise net traffic with DB is disabled
            systemctl start httpd.service

          params:
            db_rootpassword: { get_param: db_root_password }
            db_name: { get_param: db_name }
            db_user: { get_param: db_username }
            db_password: { get_param: db_password }
            db_ipaddr: { get_attr: [DatabaseServer, networks, osm-ext,
                 0] }
  floating_ip:
     type: OS::Neutron::FloatingIP
     properties:
        floating_network: {get_param: external_network}
  association:
     type: OS::Neutron::FloatingIPAssociation
     properties:
        floatingip_id: { get_resource: floating_ip }
        port_id: {get_attr: [WebServer, addresses, {get_param: net},
            0, port]}

outputs:
  WebsiteURL:
    description: URL for Wordpress wiki
    value:
      str_replace:
        template: http://host/wordpress
        params:
          host: { get_attr: [WebServer, networks, osm-ext, 0] }
```

En segundo lugar, tenemos las plantillas utilizadas para desplegar los servicios en contenedores proporcionados por Zun.

Fichero "*zun-template.yaml*"

```
heat_template_version: 2017-09-01

parameters:
  count:
    type: number
    default: 1
  tenant_network:
    type: string
    default: osm-ext
  external_network:
    type: string
    default: public
```

```
resources:
  secgroup:
    type: OS::Neutron::SecurityGroup
    properties:
      name: sg_wordpress
      description: wordpress security group
      rules:
      - protocol: icmp
      - protocol: tcp
        port_range_min: 80
        port_range_max: 80
      - protocol: tcp
        port_range_min: 3306
        port_range_max: 3306
  db:
    type: OS::Zun::Container
    properties:
      image: mysql:5.7
      environment:
        MYSQL_ROOT_PASSWORD: rootpass
        MYSQL_DATABASE: wordpress
      security_groups:
      - {get_resource: secgroup}
      networks:
      - network: {get_param: tenant_network}
  web_server_group:
    type: OS::Heat::ResourceGroup
    properties:
      count: {get_param: count}
      resource_def:
        type: wordpress.yaml
        properties:
          tenant_network: {get_param: tenant_network}
          external_network: {get_param: external_network}
          db_host_ip: {get_attr: [db, addresses, {get_param:
              tenant_network}, 0, addr]}
          secgroup: {get_resource: secgroup}
```

Fichero "*wordpress.yaml*"

```
heat_template_version: 2017-09-01

parameters:
  tenant_network:
    type: string
    default: osm-ext
  external_network:
    type: string
    default: public
  db_host_ip:
    type: string
  secgroup:
    type: string

resources:
  wordpress:
    type: OS::Zun::Container
    properties:
      image: "wordpress:latest"
      environment:
```

```
        WORDPRESS_DB_HOST: {get_param: db_host_ip}
        WORDPRESS_DB_USER: root
        WORDPRESS_DB_PASSWORD: rootpass
      security_groups:
      - {get_param: secgroup}
      networks:
      - network: {get_param: tenant_network}
  floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: {get_param: external_network}
  association:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      floatingip_id: { get_resource: floating_ip }
      port_id: {get_attr: [wordpress, addresses, {get_param:
          tenant_network}, 0, port]}
```

Apéndice D

Fichero despliegue de VNFs y NS en OSM.

Este anexo contiene los archivos utilizados como descriptores de VNFs y NS empleados en el despliegue de servicios lanzados desde OSM. Se incluirán los archivos descriptores y sus respectivos ficheros de *cloud-config*.

Archivo descriptor de VNF de MySQL:

```
vnfd-catalog:
    vnfd:
    -   connection-point:
        -   name: eth0
            type: VPORT
        description: MySQL server
        id: mysql
        mgmt-interface:
            cp: eth0
        name: mysql
        short-name: mysql
        vdu:
        -   cloud-init-file: cloud-init.cfg
            id: mysql1
            image: fedora-cloud-base
            interface:
            -   external-connection-point-ref: eth0
                mgmt-interface: true
                name: eth0
                position: 1
                type: EXTERNAL
                virtual-interface:
                    type: PARAVIRT
            name: mysql1
            vm-flavor:
                memory-mb: 2048
                storage-gb: 10
                vcpu-count: 1
        vendor: Telefonica
        version: '1.0'
```

```
        vnf-configuration:
            config-access:
                ssh-access:
                    default-user: fedora
                    required: true
```

Archivo *Cloud-config*:

```
#cloud-config
password: osm4u
chpasswd: { expire: False }
ssh_pwauth: True

package_update: true
packages:
- sshpass

runcmd:
- yum -y install mariadb mariadb-server
- touch /var/log/mariadb/mariadb.log
- chown mysql.mysql /var/log/mariadb/mariadb.log
- systemctl start mariadb.service

- mysqladmin -u root password admin
- cat << EOF | mysql -u root --password=admin
- CREATE DATABASE wordpress;
- GRANT ALL PRIVILEGES ON wordpress.* TO "admin"@"%"
- IDENTIFIED BY "admin";
- FLUSH PRIVILEGES;
- EXIT
- EOF
```

Archivo descriptor de VNF de WordPress:

```
vnfd-catalog:
    vnfd:
    -   connection-point:
        -   name: eth0
            type: VPORT
        description: Wordpress engine
        id: wordpress
        mgmt-interface:
            cp: eth0
        name: wordpress
        short-name: wordpress
        vdu:
        -   cloud-init-file: cloud-init.cfg
            id: wordpress1
            image: fedora-cloud-base
            interface:
            -   external-connection-point-ref: eth0
                mgmt-interface: true
                name: eth0
                position: 1
                type: EXTERNAL
                virtual-interface:
                    type: PARAVIRT
            name: wordpress1
            vm-flavor:
```

```
                memory-mb: 2048
                storage-gb: 10
                vcpu-count: 1
        vendor: Telefonica
        version: '1.0'
        vnf-configuration:
            config-access:
                ssh-access:
                    default-user: fedora
                    required: true
```

Archivo *Cloud-config*:

```
#cloud-config
password: osm4u
chpasswd: { expire: False }
ssh_pwauth: True

package_update: true
packages:
- sshpass

runcmd:
- yum -y install httpd wordpress
- sed -i "/Deny from All/d" /etc/httpd/conf.d/wordpress.conf
- sed -i "s/Require local/Require all granted/" /etc/httpd/conf.d/
    wordpress.conf
- sed -i s/database_name_here/wordpress/ /etc/wordpress/wp-config.php
- sed -i s/username_here/admin/      /etc/wordpress/wp-config.php
- sed -i s/password_here/admin/  /etc/wordpress/wp-config.php
- sed -i s/localhost/192.168.1.170/        /etc/wordpress/wp-config.
    php
- setenforce 0 # Otherwise net traffic with DB is disabled
- systemctl start httpd.service
```

Archivo descriptor del *Network Service* de OSM (NSD):

```
nsd-catalog:
    nsd:
    -   constituent-vnfd:
        -   member-vnf-index: wordpress
            vnfd-id-ref: wordpress
        -   member-vnf-index: mysql
            vnfd-id-ref: mysql
        description: NS consisting of two VNF wordpress and mysql
            connected to mgmtnet
        id: wordpress_mysql
        logo: osm.png
        name: wordpress_mysql
        ns-configuration:
            relation:
            -   entities:
                -   endpoint: db
                    id: wordpress
                -   endpoint: db
                    id: mysql
                name: database
```

```
        short-name: wordpress_mysql
        vendor: OSM
        version: '1.0'
        vld:
        -   id: mgmtnet
            mgmt-network: 'true'
            name: mgmtnet
            type: ELAN
            vim-network-name: mgmt
            vnfd-connection-point-ref:
            -   member-vnf-index-ref: wordpress
                vnfd-connection-point-ref: eth0
                vnfd-id-ref: wordpress
            -   member-vnf-index-ref: mysql
                vnfd-connection-point-ref: eth0
                vnfd-id-ref: mysql
```

Para finalizar se incluye el fichero de configuración utilizado en la instanciación del NS, donde se asignan las direcciones IP y la red a utilizar.

```
---
vld:
  -
    name: mgmtnet
    vim-network-name: osm-ext      #The network in the VIM to connect
        all nodes of the clusters
    vnfd-connection-point-ref:
      -
        ip-address: "192.168.1.169"
        member-vnf-index-ref: wordpress
        vnfd-connection-point-ref: eth0
      -
        ip-address: "192.168.1.170"
        member-vnf-index-ref: mysql
        vnfd-connection-point-ref: eth0
```