Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

IMPLEMENTACIÓN DE UNA IAAS MEDIANTE OPENSTACK Y OPENDAYLIGHT PARA DAR SOPORTE A SERVICIOS EN 5G

TRABAJO FIN DE MÁSTER

Víctor Quilón Ranera

2020

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

IMPLEMENTACIÓN DE UNA IAAS MEDIANTE OPENSTACK Y OPENDAYLIGHT PARA DAR SOPORTE A SERVICIOS EN 5G

Autor
Víctor Quilón Ranera

Director
Luis Bellido Triana

Departamento de Ingeniería de Sistemas Telemáticos

2020

Resumen

Las comunicaciones 5G serán una tecnología que cambiará la red por completo, ya que en estos momentos el incremento de los dispositivos IoT es un reto bastante complejo. Es consecuencia de que la red actualmente no tiene la flexibilidad necesaria que requieren estos dispositivos IoT. Una de las soluciones en 5G, y la cual cambiará el concepto de red actual es el Network Slicing. Para implementar la tecnología del NS de 5G hay que utilizar arquitecturas basadas en virtualización de redes mediante tecnologías como Software Defined Networking y Network Function Virtualization. Con el uso de ellas, se implementan arquitecturas ETSI MANO NFV a través de diversas plataformas de código abierto como OpenStack. Las mismas tienen una gran comunidad detrás de ellas que interactúan de forma activa en los proyectos, e incluso compañías grandes realizan inversiones y participan en el desarrollo y la proliferación de este tipo de plataformas de código abierto. La integración de un controlador SDN y plataformas NFV como OpenStack Tacker, permiten el despliegue de topologías de red que dan soporte a servicios construidos con propiedades similares al Network Slicing.

La idea principal que se quiere lograr es hacer transparente la infraestructura física que sustenta los servicios de red para el usuario, de manera que la configuración de los servicios de comunicaciones se simplifica significativamente. Esta mayor facilidad genera despliegues más rápidos y eficientes, pero a la vez requiere de perfiles altamente especializados asociados al diseño y planificación de redes. Se puede lograr mediante herramientas de automatización que facilitan el uso y el entendimiento de arquitecturas complejas, las cuales soportan el peso de las tecnologías que habilitan el camino a servicios aislados consumidos bajo la misma red, y todo sin necesidad de reconfigurar físicamente la red.

Abstract

The 5G technology communications will change the network completely, because now the increase of IoT devices is a fairly complex challenge. It is a consequence that the network currently does not have the necessary flexibility required by these IoT devices. The solutions in 5G which will change the current network concept is Network Slicing. To implement the 5G NS technology, we must use architectures based on network virtualization using technologies such as Software Defined Networking and Network Function Virtualization. With ETSI MANO NFV architectures are implemented through various open source platforms such as OpenStack. They have a large community behind them that interact actively in many projects, and even large companies make investments and participate in the development of this type of open source platforms. The integration of an SDN controller and NFV platforms such as OpenStack Tacker, allow the deployment of network topologies that support services built with properties like Network Slicing.

The main idea to achieve is to make transparent the physical infrastructure that supports the network services for the user, so the configuration of communications services is significantly simplified. This greater ease generates faster efficient deployments, but at the same time requires highly specialized profiles associated with network design and planning. It can be achieved through automation to facilitate the use and understanding of complex architectures, which support the weight of technologies that enable the path to isolated services consumed under the same network, and all without the need to physically reconfigure the network.

Palabras clave

OpenStack, OpenDaylight, IoT, 5G, Network Slicing, NFV, SDN, Virtualización, Computación en la nube, Devstack, Netoworking-ODL, SFC, VNF, NFVO, NFVI, VIM.

Keywords

OpenStack, OpenDaylight, IoT, 5G, Network Slicing, NFV, SDN, Virtualization, Cloud Computing, Devstack, Netoworking-ODL, SFC, VNF, NFVO, NFVI, VIM.

Índice general

Resumen I
Abstract II
Palabras clave III
Keywords III
Índice general IV
Índice de figuras VI
1 Introducción y objetivos 1
1.1 Comunicaciones 5G 2
1.2 Tecnologías NFV y SDN 4
1.2.1 SDN 5
1.2.2 NFV 7
1.2.3 Integración SDN-NFV 9
1.3 Tecnología Network slicing 11
1.4 Herramientas de despliegue 13
1.5 OpenStack como solución de IaaS (Objetivos) 15
2 Estado del Arte 17
2.1 Open Source Mano 17
2.2 Open Networking Automation Platform 18
2.3 Open Platform for NFV 20
2.4 Open Baton 21
2.5 Conclusión 23
3 Implementación de una IaaS 26
3.1 OpenStack 26
3.1.1 OpenStack Tacker 27
3.1.2 OpenStack Service Function Chaining 28
3.1.3 Componentes OpenStack en la ETSI MANO 30
3.2 OpenDaylight 31
3.2.1 Arquitectura de ODL 31
3.2.2 Interfaces de comunicación 33
3.3 Integración OpenStack y ODL 35

3.3.1 OpenStack Neutron: Plugin ML2 ... 36
3.3.2 Netvirt ... 37
3.3.3 OVSDB ... 38
3.3.4 Drivers ODL ... 40
3.3.5 Arquitectura tacker y ODL ... 40
3.4 Devstack ... 41
3.4.1 Herramienta de Automatización ... 41
3.5 Otras herramientas de automatización ... 46
4 Resultados ... 47
4.1 Preparación del sistema ... 47
4.1.1 Aumentar la RAM mediante SwapFile ... 47
4.1.2 Actualización e Instalaciones previas ... 48
4.1.3 Instalación de OVS ... 48
4.1.4 Aumentar el límite de watchers del sistema ... 49
4.1.5 Requisitos Previos del sistema ... 49
4.2 Instalación y configuración de Devstack ... 50
4.2.1 Instalación Devstack ... 50
4.3 Topología de red creada ... 54
4.4 Creación de VNF ... 54
5 Conclusiones y líneas futuras ... 56
5.1 Conclusiones ... 56
5.1.1 Fallos Encontrados ... 56
5.2 Líneas futuras ... 58
6 Bibliografía ... 59
7 Anexo A: Aspectos éticos, económicos, sociales y ambientales ... 65
7.1 A.1 Introducción ... 65
7.2 A.2 Descripción de impactos relevantes relacionados con el proyecto ... 65
7.3 A.3 Análisis detallado de alguno de los principales impactos ... 65
7.4 A.4 Conclusiones ... 66
8 Anexo B: Presupuesto económico ... 67

Índice de figuras

Figura 1: Pronóstico global del número de dispositivos IoT conectado [5] 2
Figura 2: Arquitectura de SDN [20] 6
Figura 3: Arquitectura SDN para Network Slicing definida por el ONF 7
Figura 4: Arquitectura NFV MANO definida por la ETSI 8
Figura 5: Implementación de la arquitectura SDN definida por la ETSI [27] 10
Figura 6: Arquitectura NFV definida por la ETSI con controladores SDN 10
Figura 7: Diagrama de las slices en redes 5G creado por NGMN 11
Figura 8: Arquitectura genérica de 5G Network Slincing [34] 12
Figura 9: Diagrama de Network Slicing [36] 13
Figura 10: Gráfico de contribuciones a Openstack Rocky 15
Figura 11: Arquitectura de OSM 18
Figura 12: Alcance de ONAP en la ETSI NFV MANO [49] 19
Figura 13: Arquitectura de la plataforma ONAP [51] 20
Figura 14: Alcance de OPNFV de la arquitectura ETSI NFV [52] 21
Figura 15: Arquitectura de Open Baton [54] 22
Figura 16: Arquitectura del componente Tacker de OpenStack 23
Figura 17: Diagrama de todos los componentes de OpenStack Rocky 27
Figura 18: Modelado de VNFs dentro de Tacker [57] 28
Figura 19: Arquitectura SFC dentro de OpenStack 29
Figura 20: Flujo de SFC en Tacker [60] 30
Figura 21: Arquitectura de ODL Oxygen 31
Figura 22: Flujo de generación de código mediante YANG 33
Figura 23: Arquitectura del Service Abstraction Layer 34
Figura 24: Evolución de la AD-SAL al MD-SAL en el SAL [63] 34
Figura 25: Integración de OpenDaylight con OpenStack [64] 36
Figura 26: Flujo de Neutron Northbound [66] 37
Figura 27: Estructura Netvirt dentro de ODL [66] 38
Figura 28: OpenStack y ODL Netvirt [67] 39
Figura 29: Componentes de Open vSwitch 39
Figura 30: Integración de OpenStack Tacker y ODL 41
Figura 31: Ejemplo de la topología de red de la infraestructura creada [82] 54

1 Introducción y objetivos

Estamos en medio de la era de la información, en la que somos capaces de digitalizar el mundo físico en tiempo real, y esto es posible gracias en gran parte a un conjunto de dispositivos denominados Internet of Things (IoT) que se encargan de tomar datos del entorno que nos rodea. Los IoT y los dispositivos tradicionales arrojan a cada minuto una cantidad de datos enorme, ya que en la actualidad se estima que hay más de 26.000 millones de dispositivos conectados [1] y existen unas previsiones de que al final de 2025 estos dispositivos generarán más de 79.4 ZettaBytes (1.000 millones de TeraBytes) por año, creciendo exponencialmente los siguientes años [2]. Los IoT se encuentran presentes en diferentes áreas [3], como por ejemplo en el ámbito de negocio, Connected Industry, Smart Agriculture, Connected Building, Smart Supply Chain, Smart Retail, interconectando una gran cantidad de dispositivos que ayudan a mejorar sus servicios y por lo tanto a hacerlos más eficientes. También se puede ver como en servicios públicos como en la sanidad, Connected Health, donde se monitorizan en tiempo real las constantes y los datos sensibles de pacientes. Además, en la parte de consumo, como Connected Car, Smart Home, las cuales permiten ayudar a la seguridad y protección de las personas vigilando y tomando medidas en tiempo real del coche o la casa [4]. Esto supone un reto para las redes actualmente desplegadas, ya que deben tener unas características específicas como por ejemplo poder manejar una tasa de datos muy alta junto con una baja latencia y numerosos dispositivos conectados en regiones adyacentes. Por ello hay que tener en cuenta también que existe un incremento constante de dispositivos que supone un gran hándicap para las redes que hoy actualmente puedan soportar esa carga.

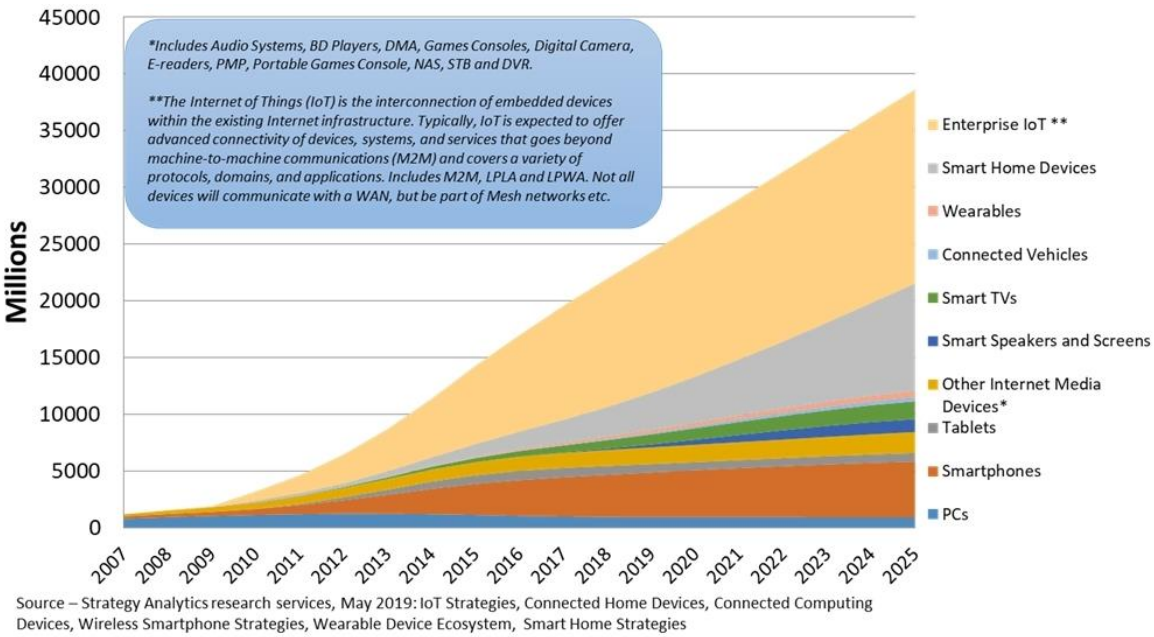

Figura 1: Pronóstico global del número de dispositivos IoT conectado [5]

1.1 Comunicaciones 5G

Existen diferentes tecnologías de comunicaciones que intentan dar soporte a esta gran variedad y cantidad de dispositivos IoT que se desplegarán en los siguientes años. Por todo ello se han desarrollado nuevas tecnologías Low-Power Wide-Area Network (LPWAN) que como su nombre indica son estandarizaciones que tienen un largo alcance, muy baja potencia, baja tasa de transmisión y a bajo coste. Un ejemplo de ello son los estandarizados por 3GPP lanzados en la Release 13, LTE-M y NB-IOT. Por otro lado, las tecnologías procedentes de diferentes empresas como SigFox, LoRa o ZigBee, que son un estándar inalámbrico para conectar M2M a bajo coste y consumo eléctrico [6]. Las estandarizaciones ya mencionadas LTE Category M1 (LTE-M) y Narrowband IoT (NB-IOT) son las más extendidas, lanzadas por porte de 3GPP en la Release 13. En esta Release se introdujeron notables mejoras en comunicaciones de tipo máquina (eMTC), por ejemplo, anchos de banda más bajos tanto en enlace de subida como de bajada (a 1.5MHz), una potencia de transmisión menor, lo que influye en una mejora notable de autonomía de las baterías. Además, al tener esta reducción se produjo un abaratamiento de los costes de un 50% y también permitió acceder a dispositivos con mala cobertura debido a que se amplió el alcance en 15dB [7]. La estandarización LTE-M es la variante de LTE que da soporte a dispositivos IoT [8] pensada para soportar la movilidad, por ejemplo, dispositivos de medición en diferentes medios de transporte. Y por otro lado el estándar NB-IoT [9] se define específicamente para un conjunto de dispositivos que no son sensibles a retardo, ya que tiene una latencia de entre 1,6s y 10s, algo alto en comparación con LTE-M y se encuentran en zonas con cobertura escasa, o dentro de edificios. Al tener una buena cobertura en interiores, se incrementan el número de dispositivos conectados, un bajo consumo eléctrico y un abaratamiento de los dispositivos que

lo utilicen. Puede observarse una comparación de las dos estandarizaciones en la Tabla 1 de abajo.

La llegada de la quinta generación de tecnologías de comunicaciones móviles, conocida por las siglas 5G, conlleva un conjunto de mejoras para las comunicaciones, como un incremento en la tasa de datos, bajas latencias, un gran ancho de banda y multitud de dispositivos conectados al mismo tiempo con un menor consumo de energía. Estas características abren las puertas a la tecnología IoT pudiendo así beneficiarse de los datos que generan estos dispositivos. Las tecnologías predecesoras del 5G son capaces de soportar comunicaciones en las que participan grandes cantidades de dispositivos IoT desplegados. Por lo que esta nueva tecnología de comunicaciones 5G, será una evolución de tecnologías actuales con mejoras significativas y nuevas características que permitirá la gran escalabilidad que protagonizarán en adelante los dispositivos IoT y otros servicios.

	LTE-M (4G) **LTE Cat. M1**	**NB-IoT (4G)** **LTE Cat. NB1**	**5G**
Downlink peak rate	1 Mbps	100 kbps	10 Gbps
Uplink peak rate	1 Mbps	144 kbps	
Latencia	10ms–15ms	1.6s–10s	1ms
Modo Duplex	Half Duplex	Half Duplex	Full Duplex
Vida de la batería	10 años	Más de 10 años	Más de 10 años
Cobertura interior	Buena	Excelente	Excelente

Tabla 1: Evolución tecnologías de comunicación

Como se puede ver, entre las mejoras en 5G frente a las estandarizaciones predecesoras destacan una mayor tasa de datos, comunicaciones ultra confiables o una reducción de la latencia. Además, se implementan diferentes optimizaciones para comunicaciones en dispositivos IoT como un incremento en el número de dispositivos conectados (100 veces más que lo que LTE puede actualmente), ampliación de la vida de la batería, o una reducción de la energía usada. Por estos motivos, el 5G será clave para el desarrollo de tecnologías de comunicación para dispositivos IoT [10].

Esta tecnología de comunicaciones está prevista para ser completamente operacional en el 2020 y para entonces se espera que haya más de 30.000 millones de dispositivos IoT conectados. Esto supone que deberá haber tecnologías que sean capaces de hacer que la red se adapte a cada uno de estos dispositivos mediante 5G. Una de las formas, es el uso de la tecnología de Network Sclicing (NS) que ofrece la capacidad a los operadores de segmentar una red por servicio, dependiendo de los requisitos que se necesiten individualmente, pudiendo desplegar una red lógica por cada tipo de servicio en una única infraestructura física en común [11]. Según 5G Infrastructure Public Private Partnership (5G PPP) [12], NS es comunicación Edge to Edge (E2E) que abarca segmentos de la red como Radio Access Network (RAN) y Core Network (CN). Esta tecnología es un conjunto de capas de red,

conocidas como Network Slices, y se definen como un grupo gestionado de recursos, funciones de red, control, administración/orquestación y servicio en cualquier momento dado. Además, las características de una Network Slice son que es programable, puede mostrar sus propiedades y soporta al menos un tipo de servicio. En su estructura se pueden diferenciar tres componentes diferentes, la red de acceso, la red de transporte, la red central y las redes de frontera, estando cada capa aislada una de la otra, por lo que los fallos no tienen impactos en capas adyacentes [13]. De esta forma podemos resumir las siguientes características de una Network Slice de la siguiente forma:

- Autocontenida.
- Mutuamente aisladas una de otras.
- Manejable y programable.
- Soporte multiservicio.
- Multi-cliente.

La tecnología Network Slicing de 5G parte del concepto de abrir las redes que poseen los operadores a los clientes, otorgándoles la capacidad de desplegar diferentes redes autocontenidas y aisladas de forma vertical, desde la capa física, la de virtualización hasta la de servicios. Esta virtualización de la red se apoya en los conceptos de tecnologías como Software-Defined Networking (SDN) y Network Function Virtualization (NFV), de las cuales, NFV satisface los requisitos de flexibilidad, agilidad o escalabilidad, y SDN se encarga de la programación de redes y la maximización del rendimiento del flujo de tráfico. Estas tecnologías son claves para el desarrollo y el despliegue de NS ya que cumplen con todos los requisitos que necesitan las arquitecturas de comunicaciones 5G. De esta forma es posible hacer uso de NS mediante arquitecturas como la propuesta por el Open Networking Foundation (ONF) para SDN y la arquitectura ETSI NFV MANO desarrollada por el ETSI NFV Industry Special Group. De hecho, existen diferentes proyectos de NS que hacen uso de la combinación de SDN y NFV como 5GNORMA [14], 5GEx [15], 5GinFIRE [16] o 5G!Pagoda [17]. Estos proyectos constan de arquitecturas muy complejas que en parte hacen uso de SDN y NFV para poder construir una infraestructura capaz de ser flexible a cambios y que se pueda beneficiar de la virtualización y programación de la red que tienen estas tecnologías. Por otro lado, existen diferentes proyectos que hacen uso de plataformas de código abierto y facilitan el uso y comprensión de estas tecnologías.

1.2 Tecnologías NFV y SDN

Las tecnologías NFV y SDN son claves para la construcción de arquitecturas de red programables que permitan mayor flexibilidad que las redes tradicionales, y por lo tanto son el núcleo para el funcionamiento de infraestructuras que dan soporte al NS. Además, un estudio realizado por el Instituto Europeo de Normas de Telecomunicaciones (ETSI) muestra que mediante el uso de NFV y SDN es posible ahorrar en costos de mantenimiento, auto

escalado o mejorar la capacidad de recuperación del sistema [18]. Debido a esto, el uso de NFV y SDN da la capacidad a los operadores de personalizar las redes de forma más rentable gracias a que podrían construir redes dedicadas de acuerdo con los requisitos de cada usuario.

1.2.1 SDN

La principal característica más importante del paradigma de red SDN es la separación del plano de control del plano de datos, esto quiere decir que facilita la implantación de servicios de red de forma escalable, sin necesidad de tener que administrar la red a un bajo nivel. Nace debido a la necesidad de mejorar el rendimiento de la red, y romper con el uso continuo de equipos y sustitución de estos para pasar a usar los ciclos más ágiles del software y que abarcan más plataformas de proveedores. Se desarrolla a principios del 2000 pero no se implementaría la idea principal de la separación del plano de control del plano de datos debido a que podría acarrear riesgos de seguridad en caso de que el plano de control fallara. Por ello, no es hasta el 2008 cuando se crea una API para OpenFlow [19] (protocolo desarrollado por la Universidad de Standford que determina el camino de reenvío de paquetes en una red de switches) que se encargase de las comunicaciones entre los planos de control. Unos años después se funda el Open Networking Foundation (ONF) para promover el uso de SDN y OpenFlow. Tener esta separación facilita la configuración de la red, se simplifica y se obtienen unas políticas de red más flexibles, aparte de que se fomenta la contribución de investigadores ya que implica un reto. Además, fue necesario debido a que la configuración individual de los equipos de red necesitaba de programación a bajo nivel específico para cada distribuidor, y en cuanto había fallos o cambios en la red, la flexibilidad era muy baja.

En la Figura 2 se separa el SDN Controller o plano de control, que se encarga de cómo manejar las políticas y las reglas de la red, del plano de datos donde se lleva a cabo el reenvío de información dado un conjunto de reglas. El controlador SDN a través de las API puede realizar diferentes tareas como, crear y administrar un conjunto de reglas mediante el uso de la API Northbound, incluyendo soporte a aplicaciones SDN como enrutamiento, firewall o calidad de servicio. Por otro lado, mediante la API Southbound otorga comunicación entre el SDN Controller y los elementos del plano de datos como los switches.

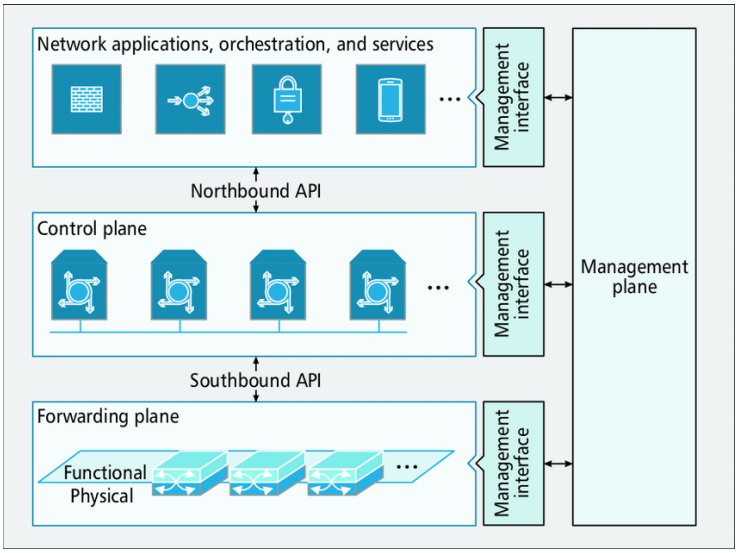

Figura 2: Arquitectura de SDN [20]

Más abajo se puede observar una arquitectura parecida a la normal de SDN, pero esta es una variante definida por ONF para implementar Network Slicing y desde el punto de vista del flujo entre cliente y servicio [21]. En esta arquitectura se puede observar como el controlador SDN, en el centro, media con los requisitos del cliente realizando cambios de flujo en la red, el estado, parámetros de servicio o disponibilidad de recursos. Lo que se destaca en la Figura 3 es la relación entre los dos principales contextos, el lado de cliente y el lado de servidor. Se entiende un cliente como cualquier usuario que hace uso, u otra entidad dentro del mismo sistema. Este representa todo recurso necesario en el controlador para dar soporte al cliente, incluyendo toda la información relacionada con el servicio. De igual forma el contexto de servidor se refiere a todo lo que necesite el controlador para interactuar con un conjunto de recursos.

La mayoría de estos recursos del que requiere el controlador son proporcionados por NFV centrándose en la creación y soporte de estas funcionalidades y recursos y el SDN otorga a ambos lados una abstracción las funciones de red. La administración, llevada por el "Administrador" que se muestra en la Figura 3, se llevaría a cabo mediante un OSS, siendo el responsable de las configuraciones y políticas de los contextos de cliente y servidor. Este se encarga de las configuraciones y restricciones de ambos contextos. Además, cuando existe una relación de negocio entre el contexto de cliente y el de servidor, el controlador de SDN administra la sesión del cliente pudiendo modificar estos servicios mediante el uso de acuerdos de servicio o Service Level Agreement.

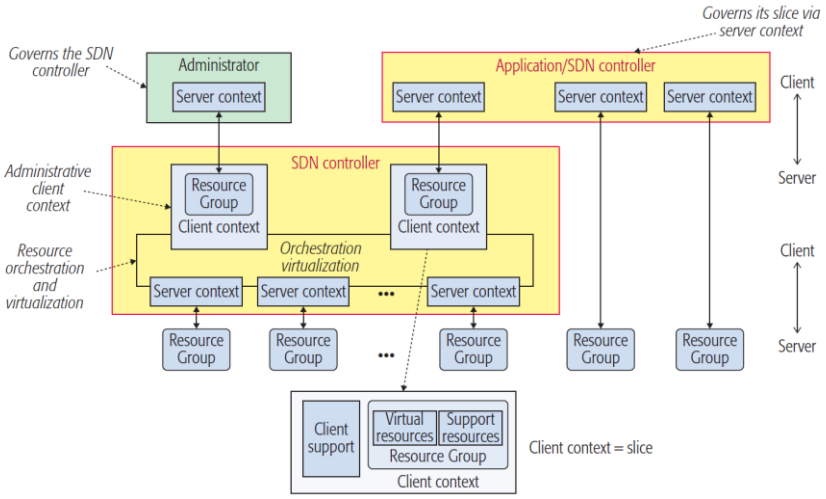

Figura 3: Arquitectura SDN para Network Slicing definida por el ONF

Esta arquitectura realizada por ONF permite la implementación de NS, ya que una de las características que se definen para una slice, es abstraerse de los recursos o estados como hace el contexto de cliente, pudiendo controlarlos mediante las API. Además, la arquitectura tiene un control lógicamente centralizado, que significa que el control se realiza desde fuera, viéndose todo como una entidad única.

1.2.2 NFV

Esta tecnología nace de la necesidad de implementar soluciones ágiles y efectivas contra las redes para otorgarles mayor escalabilidad y facilidad de implementación en diferentes puntos para la gestión de la red. Previamente las redes estaban limitadas a HW y SW ya que estaban embebidos en la propia infraestructura de la red. La primera vez que se mencionó fue a finales del 2012 en un artículo de la ETSI en Alemania donde se trataban también tecnologías como SDN y OpenFlow [22]. En este "Whitepaper" se resumían los beneficios que otorgarían la implementación de VNF en redes, entre las cuales se definen:

- Reducción de costes en equipos y consumo eléctrico debido a las economías de escala.
- Permitir a los operadores de red llegar antes al punto de madurez minimizando el ciclo de innovación de red. De esta forma se incrementa así la velocidad del Time to Market (Plazo comprendido entre el concepto del producto hasta que se dispone al cliente).
- Compartir los recursos de una plataforma dando la posibilidad de proporcionar diferentes servicios a diferentes clientes.
- Los servicios pueden ser fácilmente escalables.
- Integración de múltiples aplicaciones virtuales de la mano de diferentes distribuidores.

NFV transforma las comunicaciones de las redes, ya que permite a consumidores y a empresas utilizar las funciones de red de determinados distribuidores a software alojado en

plataformas con una preconfiguración establecida para un uso específico, también denominados plataformas Commercial Off-The-Self (COTS). De esta forma el uso de NFV provee servicios como virtualización de máquinas donde cada una realiza una función de red como firewalls, balanceadores de carga, etc. La Figura 4 muestra la arquitectura NFV definida por la ETSI [23], donde se pueden observar las diferentes partes que la componen.

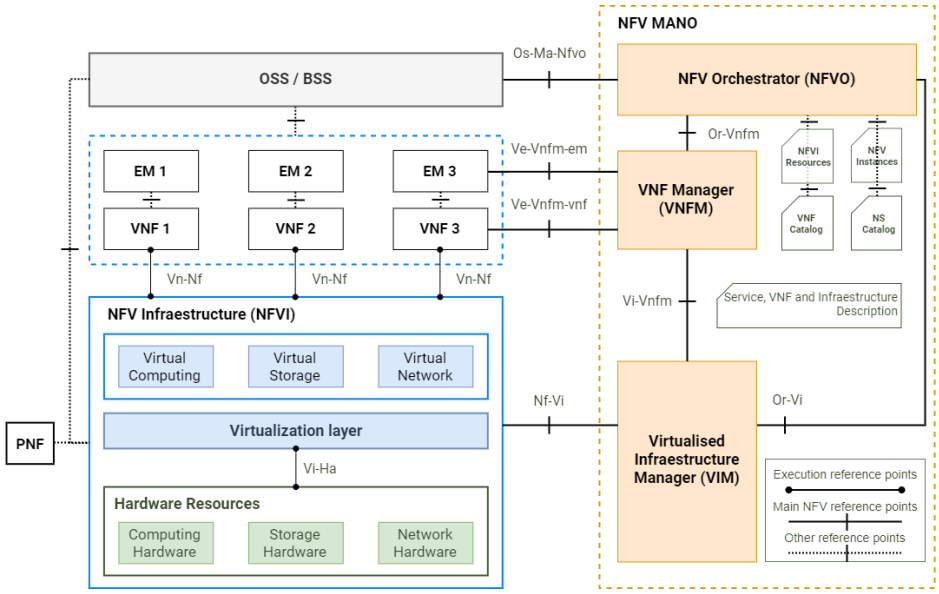

Figura 4: Arquitectura NFV MANO definida por la ETSI

- La NFV Infraestructure (NFVI), contiene los módulos de computación virtual, de almacenamiento y de red.
- La capa de función de red consta de los Virtual Network Function (VNFs) y con cada uno sus sistemas de administración. Además, cada VNF puede estar compuesto de subfunciones llamadas VNF Components (VNFCs).
- Los Physical Network Functions (PNF) junto con los VNF pueden formar el plano de reenvío (forwarding plane) [33].
- Dentro del NFV Management and Orquestation (MANO) encontramos el Virtualised Infrastructure Manager (VIM) que administra y controla los recursos de almacenamiento, computación y red de la NFVI. También el VNF Manager (VNFM) que se encarga de vigilar el despliegue, escalar y borrar cada VNF en el VIM.
- Por otro lado, el módulo NFV Orchestration (NFVO) se encarga del ciclo de vida de VNF y además de los recursos de la NFVI [25].
- El Network Management System (NMS), el cual realiza las tareas de gestión de redes e interactúa con los componentes de MANO. El NMS está compuesto de los Element Management (EMs) que son los responsables de las FCAPS (Fault, Configuration, Accounting, Performance & Security) de un VNF.
- El Operation/Business Support System (OSS/BSS) es una colección de sistemas y administración de sistemas que los proveedores de servicios de la red usan para operar en sus servicios.

1.2.3 Integración SDN-NFV

La arquitectura NFV puede estar implementada junto con SDN incluyendo su controlador, aplicaciones y recursos de múltiples formas. Por lo que existen diferentes escenarios en los que las entidades mencionadas se relacionan entre ellas. En primer lugar, posibles escenarios donde se encuentran los controladores SDN:

- **VNF 3**: Como un VNF siendo parte del NFVI.
- **VIM**: El controlador está unido con el VIM.
- **NFVI**: Una configuración común es introducir el controlador en el NFVI, pero sin estar implementado como una VNF.
- **OSS/BSS**: El controlador forma parte del OSS.
- **PNF**: Por último, el controlador como un PNF.

De esta forma los recursos de SDN pueden ubicarse en los siguientes lugares dependiendo de donde se encuentren los controladores SDN:

- **Network Hardware**: En la parte de los recursos físicos en un ruter o un switch y puede encontrarse el controlador en el VIM, el NFVI o como un VNF.
- **Virtual Network**: También como un ruter o un switch virtual, con el controlador en el NFVI, VIM o como un VNF.
- **Computing Hardware**: En un servidor de registro de nombres de domino o Network Information Center [26], en un conmutador habilitado para SDN basado en software, un e-switch. Se comunica estando el controlador en NFVI, VIM o en un VNF.
- **VNF 1**: Dentro de las funciones virtuales de red, que haga de función de switch o ruter dentro de un VNF y puede comunicarse con controlador estando en el OSS, NFVI, VIM o VNF.

Por último, las aplicaciones de SDN pueden tener diferentes posiciones pudiendo conectar con el controlador si está en determinadas localizaciones:

- **Network Hardware**: Como un dispositivo físico que se comunique con el controlador si está ubicado en el NFVI, VIM, o como un VNF.
- **VIM**: Dentro del VIM como una aplicación que conecta con el controlador SDN ubicado en el mismo VIM, en el NFVI, o como un VNF.
- **VNF 3**: Como una VNF comunicándose con el controlador para la gestión de las políticas del filtro de direcciones, pudiendo estar en el NFVI, el VIM o como un VNF.
- **EM 3**: Puede coleccionar métricas o configurar parámetros siendo un EM y hace interfaz únicamente si el controlador es un VNF.
- **OSS/BSS**: Una aplicación SDN para definiciones del servicio dentro del OSS/BSS pudiendo comunicarse con controladores localizados en el OSS, en el VIM o si es un VNF.

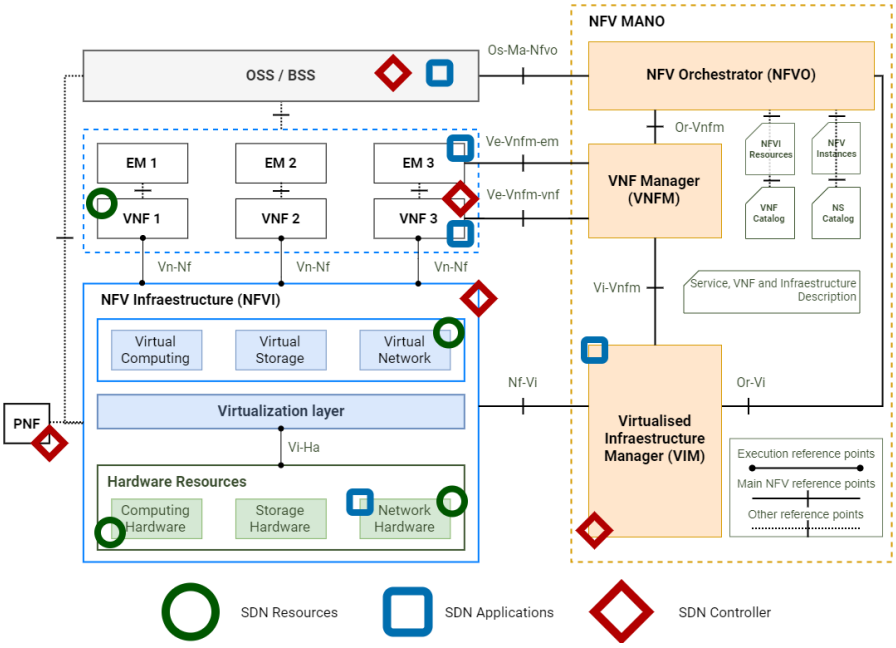

Figura 5: Implementación de la arquitectura SDN definida por la ETSI [27]

Una posible implementación de SDN para su uso en Network Slicing se realiza mediante la agregación de dos controladores en la arquitectura, el Infrastructure SDN Controller (IC) y el Tenant SDN Controller (TC) [28]. El IC está implementado en el NFVI o en el VIM y está administrado por el VIM, que configura y administra los recursos de la red para proporcionar conectividad a los VNF [29]. El otro controlador, el TC, ubicado en un VNF administra dinámicamente los VNF usados para realizar los servicios de red del cliente y estas tareas de operación y administración del TC se activan por las aplicaciones que estén sobre él, como el OSS [30]. Cada controlador se puede controlar mediante la API Southbound, utilizando protocolos como OpenFlow o NETCONF [31].

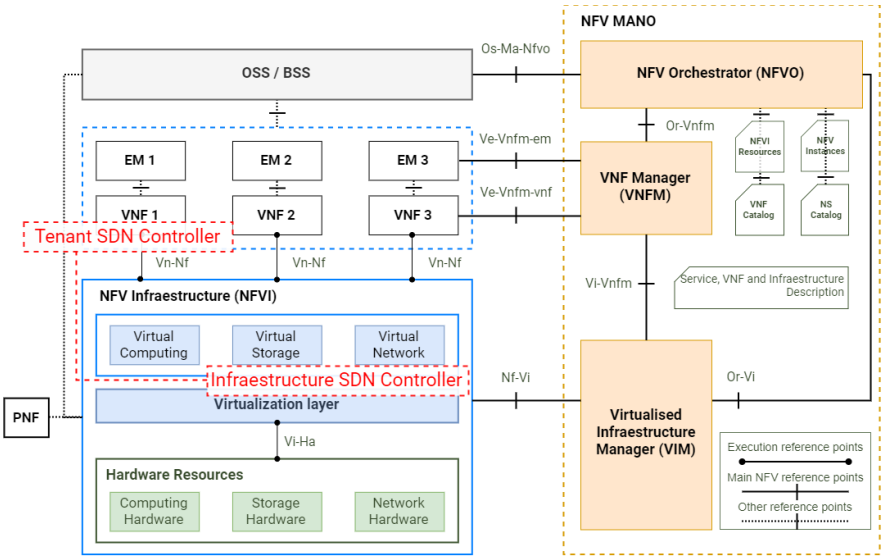

Figura 6: Arquitectura NFV definida por la ETSI con controladores SDN

Por otro lado, que el controlador esté unido con el VIM es compatible con la arquitectura SDN definida por el ONF para su uso con Network Slicing mostrada en la Figura 3. Además, es la que implementan multitud de plataformas de código abierto y estas se dedican al despliegue de redes SDN con arquitecturas NFV con controladores SDN como ONOS.

1.3 Tecnología Network slicing

La capacidad de poder multiplexar redes virtuales independientes y aisladas en una red física es la definición básica del Network Slicing. Desde el punto de vista teórico se entiende como la implementación de las arquitecturas NFV-SDN definidas previamente, ya que éstas son las que le dan soporte dentro de las comunicaciones 5G. Por otro lado, desde el punto de vista de negocio la infraestructura de una slice se define como un conjunto de todos los recursos que un operador de red móvil virtual (MNVO) administra. Con ello, un MNVO puede desplegar diferentes slices en función de la disponibilidad de estos recursos que proporciona el proveedor de la infraestructura. La Figura 7 muestra un ejemplo de cómo se estructuran las slices dentro de las comunicaciones 5G y como están soportadas por una infraestructura en común que sería donde se encuentran las tecnologías de NFV y SDN. Este tipo de modelo es el que se denomina Network Slice As A Service (NSaaS).

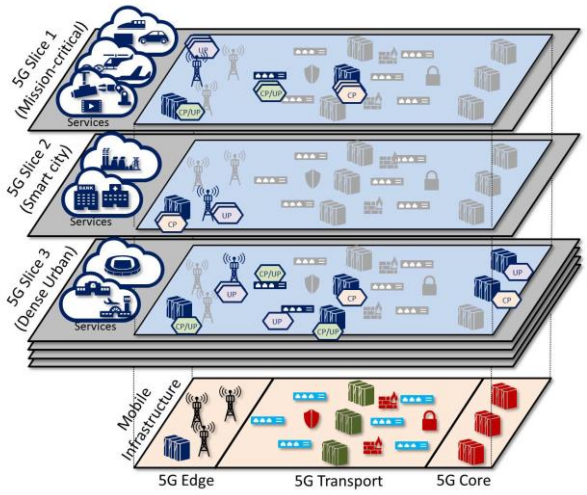

Figura 7: Diagrama de las slices en redes 5G creado por NGMN

La arquitectura ETSI NFV MANO permite la solución a la compleja arquitectura de las redes 5G la cual hace una administración óptima para la orquestación de múltiples recursos en diferentes capas. Es por esto que hay definidas diferentes estructuras de Networking Slicing como las de NGMN Alliance [32] o 5GPPP [33] que proponen diferentes arquitecturas, de las cuales se puede extraer el concepto en común como el que se muestra en la Figura 8, estando desplegadas en diferentes proyectos de código abierto. Esta arquitectura está formada por tres capas principales:

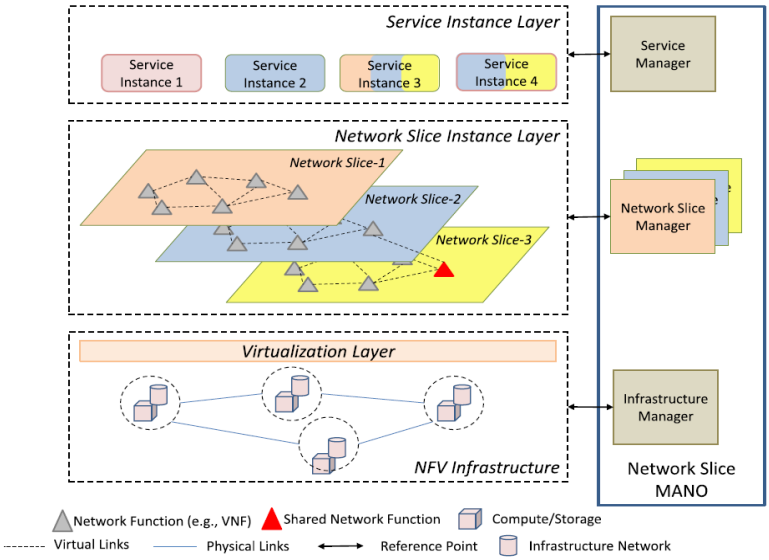

Figura 8: Arquitectura genérica de 5G Network Slicing [34]

- **Capa de servicio (Service layer)**, comparte los recursos físicos directamente como interfaz a las entidades de negocio, como por ejemplo operadores de móviles virtuales, dotando así de una visión amplia de los requisitos de los servicios. De esta forma cada servicio incorpora todas las características de red en requisitos de Service-Level Agreement (SLA), lo cual significa que se comprometen entre el proveedor de servicios y sus clientes a satisfacer la creación de una Slice adecuada a ese servicio.
- **Capa de funciones de red (Network function layer)**, en esta capa se lleva a cabo la creación de las slices que satisfagan los requisitos de la Service layer, estas slices se construyen mediante la concatenación de funciones de red desplegadas sobre la red virtualizada. Se gestiona tanto la configuración como el ciclo de vida de estas funciones de red y para optimizar su uso, se comparten las funciones de red con diferentes servicios. Por otro lado, es posible utilizar funciones de red dedicadas a un único servicio con lo que se obtendría un uso ineficiente de los recursos.
- **Capa de infraestructura (Infraestructure layer)**, esta capa representa la capa de recursos, la infraestructura física de red real, donde se encuentran la red de acceso de radio (RAN), la red central (CORE Network) y la red de transporte. Esta capa es la que proporciona los recursos de red para que sobre ella se desplieguen las funciones de red que componen las slices.

En comparación con la arquitectura NFV MANO mostrada en la Figura 4, el VIM, el VNFM y el NFVO tienen su equivalencia con el Administrador de Infraestructura, el Administrador de Network Slices y con la Capa de Servicio respectivamente en ese orden. Por otra parte, el Network Slice MANO es un sistema que se encarga de orquestar múltiples tareas complejas para asegurar el funcionamiento y controlar los servicios dentro de cada slice, y al mismo tiempo provee de otras funciones como administración de las imágenes, control de la política o la administración de la calidad de servicio (QoS). Además, con esta arquitectura el bloque de control, configuración y orquestación MANO administra el ciclo de vida de las slices

controlando los VNF, ya que estos son el bloque básico para construir una slice dentro de la capa de virtualización.

La Figura 9 de abajo nos muestra el concepto de NS, donde tenemos cada slice autocontenida y está implementada de forma E2E desde el CN al RAN, virtualizando las funciones de red con NFV y dando conectividad a cada slice mediante SDN [35]. De esta forma el procesamiento del plano de control es común a cada slice mientras que el procesamiento del plano de usuario está manejado en diferentes slices.

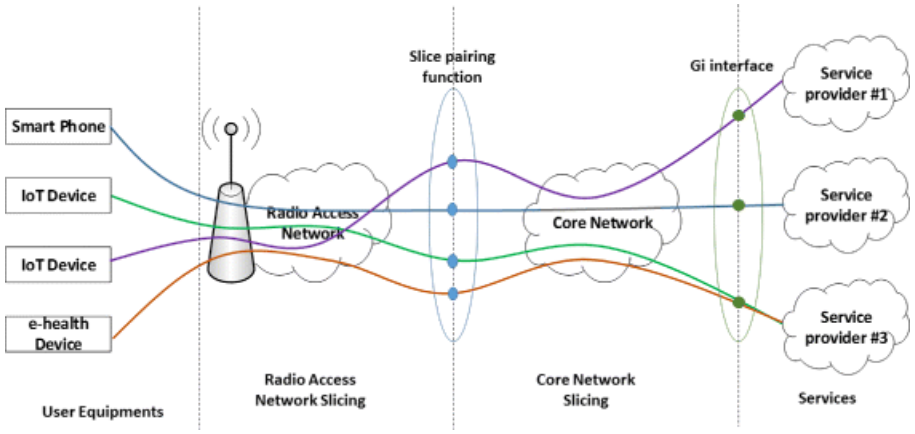

Figura 9: Diagrama de Network Slicing [36]

1.4 Herramientas de despliegue

Todos los proveedores de red explotan el hardware y software del que disponen para maximizar el rendimiento o la disponibilidad de la que requiere cada usuario que hace uso de sus centros de datos, a la par que manteniendo unos gastos operativos (OPEX) y capitales (CAPEX) bajos. Sin embargo, no es posible mantener un OPEX y CAPEX bajo, debido a que se necesitan expertos cada vez que se quieren agregar nuevas funciones a las redes, como balanceadores de carga, mejoras de seguridad, etc. Estas mejoras desembocan en la compra de nuevos equipos que hagan estas funciones de red, además de llevar la gestión y configuración del centro de datos. Por lo que todas estas tareas para cumplir las distintas necesidades de usuarios no son nada flexibles a la hora de la configuración o despliegue de estas funciones de red [37]. Es aquí donde entra el uso de NFV y SDN, ya que estas virtualizaciones de funciones de red consiguen resolver estos problemas no necesitando de nuevos equipos y expertos en instalación y configuración de hardware. Tal y como se explicó antes, el NFV tiene el potencial de incrementar la flexibilidad de la red y reducir a la vez el coste en comparación de los métodos tradicionales [38].

Por esta razón, tecnologías como NFV y SDN están siendo explotadas por plataformas de código abierto, las cuales en muchas de ellas se realizan aportaciones por parte de grandes compañías. Esto es debido a que hay un creciente interés en el desarrollo de todas estas tecnologías y que cada vez se amplíe más la comunidad que participa. Es por ello, por lo que

las plataformas de código abierto constituyen el pilar fundamental donde cualquier desarrollo toma partida para implementar funcionalidades de red virtualizadas, como la arquitectura de ONF la MANO NFV–SDN mostrada en la Figura 3 que da servicio a NS en comunicaciones 5G.

Gracias a estas tecnologías se puede desarrollar una infraestructura funcional de forma sencilla y flexible con la ayuda de plataformas de código abierto, creando servicios de computación en la nube como los que constituyen una Infrastructure as a Service (IaaS). En este tipo de servicios el proveedor tiene control total de las redes a bajo nivel virtualizándolas y pudiendo controlar todas las funcionalidades y servicios. Una IaaS proporciona a usuarios el acceso a recursos como almacenamiento, servidores y manejo de redes para su uso como computación en la nube, y es posible construir un sistema que proporcione este servicio, mediante plataformas de Open Source.

Existen diferentes plataformas del tipo Open Source, o servicios de código abierto, las cuales se caracterizan porque se pueden utilizar, modificar y redistribuir de manera altruista. Comenzaron su camino en los 90, cuando el padre del software libre, Richard Stallman, crea el primer proyecto con licencia GNU General Public License. Gracias a esto, en el 2010, la NASA y Rackspace Hosting comienzan un proyecto para promover el desarrollo de servicios de Cloud Computing ejecutándose en hardware estándar básico. Años después OpenStack Foundation administra el proyecto llamado OpenStack para promover su comunidad y su software basado en Python [39]. Como plataforma de código abierto, proporciona computación en la nube de una manera sencilla virtualizando los recursos y proporcionando una IaaS. Existen diferentes plataformas de código abierto como OpenMano [40], OpenBaton [41] u OPNFV [42] pero OpenStack es la que está mayormente respaldada, por multitud de grandes compañías, como IBM, Red Hat o Huawei como se puede ver en la de más abajo. Es por esto por lo que OpenStack se sitúa como el líder para las plataformas de tecnologías NFV. Además, la mayoría del resto de plataformas de código abierto hacen uso del SDK de OpenStack para la virtualización en el VIM.

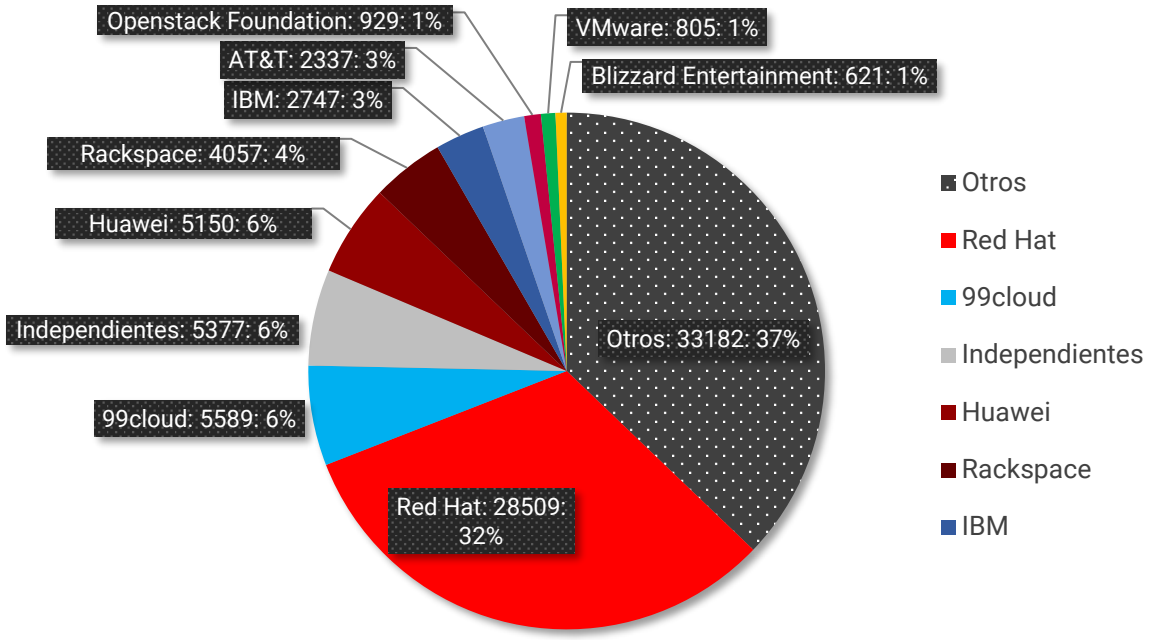

Figura 10: Gráfico de contribuciones a Openstack Rocky

1.5 OpenStack como solución de IaaS (Objetivos)

La plataforma de OpenStack está preparada para poder desplegarse de forma escalable desde un entorno de desarrollo en donde un único sistema forma parte del servicio hasta un rack de servidores, desplegados de tal manera que forme un entorno interconectado y funcional de forma sencilla y ágil. La gran mayoría de las plataformas existentes están preparadas para su uso en sistemas operativos basados en Linux, pero existen alternativas que utilizan herramientas de virtualización que son capaces de desplegar estas plataformas dentro de otros sistemas operativos como Windows. Un ejemplo es el proyecto V-MAGINE de Cloudbase, hacen uso del hipervisor Hyper-V de Windows que es capaz de desplegar un entorno de OpenStack con flexibilidad para su uso en múltiples sistemas de forma escalable [43].

Mediante el uso de estas plataformas abiertas se pretende llegar a desarrollar de forma sencilla y escalable desplegar un entorno que proporcione la capacidad de implementar una IaaS mediante el uso de tecnologías de SDN y NFV. Esta IaaS, podrá dar soporte al concepto de NS de 5G para adaptarse a las nuevas tecnologías de comunicación de redes y dar conectividad a los diferentes dispositivos que se van a desplegar. Por lo que el uso de esta plataforma conllevaría a una reducción del coste de CAPEX y OPEX ya que no se necesitarán equipos físicos al tener virtualizadas las funciones de red y poder multiplexar las múltiples redes virtualizadas.

OpenStack opera bajo la licencia Apache 2.0 y es conocida como "Linux of the Cloud". Aparte de los beneficios que tienen estas plataformas de la reducción de los costes, OpenStack tiene además otros beneficios ya mencionados anteriormente, de tal modo que el resumen es:

- **Gran comunidad y soporte**, Como ya hemos visto en cuenta con un soporte muy extenso de 89.303 reviews de la reléase Rocky [44], proveniente de grandes compañías.
- **Compatibilidad**, ya que si API se puede implementar en diferentes plataformas de Cloud.
- **Escalabilidad**, es de los puntos más fuertes ya que por definición básica OpenStack tiene la capacidad de poder virtualizar recursos de forma escalable para ofrecer servicio en función de lo que la demanda requiera.
- **Seguridad**, el acceso es basado en roles, esto quiere decir que el control y administración de los recursos dependerá del nivel diferentes de los usuarios, los roles asignados y podrán ser independientes entre servicios dentro de OpenStack.
- **Easy Frontend**, esto quiere decir que esta IaaS cuenta con un panel de control agradable visualmente, por lo que su usabilidad y control es muy sencilla a la hora de manejar las diferentes herramientas disponibles.

De esta forma OpenStack supone a priori una apuesta segura ya que los costes de implementación de estas plataformas de código abierto son escasas, siendo las principales únicamente la inversión de desarrollo e implementación de la plataforma. El mayor reto reside en poder construir una infraestructura de cloud computing que permita abordar todos los servicios necesarios. Mediante el uso de herramientas de automatización y realizando diseños en primera instancia para poder desplegar la IaaS y así abaratar los costes en equipos y manutención constantes. Y finalmente cuando toda la IaaS esté desplegada, un proveedor de servicios podría desplegar y administrar servicios complejos de manera sencilla y ágil, sin necesidad de tener que pedir asesoramiento a expertos ni tener que realizar inversiones en nuevos equipos y licencias de software [45].

2 Estado del Arte

Es más frecuente que plataformas y proyectos se realicen de forma Open Source, lo que genera un entorno de desarrollo en el que investigadores independientes pueden experimentar y retroalimentarse unos de otros, pudiendo generar mayor conocimiento en el desarrollo. Es por eso por lo que dentro del entorno de las arquitecturas NFV y SDN existen diferentes proyectos de código abierto que permiten implementar una arquitectura de forma automatizada para diferentes arquitecturas de software y múltiples nodos.

2.1 Open Source Mano

Conocido como OSM es un proyecto de gestión y orquestación MANO dentro de la ETSI bajo licencia pública Apache 2.0 y alineado con otros desarrollos que se hacen en las arquitecturas NFV. Fue lanzado en el World Mobile Congress del 2016, y comenzó con inversiones provenientes de Telefónica, Canonical, Intel RIFT.io entre otros. La meta de este proyecto es la del desarrollo de un orquestador de servicios de red E2E que sea capaz de modelar y automatizar los servicios TELCO en entornos complejos de negocio.

Se define como un Network Service Orchestrator (NSO), ya que administra los servicios de red otorgando la capacidad a la creación de estos y controlando todo su ciclo de vida mediante el uso de su Northbound API y el modelo de información. El modelo de información engloba a cualquier servicio y a las funciones de red ya sea del tipo de VNF, PNF o un híbrido (HNF). Define dos tipos de NaaS

- **Network Service**, es la pieza única que necesita para proveer a la red de un servicio, y su objeto se almacena dentro de las funciones de red interconectadas. Este servicio pasa por los estados de preparación, incorporación, creación del servicio, explotación del funcionamiento y finalización.
- **Network Slices**, puede proveerlas como servicio para su uso en casos de habilitar 5G. Estas Network Slices pueden estar compuestas de Network Slices Subnet con su propio ciclo de vida y contienen funciones de red. De tal forma que tiene 4 fases en su ciclo de vida, preparación, proceso de iniciarla después configurarla y activarla, inicio de ejecución y por último desmantelamiento, después deja de existir.

En ambos servicios el ciclo de vida es similar respectivamente en cada fase, siempre y cuando OSM controle y administre las slices y el NFVO ya que OSM administra las slices de forma diferente a como define el 3GPP. Este define una preparación intermedia adicional para garantizar un trazado de estas apropiado.

La arquitectura MANO mostrada más abajo en la Figura 11 es similar a la arquitectura NFV convencional, estando OSM en la parte de MANO, excepto el VIM. Puede controlar un módulo más el WAN Infraestructure Manager (WIM), el cual administra las redes definidas por software, y trabaja dentro del dominio de transporte. Este se encarga de dar conectividad mediante la API para administrar las funciones disponibles del SDN Controller (SDNC) y la

del transporte Controllers for Software-Defined Transport Network (SDTN). Soporta diferentes protocolos conocidos de operaciones SDN como OpenFlow, OVSDB o también tradicionales como NETCONF/YANG [46].

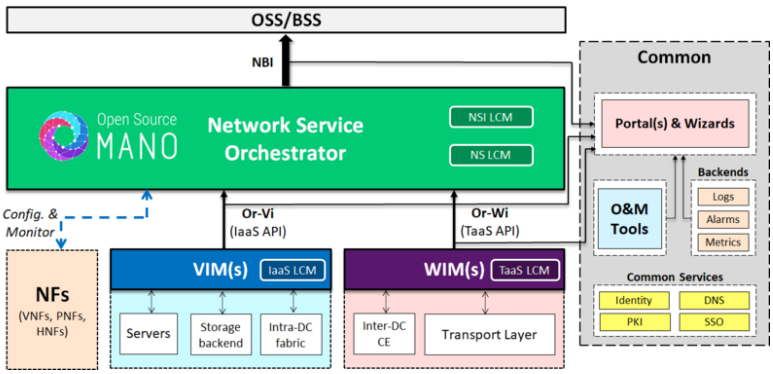

Figura 11: Arquitectura de OSM

OSM es un proyecto que abarca el módulo MANO dentro de la infraestructura ETSI NFV junto con SDN, y además cuenta con características para su uso en 5G y Network Slicing. Además, su arquitectura es simple y directa, y cuenta con una documentación extensa, artículos y también actualiza y organiza conferencias para tratar las nuevas versiones. Sin embargo, la implementación completa de la arquitectura depende de otros módulos y proyectos adheridos, como el módulo VIM. Un ejemplo fue el realizado por Intel en el Stand de Intel dentro del Mobile World Congress del 2016. Este hizo la implementación de OSM junto con módulos del proyecto OpenMANO de Telefónica, Juju generic VNF Manager de Canonical y el orquestador de Rift.io. De tal forma que se montó una configuración multi-VIM soportando múltiples instancias VIM, una OpenStack y la otra era uno de los módulos del proyecto OpenMANO, el OpenVIM [47]. Por lo que el uso de esta plataforma es clave en un entorno de producción, pero en entornos de desarrollo implica la investigación de diferentes proyectos y plataformas para poder desplegar una infraestructura ETSI NFV-SDN completa.

2.2 Open Networking Automation Platform

Conocido como ONAP es un proyecto de código abierto nace en 2017 de la unión de los proyectos OpenECOMP y Open-O y es organizado por Linux Foundation. En enero de 2018 entra como proyecto dentro de del LF Networking Fund, organización fundada por Linux Foundation para abarcar los proyectos de código abierto de proyectos de redes que tengan tecnologías como NFV y SDN [48]. Actualmente está en el quinto lanzamiento, El Alto, en este se han centrado en mejorar aspectos de seguridad, escalabilidad, rendimiento, ampliar su documentación o mejorar la estabilidad de los lanzamientos predecesores.

ONAP está presente en múltiples servicios y dentro de la arquitectura ETSI NFV MANO incluye los módulos que se muestran en la Figura 12. Se puede observar que aborda la administración de los servicios de red, su ciclo de vida o gestión de gráficos de reenvío de

VNFs (NFVO). También la gestión de fallos y gestión de las imágenes (VNFM) además de control de FCAPS o diseño de la arquitectura.

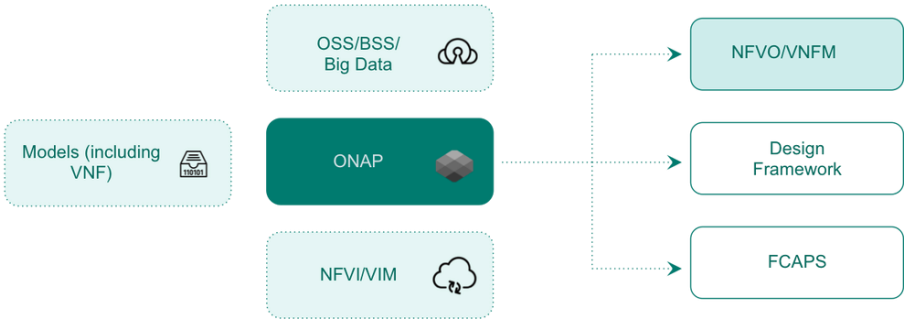

Figura 12: Alcance de ONAP en la ETSI NFV MANO [49]

Pero, por otro lado, la arquitectura general de ONAP, la cual se puede ver más en detalle en la Figura 13 se divide en dos partes [50]

- Marco de diseño, el cual consta de un entorno donde hay herramientas, repositorios y técnicas para el desarrollo de toda recursos, servicios o productos que se necesiten, generando así nuevas capacidades.
- Marco de ejecución, es un entorno donde se ejecutan reglas y políticas creadas en el entorno de diseño. Los principales son:
  - Biblioteca disponible con recursos y servicios.
  - Controlador que administra el estado de un recurso simple o único como un VNF o la red.
  - Colección de datos, análisis y eventos para su uso en la obtención del rendimiento y los datos de configuración.
  - Orquestador del servicio que automatiza las instanciaciones o migraciones de los VNF.
  - Marco de seguridad, el cual está basado en la plataforma anterior ECOMP.

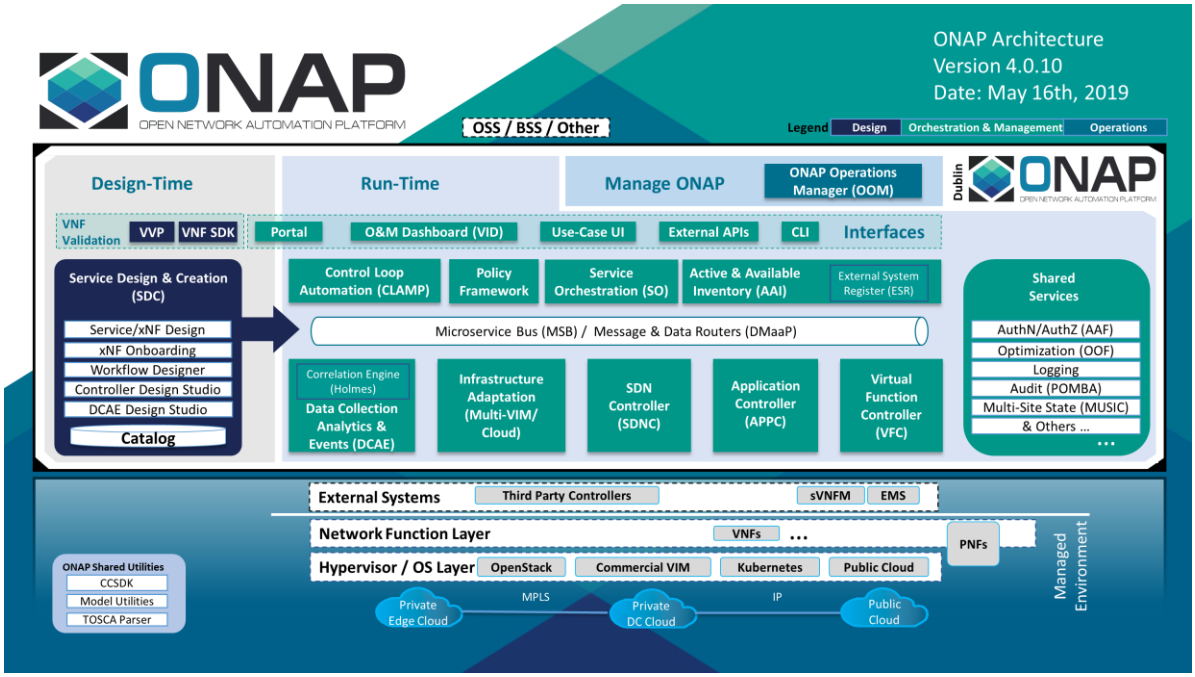

Figura 13: Arquitectura de la plataforma ONAP [51]

ONAP cuenta con una estructura que avanza en cada lanzamiento y soluciona fallos de implementación, además está involucrado en la parte de proveedores de servicio, participando en proyectos de VoLTE, CCVPN en combinación con SOTN que son tecnologías de comunicación vía óptica o en BroadBand Service donde ONAP contribuye a reducir costes operacionales y procesos IT complejos. Aparte, proporciona una documentación para su implementación mediante el uso de Heat un componente de OpenStack, por lo que el uso de los diferentes módulos de ONAP se realiza mediante la instalación de OpenStack.

2.3 Open Platform for NFV

OPNFV es una plataforma que cuenta con una comunidad y proyectos muy extensos que dan solución al desarrollo, implementación y evolución de los componentes NFV en diferentes ecosistemas de código abierto. En esencia OPNFV no es un proyecto que desarrolla una arquitectura ETSI NFV, sino una comunidad donde abarca proyectos de diferentes compañías, todas con licencias públicas, que investigan en el desarrollo e implementación de los diferentes módulos para su uso en NFV y SDN. Por ejemplo, la comunidad desarrolla y prueba la implementación de módulos como el VIM (OpenStack, Kubernetes), controladores SDN, o sistemas MANO. Esta plataforma fue creada por Linux Foundation en 2014 y más tarde desde 2018 está siendo administrada por LF Networking Fund además de trabajar en conjunto con la ETSI para realizar estandarizaciones de forma consistente. Esta plataforma se centra principalmente en el desarrollo del módulo VIM integrando múltiples componentes de diferentes proyectos como OpenStack, OVN, OpenDaylight, Kubernetes, Ceph Storage, KVM o OpenvSwitch. En 2019 lanzo su última versión llamada Hunter 8.1.

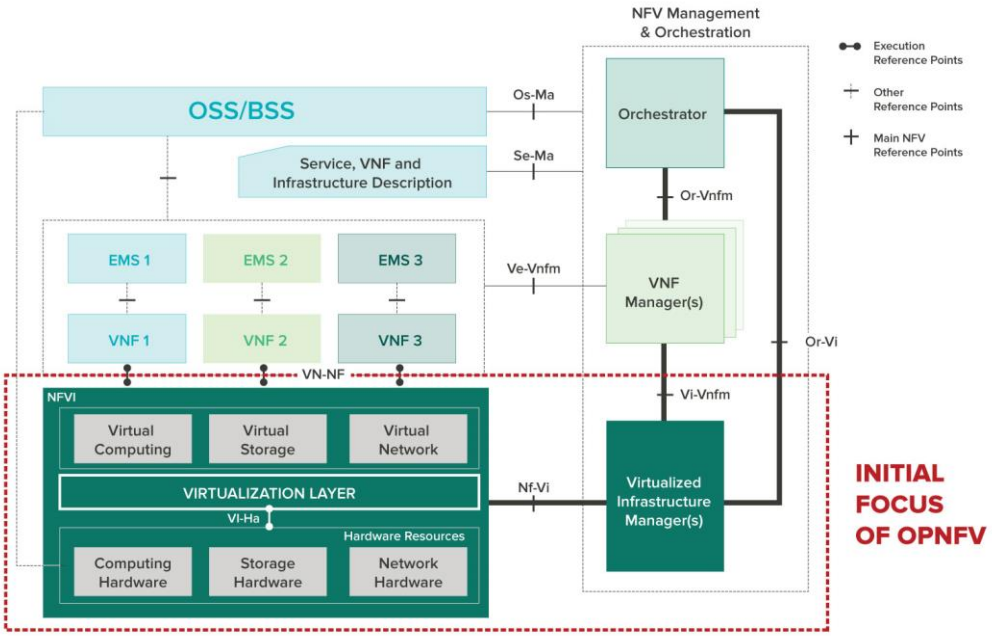

Figura 14: Alcance de OPNFV de la arquitectura ETSI NFV [52]

Ofrece una amplia documentación en la cual te guía para montar una infraestructura similar a la ETSI NFV, además ofrece diferentes herramientas para la instalación y configuración, pero utiliza diferentes plataformas por lo que lo hace un despliegue complejo. OPNFV es una comunidad la cual permite a empresas y desarrolladores la investigación de construir arquitecturas modulares de diferentes proyectos para probar y promover el uso de estas plataformas de código abierto. Sin embargo, es complicado definir un estándar de arquitectura NFV, ya que la propia arquitectura de OPNFV se basa en el desarrollo continuo, es decir más enfocado a dar soporte de múltiples proyectos y no abarca el desarrollo completo de la arquitectura ETSI NFV. Como se puede ver en la Figura 14 el enfoque inicial de esta plataforma fue la de centrarse en el VIM y en el NFVI, junto con la implementación de otros proyectos como OSM, Juju, OpenStack Tacker u ONAP para completar el resto de módulos de la arquitectura ETSI NFV y OpenDaylight u ONOS como controladores SDN [53].

2.4 Open Baton

Open Baton es lanzado como proyecto de código abierto por la Universidad Técnica de Berlín y por el instituto Fraunhofer FOKUS para satisfacer servicios de redes orquestados mediante una arquitectura ETSI MANO a lo largo de múltiples puntos de presencia. Como resultado en la versión actual es una arquitectura NFV extensible y configurable la cual se puede administrar mediante diferentes herramientas y lenguajes. Se puede realizar instalación de forma automatizada mediante el uso de Docker, y ofrece una documentación no muy extensa, pero clara sobre cómo es su estructura de servicios. La Figura 15 muestra la arquitectura que forma y los servicios que tiene, como se puede ver implementa completamente la arquitectura ETSI MANO. No cuenta con una comunidad muy extensa

como otras plataformas, pero es una plataforma sencilla y fácil de implementar la cual sirve tanto en entornos de desarrollo como de producción.

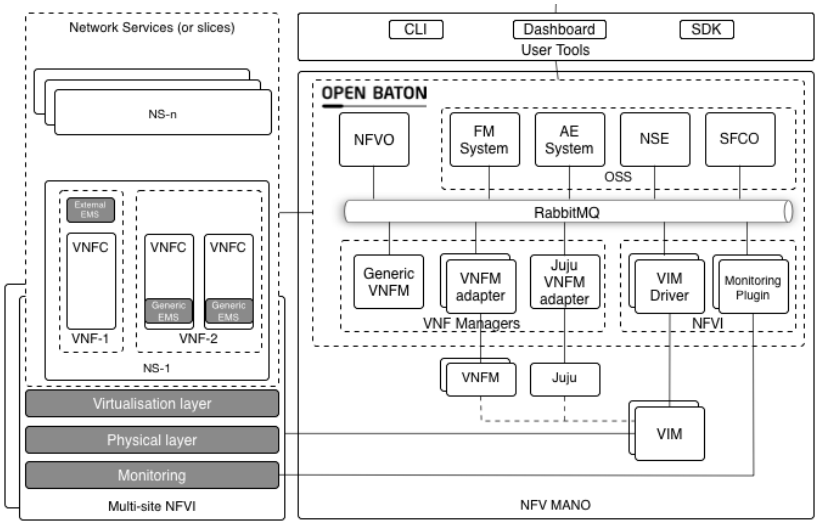

Figura 15: Arquitectura de Open Baton [54]

OpenStack es una plataforma la cual está presente en las otras plataformas mencionadas, ya sea de forma parcial, utilizando alguna API, o haciendo uso completo de alguno de los componentes que forman parte de OpenStack. Nace de la necesidad de reestructurar un código de servidores en la nube de Rackspace, y otro proyecto de la NASA, así que se unieron para comenzar un proyecto. Este se presentó en 2010 en el primer Design Summit a más de 25 compañías y más tarde se presenta la misión de OpenStack como "producir la omnipresente plataforma de computación en la nube de código abierto que satisfaga las necesidades de las nubes públicas y privadas, independientemente de su tamaño, siendo simple de implementar y escalable de forma masiva". Tras el primer lanzamiento llamado Austin en 2010, años más tarde en 2012 se crea el OpenStack Foundation para promover y empoderar el software de OpenStack y su comunidad.

A finales de 2016 salieron dos nuevas versiones llamadas Mitaka y Newton, las cuales incluía un API de un componente llamado Tacker, junto con una interfaz visual web que era una extensión del componente Horizon, denominado Tacker-Horizon. Este componente se construye para dar soporte a la tecnología NFV, de modo que implementaba un VNFM y un NFVO. La arquitectura de Tacker basada en la ETSI MANO para proveer servicios de red o VNFs y todos desplegados en una NFVI como es OpenStack en conjunto. De esta forma se puede construir un orquestador de servicios de red E2E usando los catálogos de VNF que proporciona mediante el uso de plantillas TOSCA. Por otro lado, cuenta con un proyecto llamado Devstack que le permite realizar una instalación automatizada de la plataforma OpenStack mediante un archivo de configuración.

Con la arquitectura NFV desplegada como la que muestra la Figura 16 se podría hacer uso de OpenStack como IaaS para poder otorgar servicios con características similares a las del Network Slicing. Pero además OpenStack permite la flexibilidad de sustituir el componente de Neutron, que administra las redes para incluir un controlador SDN como OpenDaylight. De esta forma se implementaría una arquitectura completamente funcional con todos los módulos y haciendo uso de las tecnologías NFV y SDN.

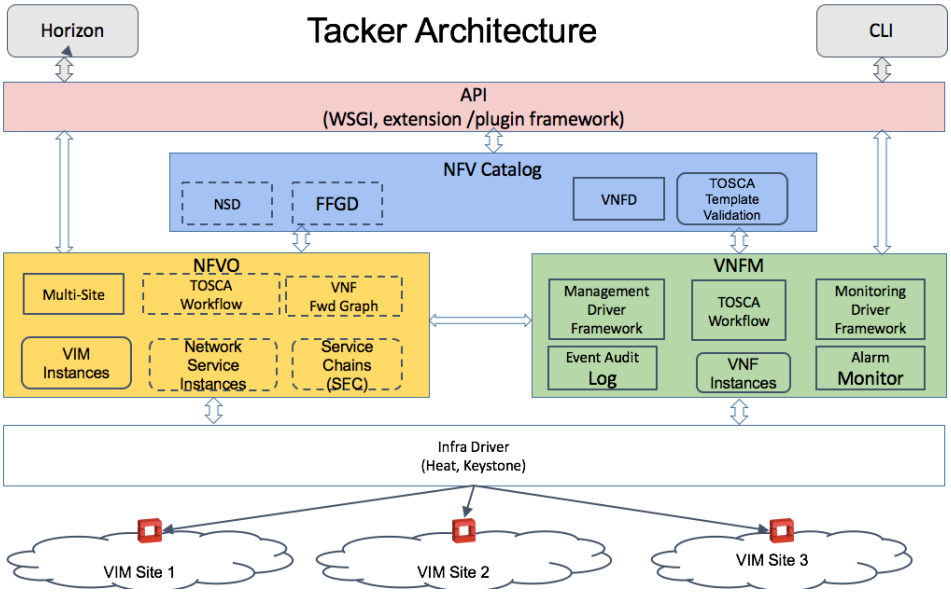

Figura 16: Arquitectura del componente Tacker de OpenStack

2.5 Conclusión

Existen diferentes plataformas que abordan la implementación de la arquitectura ETSI MANO NFV junto con SDN, pero estas además ofrecen otro tipo de servicios y herramientas que son beneficiosos dentro de entornos de producción y en el ámbito del 5G, como la administración de Network Slicing y servicios de red. Pero la mayoría se desarrollan en conjunto de tal forma que tienen una estructura modular, lo que significa que cada plataforma son piezas que pueden encajar las unas con las otras, creando una estructura compleja y funcional a partir de partes más simples. Pero para llegar a un desarrollo final, hace falta investigación dentro de cada plataforma, ya que implementarlas requiere de conocimiento específico de cada una. Por ello las herramientas que ofrecen para su implementación y conexión entre ellas es esencial. De esta forma podemos resumir las características de los proyectos de cada plataforma y herramientas que hacen que se pueda desplegar una arquitectura ETSI NFV funcional para poder implementar una IaaS de forma sencilla y ágil.

Plataformas	*OSM*	*ONAP*	*OPNFV*	*Open Baton*	*OpenStack (Tacker + ODL)*
Open Source					
Comunidad					
Documentación					
Automatización					
Modularidad					
ETSI MANO					
Multi-Site VIM					
Monitorización y FCAPS					

Tabla 2: Comparación de plataformas NFV

Tal y como podemos ver tenemos diferentes características para comparar las plataformas, separándolas en principales características más relevantes y en otras más secundarias, pero es igual de necesario analizarlas. El significado de cada una de ellas es:

- **Open Source**, si la plataforma es de código abierto.
- **Comunidad**, como de amplia es su comunidad, ya sean por desarrolladores independientes, como por inversiones de grandes compañías. Es importante porque la detección de fallos y mejoras en el rendimiento es gracias a los aportes de la comunidad.
- **Documentación** de la que está dotada la plataforma, es decir, una forma de medir la organización de esta ya sea por workshops, eventos, artículos o wiki donde se detalla la arquitectura y el desarrollo.
- **Automatización**, se refiere a si provee de herramientas para la implementación de su arquitectura de forma sencilla mediante una configuración inicial y scripts o compatibilidad para uso de software externo de automatización.
- **Modularidad**, trata el concepto de poder desplegar la arquitectura ETSI MANO utilizando diferentes plataformas o software en conjunto.
- **Solución ETSI MANO**, en parte es similar a la anterior ya que hace referencia a si la plataforma cubre todos los módulos de la arquitectura ETSI MANO NFV+SDN de forma completa, o necesita de otros componentes para poder implementarla.
- **Multi-Site VIM**, es una característica que permite la adopción de diferentes módulos VIM operando dentro de la misma infraestructura. Es una necesidad ya que los operadores de red necesitan desplegar específicas VNFs cerca de sus clientes para incrementar el rendimiento. Por ello se necesita una selección de múltiples VIM de forma óptima, y esto se realiza mediante la colocación de los diferentes VNFs en el mejor VIM que otorgue el SLA conveniente a ese cliente, como una ubicación cercana geográficamente [55]. Es posible ya que cada VIM podría estar ubicado en diferentes localizaciones, o tener otras configuraciones.

- **Monitorización y FCAPS**, es importante ya que el uso de las FCAPS permite administrar y controlar las incidencias de fallos, configuración, contabilidad, rendimiento y seguridad.

En definitiva, el uso de estas plataformas habilita la implementación de la arquitectura ETSI MANO, y todas las plataformas son capaces de interactuar las unas con las otras de forma modular, otorgando gran flexibilidad a la hora de realizar investigaciones y utilizar diferente software en función de las necesidades o del alcance que se necesite.

Destaca la implementación de OpenStack junto con OpenDaylight, ya que abarca totalmente la arquitectura para implementar NFV y SDN y permite su instalación de manera automatizada pudiendo escalar la instalación. Además, cuenta con multitud de soporte por parte de la comunidad y hay mucha documentación en cuanto a artículos, y realizan diversos eventos donde tratan los proyectos que trabajan con componentes de la plataforma de OpenStack.

3 Implementación de una IaaS

Tal y como se ha visto, para poder construir una infraestructura que proporcione la capacidad de virtualizar servicios, programar las redes y probar nuevas tecnologías de 5G, es necesario el uso de plataformas que implementen las tecnologías de NFV y SDN definidas anteriormente abaratando el coste frente a redes tradicionales. Además, es necesario el uso de herramientas de automatización para implementar estas plataformas, ya que poseen una arquitectura de servicios compleja y con requisitos muy específicos. De modo que vamos a analizar e investigar cómo está construida la plataforma de OpenStack junto con OpenDaylight, para poder entender la configuración que satisfaga las necesidades de una arquitectura NFV–SDN.

3.1 OpenStack

Como resultado del análisis de las plataformas existentes que proporcionan una IaaS, se observa en el resumen de la Tabla 2, que destaca OpenStack, ya que cumple con los requisitos para poder utilizar Network Slicing y ofrece facilidad de desarrollo y modularidad para poder configurar la arquitectura de servicios que tiene disponible, ya sea con sus componentes o agregando otros proyectos compatibles. Esta plataforma dispone de diferentes componentes [56] donde los principales se encargan de,

- **Heat**, se encarga de la orquestación de recursos y sus dependencias a través de plantillas Heat Orchestration Template (HOT), que son plantillas de tipo AWS CloudFormation.
- **Nova**, es el administrador de las máquinas virtuales y se encarga de iniciar, parar, suspender o reiniciar las instancias de las imágenes que haya almacenadas. Según la utilidad o servicio se divide en Nova-api, Nova-compute, Nova-schedule, Nova-conductor, Nova-consoleauth, Nova-cert y la Base de datos de los estados.
- **Neutron**, responsable de la gestión de redes de la topología de OpenStack. Además, realiza la conectividad entre las instancias e interfaces creadas por Nova y mediante el uso de DHCP se le asigna una IP.
- **Cinder y Swift**, son los componentes de almacenamiento de OpenStack. Cinder se encarga del almacenamiento de volúmenes de instancias en ejecución. Y Swift almacena objetos como cuentas, contenedores u objetos. Donde una Cuenta se refiere a un proyecto o usuario, un Contenedor se refiere al espacio de nombres de objetos y un Objeto es cualquier información o archivo con un identificador único.
- **Keystone**, es el encargado de la autenticación del resto de componentes, usuarios o proyectos. Ya sea gestión de roles o usuarios se encarga de autenticar los accesos y de si tienen permiso el uso de los componentes mediante peticiones API. Es necesario que un usuario tenga un rol que defina los privilegios asignado a un tenant, donde este es el concepto que engloba a un usuario o conjunto de estos que comparte acceso a una instancia.
- **Placement**, proporciona una API HTTP que rastrea los inventarios y el uso de recursos en la nube para ayudar a otros servicios a administrar y asignar sus recursos de manera efectiva.

- **Glance**, es el componente que proporciona la biblioteca de imágenes almacenadas para que después Nova pueda crear las instancias. Puede almacenar snapshots de instancias Swift además de las propias imágenes y metadatos
- **Horizon**, es la interfaz visual web de OpenStack, desde este se puede administrar y configurar el resto de los componentes de forma sencilla y cómoda.

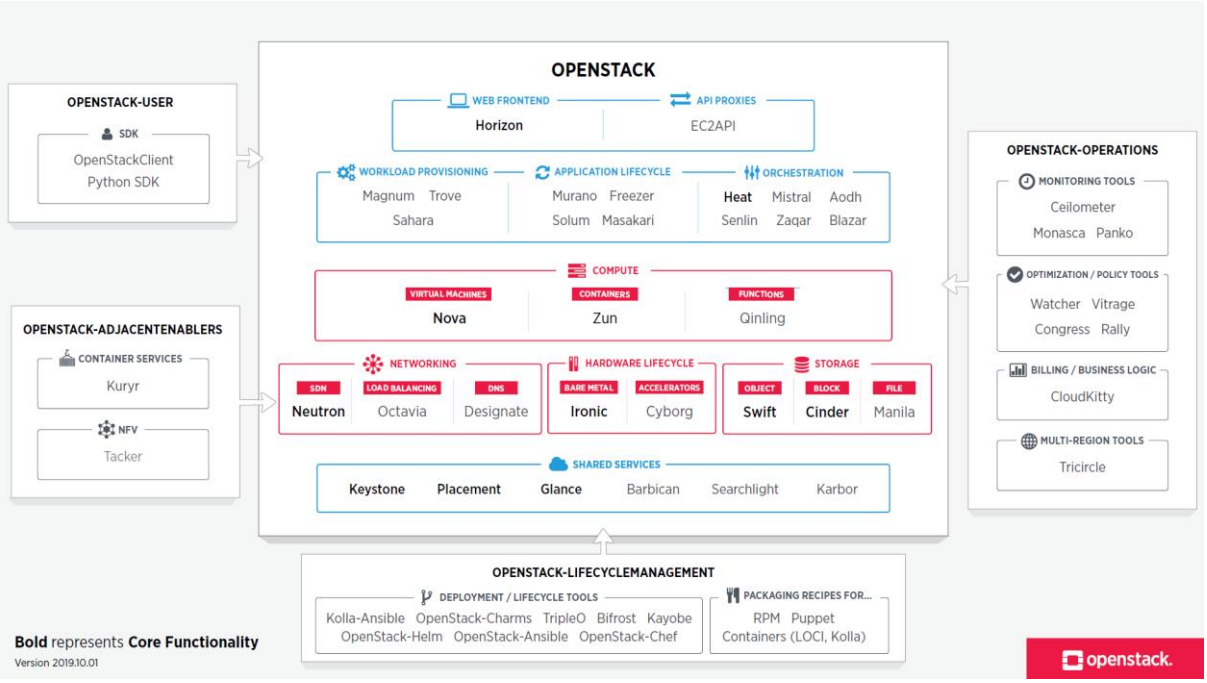

Figura 17: Diagrama de todos los componentes de OpenStack Rocky

3.1.1 OpenStack Tacker

Como se ha mencionado previamente, OpenStack cuenta con un componente llamado Tacker el cual proporciona una arquitectura para implementar NFV. La Figura 16 muestra cómo se estructura Tacker en un alto nivel donde se visualizan tres módulos principales, el Catálogo VNF, el NFVO y el VNFM. El Catálogo VNF, donde se encuentran los Descriptores VNF (VNFD), Descriptores de servicios de red (NSD) y los VNF Forwarding Graph Descriptors (VNFFGD). El NFVO, realiza la orquestación de los VNF y los conecta haciendo uso de los VNFFGD. Es capaz de orquestar diferentes VNFs a lo largo de múltiples VIM y múltiples sitios de presencia (POPs). Por último, el VNFM, administra el ciclo de vida de los VNF que se encuentran dentro de los VIM, facilitan el escalado de los recursos que tienen asociados los VNF basados en políticas.

Se hace uso del lenguaje OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) para la definición de metadatos VNF y de Glance para almacenar y administrar las imágenes VNF. Estas plantillas codificadas en YAML Ain't a Markup Language (YAML) son leídas por el TOSCA parser (Heat), y define propiedades como, las redes, imagen, escala y monitorización de un VNF. Además, se pueden definir múltiples

Virtual Deployment Units (VDUs) para describir instancias en una única plantilla. Una vez cargadas en OpenStack se traducen a plantillas HOT para que el componente de Heat pueda entenderlas y procesarlas. Por último, Tacker recibe estas plantillas al NFVO y al VNFM para procesar el ciclo de las VNFD, NSD y VNFFG definidas.

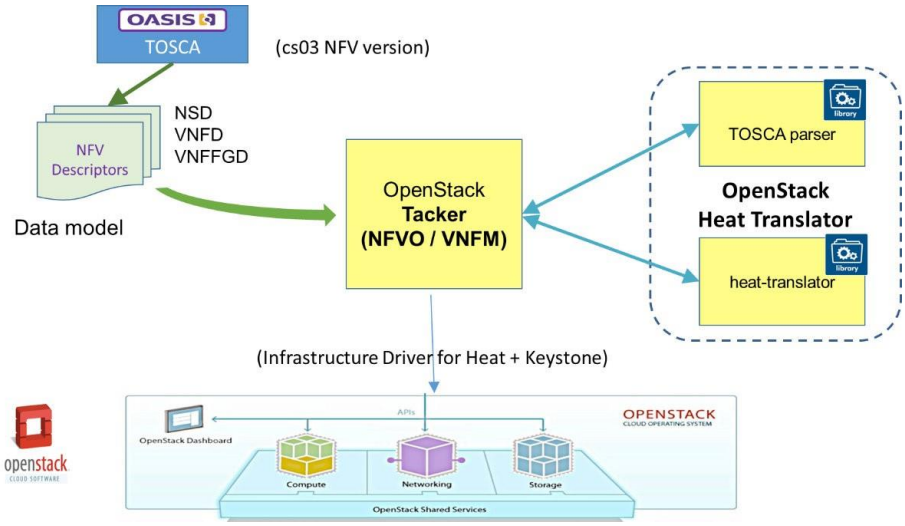

Figura 18: Modelado de VNFs dentro de Tacker [57]

Una vez traducidas y enviadas al NFVO, el componente Mistral, convierte las plantillas a un flujo de trabajo. Mistral es una parte integral de Tacker el cual puede monitorear el flujo de los VNF y los VIM y puede describir un conjunto de tareas y relaciones entre estas para controlar la ejecución, sincronización y el estado de administración. Por otro lado, Ceilometer es un componente de OpenStack el cual colecciona datos, los normaliza y los transforma a través de todos los componentes para proveer de capacidad de seguimiento de los recursos. Este componente se utiliza en Tacker para controlar los VNF y proporcionar control de fallos y rendimiento ya que el monitoreo es un aspecto importante en la arquitectura ETSI MANO. Por último, el componente de Barbican diseñado para la seguridad de almacenamiento, aprovisionamiento y administración de claves. Tacker hace uso de este componente para registrar credenciales asociadas a un VIM que son usadas en el NFVO y el VNFM para operar los recursos disponibles en el NFVI. Las credenciales pueden ser un usuario, contraseña e información del proyecto. Adicionalmente se utiliza el componente Aodh, es un proyecto que permite describir reglas que se activan en función de los datos recibidos por Ceilometer, de esta forma se genera un servicio de alarmas y eventos.

3.1.2 OpenStack Service Function Chaining

A la hora de implementar Tacker es necesario hacer uso de OpenStack SFC, el cual es un proyecto que otorga una API para dar soporte a Service Funtion Chaining (SFC) en Neutron. Se necesita debido a que dadas unas VNFs implementadas en diferentes máquinas virtuales, el flujo de tráfico debe fluir a través de las máquinas virtuales en cada salto de la cadena, pero actualmente la red utiliza algoritmos de hash para distribuir diferentes flujos a través de las

diferentes máquinas en cada salto. Por ello si se quiere ir de un punto A hacia otro B a través de otro punto C, pero en realidad el punto C no está entre los otros dos desde el punto de vista de la tabla de enrutamiento, no podría fluir el tráfico. SFC es un mecanismo para anular el reenvío basado en el destino básico típico de las redes IP y es considerado una tecnología SDN, aunque está relacionado con el enrutamiento basado en políticas en redes físicas o policy-based routing (PBR) [58].

El uso de este servicio hace necesario la implementación del Network Service Headers (NSH) ya que son un protocolo de encapsulación de Service Chaining fundamental para poder hacer uso de SFC. El NSH define un nuevo protocolo del plano de datos, ya que está diseñado para encapsular el paquete original con una cabecera de encapsulamiento de transporte [59]. Este está compuesto por tres elementos, la identificación del Service Function Path (SFP), la localización dentro de del SFP y metadatos del paquete. El SFP consta de un conjunto de puertos que definen la SFC y un conjunto de clasificadores de flujo que especifican los flujos de tráfico clasificados que ingresan a la cadena. Otros conceptos son el Service Function (SF) y el Service Function Forwarder (SFF) donde un SF es un servicio que forma parte de un dominio disponible para consumo y el SFF es el responsable de reenviar paquetes a uno o más SF utilizando la información transmitida en la trama encapsulada SFC.

Networking-sfc

Dentro de OpenStack tal y como se muestra en la Figura 19 el Port Chain Plugin de SFC puede estar respaldado por diferentes proveedores de servicios, ya sea por Controladores OVS o SDN. Estos controladores pueden proporcionar diferentes implementaciones para la representación del SFP controlados mediante el Common Driver API.

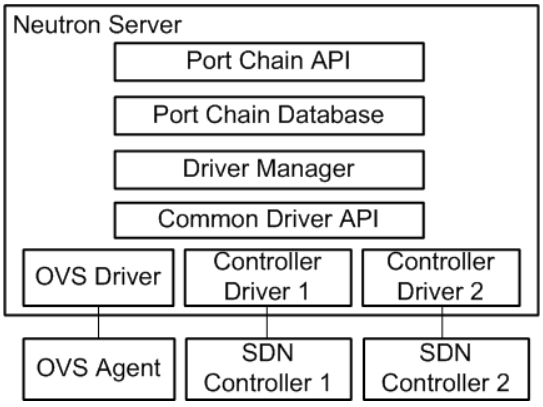

Figura 19: Arquitectura SFC dentro de OpenStack

De esta forma el flujo de Networking SFC se compone de un clasificador ingress de entrada y otro egress de salida, que se encargan de clasificar los paquetes que reciben en función de políticas definidas, ya sea por el puerto de destino o IP. Los paquetes son clasificados en las diferentes cadenas y redirigidos al correspondiente SF mediante el SFF. Cuando se determina

que SFP va en el paquete se devuelve de nuevo al SFF que determina si la cadena se termina y se elimina la encapsulación de SFC para entregar finalmente le paquete al clasificador egress que entrega el paquete al destino.

En cuanto al uso con Tacker es esencial para dar conectividad, ya que SFC crea una lista ordenada de que tráfico pasa por cada VNF y los clasificadores deciden en por donde debería fluir el tráfico. Los VNFFG definidos en las plantillas TOSCA son traducidos en SFCs y clasificadores a partir de los VNFFGD.

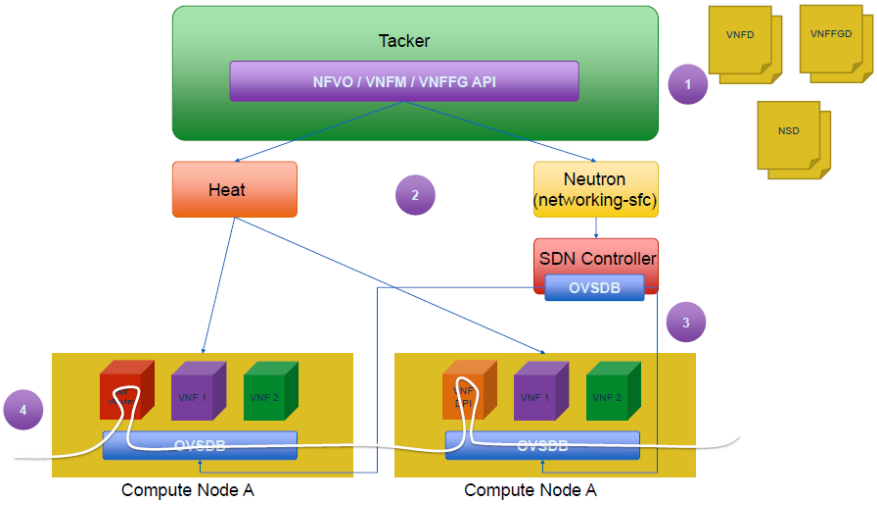

Figura 20: Flujo de SFC en Tacker [60]

3.1.3 Componentes OpenStack en la ETSI MANO

Definidos todos los componentes de los que hace uso Tacker para construir la arquitectura ETSI MANO, podemos resumir en la Tabla 3 el uso de los componentes para cada módulo de la arquitectura.

<table>
<tr><th colspan="6">Componentes OpenStack</th><th>Módulos relacionados</th></tr>
<tr><td rowspan="3">Tacker</td><td rowspan="3">Heat</td><td rowspan="3">Mistral</td><td rowspan="3">Ceilometer</td><td rowspan="3">Barbican</td><td rowspan="3">Aodh</td><td>NFVO</td></tr>
<tr><td>VNF Catalog</td></tr>
<tr><td>VNFM</td></tr>
<tr><td colspan="6">Nova</td><td rowspan="6">VIM / NFVI</td></tr>
<tr><td colspan="6">Neutron</td></tr>
<tr><td colspan="6">Glance</td></tr>
<tr><td colspan="6">Placement</td></tr>
<tr><td colspan="6">Cinder</td></tr>
<tr><td colspan="6">Keystone</td></tr>
</table>

Tabla 3: Componentes OpenStack asociados a los módulos de la arquitectura ETSI MANO

3.2 OpenDaylight

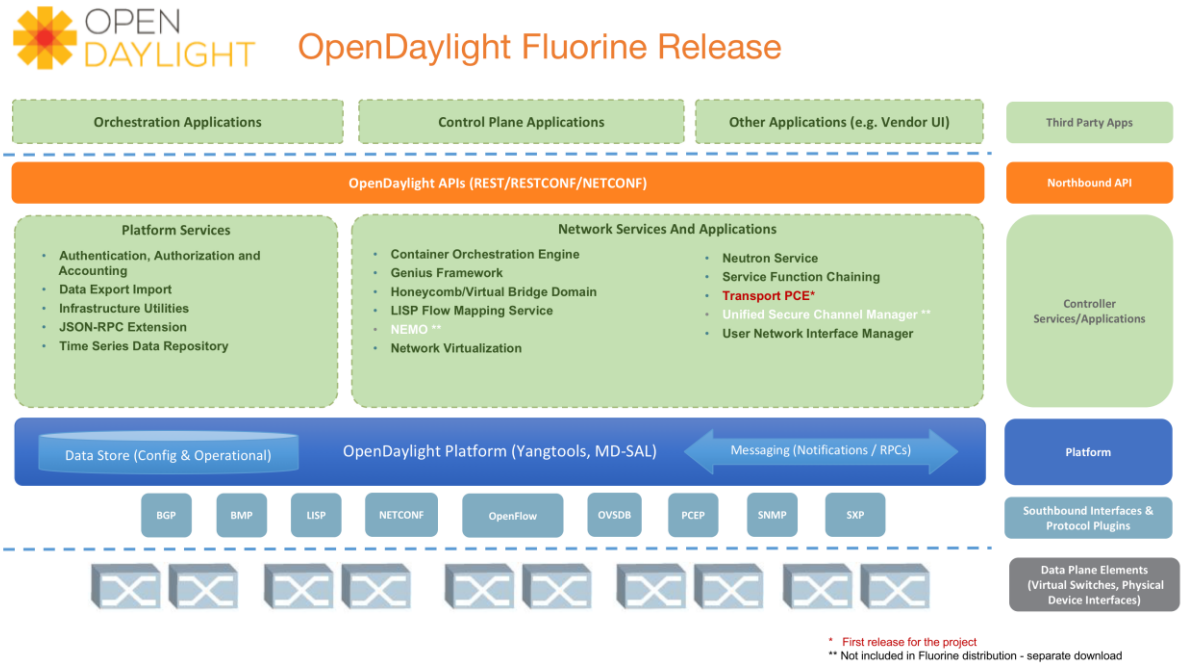

Figura 21: Arquitectura de ODL Oxygen

OpenDaylight (ODL), es una plataforma modular para implementar un controlador multiprotocolo de SDN. Linux Foundation anunció en 2013 el proyecto OpenDaylight como solución a código abierto para acelerar la innovación sobre la tecnología SDN y NFV, en la cual se incluyeron múltiples inversores como Cisco, IBM o HP. Desde 2014 que salió la primera versión Hydrogen ha habido muchas mejoras hasta la versión actual, Sodium, que se aplican a NFV. La arquitectura mostrada en la Figura 21 es modular ya que se pueden utilizar los componentes que la forman de forma separada, instalando los plugin que se necesiten. Su estructura se separa en tres capas principales, que son aplicaciones de red, orquestación y servicios, plataforma de controlador y el plano de datos. Estas están separadas por dos interfaces de comunicación, el Northbound y el Southbound, similar a la arquitectura SDN mostrada en la Figura 2.

3.2.1 Arquitectura de ODL

Aplicaciones de orquestación

La capa que se encuentra más arriba, la componen diferentes aplicaciones de negocio y de orquestación que se utilizan para administrar el controlador SDN desde la aplicación del panel de control. Esta capa se comunica con el controlador mediante el uso de una Northbound API, la cual en el caso de ODL, lo realiza mediante el protocolo RESTCONF o de NETCONF.

En esta capa además se encuentra una aplicación llamada DLUX, esta es una interfaz visual que facilita y simplifica el desarrollo y las pruebas renderizando una interfaz de usuario basado en los modelos Yet Another Next Generation (YANG) cargados en ODL. Muestra diferentes pantallas en las que están los módulos,

- **Topology** que representa una abstracción gráfica de la topología de red creada.
- **Nodes**, muestra una tabla con los nodos descubiertos y el tráfico cursado sobre estos.
- **YANG UI**, Carga una interfaz donde muestra información del modelo YANG cargado en ODL y permite interacción a través de una API.
- **Yang Visualizer**, hay representado una topología ramificada con todos los nodos de la red.

Dlux para obtener información de la red y proporcionar administración de esta, hace uso de los servicios de MD-SAL.

Plataforma de controlador

Compuesta de servicios de plataforma y aplicaciones y servicios de red, es la capa central donde las abstracciones SDN se encuentran. El controlador está escrito en Java y por lo tanto está contenido dentro de su propia Java Virtual Machine (JVM) pudiéndose ejecutar en cualquier plataforma con este software. Éste se basa en las siguientes tecnologías:

- **Maven**, es una herramienta de administración de proyectos, que simplifica y automatiza el uso de dependencias. La configuración se realiza mediante un fichero en formato XML, donde se definen los módulos y dependencias del proyecto.
- **Open Service Gateway Interface (OSGi)**, es la parte de back-end de OpenDaylight y está implementada bajo dos versiones, OSGi Equinox y Apache Felix. Se usan ambas ya que Equinox implementa varios servicios de OSGi modulares denominados bundles. Estos se ejecutan encima de Equinox. Por otro lado, Apache Felix se utiliza a la hora de construir los bundles cuando se realiza la instalación y control de las dependencias del controlador. Se usan ambos debido a que Equinox no tiene la suficiente flexibilidad a la hora de la administración del ciclo de vida de los bundles.
- **Karaf**, es un contenedor de aplicaciones construidas encima de OSGi, que simplifica todos los aspectos de la instalación de diferentes módulos y plugins de ODL. Además, se pueden realizar instalaciones mediante el uso de features, los cuales son agrupaciones de diferentes bundles. Proporciona características como configuración dinámica, despliegue dinámico, sistemas de registros, seguridad, etc.
- **YANG**, es el lenguaje de modelado basado en model-driven que permite la configuración del modelo y el estado de los datos en los dispositivos de red. ODL lo utiliza con protocolos como NETCONF para definir como se almacenan los datos y dar soporte a las diferentes interacciones entre aplicaciones, y RESTCONF para el control de accesos externos.

Plano de datos

Es la capa de más bajo nivel donde se encuentran los equipos de red físicos y virtuales que gestiona ODL, como por ejemplo switches.

3.2.2 Interfaces de comunicación

Para la comunicación entre las diferentes capas se encuentran el Northbound API y el Southbound API, ambos son modelados mediante el uso del lenguaje model-driven YANG. La Northbound API está compuesta por diferentes protocolos como el RESTCONF y el NETCONF, estos son modelados mediante YANG [60]. A través de una API REST las aplicaciones de la capa superior se comunican con el controlador donde una capa intermedia actúa de protección para dar seguridad a la arquitectura. En el Southbound API se encuentran diferentes plugin que hacen uso de protocolos que controlan la infraestructura de red del plano de datos. Las aplicaciones de red obtienen información sobre el estado de la red mediante el uso de plugin como OpenFlow, OVSDB, Netconf, SNMP entro otros.

YANG

Dentro de ODL se utiliza lenguajes de modelado en el ámbito de las redes para describir la funcionalidad de servicios de red, políticas y APIs. El Model Driven Software Engineering (MDSE) [61] organizado por el Object Management Group (OMG), describen mediante un lenguaje modelado fuentes de datos y APIs, es decir, tienen la habilidad de generar código a partir de modelos. Este tipo de model-driven incrementan la productividad al aumentar la compatibilidad entre distintos sistemas ya que estandarizan el mapeo de los modelos. De esta forma YANG surge como solución al modelado de datos para el ámbito de las redes y tiene un flujo de modelado como el que se muestra en la Figura 22. Como se puede ver YANG se utiliza para modelar la funcionalidad de una aplicación, y generar estas APIs, para después realizar implementaciones de código.

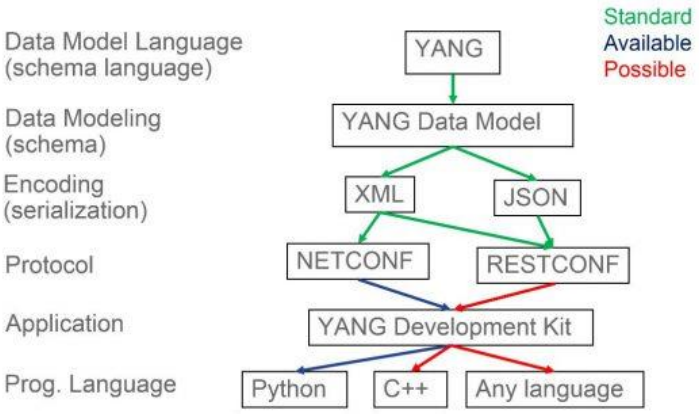

Figura 22: Flujo de generación de código mediante YANG

Service Abstraction Layer

La Service Abstraction Layer (SAL) es la principal innovación de ODL dentro del marco de SDN, ya que ésta separa los plugin de protocolos de Southbound del plugin de aplicaciones del Northbound. No necesita conocer los protocolos SDN a la hora de la comunicación ya que esta SAL adapta el plugin de funciones del Southbound a un nivel superior de funciones de aplicación provistas por la Northbound API del controlador. De esta forma permite al

controlador soportar múltiples plugins del Southbound y un conjunto de servicios a través del Northbound API [62].

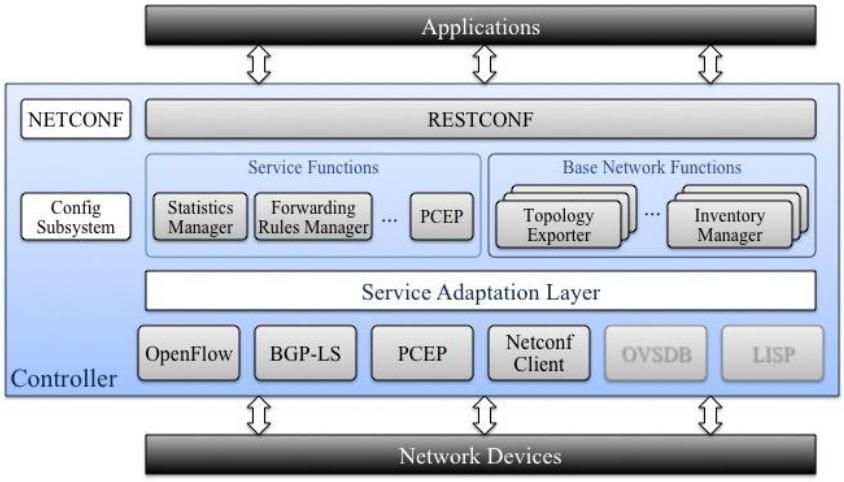

Figura 23: Arquitectura del Service Abstraction Layer

En primera instancia ODL implementaba SAL mediante otro enfoque, el API-Driven SAL (AD-SAL), en la cual un desarrollador tenía que especificar manualmente que APIs iba a utilizar mediante un plugin específico, y programar la funcionalidad entre los plugins del Northbound y el Southbound. De hecho, para lograr que el Northbound API fuera accesible por múltiples plugin del Southbound API cada uno con formatos distintos, estos desarrolladores tenían que mapear cada uno de los plugin del Southbound al Northbound API. De tal forma que las adaptaciones de los datos estaban definidas de forma estática cuando se realizaba la compilación. Por lo que, dado su inefectividad, se trasladó a un nuevo enfoque, el Model-Driven SAL (MD-SAL). Se puede observar la evolución en la Figura 24.

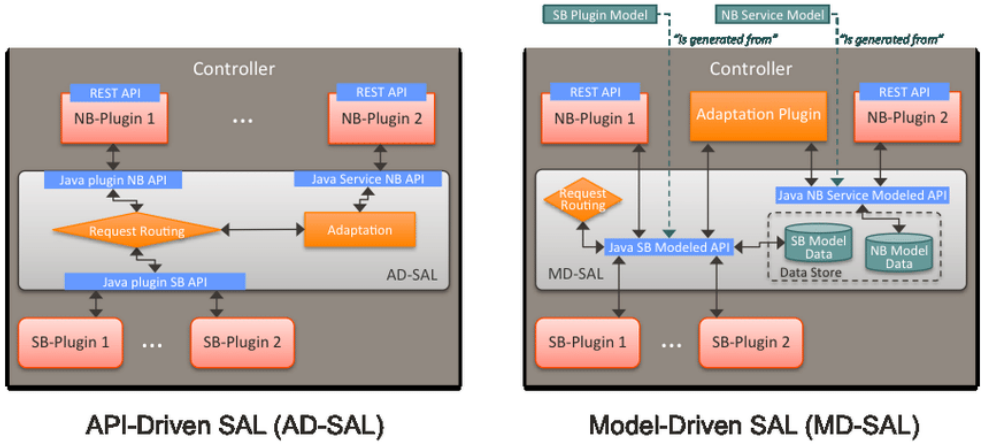

Figura 24: Evolución de la AD-SAL al MD-SAL en el SAL [63]

Este nuevo enfoque MD-SAL es un conjunto de servicios de infraestructura que ayudan a proporcionar soporte común y genérico a los desarrolladores de aplicaciones y plugin. Define funcionalidades de mensajería y almacenamiento para datos, notificaciones, modelado RPC y

además utiliza YANG como lenguaje de modelado para definiciones de servicio y datos. Ahora los desarrolladores pueden describir la funcionalidad de las aplicaciones en modelos YANG y generar APIs, que no son más que clases Java generadas automáticamente.

Con MD-SAL, ahora se puede utilizar para almacenar datos o servicios y consumirlos actuando como proveedor o consumidor, esto se puede realizar mediante el protocolo NETCONF [64] modelado por YANG. NETCONF es un protocolo de administración de red definido por el IETF, que describe las configuraciones y las operaciones conceptuales da almacenamiento de los datos, que son, Created, Retrieve, Update y Delete (CRUD). Además, NETCONF soporta el uso de las Remote Procedure Calls (RPCs) las cuales sirven para realizar peticiones desde otro sistema para realizar alguna operación.

Como consecuencia de que se utilice el mismo lenguaje de modelado, solo hace falta definir una API que mediante YANG se traduce a un formato compatible con una REST API común a todas las aplicaciones. Esta se traduce en ODL como RESTCONF, y es un protocolo que provee de una interfaz sobre HTTP para acceder a los datos utilizando la configuración definida en el NETCONF. Se pueden obtener mediante el método de HTTP GET y se modifican con los métodos HTTP PUT, POST, PATCH y DELETE. Los datos obtenidos o que es quieran modificar se pueden codificar tanto en formato XML como en JSON. Esta REST API es autogenerada mediante los modelos YANG, pero manteniendo las interacciones de una REST típica.

En definitiva, MD-SAL interpreta los plugin de las APIs una vez son cargados en el controlador, sin necesidad de que en el SAL haya código o la API, sino que se adapta a cualquier plugin o aplicación cargada en el controlador. Además, ODL incluye herramientas como el proyecto YANGTools, que genera las APIs a partir del modelo de YANG, y son básicamente analizadores de ficheros YANG.

3.3 Integración OpenStack y ODL

ODL es un controlador SDN muy potente y la integración con plataformas de computación en la nube como OpenStack simplifica la gestión de la red y otorga de mayor flexibilidad y escalabilidad a las redes. Con ello, se puede realizar la integración de ambas y se puede hacer mediante la instalación de módulos compatibles mediante plugin y drivers, ya que ambos son plataformas modulares.

Ambas plataformas pueden trabajar con su propia arquitectura, una separada de la otra, beneficiándose mutuamente sin interferir la una con la otra, mediante una conexión entre ambas comunicándose y operando en conjunto. Es necesario la modificación de los componentes de OpenStack, y el controlador de ODL quedando como resultado una topología

como la mostrada en la Figura 25. En concreto, para crear esta conexión, se realiza modificando el componente de Neutron OpenStack, teniendo que añadir algunos drivers y módulos. ODL quedaría con estos drivers realizando funcionalidades que antes operaban dentro de Neutron.

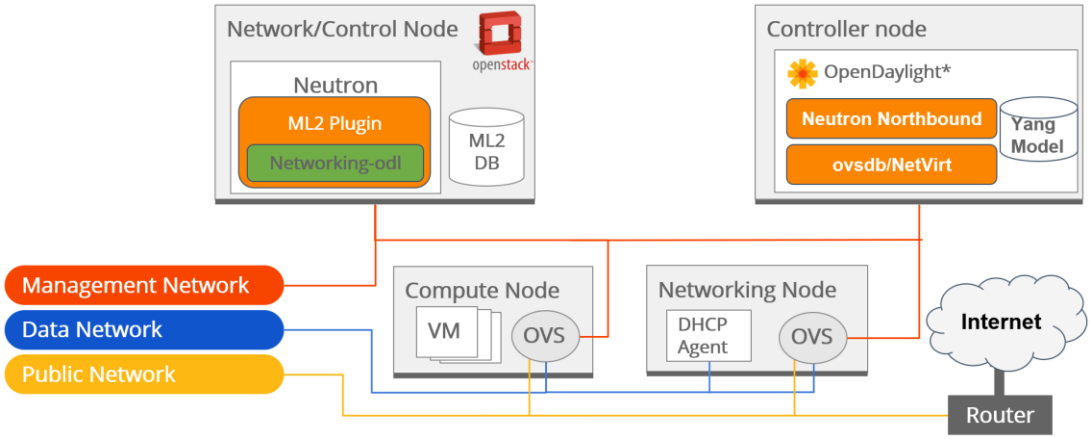

Figura 25: Integración de OpenDaylight con OpenStack [64]

3.3.1 OpenStack Neutron: Plugin ML2

Dentro de OpenStack en el componente de Neutron se hace uso del plugin Modular Layer 2 (ML2) [65] el cual permite el uso de tecnologías de nivel 2 como Open vSwitch o Linux Bridge. Dentro del marco de ML2 existen dos tipos de drivers que pueden ser configurados:

- **Type Drivers**, estos drivers mantienen cualquier necesidad en el estado de la red, validan los proveedores de red y realizan la asignación de redes de los usuarios. Los Type Drivers pueden ser,
  - **Local**, segmento de red que da conectividad entre máquinas virtuales localizadas en el mismo nodo de computación.
  - **Flat**, en estas redes todo el mundo comparte el mismo segmento de red.
  - **VLAN**, cada cliente está separado porque cada uno está asignado a una VLAN.
  - **GRE** y **VXLAN**, también proporciona separación entre clientes y también permite la superposición de subredes y rangos de IP, pero lo hace encapsulando el tráfico de los clientes en túneles.
- **Mechanism Drivers**, es responsable de tratar la información establecida por el Type Driver y asegurar que se aplica por el mecanismo de red que han sido seleccionados. Son los que dan acceso a la red mediante agentes L2 en los nodos de computación que realizan llamadas RPC. Tratan tecnologías basadas en agentes, en controladores como ODL y dispositivos físicos de red.

De esta forma Neutron utiliza el mechanism driver de ODL y el plugin L3 y se pasan todas las llamadas a la API de Neutron hacia ODL, y dentro de ODL hay un servicio REST en la interfaz Northbound de Neutron. El Neutron Northbound ubicado dentro de ODL, traslada las llamadas realizadas a la API a cada servicio de ODL usando YANG como lenguaje de

modelado. Además del Neutron Northbound hacen falta implementar otros plugin dentro de ODL, como el OVSDB Southbound.

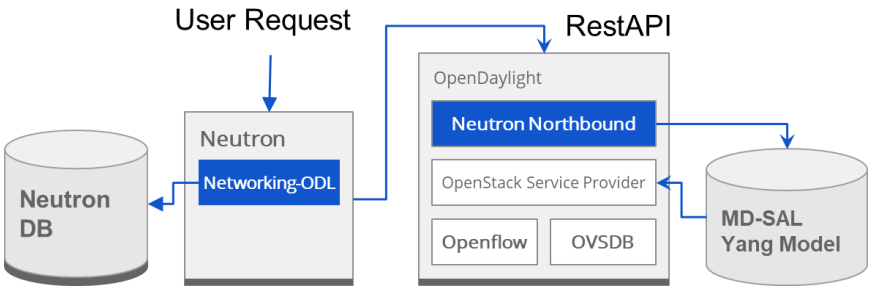

Figura 26: Flujo de Neutron Northbound [66]

Agentes L2 y L3

Para hacer uso del plugin ML2 mediante Mechanism Driver y un agente como Open vSwitch este necesita hacer uso de los agentes de L2 y L3 de Neutron. Un agente L2 da conectividad al nivel 2 de red a los recursos de OpenStack, y se ejecuta en cada nodo de red y en cada nodo de computación de OpenStack. Por otro lado, el agente L3 ofrece servicios avanzados en el nivel 3 de red, como routers virtuales o IPs flotantes [67].

3.3.2 Netvirt

Netvirt es una aplicación de virtualización de red desarrollado como proyecto dentro de OpenDaylight, previamente formaba parte del plugin OVSDB. Consiste en la implementación de sub-servicios modulares para dar soporte a otros plugin o proyectos tales como L2, L3, ACL, NAT, DHCP, control IPv6, OVSDB, virtualización basada en OVS para switches virtuales, OVS data paths aceleradas por DPDK, L3VPN (BGP VPN), SFC, QoS y Hardware VTEP entre otros [68]. Hardware VTEP son endpoints que encapsulan VXLAN, denominados VXLAN Tunnel Endpoints [69]. Actualmente da soporte y controla OVS y L2 Gateways, y está pensado para dar soporte en un futuro a Cisco's Virtual Packet Processing y Container Network Interface para Kubernetes.

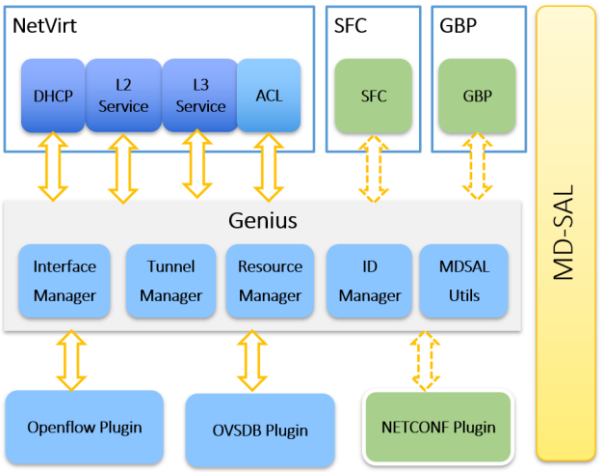

Figura 27: Estructura Netvirt dentro de ODL [66]

Genius

El proyecto de Genius fue introducido en la versión Boron de OpenDaylight y provee de una Generic Network Interfaces, Utilities and Services (Genius). Es un marco que integra diferentes servicios de red y provee de una visión de los interfaces de red a través de los servicios. También desacopla la aplicación de NSF del plugin de Southbound, y comparte recursos de OpenFlow.

SFC

Como ya se ha definido anteriormente SFC, es importante a la hora de la implementación de OpenStack Tacker para crear una cadena virtual de servicios haciendo uso de SDN y así tener interconectados diferentes VNF en máquinas separadas. Dentro de ODL el proyecto SFC, el ACL (clasificador SFC) y SFF se controlan a través de OVSDB y OpenFlow.

3.3.3 OVSDB

Se trata del plugin de OpenFlow y OVSDB, donde el OVSDB se utiliza para manejar las operaciones de tipo CRUD en los bridges de OVS, switches y túneles. Este plugin se utiliza para proporcionar a soporte a OpenStack a través de Neutron Northbound. Por debajo se implementa Open vSwitch (OVS), que es considerado como el estándar no oficial para conmutación virtual. Interactúan a través de dos canales Southbound,

- **OpenFlow**, que se encarga del reenvío de la funcionalidad OVS.
- **OVSDB**, por otro lado, aborda el plano de administración configurando datos estáticos en OVS.

Por otro lado, se encuentra el Open vSwitch y el Hardware VTEP, donde la idea es que se cree una red superpuesta que interconecte diferentes máquinas virtuales y físicas de forma que desde su punto de vista estén en el mismo nivel 2 de red.

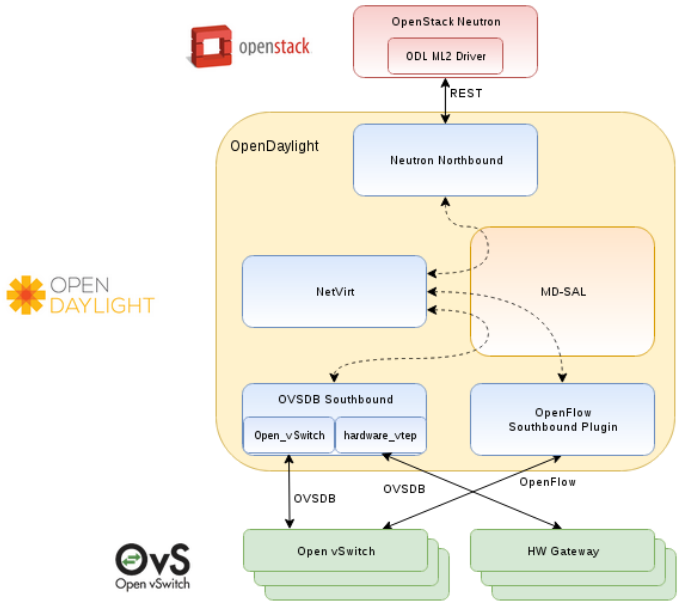

Figura 28: OpenStack y ODL Netvirt [67]

Open vSwitch

Open vSwitch (OVS) es un software de código abierto bajo licencia Apache 2.0, pensado para que realizar funciones de switches virtuales en entornos de virtualización. Este software multicapa para switches, es una implementación de OpenFlow [67]. También proporciona control y administración remota sobre OVSDB u OpenFlow 1.0-1.3, y ofrece múltiples características en los entornos virtuales como,

- VLANs
- Protocolos de túneles
- LACP, permite balancear el tráfico.
- QoS, define varios niveles como disponibilidad, ancho de banda, etc.
- Access Control Lists (ACLs) desde protocolos de nivel 2 a 4.
- NetFlow, sFlow, IPFIX.

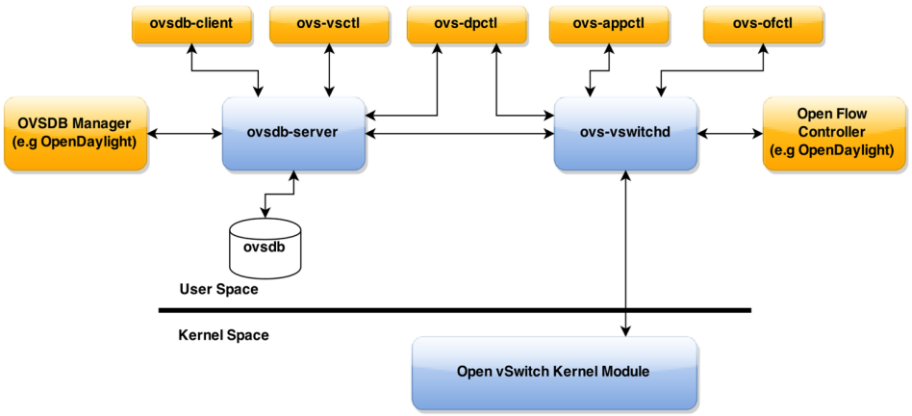

Figura 29: Componentes de Open vSwitch

Tal y como se muestra en la Figura 29 los componentes principales son el ovsdb-server, el cual mantiene una base de datos donde se encuentra la configuración, y tiene propiedades como restricciones de valor, referencias débiles, recolección de basura. Mediante el uso de protocolos de administración como OVSDB para administrar el ovs-vswitchd. Este es el otro componente principal y componente central en el sistema. El ovs-vswitchd se comunica fuera de la red utilizando OpenFlow, con el sistema a través de interfaces abstractas y con el módulo Open vSwitch Kernel.

OVS es importante a la hora de implementar SFC, ya que el bridge creado por OVS el Neutron Integration (br-int), actúa como SFF.

3.3.4 Drivers ODL

Como se ve en la Tabla 4 existen otros componentes y drivers que se pueden agregar dentro de ODL para su uso. Son drivers que realizan funciones que el componente de Neutron de OpenStack tiene, y se pueden realizar dentro de ODL mediante la integración de OpenStack y OpenDaylight. Se pueden agregar estos drivers dentro de ODL ya sea mediante Netvirt u OVSDB para proporcionar Quality of Service (QoS), Load Balancing as a Service (LBaaS), FireWall as a Service (FWaaS) o L3VPN (BGP VPN).

<table>
<tr><td colspan="9">Neutron Server</td></tr>
<tr><td colspan="6">ML2 Plugin</td><td rowspan="3">ODL L3 Plugin</td><td colspan="6">Service Plugins</td></tr>
<tr><td colspan="4">Type Manager
TypeDrivers</td><td colspan="2">Mechanism Manager</td><td>FWaaS</td><td>L2GW</td><td>QoS</td><td>LBaaS</td><td>SFC</td><td>BGP VPN</td></tr>
<tr><td>GRE</td><td>VLAN</td><td>VXLAN</td><td>FLAT</td><td>ODL Mech Driver</td><td>SR-IOV</td><td>ODL Driver</td><td>ODL Driver</td><td>ODL Driver</td><td>ODL Driver</td><td>ODL Driver</td><td>ODL Driver</td></tr>
</table>

Tabla 4: Resumen Plugin y Drivers para integrar OpenStack con ODL

3.3.5 Arquitectura tacker y ODL

Una vez integrados OpenStack y OpenDaylight, el uso de OpenStack Tacker no cambia esta integración, ya que añade funcionalidades o sustituye otros componentes que no forman parte de la integración, ya que solo es necesario modificar el componente de Neutron y los compontes que forman el VIM dentro de Tacker. De esta forma mediante Tacker y ODL queda una arquitectura tal y como se puede ver en la

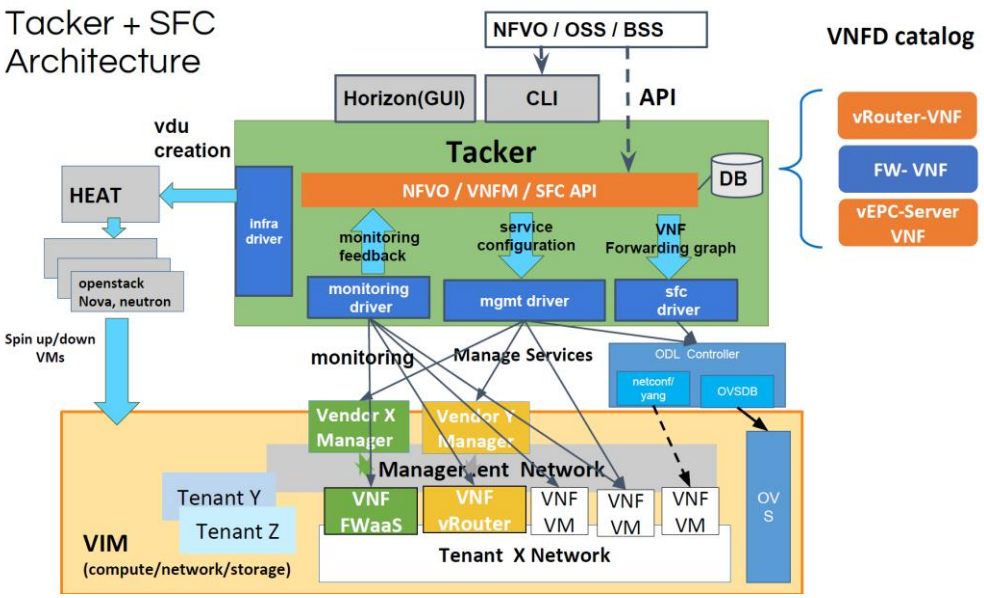

Figura 30: Integración de OpenStack Tacker y ODL

3.4 Devstack

3.4.1 Herramienta de Automatización

Dentro de los proyectos existentes de OpenStack, existe una herramienta de automatización llamada Devstack, el cual sirve de herramienta de automatización para desplegar una arquitectura OpenStack completa. Es capaz de instalar y levantar todos los servicios centrales dentro de un entorno de desarrollo, ya que es una herramienta para probar posibles configuraciones de una infraestructura. No fue desarrollado pensando en ser una herramienta de instalación estándar, sino que ha evolucionado hasta convertirse en un proyecto en el que se pueden realizar multitud de configuraciones y alternativas mediante plugins que no se han probado todas, y por lo tanto en ocasiones ciertas configuraciones no tienen un soporte. Pero esto deja hueco a la investigación del uso de esta herramienta ya que es muy flexible y permite ajustar OpenStack a las necesidades de cada arquitectura a implementar [71].

Devstack es configurable mediante la modificación del fichero *local.conf* situado en la raíz de su estructura, aunque algunas implementaciones necesitan además la modificación de otros ficheros de configuración. OpenStack ofrece una documentación extensa para instalar diferentes plugin, estos plugin son proyectos de OpenStack o componentes, ya que tiene una arquitectura modular. De esta forma se pueden instalar los diferentes componentes mediante el agregado de plugin dentro del fichero de configuración de Devstack.

OpenStack Base

Para la instalación OpenStack base, es decir los componentes básicos se realiza mediante la siguiente configuración:

```
[[local|localrc]]
# Pip
PIP_USE_MIRRORS=False
USE_GET_PIP=1
ADMIN_PASSWORD=password
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
MYSQL_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=supersecrettoken

HOST_IP=192.168.183.137
#HOST_IPV6=2001:db8::7
LOGFILE=$DEST/logs/stack.sh.log
LOGDAYS=2

SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
SWIFT_REPLICAS=1

SWIFT_DATA_DIR=$DEST/data
enable_service dstat
enable_service g-api
enable_service g-reg
enable_service key
enable_service mysql
enable_service n-api
enable_service n-cond
enable_service n-cpu
enable_service n-crt
enable_service n-novnc
enable_service n-sch
enable_service placement-api
enable_service placement-client
enable_service neutron-dhcp
enable_service neutron-metadata-agent
enable_service neutron-api
enable_service rabbit
disable_service c-api
disable_service c-vol
disable_service c-sch

SKIP_EXERCISES=boot_from_volume,bundle,client-env,euca

SYSLOG=False
LOGFILE=/opt/stack/new/devstacklog.txt
VERBOSE=True
ENABLE_VERBOSE_LOG_LEVEL=True
FIXED_RANGE=10.1.0.0/20
FLOATING_RANGE=172.24.5.0/24
PUBLIC_NETWORK_GATEWAY=172.24.5.1
FIXED_NETWORK_SIZE=4096
VIRT_DRIVER=libvirt

export OS_NO_CACHE=1
DATABASE_QUERY_LOGGING=True
```

Networking-ODL

Una vez con la configuración base de OpenStack hay que instalar ODL mediante la instalación del plugin llamado networking-odl, el cual es una biblioteca de drivers que integran la API de Neutron con el Backend de OpenDaylight. Lo hace mediante el uso de los plugin ML2 y L3 para comunicarse con el L2 y L3 de Neutron. Esta integración se realiza mediante la siguiente configuración en el fichero *local.conf*,

```
enable_plugin networking-odl http://git.openstack.org/openstack/networking-odl
stable/rocky

ODL_RELEASE=oxygen-latest

ODL_NETVIRT_KARAF_FEATURE=odl-netvirt-vpnservice-openstack, odl-restconf-
all,odl-aaa-authn,odl-dlux-core,odl-mdsal-apidocs

ODL_NETVIRT_KARAF_FEATURE+=,odl-dluxapps-nodes,odl-dluxapps-topology,odl-
dluxapps-yangui,odl-dluxapps-yangvisualizer

ODL_NETVIRT_KARAF_FEATURE+=,odl-l2switch-switch,odl-l2switch-switch-ui,odl-
ovsdb-hwvtepsouthbound-ui,odl-ovsdb-southbound-impl-ui,odl-netvirt-ui

ODL_NETVIRT_KARAF_FEATURE+=,odl-openflowplugin-flow-services-ui

ODL_NETVIRT_KARAF_FEATURE+=,odl-netvirt-openstack

ODL_NETVIRT_KARAF_FEATURE+=,odl-neutron-logger

ODL_L3=True
```

Como se puede ver se realizan la instalación de los diferentes servicios de ODL mediante la línea *ODL_NETVIRT_KARAF_FEATURE*. Se agregan las características de la siguiente forma,

Grupo	Característica de ODL	Descripción detellada
Características básicas	*odl-netvirt-vpnservice-openstack*	Nueva versión de Netvirt mediante uso de VPN Service en vez del basado en OVSDB
	odl-restconf-all	Característica de Restconf
	odl-aaa-authn (odl-aaa-shiro)	Habilita AAA en todas las interfaces RESTFUL ODL
	odl-dlux-core	Núcleo que proporciona funcionalidad básica de diseño, AAA, etc.
	odl-mdsal-apidocs	característica de Restconf MD-SAL
Características Karaf DLUX necesarias para la GUI ODL	*odl-dluxapps-nodes*	Muestra la lista de nodos en topología de OpenFlow (flow: 1)
	odl-dluxapps-topology	Muestra nodos y enlaces desde topología OpenFlow (flow: 1)
	odl-dluxapps-yangui	Aplicación YangUI

	odl-dluxapps-yangvisualizer	Visualización tipo topología de modelos Yang cargados en el controlador
Características para L2 y la GUI DLUX	*odl-l2switch-switch*	Abstracción básica del conmutador L2 sobre múltiples conmutadores utilizando OpenFlow
	odl-l2switch-switch-ui	Caracterísitca L2 Switch con interfaz REST y UI
	odl-ovsdb-hwvtepsouthbound-ui	OVSDB Southbound Hardware VTEP con interfaz REST y UI
	odl-ovsdb-southbound-impl-ui	OVSDB Southbound Open vSwitch con interfaz REST y UI
	odl-netvirt-ui	Característica para Netvirt con interfaz REST y UI
OpenFlow	*odl-openflowplugin-flow-services-ui*	Función de envoltura para aplicaciones estándar con interfaz REST y UI
Netvirt	*odl-netvirt-openstack*	Integración mediante la característica de Netvirt
Log de YANG Neutron	*odl-neutron-logger*	Registrador de actividad en modelos Neutron YANG

Tabla 5: Características Karaf de ODL

Networking-SFC

Para habilitar SFC en Devstack hace falta activar el proyecto plugin networking-sfc, el cual provee de la capacidad mediante Netvirt, y solo hace falta realizar la siguiente configuración,

```
## FOR SERVICE FUNCTION CHAINING
#enable_plugin networking-ovs-dpdk
http://git.openstack.org/openstack/networking-ovs-dpdk stable/rocky
SKIP_OVS_INSTALL=True
# enable the networking-sfc plugin:
enable_plugin networking-sfc https://github.com/openstack/networking-sfc.git
stable/rocky

# enable the odl-netvirt-sfc Karaf feature in OpenDaylight
ODL_NETVIRT_KARAF_FEATURE+=,odl-netvirt-sfc
# enable the networking-sfc OpenDaylight driver pair
[[post-config|$NEUTRON_CONF]]
[sfc]
drivers = odl_v2
[flowclassifier]
drivers = odl_v2
```

OpenStack Tacker

Una vez cargados los plugin anteriores, para utilizar OpenStack Tacker, se debe configurar como se muestra en el código de abajo. Mediante las variables *Q_Plugin* y *Q_Agent* se configura el plugin ML2 y se selecciona el Open vSwitch. Además, hay que descargar e instalar los componentes de OpenStack adicionales, *Heat*, *Barbican*, *Mistral*, *Ceilometer*, *Aodh* y *networking-sfc*, para poder desplegar Tacker, tal y como se describió previamente, ya que Tacker hace uso de estos componentes.

local.conf

```
Q_PLUGIN=ml2
Q_AGENT=openvswitch

Q_USE_SECGROUP=False
LIBVIRT_FIREWALL_DRIVER=nova.virt.firewall.NoopFirewallDriver

enable_plugin heat https://git.openstack.org/openstack/heat stable/rocky
enable_plugin barbican https://git.openstack.org/openstack/barbican stable/rocky
enable_plugin mistral https://git.openstack.org/openstack/mistral stable/rocky

# Ceilometer
enable_plugin ceilometer https://git.openstack.org/openstack/ceilometer
stable/rocky
enable_plugin aodh https://git.openstack.org/openstack/aodh stable/rocky

# Tacker
enable_plugin tacker https://git.openstack.org/openstack/tacker stable/rocky

enable_service n-novnc
enable_service n-cauth

disable_service tempest

KUBERNETES_VIM=False
Q_PLUGIN_CONF_FILE=/etc/neutron/dhcp_agent.ini

[[post-config|$Q_PLUGIN_CONF_FILE]]
[DEFAULT]
enable_isolated_metadata = True
```

Servicios Adicionales

Para activar FWaaS y L2GW

```
enable_plugin networking-l2gw http://opendev.org/openstack/networking-l2gw
enable_service l2gw-plugin
NETWORKING_L2GW_SERVICE_DRIVER=L2GW:OpenDaylight:networking_odl.l2gateway.driver
_v2.OpenDaylightL2gwDriver:default
```

Para activar LBaaS.

```
enable_service q-lbaasv2
NEUTRON_LBAAS_SERVICE_PROVIDERV2="LOADBALANCERV2:opendaylight:networking_odl.lba
as.lbaasv2_driver_v2.OpenDaylightLbaasDriverV2:default"
enable_plugin neutron-lbaas https://git.openstack.org/openstack/neutron-lbaas
stable/rocky
```

Para activar L3 VPN (BGP VPN)

```
enable_plugin networking-bgpvpn https://opendev.org/openstack/networking-
bgpvpn.git
[[post-config|$NETWORKING_BGPVPN_CONF]]
[service_providers]
service_provider=BGPVPN:OpenDaylight:networking_odl.bgpvpn.odl_v2.OpenDaylightBg
pvpnDriver:default
```

Para habilitar QoS en el Backend de ODL hay que modificar el fichero de configuración de Neutron y el de ML2.

/etc/neutron/neutron.conf

```
service_plugins = qos, odl-router
```

/etc/neutron/plugins/ml2/ml2_conf.ini

```
extensions_drivers = qos, port_security
```

3.5 Otras herramientas de automatización

A parte de Devstack, existen otras herramientas que proporcionan un entorno de desarrollo para la implementación de OpenStack junto con ODL. Entre estos destaca Kolla-Ansible [1], el cual, mediante Ansible, una herramienta de software libre que permite administrar, configurar y realizar instalaciones multi-nodo. Esta herramienta es recomendable en entornos de desarrollo y producción debido a que además utiliza Docker para encapsular los servicios de OpenStack, y mantener un entorno aislado, pudiendo borrar estos contenedores una vez se termine el desarrollo. Además, permite la integración de ODL [2] y Tacker [3]. Como otras opciones se encuentran, Fuel [4] o V-Magine [5] una herramienta ara instalaciones mediante el uso de Hyper-V, el hipervisor de virtualización del sistema operativo Windows.

4 Resultados

La infraestructura definida previamente de OpenStack Tacker junto con el controlador ODL, finalmente se decidió realizarla mediante el uso de la herramienta de automatización Devstack. A la hora de construir un entorno de investigación y desarrollo, Devstack es la mejor opción, y se ha escogido esta herramienta debido a que permite mayor flexibilidad, ya que opera con los proyectos directamente de los repositorios oficiales de OpenStack. Además, cuenta con una documentación extensa y una comunidad que realiza parches de fallos que se van encontrando, para intentar abordar todas las configuraciones posibles.

A la hora de abordar la instalación y configuración de la plataforma IaaS definida, se ha implementado el desarrollo en una única máquina Linux virtualizada de forma que se simplifique la instalación para poder mostrar cómo construir la arquitectura. La máquina tiene las siguientes características,

- OS: Ubuntu 18.04.3
- Procesador: 2 procesadores con 2 núcleos - Intel® Core™ i7-6700HQ (6M Cache, 2.6GHz hasta 3.5GHz).
- Memoria RAM: RAM 8GB DDR4 2133MHz y 8GB en fichero Swap
- Disco Duro: Partición 50 GB 5400-RPM SATA.

4.1 Preparación del sistema

Para comenzar la instalación hacen falta algunos requisitos previos, para después tener una mejor optimización en el despliegue, requisitos para la instalación y otras herramientas que no vienen por defecto en Ubuntu.

4.1.1 Aumentar la RAM mediante SwapFile

Cuando se realiza la instalación de Devstack y se ejecuta el proceso, va a lanzar muchos servicios, los cuales, requieren de mucha memoria RAM. Como es un entorno de desarrollo y no se disponen de otras máquinas para que el sistema no se colapse y se quede congelado es necesario aumentar el fichero swapfile para conseguir una ejecución más rápida. Esto se puede lograr mediante los siguientes comandos,

En primer lugar, se desconecta el fichero swapfile para poder editarlo

```
sudo swapoff /swapfile
```

Se actualiza al tamaño que se quiera

```
sudo fallocate -l 8G /swapfile
```

Se cambian los permisos y se crea el fichero swap

```
sudo chmod 600 /swapfile
sudo mkswap /swapfile
```

Para finalizar se ejecuta el siguiente comando para que los cambios sean permanentes

```
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
sudo swapon /swapfile
```

También se pueden visualizar mediante los siguientes comandos si se ha configurado bien

```
sudo free -h
              total        used        free      shared  buff/cache   available
Mem:           7.8G        3.3G        1.5G        4.2M        3.0G        4.2G
Swap:          8.0G          0B        8.0G
```

4.1.2 Actualización e Instalaciones previas

Hace falta realizar instalaciones de herramientas como el uso del comando *git* para descargar repositorios, o instalación de *net-tools* y de *virtualenv* como requisitos del despliegue.

```
sudo apt-get update
sudo apt install git
sudo apt install virtualenv
sudo apt install net-tools
```

4.1.3 Instalación de OVS

Antes de realizar la ejecución de Devstack es necesario realizar una compilación e instalación previas de Open vSwitch de forma manual [77]. Es debido a que se necesita soporte para OpenFlow NSH cuando se realiza la instalación de Networking-odl y Networking-sfc, ya que esto no lo hace de forma automática Devstack. Para ello se deben realizar las siguientes líneas de comando para instalar OVS 2.9,

Primero, se instalan todas las herramientas necesarias para compilar e instalar OVS

```
sudo apt-get install -y autoconf libtool git dh-autoreconf dh-systemd software-
properties-common
sudo apt-get install -y linux-image-generic libssl-dev openssl build-essential
fakeroot graphviz python-all python-qt4
sudo apt-get install -y python-twisted-conch dkms
```

Después se clona mediante *git* el repositorio de OVS

```
git clone https://github.com/openvswitch/ovs.git
cd ovs
git checkout -b v2.9.2
```

Debido a que se está utilizando una versión diferente de Ubuntu a la definida por la documentación descrita en ODL [77], hay un fallo a la hora de compilar las dependencias, y hay que modificar el fichero *lib/automake.mk* para que cuando se compile borre el valor *static* de las funciones definidas en *lib/dhparams.c* [78].

lib/automake.mk: 436-445

```
lib/dhparams.c: lib/dh1024.pem lib/dh2048.pem lib/dh4096.pem
	$(AM_V_GEN)(echo '#include "lib/dhparams.h"' &&          \
	openssl dhparam -C -in $(srcdir)/lib/dh1024.pem -noout && \
	openssl dhparam -C -in $(srcdir)/lib/dh2048.pem -noout && \
	openssl dhparam -C -in $(srcdir)/lib/dh4096.pem -noout)   \
```

```
        | sed 's/\(get_dh[0-9]*\)()/\1(void)/' > lib/dhparams.c.tmp &&  \
+       sed -i '/\(get_dh[0-9]*\)(void)/s/^static//' lib/dhparams.c.tmp && \
        mv lib/dhparams.c.tmp lib/dhparams.c
else
lib_libopenvswitch_la_SOURCES += lib/stream-nossl.c
```

Una vez resuelto se compila mediante la siguiente ejecución

```
sudo DEB_BUILD_OPTIONS='parallel=8 nocheck' fakeroot debian/rules binary
```

Esto genera varios archivos *.deb* que son los paquetes de instalación. Se mueven a una carpeta en la raíz del sistema */vagrant/ovs_debs* y se instala mediante el comando *dpkg -i*.

```
cd ..
sudo mkdir -p /vagrant/ovs_debs
sudo cp ./libopenvswitch_*.deb ./openvswitch-common*.deb ./openvswitch-
switch*.deb /vagrant/ovs_debs/
sudo dpkg -i ./libopenvswitch_*.deb ./openvswitch-datapath-dkms* ./openvswitch-
common* ./openvswitch-switch* ./python-openvswitch*
```

Ya está instalado OVS, asique para lanzar el servicio se puede ejecutar mediante

```
service openvswitch-switch restart
```

4.1.4 Aumentar el límite de watchers del sistema

En el sistema Linux se utiliza inotify para monitorear los cambios en directorios, y es común que debido a procesos y servicios que realizan muchos cambios en el sistema llegue al límite establecido [79]. Puede notificar con un fallo parecido a

```
Failed to watch /var/log/messages; upper limit on inotify watches reached!
```

Para aumentar el número de inotify watchers se realiza con el siguiente comando

```
echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

4.1.5 Requisitos Previos del sistema

Para poder llevar a cabo la instalación de Devstack se necesita tener instalado Python 2.7.X, y tenerlo definido por defecto si hay otra instalación de Python 3.X. Para comprobar la versión por defecto de Python se puede comprobar mediante la ejecución del comando *python -V*. En caso de que se tenga Python 3.X, y existe otra instalación de Python 2.7.X se puede cambiar la versión de Python por defecto mediante *update-alternatives*

```
update-alternatives --install /usr/bin/python python /usr/bin/python2 1
```

De esta forma crea un enlace simbólico de la ubicación del comando *python* para que cuando se ejecute se haga mediante la versión Python 2.7.

Es necesario realizar la instalación mediante esta versión de Python 2.7.X debido a que el plugin de Devstack de ODL no es del todo compatible con Python 3.X en algunas versiones de ODL [80]. Es debido a que debido a que Devstack empezó desarrollándose inicialmente en Python 2.7. De hecho, para que el plugin de ODL funcione con Python 3.X hay que realizar

cambios en el fichero *networking-odl/networking_odl/cmd/set_ovs_hostconfigs.py* de forma que use librería *six.py* para la compatibilidad de Python 2.7.X y Python 3.X.

4.2 Instalación y configuración de Devstack

4.2.1 Instalación Devstack

En el desarrollo se ha instalado Devstack con la versión de OpenStack Rocky la cual fue lanzada en agosto del 2018. Para la versión de ODL se ha optado por la versión Oxygen lanzado en marzo de 2018. Se han elegido estas versiones ya que fueron lanzadas en las mismas fechas, y son versiones estables para poder realizar correctamente la integración de Tacker junto con el controlador SDN. A continuación, se indican los pasos seguidos de la documentación de OpenStack [81] para la instalación.

Descarga del repositorio

Primero se añade un usuario nuevo para realizar la instalación lo más limpia posible en la carpeta */opt/stack*.

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

A continuación, se clona el repositorio de Devstack de la rama de Rocky y también Tacker y el proyecto Requirements.

```
git clone https://git.openstack.org/openstack-dev/devstack -b stable/rocky
git clone https://git.openstack.org/openstack/tacker -b stable/rocky
git clone https://git.openstack.org/openstack/requirements.git -b stable/rocky
```

Fallos en la ejecución de Devstack

Una vez descargado todos los proyectos, se deben modificar el de Tacker y Requirements debido a que la versión de Rocky contiene algunos fallos a la hora de ejecución. Dentro de Tacker, en el fichero */opt/stack/tacker/etc/config-generator.conf* tiene un error en el nombre de una de las variables *namespace* que define. Este fallo es debido a que el fichero ubicado en *tacker/tacker/vnfm/infra_drivers/kubernetes/kubernetes.py* no existe, sino que se llama *kubernetes-driver.py*. Este fallo se descubrió mediante el análisis exhaustivo de los logs que Devstack almacena.

```
  namespace = tacker.vnfm.plugin
  namespace = tacker.vnfm.infra_drivers.openstack.openstack
- namespace = tacker.vnfm.infra_drivers.kubernetes.kubernetes
+ namespace = tacker.vnfm.infra_drivers.kubernetes.kubernetes_driver
  namespace = tacker.vnfm.mgmt_drivers.openwrt.openwrt
```

El proyecto Requirements se ha clonado manualmente debido a que antes de ejecutar Devstack hay que iniciar el environment de Virtualenv mediante el comando

```
virtualenv /opt/stack/requirements/.venv/
```

CONFIGURACIÓN Y EJECUCIÓN

Con el entorno ya preparado para la ejecución se pasa a configurar el fichero *local.conf* para poder implementar la arquitectura que se ha definido. La configuración final queda de la siguiente forma,

```
[[local|localrc]]
# Pip & GIT
# En esta configuracion se establecen diferentes valores para la instalación de
pip y la forma de clonar de los repositorios
PIP_USE_MIRRORS=False
PIP_UPGRADE=False
USE_GET_PIP=1
RECLONE=False
# CREDENCIALES
ADMIN_PASSWORD=password
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
MYSQL_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=supersecrettoken

# LOGGING
LOGFILE=$DEST/logs/stack.sh.log
LOGDAYS=2
# -----

# CONFIGURACIÓN DE SWIFT
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5

# El número de replicas que se configura en Switf, per para entornos de
# desarrollo no se recomienda más de 1, el valor por defecto es de 3 réplicas.
SWIFT_REPLICAS=1
SWIFT_DATA_DIR=$DEST/data

### PLUGIN DE NETWORKING-ODL
enable_plugin networking-odl http://git.openstack.org/openstack/networking-odl
stable/rocky
# Se especifica la versión de ODL
ODL_RELEASE=oxygen-latest

### CARACTERÍSTICAS KARAF
# Para la version de OVSDB de Netvirt
#ODL_NETVIRT_KARAF_FEATURE=odl-neutron-service,odl-restconf-all,odl-aaa-
authn,odl-dlux-core,odl-mdsal-apidocs

#Para la versión VPN Service de Netvirt (Es la nueva)
ODL_NETVIRT_KARAF_FEATURE=odl-restconf-all,odl-aaa-authn,odl-dlux-core,odl-
mdsal-apidocs,odl-netvirt-vpnservice-openstack
ODL_BOOT_WAIT_URL=restconf/operational/network-topology:network-topology/

# Características para la interfaz DLUX
ODL_NETVIRT_KARAF_FEATURE+=,odl-dluxapps-nodes,odl-dluxapps-topology,odl-
dluxapps-yangui,odl-dluxapps-yangvisualizer

ODL_NETVIRT_KARAF_FEATURE+=,odl-l2switch-switch,odl-l2switch-switch-ui,odl-
ovsdb-hwvtepsouthbound-ui,odl-ovsdb-southbound-impl-ui,odl-netvirt-ui
```

```
ODL_NETVIRT_KARAF_FEATURE+=,odl-openflowplugin-flow-services-ui

ODL_NETVIRT_KARAF_FEATURE+=,odl-netvirt-openstack

ODL_NETVIRT_KARAF_FEATURE+=,odl-neutron-logger

# Agente L3 de ODL
ODL_L3=True

# Inicialización de los Bridges de ODL y OpenStack
Q_USE_PUBLIC_VETH=True
Q_PUBLIC_VETH_EX=veth-pub-ex
Q_PUBLIC_VETH_INT=veth-pub-int
ODL_PROVIDER_MAPPINGS=public:${Q_PUBLIC_VETH_INT}

ODL_NETVIRT_DEBUG_LOGS=True

# Esta es la ubicación donde se van a descargar los plugin
DEST=/opt/stack
DATA_DIR=/opt/stack/data

# Servicios de OpenStack necesarios para la integración con ODL
enable_service dstat
enable_service g-api
enable_service g-reg
enable_service key
enable_service mysql
enable_service n-api
enable_service n-cond
enable_service n-cpu
enable_service n-crt
enable_service n-novnc
enable_service n-sch
enable_service placement-api
enable_service placement-client
enable_service neutron-dhcp
enable_service neutron-metadata-agent
enable_service neutron-api
enable_service rabbit

# Se comenta este servicio debido a un error en la ejecución
#enable_service tempest

# No es necesario activar estos servicios para la integración
disable_service c-api
disable_service c-vol
disable_service c-sch

SKIP_EXERCISES=boot_from_volume,bundle,client-env,euca

# Almacenamiento de la ejecución de Devstack
SYSLOG=False
LOGFILE=/opt/stack/new/devstacklog.txt
VERBOSE=True
ENABLE_VERBOSE_LOG_LEVEL=True

HOST_IP=192.168.80.129
FIXED_RANGE=10.1.0.0/20
FLOATING_RANGE=172.24.5.0/24
```

```
PUBLIC_NETWORK_GATEWAY=172.24.5.1
FIXED_NETWORK_SIZE=4096

# Se utiliza el driver de virtualización de Linux libvirt
VIRT_DRIVER=libvirt

export OS_NO_CACHE=1
DATABASE_QUERY_LOGGING=True
EBTABLES_RACE_FIX=True

## SOLUCION AL FALLO DE NEUTRON A LA HORA DE CREAR LAS REDES INICIALES
NEUTRON_CREATE_INITIAL_NETWORKS=False

### CONFIGURACIÓN PARA SFC
# No es necesaria la instalación de OVS debido a que se ha realizado manualmente
SKIP_OVS_INSTALL=True

enable_plugin networking-sfc https://github.com/openstack/networking-sfc.git
stable/rocky

# Se activa la característica Karaf de SFC en el controlador ODL
ODL_NETVIRT_KARAF_FEATURE+=,odl-netvirt-sfc

### OPENSTACK TACKER
Q_PLUGIN=ml2
Q_AGENT=openvswitch

# Se desactivan los grupos de securidad
Q_USE_SECGROUP=False
LIBVIRT_FIREWALL_DRIVER=nova.virt.firewall.NoopFirewallDriver

# SE ACTIVA HEAT, BARBICAN y MISTRAL
enable_plugin heat https://git.openstack.org/openstack/heat stable/rocky
enable_plugin barbican https://git.openstack.org/openstack/barbican stable/rocky
enable_plugin mistral https://git.openstack.org/openstack/mistral stable/rocky

# Ceilometer
enable_plugin ceilometer https://git.openstack.org/openstack/ceilometer
stable/rocky
enable_plugin aodh https://git.openstack.org/openstack/aodh stable/rocky

enable_plugin tacker https://git.openstack.org/openstack/tacker stable/rocky

enable_service n-novnc
enable_service n-cauth

disable_service tempest

# Se desactiva el uso de Kury-Kubernetes
KUBERNETES_VIM=False
#enable_plugin kuryr-kubernetes https://git.openstack.org/openstack/kuryr-
kubernetes stable/rocky

### PLUGIN Q-LBAAS V2
enable_service q-lbaasv2
NEUTRON_LBAAS_SERVICE_PROVIDERV2="LOADBALANCERV2:opendaylight:networking_odl.lba
as.lbaasv2_driver_v2.OpenDaylightLbaasDriverV2:default"
enable_plugin neutron-lbaas https://git.openstack.org/openstack/neutron-lbaas
stable/rocky
```

```
### Solución debido a que la variable Q_PLUGIN_CONF_FILE no estaba iniciada
Q_PLUGIN_CONF_FILE=/etc/neutron/dhcp_agent.ini
[[post-config|$Q_PLUGIN_CONF_FILE]]
[DEFAULT]
enable_isolated_metadata = True

# enable the networking-sfc OpenDaylight driver pair
[[post-config|$NEUTRON_CONF]]
[sfc]
drivers = odl_v2
[flowclassifier]
drivers = odl_v2
```

Con el fichero *local.conf* ya configurado, el siguiente paso es ejecutar Devstack mediante el comando *./stack.sh*.

4.3 Topología de red creada

A partir de la configuración del fichero mostrado previamente se obtiene una topología de red como la que se muestra en la Figura 31, ya que se ha realizado en un único nodo, donde se han creado bridges dentro del sistema, en los cuales se encapsula el tráfico de Management, Data y External.

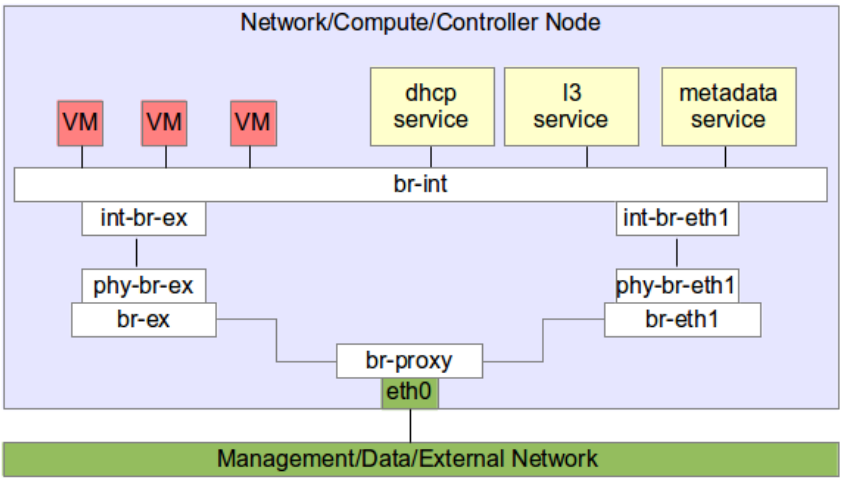

Figura 31: Ejemplo de la topología de red de la infraestructura creada [82]

4.4 Creación de VNF

Una vez construida la arquitectura de Tacker y ODL, se ha realizado una serie de ejecuciones en la línea de comandos mediante el uso de la API de OpenStack para crear los VNFD y lanzar los servicios de red de las VNF. Por otro lado, también se pueden realizar estas instancias mediante la interfaz visual web accediendo desde un navegador en la dirección *192.168.80.129/dashboard* donde se encuentra Horizon con interfaz añadida de Tacker.

Antes de crear los descriptores de los VNF, primero hay que registrar un VIM en Tacker. Pero primero se necesitan las credenciales para que se puedan ejecutar comandos. Se pueden

obtener accediendo a Horizon, en Access & Security y descargando el fichero RC con las claves. Se debe acceder desde el usuario *nfv_user* que crea Tacker el cual está asociado aún proyecto llamado *nfv*. Para registrar el VIM se necesita un fichero YAML *vim_config.yaml* donde se pueden definir el usuario y el proyecto donde se va a registrar.

```
auth_url: 'http://192.168.80.129/identity'
username: 'nfv_user'
password: 'devstack'
project_name: 'nfv'
project_domain_name: 'Default'
user_domain_name: 'Default'
cert_verify: 'False'
```

Con este podemos comenzar a ejecutar comandos de OpenStack, y para registrar el VIM lo hacemos mediante la siguiente instrucción,

```
source nfv_user.sh
openstack vim register --config-file vim_config.yaml --description Default VIM'
--is-default VIM0
```

En primer lugar, se define el VNFD

```
openstack vnf descriptor create --vnfd-file tosca-cirros-vnfd-multi-vdu.yaml
cirros-vnfd-multi-vdu
```

Obteniendo la información resumen del VNFD, y mediante su id podemos proceder a la creación del VNF

```
openstack cnf create --vnfd-id d15225f2-37d6-4ae4-8b76-bf4fa952125a cirros-vnfd-
multi-vdu
```

Comprobar si se ha creado el VNF mediante el uso del siguiente comando o dentro de la interfaz de Horizon

```
openstack vnf list
```

Después se puede proceder con la realización de un ping a la máquina creada para comprobar que está bien conectada a las redes, y que tiene correctamente configuradas las interfaces.

5 Conclusiones y líneas futuras

Para finalizar dicho trabajo se presentan las conclusiones extraídas de la investigación y análisis de las herramientas usadas para desplegar plataformas que otorguen un servicio de IaaS para aplicarse en diferentes soluciones.

5.1 Conclusiones

El aumento de los servicios que requieren de un gran ancho de banda, y baja latencia es cada vez mayor y sobre todo con la llegada de los dispositivos IoT. Las tecnologías analizadas cumplen en la teoría con las especificaciones que se necesitan. Por otro lado, la implementación de éstas, como por ejemplo uso de plataformas de código abierto como OPNFV, ONAP u OpenStack, cumplen también con las capacidades que se les requiere. Además, el uso de este tipo de plataformas reduce gastos y costes económicos y la producción de basura de dispositivos obsoletos, al tener los equipos como switches o routers virtualizados.

En cuanto al despliegue de éstas, no es una tarea sencilla, ya que encapsulan un conjunto de tecnologías y plugin que han de implementarse para poder construir una arquitectura que sea flexible y sencilla de configurarla a posterior. Para ello se dispone de diferentes herramientas de automatización, las cuales ofrecen una documentación extensa a la par que una comunidad activa.

Aun así, exige un entendimiento de cada plugin y driver que hay, para poder realizar una integración completa de OpenStack Tacker y ODL. Es debido a que la herramienta de automatización, Devstack, permite la configuración de escenarios que aún no han sido probados, por lo que no existe una guía de configuración en cada caso.

En definitiva, OpenStack y ODL marcan el camino para poder lograr la configuración y el despliegue de una IaaS. Con esta, se podría crear una arquitectura que fuera capaz de dar servicio a dispositivos IoT y a las nuevas redes 5G. La creciente subida de números de dispositivos, y la evolución de las redes es cada vez más alcanzable gracias a plataformas de código abierto como OpenStack, tanto en entornos de desarrollo como de producción ya que proporciona escalabilidad de los recursos virtualizados en cuanto estos se necesiten cambiando las políticas y el control del tráfico gracias a la programación de las redes por software.

5.1.1 Fallos Encontrados

Cuando se realizó el desarrollo de la instalación del entorno y configuración de la herramienta Devstack se encontraron múltiples fallos. Algunos de estos ya se han comentado previamente, y a continuación se detallan a modo de resumen los fallos que se encontraron y se consideran importantes en cuanto al desarrollo.

Fallos en la Configuración de Devstack

- Error de la instalación de Devstack al no tener cargado el virtualenv. Hay que ejecutar el siguiente comando para cargar el environment. [1]

```
$  virtualenv /opt/stack/requirements/.venv/
```

- Error en el namespace de los kubernetes de tacker en el fichero ubicado en */opt/stack/tacker/etc/config-generator.conf*. [2]

```
- namespace = tacker.vnfm.infra_drivers.kubernetes.kubernetes
+ namespace = tacker.vnfm.infra_drivers.kubernetes.kubernetes_driver
```

- Cambiar el owner de las carpetas que crea en */opt/stack/.cache* ya que al crear la carpeta la crea con usuario de root, y hay que cambiarlo mediante la ejecución de

```
$  sudo chown stack:stack /opt/stack/.cache
```

Fallos en la creación de redes inicial

- Cuando se utiliza el plugin de networking-odl aparece un fallo en la creación de la red inicial, por ello hay que añadir en el fichero de configuración

```
NEUTRON_CREATE_INITIAL_NETWORKS=False
```

Compatibilidad con Python

Uno de los fallos más graves es la compatibilidad con Python 3.X, ya que debido a que inicialmente se realizó Devstack con Python 2.7.X es complicado realizar una instalación con la actual versión de Python. Además, la versión que hay por defecto de la herramienta de control de paquetes Python llamada PIP es una versión antigua, la versión 9, la cual dejará de tener soporte en 2020 al igual que Python 2.7.X. Actualmente dan una solución para algunos módulos que es el uso de Six, un paquete de Python para ejecutar código en Python 3 que este escrito en Python 2. Aun así, los cambios que hay no se aplican en todos los componentes de OpenStack que hay en Devstack.

Problemas con el Idioma

Un fallo a tener en cuenta por el concepto más que por un fallo funcional, es la compatibilidad con otros idiomas. Esto es debido a que OpenStack es compatible con otros idiomas, pero cuando se utiliza la CLI, si el idioma del sistema Linux en el que se está ejecutando es otro diferente al inglés, OpenStack lo utilizará. El problema radica que cuando se ejecutan determinadas ejecuciones muestra mensajes escritos en el idioma que esté configurado y al utilizar una versión de Python 2.7.X la codificación de texto se realiza mediante unicode, en vez de UTF-8. Esto resulta en un error del tipo *TypeError: a bytes-like object is required, not 'str'*, el cual no te ofrece ningún log para saber cuál fue el problema, y te incapacita a ejecutar ciertos comandos. La solución de este fallo solo se puede realizar si se está familiarizado con este tipo de fallos que hay cuando se tienen conocimientos en Python 2.7.X y sabes que es por cuestiones del idioma. El origen se encontraba en el componente de

Keystone, el cual cuando enviaba un mensaje de algún tipo incluía alguna palabra con acentos y saltaba el error.

5.2 Líneas futuras

Dentro del desarrollo e investigación que se han abordado, se han quedado varios aspectos que no se han llegado a desarrollar y se establecen como líneas futuras de investigación.

- La implementación en la topología creada el módulo de OSS/BSS que hay en la arquitectura de ETSI MANO.
- Probar la instalación en múltiples nodos virtualizados, para probar la escalabilidad de OpenStack.
- Instalación en un clúster con diferentes nodos físicos, conectados mediante un switch, instalados con Devstack en un sistema operativo basado en Linux.
- Uso de diferentes herramientas de automatización, como Kolla-Ansible, y Dockers o Kubernetes para realizar la instalación y montar cada servicio embebido en environments aislados dentro de contenedores.

6 Bibliografía

[1] [En línea]. Available: https://ipropertymanagement.com/iot-statistics.

[2] [En línea]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html.

[3] [En línea]. Available: https://iot-analytics.com/top-10-iot-project-application-areas-q3-2016/.

[4] M. Bani Yassein, S. Aljawarneh y A. Al-Sadi. Ras Al, *Challenges and Features of IoT Communications in 5G Networks,* United Arab Emirates, 21-23 de Noviembre 2017.

[5] [En línea]. Available: https://news.strategyanalytics.com/press-release/iot-ecosystem/strategy-analytics-internet-things-now-numbers-22-billion-devices-where.

[6] «IoT Connectivity – Comparing NB-IoT, LTE-M, LoRa, SigFox, and other LPWAN Technologies:,» [En línea]. Available: https://www.iotforall.com/iot-connectivity-comparison-lora-sigfox-rpma-lpwan-technologies/.

[7] «eMTC y NB-IoT abren el camino hacia 5G/IoT,» [En línea]. Available: https://www.rohde-schwarz.com/es/soluciones/test-and-measurement/wireless-communication/wireless-5g-and-cellular/lte-lte-advanced/temas-destacados/emtc-y-nb-iot-abren-el-camino-hacia-5g-iot_230416.html.

[8] «Three 3GPP IoT Technologies To Get Familiar With,» [En línea]. Available: https://www.link-labs.com/blog/lte-iot-technologies.

[9] «What is Narrowband IoT (NB-IoT)? – Explanation and 5 Business Benefits,» [En línea]. Available: https://www.iotforall.com/what-is-narrowband-iot/.

[10] «Introducing 5G networks,» [En línea]. Available: Characteristics and usages: https://www.gemalto.com/mobile/inspired/5G.

[11] X. Foukas, Georgios Patounas, A. Elmokashfi y M. K. Marina, *Network Slicing in 5G: Survey and Challenges,* IEEE Communications Magazine: Electronic ISSN: 1558-1896, 12 de Mayo 2017.

[12] *5G Architecture White Paper: View on 5G Architecture,* 5GPPP Architecture Working Group, Diciembre 2017.

[13] A. Galis y K. Makhijani., *Network Slicing Landscape: A holistic architectural approach, orchestration and management with applicability in mobile and fixed networks and clouds,* Montreal, Canada, 25-29 de Junio 2018.

[14] [En línea]. Available: http://www.it.uc3m.es/wnl/5gnorma/.

[15] «5gex,» [En línea]. Available: http://www.5gex.eu/.

[16] «5ginfire,» [En línea]. Available: https://5ginfire.eu/.

[17] «pagoda,» [En línea]. Available: https://5g-pagoda.aalto.fi/.

[18] *Network Functions Virtualisation White Paper #3,* ETSI, Oct. 2014.

[19] «OpenFlow,» [En línea]. Available: https://es.wikipedia.org/wiki/OpenFlow.

[20] «researchgate,» [En línea]. Available: https://www.researchgate.net/figure/High-level-conceptual-architecture-of-SDN_fig1_271132741.

[21] *Applying SDN Architecture to 5G Slicing,* ONF.

[22] [En línea]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[23] *Network Functions Virtualisation (NFV); Architectural Framework,* ETSI GS NFV 002 V1.1.1., Octubre 2013.

[24] *Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework.*

[25] «What is NFV MANO?,» [En línea]. Available: https://www.sdxcentral.com/nfv/definitions/nfv-mano/.

[26] [En línea]. Available: https://es.wikipedia.org/wiki/Network_Information_Center.

[27] *Network Function Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework,* Dec. 2015.

[28] *Network Slicing for 5G with SDN-NFV Concepts, Architectures and Challenges.*

[29] *Network Functions Virtualisation (NFV); Infrastructure Overview,* Jan. 2015.

[30] J. O. Lucena, P. Ameigeiras, D. Lopez, J. J. R. Muñoz, J. Lorca y J. Folgueira., *Network Slicing for 5G with SDN/NFV: Concepts, Architectures and Challenges,* Granada: Universidad de Granada, 2017.

[31] *Network Configuration Protocol (NETCONF),* RFC 6241.

[32] *Description of Network Slicing Concept.,* NGMN Alliance. , 30 de Agosto de 2019.

[33] *View on 5G Architecture (Version 2.0),* 30 de Agosto de 2019..

[34] X. Foukas, G. Patounas, A. Elmokashfi y M. K. Marina, *Network Slicing in 5G: Survey and Challenges,* IEEE Communications Magazine, 2017.

[35] T. Irshad, *Design and Implementation of a Testbed for Network Slicing,* Aalto University, 12 de Febrero 2018.

[36] Z. Kotulski, T. W. Nowak, M. Sepczuk, M. Tunia, R. Artych, K. Bocianiak, T. Osko y J. Wary., *Towards constructive approach to end-to-end slice isolation in 5G networks,* EURASIP Journal on Information Security.

[37] *Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges and Call for Action,* ETSI, 2012.

[38] M. K. F. W. H. W. D. J. A. C. 2. R. Szabó, *Elastic Network Functions: Opportunities and Challenges,* IEEE Network June, 2015.

[39] [En línea]. Available: https://docs.openstack.org/project-team-guide/introduction.html.

[40] [En línea]. Available: http://www.tid.es/es/innovacion-de-largo-plazo/network-innovation/telefonica-nfv-reference-lab/openmano.

[41] [En línea]. Available: https://openbaton.github.io/.

[42] [En línea]. Available: https://www.opnfv.org/.

[43] [En línea]. Available: https://cloudbase.it/v-magine/.

[44] [En línea]. Available: https://www.stackalytics.com/?release=rocky.

[45] [En línea]. Available: https://ubuntu.com/blog/what-it-directors-should-know-about-the-economics-of-openstack.

[46] [En línea]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Scope_and_FunctionalityArquitectura.

[47] [En línea]. Available: https://www.telefonica.com/es/web/sala-de-prensa/detalle-noticia/-/asset_publisher/O34kxJNk5Exu/content/telefonica-se-une-a-otras-22-operadoras-y-proveedores-de-equipos-para-crear-una-comunidad-global-de-open-source-mano-osm-.

[48] [En línea]. Available: https://en.wikipedia.org/wiki/ONAP.

[49] [En línea]. Available: https://www.aarnanetworks.com/post/2017/10/24/understanding-onap.

[50] [En línea]. Available: https://www.onap.org/architecture.

[51] [En línea]. Available: https://www.onap.org/architecture.

[52] [En línea]. Available: https://www.opnfv.org/blog/2014/09/30/enabling-the-transition-introducing-opnfv-an-integral-step-towards-nfv-adoption.

[53] [En línea]. Available: https://wiki.opnfv.org/download/attachments/6819410/OPENO-OPNFV-MANO-Integration%20Requirments.pptx?version=1&modificationDate=1466627924000&api=v2.

[54] C. G.A., P. M., M. T., C. M., B. P. y F. L., *Prototyping nfvbased multi-access edge computing in 5G ready networks with open baton.,* IEEE Conference, 2017.

[55] [En línea]. Available: https://specs.openstack.org/openstack/tacker-specs/specs/mitaka/multi-site-feature.html.

[56] [En línea]. Available: https://www.openstack.org/software/project-navigator/openstack-components#openstack-services.

[57] [En línea]. Available: https://www.extremenetworks.com/extreme-networks-blog/creating-possibilities-and-enabling-virtualization-with-openstack/.

[58] [En línea]. Available: https://docs.openstack.org/networking-sfc/latest/.

[59] P. Quinn, *Service Function Chaining,* Internet-Draft Cisco, May 7, 2018.

[60] D. Kushwaha, B. Thiruveedula, T. Somanchi y S. Ramaswamy, *OpenStack Collaboration Made in Heaven,* VERIZON INDIA;NEC;NXP;CISCO;Tacker;Ex-PTL, 06.11.2017.

[61] [En línea]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:MD-SAL_Document_Review:Architecture#OpenDaylight_YANG_Models.

[62] [En línea]. Available: https://en.wikipedia.org/wiki/Model-driven_engineering.

[63] [En línea]. Available: https://blogs.cisco.com/getyourbuildon/yang-opensource-tools-for-data-modeling-driven-management.

[64] A. H. R. Sieg, *Facilitation of The OpenDaylight Architecture,* Sankt Augustin, Germany: University of Applied Sciences.

[65] *YANG - A Data Modeling Language for the Network Configuration Protocol.*

[66] [En línea]. Available: https://wiki.openstack.org/wiki/Neutron/ML2.

[67] S. Hague, I. Yamahata y A. Fredette, *Openstack with OpenDaylight (A Workshop),* India: Red Hat;OpenDaylight Forum, 2016.

[68] [En línea]. Available: https://docs.openstack.org/newton/networking-guide/config-ml2.html.

[69] [En línea]. Available: https://wiki.opendaylight.org/view/NetVirt#Documentation.

[70] [En línea]. Available: https://blogs.vmware.com/latam/2017/05/hardware-vtep-vmware-nsx.html.

[71] [En línea]. Available: https://wiki.openstack.org/wiki/DevStack .

[72] [En línea]. Available: https://docs.openstack.org/kolla-ansible/rocky/user/quickstart.html.

[73] [En línea]. Available: https://docs.openstack.org/kolla-ansible/rocky/reference/networking-guide.html.

[74] [En línea]. Available: https://docs.openstack.org/tacker/rocky/install/kolla.html.

[75] [En línea]. Available: https://wiki.openstack.org/wiki/Fuel.

[76] [En línea]. Available: https://cloudbase.it/v-magine/.

[77] [En línea]. Available: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main#Building_Open_vSwitch_with_VxLAN-GPE_and_NSH_support.

[78] [En línea]. Available: https://patchwork.openembedded.org/patch/158157/.

[79] [En línea]. Available: https://github.com/guard/listen/wiki/Increasing-the-amount-of-inotify-watchers.

[80] [En línea]. Available: https://bugs.launchpad.net/networking-odl/+bug/1822559.

[81] [En línea]. Available: https://docs.openstack.org/devstack/latest/.

[82] [En línea]. Available: https://fosskb.in/2014/06/10/managing-openstack-internaldataexternal-network-in-one-interface/.

[83] [En línea]. Available: https://www.dangtrinh.com/2018/07/got-optstackrequirementsvenvbinpip-no.html.

[84] [En línea]. Available: https://opendev.org/openstack/tacker/commit/eab8c1b71fcf9e05140d2d93f2f508f14d34f6d7?lang=lv-LV.

[85] «Portal web de la ETSIT UPM,» [En línea]. Available: http://www.etsit.upm.es/.

[86] B. Naudts, M. Kind, S. Verbrugge1, D. Colle y M. Pickavet, *How Can a Mobile Service Provider Reduce Costs with Software-Defined Networking?*.

7 Anexo A: Aspectos éticos, económicos, sociales y ambientales

7.1 A.1 Introducción

Dentro de las tecnologías que hay en una Infraestructura como servicio, se encuentran el SDN y el NFV, estas son claves para su uso en numerosos servicios tales como 5G o IoT. El uso de estas tecnologías engloba múltiples beneficios tanto económicos a los proveedores de quienes las utilizan como medioambientales de manera global. De esta forma se van a analizar el impacto económico en el uso de las tecnologías SDN y NFV, que se implementan en plataformas de código abierto, y el impacto ambiental de la transformación física a virtual que está realizando.

7.2 A.2 Descripción de impactos relevantes relacionados con el proyecto

El desarrollo de las infraestructuras desplegadas en la nube con sistemas virtualizadas habilita el uso de las IaaS de una manera más flexible y sencilla que tradicionalmente. El uso de tecnologías como NFV y SDN permiten esta virtualización de los recursos y habilita la programabilidad de las redes. Poder contar con estas tecnologías proporciona una reducción significativa de los gastos operativo y capitales que tiene que realizar un CPD. Se debe a que en los CPD tradicionales cada dispositivo esta enlazado con el software único del proveedor, el cual desemboca en realizar muchos gastos siempre que se necesite actualizar el sistema o la infraestructura, teniendo que recurrir también a expertos para realizar estas tareas. Además, desemboca en un uso incrementado de equipos físicos que cuando su ciclo de vida llega a su fin el hardware no se puede aprovechar o reciclar.

7.3 A.3 Análisis detallado de alguno de los principales impactos

El impacto más relevante es el económico debido a que las tecnologías descritas en una IaaS reducen significativamente los costes CAPEX y OPEX. Los costes CAPEX son aquellos gastos e inversiones que desembolsa la empresa para la obtención de bienes físicos, y en este caso también el software y la instalación de este [86]. Y el OPEX es el gasto operativo con todos los servicios como el coste de mantenimiento, reparación, energía, provisión del servicio y administración de este. El uso de SDN reduce el CAPEX debido a que el control de plano de los dispositivos físicos y pasa al ser controlado por software. Pero no influye el coste de este software debido a la existencia de controladores SDN de código abierto, como por ejemplo ODL. Las siguientes características que involucra el CAPEX con las tecnologías de SDN y NFV son,

- Componentes, el cual se separa en número de dispositivos desplegados por el coste por dispositivo

- Componentes SDN, números de controladores SDN por el coste del controlador
- Coste de instalación, es el coste del desarrollo de software y de la primera instalación

Por otro lado, el OPEX, se tienen en cuenta el coste continuo para mantener la infraestructura y la energía que necesita para funcionar que ambos se pueden asociar a la superficie donde están los equipos instalados. Es notable el ahorro del gasto de energía en cuanto se utilizan tecnologías SDN y NFV, debido a que la superficie que ocupa un switch virtual u otros recursos son menores en comparación a los que pueden ocupar un rack de servidores virtualizando una mayor cantidad de estos switches. Y además el costo de mantenimiento también se ve reducido debido a que habrá menos equipos que reparar y el software será común. Otra de las consideraciones a tener en cuenta es el gasto que se produce en caso de caída de servicio, en el cual tiene mucha relevancia el tiempo que transcurre entre que se detecta y se soluciona, viéndose reducido en el caso de tecnologías como SDN o NFV. Los costes asociados a OPEX se pueden separar en,

- Coste continuo de infraestructura, gasto energético los racks y coste de estos
- Mantenimiento y reparación, se mide en gasto en realizar un recambio más los gastos asociados a la resolución de un fallo de hardware o software, entre estos están los tiempos asociados a resolver el fallo
- Aprovisionamiento de servicios, es el gasto que consume el tiempo que se tarde en levantar las conexiones
- Mantenimiento de servicios, es el gasto que consume el tiempo que tarda en reparar las conexiones

A parte del entorno económico también existe un impacto medioambiental, debido a que supone un ahorro en generación de equipos físicos notable.

7.4 A.4 Conclusiones

En definitiva, el creciente interés que ha habido en estas tecnologías ha habilitado la creación acelerada de plataformas abiertas y código de libre uso, ya sea por parte de empresas que realizan inversiones o desarrolladores independientes que participan en la comunidad. Creando una dependencia buena de estas plataformas en multitud de empresas, ya que al participar en estas tecnologías terminan formando parte de sus proyectos. Finalmente se puede concluir que la investigación y aplicación de tecnologías como NFV, SDN y aplicarla en infraestructuras como las mostradas conlleva a una reducción significativa en gastos y en desperdicio de equipos físicos que pueden contaminar.

8 Anexo B: Presupuesto económico

Según diferentes plataformas de trabajo, un Desarrollador Senior de OpenStack cobra de media en torno a 32.000€ netos al año, aplicando entre un 16% y un 20% de retención sobre el bruto. Y trabajando una media de 5 horas diarias durante 6 meses, queda un presupuesto de la siguiente forma.

VALORES	*Watts del PC*	*Coste/kW día*	*Horas/día*	*Días/mes*	*Meses*
	120	0,1143 €	5	20	6

COSTE DE MANO DE OBRA (Coste Directo)	*Neto/año*	*Precio/hora*	*Horas totales*	TOTAL
	32.000,00 €	27 €	600	16.000 €

COSTE DE RECURSOS MATERIALES (Coste Directo)	***Precio/mes***	***Precio/compra***		***Amortiz.(años)***		***TOTAL***
Ordenador personal (Software incluido)		1.200,00 €		5		120,00 €
Gastos de electricidad	12,00 €					72,00 €
Conexión Internet	23,60 €					141,60 €
TOTAL						333,60 €

GASTOS GENERALES (Costes Indirectos)	15%	sobre CD				2.450,04 €
BENEFICIO INDUSTRIAL	6%	sobre CD+CI				1.127,02 €

SUBTOTAL PRESUPUESTO		19.910,66 €
IVA APLICABLE	21%	4.181,24 €

TOTAL PRESUPUESTO	24.091,90 €