Universidad Zaragoza
1542

Trabajo Fin de Grado

Estudio de la virtualización de funciones de red (NFV) y aplicación del encadenado de funciones (SFC) para un diseño flexible de red

Study of Network Function Virtualization (NFV) and application of Service Function Chainning (SFC) for a flexible network design

Autor

Asier Carbonel Martínez

Director/es

María Canales Compés

Escuela de Ingeniería y Arquitectura (EINA) / Universidad de Zaragoza

2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Asier Carbonel Martínez,

con nº de DNI 78777317V en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Estudio de la virtualización de funciones de red (NFV) y aplicación del encadenado de funciones (SFC) para un diseño flexible de red,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de Noviembre de 2018

Fdo: Asier Carbonel Martínez

Resumen

El rápido crecimiento de las redes de telecomunicación y los cada vez más demandados servicios de red por parte de los usuarios, está provocando que el desarrollo y la innovación en el sector IT suponga la aparición de grandes retos para los principales operadores de red.

Aunque las redes hayan evolucionado de manera exponencial desde sus inicios, la aparición de nuevas necesidades de ancho de banda, flexibilidad o servicios de red necesarios, han hecho a los desarrolladores plantearse nuevas maneras de reinventar el concepto de redes de telecomunicaciones.

Así nace la virtualización de funciones de red (NFV) y las redes definidas por *software* (SDN), ambas tecnologías proporcionan soluciones a los problemas comentados anteriormente.

NFV proporciona soluciones al rápido crecimiento de las redes de telecomunicaciones y a la escasa flexibilidad de adaptación ante nuevas demandas de servicios en redes cuya arquitectura ya está definida facilitando la arquitectura suficiente para el despliegue de funciones de red virtuales en equipos genéricos. Mientras que SDN permite tener el control total del tráfico cursado en una red mediante *software,* esto hace mucho más eficiente el desarrollo de la Ingeniería de Tráfico. SDN permite desacoplar el plano de datos del plano de control, este estará controlado por una aplicación *software* capaz de decidir de manera dinámica el funcionamiento de la red.

Por último, todo tráfico extremo a extremo necesita atravesar varias funciones de red antes de llegar a su destino, sin embargo, no todo el tráfico es igual. Para suplir esta necesidad surge el encaminamiento de funciones de servicio (SFC), este permite crear cadenas de funciones de servicio diferenciando cada tipo de tráfico y haciendo que cada uno viaje a través de la cadena que más se adapte a sus necesidades.

En este proyecto se evalúa Openstack como herramienta *cloud* NFV junto con OpenDayLight como herramienta SDN. En primer lugar, se estudian los principios de la virtualización de funciones de red. A continuación, se evalúan diferentes escenarios de red, cada uno con diferentes funcionalidades. Se despliega un escenario SFC para el cual se analiza las características principales del tráfico en una red de dominio SFC.

Para concluir, se analiza el alcance de NFV, SDN y SFC en la evolución de las redes actuales y se indican posibles trabajos futuros relacionadas con las estas tecnologías.

Agradecimientos

En primer lugar, agradezco a mis padres todo el apoyo y mensajes de ánimo que me han transmitido a lo largo de todo el grado para poder llegar hasta aquí.

En segundo lugar, agradezco y dedico este trabajo de fin de grado a todos los profesores que me han formado a lo largo de estos años. En especial, a mi tutora María Canales por todo el esfuerzo dedicado estos meses durante la realización de este proyecto.

En tercer lugar, agradezco a Manuel Buil, responsable del proyecto SFC en OPNFV, toda su ayuda y paciencia.

Por último, agradezco a mis amigos y compañeros que han estado a mi lado en todo momento. A mis amigos de Tudela y a los que han aparecido estos años en Zaragoza, especialmente a Alfonso Cay, un gran amigo y compañero a lo largo de estos años.

Tabla de contenido

Tabla de figuras

Lista de acrónimos

API	Application Programming Interface
BGP	Border Gateway Protocol
CLI	Command line interface
CPU	Central processing unit
DPI	Deep Packet Inspection
ETSI	European Telecommunications Standards Institute
E2E	End to end
GUI	Graphical user interface
HOT	Heat Orchestration Template
HTTP	Hypertext Transfer Protocol
IAAS	Infrastructure as a Service
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IRC	Internet Relay Chat
IT	Information Technology
KVM	Kernel-based Virtual Machine
LACP	Link Aggregation Control Protocol
LSP	Language Service Provider
LXC	Linux Containers
MANO	Management and Orchestration
MPLS	Multiprotocol Label Switching
NETCONF	Network Configuration Protocol
NFV	Network functions Virtualization
NFVI	Network functions Virtualization Infrastructure
NSH	Network Service Header
OAM	Operations, Administration and Management
ODL	OpenDayLight
ONOS	Open Network Operating System
OPNFV	Open Platform for NFV

OSA	OpenStack Ansible
OSM	Open Source MANO
OSS/BSS	Operations Support Systems / Business Support Systems
OVN	Open Virtual Network
OVS	Open vSwitch
OVSDB	Open vSwitch Database
PCE	Path Computation Element
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request for Comments
RSA	Rivest, Shamir y Adleman
SBC	Session Border Controller
SDN	Software Defined Networking
SF	Service Function
SFC	Service Function Chaining
SFF	Service Function Forwarder
SFP	Service Function Path
SI	Service Index
SPI	Sevice Path Identifier
SSD	Solid State drive
SSH	Secure Shell
TOSCA	Topology and Orchestration Specification for Cloud Applications
TTL	Time to Live
UI	User Interface
VLAN	Virtual Local Area Network
VNF	virtual network functions
XCI	Cross Community Continuous Integration

1. Introducción

El modelo actual de las redes de comunicaciones está muy limitado a causa de diferentes factores tales como altos costes energéticos, tamaño y arquitectura de la topología de red existente, grandes inversiones de capital en investigación o el elevado precio de la infraestructura de red necesaria para desplegar nuevos servicios e incluso la dificultad de integrarlos y operar con la arquitectura de red existente.

Por otro lado, los equipos de red que existen actualmente tienen una gran dependencia con *hardware* propietario, cuyo ciclo de vida está acotado. Esto hace que en muchas ocasiones el avance de la tecnología obligue a los operadores a renovar los equipos *hardware* debido a que estos se han quedado obsoletos.

Esto ha hecho que los principales operadores de red se planteen alternativas para poder dar soporte de manera eficiente a las actuales necesidades que surgen en las redes que gestionan. Las tecnologías de virtualización junto con la necesidad de desacoplar el control del tráfico y el propio flujo de datos han incentivado el desarrollo de nuevas soluciones basadas en la programabilidad de la red: virtualización de funciones de red y desarrollo de redes definidas por *software*.

La virtualización de funciones de red (*Network Functions Virtualisation*, NFV) tiene como objetivo convertirse en un estándar para resolver todos estos problemas y sustentar el futuro de las redes de comunicaciones. Aprovechando las actuales herramientas de virtualización, se pretende llevar todas aquellas funciones de red que actualmente tienen que estar físicamente en instalaciones como empresas a lo que se denomina "la nube". El principal objetivo es virtualizar, orquestar y gestionar todas aquellas funciones de red (*router*, *switch*, *firewall*, *Session Border Controller* (SBC*), Deep packet inspection* (DPI), etc.) de las que se componen las redes actuales de comunicación.

Otra tecnología que está ganando mucho peso en el sector de las tecnologías de la información, o en inglés *Information technology* (IT), son las redes definidas por *software*, sus siglas en ingles SDN (*Software Defined Network)*. Su principal objetivo es crear redes cuyo funcionamiento sea programable con una gestión centralizada en un elemento de red denominado "controlador". Las redes definidas por *software* permiten separar el plano de control del plano de datos, facilitando y simplificando la configuración, gestión y mantenimiento de la red, la cual se realiza mediante *software*.

Es posible aprovechar todas las fortalezas de SDN en NFV para la gestión de las funciones de red virtuales creadas mediante NFV. Ambas tecnologías están muy relacionadas y actualmente se está apostando fuertemente por su integración. Aunque ambas pueden trabajar de manera independiente, su sinergia puede significar una revolución en las redes de comunicaciones como hoy en día las conocemos.

Partiendo del panorama actual y de las tecnologías anteriormente citadas, una nueva metodología llamada encaminamiento de funciones de servicio, o en inglés *Service Function Chaining* (SFC), está ganando mucho peso en la manera de gestionar el tráfico de red. Gracias a SFC es posible dejar atrás las técnicas

tradicionales de encaminamiento y poder decidir de manera dinámica y eficiente las funciones de red que van a atravesar los diferentes tipos de tráfico en nuestra red. Gracias a esto es posible diferencias diferentes flujos de datos, analizar sus requisitos tanto de enlace como de servicios intermedios y poder tratarlo adecuadamente. Esto permitirá a los operadores dar servicios más personalizados a los clientes y conseguir mayor calidad de servicio.

En este contexto, este proyecto analizará el potencial de NFV y su aplicación al encadenado de funciones, aspecto el que, además, se aprovechará la sinergia con SDN. El trabajo se desarrolla en el contexto de investigación del grupo CeNIT (*Communication Networks and Information Technologies Group*) de la Universidad de Zaragoza, una de cuyas líneas de investigación aborda la *sofwarización* y virtualización de funciones de red.

1.1. Motivación

Como ya se ha comentado, el constante cambio e innovación de las redes de comunicaciones nos abre a los ingenieros de telecomunicación un gran abanico de posibilidades en este sector.

La evolución tecnológica mediante NFV y SDN cambia por completo el paradigma de las redes, la manera de configurarlas y de gestionarlas. Como ejemplo, la apuesta hacia la integración de estas tecnologías en la nueva generación de redes móviles (5G) [12]. Del mismo modo, en la actual era de la información y del *big data*, los principales proveedores de estos servicios están apostando fuertemente por implantar estas tecnologías en sus centros de datos para mejorar la eficiencia de sus operaciones. Así, resulta de gran interés realizar estudios acerca de su flexibilidad y su alcance.

Es muy probable que, dentro de unos años, estas tecnologías estén implantadas como están a día de hoy las tecnologías tradicionales de red, por lo que considero que es una buena oportunidad para aprender y avanzar más allá en la formación que se nos ofrece en la rama de telemática.

1.2. Objetivos y metodología

El **objetivo principal** del proyecto es estudiar y analizar la flexibilidad que nos brinda la virtualización de funciones de red (NFV) y, concretamente, la capacidad de crear diferentes cadenas de servicios (SFC) para la gestión de diferentes tipos de tráfico.

Mediante un entorno de virtualización en la nube (*cloud*) se pretende desplegar un escenario de red controlado que nos permita evaluar dichas tecnologías y estudiar su alcance.

Se estudiará el estado actual de NFV y SFC, los componentes que lo forman y las herramientas necesarias para su implementación.

De acuerdo a este objetivo general, se han establecido estos **objetivos parciales**, de acuerdo a los mismos se han desarrollado las distintas etapas del trabajo con la **metodología** descrita:

1) Como primer objetivo se establece la necesidad de estudiar el **estado del arte** relacionado con las tecnologías objeto del trabajo: NFV y SDN para identificar los aspectos particulares en los que se profundizará (como SFC) y las herramientas utilizadas para realizar un despliegue controlado.
2) Identificadas las **herramientas de trabajo**, se elaborará un estudio detallado de las mismas para poder realizar los despliegues de laboratorio a estudiar facilitando, por una parte, el análisis de los escenarios particulares, y por otra, documentando el trabajo como base a futuros estudios. Como punto de partida, se contará con el entorno de trabajo OPNFV [50] [9] [5], basado en despliegues en la nube mediante Openstack. Así pues, un objetivo clave del trabajo será la comprensión de dicho entorno y, especialmente de Openstack, lo que incluye familiarizarse con el intérprete de comandos (CLI) o la interfaz gráfica (GUI), entender los módulos que contiene la instalación y lo más importante, saber analizarlos para solucionar posibles errores inesperados.
3) El siguiente objetivo, centrado en el análisis específico de NFV, es el **desarrollo técnico de una primera propuesta de trabajo**: se desplegará un pequeño escenario de red básico que permita entender y evaluar el correcto funcionamiento de la virtualización de funciones de red (NFV).
4) Por último, y en línea con el objetivo final del proyecto, se desplegará un **escenario de red incorporando encadenado de funciones de servicio, SFC**. En este caso, el objetivo se centrará en el análisis particular de la capacidad de NFV para proporcionar una solución flexible de SFC.

A la hora de abordar los objetivos tecnológicos descritos previamente se requiere no solo del *software* necesario sino del *hardware* adecuado para realizar los despliegues, atendiendo a las especificaciones de la propia plataforma de trabajo (OPNFV). En este sentido, en el grupo de trabajo CeNIT se dispone de un servidor en la universidad con recursos suficientes, en el que estará alojado el entorno *cloud* y al que se permite el acceso remoto mediante SSH.

Para llevar a cabo el proyecto, serán necesarios adicionalmente conocimientos en *Shell scripting* y Python para facilitar la creación de *scripts* de configuración automática y generación de tráfico en la red con el que automatizar el despliegue y estudio de los escenarios.

En la Figura 1 se ve el avance temporal de los objetivos parciales del proyecto.

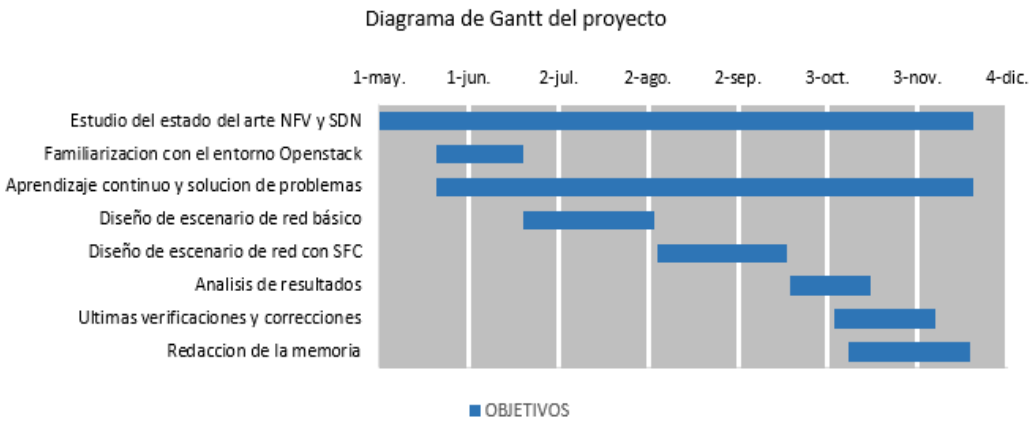


Figura 1: Diagrama de Gantt del Proyecto

2. Estado del arte

En este capítulo se expondrán de manera teórica las bases de la virtualización de funciones de red, los principios del encadenado de funciones de servicio y las principales herramientas existentes hoy en día que nos permiten implementar esta tecnología.

2.1. Virtualización de funciones de red (NFV)

A menudo, cuando se piensa en despliegues de red tradicionales, se piensa en numerosos dispositivos *hardware* dedicados para cumplir con las funciones de la red a desplegar. Estos dispositivos están directamente atados a *hardware* propietario.

NFV permite desacoplar las funciones de red de los dispositivos *hardware* propietarios y trasladarlas a servidores virtualizados. De esta manera, se pretende reducir costes en los equipos más cercanos al cliente e incluso costes de instalación, mantenimiento y servicio técnico.

Una de las principales ventajas que ofrece la virtualización de funciones de red es la flexibilidad que ofrece a los operadores de red de gestionar los servicios ofrecidos. Mediante NFV los administradores de red son capaces de orquestar funciones de red de manera dinámica para dar soporte a las necesidades del cliente sin la necesidad de modificar el *hardware* de las instalaciones del cliente, pudiéndose adaptar completamente a las necesidades de la red.

Como la gran mayoría de estándares y proyectos importantes de telecomunicaciones NFV está regulado por la ETSI [15]. De acuerdo a las especificaciones de dicho organismo, y en concreto del grupo de trabajo de NFV, el desarrollo de NFV, junto con redes definidas por *software* (SDN), se basa en 3 criterios clave [1].

- Desacoplo del *software* y *hardware* propietarios
- Despliegue flexible de funciones de red
- Gestión dinámica

Si bien NFV nació en 2012 [2], y desde entonces el proceso de estandarización ha madurado, no deja de ser todavía una tecnología en desarrollo. Aunque comienzan a aparecer pruebas de concepto y de interoperabilidad, su implementación práctica todavía está lejos de ofrecer soluciones directas al usuario.

2.1.1. Arquitectura

A continuación, se va a detallar la estructura básica con sus principales dominios de trabajo tal y como los define la ETSI [1].

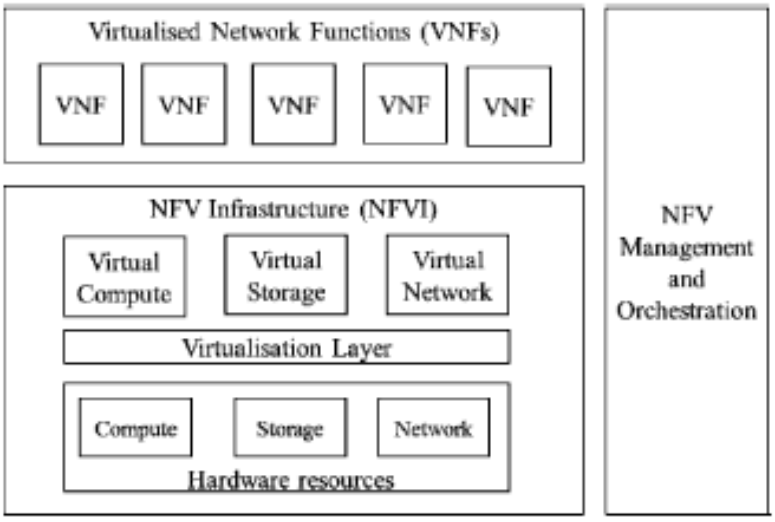


Figura 2: Arquitectura NFV según ETSI [1]

ETSI establece una arquitectura de referencia (Figura 2) en la que se definen tres bloques estructurales importantes:

- *Virtualised Network Function (VNF)*, Función de red virtualizada: implementación *software* de todas las funciones de red (*firewall*, *vRouter*, balanceadores de carga…) que son capaces de ejecutarse sobre NFVI.
- *NFV Infrastructure (NFVI)*, Infraestructura NFV*:* incluye toda la diversidad de recursos físicos disponibles en la máquina y cómo se pueden virtualizar. Mediante la capa de virtualización, gestiona la relación entre los recursos físicos y los recursos virtualizados. NFVI soporta la ejecución de las VNF.
- *NFV Management and Orchestration (MANO)*, Orquestación y gestión de NFV*:* se encarga de la organización y la gestión de los recursos físicos y/o *software* que permiten la virtualización de la infraestructura. También se encarga de la gestión del ciclo de vida de las VNF.
  MANO se centra en todas las tareas de administración específicas de la virtualización necesarias en el marco de NFV. Se considera el núcleo de la arquitectura y es el encargado de interactuar con el resto de bloques.

De acuerdo con esta descripción funcional de los componentes principales se define una arquitectura más detallada a bajo nivel para NFV (Figura 3):

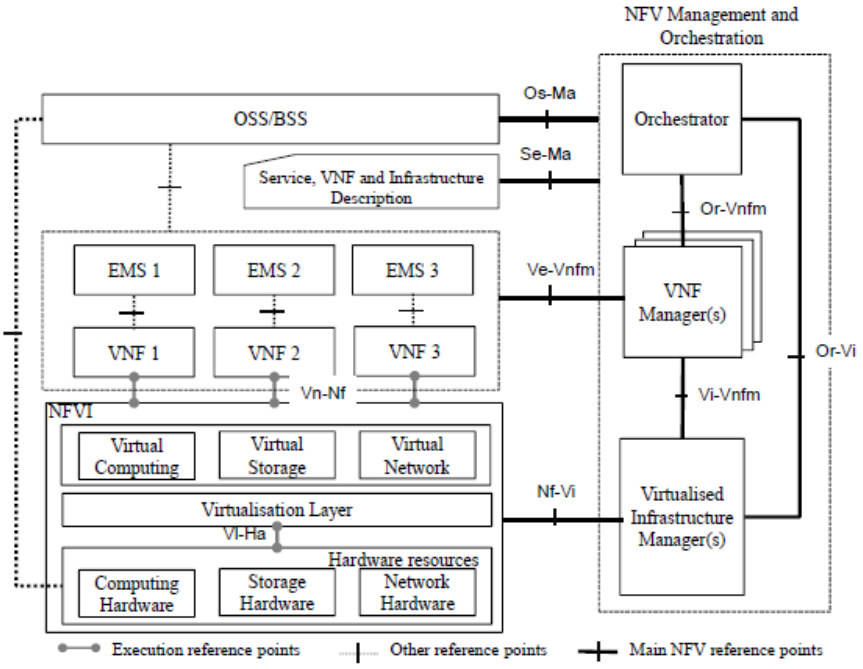


Figura 3: Arquitectura de referencia NFV según ETSI [1]

– NFV-MANO

Orchestrator (orquestador)*:* es el encargado de la orquestación y de la gestión de la infraestructura NFV. Gestiona los recursos de almacenamiento, cómputo y de red a través del VIM. También se encarga de los servicios de red.

- *VNF Manager* (gestor de VNF)*:* es el encargado del ciclo de vida de las VNF (creación, actualización, monitorización, escalado y destrucción).
- *Virtualised Infrastructure Manager (VIM)* (gestor de la infraestructura de virtualización)*:* se encarga de gestionar y administrar la interacción de las VNF con los recursos de computación, almacenamiento y red bajo su domino.
  Se encarga de desplegar las máquinas virtuales a partir de los recursos *hardware* disponibles, su aprovisionamiento y conexionado de red.

– OSS-BSS

Corresponde con las siglas de *Operations Support Systems and Business Support Systems*. Comprende los sistemas de gestión de cada empresa.

– SERVICE, VNF AND INFRASTRUCTURE DESCRIPTION
  Contiene los modelos de datos, plantillas de implementación de las VNF, y de los VNF *Forwarding Graph*, relacionados con la funcionalidad SFC detallada posteriormente[1].

– VNF BLOCK
  - EMS: Son las siglas de *Element Management System*. Se encarga de las funciones de monitorización y gestión de las VNF a nivel individual. Para ello se comunica con el VNF Manager. También proporciona datos acerca de las VNF al bloque OSS/ BSS.

2.1.2. Ventajas NFV

NFV ofrece numerosos beneficios a los operadores, ayudando a cambiar radicalmente la situación en la industria de las telecomunicaciones. Lo beneficios que ofrece NFV según la ETSI [2] son los siguientes:

– Reduce los costes de los equipos y disminuye los gastos energéticos a través de la fusión de equipos y la explotación de las economías de escala de la industria IT.
– Permite aumentar la velocidad del tiempo de comercialización minimizando el ciclo de innovación del operador de red. Las economías de escala tradicionales para los equipos basados en *hardware* ya no son aplicables a las soluciones de red basadas en *software.* NFV debería permitir a los operadores reducir los ciclos de maduración.
– Posibilita realizar ensayos y producir en la misma infraestructura: facilita ensayos mucho más eficientes y mayor integración, reduciendo los costes en desarrollo y el tiempo de creación.
– Permite establecer servicios basados en la situación geográfica. Los servicios se pueden escalar rápidamente y pueden ser aprovisionados rápidamente mediante *software* sin necesidad de instalar nuevo *software*.
– Fomenta la diversidad de ecosistemas. Abre un mercado de dispositivos virtuales a los creadores de *software*, o para uso académico, creando nuevas fuentes de negocio con menor riesgo.
– Permite la optimización de la configuración y/o de la topología de red en tiempo real basada en la demanda de tráfico actual.
– Facilita dar soporte a múltiples clientes ofreciéndoles diferentes servicios sobre el mismo *hardware* manteniendo una separación segura de dominios administrativos.
– Permite reducir costes energéticos explotando las características energéticas de los servidores y sistemas de almacenamiento estándar.

[1] La función *VNF Forwarding Graph* se utiliza para organizar y administrar el tráfico a través de VNF. En resumen, las definiciones abstractas de VNFFG se representan en cadenas de funciones de servicio (SFC) y clasificadores. El SFC conforma una lista ordenada de VNF para que el tráfico atraviese, mientras que el clasificador decide qué tráfico debe pasar a través de ellos. (extracto de https://docs.openstack.org/tacker/latest/user/vnffg_usage_guide.html, acceso 22/11/2018)

2.2. *Service Function Chaining* (SFC)

Normalmente, todo tráfico extremo a extremo (*end-to-end*, E2E) requiere atravesar varias funciones de servicio intermedias como por ejemplo *firewall*, balanceadores de carga, etc. A esto se le denomina cadena de servicios. "*Definir e instanciar un conjunto ordenado de funciones de servicio y el posterior direccionamiento del tráfico a través de ellas es realizar el encadenado de funciones, o Service Function Chaining"* [22]

Un ejemplo sencillo de aplicación SFC se puede observar en la Figura 4.

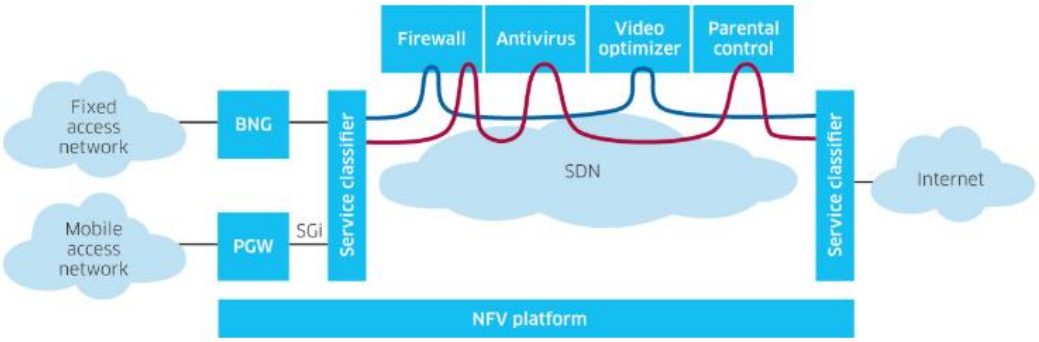


Figura 4. Ejemplo de SFC [61]

El despliegue de funciones de servicio está directamente condicionado por la topología de la red y los recursos físicos, reduciendo en gran medida o eliminando la capacidad de un operador para introducir nuevos servicios o crear dinámicamente cadenas de funciones de servicio.

Tal y como se explica en [59], esta serie de restricciones dan lugar a implementaciones relativamente rígidas y estáticas. Además de reducir la capacidad de desarrollo de los operadores, tiene un efecto en cascada, esto quiere decir que normalmente el cambio de una función de servicio afecta a otros elementos de la cadena de servicios empeorando así la actividad del operador. Este problema es particularmente grave en entornos de servicios elásticos que requieren una creación, destrucción o movimiento relativamente rápidos de funciones de servicio físicas, virtuales o funciones de red.

Además, la transición hacia plataformas virtuales exige modelos de encadenado de servicios que admitan la entrega de servicios dinámica y elástica. Concretamente, son necesarias las siguientes funciones.

- Capacidad de movimiento de funciones de servicio y cargas de trabajos de aplicaciones en la red.
- Capacidad de vincular fácilmente la política de servicio a la información granular.
- Capacidad de dirigir el tráfico al servicio requerido.

Para dar respuesta a estas necesidades el IETF estandarizó el encadenado de funciones de servicio (*Service Function Chaining* (SFC)) [22], definiendo una arquitectura genérica, así como los principios básicos de funcionamiento de SFC. A continuación, se describen los principales detalles.

2.2.1. Arquitectura SFC

El encadenado de funciones de servicio (*Service Function*, SF) permite la creación de servicios de red que consisten en un conjunto ordenado de SF que deben aplicarse a los paquetes, marcos y/o flujos seleccionados como resultado de la clasificación. Se hace referencia a cada SF utilizando un identificador que es único dentro de un dominio habilitado para SF. SFC es un concepto que no solo proporciona la aplicación de un conjunto ordenado de SF para el tráfico seleccionado; más bien, describe un método para desplegar los SF de manera que permite el orden dinámico y la independencia topológica de esos SF, así como el intercambio de metadatos entre las entidades participantes.

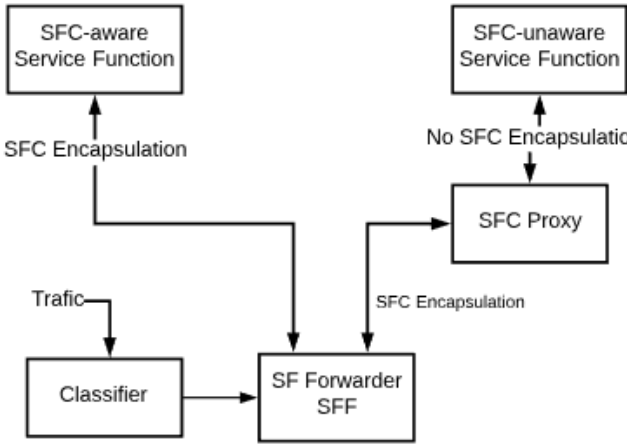


Figura 5: Arquitectura de referencia NFV [22]

La Figura 5 muestra la arquitectura funcional de SFC, de acuerdo a las especificaciones del IETF [22].

- *Service Function* (SF): las funciones de servicio son las encargadas del tratamiento de los paquetes que las atraviesan. Deben de ser conscientes del encapsulado SFC (*aware functions*) para identificar correctamente la cadena correspondiente y el siguiente paso en el encadenado. Las funciones incompatibles con el mismo (*unaware functions*) necesitarán de un elemento intermedio *proxy SFC* que lo procese por ellas.
- *Service Function Path* (SFP): se trata del camino que deberán seguir los paquetes asignados a una determinada cadena dentro del dominio SFC.
- Encapsulado SFC: Se trata de información sobre el SFP, proporciona la identificación del SFP. También incluye información adicional que sirve para saber en qué punto de la cadena se encuentra el paquete en todo momento.

- *Service Function Forwarder* (SFF): es el encargado de reenviar los paquetes que le llegan en función de la información transportada en el encapsulado SFC. También es el responsable de entregar tráfico a un clasificador, a otro SFF o de terminar el camino SFP.
- SFC Proxy: se encarga del tratamiento del encapsulado SFC en nombre de las funciones de servicio incompatibles (*unaware functions*).
- Clasificador de flujo: es el encargado de clasificar todo el tráfico de red de entrada al dominio SFC y asignarles un SFP para su tratamiento.

Tal y como muestra la Figura 5, inicialmente el paquete llega al clasificador donde se identifica el tipo de tráfico que transporta y de acuerdo al mismo se clasifica y se asigna el correspondiente identificador. Posteriormente, de acuerdo a dicha clasificación se encapsula y se entrega al SFF. Este analiza el encapsulado SFC y entrega el paquete a la SF dependiendo el SFP que le corresponda. En caso de que la SF destino sea incompatible con el encapsulado SFC, deberá existir un SFC Proxy que se encargue del tratamiento del encapsulado SFC.

2.2.2. Encapsulado SFC

Un elemento fundamental en la arquitectura propuesta es la utilización de encapsulado SFC, encargado de transportar la información relativa al SFP. Sin este encapsulado, los SFF no sabrían a qué cadena de servicio corresponde cada tráfico y no sería posible el SFC.

El encapsulado SFC permite la selección de la ruta de funciones de servicio. También permite compartir información de metadatos/contexto cuando se requiere dicho intercambio de metadatos. El encapsulado SFC lleva información explícita utilizada para identificar el SFP. Sin embargo, no es un encapsulado de transporte en sí mismo: no se utiliza para reenviar paquetes dentro del tejido de la red. Si los paquetes necesitan fluir entre plataformas físicas separadas, el encapsulado SFC debe encapsularse nuevamente sobre un transporte de red externo. Los elementos de reenvío de tránsito (encaminadores y/o conmutadores) utilizan este encapsulado externo (no SFC) para enviar los paquetes encapsulados SFC a través de la red de transporte particular. Así, uno de los principios clave de la arquitectura de SFC es que el encapsulado de SFC permanece independiente del transporte. Como tal, cualquier protocolo de transporte de red se puede usar para transportar el tráfico encapsulado SFC, como VxLAN-GPE (*Generic Protocol Extension for Virtual eXtensible Local Area Network*), GRE (*Generic Routing Encapsulation*), Ethernet, etc.

No existe una única alternativa de encapsulado SFC y, de hecho, se encuentran en estudio diversas soluciones que, en cualquier caso, respeten los principios descritos. En este sentido, una de las opciones que más esfuerzos ha concentrado y que ha sido recientemente estandarizada, es la utilización de la cabecera NSH [60]. Opciones alternativas (como *Segment Routing* [16] [63] o MPLS [17]) están recibiendo atención en los últimos meses, encontrándose no obstante en proceso de desarrollo (documentos del IETF en borrador).

Así, en el contexto del proyecto, se trabajará con el encapsulado NSH, cuyos detalles se explican a continuación.

2.2.3. *Network Service Header* (NSH)

Como ya se ha visto, la arquitectura SFC requiere que el tráfico entre SF vaya encapsulado con la información SFC pertinente. La cabecera NSH proporciona dicho encapsulado permitiendo además transportar metadatos[2] a lo largo de las rutas de servicio instanciadas. Las especificaciones de dicha cabecera han sido recientemente estandarizadas en la RFC 8300 [60]. La NSH está diseñada para encapsular un paquete o trama original y, sucesivamente, encapsularse en un encapsulado de transporte exterior (utilizado para entregar los paquetes NSH a los elementos conocedores del mismo), tal y como muestra la Figura 6.

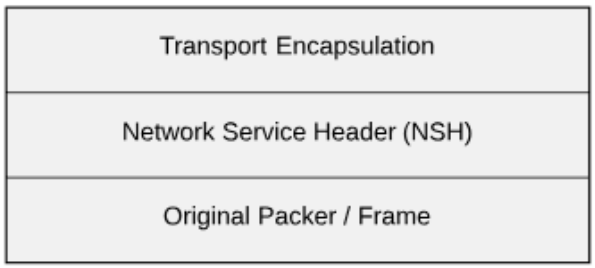


Figura 6: Encapsulación NSH

La cabecera la componen los siguientes elementos (Figura 7):

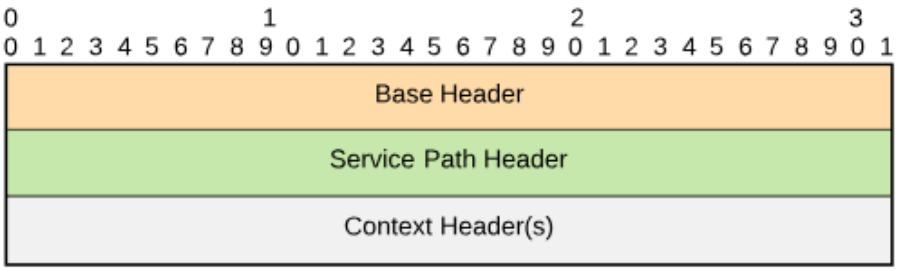


Figura 7. Cabecera NSH

- *Base Header:* proporciona información sobre la cabecera de servicio y el protocolo encapsulado en el *payload*.
- *Service Path Header*: proporciona el identificador de la ruta y la localización en todo momento.
- *Context Header:* transporta metadatos relacionados con el contexto SFC.

[2] Metadatos: información sobre la clasificación utilizada para la aplicación de políticas o información sobre el contexto de red para el reenvío siguiente, este último es importante cuando el tráfico va a cambiar de dominio de red en el siguiente salto.

A continuación, se explicarán estos campos en detalle.

2.2.3.1. *NSH Base Header*

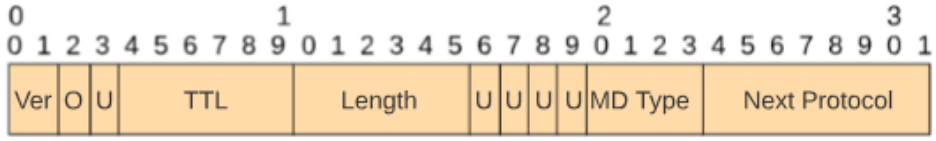


Figura 8: NSH Base Header

La cabecera base NSH (Figura 8) está formada por:

- *Version*: garantiza compatibilidad con otras versiones.
- *Bit O:* indica que es un paquete de Operaciones, Administración y Mantenimiento (OAM).
- *TTL*: indica el máximo de numero de saltos a través del SFF para un SFP.
- *Length*: indica la longitud total de la cabecera SNH en palabras de 4 bytes.
- *MD Type:* define el formato de los metadatos que se transportan.
- *Next Protocol:* indica el protocolo de los datos encapsulados. Se definen los siguientes valores de protocolos.
  - 0x0: Sin asignar
  - 0x1: IPv4
  - 0x2: IPv6
  - 0x3: Ethernet
  - 0x4: NSH
  - 0x5: MPLS
  - 0xFE: Experiment 1
  - 0xFF: Experiment 2

2.2.3.2. *Service Path Header*

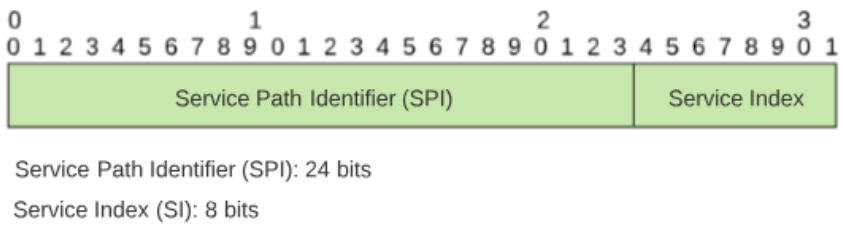


Figura 9: NSH Service Path Header

La cabecera *Service Path* (Figura 9) está formada por:

- *Sevice Path Identifier* (SPI): identifica la ruta de servicio. Inicialmente, el clasificador debe asociar el identificador de la ruta a cada tráfico que clasifica.
- *Service Index* (SI): identifica la ubicación del paquete dentro de un SFP. Inicialmente el clasificador debe establecer el valor del SI en 255 (configurable en función de la longitud de la ruta). Este valor deberá ser decrementado en una unidad por las SF (o por los *proxys SFC*) en cada salto de la cadena.
  A pesar de que el mecanismo principal de detección de bucles es el TTL que está contenido en la *Base Header*, el SI es un mecanismo secundario de detección de este tipo de fallos.

2.3.SDN

El término "Redes definidas por *software*" (*Software Defined Netwoking* (SDN) [14]), hace referencia a la capacidad de controlar, cambiar y gestionar el funcionamiento de una red mediante una aplicación *software* llamada "Controlador". De esta manera es posible separar el plano de control del plano de datos permitiendo así que el control de la red sea totalmente programable. De esta manera se pueden distinguir dos entidades, entidad controlada (plano de datos) y entidad controladora (plano de control). El funcionamiento de la red se basa en que el elemento controlador, el denominado "cerebro" de la red, dispone de una visión centralizada de la topología completa de red y permite al administrador manipular de manera dinámica e inteligente los elementos subyacentes (conmutadores y encaminadores) para definir el funcionamiento del reenvío del tráfico en la red que controla.

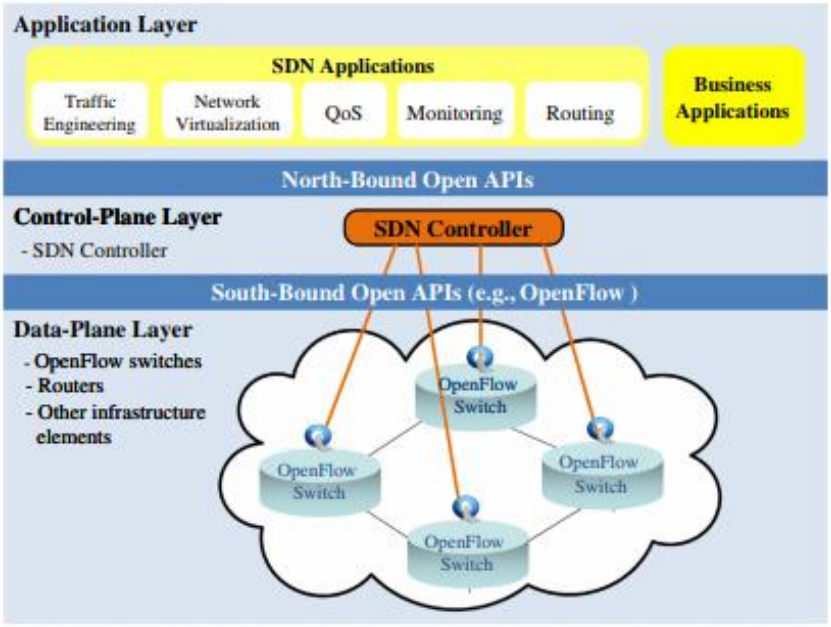


Figura 10: Arquitectura SDN [20]

Tal y como se muestra en la Figura 10, el elemento controlador es el encargado del plano de control, proporciona a las aplicaciones la capacidad de obtener datos de la red y de su configuración a través de la *north-bound interface.* Esta engloba el conjunto de API (basadas mayoritariamente en REST API [4]) que permiten la interacción entre el controlador y las aplicaciones que demandan los servicios del controlador.

Para el correcto funcionamiento del plano de datos, el controlador tiene comunicación directa con los elementos de la red a través de la *south-bound interface*. Esta engloba el conjunto de protocolos que permiten la interacción entre el controlador y los elementos de red (siendo OpenFlow [71] uno de los ampliamente utilizados hoy en día). A través de esta interfaz, el controlador configura el funcionamiento de los elementos de red. Por ejemplo, la instalación de flujos en un *switch*.

Cuando se habla de virtualización de red, habitualmente se referencia tanto a NFV como SDN. Dichas tecnologías, como se ha descrito, no son lo mismo y se corresponden con ámbitos de aplicación distintos (virtualización de funciones – NFV – frente a separación de planos de datos y control –SDN–). No obstante, sus rasgos comunes, esencialmente la programabilidad y flexibilidad, y su nexo con la gestión de servicios en la nube (*cloud computing*) hacen que su funcionamiento conjunto tenga gran potencial. En este sentido, si bien este trabajo se centra en el estudio de NFV, su aplicación al diseño de SFC tiene especial relación con SDN, tal y como se describe a continuación.

2.4. SDN y NFV: SFC como caso de uso

Entre los múltiples casos de uso de SDN y NFV, SFC surge como un escenario de aplicación común a ambas tecnologías. Actualmente SDN se presenta como una opción interesante para controlar mediante *software* el tráfico cursado por las funciones virtuales creadas por NFV.

Gracias a la capacidad de SDN de programar el funcionamiento del flujo de tráfico en la red, una de sus claras aplicaciones es la Ingeniería de Tráfico (*Traffic engineering*, TE). En este contexto, SFC se presenta como un caso de uso evidente [3], puesto que el encadenado de funciones no deja de ser el establecimiento de un camino predefinido (como los LSP de MPLS) que garantice que cada flujo de datos específico atraviese la cadena de VNF que le corresponde.

Así, un controlador SDN puede generalizarse como el *Path Computation Element* (PCE) [18], elemento centralizador del cálculo de los caminos en la arquitectura de TE [18]. Dicho elemento se encarga de calcular rutas de tráfico óptimas a través de una red en función de criterios diversos, más allá de los clásicos algoritmos de encaminamiento, gracias a la visión global de la red. Del mismo modo, puede actualizar las rutas para reflejar cambios en la red o las demandas de tráfico.

SFC, como se ha descrito previamente, hace referencia al concepto general de encadenado de funciones o servicios, no siendo exclusivo además de funciones virtuales. Por otra parte, los estándares relacionados establecen la arquitectura funcional genérica y no el mecanismo específico con el que este se lleva a cabo

(MPLS, SDN…) o el método de encapsulado (como ya se ha indicado, la cabecera NSH descrita es una opción, existiendo alternativas). En este contexto tan abierto, en este proyecto se ha adoptado la filosofía aquí descrita basada en SDN, tomando como referencia la flexibilidad que la combinación SDN-NFV. Así pues, de ahora en adelante la referencia a SFC y su implementación se refiere al encadenado de funciones virtuales mediante control centralizado y encapsulado NSH.

3. Soluciones y herramientas *software*

En este capítulo se van a plantear las principales soluciones *software* a emplear en el desarrollo técnico del proyecto (capítulo Desarrollo técnico). También se expondrán las principales herramientas existentes hasta el día de hoy en términos de virtualización.

3.1. Herramientas de virtualización

La virtualización es la tecnología que permite crear entornos simulados (Sistemas Operativos, almacenamiento, aplicaciones, redes, etc.) a partir de un sistema *hardware* tradicional. El concepto tradicional de virtualización está ligado a la virtualización de sistema operativo, para lo cual se emplean hipervisores.

Más adelante surgió el concepto de virtualización de "aplicaciones", en el cual independientemente del sistema operativo, se aísla una aplicación *software* de las capas subyacentes, haciendo que esta sea totalmente compatible y exportable a cualquier sistema operativo, para ello se emplean contenedores.

3.1.1. Hipervisores

Los hipervisores son aplicaciones que presentan a los sistemas operativos virtualizados (sistemas invitados) una plataforma operativa virtual (*hardware* virtual), a la vez que ocultan a dicho sistema operativo virtualizado las características físicas reales del equipo sobre el que operan.

Existen hipervisores que se ejecutan directamente sobre el *hardware* de equipo (tipo 1, como VMware ESXi [70]) o en cambio, hipervisores que se ejecutan sobre un sistema operativo anfitrión (tipo 2, como KVM [23] o VirtualBox [68]), ambos se muestran en las Figuras 11 y 12.

Figura 11. Hipervisor tipo 2

Figura 12. Hipervisor tipo 1

Para el desarrollo de este proyecto, se empleará KVM [23], un hipervisor de código abierto que funciona tanto sobre clientes Linux como sobre Windows. Con este se crearán máquinas virtuales OpenSuse Leap 42.3, en las cuales estarán instalados los nodos de Openstack. También se empleará para crear las máquinas virtuales en el nodo Compute00 (VNF) (todo ello detallado en el capítulo Desarrollo técnico).

3.1.2. Contenedores

Tal y como se ha comentado anteriormente, una máquina virtual es un *software* que funciona de manera aislada sobre un sistema operativo subyacente. Sin embargo, mediante contenerización lo que se consigue es aislar diferentes aplicaciones del sistema operativo.

A simple vista, los contenedores ejecutan su propio sistema operativo, pero realmente comparten el sistema operativo anfitrión. El contenedor también ve su propio sistema de ficheros, esto es resultado de la superposición de diferentes capas de abstracción entre la maquina física, el sistema operativo anfitrión y los contenedores.

A su vez, los recursos *hardware* se asignan dinámicamente a los contenedores dependiendo de la demanda en cada momento, de esto se encarga el motor del contenedor correspondiente. Esto hace a los contenedores más eficaces en ciertos tipos de aplicaciones que las máquinas virtuales tradicionales, puesto que a estas se les asigna los recursos de antemano.

La Figura 13 muestra la arquitectura conceptual de contenedores.

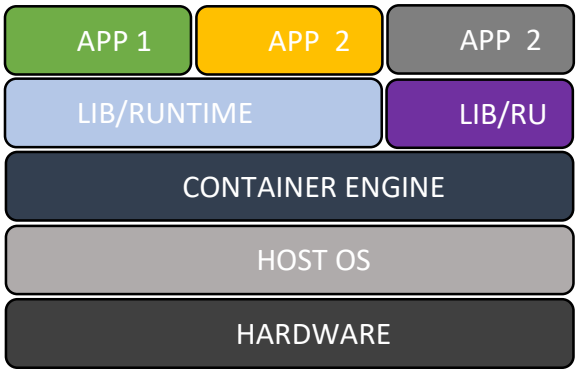


Figura 13: Arquitectura básica contenedores virtuales.

Existen diferentes implementaciones de contenedores como, por ejemplo, contendores LXC de Linux [73] o Docker [13].

Para la instalación de la infraestructura *cloud* Openstack se emplean contenedores LXC para poner en funcionamiento en el nodo de control los diferentes módulos/servicios de Openstack de manera independiente (explicados en el apartado Openstack), estos son contenedores Linux de sistema, lo que quiere decir que comparten el kernel de Linux con la maquina anfitriona.

3.2. *Software* SDN

Como se ha mencionado previamente, la implementación de SFC realizada en este proyecto contempla la centralización de funciones mediante SDN. Así, en este apartado se va a presentar el *software* SDN específico utilizado dentro del desarrollo técnico completo de NFV descrito posteriormente. Concretamente, se describen los dos componentes clave de la arquitectura SDN, el controlador y los equipos de red compatibles con el mismo.

3.2.1. Controlador ODL

Actualmente existen varios controladores SDN disponibles, tanto propietarios como de código abierto. Estos últimos cuentan con un desarrollo activo gracias a la comunidad de investigación y la industria y, dado su carácter gratuito y accesible a la programación, son los mayoritariamente contemplados en fases de desarrollo y evaluación como la del presente proyecto. Entre ellos, existen múltiples alternativas, como OpenDayLight (ODL) [28], *Open Network Operating System* (ONOS) [26], Tungsten Fabric [66], etc. Dado que todos ellos responden a las necesidades del desarrollo técnico a realizar, se ha optado por la utilización de ODL, uno de los de mayor expansión, y compatible con la plataforma de evaluación de NFV descrita posteriormente.

Con 150 empresas del sector IT asociadas y el aval de la *Linux Foundation* [65], OpenDayLight, es uno de los proyectos actualmente de referencia cuando se habla de SDN. La plataforma OpenDayLight es multiprotocolo y modular, lo que significa que está pensada para satisfacer las necesidades de cualquier usuario.

El controlador ODL expone interfaces abiertas con la posibilidad de ser usadas por las aplicaciones para recopilar información de la red a través del controlador, monitorizar y administrar la red (*north-bound* API). Esta interfaz define una serie de requisitos que permiten a las aplicaciones dicha comunicación, la más conocida es API REST, la cual establece cómo debe ser una comunicación mediante peticiones REST HTTP entre las aplicaciones y el controlador.

Por otra parte, como interfaz de comunicación con los elementos controlados de la red (*south-bound* API), soporta distintos mecanismos como OpenFlow, *Open vSwitch Data Base* (OVSDB), NETCONF y *Border Gateway* Protocol (BGP). Mediante estos protocolos, el controlador se comunica con los elementos de red para configurar su funcionamiento o extraer datos de interés.

En nuestro caso, el controlador ODL se encargará de configurar mediante OpenFlow un *switch* Open vSwitch (OVS) [58], elemento central en el establecimiento de las cadenas de servicio (SFC) como se describirá en el siguiente apartado.

3.2.2. Open vSwitch

Tradicionalmente los servidores se conectaban a conmutadores *hardware* físicos. Sin embargo, con la aparición de las máquinas virtuales, y ahora de los contenedores, han aparecido elementos de conmutación virtuales. Actualmente, existen tres conmutadores virtuales populares: conmutador virtual VMware [69], Cisco Nexus [11] y Open vSwitch.

Open vSwitch es un conmutador virtual multicapa bajo la licencia de código abierto Apache 2.0. [64]. No es exclusivo de SDN ya que se puede emplear como un *switch* tradicional, pero, gracias a la compatibilidad con el protocolo OpenFlow que ofrece, puede controlarse mediante un controlador SDN, como ODL.

Open vSwitch fue creado por NICIRA con la finalidad de satisfacer las necesidades de la comunidad de código abierto en Linux. Inicialmente se creó para dar soporte a hipervisores como KVM y XEN, pero a día de hoy se puede considerar como el conmutador virtual de insignia para entornos XEN.

Soporta protocolos como NetFlow, sFlow, OpenFlow, duplicación de puertos, VLAN, LACP, etc. Por esto, resulta de interés cuando se habla de SDN y NFV. Actualmente desemplea un papel importante dentro de Openstack, por lo que se hará hincapié en este conmutador virtual más adelante.

3.3. Openstack

En 2012 nació una fundación sin ánimo de lucro llamada *Openstack Foundation*, creada para promover el desarrollo del proyecto Openstack [31] [62]. La *Openstack Foundation* está formada por más de 200 empresas punteras del sector de las telecomunicaciones, entre ellas se encuentran AT&T, Suse, Ericsson, Huawei, Intel, etc. [32]. Openstack es una plataforma *Open Source* de *Cloud Computing* que permite crear todo tipo de nubes (privadas o públicas) y desplegar una infraestructura *cloud* conocida como *Infrastructure as a Service* (IaaS). IaaS proporciona acceso a recursos informáticos situados en un entorno virtual, a través de una conexión pública. Estos recursos no son nada menos que *hardware* virtualizado que, como en el caso que nos ocupa, pueden implementar funciones de red virtualizadas. [30] [21]

Es un *software* de código abierto, lo que significa que además de estar respaldado por las empresas más importantes en el sector, tiene una enorme comunidad de desarrolladores detrás, por lo que es un proyecto que poco a poco va ganando fuerza y robustez en el sector. Si bien su característica abierta puede dificultar el acceso a documentación estructurada o depuración de fallos, dada su gran expansión existen numerosas fuentes de apoyo, como pueden ser foros, wikis o canales de chat *Internet Relay Chat* (IRC).

Openstack está formado por diferentes proyectos interrelacionados con diferentes funcionalidades que en conjunto crean una sinergia muy potente. El total de módulos disponibles es muy extenso y queda fuera del ámbito de este proyecto. No obstante, a continuación, se definen los componentes principales para entender su aplicación, especialmente en el contexto posterior del desarrollo técnico.

El núcleo de Openstack consta de 9 módulos. La Figura 14 representa la arquitectura conceptual de Openstack. En dicha figura se han resaltado los módulos utilizados en el desarrollo técnico, cuya función específica se detalla a continuación.

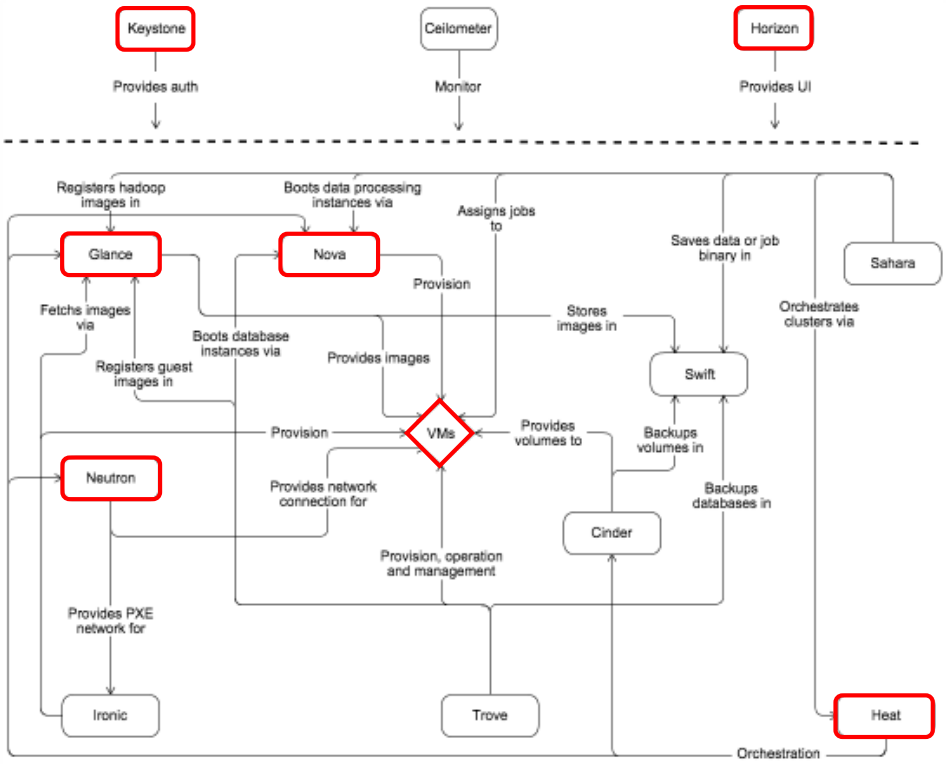


Figura 14. Arquitectura conceptual de Openstack [35]

3.3.1. Nova

"Nova es el proyecto Openstack que proporciona una manera de aprovisionar instancias de cómputo (también conocidos como servidores virtuales). Nova admite la creación de máquinas virtuales, servidores baremetal[3] *(mediante el uso de ironic) y tiene soporte limitado para contenedores de sistemas. Nova se ejecuta como un conjunto de demonios sobre los servidores Linux existentes para proporcionar ese servicio."* [34]

Es el motor que nos permite desplegar y administrar máquinas de computo en Openstack. En el contexto de este proyecto, se utiliza para la creación y destrucción de las VNF y del almacenamiento de sus datos.

3.3.2. Keystone

"Keystone es un servicio de Openstack que proporciona autenticación de cliente de API, detección de servicios y autorización distribuida de múltiples inquilinos mediante la implementación de la API de identidad de Openstack" [39]

Es el modulo que se encarga de la autenticación de los diferentes usuarios que emplean la infraestructura y del acceso a diferentes servicios.

[3] Servidor Baremetal: también se denominan servidores dedicados, hace referencia a un servidor en el cual una máquina virtual se instala directamente en disco sin necesidad de un sistema operativo.

3.3.3. Neutron

"Neutron es un proyecto de Openstack para proporcionar "conectividad de red como servicio" entre dispositivos administrados por otros servicios de Openstack (por ejemplo, nova). Implementa la API de Neutron." [48]

Es el módulo encargado de todas las tareas de interconexión de redes, tanto la conectividad interna como la asignación de direcciones públicas externas a las VFN. También se encarga del tráfico de red formando parte, por lo tanto, del establecimiento de SFC.

Concretamente, en este proyecto Neutron se empleará para la creación y la gestión de las redes y dar conectividad a las VNF con Internet.

A la hora de establecer cadenas de servicio (SFC), Neutron cuenta con un *driver* propio, *networking-sfc* [43] y el *plugin ml2* [41] para comunicarse con los *switch* OVS y configurar el SFC. No obstante, también permite utilizar controladores SDN externos (como ODL) para realizar esta funcionalidad. Es el caso contemplado en este proyecto. En este caso, Neutron no se comunica directamente con los *switch* OVS sino a través de ODL, tal y como muestra la Figura 15. De este modo ODL es el responsable de configurar adecuadamente dichos *switch* mediante la información recibida de Neutron.

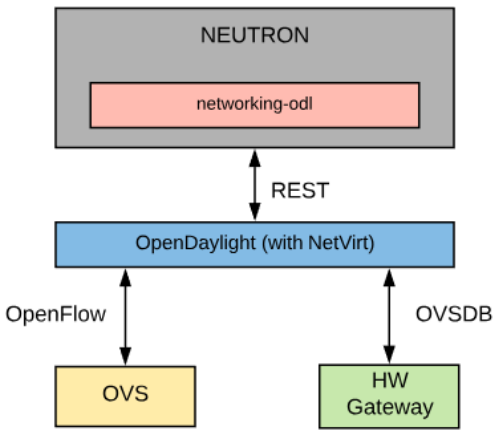


Figura 15. Comunicación entre Neutron, ODL y OVS

3.3.4. Cinder

"Cinder es un proyecto de Openstack para proporcionar "almacenamiento de bloques como un servicio". [42]

Es el encargado del almacenamiento, semejante al almacenamiento en unidades de disco tradicional.

3.3.5. Glance

"El proyecto del servicio de imagen (glance) proporciona un servicio donde los usuarios pueden cargar y descubrir activos de datos que están destinados a ser utilizados con otros servicios. Esto incluye actualmente imágenes y definiciones de metadatos." [47]

En este proyecto, el módulo Glance se usa para almacenar y proveer las imágenes de los sistemas operativos que se emplearan en las VNF. En el contexto de este proyecto, Glance se usará mediante el CLI *Openstack image create…* para crear las imágenes de los Sistemas Operativos de las VNF.

3.3.6. Swift

"El proyecto Openstack Object Store, conocido como Swift, ofrece software de almacenamiento en la nube para que pueda almacenar y recuperar gran cantidad de datos con una API simple. Está diseñado para escalar y optimizar la durabilidad, disponibilidad y concurrencia de todo el conjunto de datos. Swift es ideal para almacenar datos no estructurados que pueden crecer sin límite." [33]

Se encarga de almacenar los archivos del sistema, asegurar su integridad y de replicarlos en las diferentes unidades de disco de la infraestructura. En el contexto de este proyecto, no se empleará este módulo.

3.3.7. Ceilometer

"El proyecto Ceilometer es un servicio de recopilación de datos que brinda la capacidad de normalizar y transformar datos en todos los componentes principales de Openstack actuales. Ceilometer es un componente del proyecto de telemetría. Sus datos se pueden usar para proporcionar facturación a los clientes, seguimiento de recursos y capacidades alarmantes en todos los componentes principales de Openstack." [46]

Permite obtener datos sobre los servicios de Openstack. En el contexto de este proyecto, no se empleará este módulo.

3.3.8. Heat

"Heat es un servicio para orquestar aplicaciones de nube compuestas utilizando un formato de plantilla declarativa a través de una API REST nativa de Openstack" [50]

Proporciona orquestación a partir de plantillas. Estas describen la infraestructura necesaria para una instancia. Los requerimientos de la instancia se almacenan en un fichero que más a delante se usa para su despliegue. En este proyecto se empleará Heat por medio de Tacker para orquestar funciones de red virtualizadas tal y como se explica en el detalle del módulo Tacker.

3.3.9. Horizon

"Horizon es la implementación canónica del panel de control de Openstack, que proporciona una interfaz de usuario basada en web para los servicios de Openstack, incluidos Nova, Swift, Keystone, etc." [38]

Se trata de una interfaz gráfica accesible a través de un navegador web. Permite visualizar de manera gráfica los servicios que están alojados en tu nube. En este proyecto, Horizon se empleará ocasionalmente para verificar gráficamente los despliegues de red creados. Para ello, en nuestro caso, se accede vía web (a través de la URL indicada en la instalación: *http://192.168.122.3*) *y* se introducen

los datos de inicio de sesión configurados en el momento de la instalación. De este modo tendremos disponible el GUI de Openstack.

Con la antigua versión del CLI de Openstack, se diferenciaba el uso de cada módulo y sus diferentes utilidades. Sin embargo, la nueva versión del CLI enmascara el uso de cada módulo haciéndolos invisibles al usuario.

Además de los módulos principales de Openstack, la plataforma dispone de módulos adicionales cuya utilización depende del contexto de aplicación. Es el caso del módulo Tacker, cuya utilidad en este trabajo se detallará posteriormente, avanzando en este caso su funcionalidad:

3.3.10. Tacker

"Tacker es un servicio Openstack para NFV Orchestration con un VNF Manager de propósito general para implementar y operar funciones de red virtual (VNF) y servicios de red en una plataforma NFV. Está basado en el Marco Arquitectónico ETSI MANO." [49]

Se trata de un NFV MANO que permite orquestar funciones y controlar su ciclo de vida.

A simple vista se podría llegar a pensar que Heat y Tacker son módulos equivalentes, con funcionalidades similares, pero en la práctica se podría decir dichas funcionalidades sí son similares, pero no independientes.

Como se muestra en la Figura 16, y siguiendo la arquitectura NFV-ETSI, Tacker actúa como VNFM y VNFO a un nivel superior que Heat. Sin embargo, Tacker no puede interactuar directamente con el entorno Openstack, por lo que se comunica con Heat y es este es el que realiza la gestión y orquestación en Openstack a partir de las plantillas creadas con Tacker.

Mientras que Heat utiliza plantillas Heat *Orchestration Templates* (HOT) [36], Tacker emplea plantillas *Topology and Orchestration Specification for Cloud Applications* (TOSCA), lenguaje estándar de OASIS [25]. Así, Heat implementa la funcionalidad *heat-tanslator* con la que es capaz de traducir plantillas TOSCA en HOT [37].

En el Anexo 2 queda definido el CLI de Tacker en el cual se detalla su uso.

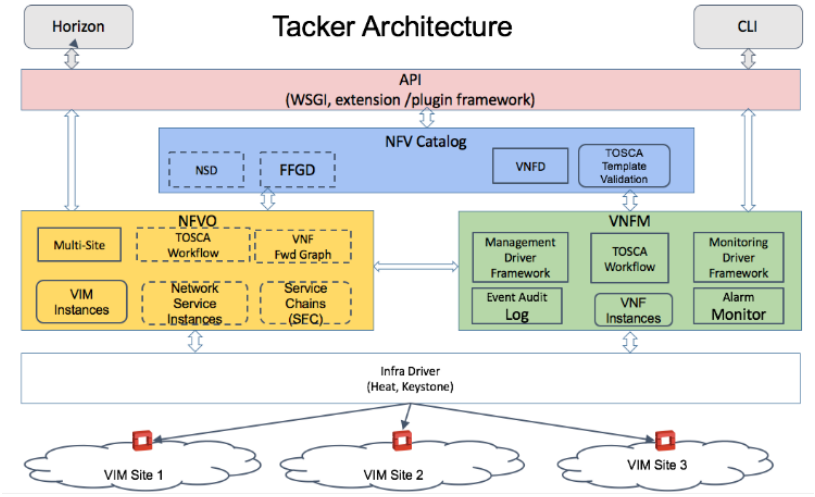


Figura 16. Arquitectura de Tacker [44]

4. OPNFV

Open Platform for NFV es una plataforma que facilita el desarrollo de NFV a través de diferentes proyectos de código abierto. Su principal finalidad es buscar un desarrollo ascendente de NFV, pero también trabaja en la integración de esta con proyectos existentes como OpenDaylight, ONOS, Tungsten Fabric, Openstack, Kubernetes [72], KVM [23], Open vSwitch y Linux. La Figura 17 muestra la arquitectura de referencia de OPNV, y su relación con la arquitectura NFV.

OPNFV nació como un proyecto dentro de la *Linux Foundation* en el cual participan numerosas empresas punteras en el sector IT como AT&T, Brocade, China Mobile, Cisco, Dell, Ericsson, HP, Huawei, IBM, Intel, Juniper Networks, NEC, Nokia Networks, NTT DOCOMO, Red Hat, Telecom Italia y Vodafone.

Con apenas cuatro años de vida, OPNFV se ha ganado un nombre en el mercado de las telecomunicaciones gracias a sus soluciones NFV, en las cuales integra diferentes tecnologías de red externas a NFV, pero las cuales mejoran la experiencia de uso y las funcionalidades de la infraestructura [50] [9] [5].

OPNFV tiene un gran abanico de proyectos abiertos actualmente [6] centrados en la creación de entornos de desarrollo integrando Openstack con diferentes tecnologías.

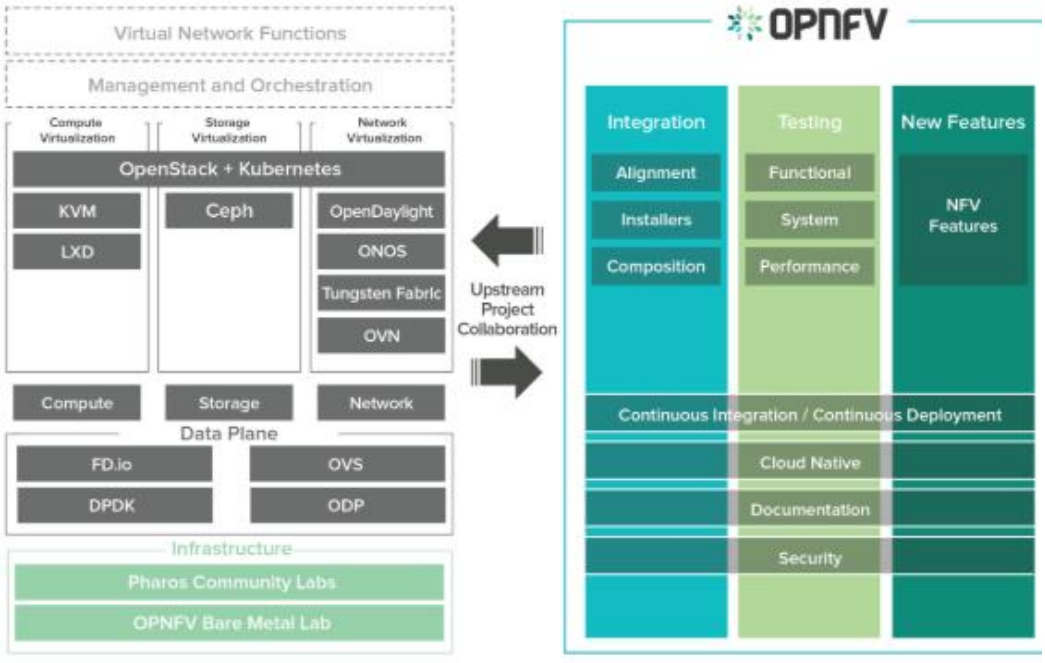


Figura 17. OPNFV Release Architecture [8]

Actualmente, la última *release* estable es Fraser [55], en la cual encontramos diversos escenarios OPNFV disponibles para su instalación. Dichos escenarios no son sino la automatización de la instalación del *software* necesario (módulos de Openstack o proyectos integrados, como ODL) para dar respuesta a los requisitos especificados. En definitiva, los escenarios OPNFV están diseñados para alojar funciones de red virtualizadas (NFV) en una variedad de arquitecturas y ubicaciones de implementación. Cada escenario proporciona capacidades y/o componentes específicos destinados a

resolver problemas específicos para el despliegue de VNF. En el caso que nos ocupa, resultan interesantes los escenarios *os-odl-sfc*, que proporcionan un despliegue inicial completo de los componentes de interés para trabajar con NFV, ODL y SFC.

Para poder realizar el proyecto de instalación de un escenario basado en la última *release* de OPNFV, encontramos diferentes proyectos que dan soporte a la instalación OPNFV Fraser como por ejemplo Fuel [7] o Releng-xci [52]. Para el desarrollo del proyecto, se ha escogido el instalador Releng-xci que despliega el proyecto de aplicación SFC con el escenario *os-odl-sfc* debido a que se ajustan perfectamente a las características NFV y SDN que se desean.

4.1. OPNFV XCI

La Integración Continua entre Comunidades, *Cross Community Continuous Integration* – XCI, en su versión inglesa, hace referencia al proyecto utilizado por la comunidad OPNFV para desarrollar, integrar, probar y lanzar la plataforma de referencia integrada para NFV. Para aumentar la velocidad de desarrollo y evolución de la plataforma, OPNFV debe proporcionar medios para sus desarrolladores y usuarios. Una de las formas de lograr esto es garantizar que los desarrolladores tengan acceso a las últimas versiones de los componentes que están desarrollando, integrando y probando. Basándose en esta necesidad, el Grupo de Trabajo de Infraestructura de OPNFV (Infra WG) inició la iniciativa XCI, incorporando un nuevo modelo de desarrollo y lanzamiento en OPNFV que se basa en prácticas y principios de Entrega Continua y DevOps[4].

4.1.1. Releng-xci

Releng-xci [52] es uno de los proyectos instaladores activos actualmente en OPNFV. Fue creado con el objetivo de permitir desplegar diferentes escenarios disponibles en las diferentes *release*. Así, este proyecto se ha integrado en sucesivas *release* [67] de OPNFV desde la *release* Brahmaputra.

Se trata de un entorno de desarrollo de Openstack muy flexible que se ajusta a las necesidades de casi todos los usuarios. Existe un gran abanico de posibilidades en función de los recursos *hardware* de los que se disponga. Desde OPNFV ponen a disposición de los usuarios o desarrolladores lo que llaman *sandbox*, el cual contiene todo lo necesario para desplegar la infraestructura. El *sandbox* proporciona:

- Una manera automatizada para levantar y derribar una pila completa.
- Varios *flavors* (configuraciones) para escoger y usar.
- Soporte para diferentes distribuciones de Linux.
- Múltiples escenarios OPNFV para instalar.
- Posibilidad de seleccionar diferentes versiones de componentes.
- Posibilidad de habilitar/deshabilitar servicios de Openstack.

[4] *Develop and Operations* (DevOps) hace referencia a una metodología de desarrollo de *software* que se centra en el desarrollo continuo y en la integración entre desarrolladores de *software* y los profesionales de sistemas en las tecnologías de la información (IT)"

El trabajo aquí presentado se desarrolla en un contexto de evaluación en el que analizar NFV y SFC implementando pruebas de concepto que faciliten el estudio en un laboratorio controlado. El instalador releng-xci para la *release* Fraser permite el despliegue de escenarios SFC (*os-odl-sfc-noha* y *os-odl-sfc-ha*) adaptándose por lo tanto a las necesidades del proyecto. Considerando además las limitaciones del *hardware* disponible, atendiendo a las propias especificaciones de OPNV, se ha optado por este entorno de trabajo realizando por tanto los despliegues mediante el *sandbox* proporcionado (detalles en el apartado 5).

Concretamente, se ha escogido el *flavor mini* que es el que más se ajusta a los recursos disponibles. Se trata de una instalación multinodo que puede instalarse en un servidor con capacidad suficiente para configurar las tres máquinas virtuales KVM (lo que posteriormente identificamos como "nodos") que constituyen el núcleo de Openstack, tal y como se explica a continuación. El despliegue, realizado mediante *Openstack Ansible* (OSA), se encarga de aprovisionar y configurar dichas máquinas virtuales, creadas durante el despliegue del sistema e interconectarlas mediante VLAN creadas a través de *bridge* Linux.

4.1.2. Diagrama de bloques

La instalación consta de tres nodos (máquinas virtuales), tal y como refleja la Figura 18:

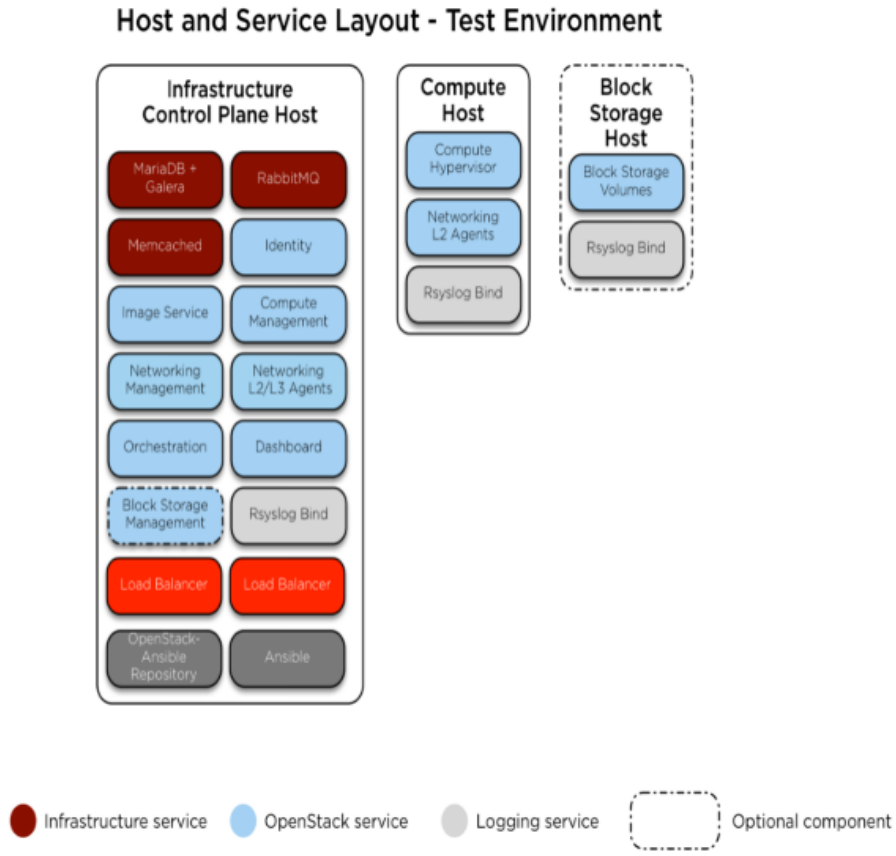


Figura 18: Diagrama de bloques nodos releng-xci

Opnfv: este nodo dirige los procesos de instalación e implementación de la red virtualizada, con el fin de garantizar que esté aislado del *host* físico y siempre se realice en una máquina limpia.

Controller: el plano de control reside en este *host*. En él se llevan a cabo los servicios de red, almacenamiento y gestión de la infraestructura.

Compute: los servicios de computo residen en este *host*. Ejecuta los recursos virtualizados (VNF en nuestro contexto) y su gestión.

4.2. Otros proyectos Open Source integrados en OPNFV

Releng-xci integra numerosos proyectos de código abierto (denominados *upstream projects* en OPNFV, algunos de ellos identificados en la Figura 17 [57]). Atendiendo a los despliegues que se utilizarán en este proyecto (escenario básico de virtualización y SFC), en este apartado se van a detallar aquellos involucrados en el funcionamiento del sistema final:

- Open vSwitch: es el encargado de interconectar las VNF que se creen. Además, como ya se verá más adelante, en el residirá el funcionamiento SFC de nuestra red.
- Controlador ODL: uno de los aspectos más importantes de releng-xci es la integración de NFV con SDN, concretamente la implementación del controlador ODL. A pesar de que no entra en las competencias de este proyecto, este, proporcionara la posibilidad de controlar el Open vSwitch añadiendo a nuestro sistema NFV toda la flexibilidad de SDN.
- Contenedores LXC: tal y como se ha comentado en el apartado Contenedores, Openstack está formado por diferentes proyectos con diferentes funciones. En lugar de tener todos los módulos funcionando directamente como procesos sobre la misma máquina virtual, estos módulos son desplegados en contenedores LXC totalmente independientes entre ellos. De esta manera se proporciona un nivel de abstracción a cada módulo que facilita la limpieza de nuestro sistema. Con el fin de comunicar estos contenedores, cada uno dispone de diferentes interfaces de red virtuales por las cuales se comparten información.

Tras una breve presentación de OPNFV y Releng-xci, su uso, funcionamiento y estructura se detallará en el capítulo Desarrollo técnico, de desarrollo técnico, presentado a continuación.

5. Desarrollo técnico

En este capítulo se abordará el despliegue e integración de los componentes de OPNFV utilizados para la evaluación de los escenarios de prueba controlados. Tal y como se ha detallado previamente, el núcleo de dichos despliegues es la infraestructura Openstack (los componentes requeridos según el escenario), la integración con ODL y OVS y la virtualización específica de funciones.

Antes de comenzar el desarrollo, resulta interesante detallar la relación entre la arquitectura de NFV propuesta por ETSI en [2] y los componentes de Openstack instalados durante el despliegue.

De todos los componentes de Openstack que se instalan, no todos tienen funcionalidades relacionadas con la arquitectura de NFV, en la Figura 19 se detallan los componentes que componen el esqueleto de NFV.

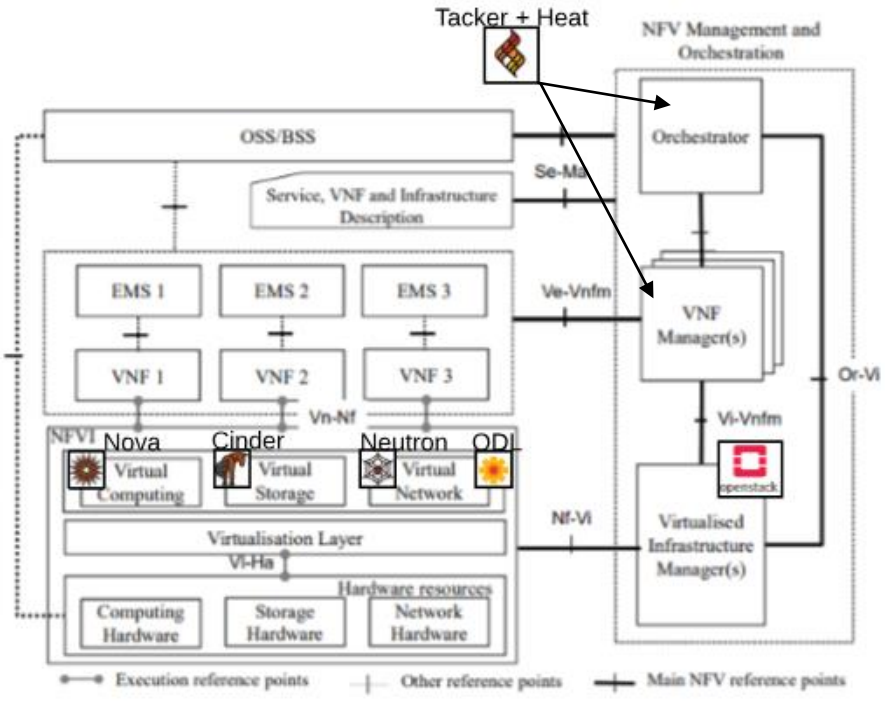


Figura 19. Arquitectura NFV-Openstack

A pesar de que OpenDayLight no representa una funcionalidad de NFV, resulta interesante indicar que trabaja de manera conjunta con Neutron para proporcionar servicios de red y SDN.

5.1. Despliegue de Openstack mediante OPNFV

Para el despliegue de Openstack se optó por un despliegue de OPNFV, este simplifica la selección de las herramientas necesarias, adaptadas al escenario SFC, lo que incluye Openstack y ODL. Como instalador, se seleccionó el proyecto Releng-xci, mencionado previamente.

Este proyecto nos permite instalar concretamente el escenario *os-odl-sfc-noh*a [56]. Se trata del entorno de desarrollo más adecuado para nuestros propósitos, puesto que nos permite realizar las pruebas de concepto sobre SFC necesarias con unos requisitos de *hardware* acordes a lo que se dispone en el laboratorio. Cabe destacar, por otra parte, el soporte para dicho escenario proporcionado por el responsable del proyecto SFC de OPNFV[5], con quien se ha mantenido una comunicación fluida, lo cual ha facilitado el correcto desarrollo del trabajo.

Tal y como se indicó en el apartado 4.1.1, Releng-xci proporciona un *sandbox* de instalación que facilita el despliegue de la arquitectura para diversas configuraciones *hardware* (flavor), como refleja la Figura 19. Atendiendo a las especificaciones de nuestro entorno de pruebas (una estación de trabajo Workstation Intel Xeon SkyLake-SP 3106 dual con 480GB de disco SSD, 8TB de disco duro y 128 GB de memoria RAM.), se ha seleccionado el despliegue del **flavor mini**.

Por otra parte, el escenario desplegado (*os-odl-sfc-noha*) integra el proyecto ODL SFC [29] en el entorno de OPNFV (*release* ODL Oxygen SR1) instalando para ello todos los componentes de ODL necesarios. El clasificador (arquitectura SFC indicada en la Figura 5) utilizado se implementa a través del proyecto Netvirt ODL. La Tabla 1 incluye todos los componentes efectivamente instalados:

SFC features	Netvirt features
odl-sfc-model	odl-netvirt-openstack
odl-sfc-provider	odl-sfc-genius
odl-sfc-provider-test	odl-neutron-service
odl-sfc-ovs	odl-neutron-nothbound-api
odl-sfc-opensflow-render	odl-neutron-spi
	odl-neutron-transcriber
	odl-ovsdb-southbound-impl-api
	odl-ovsdb-southbound-impl-impl
	odl-ovsdb-library

Tabla 1. Componentes del proyecto Netvirt instalados

Todas las características dependientes de odl-netvirt-sfc, se instalarán.

La funcionalidad necesaria de VNF Manager se implementa mediante la utilización del módulo Tacker de Openstack. Tacker se instala en el nodo Controler00 y gestiona los tiempos de vida de las VNF y coordina la creación de las VM y la configuración de los SFC entre Openstack y ODL SFC.

Una vez que se instala el escenario, es posible crear cadenas de servicio y entradas de clasificación que asocian los distintos tipos de tráfico en cadenas de servicio predefinidas. La configuración se realiza a través del CLI de Tacker.

Siguiendo las especificaciones del encapsulado SFC basado en NSH, los *switch* controlados por ODL requieren de la compatibilidad necesaria con dicha

[5] Project: Service Function Chaining, Project Lead: Manuel Buil
https://wiki.opnfv.org/display/sfc/Service+Function+Chaining+Home, acceso 22/11/2018

cabecera. En la versión instalada se dispone de un *switch* OVS que soporta encapsulado VXLAN-GPE + *ethernet* + NSH.

Las cabeceras *ethernet* o VXLAN-GPE en según qué escenarios de red son necesarias. En redes *ethernet* es necesario añadir la cabecera de transporte al paquete encapsulado mediante NSH para diferenciar a que nodo va dirigido el tráfico. Sin embargo, si el trafico van a atravesar diferentes redes IP, es necesario añadir al paquete encapsulado mediante NSH la cabecera VXLAN-GPE de nivel 3 para alcanzar el destino.

5.1.1. Despliegue del escenario os-odl-sfc-noha mediante Releng-xci

Como ya se ha indicado, atendiendo al equipamiento disponible y los requisitos del escenario, se realiza una instalación mediante Releng-xci del flavor mini en la estación de trabajo del laboratorio, a la que se instaló el sistema operativo OpenSuse Leap 42.3. Para realizar el despliegue efectivo, la configuración requiere, además, ciertos prerrequisitos *hardware/software*, como se indica a continuación:

Prerrequisitos

- Maquina *host* con suficientes recursos de CPU/RAM/Disco en función del *flavor* elegido (servidor del laboratorio adecuado a los requisitos).
- Ubuntu 16.04, OpenSUSE Leap 42.3 o CentOS 7 (OpenSUSE seleccionado)
- CPU/placa base que soporte virtualización asistida del *hardware*
- Ejecución “sudo” sin contraseña
- Claves SSH generadas por el usuario (es decir, ~/.ssh/id_rsa)
- Paquetes adicionales a instalar: git, python 2.7, pip y libvirt

De acuerdo a los prerrequisitos indicados, es necesario instalar los paquetes necesarios [54] para que el despliegue se realice con éxito: Git, Python2.7, Pip y Libvirt.

En la maquina host ejecutar:

```
sudo zypper in git
sudo zypper in python
sudo zypper in python-pip
sudo zypper in libvirt
```

Una vez instalados los paquetes se procederá a desplegar el sistema tal y como se indica en la documentación oficial del escenario deseado [10].

Lo primero de todo es configurar nuestro equipo de manera que se pueda ejecutar la sentencia sudo sin introducir contraseña.

En el fichero /etc/sudoers

```
%users ALL=(ALL) NOPASSWD: ALL
```

Siendo “users” el grupo que contiene los usuarios de la máquina.

1. Crear par de claves RSA en caso de no disponer de ellas.

```
ssh-keygen -t rsa
```

2. Clonar el repositorio que contiene el proyecto

```
git clone https://gerrit.opnfv.org/gerrit/releng-xci.git
```

3. Situarse en el directorio de instalación

```
cd releng-xci/xci
```

4. Definir las variables necesarias para el despliegue y ejecutar ponerlo en marcha.

```
export XCI_FLAVOR=mini
export DEPLOY_SCENARIO=os-odl-sfc
./xci-deploy.sh
```

XCI_FLAVOR=mini define la configuración mini. Esta está formada por 3 nodos y requiere 12 GB de memoria RAM y 80 GB de almacenamiento en disco.

DEPLOY_SCENARIO=os-odl-sfc define el escenario que se va a desplegar, en nuestro caso Openstack con ODL y SFC integrados. Este escenario contiene los plugin necesarios para trabajar con SFC. Sin embargo, cuenta con los módulos básicos de Openstack, esto permite virtualizar funciones de red de mismo modo sin necesidad de añadirle SFC. Gracias a esto, en el desarrollo técnico, se ha utilizado os-odl-sfc tanto para un escenario básico de red sin SFC como para un escenario en el que se estudia SFC.

Por último, se ejecuta el script de Shell con el que se automatiza la ejecución. Si se trabaja en remoto, es recomendable utilizar la sentencia nohup de manera que, si se cierra la sesión SSH, el proceso ignorará la señal SIGHUP(posix). De este modo no se detendrá la ejecución de la instalación. Además, la salida estándar se guardará en un fichero llamado “nohup.out”, lo cual puede ser interesante para ver los logs de la instalación.

```
nohup ./xci-deploy.sh
```

Tras aproximadamente tres horas de ejecución ya tendremos listo nuestro sistema para comenzar a trabajar con él.

5.1.2. Arquitectura de red

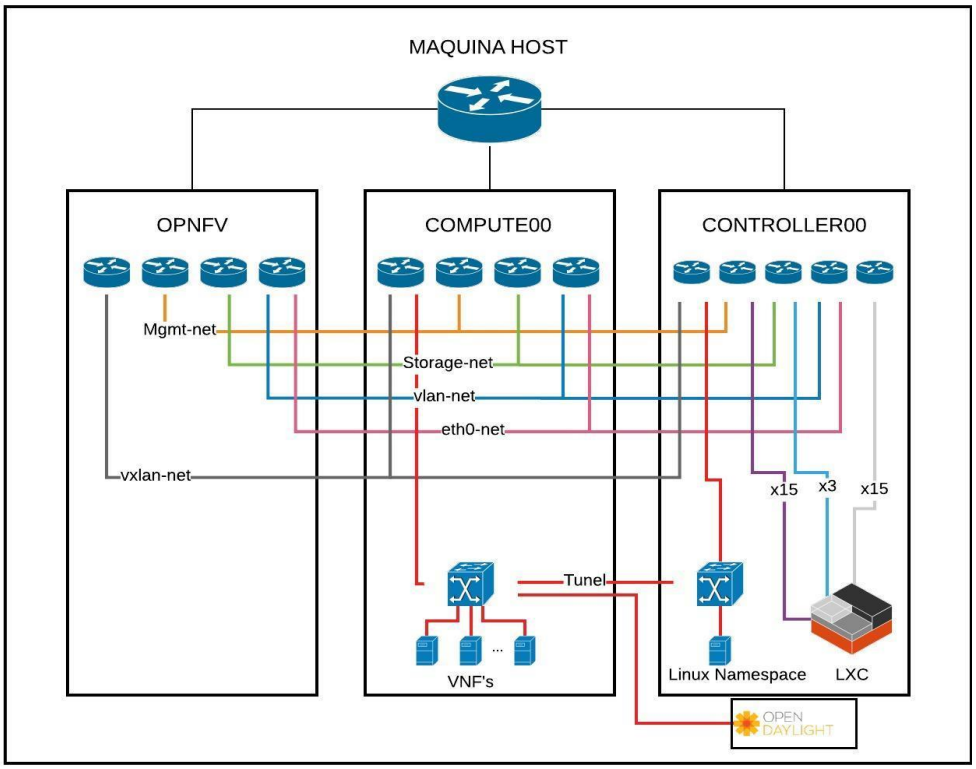


Figura 20: Arquitectura de red del despliegue releng-xci

Tras completar el despliegue, tal y como se puede observar en la Figura 20, en la maquina *host* se crean las tres máquinas virtuales OpenSuse Leap 42.3 correspondientes a los tres nodos de Openstack identificados previamente. También se crea un *bridge* con tres interfaces virtuales para interconectar los tres nodos. Se puede acceder a ellas a través de SSH en las siguientes direcciones.

Opnfv	192.168.122.2
Controller00	192.168.122.3
Compute00	192.168.122.4

También es posible acceder a la interfaz gráfica a través del *dashboard* de Horizon UI (Figura 21):

https://192.168.122.3

Figura 21: GUI dashboard Horizon

Las tres máquinas están conectadas a través de VLAN. No se va a detallar el flujo de tráfico que cursan todas las redes salvo las de interés. De manera resumida, el tráfico que cursan estas subredes se relaciona con información que comparte la maquina Opnfv con las otras dos máquinas a la hora de interactuar con el sistema *cloud*. Como ya se ha comentado, la maquina Compute00 aloja las VNF y la maquina Controller00 se encarga de la configuración y gestión de la nube. Por lo tanto, el trabajo que se realice sobre la maquina Opnfv, será comunicado a través de estas redes a las maquinas Compute00 y Controller00.

Por otro lado, para comprender el funcionamiento del sistema resulta interesante entender la existencia de los dos *switch* OVS y los contenedores LXC.

En primer lugar, en la maquina Compute00 se crea un Open vSwitch el cual interconectará las posibles VNF desplegadas. Como ejemplo, a continuación, se puede ver en la Figura 22 el estado del *switch* con cinco VNF desplegadas.

```
compute00:~ # ovs-vsctl show
a8068b32-3fla-401a-a990-1b5608575255
    Manager "tcp:172.29.237.197:6640"
    Bridge br-int
        Controller "tcp:172.29.237.197:6653"
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
        Port "eth12"
            Interface "eth12"
        Port "tapb83e5cd1-97"
            Interface "tapb83e5cd1-97"
        Port "tun546acdc9614"
            Interface "tun546acdc9614"
                type: vxlan
                options: {exts=gpe, key=flow, local_ip="172.29.240.12", remote_ip="172.29.240.11"}
        Port "tape7940275-5f"
            Interface "tape7940275-5f"
        Port "tapldcleaa7-12"
            Interface "tapldcleaa7-12"
        Port "tap1947fbc5-a6"
            Interface "tap1947fbc5-a6"
        Port "tapalbc7e8b-fe"
            Interface "tapalbc7e8b-fe"
    ovs_version: "2.9.2"
```

Figura 22: Open vSwitch Compute00

Las interfaces "tap" corresponden con las VNF mientas que la interfaz "tun" corresponde con el túnel que conecta con el *switch* de la maquina Controller00.

También se puede observar en la Figura 23 cómo el servicio de nova-compute, que es el que aloja las funciones virtuales, se ejecuta en esta máquina.

```
compute00:~ # sudo service --status-all | grep nova
nova-compute.service                    loaded active running nova-compute service
```

Figura 23: Servicio Nova Compute00

En segundo lugar, la maquina Controller00 contiene un *switch* similar al de la máquina Compute00. En este *switch* se conectarán los *qrouter y qdhcp* de openstack para posibilitar la conectividad de las VNF del nodo Compute00. *Qrouter y qdhcp* hacen referencia a los diferentes *network namespaces* [40] [24] de Linux que crea Neutron para dar conectividad con el exterior a las VNF. Concretamente permiten asociar *floating-ip*[6] de Openstack a las VNF.

En la Figura 24 se pueden ver los *switch* del nodo Compute00 y Controller00 y su interconexión.

[6] *Floating-ip*: hace referencia a las direcciones IP públicas de las que dispone la maquina host. Se pueden asociar *floating-ip* a las VNF para proporcionarles conectividad con el exterior del entorno *cloud.*
Fixed-ip: hace referencia a las direcciones IP privadas asociadas a las VNF dentro den entrono *cloud.* No son accesibles desde fuera, para ello se debe de asociar una *floating-ip* a la *fixed-ip.*

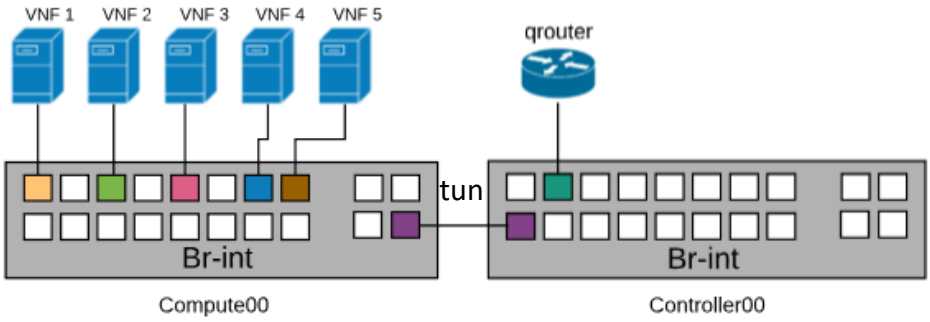


Figura 24. Esquema de conexión switch OVS

Tal y como muestra la Figura 25, el nodo Controller00 aloja los contenedores LXC que ejecutan los servicios de Openstack detallados previamente en el apartado Openstack. Cada contenedor tiene el nombre del servicio que ejecuta. En caso de tener problemas con algún servicio habría que resolverlo accediendo al contenedor específico.

El contenedor de Neutron es de gran interés puesto que tiene alojado el controlador ODL. En caso de ser necesario nos podemos comunicar con él mediante peticiones HTTP y mensajes REST.

```
controller00:~ # lxc-ls -f
NAME                                          STATE   AUTOSTART GROUPS           IPV4                                                  IPV6
controller00_ceilometer_central_container-9edc6434 RUNNING 1         onboot, openstack 10.0.3.45, 172.29.239.207                             -
controller00_cinder_api_container-0405c90f    RUNNING 1         onboot, openstack 10.0.3.68, 172.29.237.126, 172.29.247.110           -
controller00_cinder_volumes_container-46b3057f RUNNING 1         onboot, openstack 10.0.3.219, 172.29.237.166, 172.29.245.248          -
controller00_galera_container-175971b1        RUNNING 1         onboot, openstack 10.0.3.57, 172.29.238.220                           -
controller00_glance_container-f024724e        RUNNING 1         onboot, openstack 10.0.3.135, 172.29.238.11, 172.29.244.169           -
controller00_heat_api_container-076cedc7      RUNNING 1         onboot, openstack 10.0.3.36, 172.29.237.169                           -
controller00_horizon_container-284d0caa       RUNNING 1         onboot, openstack 10.0.3.91, 172.29.237.6                             -
controller00_keystone_container-53d09f68      RUNNING 1         onboot, openstack 10.0.3.130, 172.29.238.144                          -
controller00_memcached_container-d7e12e04     RUNNING 1         onboot, openstack 10.0.3.235, 172.29.239.236                          -
controller00_neutron_server_container-6b3d8f3a RUNNING 1         onboot, openstack 10.0.3.38, 172.29.237.197                           -
controller00_nova_api_container-93395cb1      RUNNING 1         onboot, openstack 10.0.3.101, 172.29.238.57                           -
controller00_rabbit_mq_container-c32e762a     RUNNING 1         onboot, openstack 10.0.3.25, 172.29.239.19                            -
controller00_repo_container-9a665b39          RUNNING 1         onboot, openstack 10.0.3.146, 172.29.239.197                          -
controller00_tacker_container-17defc8c        RUNNING 1         onboot, openstack 10.0.3.22, 172.29.236.234                           -
controller00_utility_container-b5cf5f6d       RUNNING 1         onboot, openstack 10.0.3.191, 172.29.238.204                          -
```

Figura 25: Contenedores LXC Controller00

5.2. Casos de estudio

A continuación, se describen con detalle los escenarios de pruebas considerados en el trabajo, con los que se pretende estudiar tanto la viabilidad de NFV como tecnología flexible de configuración de red como su aplicación en el diseño de cadenas de servicios (SFC).

Se van a estudiar dos casos de uso. El primero de ellos se basará en una red básica, con el objetivo principal de aprender los mecanismos de virtualización de funciones de red y desplegar una pequeña red funcional.

En el segundo caso de estudio se introducirá el concepto de SFC y se analizará las capacidades que ofrece.

5.2.1. Escenario 1: Virtualización de red básica

Para el primer escenario, cuya topología de red se muestra en la Figura 26, se va a definir una red básica con el objetivo de aprender a usar el sistema.

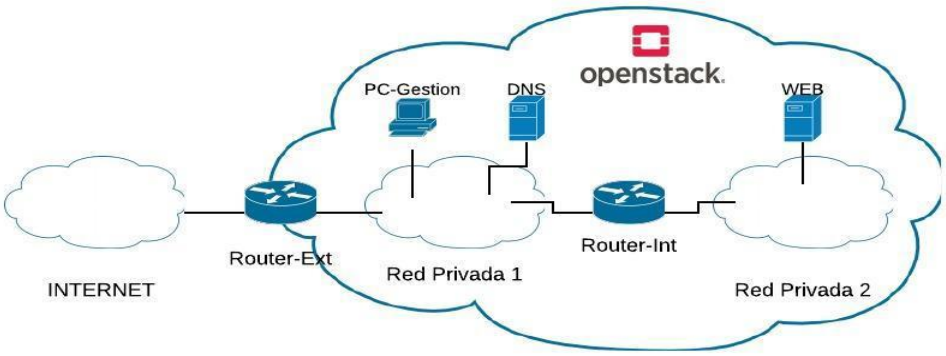


Figura 26: Topología de red escenario 1

La red está formada por dos redes privadas interconectadas por "Router-Int". La "Red Privada 1", está conectada con "Router-Ext" para dar conectividad a la red con el exterior. El servidor DNS será el servidor autorizado del dominio al que pertenece el servidor WEB (lan2.com).

El objetivo principal de este escenario es entender el funcionamiento básico de NFV estudiando el despliegue, configuración y puesta en marcha de funciones de red clásicas, como alternativa a una configuración manual en equipamiento real. Se ha optado, en este caso, por configurar un escenario sencillo con 4 SF (PC-Gestión, DNS, Router-Int y servidor WEB) representativas de un escenario de conectividad de red básico, con énfasis en el estudio de 2 funciones de red clave, un *router* y un servidor DNS. Así, las VNF desplegadas realizarán la siguiente función:

- PC-gestión: equipo utilizado para actuar como cliente dentro de la red virtual creada y facilitar el estudio del funcionamiento de las funciones de red de interés (*router* o DNS). VNF basada en una máquina Linux (Ubuntu 16.04 Xenial Xerus)
- DNS: servidor autorizado del dominio **lan2.com** y DNS local (*resolver*) para el PC-gestión. VNF basada en una máquina Linux (Ubuntu 16.04 Xenial Xerus) con la funcionalidad de servidor DNS proporcionada por el *software* bind 9.0.
- Router-Int: *router* de interconexión entre las dos redes privadas creadas en el entorno de Openstack, con funcionalidad de encaminamiento para

el acceso al servidor WEB. VNF basada en una máquina Linux (Ubuntu 16.04 Xenial Xerus) con capacidad de reenvío y funcionalidad iptables.

- WEB: servidor web, con nombre **WEB.lan2.com**. VNF basada en una máquina Linux (Ubuntu 16.04 Xenial Xerus) con funcionalidad se servidor web emulada por el *software* Netcat.

A la hora de configurar la red en el despliegue del laboratorio realizado, hay que tener en cuenta las características propias de Openstack y su modo de gestionar la conectividad de red: Openstack implementa redes virtuales, a las que asigna direcciones privadas (*fixed-ip*) y les da conectividad con el exterior (con direcciones públicas *floating-ip*[7]) mediante sus propios *router* virtuales (Router-Ext) a través del módulo Neutron-L3. Dichos *router*, entre otras cosas, realizan la traducción de direcciones necesaria –*Network Address Translation* (NAT) – entre *fixed-ip* y *floating-ip.* Para poder dar conectividad a una SF con la red externa (asignarle una *floating-ip),* esta debe estar alojada en una red en la cual esté conectado dicho *router* virtual de Openstack.

Teniendo todo esto en cuenta, al servidor WEB del escenario no se le puede asignar una *floating-ip*, por lo que no será accesible desde el exterior, quedando la evaluación por tanto dentro del escenario privado virtual.

Despliegue del escenario:

El despliegue de las VNF se realiza mediante Tacker, que actúa como el componente MANO de la arquitectura ETSI-NFV (concretamente implementando VNFM y NFVO). Para poder gestionar un VNF, el MANO necesita datos asociados a la misma, lo que se recoge en el VNF Descriptor (VNFD), una plantilla que describe los requisitos de la VNF (CPU, memoria…), cómo configurarla, y cómo manejar su ciclo de vida. Tacker utiliza VNFD basados en plantillas TOSCA (archivos YAML)[45] [25]. Para entender el detalle completo, el Anexo 1 recoge toda la información necesaria. A continuación, la Figura 27 muestra un ejemplo sencillo. Tal y como muestra dicha figura, un **VNFD** está formado por cuatro secciones:

- *Virtual Deployment Unit* (VDU): sección donde se definen las características generales y de cómputo de la VNF como, por ejemplo, nombre, imagen del sistema operativo, RAM, memoria en disco, etc.
- *Connection Points* (CP): sección donde se define la asociación entre el VDU y el VL.
- *Virtual Link (VL)*: proporciona conectividad a la VDU. Define los parámetros de red.
- *Floating IP* (FIP) (opcional): se define la asociación de una *floating-ip* a un CP para proporcionar conectividad con una red pública.

[7] Recordatorio. *Floating-ip*: hace referencia a las direcciones IP públicas de las que dispone la maquina host. Se pueden asociar *floating-ip* a las VNF para proporcionarles conectividad con el exterior del entorno *cloud.*
Fixed-ip: hace referencia a las direcciones IP privadas asociadas a las VNF dentro den entrono *cloud.* No son accesibles desde fuera, para ello se debe de asociar una *floating-ip* a la *fixed-ip.*

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
description: Ejemplo de VNFD
metadata:
 template_name: sample-tosca-vnfd-template-guide
topology_template:
 node_templates:
  VDU:
   type: tosca.nodes.nfv.VDU.Tacker
   properties:
    image: Imagen a usar por la VNF
    flavor: Flavor a usar por la VNF
    availability_zone: zona de disponibilidad (nova)
    user_data: comandos a ejecutar en tiempo de creación
    user_data_format: formato de los comandos
    key_name: clave de usuario

  CP:
   type: tosca.nodes.nfv.CP.Tacker
   properties:
    security_groups: lista de grupos de seguridad (restricciones de tráfico)
   requirements:
    - virtualLink:
       node: VL asociado a este CP
    - virtualBinding:
       node: VDU asociado a este CP
  VL:
   type: tosca.nodes.nfv.VL
   properties:
    network_name: nombre de la red a conectar
    vendor: Tacker
  FIP:
   type: tosca.nodes.network.FloatingIP
   properties:
    floating_network: red externa donde crear la floating-ip
    floating_ip_address: floating-ip a asociar
   requirements:
    - link:
       node: CP asociado a la floating ip
```

Figura 27. Plantilla VNFD de TOSCA para Tacker

El Anexo 3 recoge todos los pasos a seguir para realizar el despliegue, así como todos los VNFD necesarios. Cabe destacar que, los VNFD incluyen no solo parámetros clave para realizar su despliegue, sino para una correcta configuración del funcionamiento mediante comandos de *Shell* a ejecutar en arranque. De este modo, por ejemplo, se incluyen los archivos de zona necesarios del DNS autorizado y los parámetros de configuración del mismo (named.conf) (detalle en el fichero DNS.yaml del Anexo 4).

A continuación, se mostrarán los resultados y estudios realizados.

Análisis del escenario:

La Figura 28, muestra las VNF creadas. Se pueden observar las direcciones IP (*fixed-ip*) que toman cada VNF en las redes a las que esta conectadas, que van a resultar importantes para el análisis de los resultados.

```
linux:~ # openstack server list
+--------------------------------------+--------------+--------+-----------------------------------------------------------------------+-------------+--------------+
| ID                                   | Name         | Status | Networks                                                              | Image       | Flavor       |
+--------------------------------------+--------------+--------+-----------------------------------------------------------------------+-------------+--------------+
| 810edfa4-5b62-4e85-a263-2dd64455774a | WEB          | ACTIVE | Red-Privada-2=20.20.20.5                                              | Ubuntu-Image | Little-Flavor |
| d13b5710-8132-4ef7-a97d-2b6aecb0fe89 | Router-Ubuntu | ACTIVE | Red-Privada=10.10.10.9, 192.168.122.13; Red-Privada-2=20.20.20.1 | Ubuntu-Image | Little-Flavor |
| 9369bb7c-5d5d-4550-b282-ce4c04340106 | DNS-server   | ACTIVE | Red-Privada=10.10.10.10, 192.168.122.11                               | Ubuntu-Image | Little-Flavor |
| 600979b9-1d79-4603-a66e-936622550676 | PC_Gestion   | ACTIVE | Red-Privada=10.10.10.4, 192.168.122.12                                | Ubuntu-Image | Little-Flavor |
+--------------------------------------+--------------+--------+-----------------------------------------------------------------------+-------------+--------------+
```

Figura 28. VNF desplegadas

Una vez desplegadas las VNF se va a proceder a verificar su funcionamiento.

En primer lugar, se analizará el funcionamiento del servidor DNS, para ello se empleará el programa **dig.** Se trata de un *software* para generar una *DNS query* indicando el servidor DNS al cual le quieres realizar la petición. Se preguntará desde PC-Gestión al servidor DNS por WEB.lan2.com.

En la Figura 29 se observa el correcto funcionamiento del servidor DNS.

```
ubuntu@pc-gestion:~$ dig @10.10.10.10 WEB.lan2.com

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @10.10.10.10 WEB.lan2.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34658
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;WEB.lan2.com.                  IN      A

;; ANSWER SECTION:
WEB.lan2.com.           86400   IN      A       20.20.20.5

;; AUTHORITY SECTION:
lan2.com.               84600   IN      NS      DNS.lan1.com.

;; ADDITIONAL SECTION:
DNS.lan1.com.           6884    IN      A       209.17.116.8

;; Query time: 23 msec
;; SERVER: 10.10.10.10#53(10.10.10.10)
;; WHEN: Thu Nov 22 02:35:21 UTC 2018
;; MSG SIZE  rcvd: 96
```

Figura 29. DNS Query

En segundo lugar, se verificará la conectividad entre “PC-Gestión” y “WEB” mediante *ping* (Figura 30), utilizando directamente su dirección IP (mostrada en la respuesta del DNS de la Figura 29). De este modo se verifica la funcionalidad correcta de encaminamiento del Router-Int.

```
ubuntu@pc-gestion:~$ ping -c 4 20.20.20.5
PING 20.20.20.5 (20.20.20.5) 56(84) bytes of data.
64 bytes from 20.20.20.5: icmp_seq=1 ttl=63 time=7.46 ms
64 bytes from 20.20.20.5: icmp_seq=2 ttl=63 time=5.32 ms
64 bytes from 20.20.20.5: icmp_seq=3 ttl=63 time=3.68 ms
64 bytes from 20.20.20.5: icmp_seq=4 ttl=63 time=4.20 ms

--- 20.20.20.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 3.685/5.169/7.466/1.453 ms
```

Figura 30. Ping entre PC-Gestión y WEB

Por último, resulta interesante ver el funcionamiento completo del servidor DNS y del servidor WEB a través de una petición HTTP desde PC-Gestión a la URL del servidor WEB.

El primer paso será configurar el servidor WEB. Al tratarse de un entorno de evaluación y dadas las propias limitaciones de Openstack comentadas anteriormente en relación al direccionamiento IP, se ha considerado suficiente trabajar con Netcat como emulador de un servicio WEB.

A continuación, se muestran los pasos de configuración. Al igual que el resto de configuración de las VNF, también se podría incluir en el VNFD. Sin embargo, resulta interesante mostrarlo en esta sección.

Debido a que el servidor WEB no dispone de *floating-ip*, nos conectaremos por SSH desde PC-Gestión.

En PC-Gestión:

```
ssh –i id_rsa ubuntu@20.20.20.5
```

A continuación, se empleará *Netcat* para atender peticiones en el Puerto 80.

```
while true; do echo "Hello" | nc -l 20.20.20.5 80; done
```

Una vez arrancado el servidor WEB, se configurará el PC-Gestión de manera que su *nameserver* sea el servidor DNS. En el archivo resolv.conf se añadirá la siguiente línea.

```
nameserver 10.10.10.10
```

Por último, se realizará la petición HTTP.

```
ubuntu@pc-gestion:~$ nc WEB.lan2.com 80
Hello
```

Figura 31. Petición WEB

Como se puede observar en la Figura 31, se realiza la petición a la URL *WEB.lan2.com*, la resolución de nombres funciona correctamente y la petición llega al servidor. Finalmente recibimos la respuesta.

En definitiva, realizadas estas pruebas básicas se comprueba el correcto funcionamiento de la virtualización de funciones de red, que ha permitido desplegar una red básica y funcional (en el escenario de laboratorio de Openstack) sin la configuración directa de cada una de las máquinas de red.

5.2.2. Escenario 2: Despliegue de red con SFC

En el desarrollo del segundo escenario se pretende estudiar el funcionamiento y el flujo de tráfico en una red con SFC. Para ello, al igual que en el escenario previo, se van a crear diversas VNF, solo que, en este caso, dos de ellas se corresponden con la fuente y destino de los flujos de tráfico, mientras que las otras tres se corresponden con tres SF en el dominio SFC a emular.

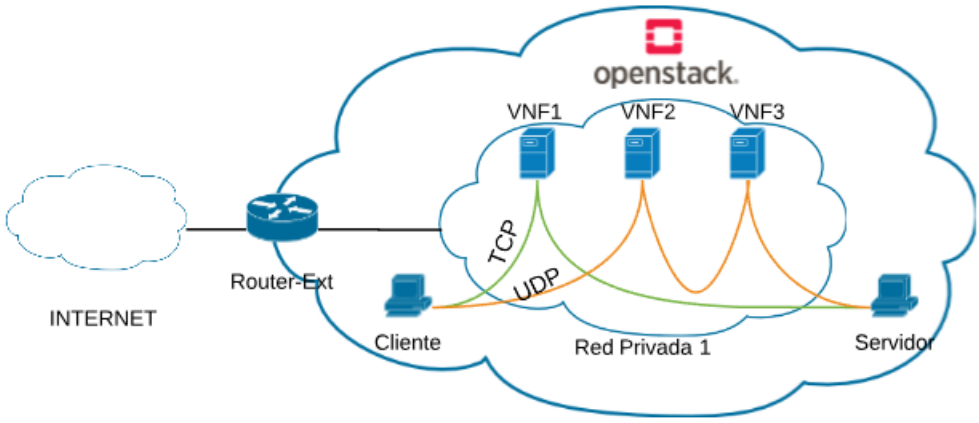


Figura 32. Topología de red escenario 2

Concretamente, tal y como muestra la topología de red en la Figura 32, la red consta de una máquina Cliente con la que se generará tráfico de diferentes tipos y una máquina Servidor que, en nuestro caso, alojará un servidor Web básico. Las otras tres VNF formarán parte, de manera selectiva, de dos cadenas de funciones de servicio. La primera de ellas, denominada HTTP-PORT-CHAIN, aplicable a tráfico TCP, estará formada por la VNF1 ("Control de acceso"). La segunda, UPD-PORT-CHAIN, a atravesar por el tráfico UDP, estará formada por VNF2 ("reenvío") y VNF3 ("*firewall*"). La funcionalidad de dichas cadenas de servicio se describe a continuación:

Funcionamiento:

El flujo de tráfico SFC del escenario se puede observar en la Figura 33.

- Cliente: generara tráfico TCP o UDP destinado a la máquina Servidor o a Internet.
- VNF1: "Control de acceso" que analizará el tráfico TCP con origen en el Cliente. Descartará los paquetes dirigidos a una URL no permitida.
- VNF2: primera VNF en la cadena para UDP que únicamente reenvía tráfico. El objetivo es analizar el contenido de los paquetes (concretamente el encapsulado NSH) que deben atravesar varias VNF en una cadena.

- VNF3: actuará como *firewall* para el tráfico UDP con el puerto destino deseado.
- Servidor: servidor web escuchando en el puerto 80

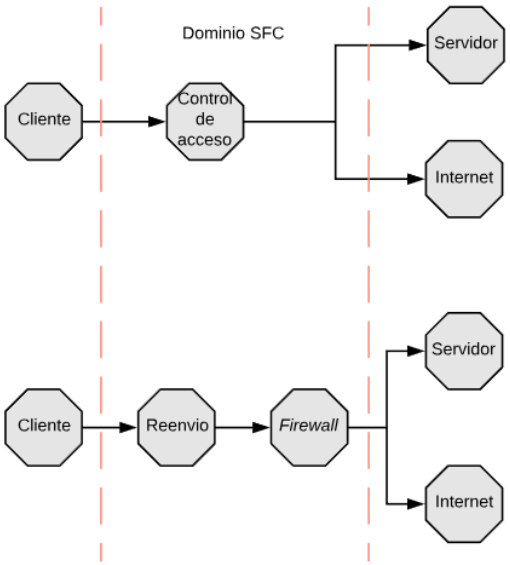


Figura 33. Cadenas SFC creadas

Desarrollo:

La configuración del escenario (comandos y VNFD correspondientes a cada SF) se detalla en el Anexo 4. A continuación, se mostrarán los resultados y estudios realizados.

En la Figura 34 se pueden observar las VNF creadas. En la Figura 35 se detallan los dos clasificadores de flujo creados. Como se puede observar, se ha creado un clasificador de flujo para todo tipo de tráfico UDP y otro para el tráfico TCP dirigido al puerto 80.

```
linux:~ # openstack server list
+--------------------------------------+-----------------+--------+-----------------------------------------+--------------+--------------+
| ID                                   | Name            | Status | Networks                                | Image        | Flavor       |
+--------------------------------------+-----------------+--------+-----------------------------------------+--------------+--------------+
| c60d97fa-6b06-42ba-a57c-4a3dba3794b8 | Cliente         | ACTIVE | Red-Privada=10.10.10.10, 192.168.122.10 | Ubuntu-Image | Little-Flavor |
| ab709349-f0ec-4586-ba01-4463db8816ac | Servidor        | ACTIVE | Red-Privada=10.10.10.12, 192.168.122.14 | Ubuntu-Image | Little-Flavor |
| bdfb5f74-dae3-45dc-8e47-db78b3176a38 | VNF-3           | ACTIVE | Red-Privada=10.10.10.5, 192.168.122.13  | Ubuntu-Image | Little-Flavor |
| 4b265837-12a9-4360-b193-39fcb8459e82 | VNF-2           | ACTIVE | Red-Privada=10.10.10.26, 192.168.122.12 | Ubuntu-Image | Little-Flavor |
| e21cbab2-880d-4121-a54e-8e7b726e4ec2 | ParentalControl | ACTIVE | Red-Privada=10.10.10.13, 192.168.122.11 | Ubuntu-Image | Little-Flavor |
+--------------------------------------+-----------------+--------+-----------------------------------------+--------------+--------------+
```

Figura 34. Instancias VNF

```
linux:~ # openstack sfc flow classifier list
+--------------------------------------+-------------+-----------------------------------------------------------------+
| ID                                   | Name        | Summary                                                         |
+--------------------------------------+-------------+-----------------------------------------------------------------+
| b1cb5720-3508-4f34-bec8-7ce1db3992e6 | manage_udp  | protocol: UDP,                                                  |
|                                      |             | source[port]: any[any:any],                                     |
|                                      |             | destination[port]: any[any:any],                                |
|                                      |             | neutron_source_port: 1dc1eaa7-12c9-437a-ad51-37b527d7dd41,      |
|                                      |             | neutron_destination_port: None,                                 |
|                                      |             | l7_parameters: {}                                               |
| d2360f46-1fbe-4e7b-9dc6-f3db12583338 | HttpControl | protocol: TCP,                                                  |
|                                      |             | source[port]: any[any:any],                                     |
|                                      |             | destination[port]: any[80:80],                                  |
|                                      |             | neutron_source_port: 1dc1eaa7-12c9-437a-ad51-37b527d7dd41,      |
|                                      |             | neutron_destination_port: None,                                 |
|                                      |             | l7_parameters: {}                                               |
+--------------------------------------+-------------+-----------------------------------------------------------------+
```

Figura 35. Clasificadores de flujo SFC

Como se puede observar en la Figura 36, se han creado dos cadenas. A cada una de ellas se le asocia uno de los clasificadores de flujo creados anteriormente.

```
linux:~ # openstack sfc port chain list
+--------------------------------------+----------------------+--------------------------------------------------------
------------------------------+------------------------------------------+--------------------------------------------
-+----------+
| ID                                   | Name                 | Port Pair Groups
                             | Flow Classifiers                         | Chain Parameters
 | Chain ID |
+--------------------------------------+----------------------+--------------------------------------------------------
------------------------------+------------------------------------------+--------------------------------------------
-+----------+
| 3cfcb0bd-9d56-4cbf-9cea-62e408824bfc | vnffg-HTTP-port-chain | [u'00659904-fa8f-4a84-b20e-2e2b4b0b8db2']
                         | [u'd2360f46-1fbe-4e7b-9dc6-f3db12583338'] | {u'symmetric': False, u'correlation': u'mpls'}
 |       51 |
| 824caadf-95f8-4f92-975c-47f0e64d4494 | vnffg-UDP-port-chain  | [u'8f5eea0b-533a-426a-9a31-5e440eb9c0e9', u'21bf2bdd-18
93-4185-8597-30d847c29cfd'] | [u'b1cb5720-3508-4f34-bec8-7ce1db3992e6'] | {u'symmetric': False, u'correlation': u'mpls'}
 |       53 |
+--------------------------------------+----------------------+--------------------------------------------------------
------------------------------+------------------------------------------+--------------------------------------------
-+----------+
```

Figura 36. Cadenas SFC

A continuación, se muestra en la Figura 36 la topología de red creada.

Como muestran las Figuras 34 y 37, se detallan las *fixed-ip* asociadas a cada VNF, todas pertenecientes a la misma red (10.10.10.0/24). Estas permiten la conectividad entre maquinas en el entorno privado de Openstack.

Por otro lado, tal y como se ve en la Figura 34, cada *fixed-ip* tiene asociada una *floating-ip* pertenecientes a la red externa (192.168.122.0/24). Estas permiten la conectividad de las VNF con el exterior a través de, entre otras cosas, del NAT proporcionado por el *router* virtual "RouterExt".

La configuración de los flujos en el *switch* OVS se puede ver en el nodo Compute00 (Figura 38). Como se puede observar en dicha figura, los flujos correspondientes al SFC se instalan en la tabla 101. Es interesante el análisis de los siguientes campos.

- Load:0x29 / 0x38 -> NXM_NX_REG2[8..31] añade un campo a la cabecera con el valor 0x29 / 0x38, que en decimal es 41 / 56. Este campo corresponde con el SPI de cada cadena, 41 (TCP) y 56 (UDP).
- Load:0xff -> NXM_NX_REG2[8..31] inicializa el valor del SI a 0xff en hexadecimal, que en decimal es 255.

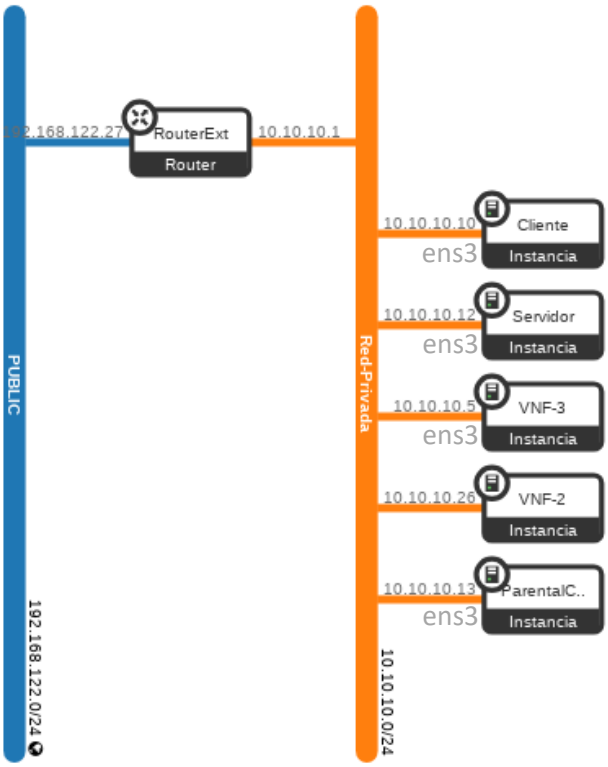


Figura 37. Escenario de red desplegado (dashboard Horizon)

```
ovs-ofctl -O Openflow13 dump-flows br-int | grep table=101
```

```
compute00:~ # ovs-ofctl -O Openflow13 dump-flows br-int | grep table=101
 cookie=0xf005ba1100000002, duration=1483417.581s, table=101, n_packets=209955, n_bytes=2046
0898, priority=500,tcp,in_port=15,tp_dst=80 actions=load:0x29->NXM_NX_REG2[8..31],load:0xff-
>NXM_NX_REG2[0..7],resubmit(,17)
 cookie=0xf005ba1100000002, duration=1483417.581s, table=101, n_packets=6343, n_bytes=160672
3, priority=500,udp,in_port=15 actions=load:0x38->NXM_NX_REG2[8..31],load:0xff->NXM_NX_REG2[
0..7],resubmit(,17)
 cookie=0xf005ba1100000002, duration=1483417.381s, table=101, n_packets=100405, n_bytes=1113
6538, priority=10 actions=resubmit(,17)
```

Figura 38. Flujos en el switch OVS

Una vez desplegada la red, su puesta en funcionamiento requiere añadir la configuración necesaria en las VNF que no ha sido posible añadir en tiempo de creación. Concretamente la funcionalidad de *proxy* SFC (apartado Arquitectura SFC) en las VNF que forman las cadenas SFC. Para ello nos conectamos a ellas con SSH a través de su *floating-ip*. Es necesario disponer en cada VNF del fichero *vxlan_tool.py* [19], una herramienta *Open Source* que actúa como *proxy* SFC, que permite a las VNF

comprender la cabecera NSH. Para este proyecto se ha modificado el fichero *vxlan_tool.py* con el fin de añadirle la funcionalidad de "Control Parental" además de las que ya incluye programadas.

Configuración de la VNF *Parental Control:*

Se lanza el *script* de Python *vxlan_tool.py* indicándole que reenvíe el tráfico que le llega por la interfaz ens3 actuando como control parental.

```
sudo nohup python vxlan_tool.py --d forward --interface ens3 --parentalControl 1 > /dev/null 2>&1 &
```

ó

```
sudo python vxlan_tool.py --d forward --interface ens3 --parentalControl 1
```

si se desea ver la información sobre el tráfico que se ofrece por pantalla.

Para el correcto funcionamiento, será necesario crear un fichero llamado *domainBlackList.txt* con las *Uniform Resource Locator* (URL) de las páginas WEB a restringir.

Los resultados, además de mostrarlos por pantalla, se almacenan en un fichero llamado registroGETs.txt

Configuración de la VNF2:

```
sudo python vxlan_tool.py --d forward --interface ens3
```

Únicamente se configura la VNF para que reenvíe el tráfico.

Configuración de la VNF3:

Se configura la VNF para que descarte el tráfico que reciba con puerto destino 5000, el resto de tráfico lo reenviará.

```
sudo python vxlan_tool.py --d forward --interface ens3 --block 5000
```

Configuración de la VNF Servidor:

Se configura un servidor web sencillo con python que atienda peticiones al puerto 80.

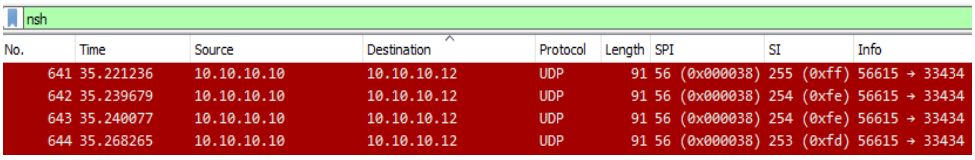
```
sudo python -m SimpleHTTPServer 80 > /dev/null 2>&1 &
```

Tras esta breve configuración, se procede a realizar las pruebas pertinentes.

Para analizar los flujos de tráfico a través de las cadenas de funciones de servicio se utiliza un programa de captura (*Wireshark*) en la máquina Compute00, habilitando la captura en todas las interfaces ya que este nodo contiene el *switch* que interconecta todas las VNF y puede así observarse todo el tráfico cursado a través de todos los puertos del *switch* OVS.

1) UDP-PORT-CHAIN: SFC para tráfico UDP

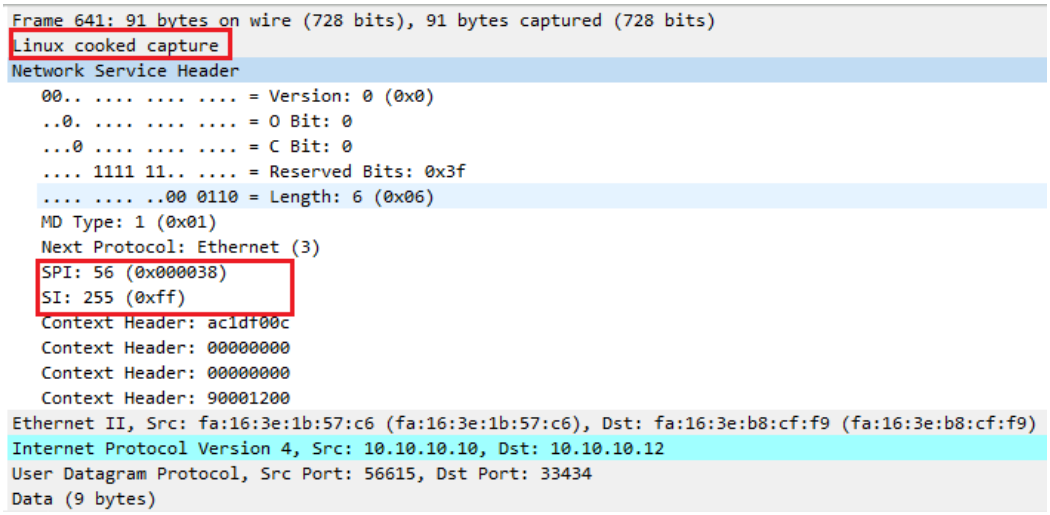
Se envía tráfico UDP desde el cliente al servidor, con puerto destino diferente a 5000.



nsh

No.	Time	Source	Destination	Protocol	Length	SPI	SI	Info
641	35.221236	10.10.10.10	10.10.10.12	UDP	91	56 (0x000038)	255 (0xff)	56615 → 33434
642	35.239679	10.10.10.10	10.10.10.12	UDP	91	56 (0x000038)	254 (0xfe)	56615 → 33434
643	35.240077	10.10.10.10	10.10.10.12	UDP	91	56 (0x000038)	254 (0xfe)	56615 → 33434
644	35.268265	10.10.10.10	10.10.10.12	UDP	91	56 (0x000038)	253 (0xfd)	56615 → 33434

Figura 39. Captura tráfico UDP SFC



```
Frame 641: 91 bytes on wire (728 bits), 91 bytes captured (728 bits)
Linux cooked capture
Network Service Header
    00.. .... .... .... = Version: 0 (0x0)
    ..0. .... .... .... = O Bit: 0
    ...0 .... .... .... = C Bit: 0
    .... 1111 11.. .... = Reserved Bits: 0x3f
    .... .... ..00 0110 = Length: 6 (0x06)
    MD Type: 1 (0x01)
    Next Protocol: Ethernet (3)
    SPI: 56 (0x000038)
    SI: 255 (0xff)
    Context Header: ac1df00c
    Context Header: 00000000
    Context Header: 00000000
    Context Header: 90001200
Ethernet II, Src: fa:16:3e:1b:57:c6 (fa:16:3e:1b:57:c6), Dst: fa:16:3e:b8:cf:f9 (fa:16:3e:b8:cf:f9)
Internet Protocol Version 4, Src: 10.10.10.10, Dst: 10.10.10.12
User Datagram Protocol, Src Port: 56615, Dst Port: 33434
Data (9 bytes)
```

Figura 40. Paquete completo

Tal y como se ve en las Figuras 39 y 40, el paquete original contiene un encabezado NSH el cual contiene los campos SPI y SI. A su vez contiene un encapsulado de transporte llamado *Linux Cooked Capture.* Esto es así debido a que la captura se ha realizado en el nodo Compute00 con el fin de capturar en todas las interfaces y ver todos los paquetes procesados por el *switch.* De esta forma no diferencia interfaces *ethernet.*

Si se captura únicamente en un dominio *ethernet*, por ejemplo, en una VNF, en lugar de ver la cabecera *Linux Cooked Capture* se ve una cabecera Ethernet II. Esta contiene las direcciones de nivel 2 correspondiente a las SF que va atravesando la cadena. Este es el llamado encapsulado de transporte necesario para el reenvió del paquete original entre las diferentes SF que forman la cadena. Si se tratara de tráfico que atraviesa diferentes redes IP, se ve una cabecera VXLA-GPE. En el ámbito de este proyecto, no se llega a ver esa cabecera debido a que se ha realizado una prueba de concepto SFC sobre SF en la misma red IP.

Como se observa en la Figura 39, el funcionamiento era el esperado, el campo SPI, que identifica a la cadena, permanece constante y el campo SI se decrementa en una unidad en cada salto que da en la cadena (cada SF atravesada por el tráfico). En la Figura 41, se detalla de manera gráfica el flujo del tráfico.

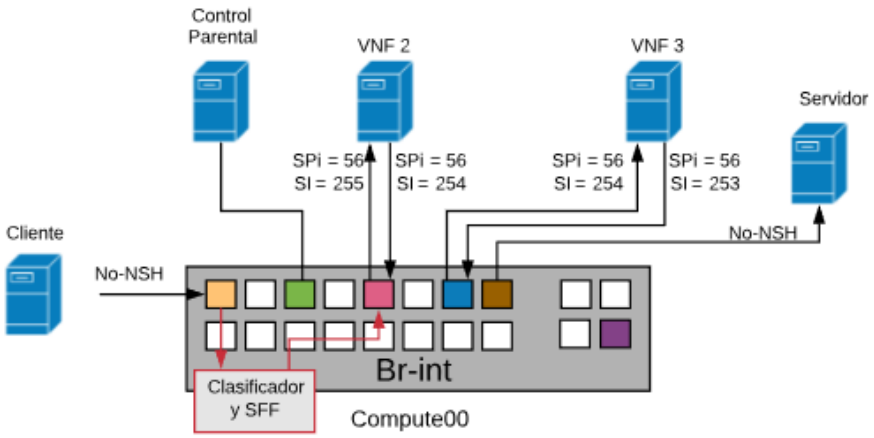


Figura 41. Flujo de tráfico UDP SFC

Ahora en lugar de *traceroute*, se va a enviar tráfico UDP desde el cliente al puerto 5000 del servidor.

```
ubuntu@cliente:~$ echo "This is my data" > /dev/udp/10.10.10.12/5000
```

Figura 42. Envió datos UDP al puerto 5000

```
ubuntu@vnf-3:~$ sudo python vxlan_tool.py --d forward --interface ens3 --block 5000
sudo: unable to resolve host vnf-3


Packet #1
Eth Dst MAC: fa:16:3e:e2:fa:a6, Src MAC: fe:16:3e:e2:fa:a6, Ethertype: 0x894f
NSH base nsp: 56, nsi: 255
NSH context c1: 0xac1df00c, c2: 0x00000000, c3: 0x00000000, c4: 0x90001200
 TCP packet dropped on port: 5000
```

Figura 43. Filtrado de paquetes al puerto 5000

Como representan las Figuras 42 y 43, la VNF3 está configurada como *firewall* para rechazar el tráfico dirigido al puerto 5000 por lo que descarta los paquetes.

La información observada en la Figura 43, corresponde con mensajes generados en *stdout* por la ejecución del fichero *vxlan_tool.py.*

2) CADENA 2: SFC para tráfico TCP

Se envía una petición HTTP desde el cliente al servidor.

```
curl 10.10.10.12
```

En la Figura 44 se puede observar la respuesta HTTP por parte del servidor.

```
ubuntu@cliente:~$ curl 10.10.10.12
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="bin/">bin/</a>
<li><a href="boot/">boot/</a>
<li><a href="dev/">dev/</a>
<li><a href="etc/">etc/</a>
<li><a href="home/">home/</a>
<li><a href="initrd.img">initrd.img@</a>
<li><a href="initrd.img.old">initrd.img.old@</a>
<li><a href="lib/">lib/</a>
<li><a href="lib64/">lib64/</a>
<li><a href="lost%2Bfound/">lost+found/</a>
<li><a href="media/">media/</a>
<li><a href="mnt/">mnt/</a>
<li><a href="opt/">opt/</a>
<li><a href="proc/">proc/</a>
<li><a href="resolvCP.conf">resolvCP.conf</a>
<li><a href="root/">root/</a>
<li><a href="run/">run/</a>
<li><a href="sbin/">sbin/</a>
<li><a href="snap/">snap/</a>
<li><a href="srv/">srv/</a>
<li><a href="sys/">sys/</a>
<li><a href="tmp/">tmp/</a>
<li><a href="usr/">usr/</a>
<li><a href="var/">var/</a>
<li><a href="vmlinuz">vmlinuz@</a>
<li><a href="vmlinuz.old">vmlinuz.old@</a>
</ul>
<hr>
</body>
</html>
```

Figura 44. Respuesta HTTP

En la Figura 45, se puede observar el comportamiento de la VNF *Parental Control.* Esta verifica que es un tráfico válido y deja pasar los paquetes.

```
NSH base nsp: 41, nsi: 255
NSH context c1: 0xac1df00c, c2: 0x00000000, c3: 0x00000000, c4: 0x90001200
Host: 10.10.10.12
Se ha permitido el acceso a Host: 10.10.10.12
```

Figura 45.Análisis de paquetes HTTP en VNF Parental Control

La información observada en la Figura 45, corresponde con mensajes generados en *stdout* por la ejecución del fichero *vxlan_tool.py.*

Por último, se envía una petición HTTP desde el cliente a *marca.es*. En este caso, se ha restringido el acceso a esta dirección web.

```
curl marca.es
```

En primer lugar, tal y como muestra la Figura 46 se observan paquetes DNS que atraviesa la cadena definida para el tráfico UDP. Es decir, la consulta DNS relativa a la URL atraviesa el filtro de la cadena UDP-PORT-CHAIN, al corresponderse con tráfico UDP al puerto 53, y no el puerto 5000. Sin embargo, el acceso web real, mediante HTTP a la dirección IP correspondiente, se filtra adecuadamente (cadena HTTP-PORT-CHAIN), tal y como muestra la Figura 47, en la que se observa como la VNF *Parental Control* descarta el tráfico dirigido a *marca.es*.

No.	Time	Source	Destination	Protocol	Length	SPI	SI	Info
2817	158.286044	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	255 (0xff)	Standard query 0x01ef A marca.es
2819	158.292017	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	255 (0xff)	Standard query 0x87be AAAA marca.es
2820	158.293316	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	254 (0xfe)	Standard query 0x01ef A marca.es
2821	158.300699	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	254 (0xfe)	Standard query 0x01ef A marca.es
2822	158.306747	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	254 (0xfe)	Standard query 0x87be AAAA marca.es
2823	158.306753	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	254 (0xfe)	Standard query 0x87be AAAA marca.es
2824	158.316527	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	253 (0xfd)	Standard query 0x01ef A marca.es
2828	158.337424	192.168.122.10	8.8.8.8	DNS	108	56 (0x000038)	253 (0xfd)	Standard query 0x87be AAAA marca.es

Figura 46. Trafico UDP DNS

```
NSH base nsp: 41, nsi: 255
NSH context c1: 0xacldf00c, c2: 0x00000000, c3: 0x00000000, c4: 0x90000400
Host: marca.es
Se ha denegado el acceso a Host: marca.es
```

Figura 47. Trafico HTTP denegado

La información observada en la Figura 47, corresponde con mensajes generados en *stdout* por la ejecución del fichero *vxlan_tool.py*.

6. Conclusiones y líneas futuras

Tras el desarrollo de los diferentes escenarios NFV y SFC y el estudio de sus características y aplicaciones principales, se presentan las conclusiones obtenidas a partir del trabajo realizado. A continuación de plantean posibles líneas de trabajo futuros con el fin de profundizar en estas tecnologías.

6.1. Conclusiones

En este trabajo de fin de grado se proponen varios objetivos parciales, de los cuales se diferencian dos objetivos principales, dominar el uso de Openstack como herramienta NFV y la evaluación practica de una implementación de red basada en *Service Function Chaining.*

Gracias a la herramienta Openstack es posible desarrollar una plataforma de computación en la nube que nos permite desplegar funciones de servicio virtuales. Esto nos ha permitido crear varios escenarios de red con el fin de comprender todos los aspectos de funcionamiento de la arquitectura NFV.

Por otro lado, SDN y SFC se propone como una potente alternativa a las técnicas de encaminamiento tradicionales, posibilitando introducir servicios de red personalizados de manera dinámica y eficiente.

Por último, a pesar de que el *software* Openstack desplegado por OPNFV se trata de un entorno de desarrollo, lo cual en ciertos momentos del proyecto ha generado problemas inesperados, los objetivos propuestos al inicio del proyecto has sido completados satisfactoriamente.

6.2. Trabajos futuros

Como se ha detallado durante el desarrollo del proyecto, NFV y SDN son dos tecnologías con un gran futuro por delante, por lo que resultaría interesante llevar a cabo los siguientes posibles trabajos futuros.

- **Estudio de encapsulados alternativos:** como se ha comentado, la cabecera NSH, aun estandarizada, no es la única opción de encapsulado SFC existiendo en la actualidad alternativas en investigación cuya comparativa sería de gran interés.
- **Realización de despliegues de laboratorio más complejos:** aun manteniendo un nivel de desarrollo preliminar, el estudio realizado, y en especial lo aprendido en el manejo de las herramientas, posibilita el avance en la realización de despliegues más complejos, incluso en escenarios de laboratorio, facilitando tanto la investigación en el campo de NFV y SDN (realizando pruebas más exhaustivas de prestaciones) como el aprendizaje de estas tecnologías a nivel académico.
- **Gestión de la red NFV mediante ODL:** como se ha comentado durante el proyecto, el despliegue de Openstack realizado incorporaba la capacidad de controlar la red mediante el controlador ODL. En el proyecto que nos ocupa, la gestión de la red no se ha realizado mediante ODL puesto que no entraba

en el alcance del trabajo. Sin embargo, resulta de gran interés la sinergia de ambas tecnologías.

- **Desarrollo de casos de uso reales:** en el desarrollo técnico de este proyecto se han realizado pequeños escenarios de red con el objetivo de realizar pruebas de concepto que nos permitan aprender las características principales de VNF y SFC. Sin embargo, en la industria IT, existen casos de uso reales como por ejemplo el *virtual Customer Premises Equipment* (vCPE) que reemplaza el actual CPE alojado en las instalaciones del usuario por un VCPE alojado en la nube del operador.

1 Bibliografía

[1] (ETSI), E. T. (2013). Network Functions Virtualisation (NFV); Architectural Framework. Obtenido de https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf

[2] (IETF), I. E. (2012). *Network Functions Virtualisation.* Obtenido de https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[3] A. Farrel, E. (s.f.). An Architecture for Use of PCE and the PCE Communication Protocol (PCEP). *Request for Coments (RFC 8283)*.

[4] *API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos*. (Marzo de 2016). Obtenido de https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[5] Arroyo, R. (1 de Octubre de 2014). *ChannelBiz. Operadores y fabricantes se unen en la Open Platform for NFV Project (OPNFV)*. Obtenido de https://www.channelbiz.es/2014/10/01/operadores-y-fabricantes-se-unen-en-la-open-platform-nfv-project-opnfv/?inf_by=5be09cd0671db884308b47aa

[6] Beierl, M. (Junio de 2017). *OPNFV. Project Directory*. Obtenido de https://wiki.opnfv.org/display/PROJ/Project+Directory

[7] Beierl, M. (Septiembre de 2018). *OPNFV. Fuel*. Obtenido de https://wiki.opnfv.org/display/fuel/Fuel+Opnfv

[8] Beierl, M. (s.f.). *OPNFV. Welcome to the OPNFV Wiki*. Obtenido de https://wiki.opnfv.org/

[9] Bilbao, N. (24 de Junio de 2016). *Silicon. OPNFV se revela como plataforma "esencial" para la industria Teleco*. Obtenido de https://www.silicon.es/opnfv-se-revela-plataforma-esencial-la-industria-teleco-2312314?inf_by=5be09b2c671db8284f8b5333

[10] Buil, M. (Noviembre de 2018). *OPNFV. Deploy OPNFV SFC scenarios*. Obtenido de https://wiki.opnfv.org/display/sfc/Deploy+OPNFV+SFC+scenarios

[11] *Cisco. Switches de Data Center*. (s.f.). Obtenido de https://www.cisco.com/c/es_es/products/switches/data-center-switches/index.html

[12] Cranford, N. (4 de 12 de 2017). *RCRWirelessNews. The role of NFV and SDN in 5G*. Obtenido de https://www.rcrwireless.com/20171204/fundamentals/the-role-of-nfv-and-sdn-in-5g-tag27-tag99

[13] *Docker. Future proof your Windows apps and drive continuous innovation*. (s.f.). Obtenido de https://www.docker.com/

[14] E. Haleplidis, E. (2015). Software-Defined Networking (SDN): Layers and Architecture Terminology. *Request for Comments (RFC 7426)*.

[15] *ETSI. Network Functions Virtualisation (NFV)*. (s.f.). Obtenido de https://www.etsi.org/technologies-clusters/technologies/nfv

[16] F. Clad, E. (2018). IETF. Service Programming with Segment Routing.

[17] Farrel, A. (2018). An MPLS-Based Forwarding Plane for Service Function Chaining . *(draft-ietf-mpls-sfc-04)*.

[18] Farrel, A. (s.f.). A Path Computation Element (PCE)-Based Architecture. *Request for Comments (RFC 4655)*.

[19] *GitHub.* (s.f.). Obtenido de https://github.com/opendaylight/sfc

[20] Ian F. Akyildiz, A. L. (s.f.). A roadmap for traffic engineering in SDN-OpenFlow networks.

[21] *Interoute*. (s.f.). Obtenido de https://www.interoute.es/what-iaas

[22] J. Halpern, E. (2015). Service Function Chaining (SFC) Architecture. *Request for Comments (RFC 7665)*.

[23] *Linux-kvm*. (s.f.). Obtenido de https://www.linux-kvm.org/page/Main_Page

[24] Man7. Linux Programmer's Manual NETWORK_NAMESPACES(7). (2018). Obtenido de http://man7.org/linux/man-pages/man7/network_namespaces.7.html

[25] *OASIS. Topology and Orchestration Specification for Cloud Applications Version 1.0*. (Noviembre de 2013). Obtenido de http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

[26] *Onosproject*. (s.f.). Obtenido de https://onosproject.org/

[27] *Open Source MANO*. (s.f.). Obtenido de https://osm.etsi.org/

[28] *OPENDAYLIGHT*. (s.f.). Obtenido de https://www.opendaylight.org/

[29] *Opendaylight. Service Function Chaining:Main*. (s.f.). Obtenido de https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

[30] *Opensource*. (s.f.). Obtenido de https://opensource.com/resources/what-is-openstack

[31] *OPENSTACK*. (s.f.). Obtenido de https://wiki.openstack.org/wiki/Governance/Foundation/Mission

[32] *OPENSTACK*. (s.f.). Obtenido de https://www.openstack.org/foundation/companies/

[33] *OPENSTACK. Swift*. (s.f.). Obtenido de https://wiki.openstack.org/wiki/Swift

[34] *OPENSTACK. Compute (Nova)*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/nova/latest/

[35] *OPENSTACK. Conceptual architecture*. (Enero de 2018). Obtenido de https://docs.openstack.org/newton/install-guide-ubuntu/common/get-started-conceptual-architecture.html

[36] *OPENSTACK. Heat Orchestration Template (HOT) Guide*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/heat/latest/template_guide/hot_guide.html

[37] *OPENSTACK. Heat-Translator*. (s.f.). Obtenido de https://wiki.openstack.org/wiki/Heat-Translator

[38] *OPENSTACK. Horizon: The OpenStack Dashboard Project*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/horizon/latest/

[39] *OPENSTACK. Keystone, the OpenStack Identity Service*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/keystone/latest/

[40] *OPENSTACK. Network namespaces*. (Enero de 2018). Obtenido de Https://docs.openstack.org/mitaka/networking-guide/intro-network-namespaces.html

[41] *OPENSTACK. Neutron/ML2*. (s.f.). Obtenido de https://wiki.openstack.org/wiki/Neutron/ML2

[42] *OPENSTACK. OpenStack Block Storage (Cinder) documentation*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/cinder/latest/

[43] *OPENSTACK. Service Function Chaining Extension for OpenStack Networking*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/networking-sfc/latest/

[44] *OPENSTACK. Tacker*. (s.f.). Obtenido de https://wiki.openstack.org/wiki/Tacker

[45] *OPENSTACK. VNF Descriptor Template Guide*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/tacker/latest/contributor/vnfd_template_description.html

[46] *OPENSTACK. Welcome to Ceilometer's documentation!* (Noviembre de 2018). Obtenido de https://docs.openstack.org/ceilometer/latest/

[47] *OPENSTACK. Welcome to Glance's documentation!* (Noviembre de 2018). Obtenido de https://docs.openstack.org/glance/latest/

[48] *OPENSTACK. Welcome to Neutron's documentation!* (Noviembre de 2018). Obtenido de https://docs.openstack.org/neutron/latest/

[49] *OPENSTACK. Welcome to Tacker Documentation*. (Noviembre de 2018). Obtenido de https://docs.openstack.org/tacker/pike/

[50] *OPENSTACK. Welcome to the Heat documentation!* (Noviembre de 2018). Obtenido de https://docs.openstack.org/heat/pike/

[51] OPNFV, C. (s.f.). *OPNFV*. Obtenido de https://www.opnfv.org/about/mission

[52] *OPNFV. 2. Sandbox and User Guide*. (s.f.). Obtenido de https://docs.opnfv.org/en/stable-fraser/submodules/releng-xci/docs/xci-user-guide.html#flavor-layouts-openstack-based-deployments

[53] *OPNFV. 2.3. Sandbox Flavors*. (s.f.). Obtenido de https://docs.opnfv.org/en/stable-fraser/submodules/releng-xci/docs/xci-user-guide.html#sandbox-flavors

[54] *OPNFV. 2.4.1. Prerequisites*. (s.f.). Obtenido de https://docs.opnfv.org/en/stable-fraser/submodules/releng-xci/docs/xci-user-guide.html#prerequisites

[55] *OPNFV. Installation*. (s.f.). Obtenido de https://docs.opnfv.org/en/stable-fraser/release/installation.introduction.html

[56] *OPNFV. os-odl-sfc-noha overview and description.* (s.f.). Obtenido de https://docs.opnfv.org/en/stable-fraser/submodules/sfc/docs/release/scenarios/os-odl-sfc-noha/

[57] *OPNFV. Upstream Projects*. (s.f.). Obtenido de https://www.opnfv.org/community/upstream-projects

[58] *OVS. Open vSwitch*. (s.f.). Obtenido de https://www.openvswitch.org/

[59] P. Quinn, E. (2015). Problem Statement for Service Function Chaining. *Request For Comments (RFC 7498)*.

[60] P. Quinn, E. (2018). Network Service Header (NSH). *Resquest for Comments (RFC 8300)*.

[61] *Sdxcentral. What is Network Service Chaining? Definition*. (s.f.). Obtenido de https://www.sdxcentral.com/sdn/network-virtualization/definitions/what-is-network-service-chaining/

[62] Sell, L. (Septiembre de 2012). *Bussiness Wire. OpenStack Launches as Independent Foundation, Begins Work Protecting, Empowering and Promoting OpenStack.* Obtenido de https://www.businesswire.com/news/home/20120919005997/en/OpenStack-Launches-Independent-Foundation-Begins-Work-Protecting

[63] Service Programming with Segment Routing. (s.f.). *Internet-Draft (draft-xuclad-spring-sr-service-programming-01)*.

[64] *The Apache software foundation*. (s.f.). Obtenido de https://www.apache.org/licenses/LICENSE-2.0

[65] *THE LINUX FOUNDATION*. (s.f.). Obtenido de https://www.linuxfoundation.org/

[66] *Tungstenfabric*. (s.f.). Obtenido de https://tungsten.io/

[67] Vidal, J. (Mayo de 2018). *OPNFV-SFC Release Plans*. Obtenido de https://wiki.opnfv.org/display/sfc/OPNFV-SFC+Release+Plans

[68] *VirtualBox. Welcome to VirtualBox.org!* (s.f.). Obtenido de https://www.virtualbox.org/

[69] *VMWare*. (s.f.). Obtenido de https://www.vmware.com/es.html

[70] *VMWare. ESXi*. (s.f.). Obtenido de https://www.vmware.com/es/products/esxi-and-esx.html

[71] *Wikipedia. OpenFlow*. (s.f.). Obtenido de https://es.wikipedia.org/wiki/OpenFlow

[72] *Wikipedia. Kubernetes*. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Kubernetes

[73] *Wikipedia. LXC*. (s.f.). Obtenido de https://es.wikipedia.org/wiki/LXC