UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

"TESTBED PARA EL ESTUDIO DE LA VIRTUALIZACIÓN DE LAS FUNCIONES DE RED NFV"

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

AUTOR: RAFAEL PATRICIO NOBOA MINDA

DIRECTOR: MSC. HERNÁN MAURICIO DOMÍNGUEZ LIMAICO

IBARRA-ECUADOR
2023

AUTORIZACIÓN DE USO Y PUBLICACIÓN
A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Identificación de la obra

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL CONTACTO			
CÉDULA DE IDENTIDAD	100376152-3		
APELLIDOS Y NOMBRES	Noboa Minda Rafael Patricio		
DIRECCIÓN	Huertos Familiares		
E-MAIL	rpnoboam@utn.edu.ec		
TELÉFONO FIJO	062601580	**TELÉFONO MÓVIL**	0999375793

DATOS DE LA OBRA	
TÍTULO	"TESTBED PARA EL ESTUDIO DE LA VIRTUALIZACIÓN DE LAS FUNCIONES DE RED NFV"
AUTOR	Noboa Minda Rafael Patricio
FECHA	Enero - 2023
PROGRAMA	Pregrado
TÍTULO	Ingeniero en Electrónica y Redes de Comunicación
DIRECTOR	Ing. Hernán Mauricio Domínguez Limaico, MsC.

Constancias

Constancias

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 09 del mes de enero de 2023

EL AUTOR

Noboa Minda Rafael Patricio

CI: 100376152-3

Certificacion

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

Certificación

MAGISTER MAURICIO DOMINGUEZ, DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN CERTIFICA:

Que, el presente trabajo de Titulación "TESTBED PARA EL ESTUDIO DE LA VIRTUALIZACIÓN DE LAS FUNCIONES DE RED NFV". Ha sido desarrollado por el señor Rafael Patricio Noboa Minda, bajo mi supervisión.

Es todo en cuanto puedo certificar en honor de la verdad.

.....

Ing. Hernán Mauricio Domínguez Limaico, MsC.

100237930-1

DIRECTOR

AGRADECIMIENTO

Mi sentimiento más profundo de gratitud a Dios y la vida, por darme la fuerza para luchar cada día y conseguir todos mis sueños. A mis padres y especialmente a mi madre y mis hermanos, el pilar de mi vida y mi motivación jamás habría logrado nada sin ellos.

Mi eterno agradecimiento a quienes más que mi familia son mis amigos y mi soporte Andrea y Alvaro por todo su cariño, preocupación y apoyo a lo largo de mi vida.

A mi director de tesis Ing. Mauricio Domínguez MSc. por ser mi guía y brindarme la oportunidad de formar parte del grupo de investigación del proyecto "TESTBED PARA EL ESTUDIO DE LA VIRTUALIZACIÓN DE LAS FUNCIONES DE RED NFV".

Agradezco infinitamente a quien fue mi compañera de vida en estos años de preparación profesional y personal, por su eterna paciencia, apoyo y amor, por ser mi amiga y ayudarme a crecer, Mercedes.

Finalmente, a mis compañeros y amigos Cristian y Paul, quienes me han brindado su apoyo y palabras de aliento y consejo. Y, a la Universidad Técnica del Norte por el compromiso con el desarrollo de este proyecto.

.

Rafael Patricio Noboa Minda

DEDICATORIA

El presente proyecto y todo el trabajo que conllevó el éxito del mismo, está dedicado a mi madre ejemplo de lucha, constancia y amor infinito, por jamás dejar de creer en mí e inculcarme los mejores valores. A mi padre y hermanos quienes forjan mis sueños y son mi inspiración.

Mi más profundo cariño...

Rafael Patricio Noboa Minda

ÍNDICE DE CONTENIDO

Contenido **Páginas**

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente trabajo de titulación presenta un Testbed para tecnología NFV con Open Source MANO (OSM), que es una plataforma de gestión y orquestación (MANO) de código abierto para redes virtualizadas. Esta plataforma proporciona los medios para gestionar y controlar funciones de red virtualizadas (VNFs) y servicios de red en un entorno virtual.

Es así como, para este proyecto se utiliza OSM en combinación con las plataformas OpenStack y OpenDaylight, que permite crear un entorno de red más centralizado, el cual despliega y gestiona configuraciones de red complejas propias para la virtualización de las funciones de red.

Por lo tanto, en este apartado se detalla una descripción general de alto nivel de cómo OSM, OpenStack y OpenDaylight pueden utilizarse para virtualizar funciones de red. Es decir, en primer lugar, se crea una VNF utilizando OSM. La creación de esta VNF es simple a través de plantillas predefinidas o cargando un paquete VNF existente. Seguidamente, la VNF se despliega sobre OpenDaylight, que proporciona la infraestructura y los recursos necesarios para ejecutar la VNF. Con OpenStack se logra gestionar la infraestructura de red y controlar el flujo de tráfico dentro de la red virtualizada, es decir, crear, gestionar y eliminar VNFs. Y por último con OSM se supervisa y gestiona la red virtualizada en su conjunto creada por VNFs. Por lo tanto, el testbed desplegado proporciona herramientas para proveer, supervisar y escalar funciones de red virtualizadas (VNFs) con los recursos de red, según sea necesario.

ABSTRACT

This degree work presents a Testbed for NFV technology with Open Source MANO (OSM), which is an open source management and orchestration platform (MANO) for virtualized networks. This platform provides the means to manage and control virtualized network functions (VNFs) and network services in a virtual environment.

Thus, for this project, OSM is used in combination with the OpenStack and OpenDaylight platforms to create a more centralized network environment, which deploys and manages complex network configurations for the virtualization of network functions.

Therefore, this section details a high-level overview of how OSM, OpenStack and OpenDaylight can be used to virtualize network functions. That is, first, a VNF is created using OSM. The creation of this VNF is simple via predefined templates or by loading an existing VNF package. The VNF is then deployed on OpenDaylight, which provides the infrastructure and resources needed to run the VNF. OpenStack is used to manage the network infrastructure and control the traffic flow within the virtualized network, i.e. create, manage and delete VNFs. And finally, OSM monitors and manages the virtualized network as a whole created by VNFs. Thus, the deployed testbed provides tools to provision, monitor and scale virtualized network functions (VNFs) with network resources as needed.

1. CAPITULO I.

ANTECEDENTES

En el presente capitulo se describe la línea base que permite abordar la escasez de banco de pruebas (testbed) que explote los conceptos NFV, y posibilita el planteamiento y desarrollo del presente trabajo de titulación, de acuerdo al siguiente detalle:

1.2 Planteamiento del problema

Las redes de comunicación están en constante evolución y se han encontrado ante la obligación de utilizar arquitecturas más flexibles y escalables (Barrado, 2018), debido a que actualmente son muy complejas e involucran una diversidad de hardware estático como: routers, switches y middleboxes, por lo tanto, el tráfico de datos que genera este tipo de redes es controlado y administrado por reglas y políticas de control (Bernal & Mejia, 2016). Es así como, la inflexibilidad que presentan las redes hoy en día hace que las futuras demandas de servicios sean difíciles de desplegar, ya que los avances en las TI, la computación en la nube y el crecimiento exponencial de dispositivos inteligentes conectados con el Internet de las cosas, hace que el tráfico generado en redes tradicionales incremente significativamente, obteniendo un bajo rendimiento de estas (Bernal & Mejia, 2016).

Por este motivo es que las redes necesitan un cambio en cómo se diseñan y operan para llegar a ser más flexibles y fáciles de actualizar, buscando desarrollar nuevas infraestructuras para poder adecuarse a los nuevos estándares, ya que, las necesidades tecnológicas de hoy exceden el límite que ofrecen las redes tradicionales, principalmente en volumen de capacidad, velocidad y flexibilidad (Barrado, 2018).

Frente a esto; surgen diferentes tecnologías de virtualización que proponen la simulación de las capacidades de muchos dispositivos de red tradicionales, donde en un entorno virtualizado no es necesario el despliegue de nuevos dispositivos y se logra desarrollar varias soluciones de red dentro de una misma plataforma virtualizada (Ramos, 2017). Por tal motivo, la virtualización normalmente se centra en la creación

de recursos lógicos que se desarrollan en un hardware estándar, es decir funciones simples de equipos virtualizadas, incorporadas y trasladadas a diversas ubicaciones de red, de acuerdo con las necesidades (Bernal & Mejia, 2016). Por otra parte, el objetivo de NFV (Network Funtion Virtualization) es utilizar la virtualización de los componentes físicos de un CPU y otras tecnologías de computación en la nube para migrar las funciones de red, de hardware dedicado a máquinas virtuales desplegadas en equipos hardware de propósito general (Bernal & Mejia, 2016).

La virtualización de funciones de red (NFV: Network Funtion Virtualization) que ofrece nuevos lineamientos para desplegar, diseñar y gestionar los servicios de red, cuyo enfoque principal se basa en la utilización de las funciones de red del hardware propietario para virtualizarlas convirtiéndolas en servicios de software útiles en diferentes plataformas, que puedan ejecutarse en hardware estándar, permitiendo varios puntos de gestión de red (Ramos, 2017). Ciertamente el concepto NFV se origina al intentar desplegar los servicios de red de manera centralizada utilizando una virtualización estándar dejando atrás el modelo tradicional de redes estáticas, proporcionando un enfoque general para el modelado y despliegue de los servicios en la nube (Matta, Nabeel, & Yuefueg, 2016), es así que para conseguir una solución NFV se implementa su arquitectura, en la que existen tres componentes fundamentales los cuales son: Virtualized Infraestructure Manager (VIM), Virtualized Network Funtions (VNF Manager) y NFV Orquestador. Dado que VIM es el encargado de gestionar los recursos físicos de almacenamiento y computo en una infraestructura virtualizada (Barrado, 2018). EL VNF Manager es el responsable de la configuración, contabilidad, performance y la seguridad de las funciones de red virtualizadas. Y el NFV Orquestador es el en cargado de orquestar los servicios formados por distintas VNFs, además se encarga de orquestar los recursos entre las diferentes infraestructuras a través del VIM que usa código abierto para su despliegue (Zurita, 2019).

Actualmente, las redes definidas por software (SDN) y la virtualización de funciones de red (NFV) se están abriendo camino en la agenda de investigación siendo los principales actores en el dominio de las redes (Riggio, Rasheed, & Granelli, 2014). De la misma manera, los bancos de pruebas y las instalaciones experimentales son

ampliamente considerados como la piedra angular fundamental para el futuro establecimiento de redes virtuales (Chou, Tseng, Tseng, & Chen, 2016). Sin embargo, diseñar y construir instalaciones experimentales difícilmente puede considerarse algo insignificante para investigadores y profesionales. La escala, la flexibilidad y la facilidad de uso son solo algunos de los desafíos que enfrenta un diseñador de banco de pruebas. Estas consideraciones están recomendadas en las plataformas como GENI en EE. UU., AKARI en Japón, FEDERICA, NOVI y OFELIA en Europa, que proporcionan instalaciones federadas y abiertas para la agenda de investigación futura de Internet (Riggio et al., 2014). *Aunque la importancia de tales instalaciones es incuestionable, hoy en día todavía hay una escasez de banco de pruebas (testbed) que explote los conceptos NFV en el dominio de redes virtuales en el Ecuador y específicamente en la Universidad Técnica del Norte.*

Existen proyectos que se han implementado un banco de pruebas experimental, tal es el caso de (Ramos, 2017) que asegura en su estudio que la plataforma más completa para el despliegue de la virtualización de funciones de red hoy en día es OpenStack, un entorno Cloud Open Source que cumple con todas las ventajas mencionadas anteriormente y con la tecnología SFC (Service Function Chaining) que consiste en realizar el encaminamiento de los paquetes a través de las distintas funciones de red virtualizadas y así logra desarrollar diferentes pruebas del comportamiento de NFV. Por otra parte, la plataforma OSM (Open Source Mano), propuesta por la ETSI, está orientada propiamente para el despliegue de las funciones de red virtualizadas ya que se encarga de la orquestación entre el VIM (Virtualized Infraestructure Manager) y VNFs (Virtualized Network Funtions) sin la necesidad de otras tecnologías como Open Stack. En este sentido, existen estudios sobre el Orquestador OSM, como por ejemplo (Barrado, 2018) que ha implementado la plataforma de orquestación Open Source MANO (OSM) para estudiarla de una manera practica y teórica, enfocándose únicamente a evaluar la plataforma, para determinar el alcance de su funcionalidad.

1.3 Objetivos

1.3.1. Objetivo general

Diseñar y desplegar un testbed donde permita estudiar el comportamiento de NFV, generando una topología que evidencie el desempeño de las funciones de red virtualizadas.

1.3.2 Objetivos específicos

- Establecer el estado del arte de la tecnología NFV y la plataforma de orquestación (OSM) para desplegar, diseñar y gestionar los servicios de red virtualizados.

- Diseñar una topología de red en un ambiente controlado por plataformas OSM, OpenStack y OpenDaylight para el despliegue de los servicios virtualizados.

- Virtualizar al menos una función de red para demostrar el funcionamiento de la tecnología NFV, mediante la implementación de un testbed que permita demostrar la orquestación de las funciones de red virtualizadas.

- Realizar las pruebas suficientes para demostrar el correcto funcionamiento del testbed implementado.

1.4 Alcance

En este proyecto se desarrollará un banco de pruebas experimental (testbed) que tiene como objetivo llenar el vacío ofrecido por la falta de testbed propios para NFV. El trabajo de investigación propuesto consistirá en una plataforma abierta sobre la cual se pueda probar la arquitectura y el despliegue a escala de la tecnología NFV, compuesto por una topología de red en un ambiente controlado por plataformas Open Source MANO, OpenStack y OpenDaylight como infraestructuras VIM donde se verificará su conectividad para virtualizar las funciones de red y tener una centralización de recursos

que será utilizado por la comunidad universitaria. Es decir, será implementado en el Centro de Datos de la Facultad de Ingeniería en Ciencias Aplicadas, el cual aloja servidores estándar y una red basada en cloud computing privado para la hiperconvergencia, es aquí donde se desplegará el testbed para la Virtualización de Funciones de Red NFV.

Se iniciará con el estado del Arte en donde se identificará la documentación existente referente a la virtualización de funciones de red (NFV) siguiendo el modelo de referencia definido por la ETSI, así como también de las diferentes organizaciones que estandarizan la computación en la nube y la orquestación de funciones de red, analizando a detalle la arquitectura, capacidad y puesta en marcha de la tecnología NFV, como también su impacto en el futuro de las comunicaciones, con el fin de elaborar documentación acerca del tema que sirvan de base a investigaciones futuras.

Además, para lograr la implementación de la virtualización de funciones de red, se empezará con el diseño de una topología que soporte varias plataformas que logren comunicarse con OSM tales como OpenStack y OpenDaylight con el objetivo de tener un ambiente de red controlado, es así como se va a integrar el VIM con OSM, por lo que es necesario asegurar que exista conectividad entre estas plataformas y posteriormente desplegar máquinas virtuales (instancias) para los servicios de red, a las que se les asignará interfaces para el acceso de invitados, de esta forma es como se comunicarán las diferentes plataformas que permita realizar un escenario real en la nube, proponiendo una infraestructura a escala para orquestar por lo menos una función de red virtualizada, la cual será un NS (Network Service) sobre una topología simple que se va a definir a lo largo de la investigación.

Por otra parte, al momento de implementar los servicios de red virtualizados se seleccionará el despliegue de VNFs simples, que estarán conectados por medio de un descriptor de enlace virtual propio de OSM, de esta manera es como se tomara en cuenta las características de los equipos, para el despliegue de la virtualización de las funciones de red que provee albergar varios servidores virtualizados logrando obtener una orquestación de recursos de red más centralizada.

El trabajo de investigación finalizará con las pruebas de funcionamiento del testbed implementado que permite verificar la correcta instalación e integración de todos los componentes de la arquitectura NFV. Destacando las virtudes que proporciona la tecnología, como la reducción significativa de recursos hardware por parte de la plataforma OSM y el fácil despliegue de la arquitectura, ya que OSM puede ser utilizado por más usuarios y administradores de red debido a su accesibilidad.

1.5 Justificación

En la actualidad la necesidad de utilizar arquitecturas de red más flexibles y escalables se ha vuelto indispensable debido a las nuevas exigencias que presenta el mercado (Martínez, 2019), por este motivo los distintos proveedores de servicios buscan desarrollar nuevas infraestructuras que implementen la virtualización y los servicios de la nube, ya que, las necesidades a corto plazo exceden el límite que ofrecen las redes tradicionales, principalmente en volumen de capacidad, velocidad y flexibilidad (Barrado, 2018). En otras palabras, la virtualización y los servicios en la nube representan un enorme salto de competitividad en el mercado puesto que generan un ahorro significativo para los operadores de red y ha iniciado la transición digital que está sucediendo (Ramos, 2017).

La virtualización de funciones de red (NFV) que ofrece nuevos lineamientos para desplegar, diseñar y gestionar los servicios de red (Pattaranantakul, He, Zhang, & Meddahi, 2018), su enfoque principal se basa en la utilización de las funciones de red del hardware propietario para virtualizarlas convirtiéndolas en servicios de software útiles en diferentes plataformas, que puedan ejecutarse en equipos hardware estándar, permitiendo varios puntos de gestión de red (Zurita, 2019).

Es así, que el diseño permite visualizar la arquitectura de red en su totalidad incluyendo capacidad de almacenamiento, velocidad y computación de red (Abdelsalam et al., 2020). Sin embargo, para crear las funciones de red virtualizadas en hardware, NFV

utiliza tecnología de virtualización estándar, volviéndolas aplicables tanto en el plano de control como en el de datos y en arquitecturas de red tradicionales (Zurita, 2019).

Open Source Mano (OSM) es una plataforma diseñada para la investigación e innovación en NFV (Lv & Xiu, 2019). El objetivo de esta plataforma es fomentar la indagación del sistema en torno a la orquestación de recursos de computación en la nube, exponer a la comunidad de la investigación a los requisitos a nivel empresarial e industrial, así como también proporcionar trazas realistas de las cargas de trabajo en la nube y los que es más importante (Vilalta, Mayoral, Mu, Casellas, & Mart, 2015), proporcionar APIs y fuentes de código abierto para la investigación de la tecnología NFV (Mauro, Galatro, Longo, & Postiglione, 2018), dando un enfoque general para el modelado y despliegue de los servicios en la nube, es así que para conseguir una solución NFV se va a implementar primero su arquitectura, tanto en el plano de control como de datos que debe seguir los estándares de NFV MANO, propuesta por la ETSI, que es una organización que estandariza las telecomunicaciones en Europa, por lo tanto, la plataforma OSM tiene que regirse a los modelos propuestos para la implementación de NFV definidos por la ETSI (Lv & Xiu, 2019).

Un testbed de virtualización de funciones de red consta de un conjunto de instalaciones, plataformas abiertas y experimentales desarrolladas a un nivel estandarizado e integradas en una infraestructura experimental confiable y diversificada (Mauro et al., 2018), que cubre aspectos importantes del estado del arte en redes y aplicaciones como la programabilidad, seguridad e interfuncionamiento (Cordero, 2017). De esta manera el despliegue (Martínez, 2019) de una infraestructura integrada en un banco de pruebas tiene como objetivo habilitar, nutrir y respaldar los diferentes conceptos utilizados al momento de desarrollar servicios, aplicaciones o crear un middleware de control de última generación basado en tecnología NFV (Arocha, 2017).

2. CAPITULO II
FUNDAMENTACIÓN TEÓRICA

Este capítulo describe la fundamentación teórica más relevante para realizar un análisis Testbed para el estudio de la virtualización de las funciones de red NFV, es así como, se define la tecnología NFV con los componentes que integran su arquitectura, también se detalla el concepto de la plataforma OSM que define su funcionamiento y las partes de su arquitectura y especialmente se analiza el estudio de plataformas OpenStack y OpenDaylight que se implementan en el bloque VIM.

2.1 Redes Definidas por Software (SDN)

SDN ha alcanzado un punto de inflexión en el que está reemplazando el modelo de red tradicional, que por lo general, actualmente son muy complejas e involucran una diversidad de hardware estático como: routers, switches y middleboxes, por lo tanto, el tráfico de datos que genera este tipo de redes es controlado y administrado por reglas y políticas de control (Bernal & Mejia, 2016) En consecuencia, las redes definidas por software brindan un nivel mejorado de flexibilidad y caracterización a las funciones de red, para satisfacer las necesidades de las nuevas tendencias de redes y TI, como plataformas cloud, movilidad, las redes sociales y video (Stallings, 2015).

Ciertamente, existen dos elementos involucrados en el reenvío de paquetes a través de enrutadores que son: una función de control, que decide la ruta que toma el tráfico y la prioridad relativa del tráfico, y una función de datos, que reenvía datos según la política de función de control (Padma y Yogesh, 2015). Antes de que se desarrolle SDN, estas funciones se realizaban de forma integrada en cada dispositivo de red (routers, switches, servidores, etc.). Sobre todo, el control en una red tradicional se despliega mediante un protocolo de red de control y enrutamiento que se implementa en cada nodo de la red, por lo que, este enfoque es relativamente inflexible y requiere que todos los nodos de la red implementen los mismos protocolos (Padma y Yogesh, 2015; Stallings, 2015). Por otro lado, con SDN, un controlador central realiza todas las funciones complejas, incluido el enrutamiento, la denominación, la declaración de políticas y las comprobaciones de seguridad como se muestra en la figura 1.

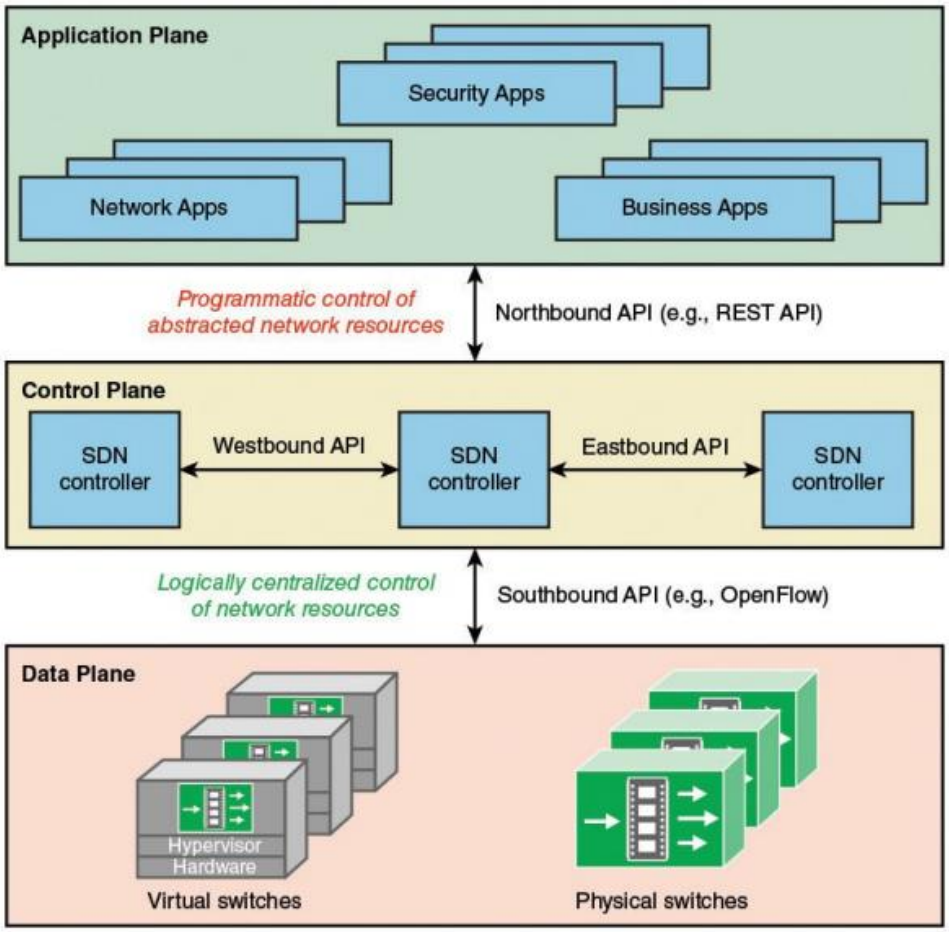

Figura 1. Arquitectura SDN

Obtenido de: (Stalling, 2015)

Si podemos observar el enfoque SDN divide la función de conmutación entre un plano de datos, un plano de control y un plano de aplicación que se encuentran en dispositivos separados como se muestra en la figura 1 (Stallings, 2015). El plano de datos es simplemente responsable de reenviar paquetes, mientras que el plano de control proporciona la "inteligencia" para diseñar rutas, establecer parámetros de política de enrutamiento y prioridad para cumplir con los requisitos de QoS[1] y QoE[2] y hacer frente a los patrones cambiantes del tráfico, por lo tanto, el plano de aplicación, donde se incluye varias aplicaciones de red, es el que se encarga específicamente de la

[1] QoS (Quality of Service) o calidad de servicio que se define como las propiedades de rendimiento medibles de extremo a extremo de una red.

[2] QoE (Quality of Expeience) se refiere a una medida subjetiva del rendimiento de la red según lo experimentado por el usuario.

administración y el control de la red (Zuo et al., 2014). Así mismo, las interfaces abiertas programables (API) se definen para que el hardware de conmutación presente una interfaz uniforme independientemente de los detalles de la implementación interna. Es así como, se definen interfaces abiertas programables (API) para permitir que las aplicaciones de red se comuniquen con los controladores SDN (Zuo et al., 2014; Stallings, 2015; Kaljic et al., 2019).

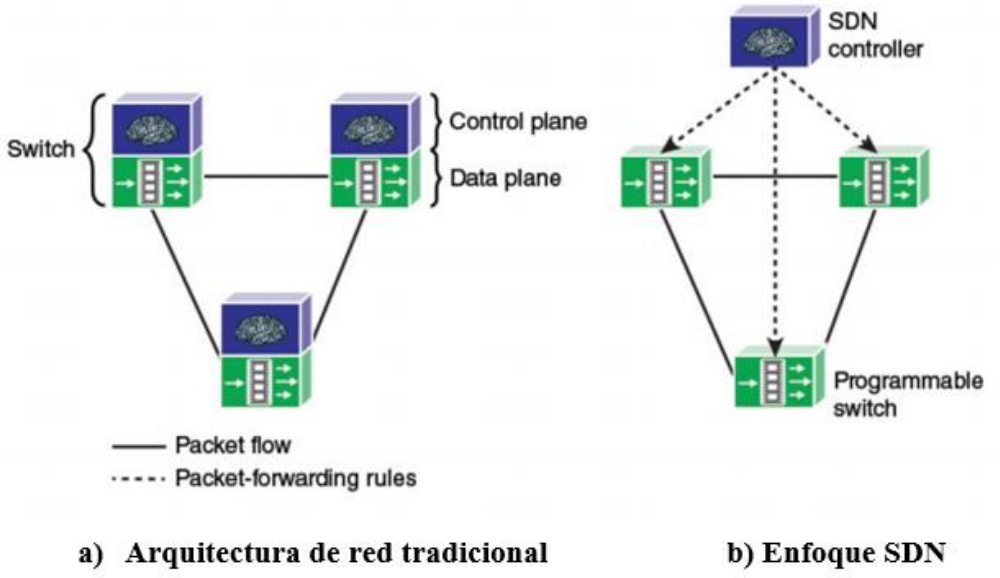

Figura 2. Planos de Control y Datos
Obtenido de: (Stalling, 2015)

2.1.1 SDN Plano de Datos

El plano de datos SDN, es denominado capa de recursos en UIT-T Y.3300 y también conocido como capa de infraestructura, es en este plano donde los dispositivos de enrutamiento de paquetes realizan el transporte y reenvío de datos de acuerdo con las decisiones tomadas por el plano de control SDN (Stallings, 2015). La característica importante de los dispositivos de red en una red SDN es que estos dispositivos realizan una función de reenvío simple, sin software integrado para tomar decisiones autónomas (Stallings, 2015; Kaljic et al., 2019).

En este sentido, las funciones realizadas por los dispositivos de red en el plano de datos SDN son:

- **Función soporte de control:**

Se comunica con la capa de control SDN para respaldar la programabilidad a través de interfaces de control de recursos. Es decir, un dispositivo que se comunica con el controlador SDN y este controlador se encarga de gestionarlo a través de un protocolo OpenvSwitch (Kaljic et al., 2019).

- **Función de reenvío de datos:**

En esta función se acepta los flujos de datos entrantes de otros dispositivos de red y equipos finales para reenviarlos a lo largo de las rutas que se han calculado y establecido de acuerdo con las reglas definidas por las aplicaciones SDN (Kaljic et al., 2019).

2.1.2 SDN Plano de Control

En la capa de control SDN se mapea las solicitudes de servicio del plano de aplicación en comandos específicos para dispositivos que se encuentran en el plano de datos, además, proporciona a las aplicaciones información sobre la topología y la actividad del plano de datos (Zuo et al., 2014). Por lo tanto, la capa de control se implementa como un servidor o conjunto cooperativo de servidores conocidos como controladores SDN.

Un controlador SDN funciona de la misma manera que un sistema operativo de red (NOS) que proporciona servicios esenciales para los desarrolladores de red con la ayuda de interfaces abiertas de programación (API) (Tatang et al. 2017). Las funciones que cumple un SDN NOS, es definir políticas de red y gestionar redes sin preocuparse de las características del dispositivo de red, las cuales pueden ser heterogéneas y dinámicas (Tatang et al. 2017; Stallings, 2015). Con la ayuda de una API en dirección norte se puede crear software independiente no solo de los detalles del plano de datos, sino que proporciona un medio uniforme para que los desarrolladores de aplicaciones y los administradores de red accedan al servicio SDN y realicen tareas de gestión de red (Bari et al., 2016).

Varias iniciativas diferentes, tanto comerciales como de código abierto, han dado lugar a implementaciones de controladores SDN. A continuación, se describen las características generales de varios controladores destacados:

OpenDaylight: Según Stallings (2015) es una plataforma de código abierto de programación de red para habilitar SDN, escrito en Java. OpenDaylight fue fundada por Cisco e IBM, y su membresía está fuertemente inclinada hacia los proveedores de redes. OpenDaylight se puede implementar como un solo controlador centralizado, y permite que los controladores se distribuyan en varias instancias que pueden ejecutarse en uno o más servidores agrupados en la red (Husni y Bramantyo, 2018).

Open Network Operating System (ONOS): es una plataforma SDN NOS de código abierto que fue desarrollado por varios operadores de red como AT&T y NTT, y otros proveedores de servicios. ONOS fue creado para usarse como un controlador distribuido y proporciona configuraciones para particionar y distribuir el estado de la red en múltiples controladores distribuidos (Eljack et al., 2019).

POX: es un controlador OpenFlow de código abierto que ha sido diseñado e implementado por varios desarrolladores de SDN, tales como VMWare, Google y NTT. POX proporciona una interfaz gráfica de usuario (GUI) basada en la web, tiene una API y documentación bien redactada y está escrito en Python, que normalmente acorta sus ciclos experimentales y de desarrollo en comparación con otros lenguajes de implementación, como C ++ (Stallings, 2015).

Beacon: es una plataforma de código abierto desarrollado en Stanford. Diseñado en Java y puesto en el entorno de desarrollo integrado (IDE) de Eclipse. Beacon fue el primer controlador que hizo posible crear un entorno SDN funcional (Stallings, 2015).

2.1.3 SDN Plano de Aplicación

Podemos decir que el plano de control SDN proporciona las funciones y servicios que facilitan el desarrollo y el despliegue rápido de aplicaciones de red, por lo tanto, el potencial de las redes definidas por software (SDN) para las redes tradicionales está en

el soporte que ofrece a las aplicaciones de red para monitorear y gestionar el comportamiento de la red (Sandoval, 2018).

El plano de aplicación incluye varias aplicaciones de red, que se encargan específicamente de la administración y el control de la red, cabe recalcar que no existe un conjunto exacto de tales aplicaciones o incluso una lista de estas aplicaciones, es así como, la capa de aplicación puede incluir servicios y herramientas de red generales que son desplegadas y pueden verse parte de la funcionalidad del plano de control (Stallings, 2015).

Una aplicación o servicio situado en el plano de aplicación se encarga de monitorear, definir y controlar los recursos y el comportamiento de la red, por lo tanto, debe interactuar con el plano de control SDN a través de interfaces de control de aplicaciones, (Chiosi et al., 2013) y así, personalizar automáticamente el comportamiento y las características de los recursos de red. Con la programación y el despliegue de una aplicación SDN logramos ver los recursos de red proporcionados por la capa de control SDN a través de la información y arquitectura de datos expuestos por la interfaz de control de aplicación (Herrera y Vega, 2016).

En la figura 3 se muestra una descripción general del plano de aplicación SDN, se observa los elementos y las áreas de aplicación especificas mediante un enfoque de abajo hacia arriba. Como se puede ver en la imagen, una interfaz de dirección norte (Northound Interface) puede ser local o remota la cual se utiliza para ejecutar las aplicaciones SDN en un mismo servidor del plano de control, convirtiéndose así la interfaz en dirección norte como un protocolo o interfaz de programación de aplicaciones (API) la cual conecta las distintas aplicaciones al sistema operativo de red del controlador (NOS) (Stallings, 2015).

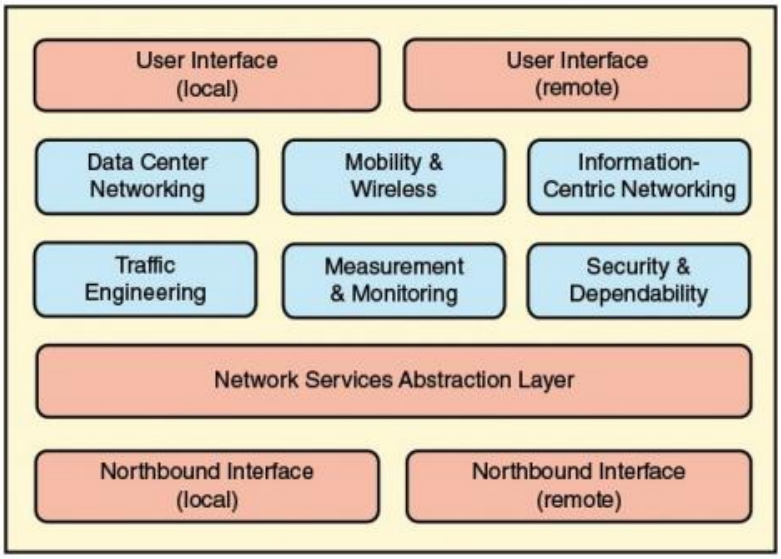

Figura 3. SDN plano de Aplicación.
Obtenido de: (Stallings, 2015).

2.2 Virtualización de Funciones de Red (NFV)

Según Chiosi et al. (2013) en la publicación de la ETSI "Network Functions Virtualization: Network Operator Perspectives on Industry Progress" la Virtualización de Funciones de Red (NFV por sus siglas en ingles Network Function Virtualization) tiene como objetivo transformar la forma en que los operadores de red diseñan redes, mediante funciones de red basadas en software que se ejecutan como máquinas virtuales para consolidar muchos tipos de equipos de red en servidores de alto rendimiento de la industria, que pueden ubicarse en centros de datos, nodos de red y en el usuario final. De manera que, NFV está comprendido como un modelo de arquitectura de red que permite la virtualización de servicios de esta, definiendo un modelo virtual que se encarga de manejar todas las funciones desplegadas por los nodos de la red, llamado Virtualized Network Function (VNFs) (Rehman, Aguiar y Barraca, 2019).

Si comparamos una solución NFV con las redes tradicionales estas, requieren de varios procesos para su despliegue, ya sea de mantenimiento, actualización de software o instalación manual, disminuyendo su eficiencia debido al mayor consumo de tiempo y

otros recursos que estas necesitan para su ejecución (Sandoval, 2018) Por otro lado, con NFV crear una máquina virtual y programar algunas líneas de código para levantar una VNF es más simple y rápido, ya que, mediante una VNF se puede proporcionar cualquier servicio en el momento que se necesite y en el lugar donde se requiera, obteniendo un ahorro de recursos (Intel, Brocade, Cyan, Red Hat y Telefónica, 2017). Es así como obtenemos una solución NFV que agrupa varias VNFs brindando un segmento de red virtualizado (Millán y Esfandiari, 2014; Sandoval, 2018).

En otras palabras, se puede decir que la virtualización de funciones de red (NFV) es la virtualización mediante la implementación de funciones de red en software y desplegadas en máquinas virtuales, por lo tanto, NFV a cambiado significativamente los enfoques tradicionales para el diseño, implementación y administración de servicios de red, puesto que, esta tecnología desacopla las funciones de red, como por ejemplo, el servicio de nombres de dominio (DNS), la traducción de direcciones de red (NAT), de dispositivos hardware estándar para que puedan desplegarse en software implementado en máquinas virtuales (Stallings, 2015; Rehman, Aguiar y Barraca, 2019).

Es por esta razón que, la virtualización de funciones de red (NFV) se basa en tecnologías de máquina virtual (VM) estándar que permite la migración de aplicaciones dedicadas y dispositivos basados en red a una plataforma unificada virtual que comprende servidores estándar. Es así como el software y el hardware se desacoplan y la capacidad de cada aplicación aumenta o disminuye al agregar o reducir recursos virtuales (Chiosi et al., 2013; Barrado, 2018).

Ciertamente, para unir múltiples sistemas operativos (SO) es necesario crear una aplicación que pueda administrar y admitir varias infraestructuras de hardware y sistemas operativos, siendo este, un proceso costoso y que demanda de varios recursos; sin embargo, la virtualización de hardware es un proceso eficaz para solucionar este problema. Es así como, la tecnología de virtualización permite que una sola computadora o servidor pueda ejecutar al mismo tiempo varios sistemas operativos o desplegar múltiples sesiones de un solo SO (Stallings, 2015). Por consiguiente, la virtualización desplegada por una máquina puede alojar numerosas aplicaciones, en

una misma plataforma hardware, que puede permitir múltiples máquinas virtuales (VM), y cada una posee diferentes características dependiendo de un sistema operativo en particular, con diferentes versiones de virtualización y que hace uso de las diferentes características físicas, de dicha plataforma hardware (Barrado, 2018).

Para lograr la solución expuesta en el anterior apartado, es necesario de un monitor de máquina virtual (VMM) o hipervisor, este software permite que varias máquinas virtuales coexistan de forma segura en un único host como plataforma hardware y compartan los recursos de este host (Herrera y Vega, 2016). Además, según Stallings (2015) en su libro Foundations of Modern Networking, menciona que un host que admite al menos seis VM posee una relación de consolidación de 6 a 1, por esta razón, los hipervisores en el ámbito comercial podrían proporcionar relaciones de consolidación de entre 4:1 y 12:1, si tenemos en cuenta, que una empresa virtualiza todos sus servidores, podría eliminar el 75 por ciento de los servidores de sus centros de datos, sobre todo, también podría eliminar el costo en comprar hardware innecesario. Es así como, con menos servidores físicos, se necesita menos energía y recursos físicos, además, esto lleva a reducir cables, routers, switchs y espacio en el centro de datos. Es por esto que, con la ayuda de un hipervisor que se encuentra entre el hardware y las máquinas virtuales actuando como un intermediario de recursos, la consolidación de servidores se convirtió y sigue siendo, la mejor forma para resolver el problema de tener una red estática y costosa (Stallings, 2015; Rehman, Aguiar y Barraca, 2019).

Cabe recalcar que una característica clave de la virtualización es, la capacidad de ejecutar varias máquinas virtuales en una máquina, por lo que, las máquinas virtuales pueden verse como recursos de red. Por otra parte, la virtualización hace posible enmascarar los recursos del servidor, como, procesadores y sistemas operativos individuales (Chiosi et al., 2013; Herrera y Vega, 2016). En resumen, esto hace posible dividir un solo host en múltiples servidores independientes, conservando los recursos de hardware, además se puede migrar rápidamente un servidor de una máquina a otra para el equilibrio de carga o para la conmutación dinámica en caso de falla de la máquina (Stallings, 2015). Finalmente, la virtualización se ha convertido en una

tecnología esencial en el manejo de aplicaciones de Big Data[3] y en la implementación de infraestructuras de computación en nube (Stallings, 2015).

2.2.1 Arquitectura NFV

La arquitectura NFV se compone de tres bloques fundamentales (1) Network Functions Virtualization Infraestructure (NFVI), (2) Virtualized Network Functions (VNFs) y (3) Management and Orchestration (MANO) como se muestra en la figura 4.

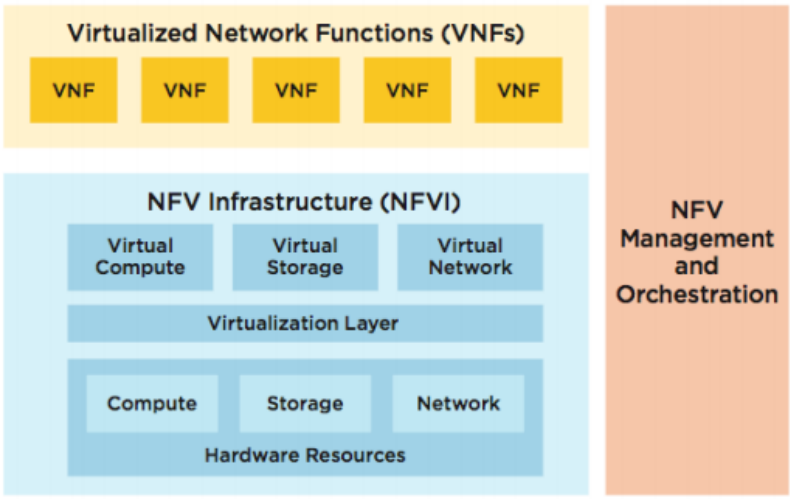

Figura 4. Arquitectura de NFV
Obtenido de: (Sandoval, 2018).

Cada bloque o parte de la arquitectura NFV cumple una función específica de dominio general, es así como, en la figura 5 se muestra una red virtualizada en la que los detalles físicos y lógicos de nivel inferior son transparentes por medio de APIs de dirección norte, además, para obtener una implementación de NFV, en el bloque NFVI se despliegan los dominios de cómputo (Compute domain), de hipervisor (Hypervisor domain) y el dominio de infraestructura de red (Infraestructure network domain) los cuales describen los componentes de hardware y software sobre los que se construyen las redes virtuales(Barrado, 2018), por otro lado, en el bloque VNF opera el dominio de aplicaciones NFV (NFV applications domain), en el cual se despliega una máquina virtual (VM) donde se alojan las diferentes funciones de red virtualizadas (Herrera y

[3] Big Data: este término se refiere a una gran cantidad de datos que es difícil de procesarlos debido a su compleja estructura.

Vega, 2016), pero, cabe recalcar que el software de aplicación se despliega en una misma máquina virtual que el software de virtualización (Chiosi et al., 2013; Stallings, 2015; Barrado, 2018). Y por último se detalla el dominio del bloque MANO que interactúa tanto con el NFVI como con las VNFs para controlar su despliegue (Stallings, 2015; Barrado, 2018).

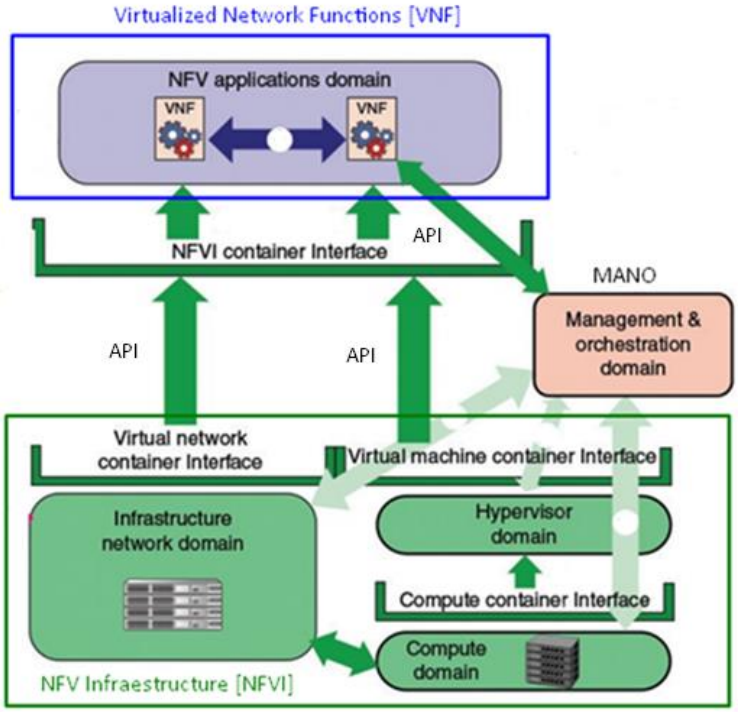

Figura 5. Arquitectura de domino general NFV
Obtenido de: (Stallings, 2015) y modificado por autor

También las interfaces que conectan cada domino en la arquitectura NFV, son interfaces de contenedor programables, es decir un entorno de ejecución en un sistema host físico dentro del cual se despliega un bloque funcional de software (Stallings, 2015; Barrado, 2018). En otras palabras, son APIs de dirección norte, que se encargan de conectar los bloques funcionales a través de software programable (Barrado, 2018).

2.2.1.1 Network Functions Virtualization Infraestructure (NFVI)

De acuerdo con (Chiosi et al., 2013; Barrado, 2018) NFVI proporciona los recursos de hardware y software necesarios para la ejecución de las funciones de red virtualizadas. Es decir, incluye hardware tecnológico disponible para alojar las máquinas virtuales y

una capa de software que virtualiza y abstrae los recursos físicos (computación, almacenamiento y conectividad) (Herrera y Vega, 2016).

Según Stallings (2015) el bloque NFVI es el corazón de la arquitectura NFV, el cual para su despliegue abarca tres dominios principales: Dominio de cómputo, Dominio de hipervisor y dominio de infraestructura de red.

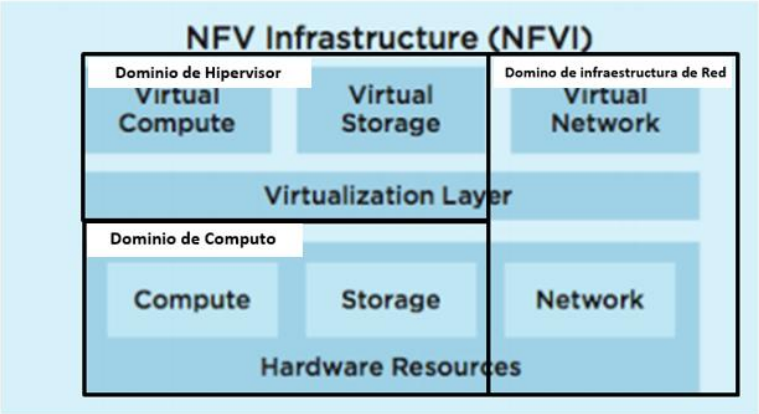

Figura 6. Dominios del Bloque NFVI
Obtenido de: (Stallings, 2015) y modificado por autor

En este sentido, el Dominio de cómputo, se encarga de proporcionar almacenamiento y servidores estándar de alta capacidad; El Dominio de hipervisor, el cual comparte los recursos del domino de cómputo a las máquinas virtuales proporcionando una abstracción de hardware; y por último, el Dominio de infraestructura de red (IND: Infrastructure Network Domain), que comprende todas las interfaces de red interconectadas para suministrar los servicios de una infraestructura de red (Stallings, 2015; Fernández, 2018).

Dominio de Computo

Según Chiosi et al. (2013) en la publicación de la ETSI "Network Functions Virtualization: Network Operator Perspectives on Industry Progress" define un término NFVI-Node como una agrupación de dispositivos físicos implementados y administrados como una sola entidad para proporcionar funciones NFVI necesarias para respaldar un entorno de ejecución para VNF, a continuación, se detalla los tipos de nodos en el dominio de cómputo:

a) Nodo de cómputo, se define como un componente funcional que es capaz de desarrollar un conjunto de instrucciones computacionales para controlar el tiempo de acceso necesario a la memoria de cada dispositivo físico (Chiosi et al., 2013).

b) Nodo de puerta de enlace, es un elemento fácil de configurar y administrar dentro del dominio de cómputo, este nodo es muy importante, debido a que implementa funciones de puerta de enlace que proporcionan la interconexión entre diferentes NFVI y redes de transporte dentro de una arquitectura NFV, además, procesa paquetes de diferentes redes agregando y eliminando encabezados a nivel de capa de red, por otro lado, es capaz de conectar redes virtuales con dispositivos de redes existentes (Chiosi et al., 2013; Barrado, 2018).

c) Nodo de almacenamiento, es el encargado de proporcionar recursos de almacenamiento mediante funciones de procesamiento de datos, almacenamiento y redes, este nodo puede desplegarse físicamente, como, por ejemplo, puede ser un dispositivo físico accesible a la red a través de tecnologías de almacenamiento, tales como, Network File System[4] (NFS) o Fibre Channel[5] (Chiosi et al., 2013; Stallings, 2015; Barrado, 2018).

d) Nodo de red, este nodo proporciona recursos de red, es decir, conmutación, enrutamiento y reenvío de paquetes, dentro del bloque NFVI, estos recursos pueden ser configurables y administrables mediante funciones de procesamiento y protocolos de red (Barrado, 2018).

[4] NFS es un protocolo de capa de aplicación, que ayuda a varios dispositivos conectados en una misma red puedan acceder a ficheros de almacenamiento remotos como si se trataran de locales.

[5] Fibre Channel definen un mecanismo de transferencia de datos de alta velocidad que se utiliza para conectar diferentes dispositivos de almacenamiento.

Dominio de Hipervisor

En este dominio se representa un entorno de software separado del hardware, en el cual se despliegan diferentes servicios, como despliegue de máquinas virtuales (VMs), uso de protocolos de red, migración en vivo y alta disponibilidad (Stallings, 2015), con el fin de gestionar recursos informáticos, de almacenamiento y de red para brindar un acceso virtualizado mediante un conmutador virtual, de tal forma que, este conmutador virtual opera como un conmutador ethernet desplegado por el hipervisor que interconecta las NICs virtuales de las VMs con la NIC[6] física del nodo de cómputo, por lo tanto, si dos VNFs se encuentran implementadas en el mismo servidor físico se conectan a través del mismo conmutador virtual, pero cabe recalcar que, si las VNFs están implementadas en servidores físicos diferentes, su conexión pasa a través del primer conmutador virtual a la NIC y luego a un segundo conmutador virtual externo (Chiosi et al., 2013; Stallings, 2015).

Dominio de Infraestructura de Red.

Ciertamente, para realizar una solución NFV, todo el tráfico de red VNF debe pasar por un conmutador virtual controlado por el hipervisor, el cual despliega una capa de software que converge entre el hardware de la red y el software VNF, generando una pérdida significativa en el rendimiento, es por este motivo que aparece una tecnología llamada Eswitch u OpenvSwitch el cual omite la creación de la capa de software y proporciona a las VNFs una ruta de acceso directo a la memoria de la NIC (Carbonel, 2018). La funcionalidad del Eswitch consiste en manejar: la carga de trabajo que ejecuta en el plano de control, mediante protocolos de control y señalización BGP; y, cargas de trabajo en el plano de datos, mediante conmutación, enrutamiento y retransmisión de paquetes en el tráfico de red (Kaljic et al., 2019). Es así como, el despliegue de un Eswitch acelera el procesamiento de paquetes sin ninguna sobrecarga en el procesador (Stallings, 2015).

2.2.1.2 Virtualized Network Functions (VNFs)

Para lograr un entorno de red virtualizado, una VNF debe alojar una función de red virtualizada denominada componente VNF (VNFC), este componente VNF se define

[6] NIC dispositivo hardware que se encarga de proporcionar conexiones de red

como el proceso de virtualizar una función de red (Cox, J.H et al, 2017; Schneider, S., et al., 2018). Entonces, una VNF consta de una o varias VNFCs conectadas lógicamente, que se despliegan como software en un dominio de hipervisor y a su vez se desarrollan como hardware en el dominio de cómputo, por lo tanto, los enlaces de red virtual que los unen se despliegan a través del dominio de infraestructura de red (IND), cabe recalcar que el IND interactúa directamente con el dominio de cómputo (Stallings, 2015; Cox, J.H et al, 2017; Schneider, S., et al., 2018).

Por lo tanto, se define este bloque como la implementación de software de una función de red que es capaz de ejecutarse sobre el NFVI, de modo que, las VNFCs son las funciones de red virtualizadas que usan máquinas virtuales para su despliegue, por otro lado, la conectividad interna entre VNFC dentro de un mismo VNF o a su vez entre varios VNFs debe ser especificada por APIs que actúan como switchs virtuales definidas por el operador de VNFs (Herrera y Vega, 2016; Barrado, 2018) y pueden ir acompañadas de un Sistema de Gestión de Elementos (EMS) que se encarga de la creación, monitorización y rendimiento de VNFs (Fernández, 2018) es así como lo mencionado, se muestra a continuación en la figura 7.

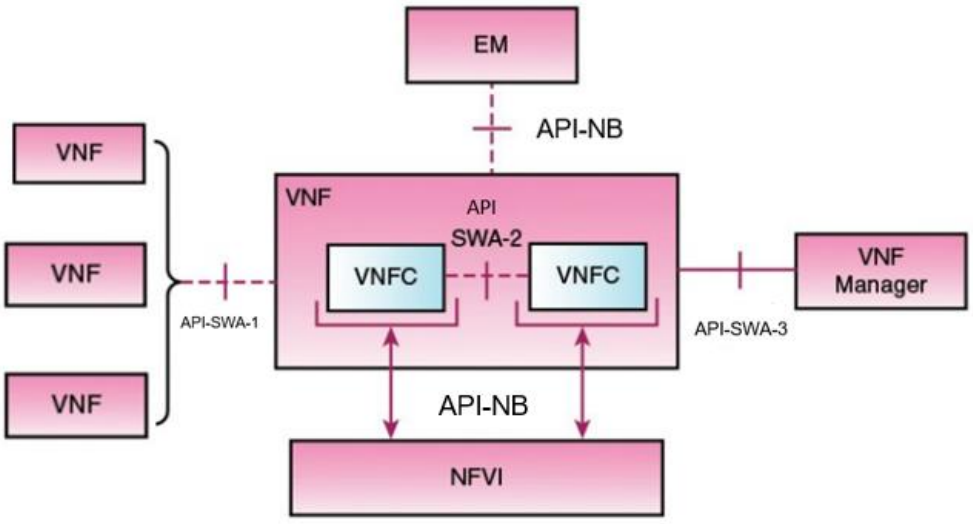

Figura 7. Estructura funcional del bloque VNF
Obtenido de: (Stallings, 2015) y modificado por autor.

Además, para habilitar la comunicación entre los diferentes bloques funcionales dentro de Virtualized Network Function (VNF), se programan APIs de interfaz virtual, en

otras palabras, con una API lógica de switch virtual-1 (API-SWA-1) es posible la comunicación entre VNFs (Stallings, 2015), es decir, conecta el conjunto funcional VNF mas no los VNFCs individuales. La API-SWA-1 son interfaces lógicas programables que se utilizan principalmente para la conectividad de los servicios de red disponibles dentro de un conjunto VNF (Stallings, 2015; Carbonel, 2018), por otro lado, la API lógica de switch virtual-2 (API-SWA-2) ayuda a la comunicación entre VNFCs dentro de un mismo VNF, es importante mencionar que, esta API puede hacer uso de los servicios de conectividad de red disponibles por la interfaz programable API- northbound (API-NB) (Kaljic et al., 2019).

Si dos VNFCs dentro de un VNF se despliegan en una misma máquina virtual, se puede usar la API lógica de switch virtual-3 (API-SWA-3) para minimizar la latencia y mejorar el rendimiento, como se describe a continuación, la interfaz lógica API-SWA-3 es propia para el administrador VNF (VNF Manager) que se encuentra dentro del bloque de administración y orquestación (MANO) (Stallings, 2015; Carbonel, 2018; Kaljic et al., 2019), es decir, el administrador VNF es el responsable de la gestión del ciclo de vida de cada VNF, para implementar esta interfaz lógica es necesario una conexión de red mediante IP (Herrera y Vega, 2016; Barrado, 2018). Las APIs-NB son interfaces lógicas que describen un entorno propio de ejecución para una instancia VNF, es decir cada VNFC es asignado a un contenedor virtual en este caso una máquina virtual, para la gestión y ejecución dado por parte del sistema de gestión de elementos (EMS) (Chiosi et al., 2013; Stallings, 2015).

A continuación, se detallan varios escenarios en los cuales se utilizan diferentes tecnologías de red para el despliegue y comunicación entre VNFCs , por lo tanto, el escenario más común para tener una VNF, es la comunicación a través de un conmutador virtual desplegado por el hipervisor, con el cual se logra la comunicación básica entre VNFCs, sin embargo, para proporcionar un rendimiento eficiente de los distintos VNFCs (Carbonel, 2018; Kaljic et al., 2019), es mediante la comunicación a través de un interruptor de hardware, es decir en este escenario las máquinas virtuales que alojan los VNFCs omiten el conmutador virtual para acceder directamente a la NIC física (Chiosi et al., 2013; Stallings, 2015).

De la misma manera, con la ayuda de un kit de desarrollo de plano de datos (DPDK), el cual es un conjunto de controladores virtuales y de interfaz de red, desplegados propiamente en el plano de datos, que sirve para mejorar el procesamiento de paquetes en plataformas de arquitectura virtual, se puede lograr un rendimiento mayor debido a que su uso acelera el procesamiento de datos y controladores de red propios de un ordenador físico (Stallings, 2015). Por otro lado, la comunicación a través de un conmutador integrado (eswitch) el cual va implementado en la NIC capaz de soportar virtualización proporciona un método para dividir un dispositivo en múltiples identidades de funciones de red virtuales de manera que se diferencie los flujos de tráfico y se puedan entregar directamente a las VMs (Kaljic et al., 2019), de esta forma es como se evita que flujos de tráfico no deseados afecten a otras VMs y con software de aceleración del plano de datos desplegado en el VNFC (Stallings, 2015; Kaljic et al., 2019).

2.2.1.3 Management and Orchestration (MANO)

El bloque NFV MANO se centra en las tareas de gestión específicas dentro de la arquitectura NFV, es así como, interactúa tanto con el NFVI como con las VNFs (Chiosi et al., 2013; Barrado, 2018). Se encarga de la orquestación y la gestión del ciclo de vida de los recursos físicos (hardware) y lógicos (software) que dan soporte a la virtualización de la infraestructura y la gestión del uso de las VNFs (Fernández, 2018).

Es así como también el segmento MANO consta de una arquitectura con bloques funcionales para el despliegue y orquestación de NFV los cuales se muestran en la figura 8.

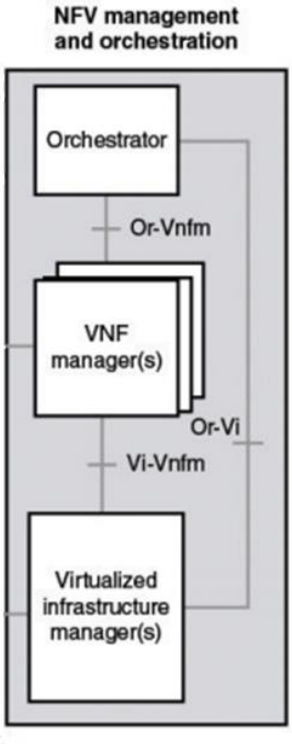

Figura 8. Arquitectura de MANO
Obtenido de: (Barrado, 2018) modificado por autor

- *Virtulization Infrastructure Manager (VIM)*

Un VIM se encarga de gestionar los recursos de una NFVI dependiendo del lugar donde se encuentre desplegado (Carbonel, 2018). Es decir, se puede implementar varios VIMs dentro de la arquitectura NFV y cada uno de ellos gestiona su respectivo NFVI (Fernández, 2018).

El bloque VIM también controla el inventario de máquinas virtuales que se asocian con los recursos virtualizados ya desplegados, que se encargan de la gestión, rendimiento y fallos en el hardware, software y recursos virtuales desplegados. Utiliza northboud APIs para exponer los recursos físicos y virtuales disponibles al sistema de gestión de VNFs (VNF Manager) (Fernández, 2018; Rehman, Aguiar y Barraca, 2019).

- *Virtualized Network Functions Manager (VNFM)*

El VNF Manager se encarga del control y gestión del ciclo de vida de una VNF, por lo tanto, crea, mantiene y termina una instancia VNF (Herrera y Vega, 2016), además es el responsable de todas las funciones FCAPS que gestionan fallos, configuran,

contabilizan, analizan el rendimiento y seguridad de las mismas (Barrado, 2018; Fernández, 2018). Así también, aumenta o reduce el uso de CPU con respecto a la memoria y el almacenamiento de las VNFs dependiendo de las necesidades (Herrera y Vega, 2016).

- *Orchestrator (NFVO)*

Este bloque fundamentalmente tiene dos roles: (1) Orquestador de servicios, que se encarga de desplegar servicios E2E de extremo a extremo formados por distintas instancias VNFs donde se requiera (Bari, Chowdhury, Ahmed, Boutaba y Duarte, 2016; Herrera y Vega, 2016); y, (2) Orquestador de recursos, que coordina, autoriza, comunica y usa recursos NFVI entre las diferentes infraestructuras a través del VIM correspondiente (Bari et al., 2016; Fernández, 2018).

2.2.2 Network Service (NS)

Para obtener una solución NFV es importante que se ejecute un Network Service (NS) que interactúa con las diferentes VNFs desplegadas, es así como, estas dos entidades se gestionan para brindar un segmento de red virtualizado (Carbonel, 2018). Podemos decir que, un NS se comporta dependiendo de cuantas VNFs se han creado en una arquitectura NFV, por lo tanto, definimos a un Network Service como el comportamiento de la combinación de VNFs individuales (Bari et al., 2016).

Cuando una VNF realiza una función de red específica, debe estar compuesta siempre con tres elementos funcionales, descritos a continuación y mostrados en la figura 8: (1) Virtual Deployment Unit (VDU), (2) Virtual Link (VL) y (3) Connection Point (CP); que crean un segmento de red virtualizado juntamente con varias VNFs (Carbonel, 2018). Estos elementos proporcionan a una VNF la capacidad de desplegar y virtualizar una función de red, por lo tanto, son propios de este bloque.

A continuación, se describe el funcionamiento de cada elemento, de modo que, un VDU es el software donde se aloja una función de red específica que comprende las capacidades físicas de un ordenador como la memoria RAM, el número de CPUs, la capacidad de almacenamiento y la interfaz de red (Fernández, 2018); por otro lado, el

VL se encarga de conectar los distintos VDUs y el CP se utiliza para crear una conexión punto a punto entre los VLs con los VDUs, tal como se muestra en la figura 8 (Bari et al., 2016; Carbonel, 2018).

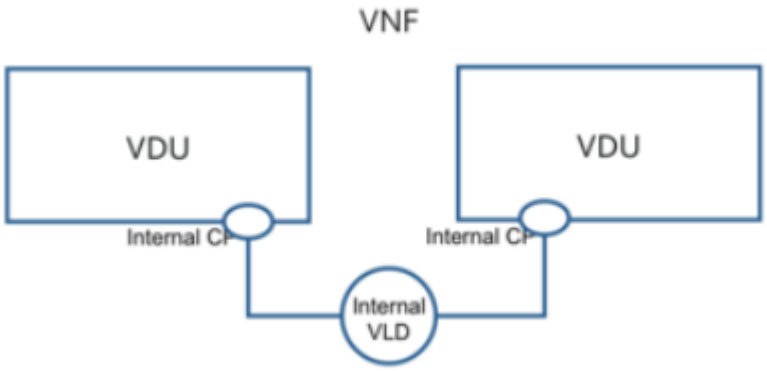

Figura 9. Elementos funcionales de una VNF
Obtenido de: (Barrado, 2018)

2.3 SDN y NFV

Las redes definidas por software (SDN) desacopla los planos de control y datos obteniendo un control del tráfico de la red, para que el enrutamiento de paquetes en la red sea más flexible y eficiente (Cox et al, 2017). Por otro lado, la virtualización de las funciones de red (NFV) desacopla las funciones de red ubicadas en hardware especifico por medio de la virtualización para hacer que la utilización de estas funciones sea más flexible y eficiente (Stallings, 2015). Al converger estas dos tecnologías se logra aplicar la virtualización a las funciones del plano de datos de los routers y otras funciones de red, incluidas las funciones del controlador SDN (Stallings, 2015; Cox et al, 2017). Por lo tanto, SDN y NFV son tecnologías independientes, pero se complementan la una con la otra para obtener mayores beneficios para las redes.

Es así como, SDN y NFV comparten varias características y tienen algunos objetivos en común, tales como: mover la funcionalidad al software, utilizan plataformas básicas y dejan de lado el uso de plataformas propietarias, se comunican con sus diferentes bloques por medio de las interfaces de programación de aplicaciones (API) ya sean

estandarizadas o de código abierto y apoyan a la evolución, implementación y reposicionamiento eficiente de las funciones de red (Schneider et al., 2018).

Se debe tomar en cuenta que, si las funciones de red se implementan en máquinas dedicadas y se usa SDN, las funciones de control desplegadas se ejecutan a través de controladores SDN centrales, que se comunican con los diferentes dispositivos de red (Cox et al, 2017). Por otro lado, si se utiliza NFV, las funciones de red se despliegan en software y se implementan en máquinas virtuales. Es así como se reduce el conjunto de dispositivos de red como son los routers, switches, servidores y se mejora el control y la administración de las funciones que realizan como es el reenvío de paquetes (Cox, J.H et al, 2017; Schneider, S., et al., 2018).
Sin embargo, si queremos implementar SDN con NFV para una red, se debe tener en cuenta las siguientes relaciones:

- Las funciones que cumple el plano de datos de red se deben implementar en las VM. Como también, las funciones que cumple el plano de control se deben desplegar en un controlador SDN dedicado o en una VM. Por lo tanto, el controlador SDN interactúa con las funciones del plano de datos que se despliegan en las VM (Stallings, 2015).

2.4 Open Source Mano (OSM)

Open Source MANO (OSM) es una comunidad de código abierto alojada en ETSI que ofrece una pila de software MANO para la producción de NFV, capaz de consumir modelos de información publicados abiertamente, disponibles y adecuados para todos los VNF que son operacionalmente significativos e independientes de VIM (Carbonel, 2018). OSM está alineado con los modelos de información de NFV, al tiempo que proporciona comentarios de primera mano basados en su experiencia en implementación (Fernández, 2018).

OSM es una herramienta para la gestión y orquestación de VNFs que utiliza descriptores de funciones de red virtualizados (VNFD) que se adhieren a cada VNF, para describir todas las VNFs y los servicios de red de una manera estándar. Es así

como, las VNFs se empaquetan en un paquete denominado VNF, que incluye el VNFD, scripts de configuración y APIs para que los administradores puedan combinar estos paquetes para describir un servicio de red en un Network Service Descriptor (NSD) y proporcionar una instancia de segmento de red virtualizado (Barrado, 2018; Yilma, Yousaf, Sciancalepore y Costa-Perez, 2020).

En la figura 10 se muestra la interacción de OSM con VIM para el despliegue de VNFs y el VL que los conecta, también interactúa con las VNFs desplegadas en un VIM para ejecutar las siguientes primitivas para su configuración (1) Cada VIM tiene un API llamado punto de conexión al que se puede acceder desde OSM, (2) cada VIM posee una red de administración la cual asigna una dirección IP a los VNFs y (3) es posible acceder a la red de administración desde OSM (Avilés, 2017; Barrado, 2018; Fernández, 2018; Yilma et al., 2020).

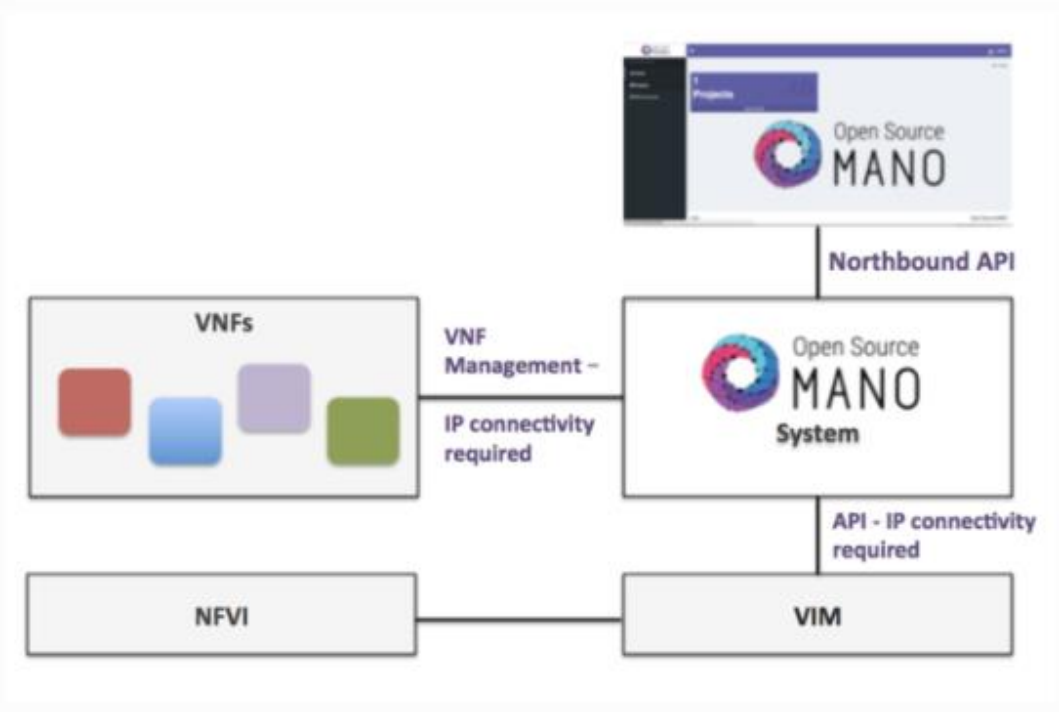

Figura 10. Interacción de OSM con VIM para el despliegue de VNFs.
Obtenido de (Carbonel, 2018)

El principal objetivo de ETSI OSM (Open Source MANO) es el despliegue de un E2E Network Service Orchestrator (E2E NSO) que proporciona servicios de telecomunicaciones virtuales, estos servicios son capaces de modelar y automatizar dispositivos de red reales a nivel de telecomunicaciones, y, además, con toda la complejidad intrínseca en entornos de producción (Reid et al., 2019). Es así como, con la plataforma OSM se acelera la maduración de las tecnologías y estándares para NFV, también, habilita un amplio ecosistema de proveedores para VNF que permiten probar, validar e interactuar con otros componentes tales como: infraestructuras comerciales de NFV (NFVI + VIM) y Funciones de red (ya sean VNF, PNF o híbridas) (Bhagwat et al., 2018).

La plataforma OSM cumple el objetivo mencionado anteriormente gracias a cuatro aspectos claves que minimizan los esfuerzos de su despliegue, los cuales son:

1) Un modelo de información (IM) bien conocido, alineado con ETSI NFV, que es capaz de modelar y automatizar el ciclo de vida completo de funciones de red (virtuales, físicas o híbridas), servicios de red (NS) y segmentos de red (NSI), desde su implementación inicial hasta su operación y monitoreo diarios (Bhagwat et al., 2018; Reid et al., 2019). Por lo tanto, la mensajería instantánea de OSM es completamente independiente de la infraestructura, por lo que el mismo modelo se puede utilizar para instanciar un elemento determinado (por ejemplo, VNF) en una gran variedad de tipos de VIM y tecnologías de transporte, lo que permite un ecosistema de modelos VNF listos para su implementación en todas partes (Bhagwat et al., 2018; Reid et al., 2019).

2) OSM proporciona una interfaz API unificada en dirección norte (NBI), basada en NFV SOL005 (Reid et al., 2020), que permite el funcionamiento completo del sistema y los servicios de red y los segmentos de red bajo su control. De hecho, el NBI de OSM ofrece el servicio de gestionar el ciclo de vida de los Servicios de Red (NS) y las Instancias de Slices de Red (NSI) (Reid et al., 2019; Reid et al., 2020), brindando como servicio todas las abstracciones necesarias para permitir el completo control, operación y supervisión del ciclo

de vida del NS / NSI sistemas del cliente, evitando la exposición de detalles innecesarios de sus elementos constitutivos (Reid et al., 2019; Reid et al., 2020).

3) El concepto Servicio de Red (NS) en OSM, debe ser capaz de abarcar los diferentes dominios identificados, como: virtuales, físicos y de transporte, por lo tanto, para controlar el ciclo de vida completo de una NS que interactúa con VNF, PNF y HNF debe ser junto con conexiones de transporte conectadas bajo el despliegue de APIs entre diferentes dominios (Bhagwat et al., 2018).

4) Por otro lado, para ofrecer un servicio agregado, OSM también puede administrar el ciclo de vida de los Network Slices, el cual asume si es necesario el rol de Slice Manager, para posteriormente extenderlo para soportar una operación integrada (Bhagwat et al., 2018; Reid et al., 2020).

Además, OSM brinda la capacidad de hacer realidad una de las promesas de NFV y con las capacidades dinámicas que aporta, es decir, crear redes bajo demanda (NaaS: Network as a Service) para su explotación directa por parte del proveedor de servicios (Reid et al., 2020).

Según lo mencionado en el anterior apartado, OSM funciona como un orquestador de servicios de red (NSO), en la cual administra una plataforma de servicios de red, con la intención de proporcionar la capacidad de crear servicios de red que se puedan ejecutar y desplegar más tarde como un controlador de operaciones de servicio de red a través de interfaces virtuales API en dirección norte propias de OSM (Bhagwat et al., 2018).Para la plataforma OSM, existen dos tipos de servicios NaaS para soportar la capacidad de una red como servicio, los cuales son: el Servicio de red (NS) y la Instancia de segmento de red (NSI), este último es una composición de varias redes o segmentos de red (Reid et al., 2019). Estos tipos de servicios pueden tratarse como una sola entidad.

La plataforma OSM en función de administrador, consume servicios de otras plataformas (por ejemplo, OpenStack o OpenDaylight) y controla una serie de funciones gestionadas para crear sus propios objetos de servicios de red compuestos de nivel superior (Avilés, 2017; Barrado, 2018; Fernández, 2018; Yilma et al., 2020). Por lo tanto, OSM consume y asume los servicios prestados por las diferentes plataformas a cargo de la Infraestructura Virtual (para obtener VM, etc.) y las plataformas a cargo de la Red Definida por Software y, una vez ensambladas, configura y monitorea las funciones de la red que la componen (VNF, PNF, HNF) para controlar el Logical Control Managment (LCM) de todo el NS / NSI que se ofrece a lo largo de su despliegue (Barrado, 2018).

En la Figura 11 se resalta lo detallado anteriormente como parte de una arquitectura en función de administrador de OSM para NFV:

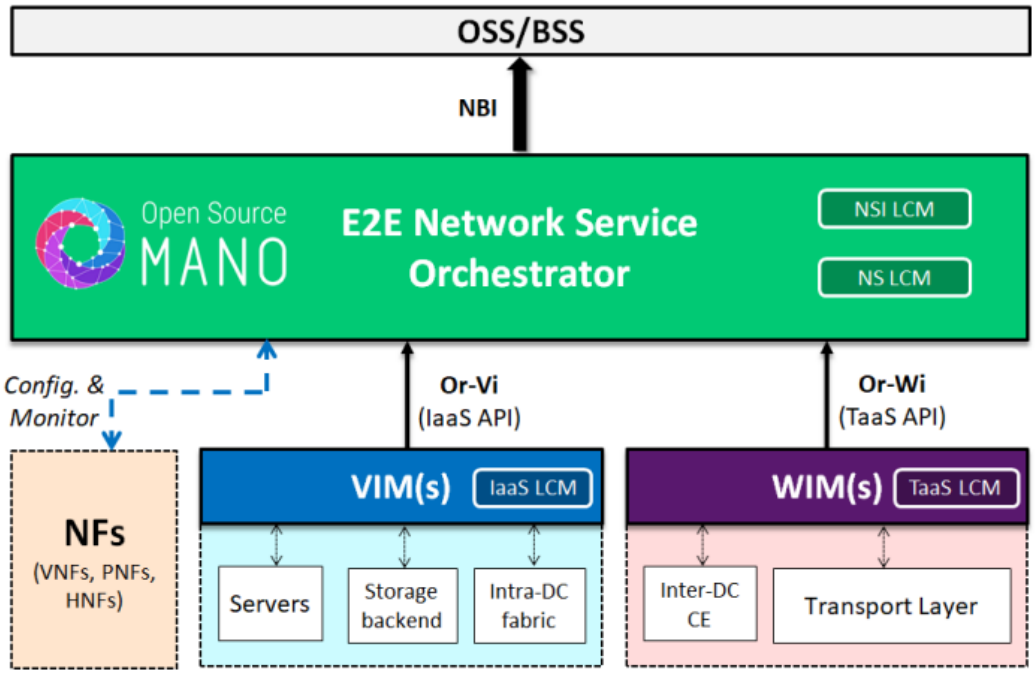

Figura 11. Arquitectura OSM en función de administrador.
Obtenido de: (Bhagwat et al., 2018)

El servicio de red (NS) en el E2E Network Orchestrator, es el bloque de construcción principal en OSM para administrar redes proporcionadas como un servicio que se agrupan en un solo objeto, es decir un conjunto de funciones de red interconectadas

(VNF, PNF y HNF) que pueden abarcar diferentes tecnologías (virtual o físicas), ubicados de forma centralizada y que pueden ocupar grandes áreas geográficas, por ejemplo, como parte del servicio de un gran cliente corporativo multinacional (Barrado, 2018; Reid et al., 2019).

Para habilitar efectivamente estos servicios de red recién creados se proporciona el resultado de un método simple y conocido como el despliegue de API (a OSM's NBI) y descriptores siguiendo el modelo de información de OSM según la ETSI (Bhagwat et al., 2018; Barrado, 2018). Estas interfaces API deben facilitar la creación de servicios de red compuestos por diferentes dispositivos de red (VNF, PNF o HNF) que provienen de diferentes proveedores (también llamados funciones de red o NF), de modo que esos dispositivos puedan venir modelados previamente por su proveedor y el proveedor de servicios puede controlar su propio servicio de red (Avilés, 2017; Barrado, 2018).

Además, cuando un servicio de red está completamente modelado en un paquete de servicios de red, el modelo funciona de manera efectiva como una plantilla que se puede particularizar (Bhagwat et al., 2018), es decir, en el momento de la creación de NS para incorporar atributos específicos para el despliegue de instancias VNFs, se devuelve un ID de instancia de NS único, útil para impulsar las operaciones de LCM (Bhagwat et al., 2018; Reid et al., 2019). La plataforma OSM también pone en marcha todos los requisitos necesarios para lograr un completo control de operación y supervisión del ciclo de vida del NS generalmente, por plataformas OSS / BSS. En consecuencia, este bloque OSS/BSS controla instancias NS, con el fin de minimizar el impacto sobre el servicio final de posibles cambios en las NF o la topología NS (Reid et al., 2020).

Para lograr el nivel deseado de flexibilidad en un servicio de red virtualizado, OSM aumenta el concepto de NS con respecto a ETSI NFV para incorporar dominios físicos y de transporte para permitir servicios E2E reales que se pueden extender más allá de los dominios virtuales (Reid et al., 2020). Por lo tanto, con OSM es posible:

- Combinar en un único NS funciones de red virtual (VNFs), funciones de red física (PNFs), e incluso, funciones de red compuestas por elementos tanto

físicos como virtuales (HNF: Hybrid Network Functions), más propios de elementos más cercanos a la red de acceso (Bhagwat et al., 2018; Reid et al., 2019).

- Implementar tales NS en una red distribuida e inclusive crear conexiones de transporte entre sitios a pedido, aprovechando las API de las plataformas de redes definidas por software (Barrado, 2018).

Se espera que tanto las plataformas OSS como BSS sean consumidores del NS creado a pedido por OSM y, a veces, incluso pueden mantener el control de algunas funciones de red constituyentes del NS si es necesario esto es bastante útil para reutilizar los nodos de red heredados sin requerir cambios importantes en el OSS (Bhagwat et al., 2018; Reid et al., 2019; Reid et al., 2020). Es por esto que, OSM también tiene la capacidad de delegar selectivamente el control de NF constituyentes específicos del NS a la plataforma OSS / BSS si se especifica explícitamente en el modelo NS, dando total libertad para soportar escenarios heredados o híbridos según se requiera (Bhagwat et al., 2018).

A continuación, se analizan y describen detalladamente las etapas relacionadas con el ciclo de vida y el funcionamiento del NS en el E2E Network Orchestrator, proceso que consta de Modelado, Inducción, creación de NS, operación de NS y Finalización; de modo que las funciones de las APIs y el IM complementario se puedan comprender mejor en un entorno de funcionamiento.

En primer lugar, se tiene la etapa de MODELADO, donde OSM proporciona un modelo de arquitectura para incluir el comportamiento completo de NS, con una descripción de la topología de NS, las operaciones del ciclo de vida y las primitivas de NS disponibles, junto con su código de automatización (Bhagwat et al., 2018; Reid et al., 2019). Por lo tanto, los Servicios de Red se componen, por definición, de una o varias Funciones de Red (VNF, PNF o HNF) de tipos heterogéneos y comportamientos internos que probablemente provengan de diferentes proveedores, por otro lado (Bhagwat et al., 2018; Reid et al., 2019; Reid et al., 2020), el IM proporciona un medio

para permitir que el proveedor describa topologías, recursos necesarios, procedimientos y ciclo de vida de las funciones de red (Reid et al., 2019; Reid et al., 2020). Esta información viene incluida en los denominados paquetes NF que proporcionan dos capas que tienen varias ventajas mencionadas a continuación:

- Permite que el diseñador del Paquete NS, esté expuesto directamente a los dominios internos de un NF, centrándose en la composición de la misma NS, basándose exclusivamente en las propiedades y procedimientos externos de un NF (Bhagwat et al., 2018; Reid et al., 2019).

- Permite la validación coherente de un NF y sus paquetes NF complementarios en todo su despliegue, de modo que el proveedor de NF pueda garantizar que sus elementos se utilicen y operen siempre de la manera adecuada (Bhagwat et al., 2018; Reid et al., 2020).

- Es así como, el mismo paquete NF se puede utilizar en más de una NS sin trabajo de modelado adicional a nivel NF (Bhagwat et al., 2018).

Una vez que los modelos están listos, la siguiente etapa es la INDUCCION, en la cual, se implementa los modelos al sistema (Bhagwat et al., 2018), es decir, de modo que se puedan usar como plantillas para la creación de NS más adelante, en un proceso que se conoce como **inducción** (Barrado, 2018).

Las APIs NBI de OSM ofrecen operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los paquetes NS y NF correspondientes (Bhagwat et al., 2018; Reid et al., 2019; Reid et al., 2020), con el fin de admitir el enfoque de modelado descrito anteriormente, además, las APIs admite operaciones CRUD específicas para manejar los paquetes NS y NF correspondientes y NST cuando corresponda como objetos independientes pero relacionados (Reid et al., 2019; Reid et al., 2020). En estas operaciones, y particularmente en el proceso de inducción, se realizan las comprobaciones necesarias para validar la consistencia en el modelo y entre modelos (Reid et al., 2019).

Según Bhagwat et al., (2018) en la publicación de OSM “OSM Scope Functionality, Operation and Integration Guidelines” Una vez que los paquetes NS y NF correspondientes se incorporan con éxito en OSM, se inicia la etapa de CREACION DE NS, donde se obtiene todo lo necesario para la creación real de NS. En consecuencia, OSM ofrece las APIs para admitir operaciones CRUD (Crear, Leer, Actualizar, Eliminar) relacionadas con instancias NS (Reid et al., 2020).

En el proceso de creación de NS, también conocida como instanciación de NS, OSM toma como entrada un paquete de NS y, opcionalmente, un conjunto de restricciones de implementación adicionales (Reid et al., 2020), por ejemplo, ubicaciones de implementación de destino para VNF específicas de la NS y parámetros para particularizar en NS, según lo permitido explícitamente por el paquete NS (Reid et al., 2019; Yilma et al., 2020).

Durante la creación de NS, OSM interactúa con diferentes plataformas de servicio en dirección sur (Yilma et al., 2020), en este caso VIM y funciones administradas (NF) para crear el objeto de servicio compuesto de la instancia de NS (Bhagwat et al., 2018).

Una vez que la NS se ha creado con éxito, comienza la etapa de OPERACIÓN NS que define requerimientos para acciones de operación, la cuales son: ciclo de vida y garantía.

Por lo tanto, cuando ya se ha creado con éxito una NS, la instancia de NS se convierte en el único objeto relevante para acciones posteriores de operación, ciclo de vida y garantía (Bhagwat et al., 2018; Barrado, 2018). La instancia NS / NSI puede estar sujeta a diferentes tipos de operaciones impulsadas por API, que se incluyen en una de estas categorías:

- *Operaciones comunes del ciclo de vida:* Debe haber una serie de APIs que permitan desencadenar acciones estándar aplicables a cualquier NS (Bhagwat et al., 2018), como acciones de escalado, pausa, reanudación, solicitudes de

monitoreo bajo demanda, actualizaciones de Software, etc (Reid et al., 2019; Reid et al., 2020).

- *Acciones derivadas de primitivas NS:* Además de las operaciones potencialmente aplicables a cualquier NS / NSI, cada NS / NSI puede tener un conjunto de operaciones que son relevantes solo para la funcionalidad específica que ofrece el NS / NSI, como la adición de nuevos suscriptores, cambios en el enrutamiento interno, etc (Reid et al., 2019; Reid et al., 2020). Esas acciones se enumeran y codifican en el paquete NS correspondiente que, a su vez, aprovechan las acciones atómicas disponibles en los paquetes NF, estas operaciones están expuestas por APIs como acciones primarias disponibles en ese NS / NSI dado (Avilés, 2017; Barrado, 2018; Fernández, 2018).

Por último, la etapa de Finalización NS; como cualquier otro servicio bajo demanda, es posible finalizar un NS y liberar los recursos que se habían asignado (Avilés, 2017; Reid et al., 2020)., preservando aquellos componentes que no deben ser eliminados, por ejemplo, recursos de alucinamiento o de red.

Para finalizar una NS existe un término mencionado por (Bhagwat et al., 2018) el cual es “delete” de la API, que se refiere a la eliminación de las operaciones CRUD antes mencionadas relacionadas con una NS, que es la encargada de desencadenar ese proceso y poner un estado de finalización.

Ahora, es importante mencionar el lenguaje de programación que utiliza OSM para desplegar VNFs, es decir, Según Chiosi et al. (2013) en la publicación de la ETSI “Network Functions Virtualization: Network Operator Perspectives on Industry Progress” establece el lenguaje de programación que opera el módulo RO el cual es Python3 y los descriptores de programación utilizan la sintaxis YAML o JSON, la ETSI optó por usar la sintaxis YAML porque es más legible que permite utilizar comentarios en el código, también posee varias características de diferentes lenguajes de programación como Perl, C, XML, HTML además, es compatible con JSON de manera que las opciones que proporciona JSON son compatibles con YAML.

Cabe recalcar, que la estructura de un archivo YAML puede ser una lista o un mapa, mediante esta opción es posible establecer pares en sentido clave-valor que deben ejecutarse antes de que se pueda cerrar el archivo YAML, por lo tanto, esto permite crear un nuevo mapa, al cual se puede agregar nuevas sentencias obteniendo un mapa adyacente al primero. Si se crea una lista, esta debe tener valores enumerados en un orden específico, que puede ser cualquier carácter, número o sintaxis necesario para su ejecución; por lo tanto, la lista es posible insertar en un mapa similar a una secuencia de programación de Python.

Por consiguiente, OSM ha optado usar YAML para ejecutar Kubernetes, en función de un estado limitado y otro preciso, es decir, al ejecutar objetos de Kubernetes es posible representar un clúster que ejecute una carga de trabajo por medio de la sintaxis de un archivo YAML.

Es por esto que, al crear un objeto de Kurbernetes, se define las características que debe presentar el estado del objeto por medio de una API de Kubernetes, la cual permite enlazar especificaciones al objeto por defecto que proporciona información a través de un kubectl de un archivo YAML. Es decir, por medio de la API la opción Kubectl se traduce a un archivo YAML Chiosi et al. (2013).

2.5 Proxmox VE

Proxmox VE es una plataforma de administración de servidores de código abierto para la virtualización empresarial. Es decir, integra estrechamente el hipervisor KVM y LXC, el almacenamiento definido por software y la funcionalidad de red en una sola plataforma, es posible administrar con la interfaz de usuario integrada basada en web, que puede gestionar fácilmente máquinas virtuales y contenedores, clústeres de alta disponibilidad o las herramientas integradas de recuperación ante desastres. Es una plataforma Cloud Computing que admite todo tipo de entornos en la nube. El objetivo de Proxmox VE es una implementación sencilla que brinde una escalabilidad masiva con un amplio conjunto de características para proporcionar una solución de infraestructura como servicio (IaaS) la cual consiste en controlar el procesamiento, almacenamiento y conectividad de infraestructuras en la nube (Garzón y Atehortua,

2018). Es así como, Proxmox VE es óptimo para ser un módulo VIM en la arquitectura MANO (Avilés, 2017; Fernández, 2018).

La plataforma Proxmox VE brinda características de clase empresarial y un enfoque 100% basado en software siendo la opción perfecta para virtualizar una infraestructura de TI, así como también, optimiza los recursos existentes y aumentar la eficiencia con un gasto mínimo. Por otro lado, es capaz de virtualizar fácilmente las cargas de trabajo de aplicaciones Linux y Windows más exigentes y complejas hasta escalar dinámicamente la informática y el almacenamiento a medida que crecen sus necesidades, es así como Proxmox VE garantiza que cualquier centro de datos se ajuste para el crecimiento futuro.

Además según Chen, L et al., (2017) Proxmox VE proporciona una capa virtual en servidores físicos, desacoplando el hardware subyacente de la carga de trabajo para proporcionar a sus usuarios recursos de red, almacenamiento y computación virtualizados (Fernández, 2018; Saghir, A., Masood, T. 2019), todo esto lo realiza por medio de dos tecnologías de virtualización: KVM (Máquina virtual basada en kernel) para el despliegue de máquinas virtuales y LXC para contenedores, los cuales reenvían paquetes de datos interna y externamente a través de componentes de red a nivel de infraestructura con una única interfaz basada en web (Garzón y Atehortua, 2018; Saghir, A., Masood, T. 2019).

Es así como, esta plataforma de virtualización proporciona una abstracción de API para permitir que los administradores de red gestionen enrutadores lógicos. Por lo tanto, estos enrutadores lógicos se pueden utilizar para conectar redes lógicas entre sí y con el mundo exterior, por ejemplo: Internet, proporcionando seguridad y capacidades NAT (Avilés, 2017; Fernández, 2018). En la actualidad, el 84% de los proveedores de servicios de comunicación confían en las capacidades de la plataforma Proxmox VE para una virtualización de servicios de red eficiente, ciertamente a medida que el mundo entra en el ámbito de la virtualización de funciones de red, numerosos proveedores de telecomunicaciones y líderes empresariales como At & T, Bloomberg, Deutsche Telecom, etc. están optando por implementar NFV con Proxmox VE (Garzón

y Atehortua, 2018). Es decir, según Saghir, A., Masood, T. (2019), los dispositivos de red dedicados, como los enrutadores, los servidores del Servicio de nombres de dominio (DNS) y los firewalls, se reemplazan por dispositivos virtuales que se ejecutan en servidores comerciales listos para usar, la ventaja de utilizar la plataforma Proxmox VE para NFV es que los proveedores de servicios pueden implementar más rápidamente nuevos servicios y VNF utilizando software en lugar de redes de hardware especializadas (Saghir, A., Masood, T. 2019).

2.5.1 Arquitectura Proxmox VE

Para la virtualización del servidor con soporte para KVM y LXC Proxmox VE se basa en Debian GNU/Linux y utiliza un kernel linux personalizado, además, el código fuente Proxmox VE es gratuito, lanzado bajo la Licencia Pública General GNU Affero, v3 (GNU AGPL, v3). Esto significa que es libre de utilizar e inspeccionar el código fuente en cualquier momento garantizando el acceso completo a todas las funcionalidades en cualquier momento, así como a un alto nivel de fiabilidad y seguridad.

La KVM (Máquina virtual basada en kernel) es la tecnología de virtualización de Linux líder en la industria para la virtualización completa. Es un módulo kernel fusionado en el kernel de Linux de línea principal y se ejecuta con un rendimiento casi nativo en todo el hardware x86 con soporte de virtualización, ya sea Intel VT-x o AMD-V. Por lo tanto, con KVM, Proxmox VE se puede ejecutar en Windows y Linux, en máquinas virtuales (VM) donde cada máquina virtual tiene hardware virtualizado privado: una tarjeta de red, disco, adaptador de gráficos, etc. La ejecución de varias aplicaciones en máquinas virtuales en un solo hardware le permite ahorrar energía y reducir costes, al mismo tiempo que le ofrece la flexibilidad de crear un centro de datos ágil y escalable definido por software (SDN) que satisfaga cualquier tipo de demanda de red.

Otra parte fundamental en la arquitectura de Proxmox VE es la virtualización basada en contenedores, tal es el caso de los contenedores Linux (LXC) los cuales son un entorno de virtualización de nivel de sistema operativo para ejecutar varios sistemas Linux

aislados en un único host de control Linux. Es por esto que, LXC funciona como una interfaz de espacio de usuario para las características de contención del kernel de Linux, es decir, los usuarios pueden crear y administrar fácilmente contenedores de sistemas o aplicaciones con una API eficaz y herramientas sencillas.

Por otro lado, un clúster de alta disponibilidad (Clúster HA) de varios nodos permite el despliegue de servidores virtuales de alta disponibilidad, es decir, un clúster Proxmox VE HA que se basa en tecnologías probadas de Linux HA, que proporcionan un servicio de alta disponibilidad estable y fiable. El administrador de recursos, denominado Proxmox VE HA Manager, supervisa todas las máquinas virtuales y el contenedor de todo el clúster y entra automáticamente en acción si se produce un error en una de ellas. Es así como, Proxmox VE HA Manager funciona de inmediato. Además, toda la configuración del clúster Proxmox VE HA se puede configurar fácilmente con la interfaz de usuario integrada basada en web.

Proxmox VE puede realizar todas las tareas de administración con la interfaz gráfica de usuario (GUI) integrada, no es necesario instalar una herramienta de administración independiente, por lo que, la interfaz web central se basa en el marco de JavaScript de ExtJS y puede utilizarla con cualquier navegador moderno. De esta manera, ayuda a controlar todas las funcionalidades de la interfaz de usuario y a obtener información general sobre el historial o los syslogs de cada nodo único. Por lo tanto, con esto incluye la ejecución de tareas de copia de seguridad, migración en vivo, almacenamiento definido por software o actividades desencadenadas por HA. La interfaz de administración basada en web es una herramienta múltiple que permite gestionar todo el clúster desde cualquier nodo del clúster; cabe recalcar que no necesita un nodo de administrador dedicado.También Proxmox VE utiliza una API RESTful con formato de datos principal JSON, la estructura de toda la API se define formalmente mediante este esquema. Por lo que, esto permite una integración rápida y sencilla para herramientas de administración de terceros, como entornos de alojamiento personalizados.

Por último, Proxmox VE utiliza un modelo de red punteado en el cual Bridges operan como conmutadores de red físicos implementados en el software en el host Proxmox VE. En resumen, todas las máquinas virtuales pueden compartir un bridge como si los cables de red virtual de cada host estuvieran conectados al mismo conmutador y para conectar las máquinas virtuales al mundo exterior, estos Bridges se asocian a las tarjetas de red físicas asignadas a una configuración TCP/IP. Para obtener una mayor flexibilidad, Proxmox VE usa el estándar VLAN (IEEE 802.1q) y la unión/agregación de red. De esta manera es posible construir redes virtuales complejas y flexibles para los hosts Proxmox VE, aprovechando toda la potencia de la pila de red Linux.

2.6 OpenDaylight

OpenDaylight es una plataforma de código abierto (open Source) creado por la Fundación Linux que incluye la participación de todas las organizaciones de redes importantes, tales como, los usuarios de tecnología SDN y los proveedores de productos SDN (Stallings, 2015; Cox et al, 2017). Por otro lado, OpenDaylight tiene como objetivo producir una plataforma de red virtual extensible y de código abierto sobre estándares existentes como OpenFlow (Cox, J.H et al, 2017; Schneider, S., et al., 2018). Además, la plataforma es capaz de permitir que los administradores de red se reúnan para desarrollar módulos centrales de código abierto en colaboración, donde los desarrolladores pueden agregar un valor único con el objetivo de crear productos y tecnologías comerciales (Stallings, 2015).

La Figura 12 proporciona una vista de nivel superior de la arquitectura de OpenDaylight. Es decir, consta de cinco capas lógicas: Network applications orchestration and services, OpenDaylight APIs, Controller functions and services, Service Abtraction Layer (SAL) y Southbound interfaces and protocols que se describen a continuación.

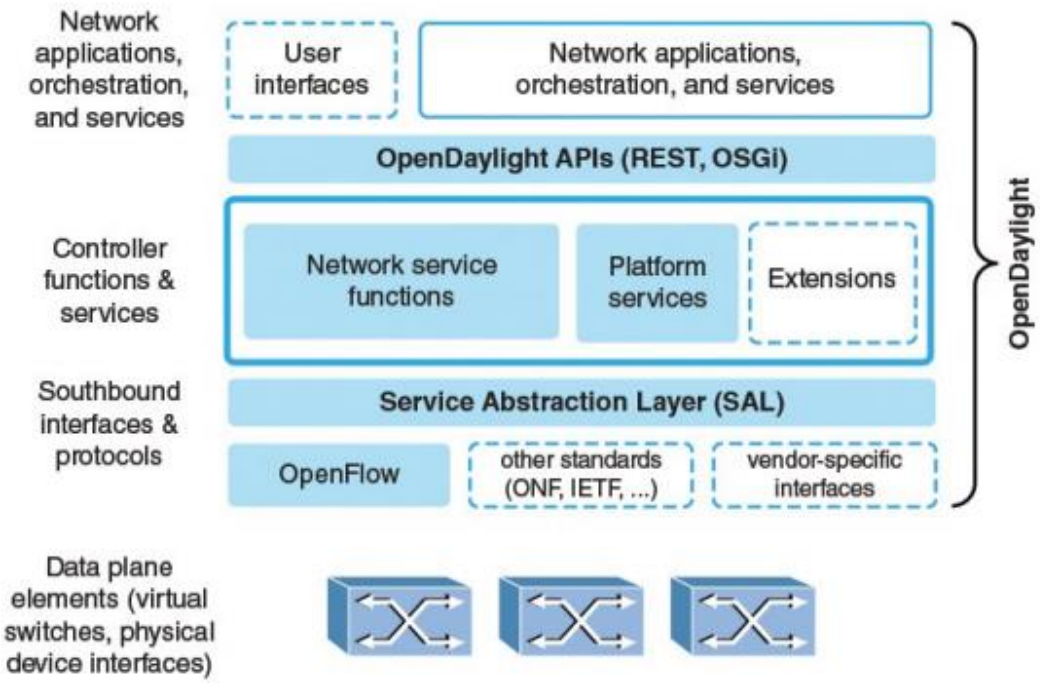

Figura 12. Arquitectura OpenDaylight

Obtenido de: (Stallings, 2015).

La primera capa de la arquitectura OpenDaylight es **Network applications, orchestration and services** que consta de aplicaciones lógicas de red que controlan y supervisan el comportamiento de la red (Bari et al., 2016; Fernández, 2018). Por lo tanto, estas aplicaciones usan el controlador para recopilar inteligencia de red, es decir, ejecutar algoritmos para realizar un análisis y luego para orquestar las nuevas reglas en toda la red (Stallings, 2015).

La segunda capa es denominada **OpenDaylight APIs** que es un conjunto de interfaces comunes para las funciones del controlador OpenDaylight, de tal forma que admite una tecnología de la iniciativa OSGi (Open Service Gateway Initiative) y REST (Representational State Transfer) bidireccional para la API en dirección norte (Stallings, 2015; Bari et al., 2016). Ciertamente OSGi se usa para aplicaciones que se ejecutan en el mismo espacio de direcciones que el controlador, mientras que la API REST (basada en web) se usa para aplicaciones que no se ejecutan en el mismo espacio de direcciones (o incluso necesariamente en la misma máquina) como controlador (Bari et al., 2016).

Como tercera capa es **Controller functions and services** donde se detallan las funciones y servicios del plano de control SDN.

La capa **SAL (Service abtraction layer)** proporciona una vista uniforme de los recursos del plano de datos, de modo que las funciones del plano de control se pueden implementar independientemente de la interfaz y el protocolo específicos en dirección sur (Fernández, 2018).

Y por último la capa **Southbound interfaces and protocols** (Interfaces y protocolos en dirección sur) que admite OpenFlow, varios protocolos estándar en dirección sur e interfaces específicas de proveedores.

Hay varios aspectos importantes de mención en la arquitectura de OpenDaylight, como por ejemplo que abarca tanto el plano de control y la funcionalidad del plano de aplicación (Stallings, 2015; Bari et al., 2016). Es decir, OpenDaylight es más que una implementación de controlador SDN que permite a los administradores de redes empresariales y de telecomunicaciones alojar software de código abierto en sus propios servidores para construir una configuración de red definida por software (Stallings, 2015; Kaljic et al., 2019). De esta manera, los proveedores pueden utilizar este software para crear productos con funciones y servicios adicionales de valor agregado en el plano de la aplicación.

Por otro lado, es importante recalcar que el diseño de OpenDaylight no está vinculado a OpenFlow ni a ninguna otra interfaz específica en dirección sur, por lo que proporciona una mayor flexibilidad en la construcción de configuraciones de red SDN (Stallings, 2015; Cox, J.H et al, 2017; Schneider, S., et al., 2018). Cabe recalcar que el elemento clave en este diseño es la capa SAL, la cual permite que el controlador admita múltiples protocolos en la interfaz en dirección sur y brinde servicios consistentes para las funciones del controlador y para las aplicaciones de capas superiores (Stallings, 2015; Cox, J.H et al, 2017). El funcionamiento de la tecnología OSGi es proporcionar la vinculación dinámica de complementos para los protocolos en dirección sur disponibles, es decir, las capacidades de estos protocolos se resumen en una colección

de características que pueden ser desplegadas por los servicios del plano de control a través de un administrador de servicios en la capa SAL (Cox, J.H et al, 2017; Schneider, S., et al., 2018). De esta manera, el administrador de servicios de red mantiene un registro que asigna las solicitudes de servicio a las solicitudes de funciones y según la solicitud de servicio, la capa SAL asigna al complemento adecuado y, por lo tanto, utiliza el protocolo en dirección sur más apropiado para interactuar con un dispositivo de red determinado (Stallings, 2015).

3. CAPITULO III
DISEÑO E IMPLEMENTACION DE LA TOPOLOGIA DE RED

Este capítulo describe el procedimiento de diseño e implementación de la topología de red, con el objetivo de tener un ambiente de red controlado por plataformas Open Source Mano, Proxmox VE y OpenDaylight, para realizar un análisis Testbed para el estudio de la virtualización de las funciones de red NFV, es así como también se detallan los requisitos de hardware y software necesarios, la descripción de la metodología usada y el procedimiento de instalación y configuración de cada plataforma. Además, por último, se detalla la conexión entre estas plataformas para que interactúen y así conseguir la arquitectura MANO para llegar a virtualizar al menos una función de red.

3.1 Metodología

En cuanto a la metodología a desarrollar, el proyecto tendrá un enfoque mixto ya que es necesario tomar un análisis bibliográfico sobre la tecnología, además de poner en práctica todo lo planteado. Por lo tanto, el proyecto se realizará en cuatro etapas: Estado del arte y análisis situación actual Data Center, Análisis de la Red Definida por Software (SDN) desplegada por plataformas Proxmox VE y OpenDaylight, Análisis y despliegue de la plataforma OpenSource MANO (OSM), por último, Topología y conectividad entre plataformas OpenSource MANO (OSM), OpenDaylight y Proxmox VE; las mismas que se detallan a continuación:

3.1.1 ETAPA 1: Estado del arte y análisis situación actual Data Center

La primera etapa menciona la recopilación de datos y análisis del estado del arte del Data Center ubicado en la Facultad de Ingeniería en Ciencias Aplicadas (FICA) analizado y documentado por Nicolalde (2021), en su Diseño de un Sistema de Detección de Intrusos (IDS), basado en redes neuronales para una Red Definida por Software (SDN). Es importante mencionar que, el Data Center permite la conectividad de toda la facultad tanto como hacia internet, a los diferentes servicios y plataformas desplegadas en el mismo. Es así como, se analizaron las plataformas Proxmox VE y

OpenDaylight (ODL) las cuales representan un escenario óptimo de una Red Definida por Software (SDN), es decir, poseen una infraestructura hiperconvergente virtualizada implementada en los servidores ProxmoxVE (PV1, PV2 y PV3), con el fin de llevar al proyecto a cumplir los objetivos planteados y documentar los procesos realizados.

El principal objetivo del Data Center ubicado en la oficina administrativa de la carrera de Ingeniería en Telecomunicaciones (CITEL), es alojar equipos TICs para crear nuevas formas de comunicación a través de herramientas de redes de nueva generación, por otro lado, también funciona como una red redundante para el anillo de fibra ubicado en la Universidad Técnica del Norte, es así como, a continuación, se detalla y describe la distribución de racks con el despliegue de sus respectivos equipos para proporcionar un ambiente de red controlado por herramientas TICs.

Seguidamente en la figura 13 se muestra el sistema de telecomunicaciones desplegado en el Data Center – FICA detallando la función que cumple cada rack dependiendo del equipo desplegado con su respectiva tecnología, por lo tanto, en el Rack 1 se encuentra un switch de capa tres, el cual permite la conexión con toda la red universitaria a través de dos hilos de fibra óptica operando como equipo de borde que permite la salida a internet. Por otro lado, en el Rack 2 se encuentran desplegados varios servidores que cumplen funciones específicas, tal es el caso de un servidor Radius que conectado a un router Mikrotik y un switch 3Com 4500G administra la red inalámbrica de la facultad, además, posee cuatro servidores denominados Proxmox Virtual (PV) de los cuales, tres de estos se encargan de proporcionar infraestructura física adecuada para virtualizar equipos, es decir, los servidores PV1, PV2 y PV3 con la plataforma de virtualización Proxmox VE permiten virtualizar cualquier equipo de red. Por último, el Rack 3 esta orientado a brindar servicios administrativos y de gestión por medio de cuatro servidores denominados Opina, Reactivos, Biométricos y Revista universitaria conectados por un switch QPcom, este rack brinda servicios educativos orientados a estudiantes y docentes de la facultad.

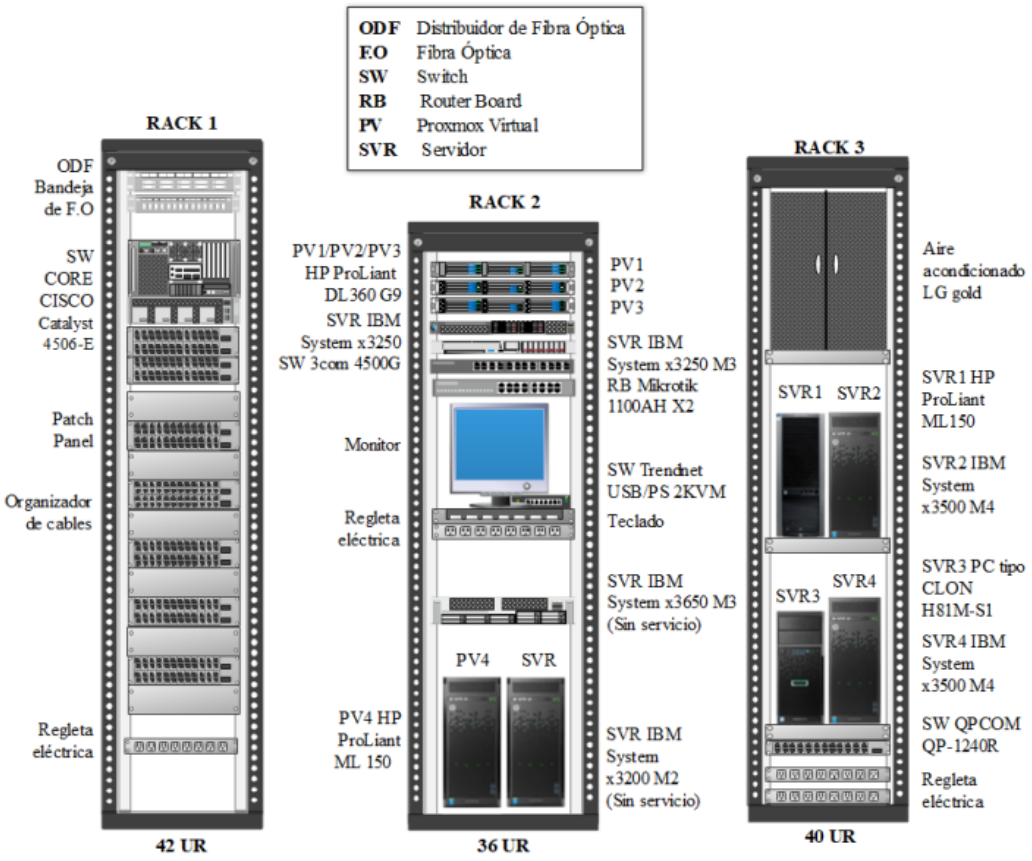

Figura 13. Análisis situación actual Data Center.

Obtenido de: Nicolalde (2021).

Además, para lograr un ambiente de red controlado, en el Data Center se ha segmentado físicamente la red de datos en tres partes, de acuerdo con el siguiente detalle:

- **Primer segmento:** es el encargado de conectar y dar acceso a los servidores Opina, Biométrico, Reactivos y Revista Universitaria, de manera que está conectado desde el puerto 28 del switch de core Catalyst 4506-E ubicado en el Rack 1 hasta el switch 3Com 3226 donde se encuentran los servidores de administración ubicados en el Rack 3 mencionados anteriormente.

- **Segundo segmento:** cumple la función de comunicar los servidores que administra la carrera de Ingeniería en Sistemas Computacionales, mediante el puerto 27 del Switch principal Catalyst 4506-E hasta un switch LinkSys SR224G.

- **Tercer segmento:** el más importante para el presente proyecto, se encarga de conectar toda la infraestructura virtualizada Proxmox, por medio del puerto 30 del switch principal Catalyst 4506-E enruta el tráfico hacia un router Mikrotik RouterBoard 1100 el mismo que se conecta con un switch 3Com 4500G, este switch tiene dos conexiones principales, es decir, una conexión hacia la infraestructura hiperconvergente Proxmox que se administra por medio de los servidores PV1, PV2 y PV3 como se muestra en la figura 14 y la otra conexión hacia un switch QPCom QP-1240R en el cual se conecta la red inalámbrica que posee la facultad, por lo tanto, se administra la comunicación de cada Access Point desplegado por medio de un servidor Redius.

En este sentido, la topología física de los tres segmentos del Data Center de la Facultad de Ciencias Aplicadas se muestra en la figura 14.

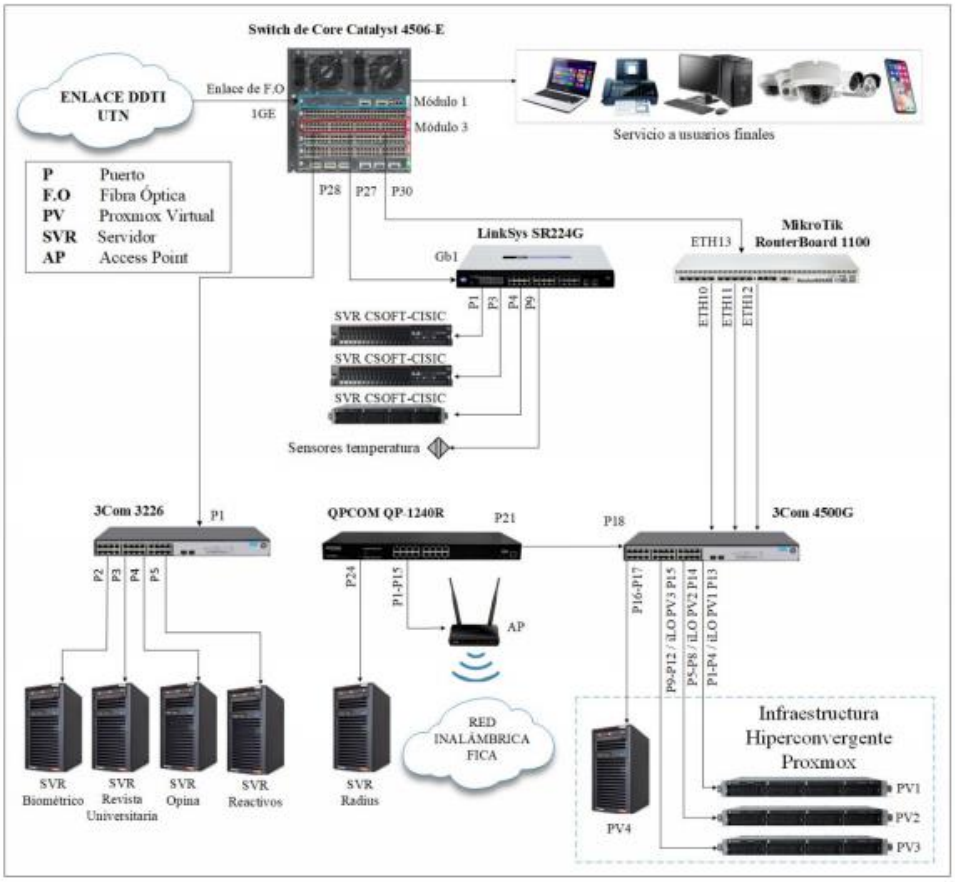

Figura 14. Diagrama topologia fisica.

Obtenido de: Nicolalde (2021)

Por otro lado, es necesario detallar la topología lógica y el esquema de segmentación virtual de la red, es decir, en el data center de la facultad su red principal física está segmentada mediante VLANs las cuales cumplen una función específica de crear redes lógicas independientes, a continuación, en la figura 15 se detalla esta segmentación lógica, además se muestra la distribución de cada VLAN en la topología lógica.

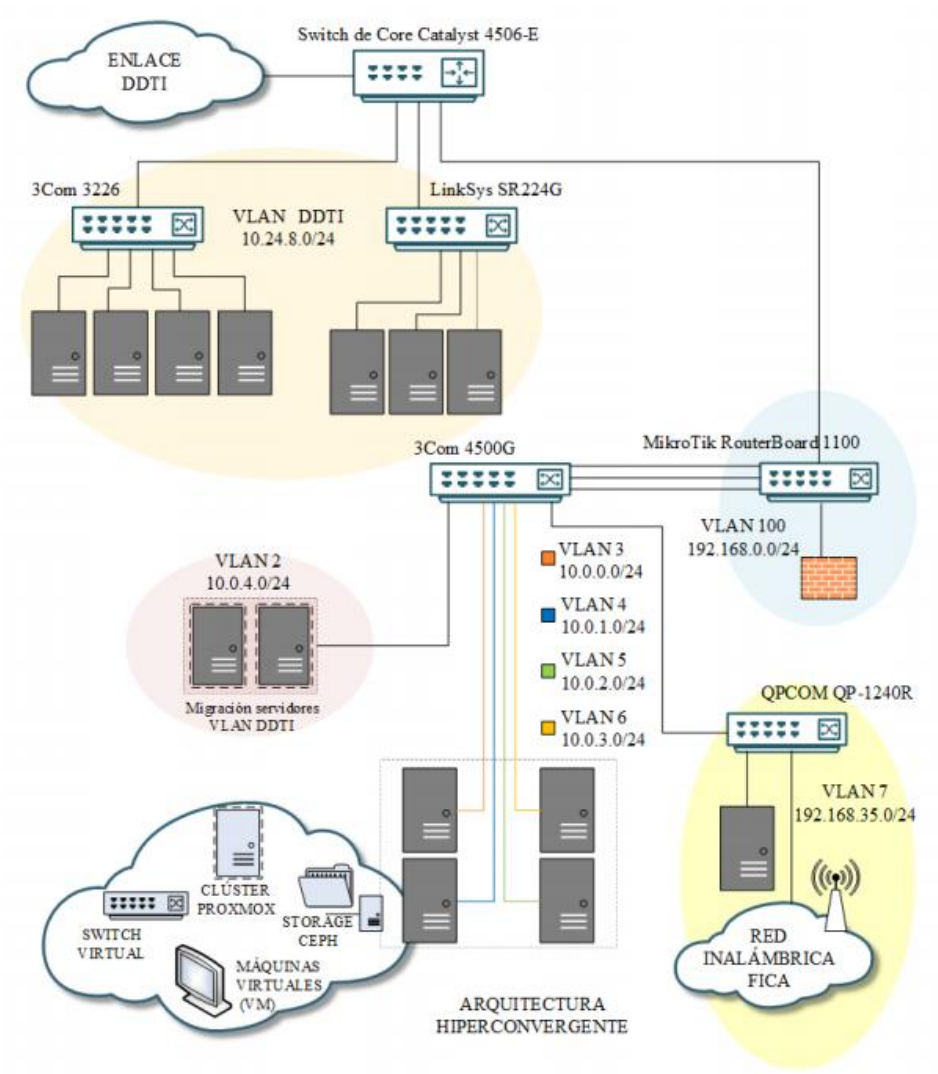

Figura 15. Segmentacion virtual de la red fisica del Data Center

Obtenido de: (Nicolalde, 2021)

En la figura 15 se observa que cada VLAN ocupa un segmento virtual dentro de la topología física del Data Center, por lo tanto, cada una de ellas posee un número ID y nombre para ser identificadas dentro de la topología virtual segmentada. De manera que, (1) la VLAN 2 se encarga de la migración de servidores DDTI y está identificada con el nombre ENLACE RED ANTIGUA; mientras que, (2) las VLANs 3,4 y 5 se encargan de segmentar los servidores Proxmox por ejemplo, la VLAN 3 está dirigida propiamente para la comunicación de los servidores físicos Proxmox, la VLAN 4 encargada de la administración remota de los servidores Proxmox y VLAN 5 que se encarga de los servidores virtualizados en Proxmox, con el nombre STORAGE CEHP; (3) la VLAN 6 se encarga de conectar todo el almacenamiento compartido entre los

servidores estándar; por el contrario (4) la VLAN 7 administra la red inalámbrica en la facultad y, (5) la VLAN 100 segmenta toda la red interna del Data Center la cual va ubicada en el Mikrotik.

Cabe recalcar que, para el presente proyecto es necesario comprender estas VLANs ya que brindan un soporte completo para el uso de la plataforma Proxmox. De modo que, en la tabla 1 se detalla el direccionamiento IPv4 implementado, el mismo que corresponde únicamente a la red interna del Data Center facilitando la comprensión de la segmentación lógica por VLANs en el Data Center.

Tabla 1. Direccionamiento IPv4

	VLAN	EQUIPO	SUBRED	IP	GATEWAY	MASCARA
3Com 226	VLAN DDTI	OPINA	10.24.8.0	10.24.8.x	10.24.8.254	255.255.255.0
		Revista Universitaria		10.24.8.x		
		Reactivos Biométricos		10.24.8.x		
Linksys		CISIC 1		10.24.8.x		
		CISIC 2		10.24.8.x		
		CISIC 3		10.24.8.x		
		Sensores		10.24.8.x		
	2	Migración Servidores VLAN DDTI	10.0.4.0	10.4.x	10.0.4.254	
3Com 4500G	3	PV1	10.0.0.0	10.0.0.1	10.0.0.254	
		PV2		10.0.0.2		
		PV3		10.0.0.3		
	4	iLO PV1	10.0.1.0	10.0.1.1	10.0.1.254	
		iLO PV2		10.0.1.2		
		iLO PV3		10.0.1.3		
	5	Máq. Virtuales (VM)	10.0.2.0	10.0.2.x	10.0.2.254	
	6	CEHP PV1	10.0.3.0	10.0.3.1	10.0.3.254	
		CEHP PV2		10.0.3.2		
		CEHP PV3		10.0.3.3		
		CEHP PV4		10.0.3.4		
	7	QPcom	192.168.35.0	192.168.35.x	192.168.35.1	
		Radium		192.168.35.x		
		AP’s		192.168.35.x		
	100	MikroTik	192.168.0.0	192.168.0.x	192.168.0.254	

Modificado de Nicolalde, (2021).

3.1.2 ETAPA 2: Análisis de la Red Definida por Software (SDN) desplegada por plataformas Proxmox VE y OpenDaylight.

En la segunda etapa se analiza la Red Definida por Software (SDN) desplegada y documentado por Nicolalde (2021), en su Diseño de un Sistema de Detección de Intrusos (IDS), basado en redes neuronales para una Red Definida por Software (SDN). En el presente proyecto se hace un estudio de esta SDN y sus funciones para lograr implementar una topología entre las plataformas Proxmox VE, OpenDaylight y Open Source Mano para crear un entorno de red controlado y desplegar al menos una función de red virtualizada.

La SDN diseñada por Nicolalde (2021) e implementada en la Facultad de Ingeniería en Ciencias Aplicadas (FICA) en los servidores Proxmox VE (PV1, PV2, PV3) cumple las funciones de infraestructura hiperconvergente virtual, compuesta por los planos: (1) Datos, (2) Control y (3) Aplicación, detallados en la sección 2.1. Además, para el despliegue de esta SDN se utilizó el controlador OpenDaylight, el cual cumplió con todos los parámetros requeridos para la instanciación de una red definida por software como son: Protocolo OpenFlow, Código abierto, API REST e Interfaz Web, detallados en la sección 2.1.2.

Por otro lado, para que la infraestructura física del Data Center soporte el protocolo OpenFlow y que todo el tráfico atraviese por la SDN, Nicolalde (2021), despliega el software Open Virtual Switch (OVS) en una máquina virtual, el cual cumple la función de switch físico y virtual que administra y gestiona los flujos y demandas requeridas por el protocolo OpenFlow. Cabe recalcar que, la plataforma OpenDaylight y el software OVS deben conectarse para que los paquetes OpenFlow atraviesen por toda la red de datos.

Por lo tanto, para entender mejor la Red Definida por Software implementada en el servidor Proxmox PV2 se detalla a continuación la tabla 2, en la cual se describe cada plano SDN, con sus respectivas plataformas, características de software, módulos de controlador e interfaces que comunican a cada plano en la infraestructura hiperconvergente virtual.

Tabla 2. Estructura y componentes SDN Facultad de Ingeniería en Ciencias Aplicadas (FICA)Nicolalde (2021).

PLANO	PLATAFORMA	PARÁMETROS DE SOPORTE	CARACTERÍSTICAS
PLANO DE DATOS	Open Virtual Switch (OVS)	Centos 7	Versión: 7 de 64 bits Hardware: 100 GB HDD y 4 GB RAM
		Interfaz virtual ovs-br0	Interfaz que se conecta al controlador ODL
		Interfaz virtual eth 0	Interfaz que se conecta a la red física
		Interfaz virtual eth 1	Interfaz bridge entre ovs-br y eth 0
PLANO DE CONTROL	Controlador OpenDaylight (ODL)	Windows	Versión: 10 Profesional de 64 bits Hardware: 100 GB HDD y 8 GB RAM
		Interfaz virtual eth	Interfaz que conecta con el OVS
PLANO DE APLICACION	OpenDaylight (ODL)	DLUX	Proporciona una interfaz gráfica de usuario intuitiva para OpenDaylight.
		Switch L2	Proporciona reenvío L2 (Ethernet) a través de conmutadores OpenFlow conectados y soporte para seguimiento de host.
		RESTCONF API	Permite el acceso de la API REST a MD-SAL, incluido el almacén de datos.
		Model-Driven SAL(MD-SAL)	Conjunto de servicios de infraestructura destinados a proporcionar soporte común y genérico
		OpenFlow Plugin	Interfaz de comunicaciones estándar para permitir la interacción entre las capas de una arquitectura SDN

En consecuencia, al comprender y analizar la tabla 2 se concluye que, la Red Definida por Software diseñada por Nicolalde (2021), es óptima y cumple con todos los requerimientos de infraestructura hiperconvergente virtual para el estudio de la tecnología NFV y la virtualización de al menos una función de red es decir, gracias a que el servidor Proxmox VE (PV2) apalanca la SDN y cumple con los recursos de hardware y software necesarios para la ejecución de las funciones de red virtualizadas, mencionados en la sección 2.2.1, el PV2 es el candidato ideal para comportarse como una NFVI.

Además, con la plataforma OpenDaylight como controlador que actúa como VIM es posible (1) gestionar los recursos NFVI obtenidos del servidor Proxmox PV2, (2) controlar el inventario de máquinas virtuales que se asocian a los recursos virtualizados ya desplegados y, (3) administrar la gestión, rendimiento y fallos en el hardware, software y recursos virtuales ya desarrollados. De manera que, se cumple con el primer bloque de la arquitectura MANO mencionada en la sección 2.2.1. Los otros bloques se desarrollarán con la ayuda de la plataforma OpenSource MANO la cual despliega y detalla en la siguiente etapa.

3.1.3 ETAPA 3: Análisis y despliegue de la plataforma OpenSource MANO (OSM)

En la tercera etapa, con la topología SDN desplegada en Nicolalde (2021), se procede a realizar la instalación de la plataforma principal para este proyecto, es decir OSM, que cuenta con varias versiones de software, siendo la versión TWELVE la más actual y la que se ha decidido desplegar para este proyecto, con base en:

- La versión TWELVE de acuerdo con (ETSI, 2020) trae una serie de mejoras con respecto a las versiones anteriores, como por ejemplo, los requisitos de instalación en versiones anteriores eran muy elevados, sin embargo, para esta versión, se redujo significativamente la funcionalidad al momento de instalar la plataforma, es decir con una máquina virtual que cumpla con las especificaciones recomendadas que se detallan en la tabla 3 es posible el despliegue de OSM de manera sencilla.
-

A continuación, la tabla 3 describe las características que debe soportar una máquina virtual, para instalar OSM versión TWELVE, por otro lado, también se ha establecido una comparación con los requisitos de las versiones anteriores, de acuerdo con la información proporcionada por (ETSI, 2020).

Tabla 3. Requisitos para instalación de plataforma OSM

REQUISITOS	MINIMO	MAXIMO	VERSIONES ANTERIORES
CPUs	2CPUs	2CPUs	Para las versiones 2 hasta 5, se necesita

			8CPUs
MEMORIA RAM	6 GB de RAM	8 GB de RAM	Para versiones 7 y 8, 16 GB de RAM es recomendado
ALMACENAMIENTO	40 GB de disco	40 GB de disco	80 GB de disco son recomendados para versiones desde 2 hasta 6
ISO	Ubuntu 18.04 (variante de 64-bits)		Para versiones inferiores Ubuntu 16.04
INTERFAZ DE RED	Que tenga acceso a internet, para descargar varios GB de datos en repositorios externos		

Otro de los aspectos que han sido mejorados en la versión TWELVE es el procedimiento de instalación de OSM, es decir, anteriormente era necesario hacer varias configuraciones previas para preparar la máquina virtual y comenzar con la instalación de la plataforma; esas configuraciones fueron remplazadas por comandos que automatizan y facilitan la instalación de la plataforma, debido a que, contiene cientos de sentencias que se van ejecutando automáticamente y evita al administrador realizarlo de manera manual.

A continuación, la figura 16 muestra los comandos para instalar y ejecutar la plataforma OSM, que también evitan la configuración del servicio Linux container hypervisor (LXD), el cual se menciona en (Barrado, 2018) es un hipervisor de contenedores donde se implementa la arquitectura OSM. Por otro lado, con la ejecución de los comandos se instala de manera automática el paquete ZFS, que se utiliza como backend de almacenamiento LXD.

```
wget https://osm-download.etsi.org/ftp/osm-10.0-ten/install_osm.sh
chmod +x install_osm.sh
./install_osm.sh
```

Figura 16. Comandos para la instalacion de OSM.
Obtenido de: (ETSI OSM, 2020)

Por consiguiente, en la figura 17 se muestra la primera línea de sentencia denominada ***wget https://osm-download.etsi.org/ftp/osm-10.0-ten/install_osm.sh*** que descarga el scritp de instalación de OSM que contiene cientos de comandos los cuales se ejecutan automáticamente.

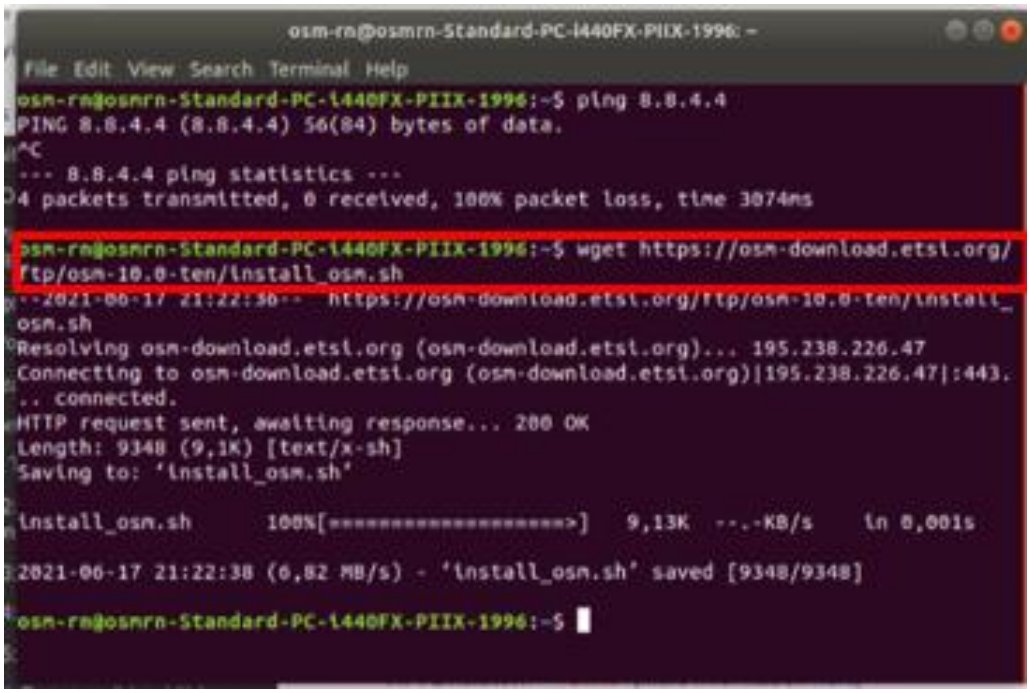

Figura 17. Comando 1 wget. Descarga de script de instalacion OSM

Por otro lado, en la figura 18 se muestra la segunda sentencia denominada ***chmod +x install_osm.sh*** que brinda permisos de ejecución al script en la máquina virtual.

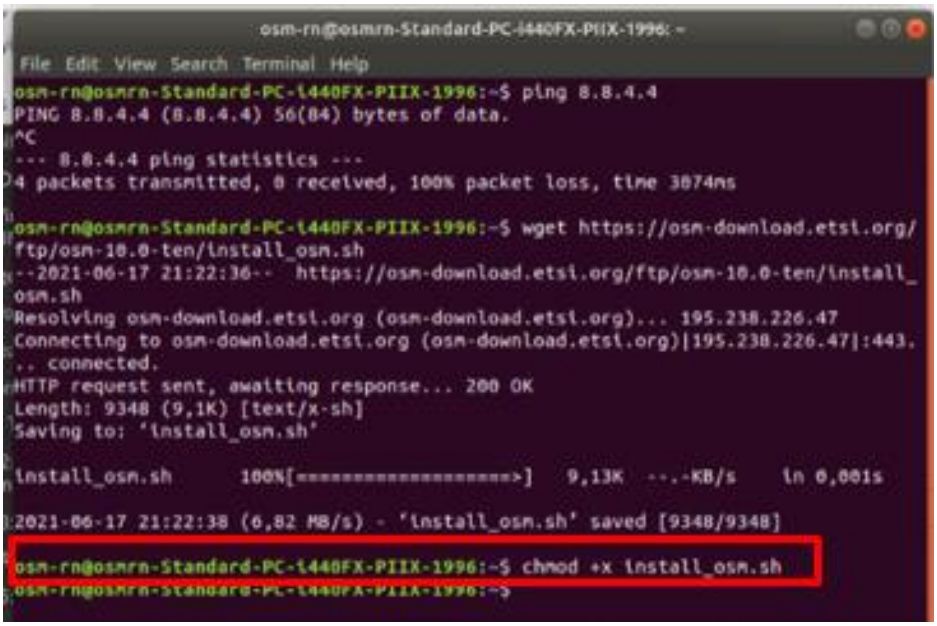

Figura 18. Comando 2 chmod. Permisos de ejecucion del script.

Y, por último, en la figura 19 se muestra el comando para ejecutar el script de instalación, se usa la sentencia ***./install_osm.sh*** además, pide una confirmación y configuración de LXD, juju[7], Docker CE y la inicialización de un contenedor Docker local, los cuales son requisitos previos mencionados en la sección 2.3. a los cuales se responde con “YES (Y)”.

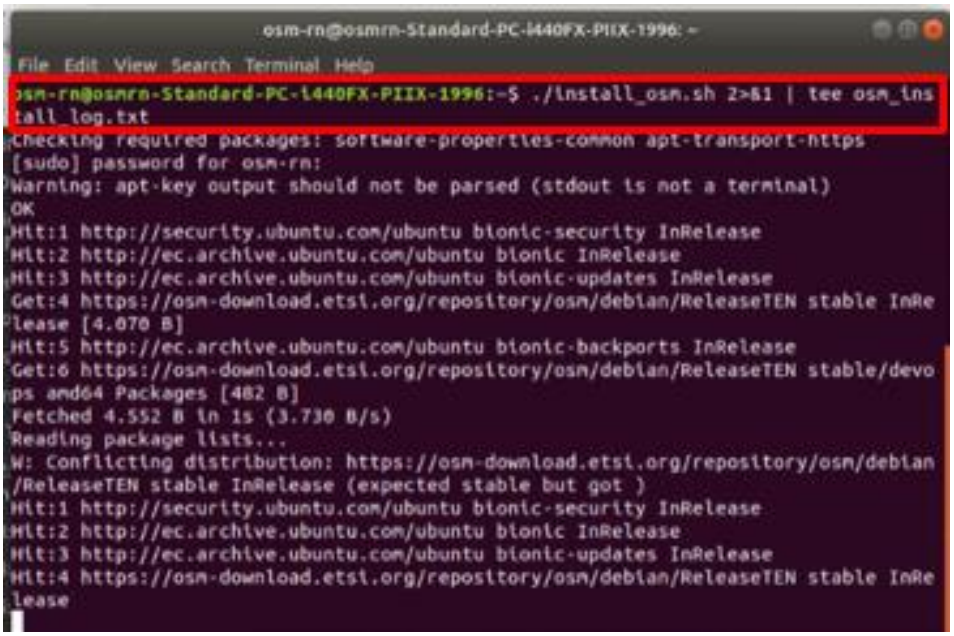

Figura 19. Comando 3 ./install_osm.sh. Ejecuta el script de instalacion

Es así como, la instalación automática incluye también el cliente de OSM el cual se encarga de gestionar los descriptores Docker, también gestiona la creación de NS (servicios de red) y el completo ciclo de vida de las VIMs creadas en una máquina virtual como host, por lo tanto, este cliente de OSM está basado en lenguaje de programación Python y para comprobar que se ha instalado correctamente es necesario acceder a la interfaz gráfica del cliente, traduciendo en el navegador la dirección IP que se haya configurado, como se muestra en la figura 20, además para acceder al login de la interfaz el usuario y contraseña por defecto son “admin”.

[7] Juju es un controlador que garantiza la alta disponibilidad (HA) de las aplicaciones implementadas, el controlador debe tener una alta disponibilidad. Esto requiere la creación de controladores adicionales, todos los cuales residen naturalmente dentro del modelo de "controlador". El controlador inicial se conoce como el maestro

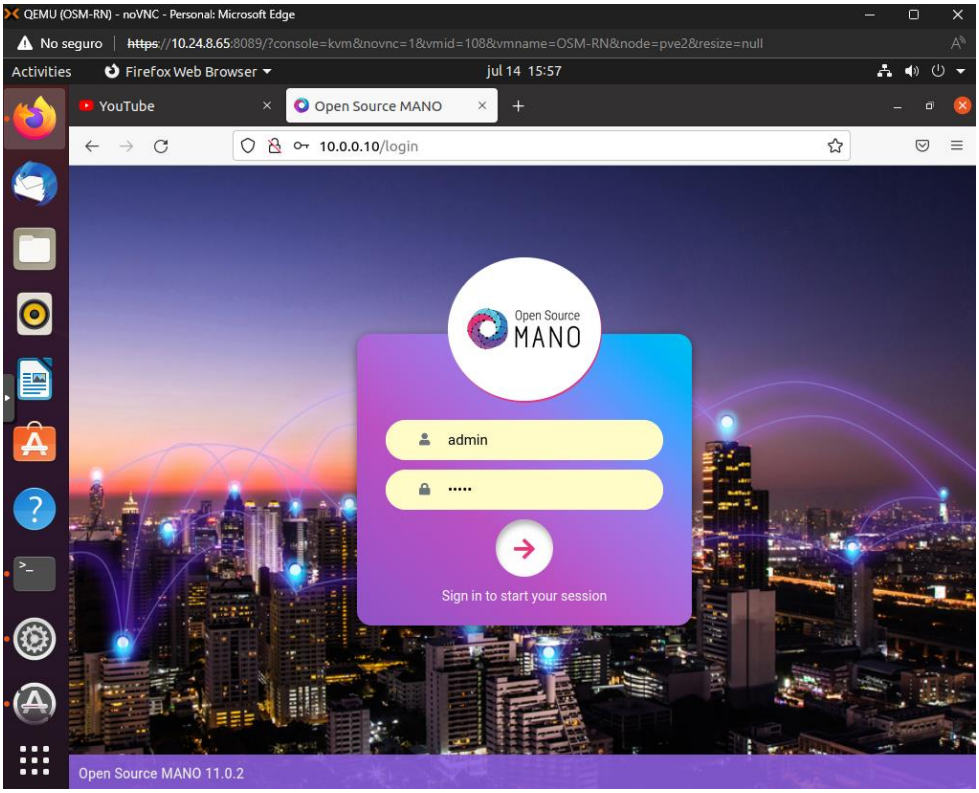

Figura 20. Interfaz gráfica de OSM.

3.1.4 ETAPA 4: Topología y conectividad entre plataformas OpenSource MANO (OSM), OpenDaylight y Proxmox VE.

Por último, en la cuarta etapa se crea una topología de red con las plataformas ya desplegadas: OSM, OpenDayligh y Proxmox VE en este procedimiento se verifica que exista conectividad y funcionalidad en estas plataformas, es así como, la integración entre las mismas pueda desplegar un testbed para la virtualización de funciones de red; de esta manera el despliegue de una infraestructura integrada en un banco de pruebas tiene como objetivo habilitar, nutrir y respaldar los diferentes conceptos utilizados al momento de desarrollar servicios, aplicaciones o crear un middleware de control de última generación basado en tecnología NFV. Es así como, para el desarrollo de la topología de red se realiza una esquematización en diagrama de bloques de cómo están diseñadas e implementadas las arquitecturas SDN y NFV en el servidor Proxmox VE, en el cual se detalla el diseño propuesto de una infraestructura hiperconvergente, donde se muestra las máquinas virtuales creadas, conexiones entre interfaces virtuales y agrupación entre tecnologías, las cuales se van desplegando en cada plano SDN y capa de la arquitectura NFV.

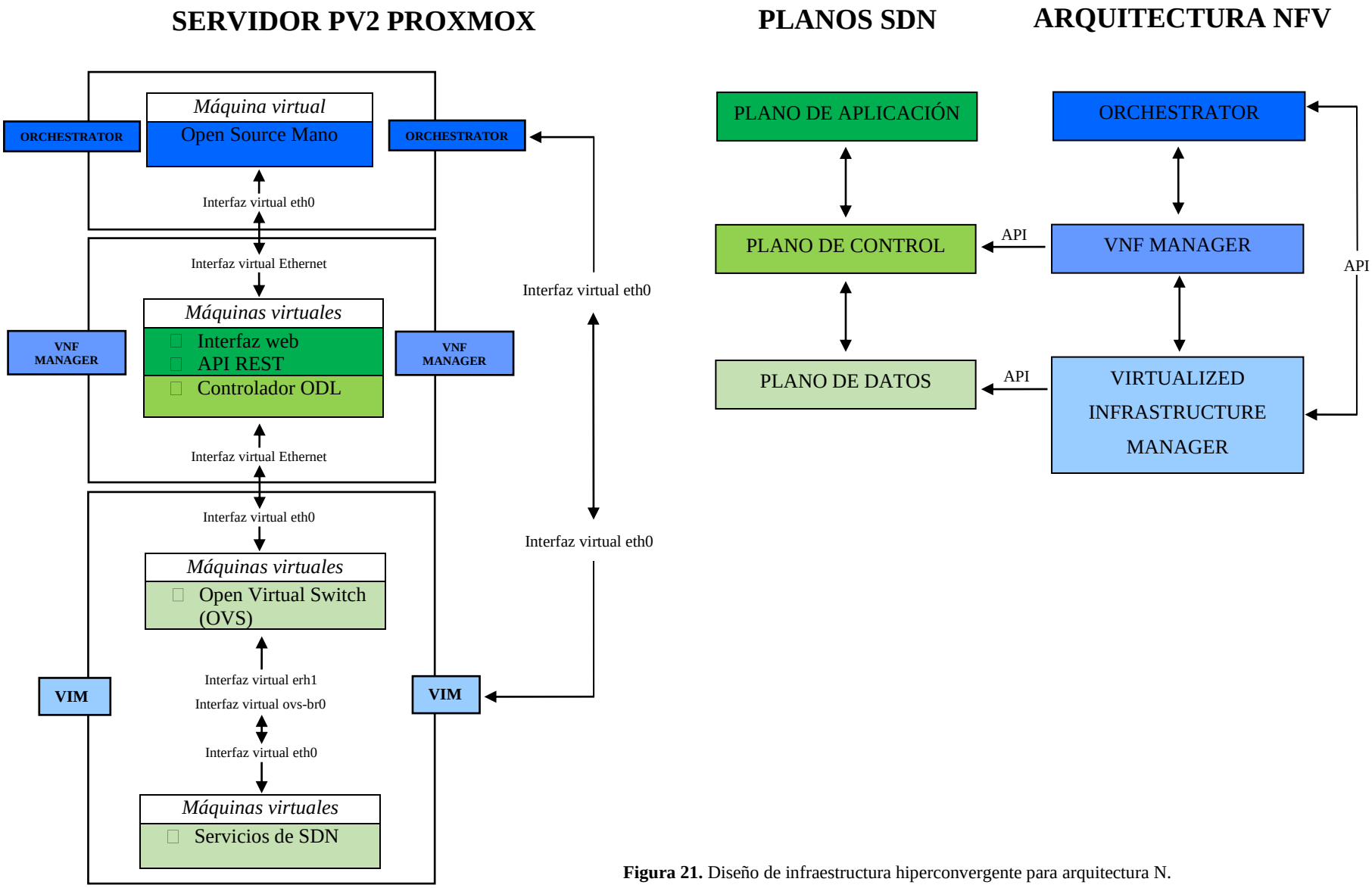

Figura 21. Diseño de infraestructura hiperconvergente para arquitectura N.

En la figura 21 se muestra el diagrama de bloques de la infraestructura hiperconvergente diseñada para soportar una arquitectura NFV, es decir, que en la interacción entre máquinas virtuales y servidores implementados en la SDN se garantice un ambiente de red controlado para diseñar una arquitectura NFV y posteriormente virtualizar al menos una función de red, por lo tanto, en la figura 21 se muestra cómo cada plano SDN forma parte de un bloque en la arquitectura NFV para que esto suceda es necesario que el software Open Virtual Switch (OVS) desplegado en una máquina virtual cumpla una función de switch físico y gestione los recursos NFVI, es decir, gestionar los recursos de creación, mantenimiento y errores asociados al hardware, software y a los recursos virtuales mediante el protocolo OpenFlow que usa ODL, ya que, este protocolo garantiza la comunicación entre servidores creados en la SDN y usuarios finales compatibles gracias al OVS desplegado en el bloque VIM. Además, el ODL es el encargado de gestionar las VNFs, por ejemplo, se encarga de crear, mantener y terminar instancias y máquinas virtuales dando un rendimiento óptimo y seguridad a las VNFs.

Finalmente, para completar el funcionamiento de una arquitectura NFV se usa el controlador OSM como orquestador el cual se encarga de gestionar las funciones de red para posteriormente virtualizarlas creando un extremo a extremo entre diferentes VNFs desplegadas por el controlador ODL y valida las solicitudes de hardware y software que se obtienen del OVS ubicado en el bloque VIM, es así como se obtiene un testbed funcional para virtualizar al menos una función de red. Para cumplir con el escenario antes mencionado, es necesario que tengan conectividad las distintas plataformas, de modo que debe existir un direccionamiento IPv4, el cual se detalla a continuación en la Tabla 4.

Tabla 4. Direccionamiento IPv4 Testbed.

PLATAFORMA	IP	MASCARA DE RED	PUERTA DE ENLACE
ODL	192.168.100.111	255.255.255.0	192.168.100.110
OSM	192.168.100.114	255.255.255.0	192.168.100.110
SERVIDOR 1	192.168.150.11	255.255.255.0	192.168.150.10
SERVIDOR 2	192.168.150.12	255.255.255.0	192.168.150.10
OVS	192.168.100.110	255.255.255.0	
	192.168.150.10	255.255.255.0	192.168.150.10

En la figura 22 se muestra la topología lógica implementada sobre la plataforma ProxmoxVE, en la cual se detalla la conectividad de cada dispositivo que compone un escenario de pruebas (Testbed) para tecnología NFV, por lo tanto, se observa la conexión entre el controlador ODL con el OVS, así como también la plataforma OSM con el OVS, dentro de una misma red como **Infraestructura de Red Virtual (NFVI)**. Por otro lado, se conectan los servidores con el OVS que van a ser utilizados como **Funciones de Red Virtualizados (VNF)**, que se despliegan en una misma red.

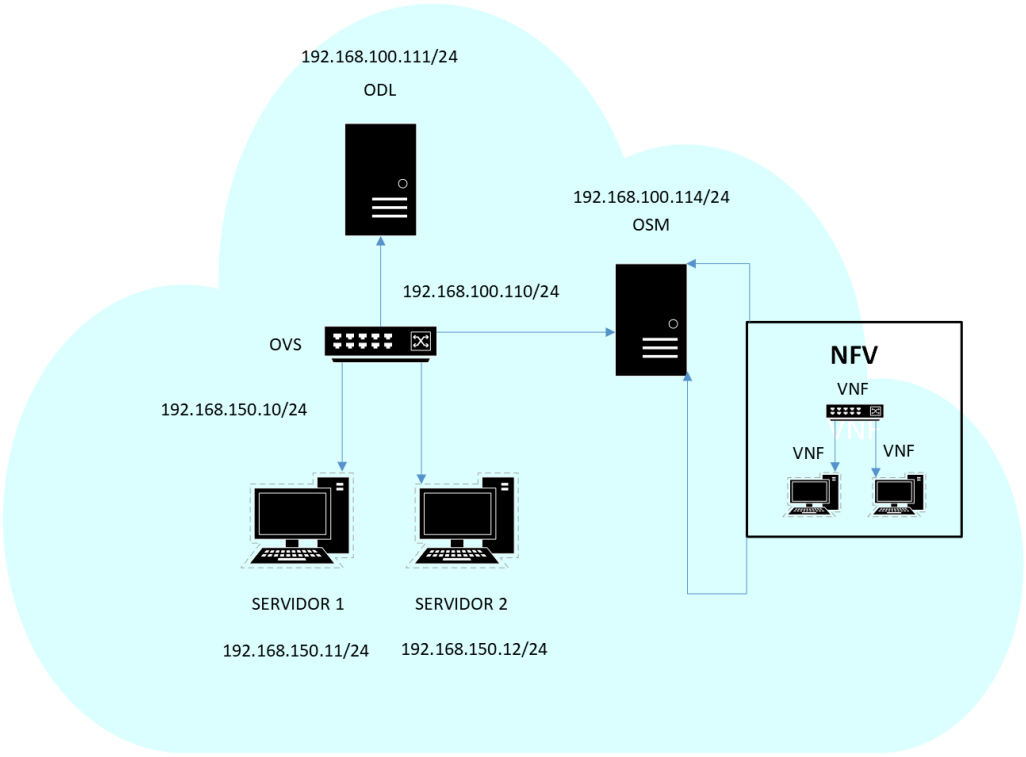

Figura 22. Topología lógica del Testbed para NFV

Con la plataforma OSM ya implementada en la red SDN, se procede a verificar la correcta conectividad entre los distintos escenarios, por lo tanto, se realiza un ping entre el OVS y la plataforma Open Source Mano, resultado que se observa en la figura 23. Es decir, se muestra el proceso de envío de paquetes ICMP entre las dos plataformas, obteniendo un resultado exitoso.

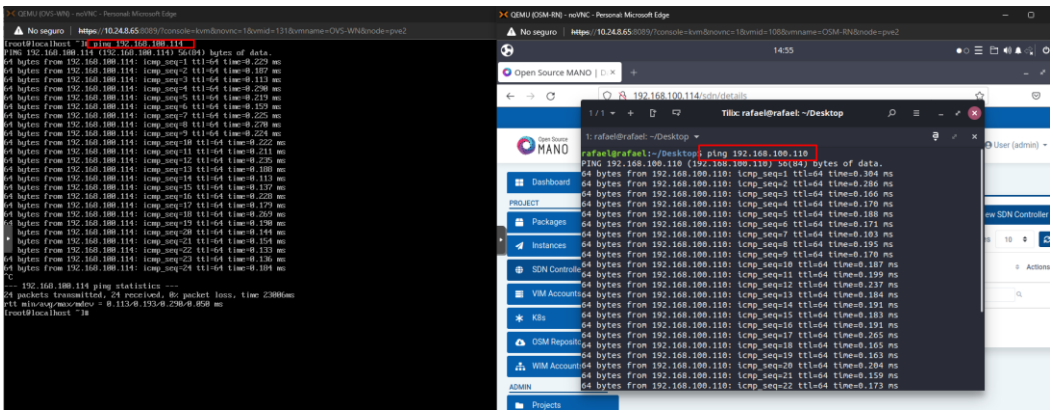

Figura 23. Prueba de conectividad entre OSM y OVS

Después que ya existe conectividad entre las diferentes plataformas, se procede a desplegar y configurar la SDN, que sirve como bloque VIM para nuestra arquitectura NFV donde se va a integrar la máquina virtual que contiene el orquestador OSM.

A continuación, se detalla el procedimiento para la correcta instalación del controlador ODL, por lo cual, es necesario tener instalado **(JRE) Java Runtine Environment,** este requerimiento se lo puede obtener directamente de la página oficial de JAVA por medio del siguiente enlace *https://www.java.com/es/download/*.

De la misma manera, es necesario descargar el software del controlador ODL, este proceso se muestra en la figura 24, la cual indica qué formato y versión es necesario, para este proyecto se escoge la distribution-karaf-0.4.4-Beryllium-SR4 compatible con el OVS, por lo tanto, se lo obtiene del siguiente enlace *https://bit.ly/2Na9gB2*.

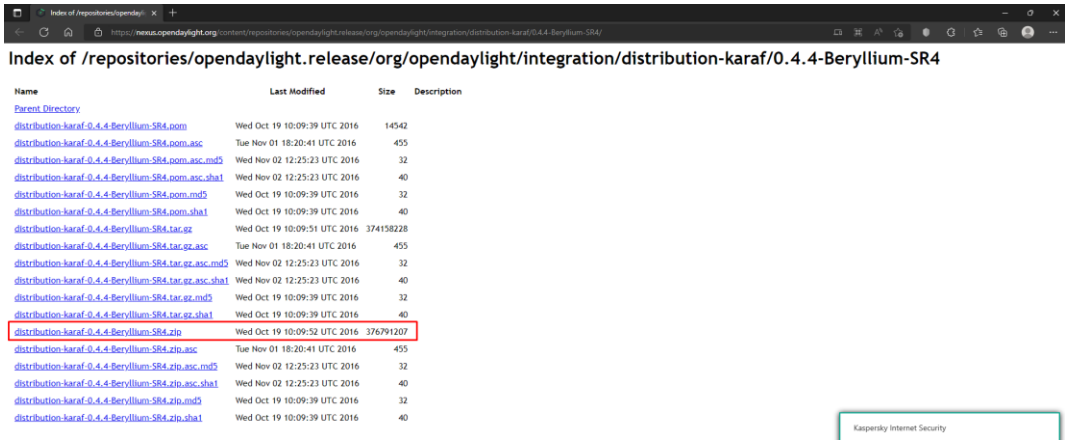

Figura 24. Distribución .karaf para controlador ODL

Por otra parte, la figura 25 muestra el archivo **distribution-karaf-0.4.4-Beryllium-SR4.zip** ya descargado, es importante destacar que este tipo de distribución es propio para el sistema operativo Windows 10.

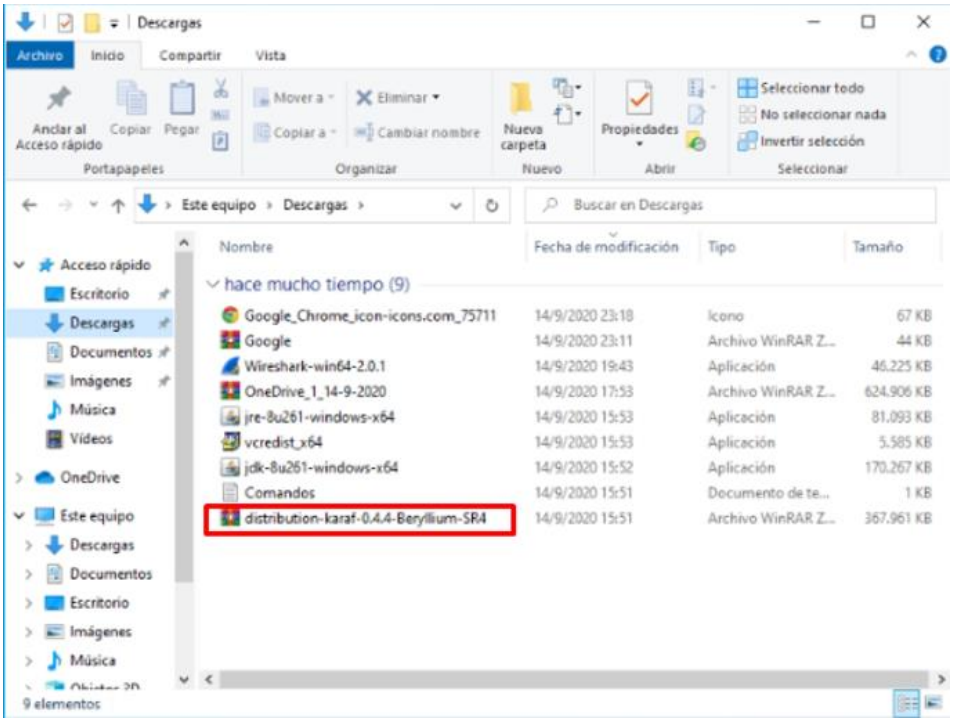

Figura 25. Archivo . karaf para Windows

Por lo tanto, después de descomprimir el archivo, es necesario realizar la configuración que se muestra en la figura 26, es decir, abrir la carpeta **etc,** en donde se encuentra el archivo **system.properties** al cual vamos a agregar la siguiente línea de comando ***org.osgi.framework.os.name = Win32*** para que el controlador ODL funcione en Windows 10.

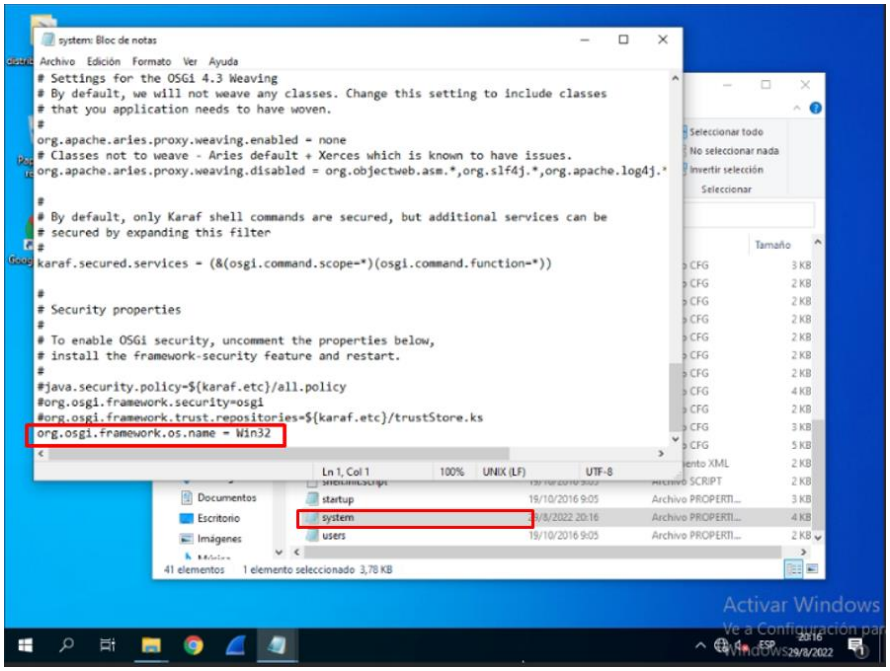

Figura 26. Configuración del archivo system.properties.

A continuación, se realiza la última configuración para poder utilizar el controlador ODL, se necesita por tanto instalar los paquetes **Features** mediante consola, para lo cual, es necesario ingresar como administrador al archivo consola **karaf.bin** que se encuentra en la carpeta **bin**, después se agrega la siguiente sentencia en comando:

feature:install odl-restconf odl-mdsal-apidocs odl-dlux-all odl-dlux-node odl-dlux-core odl-dlux-yangui odl-l2switch-switch odl-openflowplugin-all

El resultado se muestra a continuación en la figura 27.

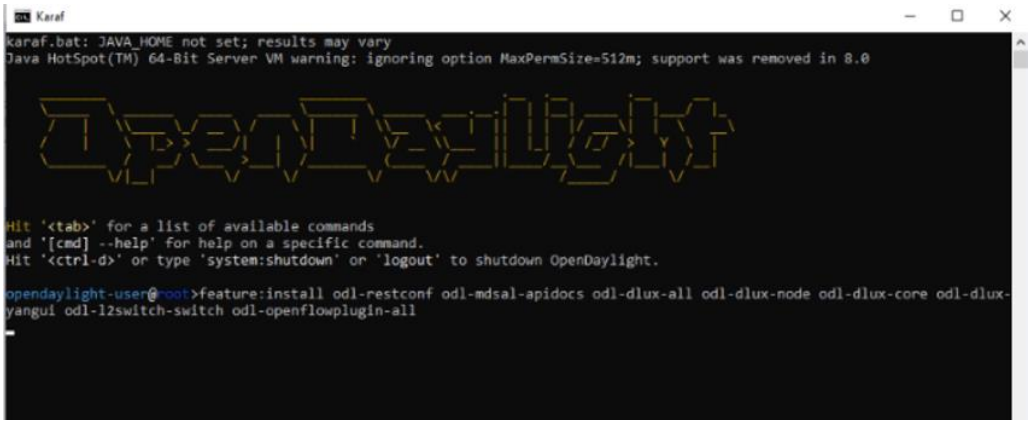

Figura 27. Instalación de comandos Feature

Por último, para acceder a la interfaz web de OpenDayLight, es necesario que en el navegador se escriba la siguiente dirección *http://localhost:8080/index.html* la cual permite, mediante una dirección ip y un puerto, visualizar la interfaz gráfica del controlador como se muestra en la figura 28.

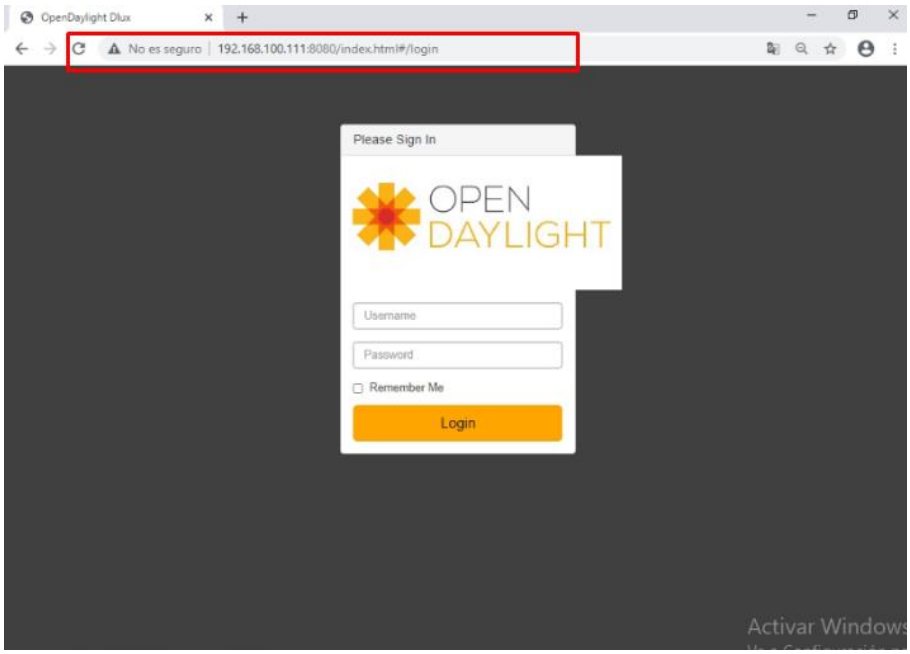

Figura 28. Interfaz web de ODL

Cabe recalcar que para ingresar al sistema de control de ODL, tanto como para el usuario y contraseña las credenciales por defecto son **admin.**

A continuación, se detalla el proceso de como habilitar el Open vSwitch para que mediante el protocolo openflow detecte los diferentes dispositivos conectados al ambiente de pruebas SDN y se comuniquen con el controlador ODL, por lo cual se procede a descargar e instalar el openvswitch con el siguiente comando:

yum install -y https://repos.fedorapeople.org/repos/openstack/EOL/openstack-juno/epel-7/openvswitch-2.3.1-2.el7.x86_64.rpm

Después que se haya descargado correctamente el OpenvSwitch (OVS), se debe crear y configurar una interfaz que permita la conexión entre el OVS y el controlador ODL. Es decir, la creación de una interfaz virtual la cual funciona como puente (bridge) entre las diferentes plataformas; esta ayuda adherir varias interfaces (APIs) tanto físicas como virtuales. Este proceso se muestra en la figura 29, que presenta el siguiente comando ***ovs-vsctl add-br ovs-br0*** el cual crea la interfaz virtual dentro del OVS.

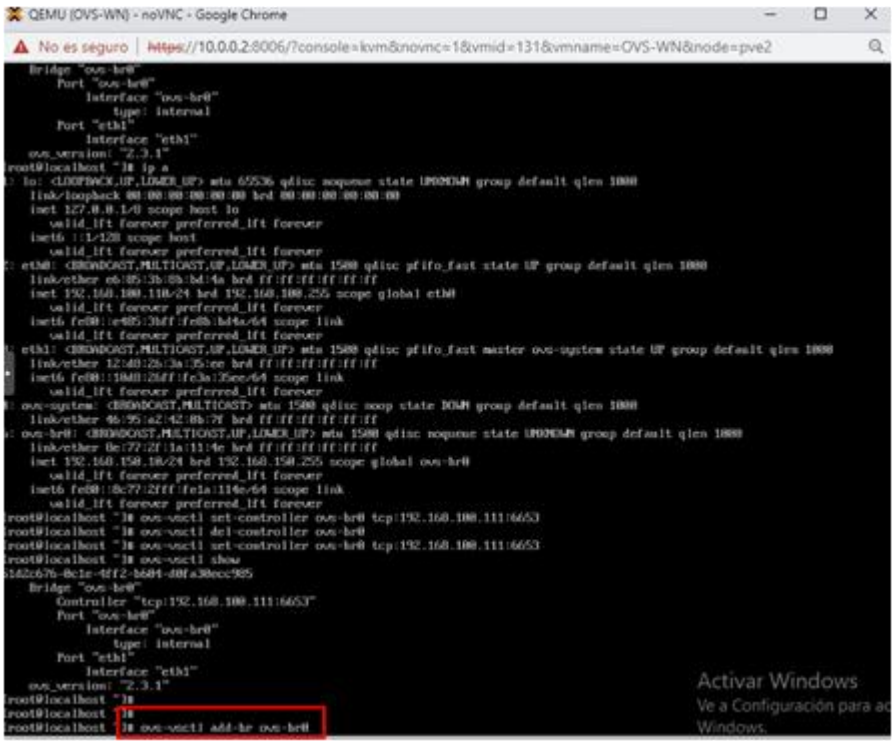

Figura 29. Creación de interfaz virtual en OVS.

Para la configuración de la interfaz virtual **ovs-br0** es necesario que contenga una interfaz física que opere como bridge, por lo tanto, para este proyecto será la interfaz eth1.

Primero se va a modificar el archivo de configuración de interfaz de eth1, el cual, se puede encontrar al escribir el siguiente comando ***/etc/sysconfig/network-scripts/ifcfg-eth1*** es aquí donde, se va a definir los parámetros mostrados en la figura 30.

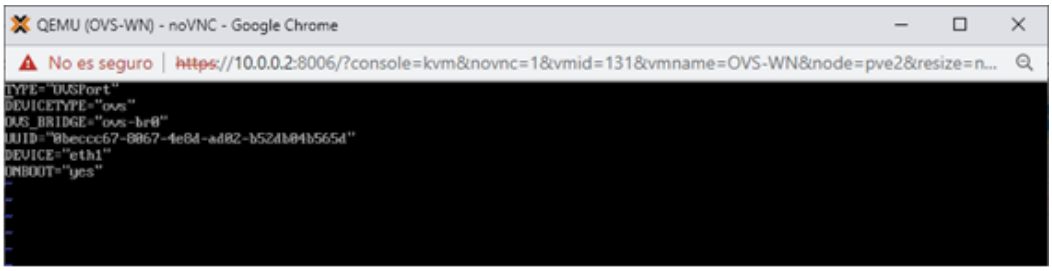

Figura 30. Configuración de la interfaz eth1.

Después, se modifica el archivo de la interfaz virtual ovs-br0 el cual se encuentra en ***/etc/sysconfig/network-scripts/ifcfg-ovs-br0*** donde se escribe las sentencias que se muestran en la figura 31.

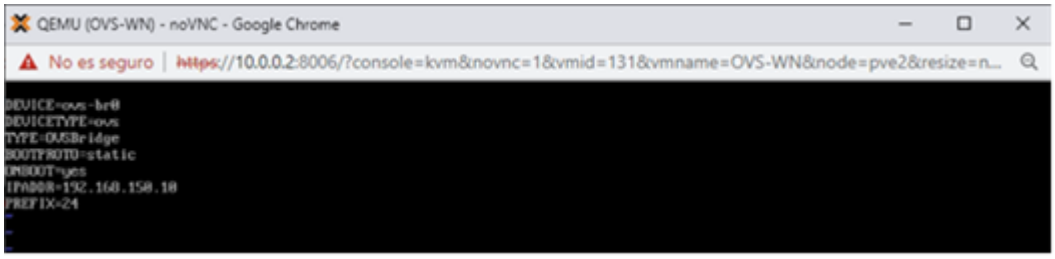

Figura 31. Configuración de la interfaz ovs-br0

Por último, para realizar la conexión del OVS con el controlador ODL, se ejecuta la siguiente sentencia en modo root ***ovs-vsctl set-controller ovs-br0 tcp:192.168.100.111:6653***. Es decir, mediante esta sentencia opera el controlador por la interfaz ovs-br0, que identifica al controlador por medio del protocolo (tcp), la ip (192.168.100.111) y el puerto de comunicación (6653) estableciendo una

conexión virtual entre estas plataformas, este procedimiento se muestra en la figura 32.

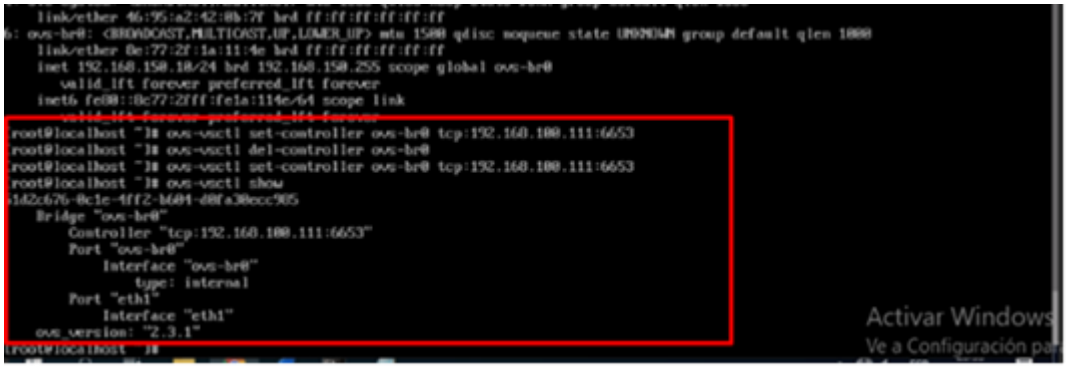

Figura 32. Conexión entre ODL y OVS

Por otra parte, se puede observar el escenario SDN en la interfaz web que dispone el controlador ODL, es decir, después de realizar los pasos mencionados antes correctamente, se puede verificar una topología de prueba, indicando cada dispositivo desplegado; esto se muestra en la figura 33.

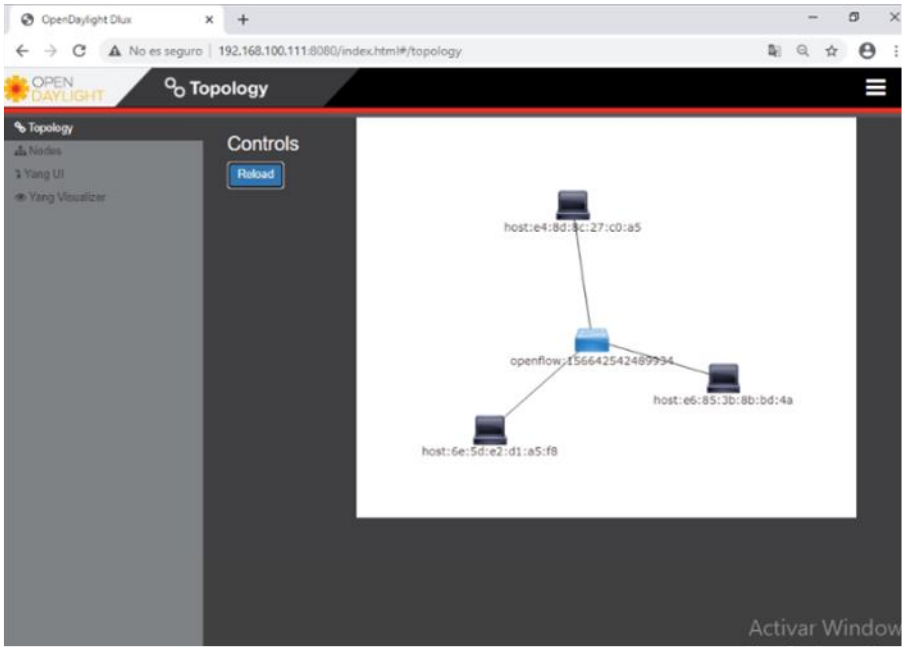

Figura 33. Topología ilustrada en el controlador ODL

Ahora, para entender cómo se implementa la topología Testbed para NFV, es importante mostrar una topología lógica partiendo de la red de segmentación del Data Center de la FICA, el cual fue mostrado en la ETAPA1 (Figura 15). Por lo

tanto, en la figura 34 se observa cada elemento tanto físico como virtual desplegado en la red del Data Center indicando especialmente en donde se aloja la topología Testbed para NFV.

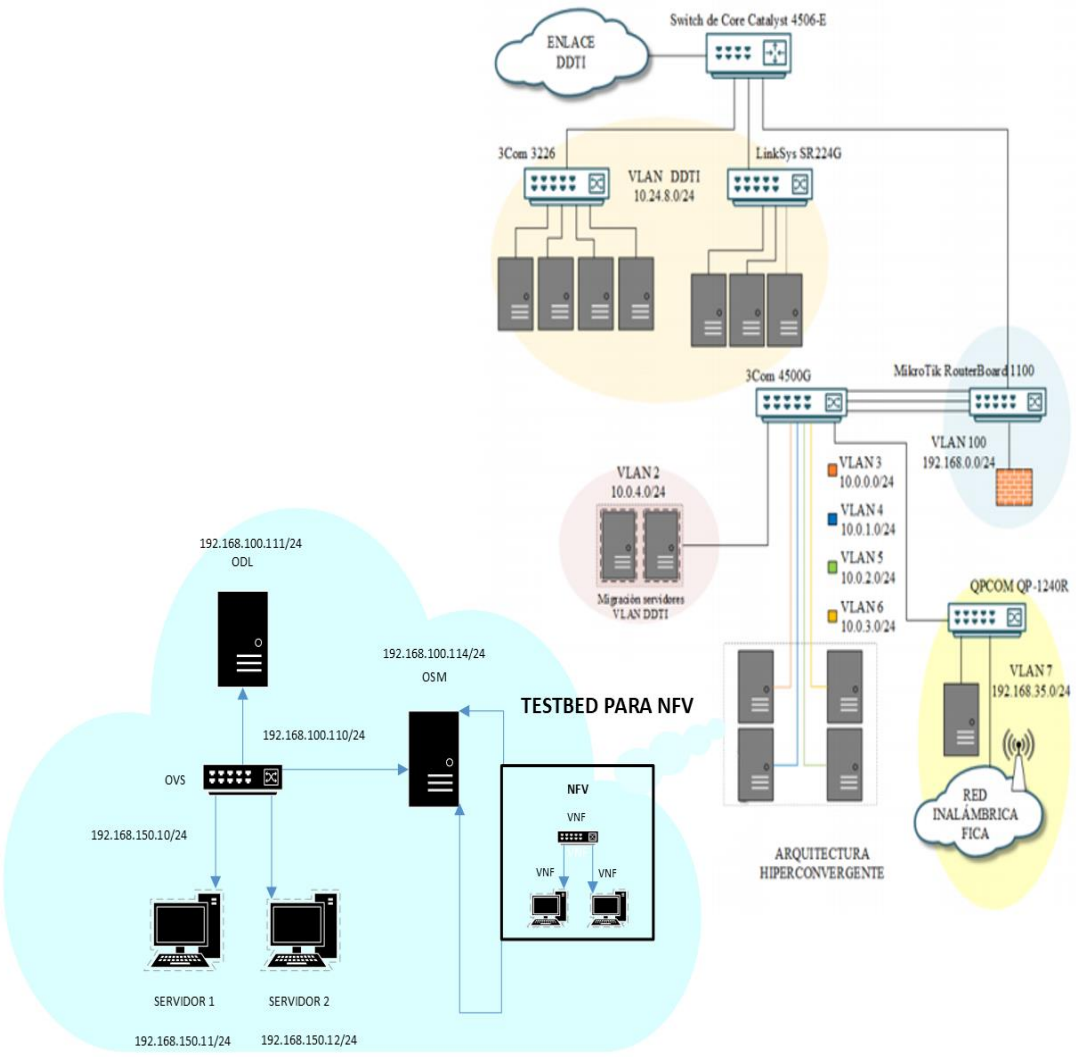

Figura 34. Topología Testbed para NFV

4. CAPITULO IV
VIRTUALIZACION DE LAS FUNCIONES DE RED NFV

Este capítulo describe el procedimiento de la virtualización de al menos un servicio de red, con el objetivo de demostrar el funcionamiento de la tecnología NFV, mediante el despliegue de un Network Service, que interactúa con las diferentes VNFs desplegadas. Esto se lo lleva a cabo mediante el sistema operativo base *CirrOS* el cual enlaza el NS con VNF mediante un Virtual Link Descriptor (VLD).

4.1. Conexión entre OSM y el bloque VIM

A continuación, se detalla el proceso de conexión entre el bloque VIM y la plataforma OSM, en efecto, como menciona la sección 2.2.1 (Arquitectura NFV), es necesario que OSM interactúe con el bloque VIM, ya que este bloque es el encargado de crear, administrar y eliminar VNFs.

Por lo tanto, la plataforma **DevStack**, que se utiliza interactivamente como un entorno de desarrollo es la apropiada para realizar operaciones de VIM en la topología Testbed para NFV. Posteriormente se detalla su instalación y configuración.

La tabla 5 describe las características que debe soportar una máquina virtual, para instalar DevStack, dentro del entorno hiperconvergente Proxmox VE.

Tabla 5. Requisitos para instalación de plataforma DevStack, los cuales se pueden obtener de la página oficial *https://docs.openstack.org/devstack/latest/*

REQUISITOS	MINIMO	MAXIMO
CPUs	**2CPUs (1 sockets, 2 cores)**	**2CPUs (2 sockets, 2 cores)**
MEMORIA RAM	**6 GB de RAM**	**16 GB de RAM**
ALMACENAMIENTO	**40 GB de disco**	**100 GB de disco**
ISO	**Ubuntu 20.04.4-live-server-amd64.iso (variante de 64-bits)**	
INTERFAZ DE RED	**Que tenga acceso a internet, para descargar varios GB de datos en repositorios externos**	

Ahora, en la figura 35 se puede visualizar la máquina virtual (***Dev-RN***) que se utilizará para la instalación de la plataforma DevStack, dentro del entorno Proxmox VE. También, se observa que los requisitos de hardware y software ya están configurados, por lo que, la máquina virtual esta lista para su uso.

Máquina Virtual para DevStack

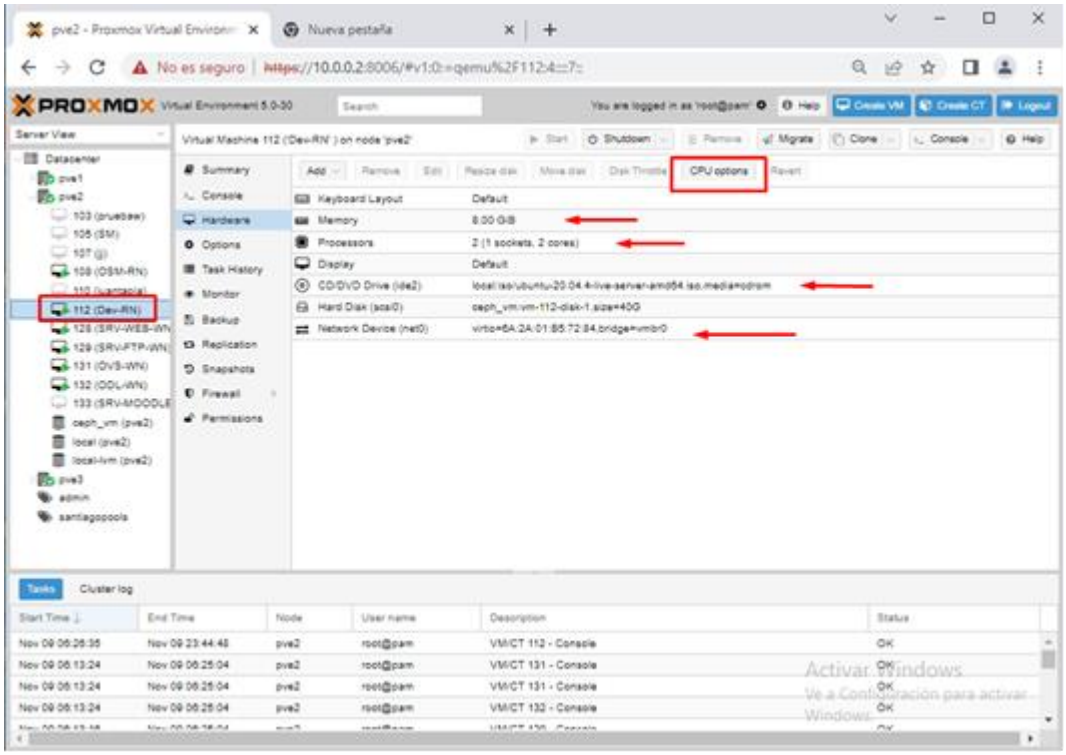

Figura 35. Máquina Virtual para DevStack

Por consiguinete, en la figura 36 se muestra la consola del sistema donde se ejecuta el comando 1 ***git clone*** ***https://opendev.org/openstack/devstack***, el cual descarga y copia la versión más reciente de la plataforma **DevStack.**

Figura 36. Comando 1. Descarga de plataforma DevStack

Por lo tanto, una vez ya descargada la plataforma se revisa si el instalador ya se encuentra en el directorio home, mediante el comando ***ls*** que se puede observar en la figura 37. Puesto que, ingresando a la carpeta devstack, por medio de la sentencia ***cd devstack/*** donde se ejecuta el comando 2 ***gedit inc/python***.
Comando 2. Configuración archivo python.

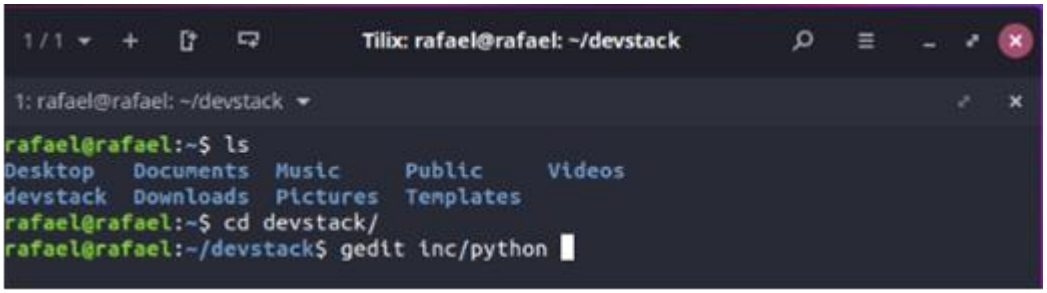

Figura 37. Comando 2. Configuración archivo python.

Seguidamente, la figura 38 muestra el procedimiento de configuración del archivo de instalación **python** en donde, se remplaza la sentencia ***$cmd_pip*** por la línea de comando ***$cmd_pip $upgrade $install_test_reqs –ignore-installed*** \. Por lo tanto, esta línea de comando da permisos como super usuario para la instalación de DevStack, es decir, se ejecuta y se actualizan todos los paquetes de la plataforma sin requerir permisos de prueba o de seguridad.

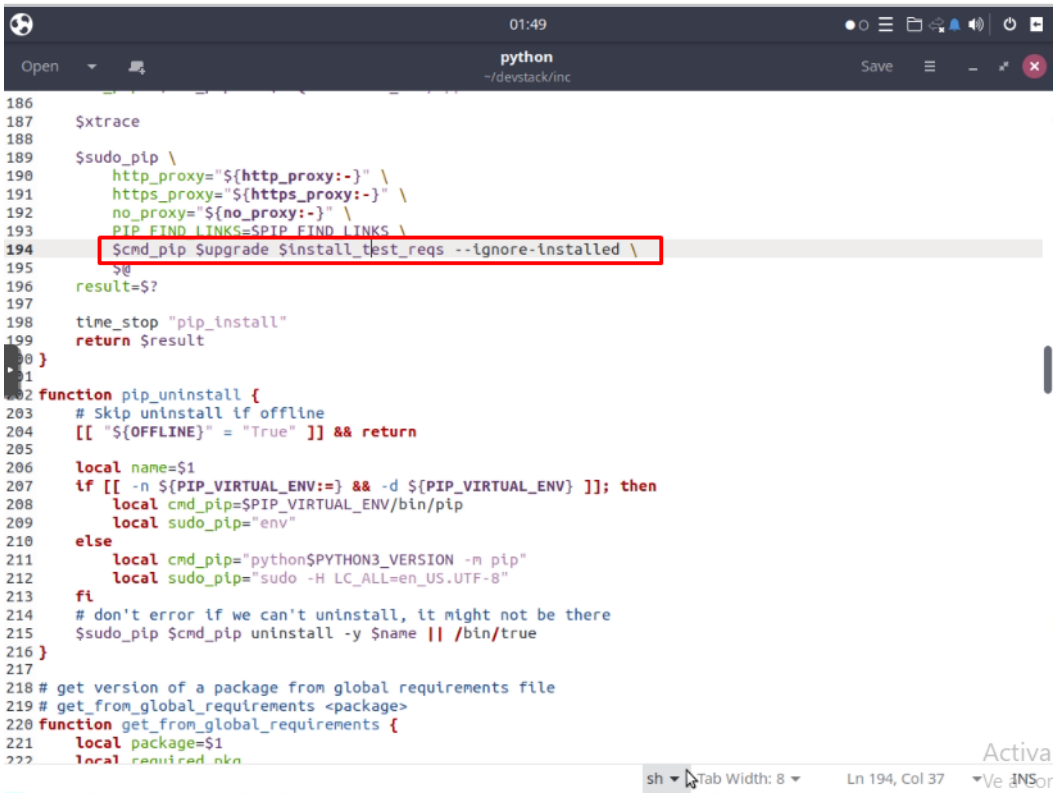

Figura 38. Configuración del archivo de instalacion inc/pyton

A continuación, se accede a la carpeta **samples** donde se extrae una copia del archivo **local.conf** a la carpeta principal de DevStack, esto se lo realiza tal como se muestra en la figura 39 mediante el comando 3 ***cp local.conf ../***

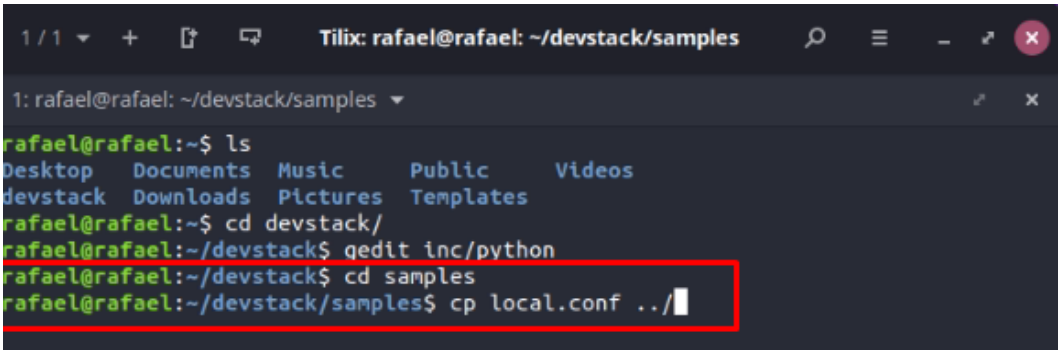

Figura 39. Comando 3. Copia del archivo **local.conf.**

Se ejecuta el comando 4 ***gedit local.conf*** que permite acceder al archivo local.conf, para su configuración. El procedimiento se muestra en la figura 40.

Figura 40. Comando 4 acceso al archivo local.conf.

Posteriormente, dentro del archivo local.conf se configura las contraseñas para acceder a la plataforma, tanto como, en la interfaz web, base de datos y configuración de instancias, por otro lado, en este archivo también se configura la dirección IP, por la cual va a funcionar DevStack.

Cabe recalcar, que debe ser la misma IP de la máquina virtual y estar dentro del rango de IPs del Testbed para que tenga conectividad con OSM. La configuración se observa en la figura 41.

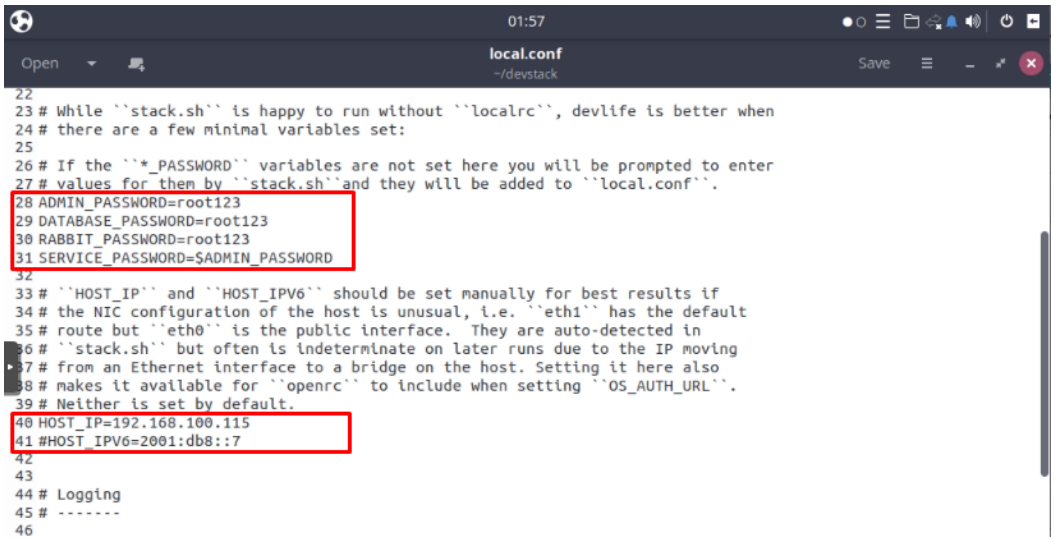

Figura 41. Configuración del archivo local.conf.

Ahora, se procede a instalar DevStack, por lo cual, se detalla el comando 5. ***./stack.sh***. Con este comando se ejecuta el script de instalación y da permisos a todos los requerimientos de la plataforma ya configurados previamente. Su proceso se detalla en la figura 42.

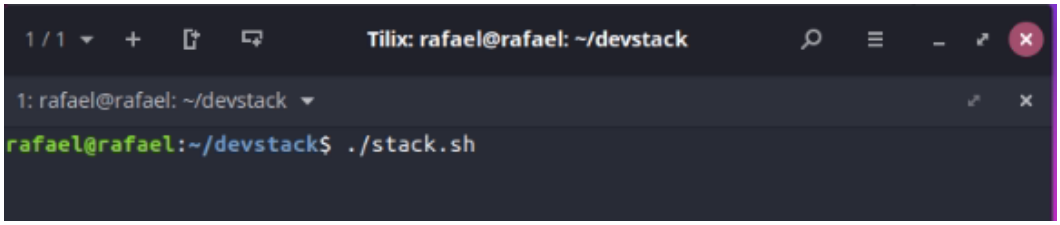

Figura 42. Comando 5. Instalación de DevStack.

Por último, ya finalizado el proceso de instalación, se ingresa a la plataforma mediante el navegador web, donde se escribe la dirección ***https://192.168.100.115/dashboard.*** Este procedimiento se muestra en la figura 43 que indica la portada de OpenStack propia de la plataforma DevStack. Lo siguiente, es acceder al sitio, por medio de las credenciales: User Name: **admin** y Password **root123.**

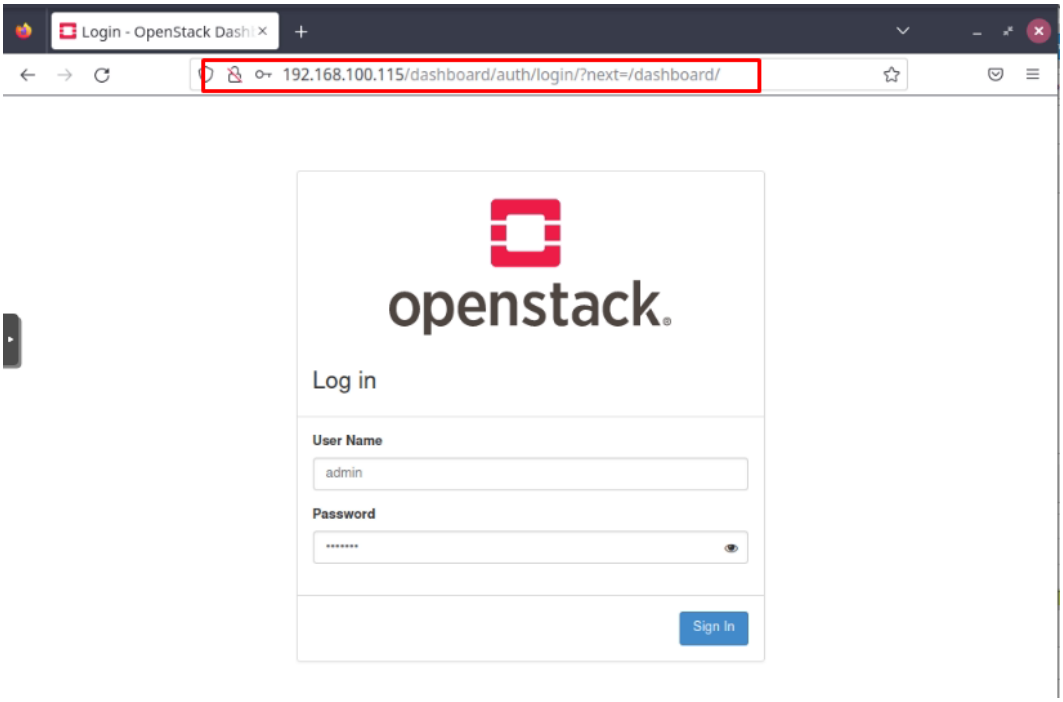

Figura 43. Interfaz web OpenStack.

Para enlazar OpenStack con OSM, es necesario saber el **VIM URL** de la plataforma VIM en este caso OpenStack, es decir, este requerimiento es la API por la cual DevStack se conecta con OSM. A continuación, se detalla el procedimiento para encontrar esta API.

Después del ingreso a la interfaz web de OpenStack mediante las credenciales mencionadas anteriormente, se selecciona la opción **API Access** donde se encuentra la identidad VIM la cual es ***http://192.168.100.115/identity*** , como muestra la figura 44.

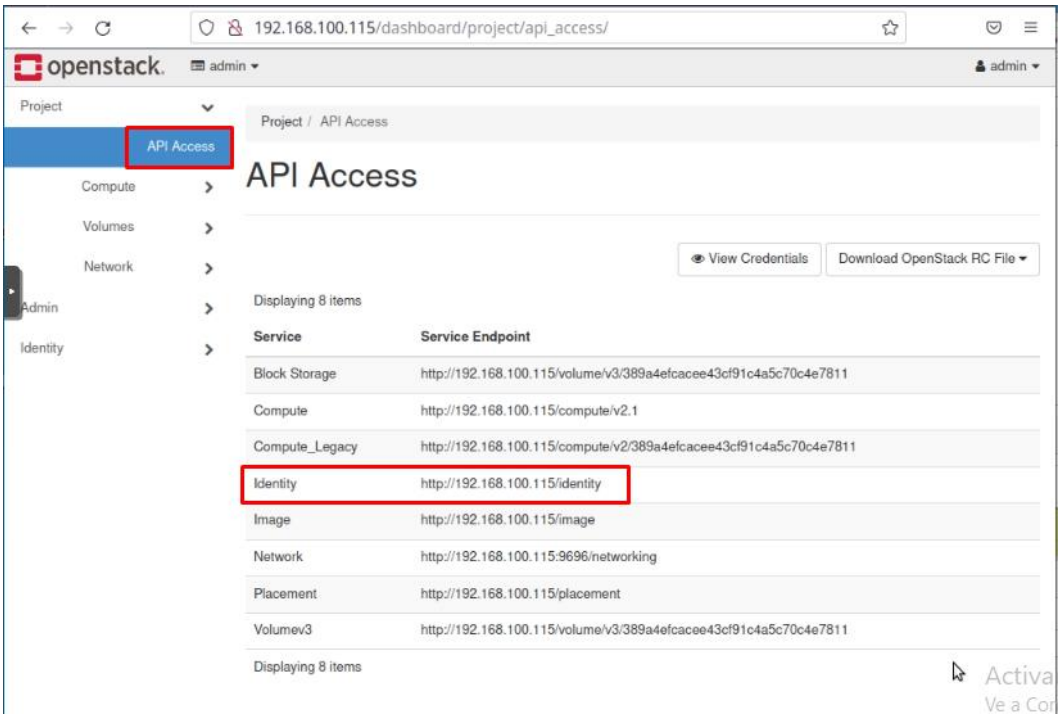

Figura 44. Identidad VIM para conexión con OSM

Por otra parte, esta dirección URL se coloca en otra pestaña añadiendo la siguiente descripción, ***http://192.168.100.115/identity/v3/*** para verificar si la plataforma funciona correctamente, como se nota en la figura 45; la plataforma es estable y puede ser enlazada a otra plataforma. Es decir, este URL es necesario para la autenticación con OSM.

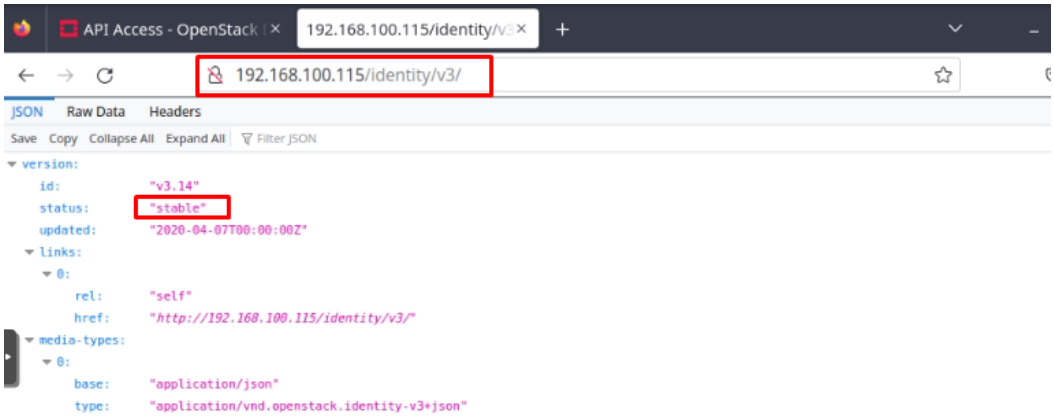

Figura 45. Verificación del VIM URL.

El propósito del procedimiento mostrado anteriormente es para usarse en la conexión con la plataforma OSM, la figura 46 presenta la interfaz web de OSM donde se realiza la configuración necesaria para establecer la autenticación con el bloque VIM, en este caso OpenStack.

Primero se selecciona **VIM Accounts,** después **New VIM**; aquí se despliegan los requisitos solicitados por OSM para una conexión exitosa, por lo tanto, los ítems que reflejan el asterisco son los más importantes y deben ser llenados necesariamente. En este caso, se configura como:

*Name** ***DevStack,*** este parámetro hace referencia al nombre del bloque VIM.
*VIM Proyect/Tenant Name** ***DevStack,*** aquí se utiliza el nombre del proyecto que se utiliza en OpenStack para que tenga relación con las instancias que se van a crear en el bloque VIM.

*Type** ***OpenStack,*** en este parámetro se selecciona el tipo de infraestructura que se utiliza, por lo que es, OpenStack.

*VIM URL** ***http://192.168.100.115/identity/v3/****,* en esta opción, se coloca la API de conexión que se mostró en la figura 45.

*VIM Username** ***admin,*** este parámetro se llena con el nombre de usuario con el cual se accede a la plataforma OpenStack.

*VIM Password** ***root123,*** al igual que el parámetro anterior, aquí se coloca el password de la plataforma OpenStack.

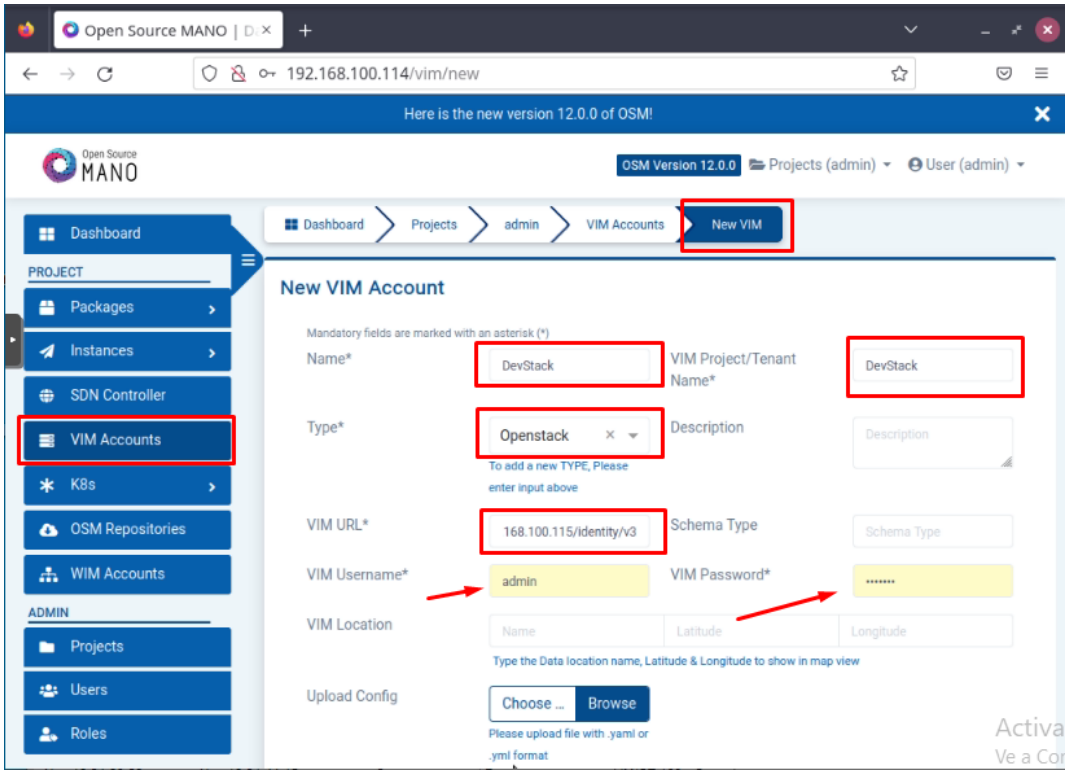

Figura 46. Configuracion del bloque VIM en OSM.

En la figura 47 se puede observar, que la conexión de OSM con OpenStack se realizó correctamente, es decir, la imagen muestra el estado de la cuenta VIM en **ENABLED** y está lista para la creación de instancias.

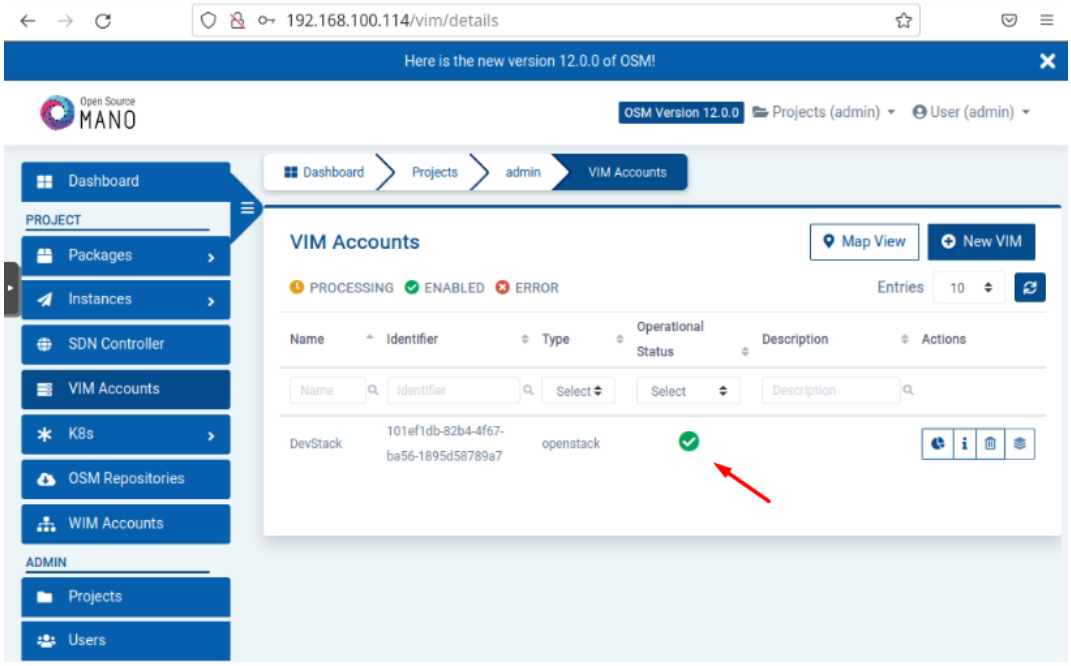

Figura 47. Bloque VIM con OpenStack **ENABLED**

4.2. Instanciación.

A continuación, se detalla el procedimiento para desplegar el NS a través de OSM, es decir, para instanciar los servicios de red virtualizados se selecciona el despliegue de VNFs simples, que se conectan a través de un descriptor de enlace virtual propio de OSM. Por lo tanto, la incorporación de un VNF en OSM consiste en preparar y agregar un paquete VNF adecuado al sistema NFV. Este proceso se lo puede realizar mediante imágenes de VM propias para el bloque VIM en este caso DevStack, donde se instanciará. De este modo se presentan los siguientes literales con el proceso para crear las instancias.

a) Crear una imagen en el bloque VIM

En la página oficial de OSM ***https://osm.etsi.org/*** se puede obtener el siguiente enlace:

https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64.img

el cual permite descargar una imagen adecuada para crear instancias propias de Ubuntu en la plataforma de bloque VIM.

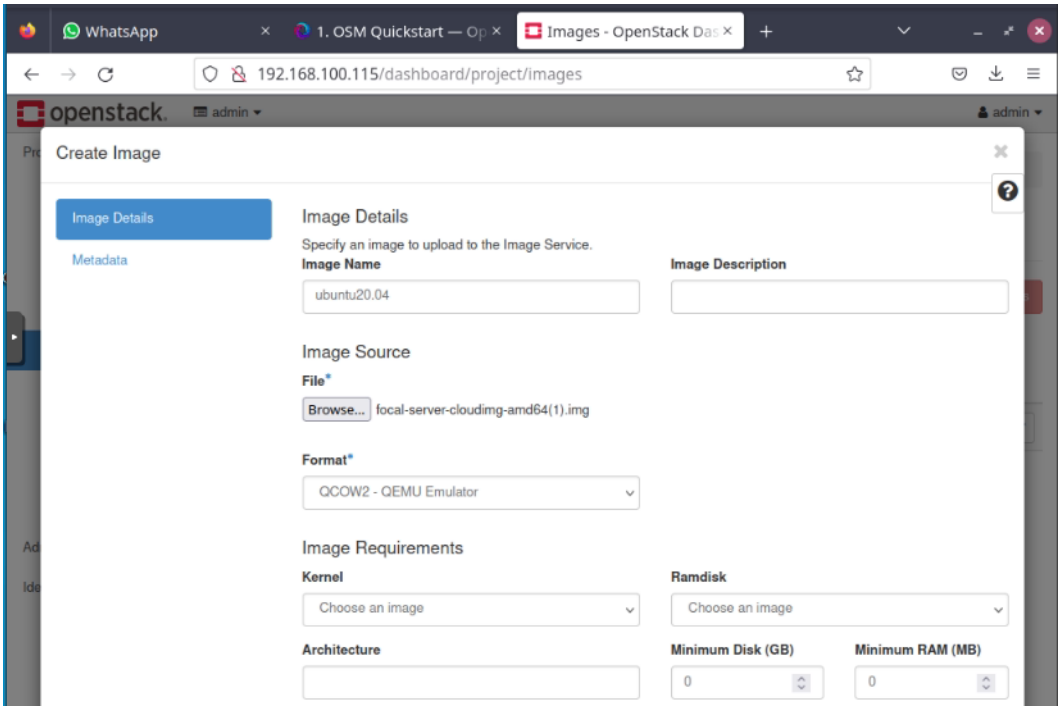

Figura 48. Detalles de imagen creada en DevStack.

En la figura 48 se observa, los parámetros más importantes a configurar para la creación de la imagen en DevStack, es decir, el parámetro más importante es el ***Image Name*** ya que debe tener el mismo nombre en la programación de las VNFs en OSM, este proceso se lo detalla más adelante. Después es importante seleccionar el archivo ***.img*** descargado inicialmente desde la página oficial de OSM, en este caso, se observa que el archivo es ***focal-server-clouding-amd64.img*** y, por último, se debe seleccionar el formato que va a tener la imagen, para este proyecto se utilizara **QCOW2 – QEMU Emulator,** el cual es propio para el despliegue de VNFs basadas en una imagen de Ubuntu.

Posteriormente, ya seleccionados adecuadamente los parámetros detallados en el párrafo anterior, en la figura 49 se muestra que la imagen esta creada correctamente, por lo que, se observa que se mantiene con estado **Active** y refleja el nombre que se utilizará en las VNFs.

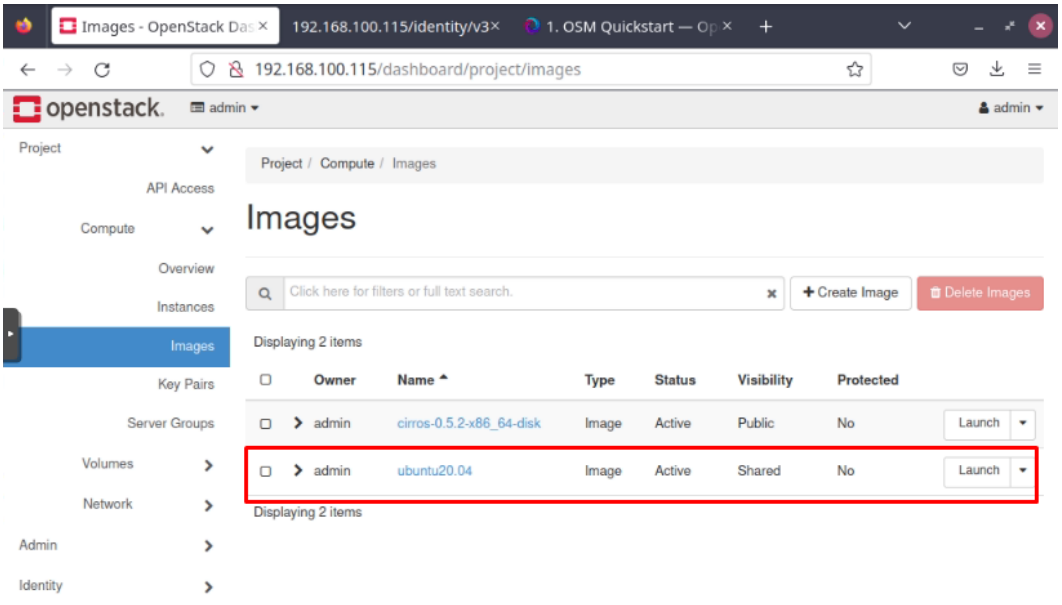

Figura 49. Imagen en DevStack creada y en estado Active.

b) Incorporación de un paquete VNF

Para desplegar la instancia VNF en OSM, es necesario descargar los paquetes VNF y NS propios de OSM los cuales se puede obtener desde el siguiente URL: ***https://osm-download.etsi.org/ftp/Packages/examples/*** cabe recalcar que estos paquetes son diseñados por la comunidad de OSM. Es decir, este paquete es muy importante debido a que contiene librerías y códigos necesarios para el diseño de una instancia, por lo que, debe ser el paquete VNF correspomdiente al sistema debido a que las imágenes VM estan disponibles en el bloque VIM en este caso DevStack.

Desde la interfaz de usuario que presenta OSM, se va a cargar los paquetes VNFs necesarios previamente ya descargados, por consigueinte, en el menú **Packages** de la izquierda se selecciona la opción **VNF Packages** como muestra la figura 50. Es aquí donde, se va a cargar un paquete VNF por lo que, en el área de importación se arrastra el archivo, cabe recalcar que el archivo debe ser **.vnf.tar.gz.**

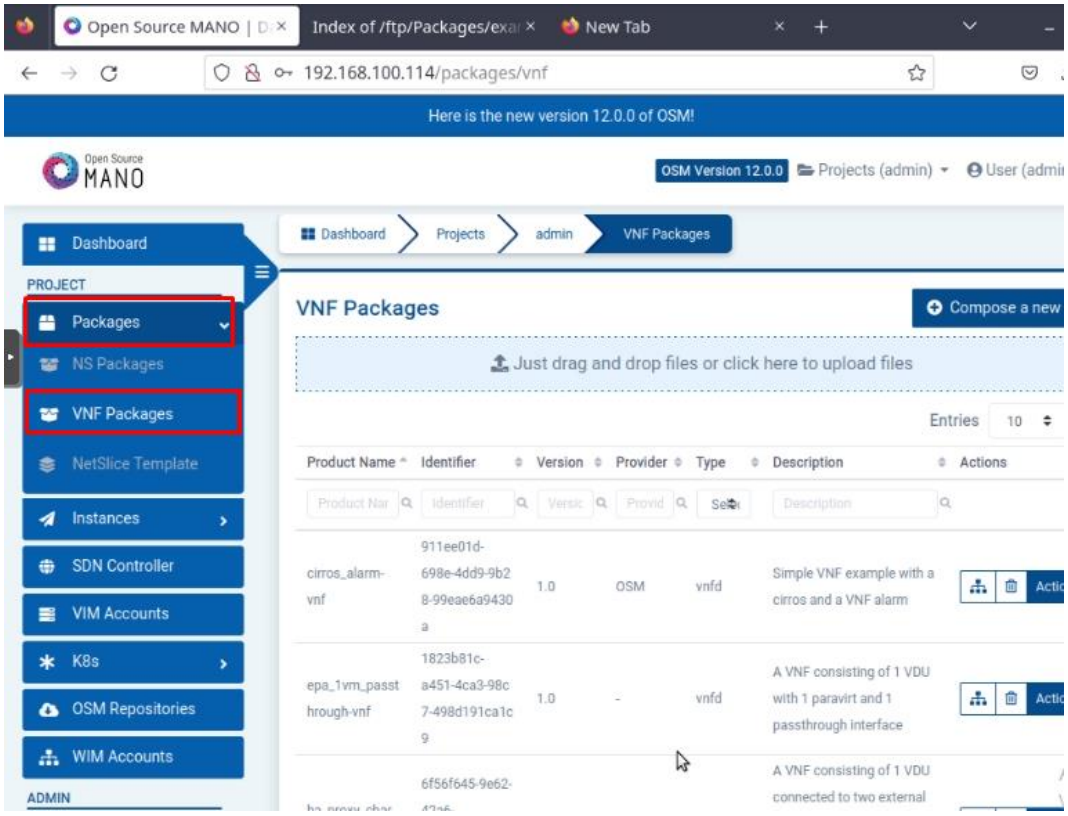

Figura 50. Carga de paquetes VNF.

Este procedimiento también se lo puede realizar mediante comandos por consola, a continuación, se muestra los comandos específicos para cargar un paquete VNF a OSM.

osm nfpkg-create paquete_vnf_vnf.tar.gz

osm nfpkg-lis

c) Incorporación de un paquete NS

Ahora, al igual que el literal **b.** se va a cargar un paquete NS previamente descargado desde el repositorio de OSM. Por lo que, se accede a la interfaz gráfica de usuario en OSM y en el menú se selecciona **Packages**, posteriormente, en **NS Packages** se arrastra el archivo paquete NS descargado el cual debe tener la terminación **.ns.tar.gz.** Proceso que se observa en la figura 51.

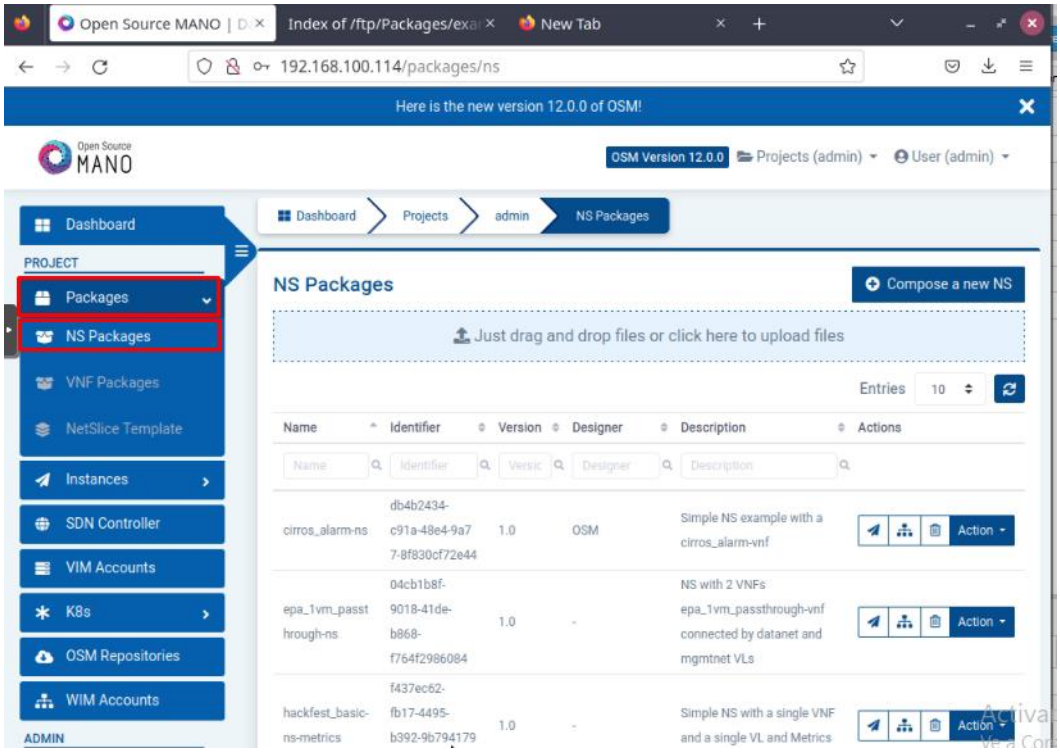

Figura 51. Carga de paquete NS.

Este procedimiento también se lo puede realizar mediante comandos por consola, a continuación, se muestra los comandos específicos para cargar un paquete NS a OSM.

osm nfpkg-create paquete_ns_ns.tar.gz

osm nfpkg-lis

d) Configuración del paquete VNF

En este literal se detalla el procedimiento de configuración del paquete VNF, por lo que se eligie el paquete **hackfest_cloud init-vnf,** el cual presenta una configuración con imagen Ubuntu óptima para la creación de instancias en DevStack y poder tomar recursos de los servidores desplegados en SDN. Por lo cual, es necesario acceder al archivo **YAML**, mediante la interfaz gráfica que ofrece OSM. La figura 52 muestra como acceder al archivo **YAML,** es decir, se selecciona en **VNF Packages** la opción **Action** y luego click en **Edit.**

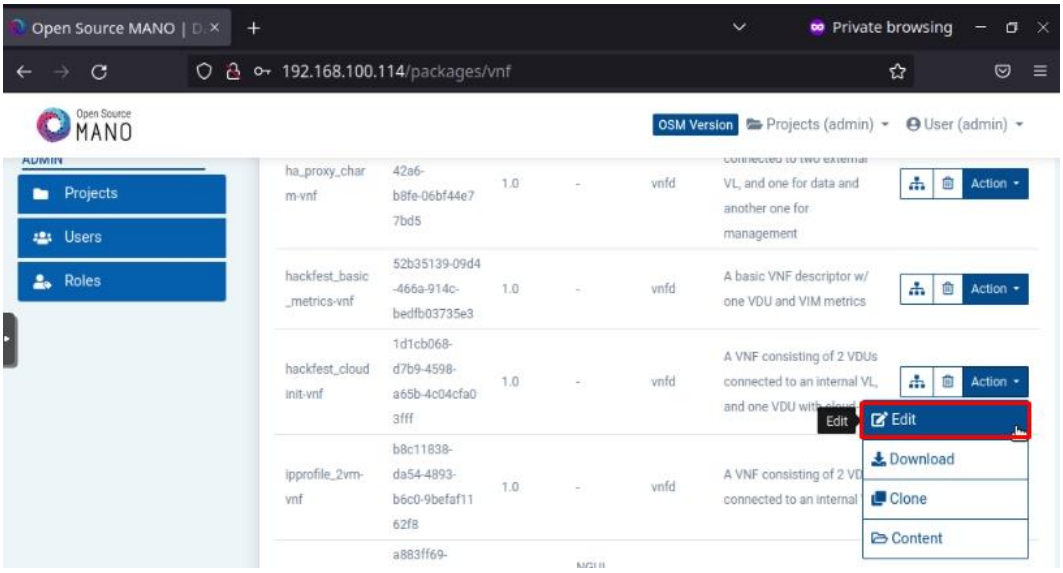

Figura 52. Acceso al archivo YAML.

Posteriormente, se despliega el archivo yaml y es posible configurarlo dentro de OSM, a continuación, su configuración se detalla en los siguientes pasos:

1) Es importante ejecutar el nivel de instanciación que se va a presentar en este proyecto, por lo que, se elige **default** (*línea de comando número 5*) esto quiere decir que OSM define el nivel de instancia para su despliegue. Además, es necesario crear el tipo de **VDU** que va a necesitar la instancia, por lo que se usará dos, las cuales son: **mgmtVM** y **dataVM** (*líneas de comando 10 y 12*)**.**

   Estos son requerimientos por defecto de la máquina virtual, es decir, **mgtVM** hace referencia a la interfaz de administración y **dataVM** se refiere a la interfaz de datos, por lo tanto, se crea un perfil VDU con estos nombres como se muestra en la figura 53.

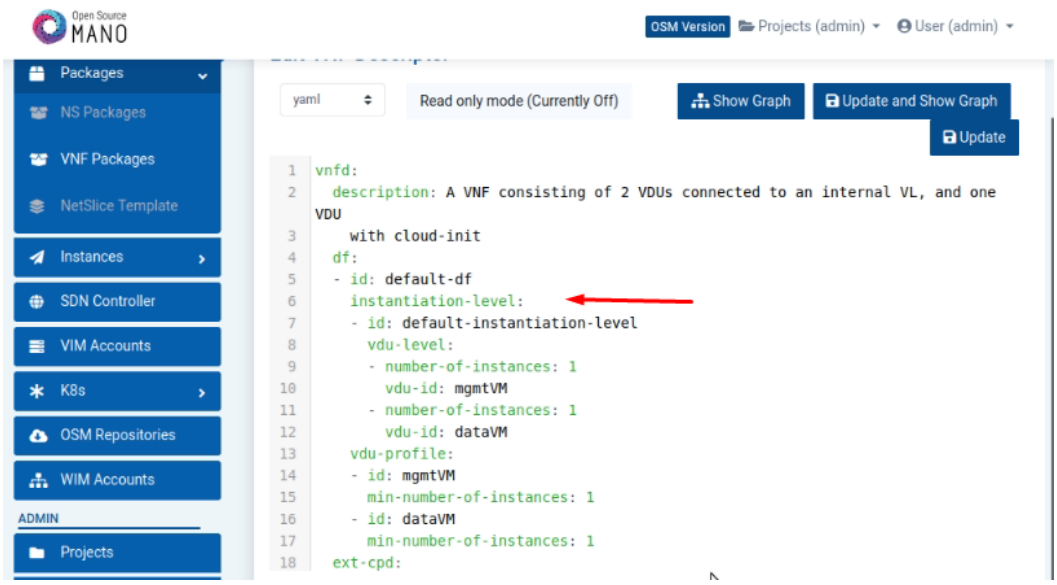

Figura 53. Configuración 1 del archivo YAML

2) Se configura las interfaces de red que deben estar enlazadas a cada **VDU** (*línea de comando 19 y 23*), este procedimiento se lo realiza mediante un **CP** (Connection Point) (*líneas de comando 21 y 25*), es decir, se crea cada punto de conexión directamente conectados con cada **VDU**, identificando el tipo de interfaz de red a utilizar, cabe recalcar que cada **VDU** tendrá una interfaz externa y dos internas. La figura 54 muestra la creación de las interfaces externas, las cuales permiten la conexión entre dos **VDUs** y directamente con su interfaz interna.

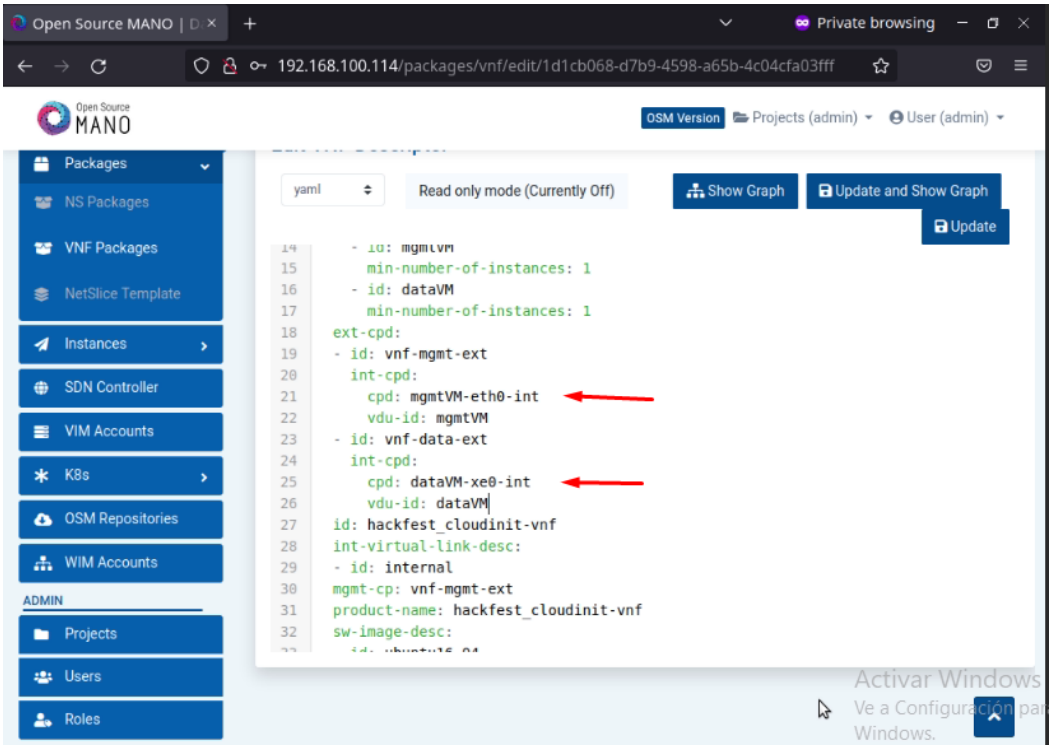

Figura 54. Configuración 2 del archivo YAML.

3) Para la configuración de las interfaces internas es importante editar el tipo de controlador que se va a utilizar; como es un ambiente virtualizado se opta por el controlador **PARAVIRT** (*línea de comando 66 y 72*) el cual es propio del kernel de Linux que permite construir VMs de sistemas virtuales hiperconvergente sobre un hipervisor, en este caso para **mgmtVM (eth0 y eth1)** (*líneas de comando 43 y 49*) y para **dataVM (eth0 y xe0)** (*líneas de comando 63 y 69*). Por lo tanto, este controlador es una API que virtualiza las operaciones que permiten la ejecución de kernel sobre un entorno virtual para la conexión nativa entre VMs sin Hipervisores.

Además, se configura uno de los aspectos más importantes en **NFV,** el cual es la imagen que se usa para la conexión con el bloque VIM, es decir, se ejecuta el nombre de la imagen que fue creada en el literal a) en este caso **Ubuntu16.04**, (*línea de comando 74*).

Es así como, en la figura 55 se observa la configuración de cada interfaz interna y la imagen a utilizar con DevStack**.**

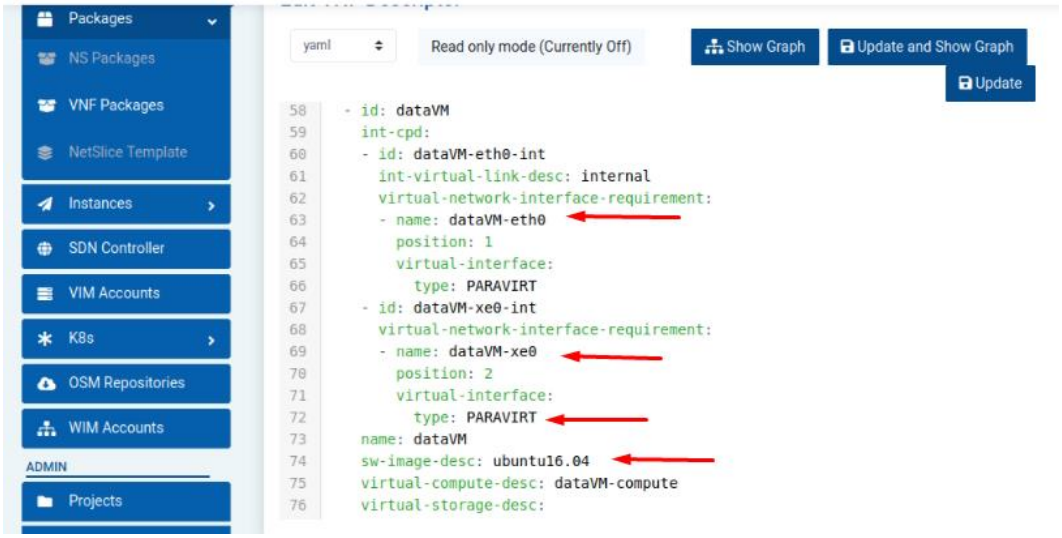

Figura 55. Configuración 3 del archivo YAML.

4) Por último, se configura los parámetros de cómputo, alamacenamiento y memoria que van a tener las VMs instanciadas, como anteriormente se configuró la instanciación por **default**; OSM define las capacidades de cada parámetro por defecto, como se puede ver en la figura 56 (*lineas de comando 80, 85, 91 y 93)*. Cabe recalcar que las capacidades de memoria, almacenamiento y computo, se define en DevStack al momento de definir la capacidad total de cada instancia.

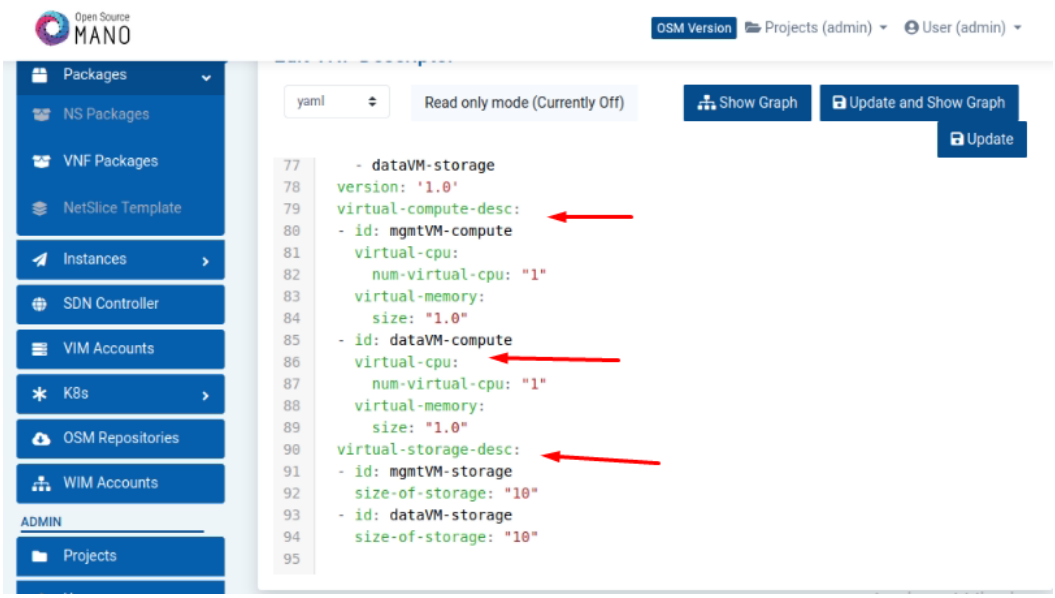

Figura 56. Configuracón 4 del archivo YAML.

a) Creación de la Instancia Network Service (NS)

Para crear la instacia NS, a través de la interfaz web que ofrece OSM, se selecciona la opción NS Instances, después el boton New NS, por lo cual se despliega un recuadro donde se va a cofigurar los requisitos necesarios de la instancia. Como se muestra en la figura 57 los parámetros que tienen el * son los más importantes a configurar, para este proyecto se escribe el nombre de la instancia como Testbed y en descripción Testbed para NFV, y el más importante en VIM Account se selecciona la plataforma DevStack que ya se muestra activa en OSM y por último click en crear.

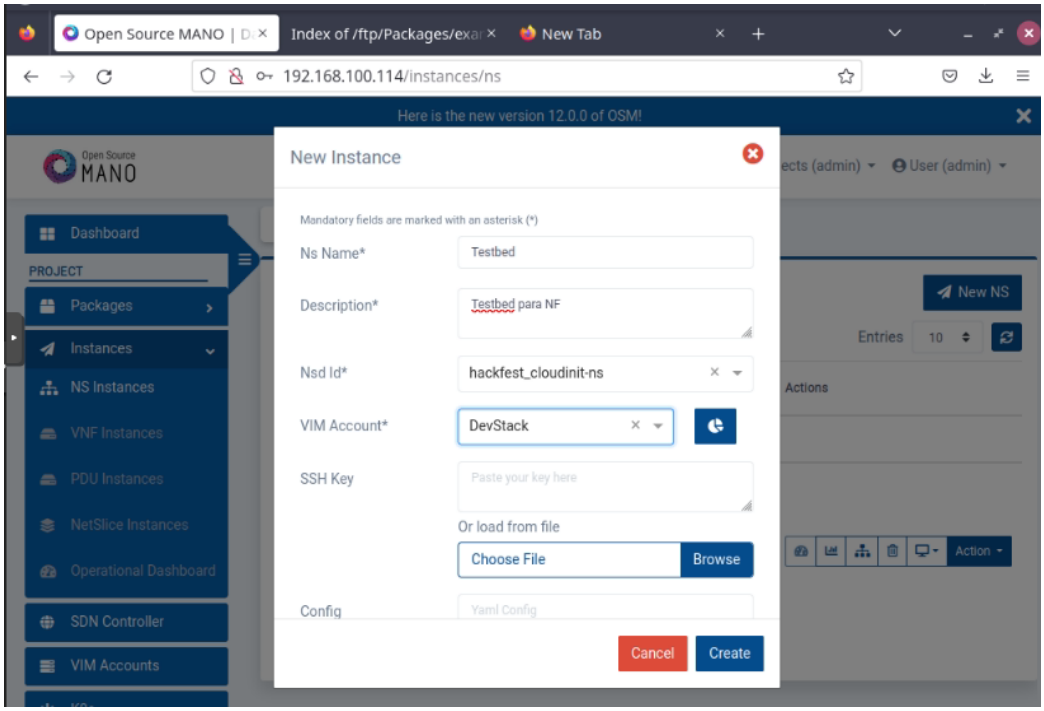

Figura 57. Parámetros de la Instancia NS.

El proceso de creación tarda unos minutos, sin embargo, en la figura 58 se observa que la instacia está configurada y ejecutandose (running) correctamente.

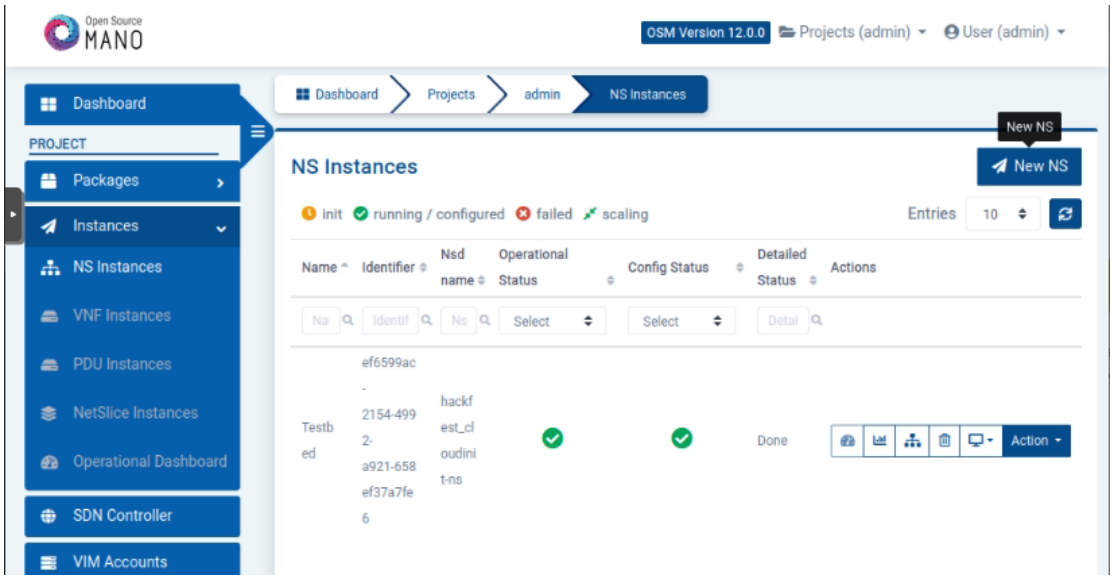

Figura 58. Instancia creada y corriendo.

Ahora, en la plataforma DevStack al ingresar a **Instances** se observa la instancia **Testbed** que fue creada en OSM, la cual presenta cada **VNF** instanciada,

indicando su nombre, imagen, direccionamiento interno como externo, es decir se muestra la ip denominada **Testbed-Internal** la cual hace referencia a la ip con la cual se comunica internamente con sus interfaces, sin embargo, tambien se muestra la ip denominada **Testbed-datanet** la cual se entiende como ip externa que comunica con cada VNF. Además, se verifica su estado, que es **Activo y Ejecutando.**

Por otro lado, la opción **Action** permite administrar cada VNF revisando sus logs, consola y requerimientos para su comporbación enable al ejecutarse. Los detalles mencionados se muestran en la figura 59.

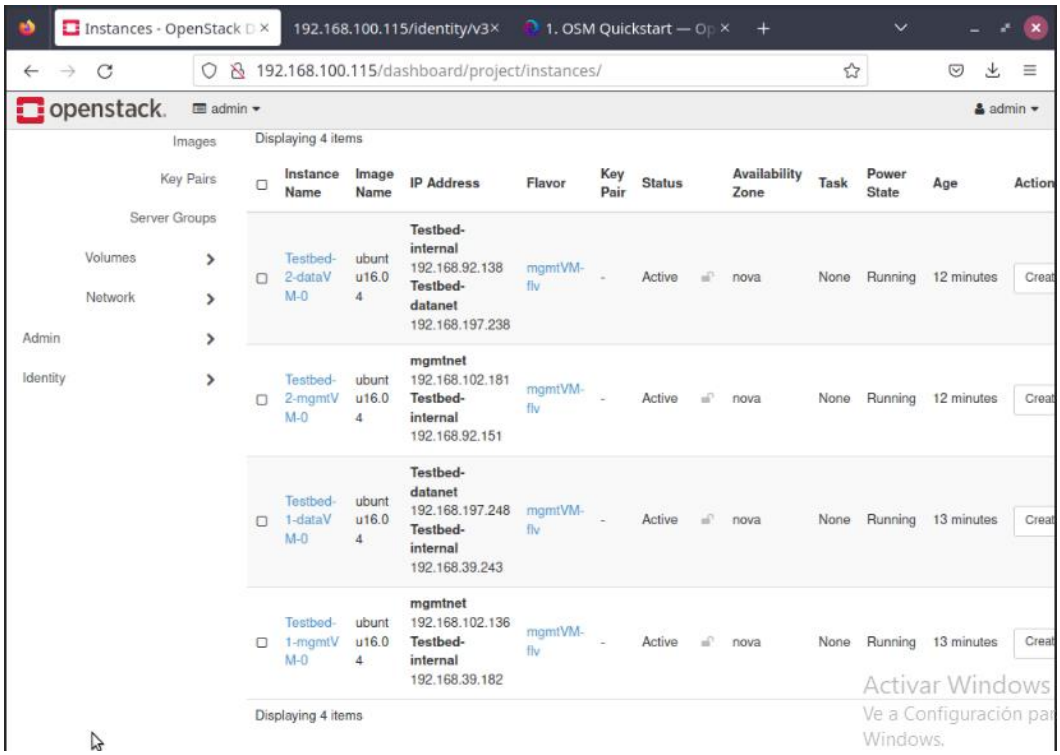

Figura 59. Instancia creada activa y corriendo en DevStack.

Por otro lado, para verificar la correcta virtualización de los servicios de red, en la figura 60 se muestra a la plataforma OSM, donde se despliega una topología ilustrada por la instancia creada, es decir, se refleja cada **VNF, VL y CP** detallados en el paquete VNF.

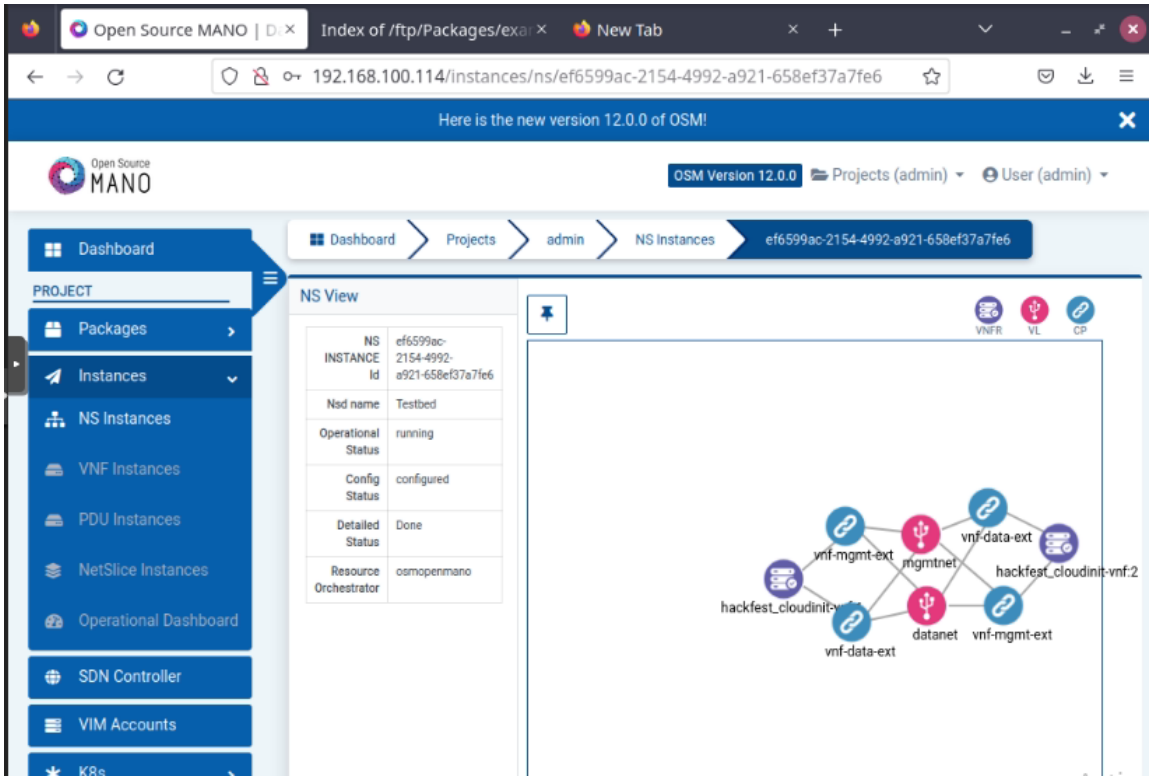

Figura 60. Topología de los servicios de red virtualizados.

5. CAPITULO V
RESULTADOS Y PRUEBAS DE FUNCIONAMIENTO DEL TESTBED

En este capítulo se detallan los resultados de este trabajo de investigación mediante las pruebas de funcionamiento del Testbed implementado, que permite verificar la correcta instalación e integración de todos los componentes de la arquitectura NFV. Destacando las virtudes que proporciona la tecnología, como la reducción significativa de recursos hardware, fácil implementación de servicios de red y funcionalidad al momento de integrar varias plataformas, es decir, el fácil despliegue de un entorno de infraestructura como servicio, ya que OSM puede ser utilizado por más usuarios y administradores de red debido a su accesibilidad. Por consiguiente, en esta sección de acuerdo con la metodología realizada, los resultados detallados a continuación corresponden a cada objetivo.

5.1. Funcionamiento de la Topología de Red en un ambiente controlado por plataformas OSM, OpenStack, OpenDayLight y Proxmox VE.

Es importante destacar que la tecnología NFV necesita de su arquitectura para poder virtualizar una función de red. Por lo cual, cada plataforma que se desplegó en el ambiente Testbed, hace referencia a un bloque en la arquitectura NFV.

5.1.1 Bloque NFV Infrastructure (NFVI).

NFVI proporciona los recursos de hardware y software necesarios para la ejecución de las funciones de red virtualizadas. Por lo tanto, para el Testbed se desplegó una SDN con el protocolo OpenFlow mediante la plataforma OpenDayLight, la cual proporciona los mismos recursos que necesita un NFVI. A continuación, en la figura 61 se muestra la interfaz web de la plataforma OpenDayLight que mediante el protocolo OpenFlow brinda un dominio de cómputo, el cual, se encarga de proporcionar recursos de almacenamiento y computo, también, ejecuta el dominio de hipervisor, mediante el cual se comparte los recursos del dominio de cómputo a las máquinas virtuales proporcionando una

abstracción de hardware; y por último, con un dominio de infraestructura de red que comprende todas las interfaces de red interconectadas. Además, en la figura 61 se observa los servidores que se usó para compartir sus recursos y el servidor donde se alojó la plataforma OSM.

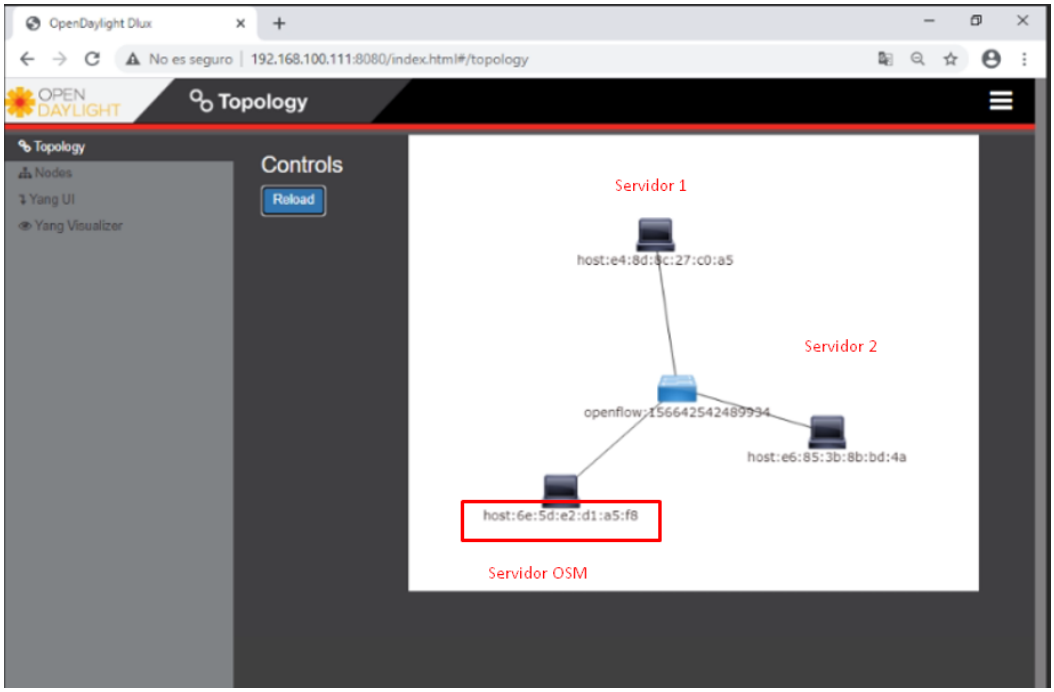

Figura 61. Bloque NFVI.

Como se mencionó en el anterior párrafo, la figura 61 muestra la topología SDN funcionando, la cual es parte del bloque NFVI, en ese caso, se observa que la máquina virtual cuya direccion ***MAC: 6e:5d:e2:d1:a5:f8*** corresponde al servidor donde se aloja la plataforma OSM, posteriormente en la figura 62 se muestra los resultados de ejecutar el comando **ip add**, donde se comprueba que esta MAC corresponde a la platarforma OSM.

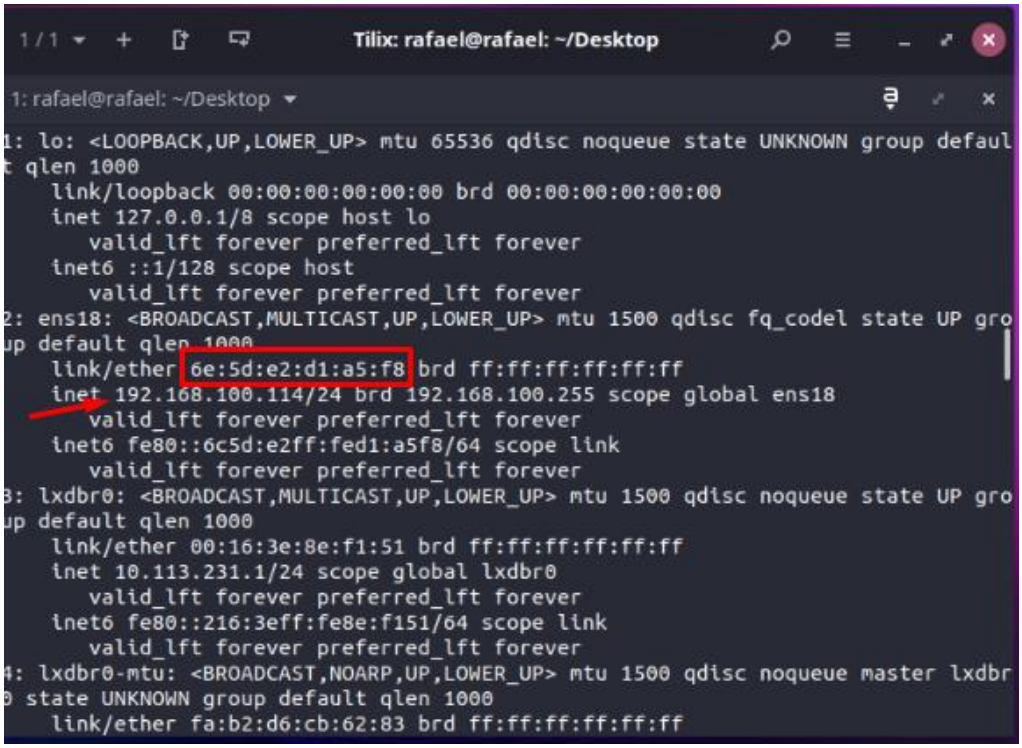

Figura 62. MAC de plataforma OSM.

Por otro lado, para verificar la conectividad entre las diferentes plataformas se realizó una captura de wireshark donde se evidencia la conectividad entre OSM y OVS, cabe recalacar que OVS es el OpenvSwitch el cual genera paquetes openflow para el funcionamiento de SDN. Es decir, a continuación la figura 63 muestra los paquetes openflow que inundan el bloque NFVI.

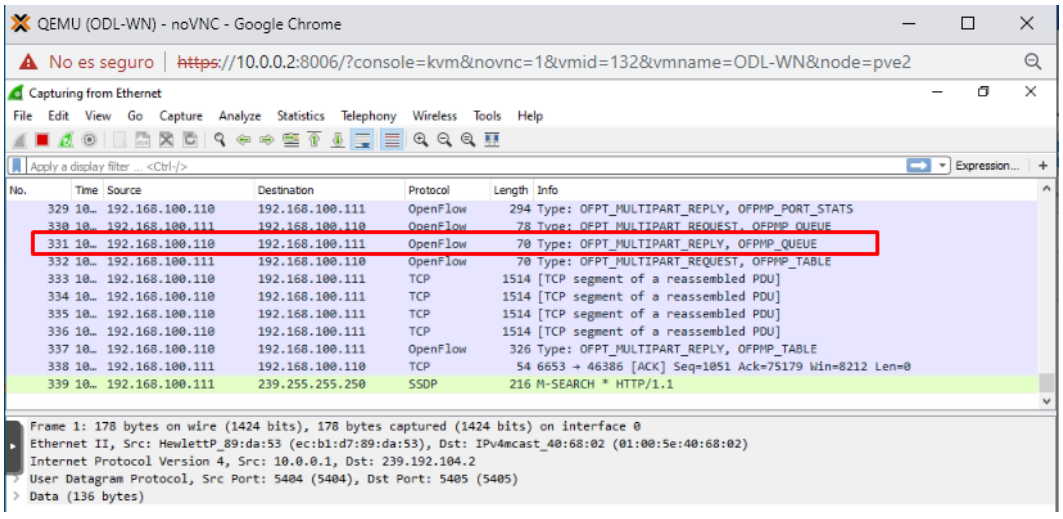

Figura 63. Captura de Wireshark de paquetes OpenFlow.

Por último, la figura 64 muestra la interfaz web de OSM donde se puede observar que el **SDN Controller** está en modo **Enable**. La figura también muestra los parámetros del **SDN Controller** los cuales son propios de la plataforma OpenDayLight, es decir, el **URL** ***https://192.168.100.111:8080*** por el cual funciona su interfaz web, el **datapath_id (DPID) 00:00:f6:17:02:08:c0:4b** el cual contiene el ID de instancia y la direccion MAC de bridge del OVS.

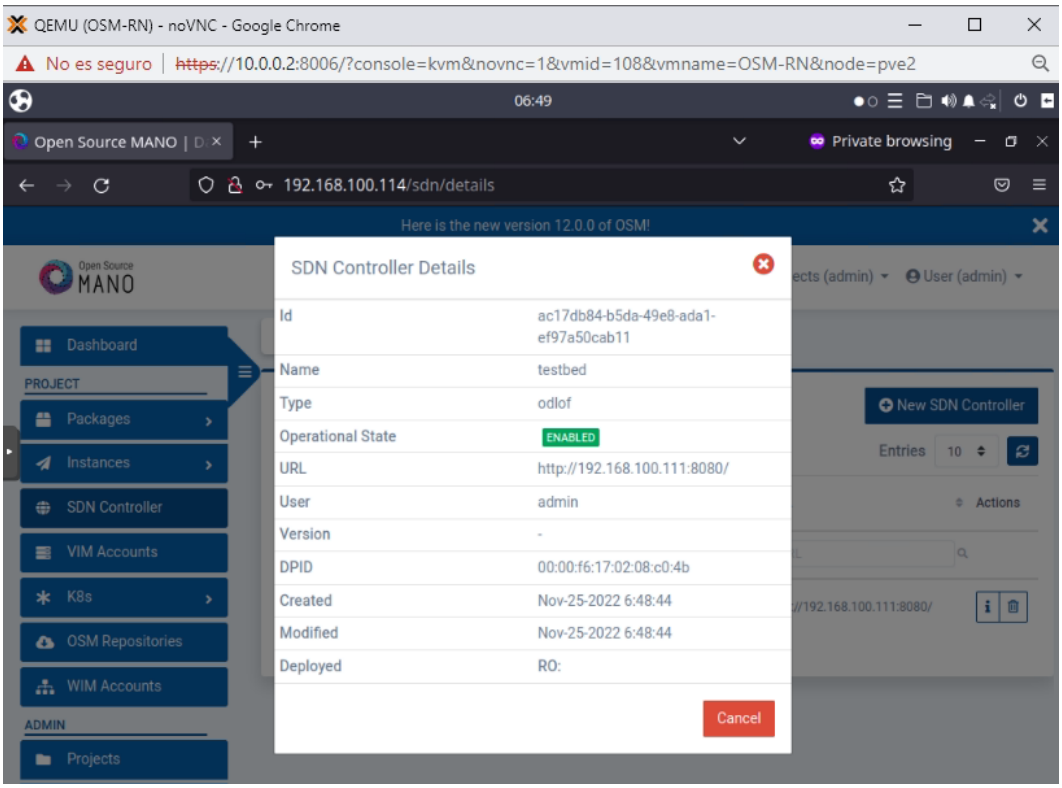

Figura 64. SDN Controller Enable en OSM.

5.1.2 Bloque Virtual Infrastructure Manager (VIM VNFs)

Para este bloque se desplegó la plataforma OpenStack mediante el protocolo DevStack, esta plataforma fue la indicada para ser el bloque VIM, ya que sus caracteristicas permite crear, administrar y gestionar VNFs. En consecuencia, con DevStack la implementación de software de una función de red es capaz de ejecutarse sobre el NFVI, de modo que, las VNFs son las funciones de red virtualizadas que usan máquinas virtuales para su despliegue, por otro lado, la

conectividad interna entre VNFs se realizó por APIs que actúan por medio de la plataforma OSM y que se configuran a partir de un paquete VNF.

Por lo tanto, en la figura 65 se muestra la interfaz web de la plataforma OSM, la cual indica los detalles de la plataforma que esta operando como bloque VIM, es decir la plataforma **DevStack.** Cabe recalcar, que cada parámetro que se observa en la figura corresponde a DevStack y además, se encuentra en estado **Running/Configured.**

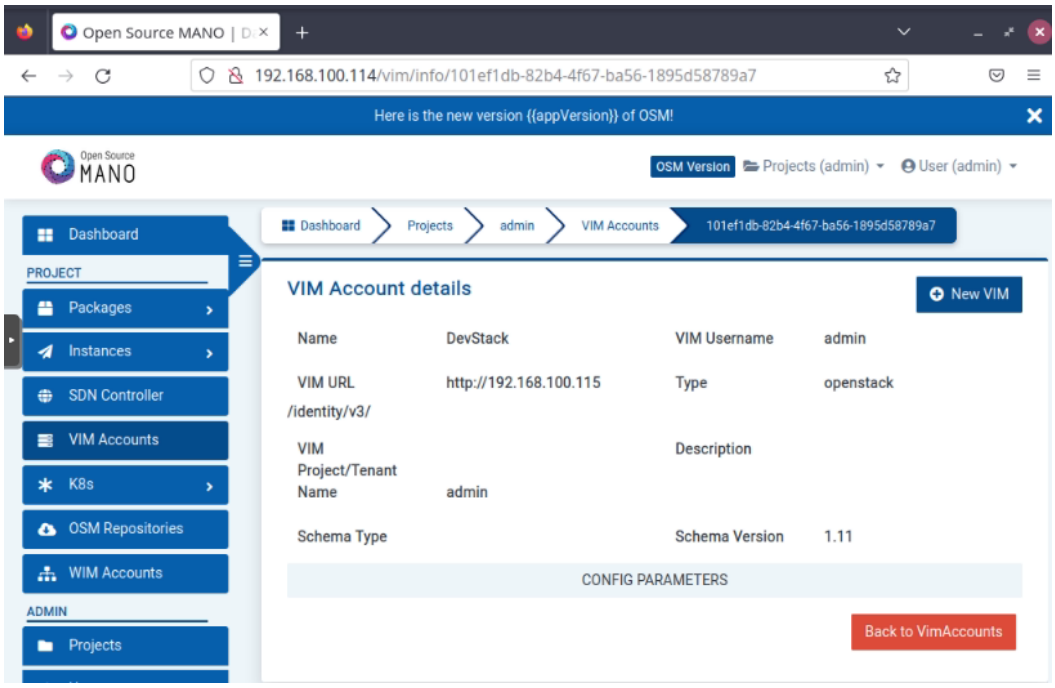

Figura 65. Conexión de OSM con bloque VIM (DevStack).

Además, en la figura 66 se muestra el ambiente virtual que fue creado a partir de la imagen Ubuntu en la sección 4.2. El cual tiene un vinculo con OSM que sirve para desplegar las funciones de red. Es decir, en la figura se muestra que el ambiente virtual ya cuenta con 4 instancias, que poseen un valor en **RAM y Storage** los mismos que fueron configurados en el paquete VNF a través de OSM, por lo que, se presenta la cantidad en uso y el total de los recursos compartidos a cada instancia.

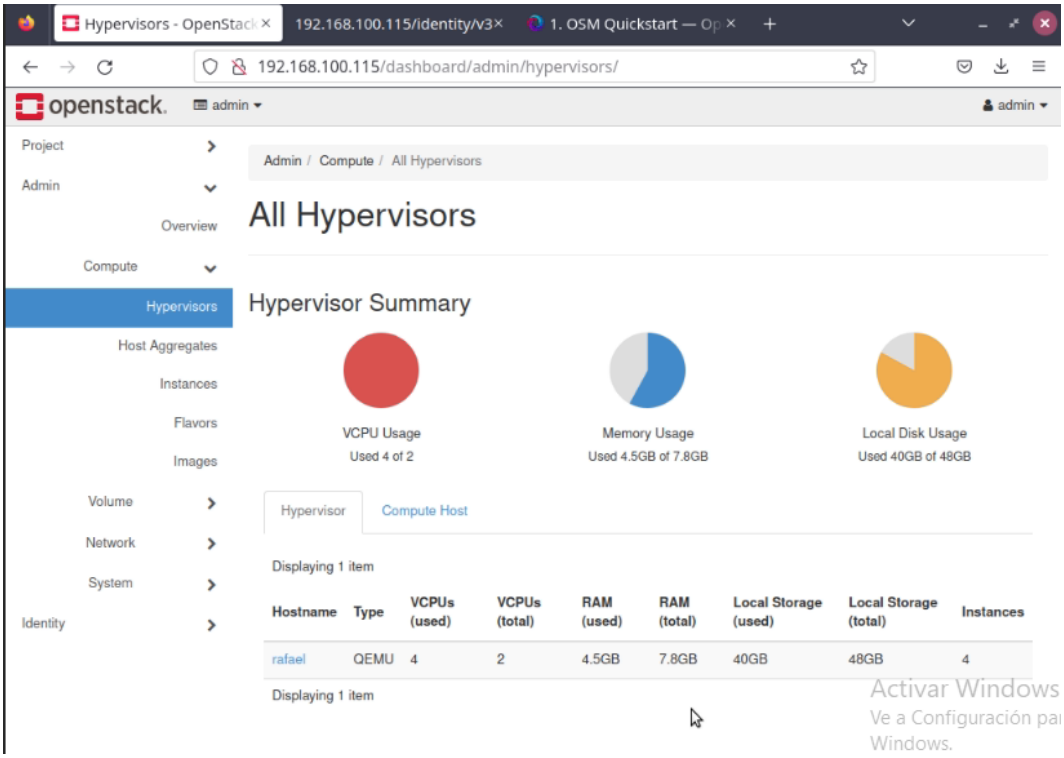

Figura 66. Testbed Hypervisor en OpenStack.

5.2 Virtualización de las Funciones de Red

La implementación de las funciones de red en una infraestructura virtualizada como servicio simplifica y hace que estas funciones sean independientes del hardware propietario, lo que da lugar a una reducción, tanto de inversión como de costes operativos, por lo que, se utiliza una única sintaxis para el despliegue de cada una de estas funciones, es decir, mediante un paquete VNF por medio de la plataforma OSM es posible programar e implementar funciones de red de forma rápida, flexible y escalable, lo cual permite su monitoreo y control a través de un entorno de pruebas denominado **Testbed**.

En este caso, para este proyecto se desplegaron dos VNFs simples basadas en una imagen Ubuntu 16.04, las cuales poseen dos VDUs conectados a través de dos VLs interno y externo. A continuación, se muestra en la figura 67 las VNFs instanciadas.

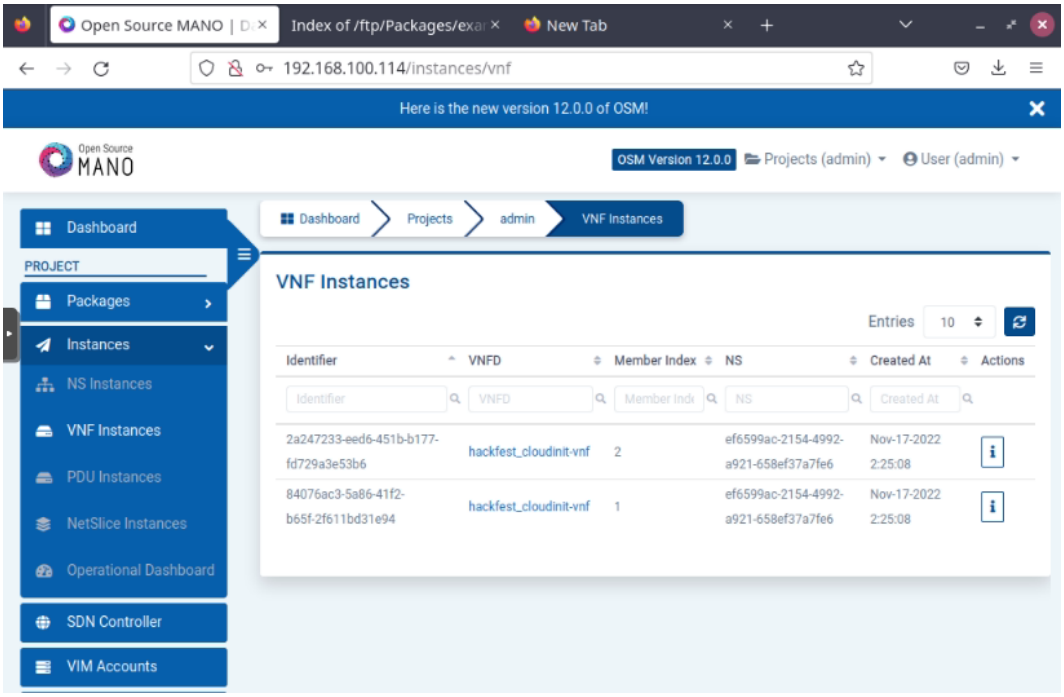

Figura 67. VNFs Instanciadas.

Por lo que, en la figura 68 se presenta el **Network Service Instances** que indica el resultado obtenido, en una topología de VNFs instanciadas, es decir, la virtualización de cada función de red representada con los siguientes elementos: Virtual Deployment Unit (VDU), Virtual Link (VL) y Connection Point (CP).

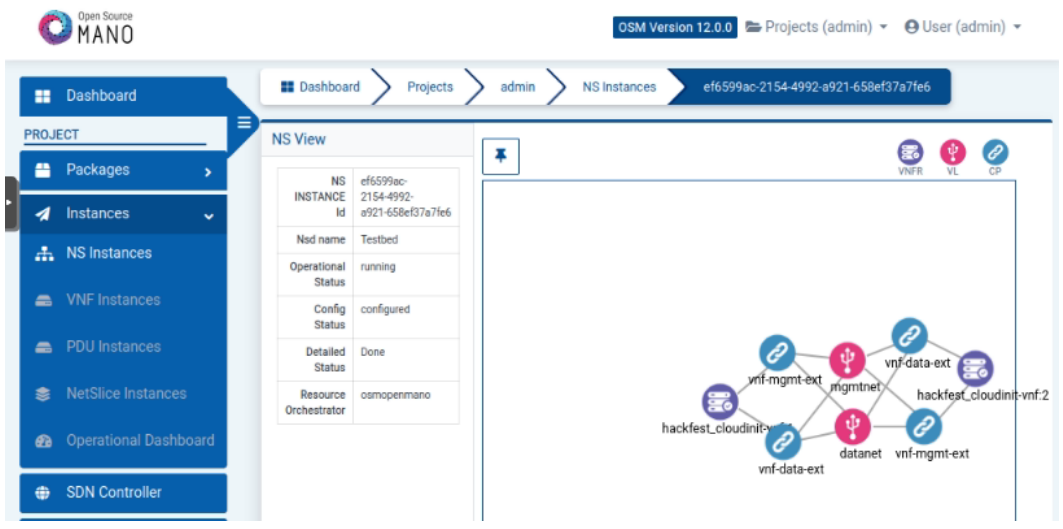

Figura 68. Network Service Instances.

Por otro lado, en OpenStack se muestra las VNFs creadas, que por medio del parámetro **Action** su configuración, monitoreo y gestión es flexible y escalable, es decir, se puede realizar procesos de verificación de software mediante la **Consola**, además, comprobación de logs por medio de la opción **View Logs.** Por lo tanto, cada parámetro mostrado en la figura 69 hace referencia al monitoreo de cada VNF.

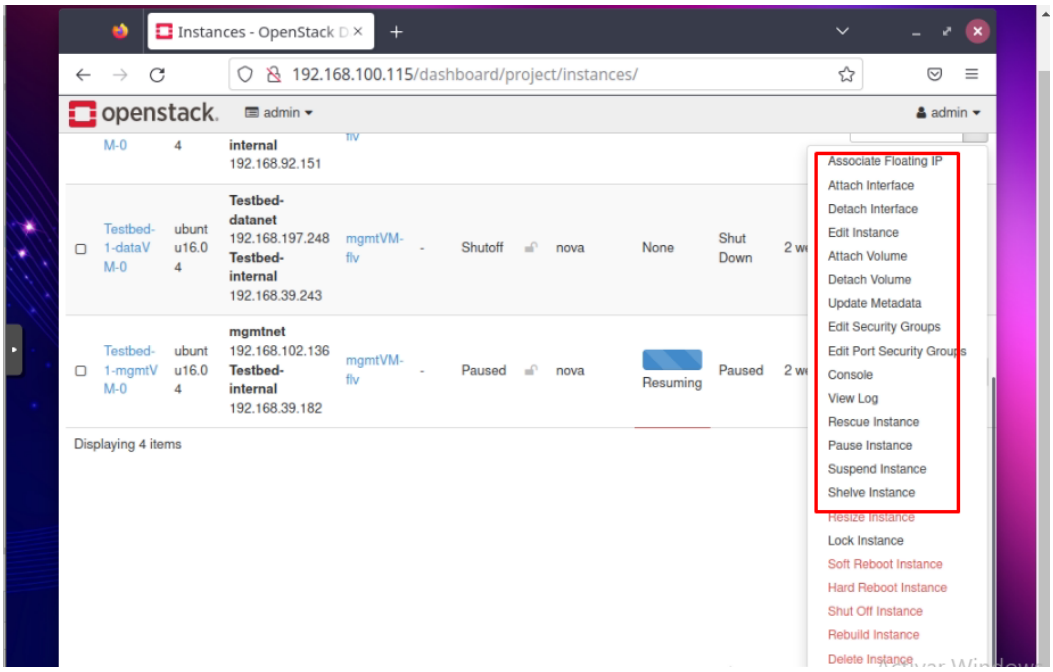

Figura 69. Control y monitoreo de cada VNF.

Por lo tanto, a continuación se presentan los parámetros que permiten el control y monitereo de cada VNF a través de la seccion **Instances** en OpenStack:

- ***Instance Console.-*** este parámetro permite realizar modos de configuración, por medio de lineas de comando para cada VNF instanciada, como se muestra en la figura 70**.**

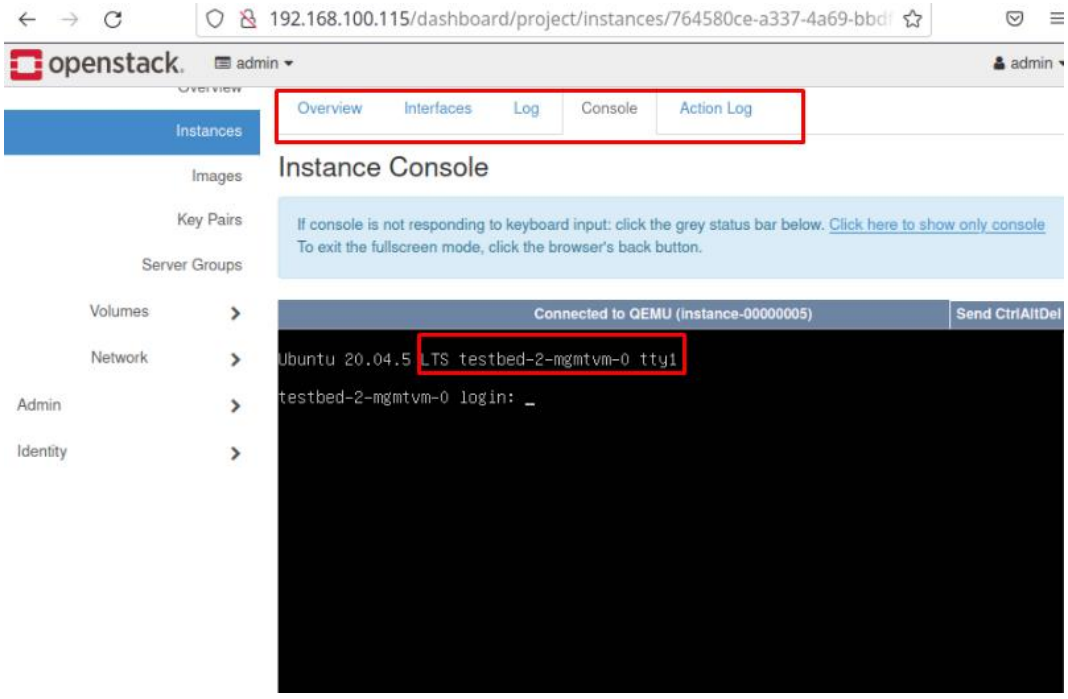

Figura 70. Configuración consola de cada VNF.

- ***Instance Console Log.-*** este parámetro ayuda con el monitoreo de cada VNF, debido a que muestra el comportamiento del sistema operativo, además el permitir al administrador verificar algún posible daño. Por lo que, lo mencionado se muestra en la figura 71.

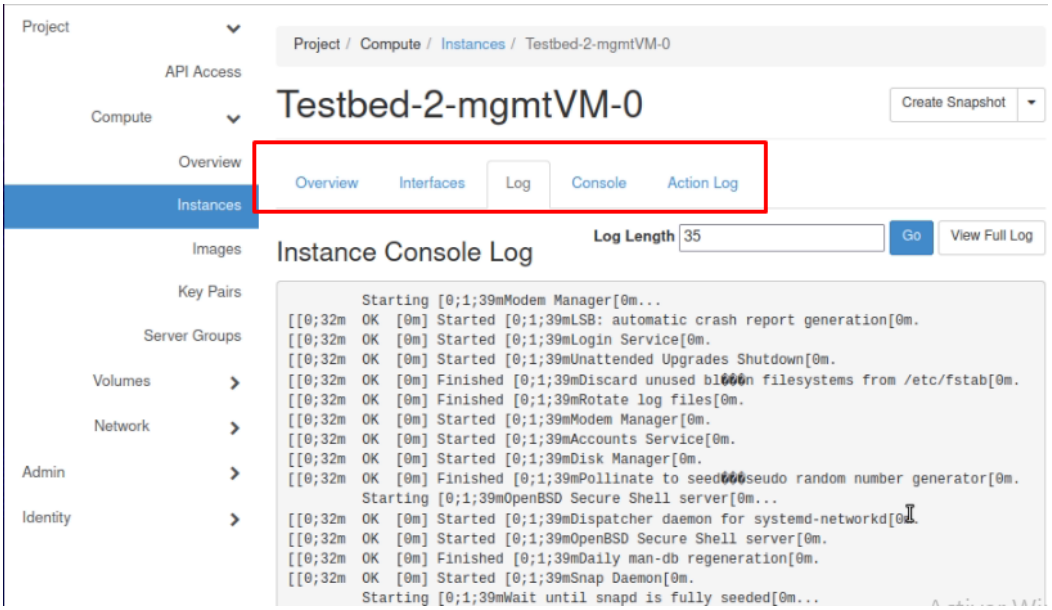

Figura 71. Configuración Logs de cada VNF.

- ***Interfaces.-*** en este parámetro se puede verificar si cada interfaz de red configurada en el paquete VNF esta funcionando, de manera que las figuras 72 y 73 muestran los items que contiene cada interfaz de red, los cuales son: **Network, Fixed IPs, MAC Address, Status** y Admin **State.**

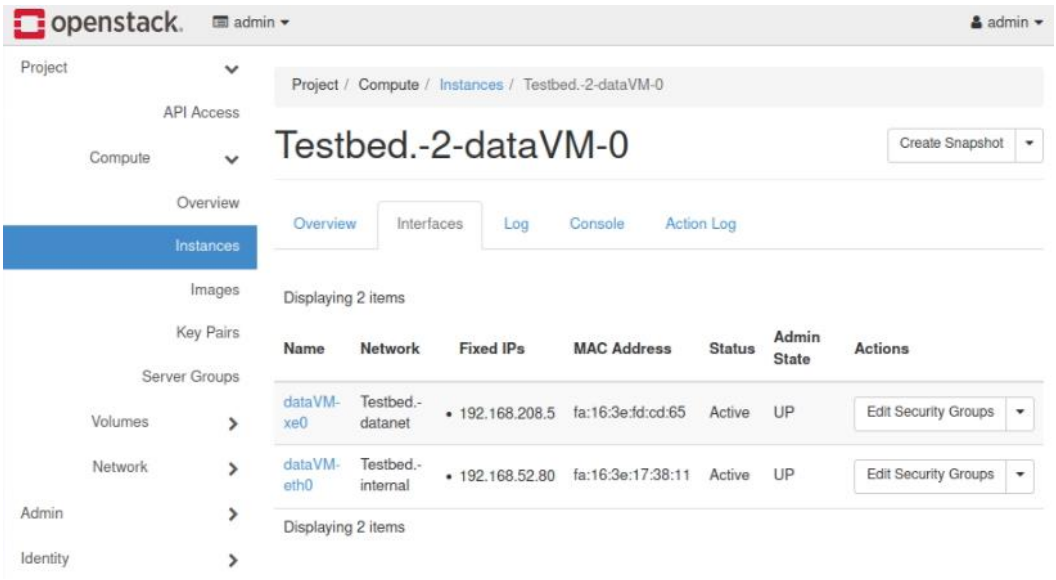

Figura 72. Interfaces de la VNF **dataVM.**

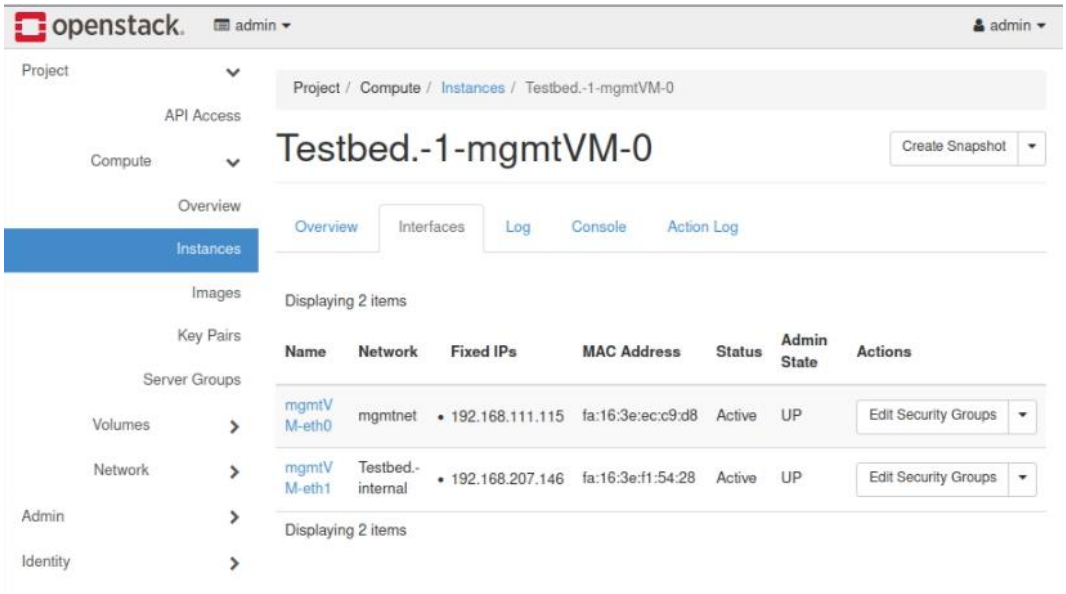

Figura 73. Interfaes de la VNF **mgmtVM.**

Por último, se puede verificar el correcto funcionamiento de las VNFs instanciadas, por medio del parámetro **Topología** ubicado en la plataforma OpenStack que muestra un diagrama de red, el cual presenta a cada VNF desplegada a traves de OSM. Este diagrama, indica el direccionamiento de red que posee cada VNF y como se mencionó en la sección 4.2 cada VNF posee dos redes, una externa (**datanet**) y otra interna (**internal**) las que permiten conectividad con el **Testbed.**

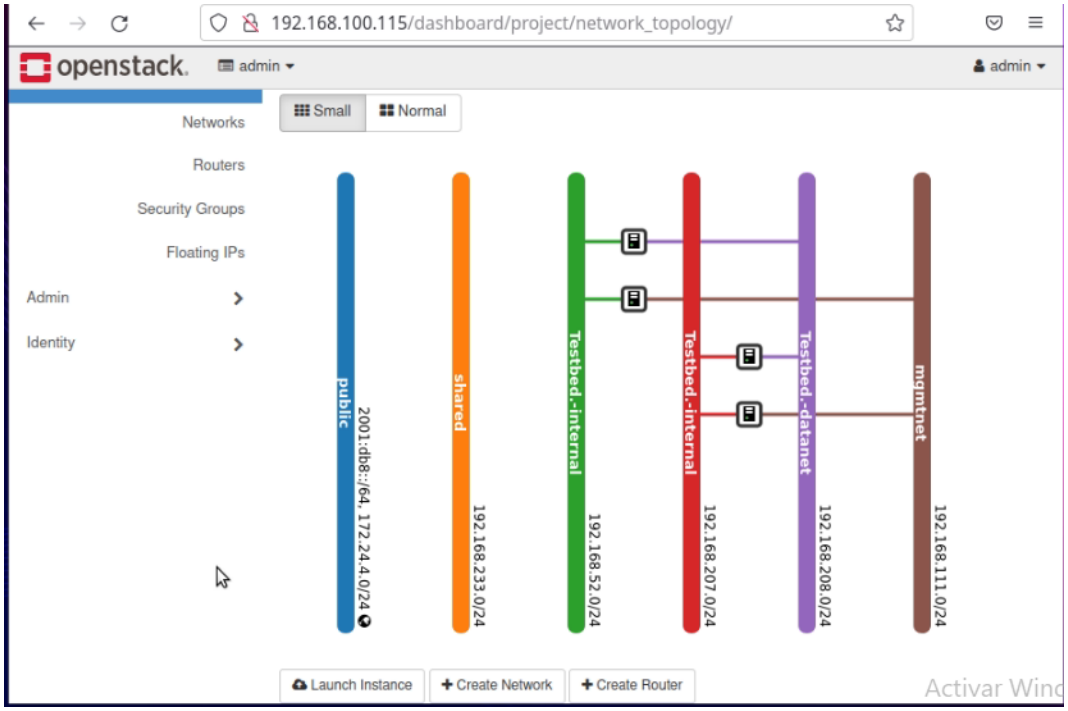

Figura 74. Topología de red en DevStack.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Con el estado del Arte realizado en este proyecto, fue posible identificar la documentación existente referente a la virtualización de funciones de red (NFV), el cual ayudo a comprender el funcionamiento de la tecnología siguiendo el modelo de referencia definido por la ETSI, así como también el análisis a detalle de la arquitectura, capacidad y puesta en marcha de la tecnología NFV, como también su impacto en el futuro de las comunicaciones, además, con la investigación bibliográfica realizada de las distintas plataformas OSM, OpenDaylight y OpenStack se pudo elaborar un Testbed funcional para lograr virtualizar al menos una función de red.

Este proyecto se enfocó principalmente en la investigación de la plataforma Open Source Mano (OSM) que fue diseñada por la ETSI para la investigación y despliegue de tecnología NFV. Por lo que, siendo una plataforma accesible, se pudo fomentar e indagar sobre un entorno de orquestación de recursos de computación en la nube, que proporciona trazas realistas de las cargas de trabajo, modelado y despliegue de los servicios de red, mediante APIs y fuentes de código abierto para uso de la tecnología NFV.

Por lo tanto, para conseguir una solución NFV se implementó una topología hiperconvergente que soporta varias plataformas Open Source, es decir, con las plataformas OpenStack y OpenDaylight se obtuvo un ambiente de red controlado, debido a que, al existir conectividad entre ellas se pudo integrar el bloque VIM y NFVI con OSM.

Al momento de implementar la topología de red para el Testbed NFV, fue necesario desplegar máquinas virtuales con requisitos normales en memoria RAM y almacenamiento, debido a que las instancias VNFs que crea OSM no necesitan mucha capacidad, ya que por medio de la SDN desplegada se toma recursos de servidores estándar.

Por otro lado, para lograr conectar la SDN con la plataforma OSM fue importante conocer sobre el parámetro DataPath ID, el cual contiene el ID de instancia y la direccion MAC bridge del OVS, es decir, por medio de estos parámetros OSM

logra detectar la red definida por software, para posteriormente conectarse con ella; así puede utilizar todos los recursos que ofrece, y fortalecer las instancias VNFs. Además, en el controlador ODL se despliega la topología de la red SDN, donde se debe mostrar a la máquina virtual de OSM, por lo cual, debe haber paquetes openflow comunicándose, indicando la conectividad entre las plataformas.

Por último, se integró el bloque VIM a OSM con la plataforma OpenStack, ya que por medio de sus caracteristicas permite crear, administrar y gestionar VNFs. Es decir, con DevStack se implementó software a través de una imagen Ubuntu para virtualizar funciones de red, que son capaces de ejecutarse sobre el NFVI, de modo que, las VNFs son las funciones de red virtualizadas que usan máquinas virtuales para su despliegue.

Para instanciar los servicios de red virtualizados en OSM se seleccionó el despliegue de VNFs simples, que se conectan a través de un descriptor de enlace virtual denominado VL que, por medio, de un paquete VNF es posible instanciar y programar las funciones de red. Este proceso se lo puedo realizar mediante imágenes de VM propias para el bloque VIM en este caso OpenStack.

Además, es importante conocer los parámetros que presenta OSM para realizar y entender la virtualización de las funciones de red, es decir, los parámetros VDU, CP y VL que permiten crear un ambiente virtual de las funciones de red, ya que el VDU crea software donde se aloja una función de red específica que comprende las capacidades físicas de un ordenador, el VL se encarga de conectar los distintos VDUs y el CP se utiliza para crear una conexión punto a punto entre los VLs con VDUs.

La plataforma DevStack crea, administra y elimina instancias, debido a que se asocian con los recursos virtualizados ya desplegados, se encarga de la gestión, rendimiento y fallos en el hardware, software y recursos virtuales desplegados. Ya que utiliza northboud APIs para exponer los recursos físicos y virtuales disponibles al sistema de gestión de VNFs (VNF Manager).

El desarrollo del Testbed de virtualización de funciones de red, consta de un conjunto de plataformas abiertas y experimentales desarrolladas a un nivel estandarizado, integradas en una infraestructura experimental, confiable y

diversificada, la cual cubre aspectos importantes del estado del arte en redes y aplicaciones como la programabilidad, seguridad e interfuncionamiento. De esta manera el despliegue de una infraestructura integrada en un banco de pruebas tiene como objetivo habilitar, nutrir y respaldar los diferentes conceptos utilizados al momento de desarrollar servicios, aplicaciones o crear un middleware de control de última generación basado en tecnología NFV.

Recomendaciones

En este apartado se sugiere nuevas investigaciones para la virtualización de funciones de red, ya que debido a la experiencia obtenida con tecnología NFV es posible proponer futuros proyectos para la carrera en telecomunicaciones.

En el presente documento, no se pudo investigar completamente los módulos de monitoreo de métricas que permiten verificar si aumenta la demanda de usuarios, direccionamiento, o recursos en la plataforma OSM, con el módulo MON de OSM permite el diseño de escenarios muy interesantes además de la creación de instancias dinámicas que controlan el flujo de datos en una arquitectura NFV. OSM MON presenta módulos de colector denominados “mon-colllector” que se encarga de recopilar información de métricas o requerimientos que necesita el sistema siempre y cuando se tenga especificado un bloque de arquitectura VIM o NFVI.

Por otro lado, es posible administrar controladores SDN externos para realizar una topología más robusta con conectividad de red subyacente en el plano de datos, en este caso para el bloque VIM. Por lo que, se debe crear una red de administración con un servidor DHCP propio para este escenario, el cual garantiza una red de administración accesible por OSM; el cual, por medio de un descriptor VCA(Juju) se configura VNFs ya ejecutadas, que mediante una red aislada de OpenStack funciona como administrador hacia diferentes proveedores, los cuales controlan sus VNFs y el administrador todo el escenario, por ejemplo, crear una red de proveedores identificada por una interfaz virtual que por medio de una VLAN se comunica con OSM.

Además, es posible el despliegue de varias funciones de red como servicio utilizando Open Source Mano (OSM), ya que el testbed implementado brinda la infraestructura adecuada para virtualizar servidores, por ejemplo, como trabajo futuro se puede crear un Firewall utilizando una VNF propia para definir reglas de Firewall, se lo puede realizar al cargar un paquete VNF en OSM, el cual contiene los archivos y metadatos necesarios para un Firewall. Por otro lado, se puede crear la VNF Firewall, lo que implica crear una instancia de la VNF para Firewall y especificar dentro de ella los parámetros de configuración necesarios, como el número de instancias que necesitara el servidor, la topología de red y las reglas de Firewall, es decir, especificar el flujo de tráfico y las políticas de seguridad que debe aplicar el Firewall, cabe recalcar que, una ves creado y configurado el servidor por medio de OSM es posible gestionarlo y organizarlo según sea necesario, así como también, modificar la configuración y eliminarlo cuando ya no se necesite. También es importante elegir el software adecuado para el servidor de Firewall ya que debe ser compatible con el sistema MANO de OSM de código abierto, estos pueden ser iptables, nftables y firewalld.

Por otro lado, se puede crear un entorno para la tecnología 5G en Open Source MANO (OSM), por lo que, es importante seguir las siguientes recomendaciones, para pueda servir como trabajos a futuro, es decir, es necesario crear y desplegar funciones de red virtualizadas (VNF) necesarias para la red 5G. Esto podría incluir VNFs para funciones de red central, como la pasarela de paquetes, o para funciones de red de acceso de radio, como el controlador de la estación base, la red de acceso radioeléctrico (RAN) y la computación móvil periférica (MEC). También, es importante que con OSM se configure y gestione las VNFs de la red virtualizada en su conjunto. Esto podría incluir tareas como la configuración del enrutamiento del tráfico, la configuración de políticas de seguridad y el escalado de recursos según sea necesario. Por último, por medio de OSM es necesario supervisar el rendimiento y el estado de la red 5G, que soluciona cualquier problema que pueda surgir.

REFERENCIAS

Abdelsalam, A., Clad, F., Filsfils, C., Salsano, S., Siracusano, G., & Veltri, L. (2020). *Implementation of Virtual Network Function Chaining through Segment Routing in a Linux-based NFV Infrastructure*. (671566).

Arocha, R. G. (2017). *Propuesta de procedimiento para la evaluación de entornos NFV Propuesta de procedimiento para la evaluación de entornos NFV*.

Avilés, J. (2017). *Análiss, diseño e implementación de un escenario didáctico NFV utilizando herramientas Open Source.* (Tesis de Maestría). Universidad Politécnica de Madrid, España.}

Bari, F., Chowdhury, S., Ahmed, R., Boutaba, R., y Duarte, O. (2016). Orchestrating virtualized network functions. *IEEE Transactions on Network and Service Management*, *13*(4), 725-739.

Barrado, A. (2018). *Estudio de la plataforma de orquestación NFV OSM.* (Tesis de Pregrado). Universidad Carlos III, Madrid.

Bernal, I., & Mejia, D. (2016). *Las Redes Definidas por Software y los Desarrollos Sobre Esta Temática en la Las Redes Definidas por Software y los Desarrollos Sobre Esta Temática en la Escuela Politécnica Nacional Software Defined Networks and Developments in This Area at the National Polytechnic School*. (July).

Bhagwat, M., Clarke, D., Eardley, P., Elizondo, A., García, G., Gronsund, P., Hoban, A., Manning, S., Nakamura, T., Ramón, F., y Reid, A. (2018). _OSM EUAG: EXPERIENCE WITH NFV ARCHITECTURE, INTERFACES, AND INFORMATION MODELS. _ ETSI. Recuperado de https://osm.etsi.org/images/OSM_White_Paper_Experience_implementing_NFV_specs_final.pdf

Carbonel, A., y Canales, M. (2018). *Estudio de la virtualización de funciones de red (NFV) y aplicación del encadenado de funciones (SFC) para un diseño flexible de red.* (Tesis de Maestría). Universidad de Zaragoza, España.

Chen, L., Huang, W., Sui, A., Chen, D., & Sun, C. (2017). The online education

platform using Proxmox and noVNC technology based on Laravel framework. 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS).

Chiosi, M., Wright, S., Clarke, D., Willis, P., (2013). Network Functions Virtualisation: Network Operator Perspectives on Industry Progress. Recuperado de https://portal.etsi.org/NFV/NFV_White_Paper2.pdf

Chou, L., Tseng, C., Tseng, C., & Chen, Y. (2016). *The Novel SDN Testbed with Virtual Network Functions Placement*.

Cordero, C. (2017). *Diseño y despliegue de Funciones de Red Virtualizadas (NFV) usando Redes Definidas por Software (SDN) dentro de una infraestructura virtual, aplicando balanceo de carga y seguridad distribuida en IPv6.*

Cox, J. H., Chung, J., Donovan, S., Ivey, J., Clark, R. J., Riley, G., & Owen, H. L. (2017). *Advancing Software-Defined Networks: A Survey. IEEE Access, 5, 25487–25526.* doi:10.1109/access.2017.2762291

Eljack, A. H., Hassan, A. H. M., & Elamin, H. H. (2019). *Performance Analysis of ONOS and Floodlight SDN Controllers based on TCP and UDP Traffic. 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE).* doi:10.1109/iccceee46830.2019.9071189

Fernández, O. (2018). *Análisis de la virtualización de las funciones de red y sus aplicaciones* (Tesis de Doctorado). Universidad Central" Marta Abreu" de Las Villas, Cuba.

Garzón, C. y Atehortua, K. (2018). *Implementación e integración de la virtualización de funciones de red, computación en la nube y las redes definidas por software, para la administración y orquestación de una infraestructura como servicio.* (Tesis de Pregrado). Universidad Católica de Pereira, Colombia.

Gladys, M. (2019). I*mplementación de una infraestructura de IT virtual para el data center de la facultad de ingeniería en ciencias aplicadas, en la Universidad Técnica del Norte, Ecuador.*

Herrera, J. y Vega, J. (2016). Network functions virtualization: A survey. *IEEE*

Latin America Transactions, 14(2), 983-997.

Husni, E., & Bramantyo, A. (2018). *Design and Implementation of MPLS SDN Controller Application based on OpenDaylight. 2018 International Symposium on Networks, Computers and Communications (ISNCC).* doi:10.1109/isncc.2018.8530900.

Intel, Brocade, Cyan, Red Hat y Telefónica. (2017). *Implementación de la virtualización de funciones de red optimizada global.* México: Intel.

Kaljic, E., Maric, A., Njemcevic, P., & Hadzialic, M. (2019). *A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking. IEEE Access, 7, 47804–47840.* doi:10.1109/access.2019.2910140

Karamichailidis, P., Choumas, K., y Korakis, T. (2019). Enabling multi-domain orchestration using Open Source MANO, OpenStack and OpenDaylight. En 2019 *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)* (pp. 1-6). IEEE.

Leyton, J. (2016). *Diseño de una red definida por software empleando una solución basada en software, para la infraestructura de cloud de la facultad de ingeniería en ciencias aplicadas (FICA)*. Universidad Técnica del Norte, Ecuador.

Lv, Z., & Xiu, W. (2019). Interaction of Edge-Cloud Computing Based on SDN and NFV for Next Generation IoT. *IEEE Internet of Things Journal*, *PP*(c), 1. https://doi.org/10.1109/JIOT.2019.2942719

Martínez, C. (2019). Desarrollo de APIs para escenarios SDN-NFV (UNIVERSIDAD POLITÉCNICA DE CARTAGENA ESCUELA). Recuperado de http://repositorio.upct.es/handle/10317/7967

Matta, I., Nabeel, A., & Yuefueg, W. (2016). Managing NFV using SDN and Control Theory. *Computer Science Department, Boston University*, *@bu*(.edu), 1113–1118.

Mauro, M. Di, Galatro, G., Longo, M., & Postiglione, F. (2018). IP Multimedia Subsystem in an NFV environment : availability evaluation and sensitivity analysis. *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 1–6.

Millán, R. y Esfandiari, S. (2014). *Qué es… NFV (Network Functions*

Virtualization). Recuperado de https://www.ramonmillan.com/documentos/networkfunctionsvirtualization.pdf

Nicolalde, W. (2021). *Diseño de un Sistema de Detección de Intrusos (IDS), basado en redes neuronales para una Red Definida por Software (SDN) en la Facultad de Ingeniería en Ciencias Aplicadas (FICA) de la Universidad Técnica del Norte* (Tesis de pregrado). Universidad Técnica del Norte, Ibarra, Ecuador.

Padma, V., & Yogesh, P. (2015). *Proactive failure recovery in OpenFlow based Software Defined Networks. 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN).* doi:10.1109/icscn.2015.7219846

Pattaranantakul, M., He, R., Zhang, Z., & Meddahi, A. (2018). *Leveraging Network Functions Virtualization Orchestrators to Achieve Software-Defined Access Control in the Clouds*. *XX*(XX), 1–14. https://doi.org/10.1109/TDSC.2018.2889709

Ramos, J. (2017). *Service Function Chaining en NFV : Evaluacion practica con OpenStack* (Universidad Carlos III de Madrid). Recuperado de https://e-archivo.uc3m.es/handle/10016/26092

Rehman, A. U., Aguiar, R. L., y Barraca, J. P. (2019). Network functions virtualization: The long road to commercial deployments. *IEEE Access*, 7, 60439-60464.

Reid, A., González, A., Elizondo, A., García, G., Xie, M., Grønsund, P., Willis, P., Eardley, P., Ramón, F., Tierno, A., Lombardo, F., Lavado, G., Shuttleworth M., Harper, M., Marchetti, M., Little V., Boyer, C., y Almagia, S. (2019). OSM SCOPE, FUNCTIONALITY, OPERATION AND INTEGRATION GUIDELINES. ETSI. Recuperado de https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf

Reid, A., Marsico, A., ElSawaf, A., García, G., Ramón, F., Grønsund, P., y Sheikh, S. (2020). Deployment and Integration of OSM. ETSI. Recuperado de

https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Deployment_and_Integration.pdf

Riggio, R., Rasheed, T., & Granelli, F. (2014). *EmPOWER : A Testbed for Network Function Virtualization Research and Experimentation*.

Saghir, A., & Masood, T. (2019). Performance Evaluation of OpenStack Networking Technologies. 2019 International Conference on Engineering and Emerging Technologies (ICEET).

Sandoval, C. (2018). *Implementación de un clúster controlador de SDN basado en un framework de software libre para la infraestructura cloud de la facultad de ingeniería en ciencias aplicadas*. Universidad Técnica Del Norte.

Sandoval, J. (2018). *Virtualización de las Funciones de Red. Las redes futuras y la virtualización de las funciones de red (NFV).* Recuperado de file:///D:/Descargas/NFV-Tema1-LasredesfuturasylaNFV.pdf

Schneider, S., Peuster, M., Tavernier, W., & Karl, H. (2018). *A Fully Integrated Multi-Platform NFV SDK. 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).* doi:10.1109/nfv-sdn.2018.8725794

Stallings, W. (2015). *Foudations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*. Indiana: Pearson Education, Inc.

Tatang, D., Quinkert, F., Frank, J., Ropke, C., & Holz, T. (2017). *SDN-Guard: Protecting SDN controllers against SDN rootkits. 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).* doi:10.1109/nfv-sdn.2017.8169856

Vilalta, R., Mayoral, A., Mu, R., Casellas, R., & Mart, R. (2015). *The SDN / NFV Cloud Computing Platform and Transport Network of the ADRENALINE Testbed*.

Yilma, G., Yousaf, Z., Sciancalepore, V., y Costa-Perez, X. (2020). Benchmarking open source NFV MANO systems: OSM and ONAP. *Computer Communications, 161*, 86-98.

Zurita, A. (2019). *Estudio de soluciones para la orquestación de recursos Cloud / NFV* (Universidad Politécnica de Madrid). Recuperado de

http://oa.upm.es/54213/1/TFG_ARTURO_ZURITA_SANCHEZ.pdf

Zuo, Q., Chen, M., Ding, K., & Xu, B. (2014). *On generality of the data plane and scalability of the control plane in software-defined networking. China Communications, 11(2), 55–64.* doi:10.1109/cc.2014.6821737

ANEXOS

ANEXO 1. Manual de uso del Testbed de virtualizaciones de funciones de red NFV.

A continuación, se detalla el procedimiento para usar el Testbed implementado en el Data Center de la FICA.

Al ingresar a la plataforma Proxmox VE mediante el url https://10.0.0.2:8006 se procede a encender las máquinas virtuales (***OSM-RN***) (***DEV-RN***) (***SRV-WEB-WN***) (***SRV-FTP-WN)*** (***OVS-WN***) (***ODL-WN***) creadas en el ambiente hiperconvergente de Proxmox como se puede observar en la figura 75.

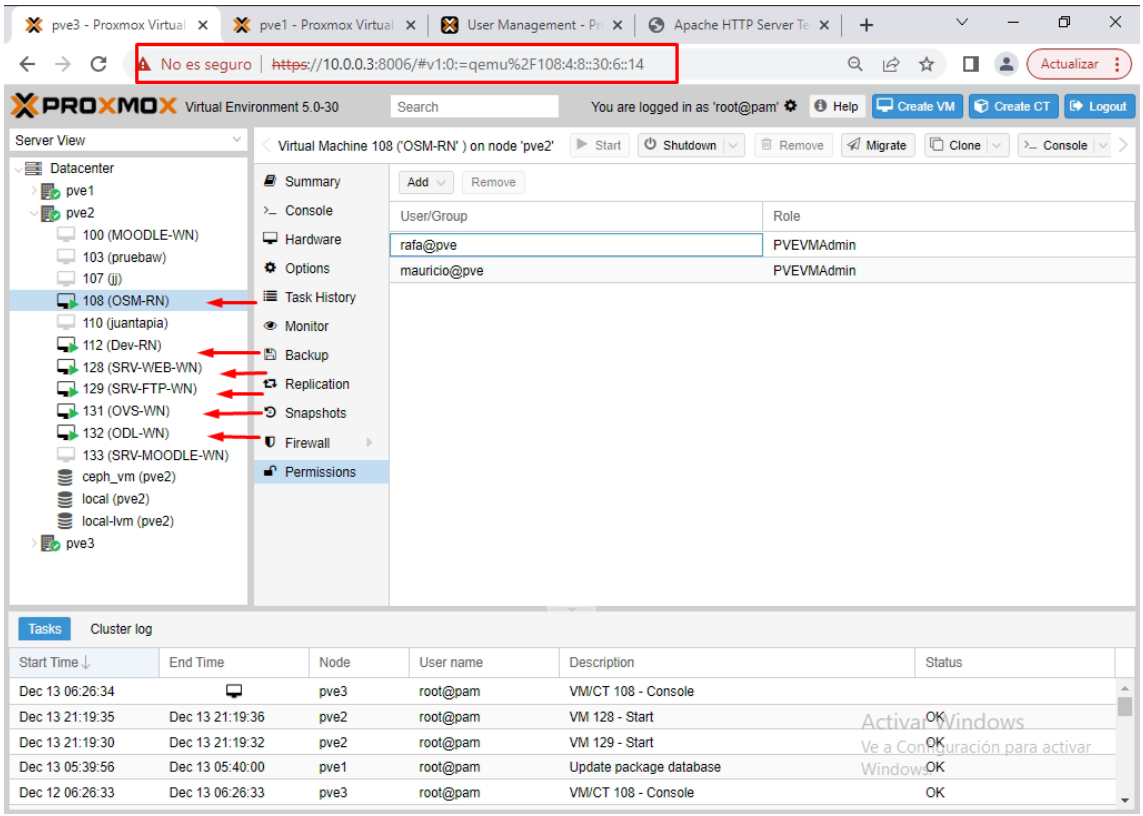

Figura 75. Plataforma Proxmox

Después que se hayan encendido las VMs mencionadas se ingresa a (***OVS-WN***) y (***ODL-WN***) para levantar la SDN, por lo que, en la máquina (***OVS-WN***) se configura la siguiente línea de comando ***ovs-vsctl set-controller ovs-br0 tcp:192.168.100.111:6653*** como se muestra en la figura 76, que permite identificar al controlador ODL con su puerto y dirección IP. Por otro lado, en la

máquina (***ODL-WN***) se ingresa al navegador web con el siguiente url *http://localhost:8080/index.html* para verificar la topología de la red definida por software desplegada como se muestra en la figura 77.

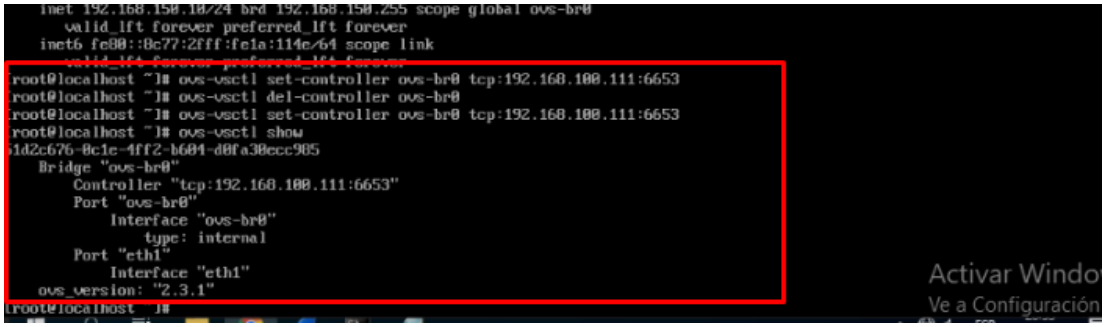

Figura 76. Comando en máquina OVS.

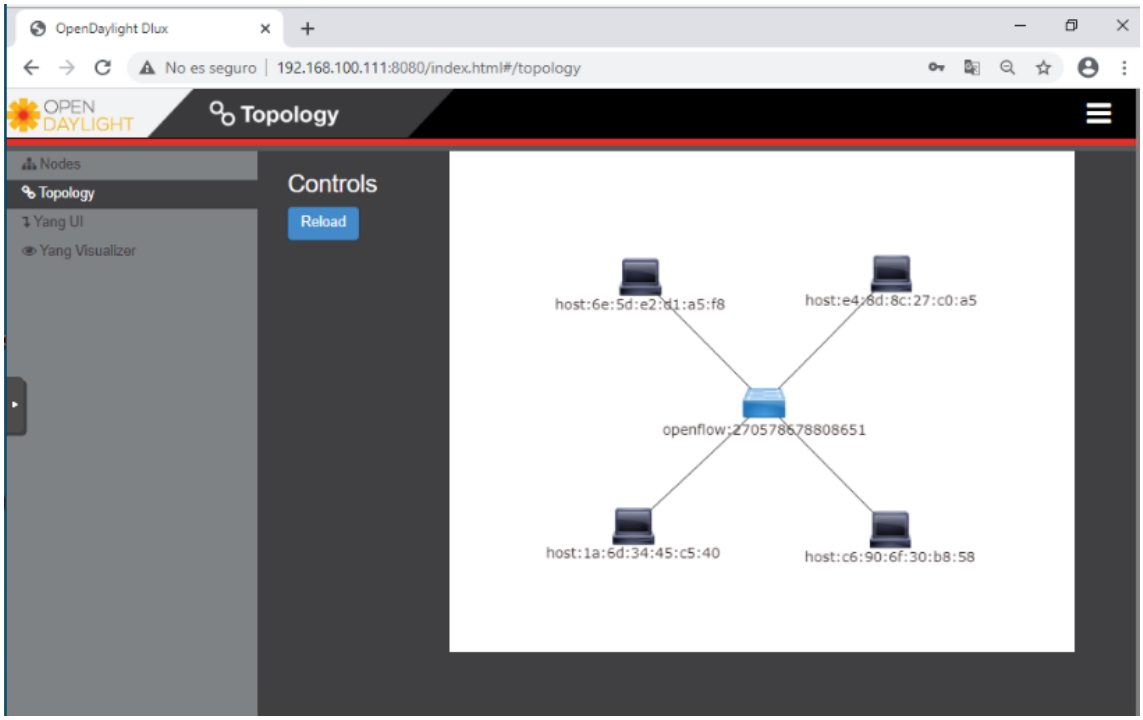

Figura 77. Topología SDN.

Una vez levantada la SDN, se procede a verificar en la plataforma (***OSM-RN***) el controlador SDN está conectado, por lo que se coloca en el navegador el url *http://192.168.100.114:login* posteriormente en **SDN Controller** se verifica el estado de la red el cual es **Enabled.** Como se observa en la figura 78.

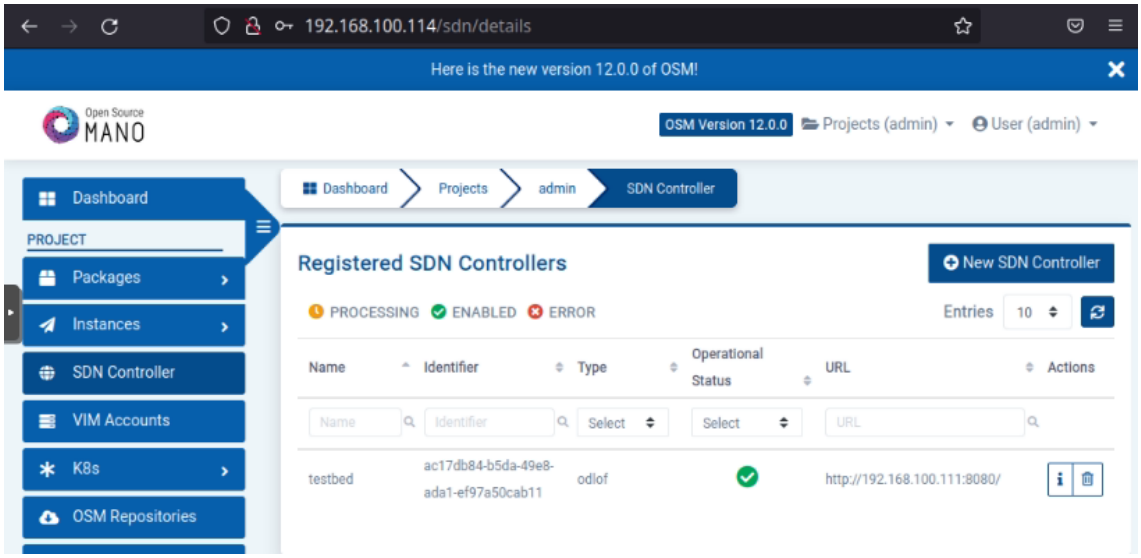

Figura 78. SDN Controller en OSM.

Ahora, para verificar si el bloque VIM se encuentra conectado, se procede a seleccionar la opción **VIM Accounts** y como resultado se obtiene que la plataforma DevStack esta como **Enabled**, es decir, al momento de iniciar la plataforma OpenStack esta se enlaza directamente con OSM como muestra la figura 79.

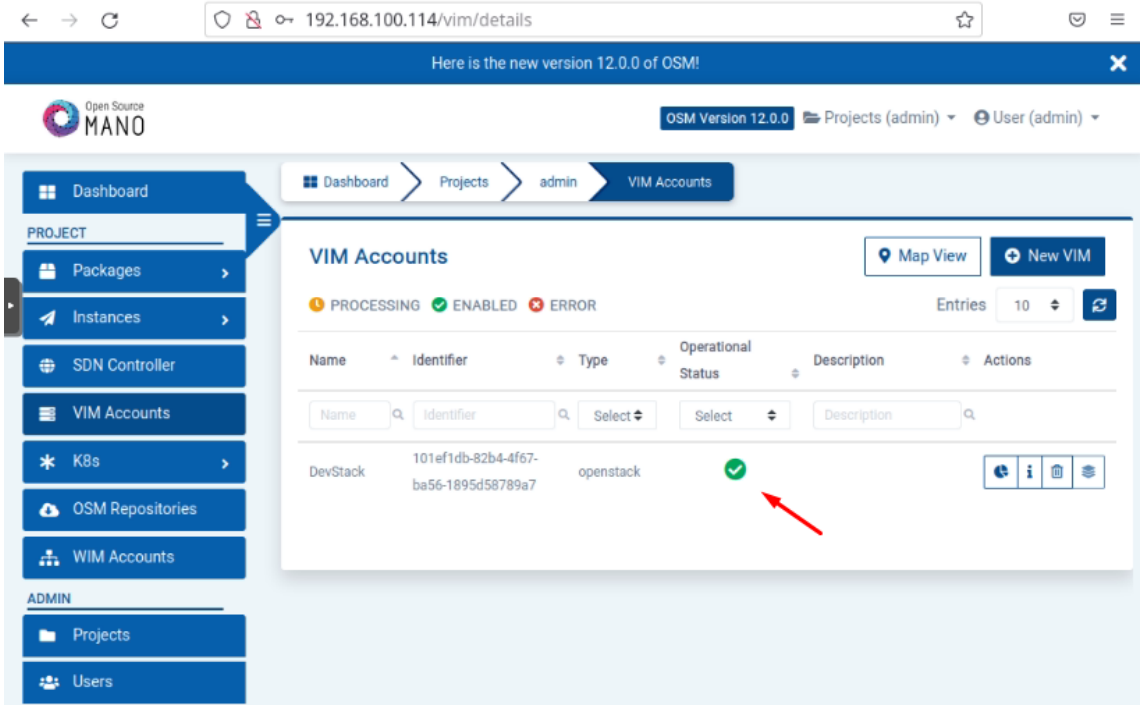

Figura 79. Plataforma OpenStack con OSM como bloque VIM.

Por lo tanto, las instancias ya desplegadas deben mantenerse en estado **Activo y Ejecutándose** por lo que, al dirigirse a la sección **Instances** en OSM se verifica que el Network Service denominado Testbed está **Configurado y Ejecutándose** como se muestra en la figura 80, es decir, las VNFs instanciadas se encuentran funcionando, esto se puede comprobar con el parámetro **topología**, donde se despliega una topología gráfica de todos los elementos que conforman una red NFV; esto se puede observar en la figura 81.

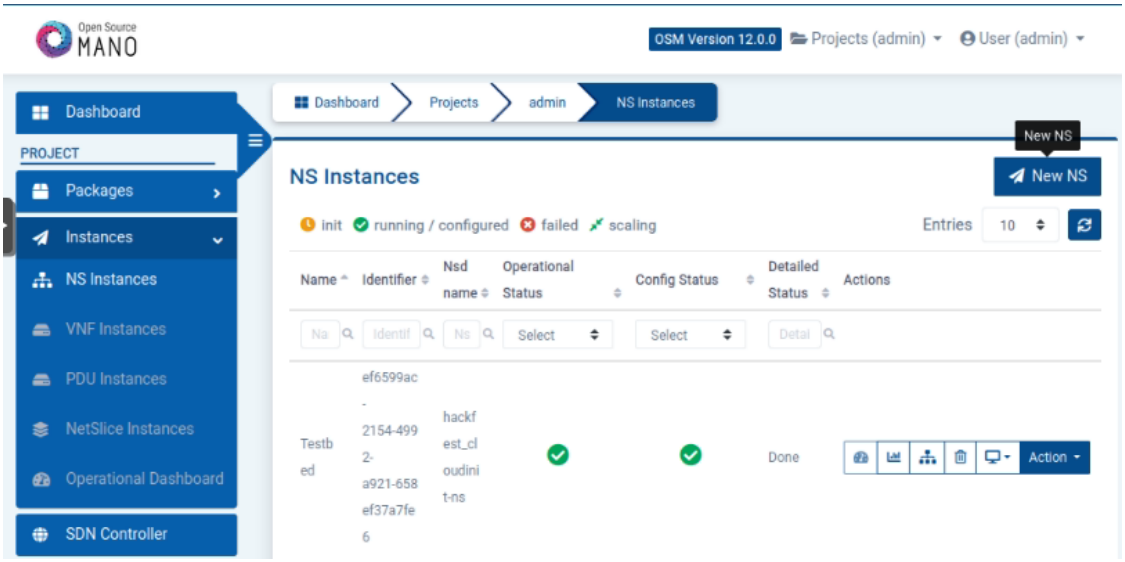

Figura 80. Network Service Testbed.

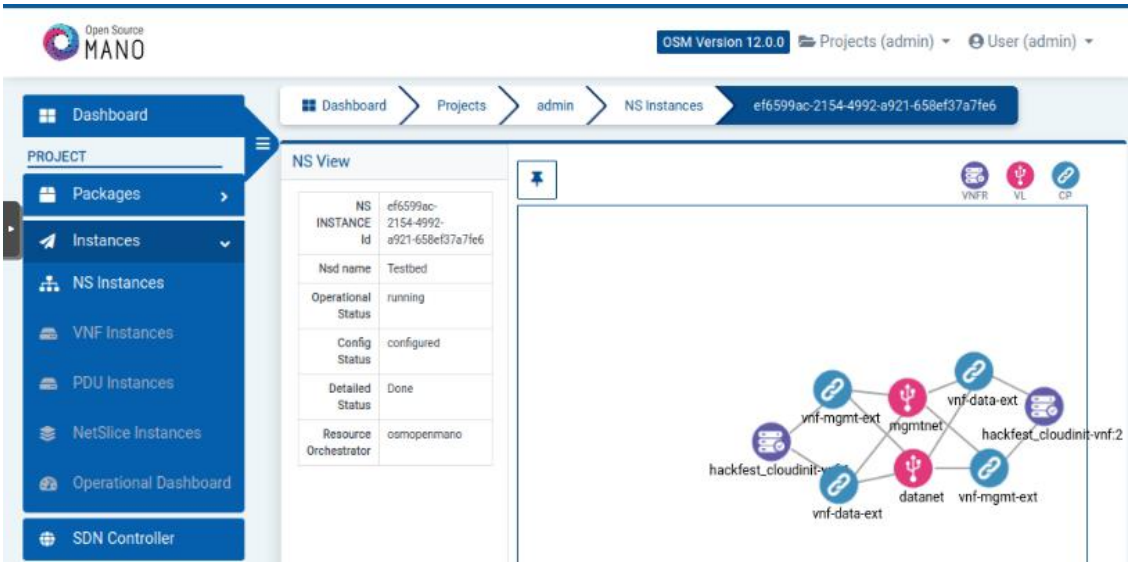

Figura 81. Topología NS Testbed.

Por último, se puede monitorear y configurar cada VNF a través del bloque VIM, por lo que, en la plataforma (***Dev-RN***) mediante el url *http://192.168.100.115/dashboard* en la sección Instances se muestra todas las VNFs creadas y por medio de la opción **Action** es posible su control, como se muestra en la figura 82.

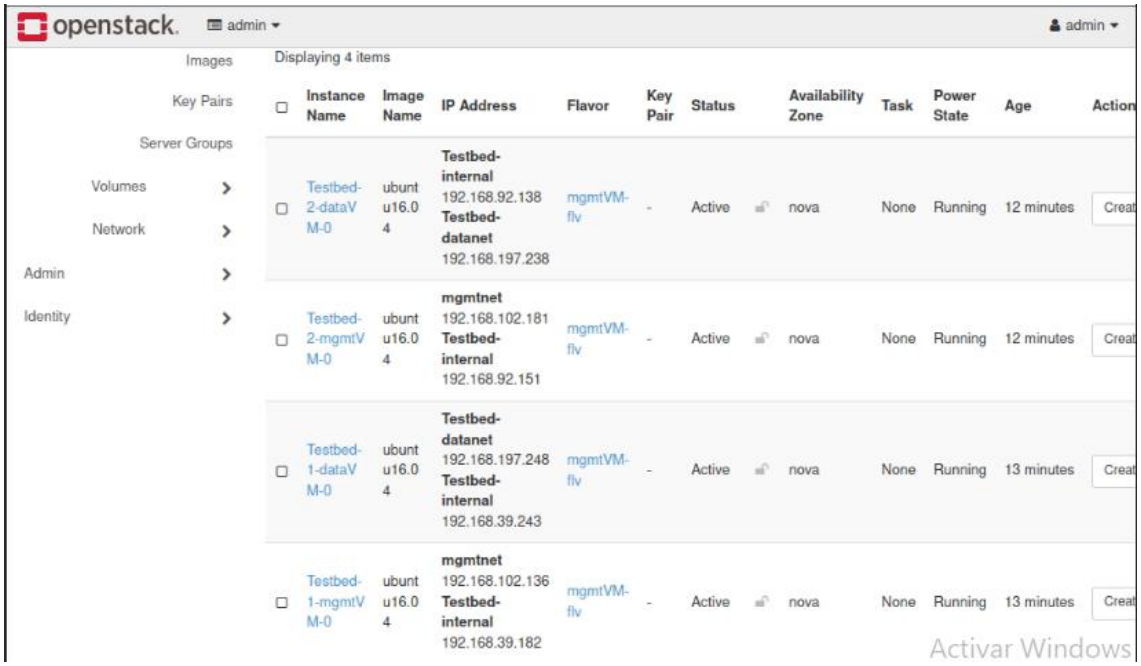

Figura 82. VNFs a través de la plataforma DevStack.

ANEXO 2. ARCHIVO YAML usado en el proyecto

```
vnfd:
  description: A VNF consisting of 2 VDUs connected to an internal VL, and one VDU
    with cloud-init
  df:
  - id: default-df
    instantiation-level:
    - id: default-instantiation-level
      vdu-level:
      - number-of-instances: 1
        vdu-id: mgmtVM
      - number-of-instances: 1
        vdu-id: dataVM
    vdu-profile:
    - id: mgmtVM
      min-number-of-instances: 1
    - id: dataVM
      min-number-of-instances: 1
  ext-cpd:
  - id: vnf-mgmt-ext
    int-cpd:
      cpd: mgmtVM-eth0-int
      vdu-id: mgmtVM
  - id: vnf-data-ext
    int-cpd:
      cpd: dataVM-xe0-int
      vdu-id: dataVM
  id: hackfest_cloudinit-vnf
  int-virtual-link-desc:
  - id: internal
  mgmt-cp: vnf-mgmt-ext
  product-name: hackfest_cloudinit-vnf
  sw-image-desc:
  - id: ubuntu16.04
    image: ubuntu16.04
    name: ubuntu16.04
  vdu:
  - cloud-init-file: cloud-config.txt
    id: mgmtVM
```

```
  int-cpd:
  - id: mgmtVM-eth0-int
    virtual-network-interface-requirement:
    - name: mgmtVM-eth0
      position: 1
      virtual-interface:
        type: PARAVIRT
  - id: mgmtVM-eth1-int
    int-virtual-link-desc: internal
    virtual-network-interface-requirement:
    - name: mgmtVM-eth1
      position: 2
      virtual-interface:
        type: PARAVIRT
  name: mgmtVM
  sw-image-desc: ubuntu16.04
  virtual-compute-desc: mgmtVM-compute
  virtual-storage-desc:
  - mgmtVM-storage
- id: dataVM
  int-cpd:
  - id: dataVM-eth0-int
    int-virtual-link-desc: internal
    virtual-network-interface-requirement:
    - name: dataVM-eth0
      position: 1
      virtual-interface:
        type: PARAVIRT
  - id: dataVM-xe0-int
    virtual-network-interface-requirement:
    - name: dataVM-xe0
      position: 2
      virtual-interface:
        type: PARAVIRT
  name: dataVM
  sw-image-desc: ubuntu16.04
  virtual-compute-desc: dataVM-compute
  virtual-storage-desc:
  - dataVM-storage
```

```
version: '1.0'
virtual-compute-desc:
- id: mgmtVM-compute
  virtual-cpu:
    num-virtual-cpu: "1"
  virtual-memory:
    size: "1.0"
- id: dataVM-compute
  virtual-cpu:
    num-virtual-cpu: "1"
  virtual-memory:
    size: "1.0"
virtual-storage-desc:
- id: mgmtVM-storage
  size-of-storage: "10"
- id: dataVM-storage
  size-of-storage: "10"
```