ESCUELA TECNICA SUPERIOR DE
INGENIERIA DE TELECOMUNICACION
UNIVERSIDAD POLITECNICA DE CARTAGENA

Trabajo Fin de Máster

Net2Plan-OpenStack: un plugin de net2plan para interactuar con OpenStack

AUTOR: Manuel Hernández Bastida
DIRECTOR: Pablo Pavón Mariño
Septiembre de 2020

Autor	Manuel Hernández Bastida
E-mail del Autor	manu.hernandez.bastida@gmail.com
Director	Pablo Pavón Mariño
E-mail del director	pablo.pavon@upct.es
Título del TFG	Net2Plan-OpenStack: un plugin de net2plan para interactuar con OpenStack
Descriptores	Java, REST, OpenStack, Cloud, VM

Resumen

El siguiente proyecto fin de estudios muestra el desarrollo de una herramienta para el control de un despliegue de OpenStack para crear nubes privadas y públicas.

El proyecto se desarrolla sobre una herramienta ya existente, Net2Plan, como un plugin adicional para permitir la interacción con un sistema OpenStack.

A lo largo del proyecto se mostrarán la estructura de la herramienta, como usarla y algunas de sus funcionalidades, un caso de uso y posibles líneas de expansión.

Titulación	MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de presentación	Septiembre 2020

Agradecimientos especiales para
el grupo GIRTEL de la UPCT
por su ayuda a largo de este tiempo.

A todos mis amigos y familiares
por el apoyo que siempre me habéis dado.

Manuel Hernández Bastida

Resumen

El siguiente proyecto fin de estudios muestra el desarrollo de una herramienta para el control de un despliegue de OpenStack para crear nubes privadas y públicas.

El proyecto se desarrolla sobre una herramienta ya existente, Net2Plan, como un plugin adicional para permitir la interacción con un sistema OpenStack.

A lo largo del proyecto se mostrarán la estructura de la herramienta, como usarla y algunas de sus funcionalidades, un caso de uso y posibles líneas de expansión.

Abstract

The following end-of-studies project shows the development of a tool to control an OpenStack deployment to create private and public clouds.

The project is developed on an existing tool, Net2Plan, as an additional plugin to allow interaction with an OpenStack system.

Throughout the project the structure of the tool will be shown, how to use it and some of its functionalities, a use case and possible expansion lines.

Índice

Índice de figuras

Capítulo 1. Introducción

En la actualidad, la computación en la Nube, conocida como "Cloud Computing", ofrece el despliegue de herramientas y servicios escalables que permiten un uso de los recursos en función de las necesidades de los usuarios que los utilizan.

Servicios como los que ofrecen Dropbox o Gmail, utilizan la computación en la Nube debido a sus ventajas como permitir el acceso y disponibilidad de nuestros archivos personales a través de aplicaciones web en cualquier lugar y momento con acceso a Internet.

Sin embargo, la computación en la Nube no se limita a ofrecer servicios y aplicaciones web, en el paradigma NFV[1] también tiene su lugar, permitiendo el despliegue de VNFs[2] que sustituyan los dispositivos hardware, como routers o firewalls, que realizan las funciones de red ahorrando así costes técnicos y facilitando la integración de nuevas funcionalidades.

En este ecosistema, en el que la Nube tiene tanto peso surge OpenStack [1], un proyecto de computación en la Nube que proporciona la infraestructura como servicio (IaaS).

OpenStack es un software libre y de código abierto para la creación de nubes privadas y públicas. Como un sistema operativo en Nube, permite el control de los recursos de computación, almacenamiento, redes…

Para gestionar sus servicios, OpenStack ofrece diferentes posibilidades como una interfaz gráfica, también conocida como Dashboard, que utiliza el componente de Horizon de OpenStack para ofrecer a los usuarios la posibilidad de manejar de manera sencilla los principales elementos de un despliegue, la línea de comandos o las APIs REST[3] de los componentes.

1.1 Motivación

Tras comprender la importancia de la computación en la Nube, conocer la existencia de OpenStack y su potencial, surge la idea de realizar el siguiente proyecto con el objetivo de ampliar las funcionalidades que el Dashboard no ofrece, realizar una herramienta que permita el control y manejo de los recursos en los despliegues en producción de OpenStack, ya que es vital para sacar un rendimiento óptimo y asegurar la calidad de los servicios que se puedan ofrecer.

La herramienta debería ser capaz de obtener los datos necesarios para que, en futuro, los usuarios pudieran disfrutar de algoritmos de optimización, que permitan sacarles el máximo partido a sus despliegues.

1.2 Objetivos

Las motivaciones que nos han llevado a desarrollar el siguiente proyecto dan lugar a una serie de objetivos que cumplir para obtener un resultado satisfactorio del mismo.

---

[1] NFV: (Network Function Virtualization) consiste en trasladar las funciones de red de dispositivos HW dedicados a servidores virtuales. Esto supone la consolidación de múltiples funciones en un solo servidor físico.

[2] VNF: (Virtualized Network Functions) función de red virtualizada. Es una tarea virtualizada realizada anteriormente por hardware propietario y dedicado. Por ejemplo, router, switch o firewall.

[3] API REST: es una interfaz para conectar varios sistemas basados en el protocolo HTTP y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.

Estos objetivos son los siguientes:

1. **Investigar y conocer OpenStack**: para adquirir los conocimientos suficientes para poder entender su funcionamiento básico.
2. **Desarrollar una herramienta en Java**: para conseguir ampliar o facilitar las funcionalidades que ofrece con una interfaz gráfica para el usuario.
3. **Realizar pruebas con diferentes situaciones**: para demostrar el correcto funcionamiento.
4. **Redactar una memoria**: documentando las partes importantes para que el trabajo pueda continuarse.

1.3 Net2Plan[2]

Para el desarrollo de la herramienta, hemos decido incluirla en Net2Plan a modo de plugin. Net2Plan es un software libre y de código abierto, programado en Java, que permite a los usuarios la planificación, optimización y evaluación de redes de comunicación.

Net2Plan gracias a su representación de red abstracta y sus componentes como los nodos, nos permiten incluir los elementos red de nuestros despliegues, que podrían utilizarse más tarde para el desarrollo de algoritmos.

Además, podemos utilizar su GUI para la representación de los datos de los elementos, y de manera intuitiva y visual realizar diferentes funcionalidades.

1.4 Capítulos

En esta sección del capítulo, con el fin de facilitar la lectura del documento, se presenta el contenido de los demás capítulos que se encuentran en la memoria:

Capítulo 1. Introducción

Capitulo presentación del proyecto, encargado de explicar el contexto en el que se desarrolla, la principal motivación del proyecto y resumen del contenido de este por capítulos.

Capítulo 2. OpenStack

En este capítulo profundizamos en OpenStack, estructura básica de un despliegue, los servicios y componentes utilizados, las versiones que existen, y el proyecto RDO[3] PackStack para RHEL[4], utilizado para la prueba de concepto de este proyecto.

Capítulo 3. Arquitectura de la herramienta

Este capítulo muestra una visión global de la arquitectura de la herramienta, como es su integración con Net2Plan, las estructuras de paquetes y clases que se han utilizado para el desarrollo del plugin, una presentación de la librería OpenStack4j[5] y las llamadas REST propias utilizadas.

Capítulo 4. Manual de usuario

En el capítulo de manual de usuario, encontraremos una guía de instalación de la herramienta, así como una vista previa de su UI donde se muestran imágenes con explicaciones de las principales funciones que la herramienta es capaz de realizar, como el acceso a los despliegues OpenStack o la monitorización de los elementos de red.

Capítulo 5. Caso de uso

En este capítulo, tras la descripción del entorno donde se realiza, se muestra un caso de uso de la herramienta dividido en tres partes: (1) creación de los elementos de red necesarios para establecer comunicación entre dos VM[4] y verificación de la conectividad, (2) monitorización de algunos de los parámetros de los elementos de la red y (3) una muestra de la gestión de los recursos de un despliegue.

Capítulo 6. Conclusiones

En este apartado del proyecto, el autor obtiene unas conclusiones sobre el mismo, permitiéndole discernir entre diferentes líneas de desarrollo futuras.

Capítulo 7. Bibliografía

En este capítulo del proyecto se encuentran las referencias utilizadas para la obtención de información y desarrollo del proyecto.

[4] VM: Una máquina virtual (VM) es un entorno virtual que funciona como sistema informático virtual con sus propias características (CPU, memoria, interfaz de red y almacenamiento), pero se crea en un sistema de hardware físico.

Capítulo 2. OpenStack

OpenStack es un sistema operativo en la nube que controla grandes grupos de recursos de computación, almacenamiento y redes a través de un centro de datos, todos administrados y aprovisionados a través de API con mecanismos de autenticación comunes (Fig. 1).

También está disponible un panel de control, que brinda a los administradores el control y permite a sus usuarios aprovisionar recursos a través de una interfaz web.

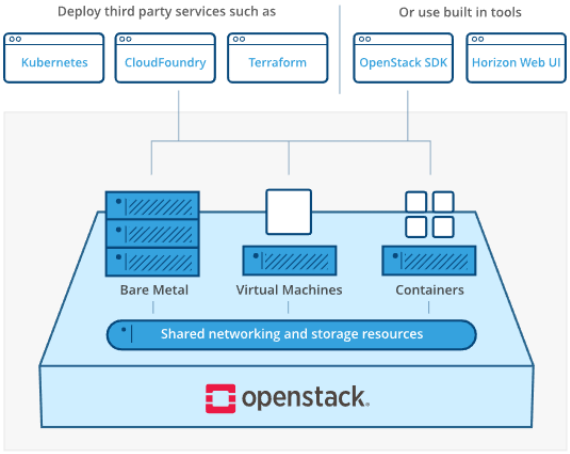

Figura 1. Esquema OpenStack.

OpenStack se divide en servicios para permitirle conectar y reproducir componentes según sus necesidades. Estos componentes proporcionan APIs que nos permiten interactuar con los recursos.

Una instalación básica de OpenStack estaría compuesta por sus elementos principales a los que se les denomina CORE.

Más allá de la funcionalidad estándar de infraestructura como servicio, los componentes adicionales proporcionan orquestación, administración de fallos y administración de servicios, entre otros servicios, para garantizar una alta disponibilidad de las aplicaciones de usuario.

Un despliegue OpenStack puede estar contenido en una o varias máquinas físicas o virtuales. Estas máquinas se denominan nodos y pueden desempeñar diferentes funcionalidades.

Generalmente, se suele hacer instalación OpenStack multinodo, en la que los nodos tienen un papel concreto, como un nodo de cómputo o almacenamiento y existe un nodo principal, denominado nodo controlador, que se encarga de que los componentes de OpenStack pueden comunicarse.

2.1 Servicios y componentes

Como se comentaba anteriormente, OpenStack está dividido en diferentes servicios como servicio de cómputo, orquestación o de red. Estos servicios están compuestos por diferentes componentes que realizan las funciones necesarias.

Los componentes CORE de una instalación OpenStack son los mínimos necesarios para prestar servicios en un despliegue. Encontramos en esta categoría a los componentes organizados por servicios[6]:

-**Servicios Web Frontend**: o Dashboard, es el servicio de OpenStack encargado de ofrecer una interfaz web con la que el usuario pueda manejar los diferentes componentes y servicios del despliegue. Horizon es el componente encargado por defecto de ofrecer este servicio, aunque no es el único disponible.

-**Servicios de Orquestación**: Heat es el componente encargado de realizar las funciones del servicio de orquestación, el cuál desempeña un papel crucial para la automatización de algunas funcionalidades del resto de servicios a través de unas plantillas en determinados formatos como YAML o JSON.

-**Servicios de Cómputo**: este servicio es el responsable de las funcionalidades relacionadas con la gestión de los recursos del despliegue en las máquinas virtuales. El componente encargado de este servicio es Nova.

-**Servicios de Red**: Neutron es el componente que proporciona las funciones necesarias para ofrecer un servicio de red que permita crear, modificar o eliminar redes dentro de un despliegue. Estas redes se utilizan para comunicar las máquinas virtuales entre sí o con una red externa.

-**Servicios de Control de hardware**: es el servicio encargado de aprovisionamiento de bare metal, a través de la comunicación con los hipervisores de la maquina o maquinas que hospedan el despliegue y la gestión de sus recursos. Ironic es el nombre del componente encargado de este servicio.

-**Servicios de Almacenamiento**: los servicios de almacenamiento que ofrece OpenStack se distinguen según lo que se busca almacenar. Dispone del servicio de almacenamiento de objetos, Swift, utilizado para el almacenamiento de objetos y programas que se encuentran dentro de una VM y servicio de almacenamiento en bloque, Cinder, donde se almacenan en Volúmenes las imágenes que corresponden con los sistemas operativos que se despliegan en una VM.

-**Servicios compartidos**: además de los servicios específicos anteriores, OpenStack dispone de una serie de servicios compartidos que permiten la autenticación del usuario ante los diferentes servicios y componentes del despliegue, Keystone es el componente y el servicio Identity o servicio de identidad, la utilización de las API HTTP de estos, Placement, y el servicio encargado del manejo de imágenes de sistemas operativos predefinidos, Glance para el servicio de imágenes.

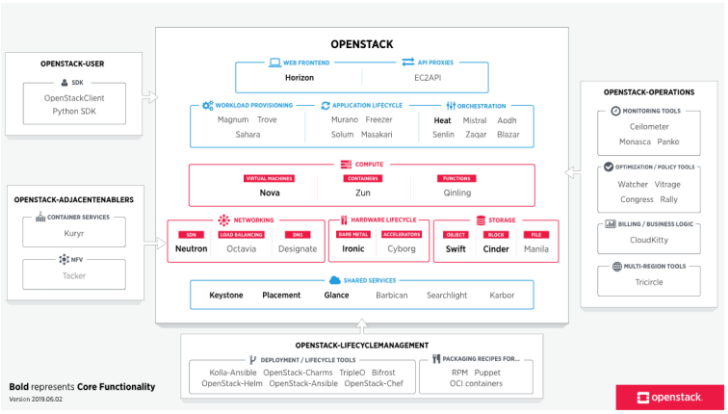

Figura 2. Esquema componentes OpenStack.

Lo que hemos visto hasta ahora son componentes y servicios del CORE, pero OpenStack además, tiene ciertas herramientas/componentes/servicios que permiten al usuario controlador del despliegue conocer más información y realizar ciertas operaciones. Entre los diferentes servicios, encontramos el servicio de telemetría, con su componente Ceilometer, que nos permite obtener información de la utilización de los recursos.

Este componente, se apoya en nuestra versión en otros componentes, Gnocchi[7], para la obtención de métricas y/o Aodh, para la creación y detección de alarmas. Utilizan bases de datos donde guardar esta información.

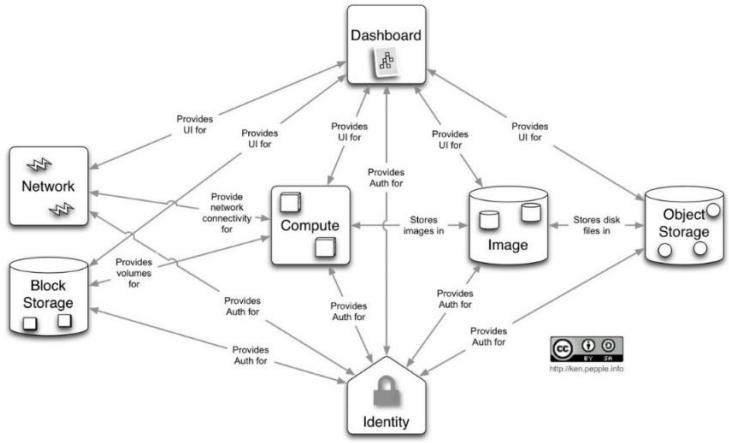

Figura 3. Esquema relaciones entre componentes de OpenStack.

Como hemos visto, OpenStack ofrece una gran cantidad de posibilidades gracias a sus servicios y componentes, sin embargo, en este proyecto no se utilizarán todos y por ello a continuación se hará una breve mención de los componentes utilizados en el caso de uso que se describirá con detalle más adelante.

Por lo que, para este proyecto utilizaremos el componente Neutron, para la creación de los elementos de red que se necesitan para establecer conexión entre dos VM, Nova, para la creación de las VMs, Glance, para crear la imagen que utilizaremos en las VMs y Gnocchi (Ceilometer) para la obtención de información de los recursos utilizados. Por supuesto, para acceder a cualquiera de estos servicios primero hay que identificarse correctamente en el servicio de Identity.

2.2 Versiones

Una nueva versión de OpenStack es desarrollada aproximadamente en ciclos de 6 meses. Desde una primera reléase de la versión se pueden encontrar una serie actualizaciones hasta encontrar una estable.

En actualidad existen muchas versiones de OpenStack, aunque muchas de ellas se encuentran en un estado en el que ya no se actualizan, como Newton o Mitaka. Para este proyecto se ha usado la versión Queens, que actualmente dispone de mantenimiento y es una versión conocida por su estabilidad.

Aunque Queens es la utilizada, se debe mencionar que esta no es la última versión disponible, Stein, y que próximamente saldrá la versión Train.

2.3 OpenStack en Red Hat Enterprise Linux

Red Hat Enterprise Linux también conocido por sus siglas RHEL es una distribución comercial de GNU/Linux desarrollada por Red Hat. Es la versión comercial basada en Fedora que a su vez está basada en el anterior Red Hat Linux.

Mientras que las nuevas versiones de Fedora salen cada aproximadamente 6 meses, las de RHEL suelen hacerlo cada 18 o 24 meses.

Cada una de estas versiones cuenta con una serie de servicios de valor añadido sobre la base de los que basa su negocio (soporte, formación, consultoría, certificación, etc).

Cada versión lanzada cuenta por el momento con soporte durante al menos 10 años desde la fecha de lanzamiento de la GA (General Availability) (o versión acabada en .0), durante este tiempo, se dividen varias etapas de soporte.

2.3.1 RDO

RDO es una comunidad de personas que usan e implementan OpenStack en CentOS, Fedora y RHEL. Pone a disposición de los usuarios documentación que sirve de ayuda para comenzar con OpenStack, listas de correo en las que puede conectarse con otros usuarios y paquetes compatibles con las versiones más actualizadas de OpenStack disponibles para descargar.

Esta comunidad ofrece diferentes paquetes de instalación de OpenStack según las necesidades del usuario:

- **TripleO**: para el despliegue una nube de producción, utilizada para el desarrollo y manejo de OpenStack
- **TripleO QuickStart**: para el desarrollo y prueba de OpenStack
- **Packstack**: paquete All-in-One para pruebas de concepto OpenStack

Ya que nuestras necesidades buscan satisfacer una prueba de concepto de una herramienta capaz de conectarse con los principales componentes de un despliegue OpenStack, vamos a optar para la instalación Packstack.

Packstack es una forma sencilla de instalación, all-in-one o todo en uno en español, que nos permite desplegar un clúster OpenStack con sus principales componentes, en uno o varios nodos, a través de un simple script de configuración “answer-file” que se puede generar automáticamente.

Capítulo 3. Arquitectura de la herramienta

A continuación, se mostrarán con detalle la arquitectura del programa, donde primero se hará una explicación global de cómo funciona y como está organizado para posteriormente entrar en profundidad en las partes que se consideran más relevantes para un posible desarrollador.

A modo introductorio, es un proyecto Maven, que es una herramienta de software para la gestión y construcción de proyectos Java con un modelo de configuración de construcción simple, basado en un formato XML y es un plugin que complementa al proyecto Net2Plan.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores.

Net2Plan cuenta con muchas clases, además de su interfaz gráfica, que nos son de utilidad para nuestros objetivos.

3.1 Arquitectura global

De manera general, en esta sección se busca explicar las relaciones que existen entre las principales clases de Net2Plan que se han utilizado para el desarrollo del plugin y las nuevas que se han creado.

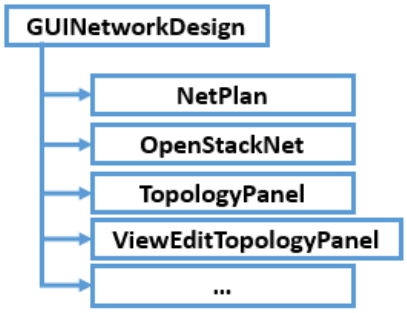

Figura 4. Clases principales del plugin.

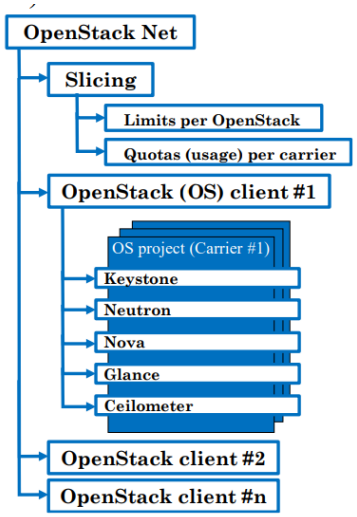

Figura 5. Contenido general de la clase OpenStackNet

La clase *GUINetworkDesign* es la clase principal (Fig. 4), que contiene e inicializa los elementos principales del programa, como el objeto *NetPlan* o *OpenStackNet*.

Esta clase es la responsable de mostrar la topología actual, actualizar el estado de las tablas, añadir nuevos despliegues o modificar el diseño, por ello, se suele pasar como argumento al resto de clases. Este concepto se denomina *callback*.

En el objeto *OpenStackNet* encontraremos una lista de objetos *OpenStackClient* con los despliegues de OpenStack hayamos añadido (Fig. 5).

Cada uno de ellos, tendrá asociados un objeto *OpenStackNetCreate*, un objeto *OpenStackNetDelete*, un objeto *NetPlan* y una serie de listas con los diferentes objetos que extienden de la clase *OpenStackNetworkElement*.

El objeto *ViewEditTopologyTables* contendrá un conjunto de objetos que extienden de la clase AdvancedJTable*_networkElement*, relacionados con las listas de objetos *OpenStackNetworkElement*, para cada uno de los diferentes objetos *OpenStackClient* que existen en el objeto *OpenStackNet*.

Cada vez que se añade algún elemento a las listas o se añade un nuevo cliente, el elemento *ViewEditTopologyTables* se actualiza para mostrar el nuevo contenido.

En la interfaz, en la parte que corresponde al objeto *ViewEditTopologyTables*, al seleccionar uno de los conjuntos correspondientes a un *OpenStackClient*, se actualiza el objeto *NetPlan* de la clase *GUINetworkDesign* con el objeto *NetPlan* del *OpenStackClient* para que el objeto *TopologyPane* muestre el diseño de red asociado a este.

3.2 Estructura de clases y paquetes principales del plugin

En esta sección se explica en una primera parte y con carácter general la estructura de paquetes del plugin con sus clases principales, y en una segunda parte, con más detalle las clases que se han desarrollado para este plugin.

Como se puede observar en la imagen (Fig. 6), tenemos en el primer nivel los paquetes *networkDesign* y *utils*. Además, la clase *GUINetworkDesign* que no está visible en la figura.

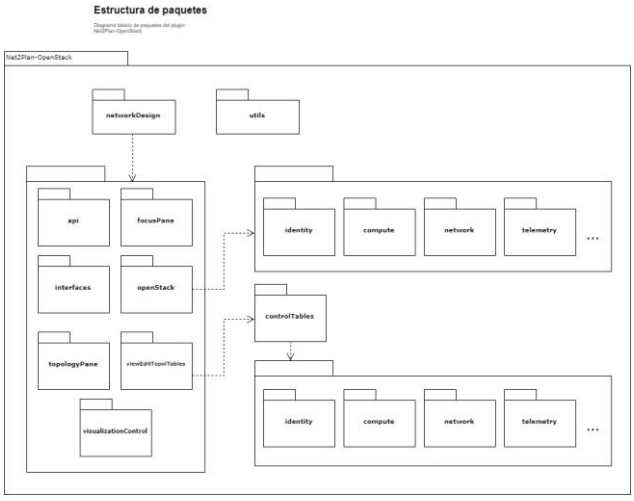

Figura 6. Estructura de paquetes del plugin

El paquete *utils*, contiene las clases:

-***AnalisisFrame***, ***OpenStackGraphCreator***: clases destinadas al análisis y creación de gráficos para las diferentes Resources que podemos encontrar en la pestaña de *Ceilometer* para un OpenStack.

-***CypherClass***: clase utilizada para el cifrado de los datos de acceso al despliegue.

-***CellRenderers*** y ***FilteredTablePanel***: clases heredadas de Net2Plan. Utilizadas para las tablas de diseño de red.

-***GeneralForm*** y ***GeneralFormUpdate***: clases generadoras de los formularios disponibles con las funciones de clic derecho de las tablas.

-***IPAddressComparator***: clase encargada de determinar si una VM pertenece a una subred a través de la comparación de los IPs.

-***OpenStackFileChooser***, ***OpenStackInformationDialogCreator***, ***OpenStackInitalButtonFunction***, ***OpenStackLoginDialogCreator***, ***OpenStackProgressBar*** y ***OpenStackUtils***: clases creadas para el soporte de las funciones iniciales de acceso al despliegue, como el panel con el formulario de acceso o la barra de progreso al añadir un despliegue.

-***SwingBrowser***: clase encargada de mostrar la consola de las VMs.

Dentro del paquete *networkDesign* encontramos los paquetes *api*, *focusPane*, *interfaces*,*openStack*,*topologyPane*, *viewEditTopoloTables*,*visualizationControl* y las clases *CellRenderes*,*ElementSelection*,*FileChooserNetworkDesign*,*GUIWindow*,*InfNumberFormat*,*Last RoqAggregatedValue*,,*NetworkDesignWindow*,*TableComparator*, que son heredadas de Net2Plan y que sirven de apoyo al plugin.

El paquete *openStack* (Fig. 7) contiene las clases *OpenStackNetworkElement*, *OpenStackNet*, *OpenStackNetClient*, *OpenStackNetCreate* y *OpenStackNetDelete*, y los paquetes *compute*, *extra,identity,image,network,telemetry.*

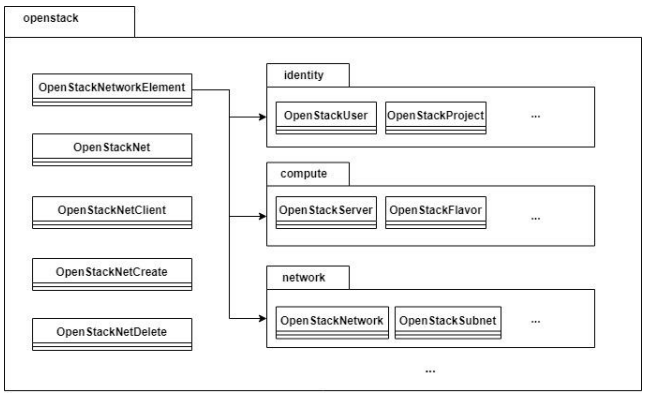

Figura 7. Clases relacionadas con OpenStack.

```
public class OpenStackUser extends OpenStackNetworkElement
{

    private String userId;
    private String userName ;
    private String userDomainId ;
    private String userEmail ;
    private String userDescription ;
    private Domain userDomain;
    private Boolean userEnable;
    private Map<String,String> userLinks;
    private User user ;
    private String userDefaultProjectId;


    public static OpenStackUser createFromAddUser (OpenStackNet osn , User user, OpenStackClient openStackClient)
    {...}

    private OpenStackUser (OpenStackNet osn , User user, OpenStackClient openStackClient)
    {...}

    @Override
    public String getId () { return this.userId; }
    public String getName () { return this.userName; }
    public Domain getUserDomain() {return this.userDomain;}
    public String getDomainId () { return this.userDomainId; }
    public String getEmail () { return this.userEmail; }
    public String getDescription () { return this.userDescription; }
    public String getUserDefaultProjectId () { return this.userDefaultProjectId; }
    public Boolean isUserEnable(){return  this.userEnable;}
    public Map<String,String> getUserLinks(){return this.userLinks;}


    @Override
    public String get50CharactersDescription()
    {...}

    public void changeUserEnabledState(Boolean value){...}
```

Figura 8. Ejemplo de clase que extiende de OpenStackNetworkElement.

La clase *OpenStackNetworkElement* es una clase abstracta que representa cualquier elemento de un despliegue de OpenStack (Fig. 8). Cada clase contenida en los paquetes *compute, extra, identity, image, network y telemetry*, son clases que heredan de esta clase y que representan un elemento especifico. Por ejemplo, dentro del paquete *network*, encontraremos la clase *OpenStackNetwork*, que representa una red dentro de un proyecto de OpenStack.

La estructura básica que sigue una clase *OpenStackSomething* que extienda de la clase *OpenStackNetworkElement* dispone de una serie de campos o propiedades que están vinculados con los parámetros del elemento al que representa con métodos para obtenerlos o modificarlos, un constructor estático que como parámetros recibe el objeto *OpenStackNet*, el objeto en cuestión obtenido con la librería OS4J y el *OpenStackClient* al que pertenece. Además, tiene los métodos heredados.

Las clases *OpenStackNetCreate* y *OpenStackNetDelete*, son clases que se utilizan para crear y eliminar objetos en un despliegue.

La clase *OpenStackClient* es la encargada de conectar con el despliegue a través del cliente de la librería OpenStack4j y contiene listas de los objetos que se pueden encontrar en él. También es responsable de generar la topología necesaria para la representación de los elementos de red disponibles.

La clase *OpenStackNet* es la clase que maneja los *OpenStackClient*. Encargada de añadir o eliminar conexiones.

El paquete *viewEditTopolTables* (Fig. 9) contiene la clase *ViewEditTopologyTablesPanel* y el paquete *controlTables*, que contiene las clases *AjtRcMenu, AjtColumnInfo, AjtRenderers, AbvancedJTable_abstractElement, AdvancedJTable_networkElement* y los paquetes *compute, extra,identity,image,network,telemetry.*

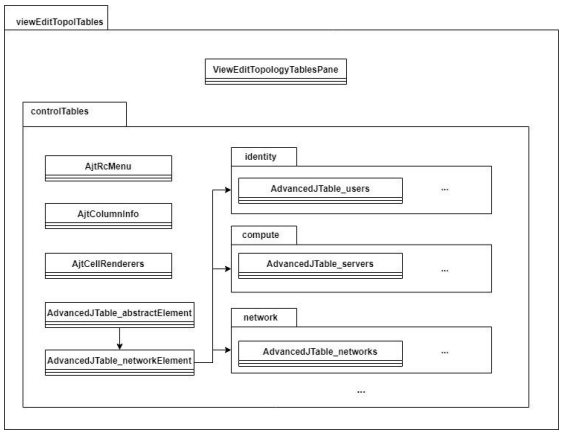

Figura 9. Clases de las tablas del panel de diseño del plugin.

```
public class AdvancedJTable_users extends AdvancedJTable_networkElement<OpenStackUser>
{
    public AdvancedJTable_users(GUINetworkDesign callback, OpenStackClient openStackClient)
    {
        super(callback, ViewEditTopologyTablesPane.AJTableType.USERS , hasAggregationRow: true, openStackClient);
    }

    @Override
    public List<AjtColumnInfo<OpenStackUser>> getNonBasicUserDefinedColumnsVisibleOrNot()
    {...}


    @Override
    public List<AjtRcMenu> getNonBasicRightClickMenusInfo()
    {...}

    public void addUser() {...}

    public void removeUser(ArrayList<OpenStackUser> user){...}


}
```

Figura 10. Ejemplo clase que extiende de AdvancedJTable_networkElement.

La clase *ViewEditTopologyTablesPanel* es la encargada de generar las tablas de diseño de red. Genera una tabla para analizar cada uno de los elementos del despliegue para cada todos los despliegues.

Las clases *AjtRcMenu, AjtColumnInfo y AjtCellRenderers* se utilizan para la creación de las tablas. Agilizan la creación de funciones de clic de derecho y la representación de la información que proporcionan los elementos.

La clase *AdvancedJTable_abstratElement* es una clase abstracta que representa una tabla de elementos de cualquier tipo. La clase *AdvancedJTable_networkElement* es otra clase abstracta que extiende de la clase anterior y que representa cualquier tabla que contenga elementos que extiendan de *OpenStackNetworkElement* (Fig. 10). Cada clase contenida en los paquetes *compute, extra, identity, image, network y telemetry*, son clases que heredan de esta clase y que representan un elemento especifico. Por ejemplo, dentro del paquete *network*, encontraremos la clase *AdvancedJTable_networks*, que representa una tabla que contiene los elementos *OpenStackNetworks* de una red dentro de un proyecto de OpenStack.

La estructura básica que sigue una clase *AdvancedJTable _somethings* que extienda de la clase *AdvancedJTable_ networkElement* dispone de una lista de *AjtColumnInfo* que contienen los campos o propiedades que están vinculados con los parámetros del *OpenStackNetworkElement* al que contienen y una lista de *AjtRcMenu* con las diferentes funciones de clic derecho disponibles para la tabla. El constructor como parámetros recibe el objeto *GUINetworkDesign* y el *OpenStackClient* al que pertenece. Este, llama al constructor de la clase padre pasándole además como parámetro el objeto *AJTableType* de la clase *ViewEditTopologyTablesPanel* que vincula la tabla con la lista que le corresponde.

3.3 OpenStack4j

OpenStack4j es un cliente de código abierto de OpenStack que permite el aprovisionamiento y el control de un sistema OpenStack.

La biblioteca compone de API abstractas relacionadas con los componentes y servicios más importantes de un despliegue de OpenStack. Estas llamas a las API devuelven un objeto o una colección de objetos con los datos de nuestro componente en el sistema. Las llamadas de esta biblioteca siguen el concepto de programación “fluent” que permiten una programación más ágil y fluida.

Cada API dispone de una serie de llamadas que permiten obtener/modificar datos de un componente de nuestro sistema.

Además, cuenta con el manejo de diferentes excepciones que nos permiten controlar posibles fallos, como la autenticación o errores en la comunicación.

En nuestro caso, hemos encontrado bastante interesante la utilización de su objeto *OSClientV3* (Fig. 11), que nos permite crear un cliente con nuestro despliegue. Además, podemos obtener el token que utilizaremos para autenticarnos con nuestras llamadas REST o para otras funcionalidades que lo requieren.

No solo es útil para realizar una autenticación con el despliegue, tiene la posibilidad de conectar con las diferentes APIs de los componentes de este, de una manera simple e intuitiva que nos permite obtener los objetos que nos resultan de interés (Fig. 12).

```
OSClient.OSClientV3 os = OSFactory.builderV3()
        .endpoint(os_auth_url)
        .credentials(os_username,os_password,Identifier.byName(os_user_domain_name))
        .scopeToProject(Identifier.byId(os_project_id))
        .authenticate();

this.token = os.getToken();
```

Figura 11. Ejemplo cliente OpenStack de OS4J.

```
/* Get elements of Network(NEUTRON) */
this.os.networking().network().list().forEach(n -> addOpenStackNetwork(n));
this.os.networking().subnet().list().forEach(n -> addOpenStackSubnet(n));
this.os.networking().router().list().stream().forEach(n -> addOpenStackRouter(n));
this.os.networking().port().list().stream().forEach(n -> addOpenStackPort(n));
```

Figura 12. Ejemplos de uso de la librería OS4J para recuperar datos.

Incluso, la librería permite crear y destruir nuevos elementos en un despliegue a partir de unos parámetros que nosotros le pasemos. Normalmente, estos creadores se llaman con la clase *Builders* pero algunos objetos pueden tener creadores especiales (Fig. 13).

Los objetos que ya existen también pueden ser modificados, ya que la librería está capacitada para ello. Nuevamente, puede hacerse uso de la clase *Builders* (Fig. 14).

```
ServerCreate sc = this.openStackClient.getClient().compute().servers().serverBuilder()
        .name(serverName)
        .flavor(serverFlavorId)
        .image(serverImageId)
        .networks(list)
        .addSecurityGroup(serverSecurtyGroupId)
        .build();
```

Figura 13. Ejemplo de constructor de elementos de OpenStack con OS4J.

```
public void setName (JSONObject jsonObject) {
    try
    {
        this.openStackClient.getClient().networking().network()
                .update(this.networkId,
                        Builders.networkUpdate().name(jsonObject.getString( key: "Name")).build());
    }catch(Exception ex){

        logPanel();
    }
}
```

Figura 14. Ejemplo de actualización de propiedades.

Por esto, la librería OS4J está presente en el desarrollo de esta herramienta facilitando la comunicación con un nodo de OpenStack.

3.4 Llamadas REST propias

OpenStack4j es una gran librería que ha facilitado el desarrollo de la herramienta, sin embargo, para las funcionalidades que nos resultaban de interés, no era capaz de recuperar los datos debido a una desactualización en la APIs de los componentes Ceilometer, Keystone y Cinder respecto a las llamadas actuales que se deben realizar a OpenStack.

Tras surgir este inconveniente, se ha considerado necesario realizar unas APIs que se comuniquen con dichos componentes para recuperar los datos de interés. Aprovechando el cliente de OpenStack4j, recuperamos el token necesario para la autenticación en las llamadas OpenStack.

```
protected HttpURLConnection connection = null;

protected Object Get(String URL,String token){


    try {
        //Create connection
        URL url = new URL(URL);
        connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty( s: "X-Auth-Token", token);
        connection.setRequestProperty( s: "Content-type",  s1: "application/json");

        //Get Response
        InputStream is = connection.getInputStream();
        BufferedReader rd = new BufferedReader(new InputStreamReader(is));
        StringBuilder response = new StringBuilder();
        String line;

        while ((line = rd.readLine()) != null) {
            response.append(line);
            response.append('\r');
        }
        rd.close();

        return response;

    } catch (Exception e) {

        e.printStackTrace();

    }
    connection.disconnect();
    connection = null;
    return null;
}
```

Figura 15. Código para la llamada GET.

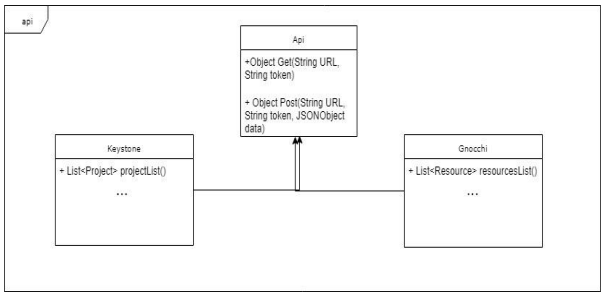

Figura 16. Estructura de clases para llamadas a las APIs.

Dentro del paquete api podemos encontrar las clases *Api, Keystone y Gnocchi* (Fig. 16).

Api es una clase que contiene los métodos genéricos que realizan las llamadas HTTP que se utilizan para recuperar o poner datos en las entidades o componentes del despliegue. Para realizar estas llamadas utiliza un objeto *HttpURLConnection* de la librería java.net.

Los métodos reciben como parámetros un *String* que contiene la URL hacia la que realizar la petición, otro *String* que contiene el token de autenticación dentro de OpenStack para el usuario que está manejando la herramienta y en el caso del método Post, un objeto JSON que contiene el objeto que se desea añadir.

Los métodos devuelven un *Object* que suele contener los datos a recuperar o información respecto a posibles errores.

Las clases *Keystone y Gnocchi* extienden de la clase *Api* (Fig. 17) y heredan los métodos protegidos *Get y Post*, para realizar las llamadas necesarias y obtener los datos que nos interesan de los componentes de OpenStack.

Ambas clases siguen una estructura parecida, con dos campos, un *String* y un *OSClientV3*, que se deben pasar con el constructor de la clase. Un *ENUM*, que contiene las diferentes URL que pueden existir dentro del mismo componente y métodos que interactúan con el componente (Fig. 18).

El *ENUM* se utiliza para completar la URL que se pasa en el constructor, ya que esta solo contiene la dirección IP del servicio.

Los métodos utilizan los métodos heredados *Get y Post*, para obtener los objetos deseados. Las respuestas de las APIs de los componentes de OpenStack tienen formato JSON y al tratarse en este caso de una solicitud que implica una lista, nos devuelve un objeto *JSONArray*. Tras las debidas conversiones, podemos recuperar los diferentes objetos que buscamos y devolverlos (Fig. 19).

Finalmente, hay que comentar que las clases que estas clases se crean cada vez que se crea un *OpenStackClient* y que se utilizan para rellenar los objetos *OpenStackNetworkElement* como si de la librería OS4J se tratase.

```
public class Keystone extends Api {

    private final OSClient.OSClientV3 osClientV3;
    public final String url;

    public enum KeystoneOption {...}

    public Keystone(String url, OSClient.OSClientV3 osClientV3) {...}

    public List<Project> projectList() {...}

}
```

Figura 17. Ejemplo clase que extiende de la clase Api.

```
public enum GnocchiOption {
    METRIC("metric/"),
    MEASURE("/measures"),
    RESOURCE("resource/generic")
    ;

    private final String tabName;
    private GnocchiOption(String tabName) { this.tabName = tabName; }
}
```

Figura 18. Ejemplo de posibles llamadas para una API.

```
public List<Project> projectList()
{
    final List<Project> projects = new ArrayList<>();

    Object responseObject = this.Get( URL: url+KeystoneOption.PROJECT.tabName,osClientV3.getToken().getId());

    JSONArray jsonArray = new JSONObject(responseObject.toString()).getJSONArray( key: "projects");

    for(Object object: jsonArray){

        JSONObject jsonObject = (JSONObject) object;

        Project project = new Project() {...};

        projects.add(project);
    }
    return projects;
}
```

Figura 19. Ejemplo de uso de una llamada para recuperar los proyectos de OpenStack.

Capítulo 4. Manual de usuario

A continuación, veremos a modo de manual de usuario, las funcionalidades que presenta la herramienta y los pasos necesarios para poder utilizarla.

Se indicarán los pasos para poder ejecutar la herramienta, una descripción de la interfaz gráfica para que el usuario se familiarice con el aspecto visual y sus ventanas principales, las diferentes formas que tenemos de acceder a un despliegue de OpenStack y las funcionalidades básicas disponibles.

4.1 Instalación

De momento no existe un ejecutable de Java disponible de la herramienta pero podemos utilizar la de forma bastante simple, solo tenemos que acceder al repositorio en GitHub del grupo GIRTEL de la UPCT y descargar/clonar la rama "fe" del proyecto Net2Plan-OpenStack[8] (Fig. 20).

Es un proyecto Maven en Java, que podemos abrir con cualquier IDE y en nuestro caso utilizamos IntelliJ.

Una vez abierto el proyecto, debemos ejecutar (run) la clase *GUILauncher* de Net2Plan disponible en el módulo Net2Plan-Launcher (Fig. 21).

Al ejecutarlo, aparecerá la ventana de inicio de Net2Plan versión 0.5.3 (Fig.22)

Como en Net2Plan, debemos seleccionar la herramienta que queremos utilizar. Para ello, pulsaremos en la barra herramientas la pestaña *Tools* y seleccionamos la herramienta *OpenStack Management Plugin* (Fig. 23).

Al hacer clic, ya dispondremos de la interfaz de nuestra herramienta (Fig. 24).

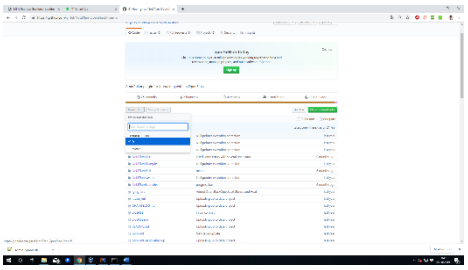

Figura 20. Repositorio GitHub del proyecto

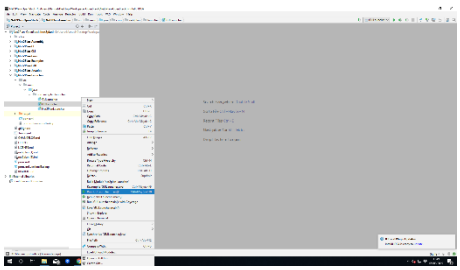

Figura 21. Proyecto abierto en Intellij y ejecutando el launcher.

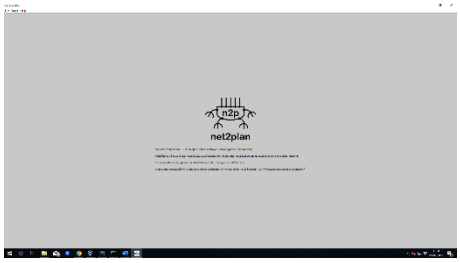

Figura 22. Ventana de inicio de Net2Plan.

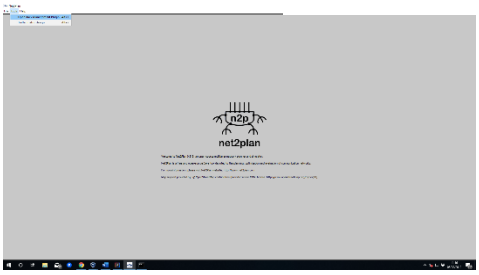

Figura 23. Selección del plugin en la pestaña Tools.

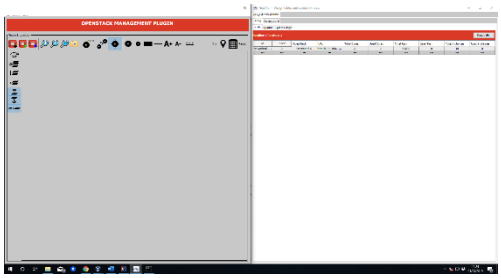

Figura 24. Ventana inicial del plugin Net2Plan-OpenStack.

4.2 UI Net2Plan-OpenStack

La UI[5] del plugin se compone de dos partes principales, el panel de la topología de la red y la ventana de diseño de red con las tablas de elementos de un despliegue OpenStack (Fig. 25).

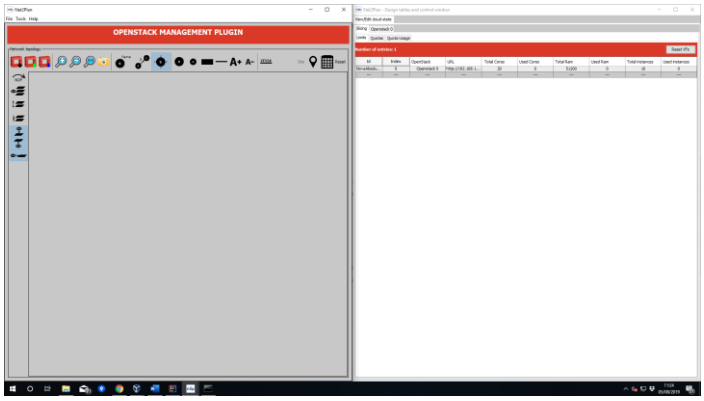

Figura 25. Ventana principal plugin, panel de topología (izquierda) y ventana de diseño de red (derecha).

4.2.1 Panel de topología

El panel de la topología se representará el esquema red disponible en un despliegue de OpenStack teniendo en cuenta todos sus proyectos, con sus respectivas redes, subredes, routers y máquinas virtuales.

El panel permanecerá vacío (Fig. 26) si se encuentra seleccionada la pestaña de las tablas Slicing[6], ya que la herramienta solo muestra la topología de red disponible en los diferentes proyectos dentro de un OpenStack y no el esquema general de todos los despliegues.

Sin embargo, si seleccionamos un despliegue de OpenStack que esté disponible, es decir, si hay alguna pestaña OpenStack X[7] seleccionada, el panel de topología representara los elementos de la red que existen para los diferentes proyectos (Fig. 27).

Al situarnos encima de uno de los nodos/iconos de elemento de red, tendremos un Tooltip[8] que nos dará información sobre el mismo (Fig. 28).

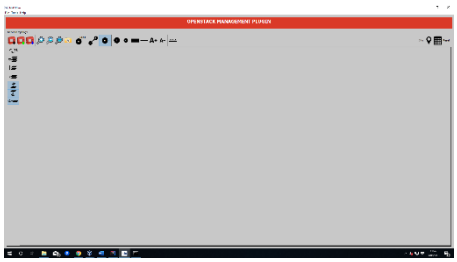

Figura 26. Panel de topología vacío.

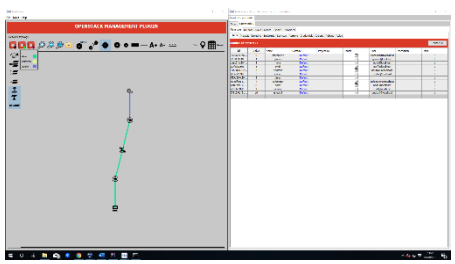

Figura 27. Representación de topología de red y contenido en las tablas.

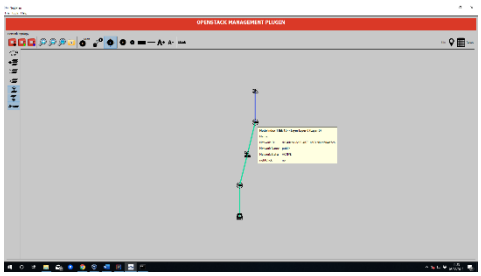

Figura 28. Tooltip informativo en elemento de red.

[5] UI: La interfaz de usuario o UI es el conjunto de elementos de la pantalla que permiten al usuario interactuar con el software.
[6] Slicing: una sola red física se puede dividir en múltiples redes virtuales, lo que permite al operador optimizar los recursos y la topología de red para el servicio específico y el tráfico que utilizará cada red virtual.
[7] OpenStack X: donde X es un número que se asocia a un despliegue. Por ejemplo, OpenStack 1.
[8] Tooltip: Un tooltip proporciona información extra cuando el usuario pasa el cursor sobre un elemento.

4.2.2 Tablas

Dentro de la ventana del diseño de red donde están las tablas, el usuario dispone de las pestañas *Slicing y OpenStack Xs* (Fig. 29).

La pestaña *Slicing*, muestra una vista general de los diferentes OpenStack Xs, lo que nos permite contemplar los recursos asignados a los despliegues y como están repartidos entre sus proyectos.

Desde esta pestaña podríamos ajustar los recursos de un proyecto de un despliegue o repartir los recursos a partes iguales, incluso obtener el porcentaje de uso de los recursos.

El resto de las funcionalidades podemos encontrarlas en la pestaña *OpenStack X* (Fig. 30), donde encontraremos subpestañas con los diferentes componentes del despliegue y con las funciones de clic derecho personalizadas dependiendo de cada elemento y componente.

Además, los campos de las tablas pueden ser editables o pueden contener hipervínculos entre los elementos disponibles en los despliegues, es decir, si tú haces clic en un hipervínculo (Fig. 31), saltaras a la tabla del elemento que se relaciona en ese campo (Fig. 32).

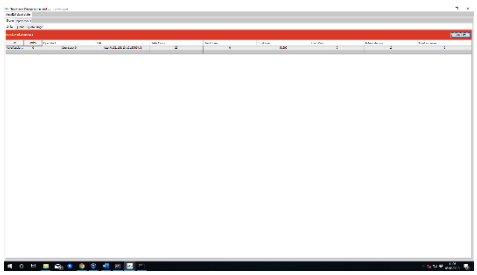

Figura 29. Ventana de diseño de red (tablas), pestaña Slicing.

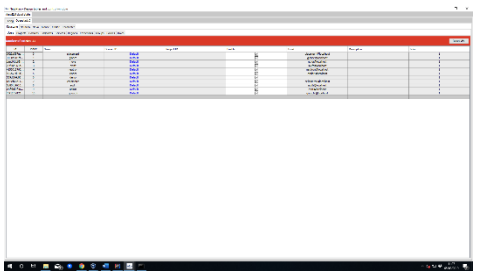

Figura 30. Ventana de diseño de red (tablas), pestaña OpenStack X.

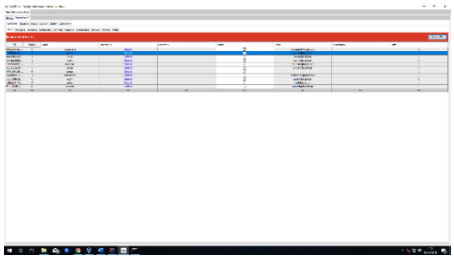

Figura 31. Clic en hipervínculo de uno de los campos de la tabla (1).

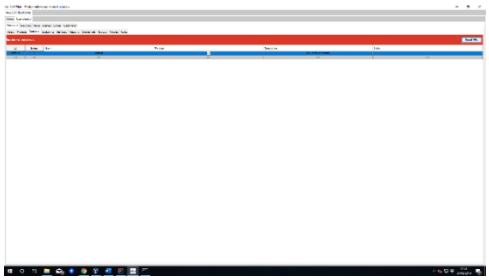

Figura 32. Salto a table relacionada con el hipervínculo.

4.3 Acceso a un despliegue OpenStack

En esta sección se explicarán las diferentes formas que existen en el plugin para poder tener acceso a un despliegue OpenStack. El plugin ofrece 3 formas diferentes: manual, a través de un archivo RC o con un archivo n2p.

Los campos que se utilizan para el acceso son los mismos en los tres casos:

Campos	OS_USERNAME	OS_PASSWORD	OS_AUTH_URL	OS_PROJECT_NAME	OS_U_DOMAIN_NAME
Definición	nombre de usuario	contraseña para el usuario	la URL donde se realiza la autenticación (identity)	nombre del proyecto	nombre del dominio del usuario

Figura 33. Tabla de campos para acceder a un despliegue.

4.3.1 Manual

Al hacer clic en el botón nos aparecerá un cuadro con campos de texto para rellenar (Fig. 34) y que contendrá los datos necesarios para acceder usando la librería OpenStack4j.

El usuario podrá encontrar los datos necesarios para acceder en los archivos de autenticación de OpenStack o en la interfaz gráfica y deberá introducirlos a mano.

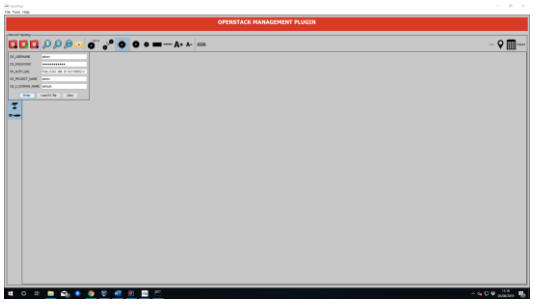

Figura 34. Ejemplo acceso manual.

4.3.2 Archivo RC de OpenStack

Al igual que en el caso anterior al hacer clic en el botón aparece el cuadro de acceso, pero esta vez hacemos clic en el botón Archivo RC (Fig. 35). Nos aparece una ventana de selección de archivo, ponemos el filtro adecuado (.sh) y pulsamos Open (Fig. 36). Automáticamente se rellenarán los campos necesarios a falta de poner la contraseña del usuario y pulsar *Enter* (Fig. 37).

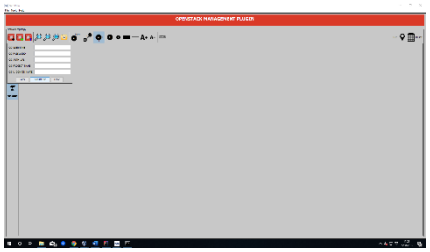

Figura 35. Desplegable al hacer clic en el botón para añadir un despliegue OpenStack.

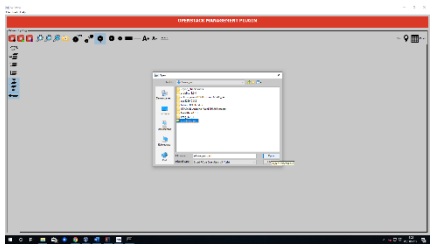

Figura 36. Selección del archivo .sh (RC) en el seleccionador de archivos.

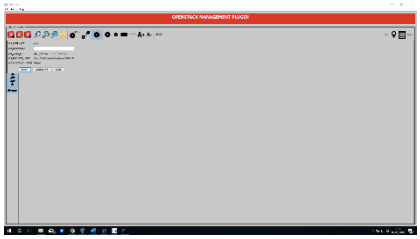

Figura 37. Campos autocompletados tras leer archivo RC.

El archivo RC lo podemos encontrar en la interfaz gráfica pulsando en Proyecto → Acceso a la API → Descargar fichero RC de OpenStack → OpenStack RC File (Fig. 38).

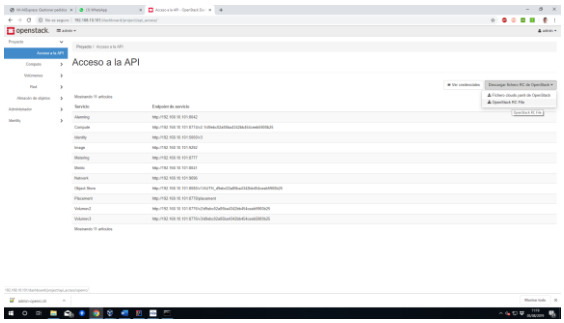

Figura 38. Localización archivo RC en OpenStack.

4.3.3 Archivo N2P de Net2Plan

Tras realizar cualquiera de las opciones anteriores si pulsamos el botón  , nos aparecería una ventana para elegir el destino y nombre de un fichero .n2p que se guardara con la información necesaria para poder utilizarla la próxima vez que se requiera (Fig. 39).

Este fichero, se puede cargar utilizando el botón  , donde nuevamente aparecería una ventana de selección de fichero, poniendo el filtro adecuado (.n2p) , seleccionando el fichero y pulsando *Open* accedería automáticamente al despliegue (Fig. 40).

Cabe decir, que en el fichero se guardarán todos los datos necesarios para todos los despliegues a los que se haya accedido.

El fichero está cifrado en base64 para proteger los datos (Fig. 41).

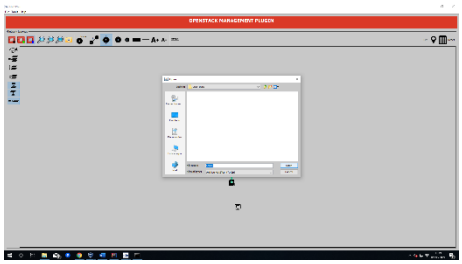

Figura 39. Guardar archivo .n2p.

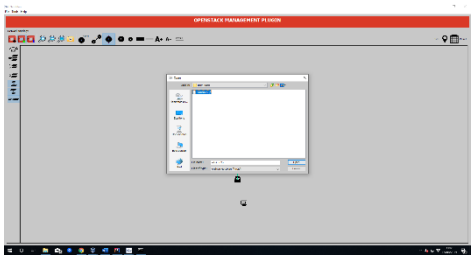

Figura 40. Leer archivo .n2p.

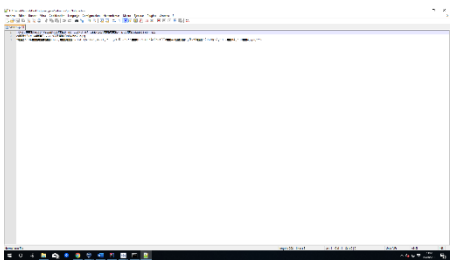

Figura 41. Contenido archivo .n2p en base64.

Si en cualquiera de los casos anteriores quisiéramos limpiar los cuadros de texto, solo tendríamos que pulsar *Clear* (Fig. 42). Igualmente, si pulsamos *Enter* en cualquiera de los casos anteriores nos saldría una barra de progreso indicándonos el estado del plugin mientras accede a los despliegues (Fig. 43).

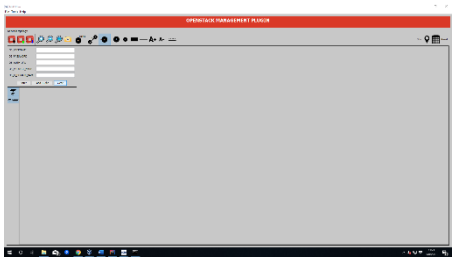

Figura 42. Campos vacíos tras pulsar el botón Clear.

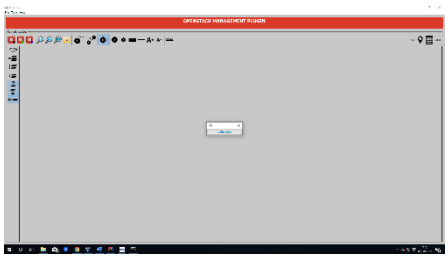

Figura 43. Barra de progreso cargando el contenido de un despliegue OpenStack al pulsar el botón Enter.

4.3.4 Funcionalidades básicas

Una vez hemos accedido correctamente a nuestro despliegue de OpenStack, las funcionalidades básicas que nos permitirán el control y manejo, de los recursos y elementos del sistema las encontraremos en todas las funcionalidades de clic derecho en tablas de los elementos de la red que permiten crean, modificar y eliminar los propios elementos o realizar acciones entre ellos como migrar una VM (Fig. 44).

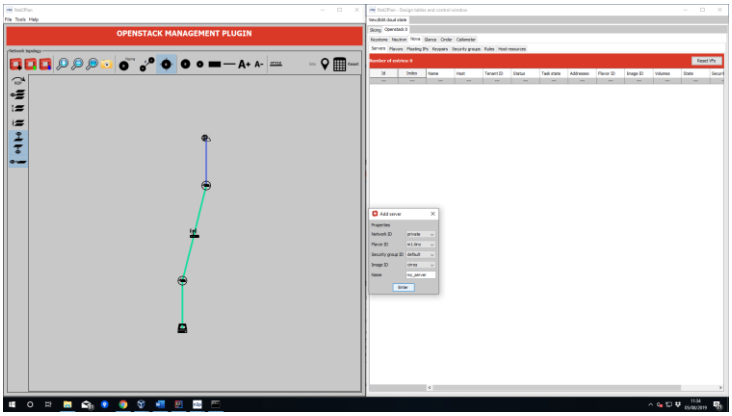

Figura 44. Ejemplo clic derecho en una tabla.

Capítulo 5. Caso de uso

En este apartado se explicará un caso de uso en el que se verán los siguientes ejemplos a través de la herramienta:

- Conexión de dos OpenStack diferentes.
- Subir imágenes a Glance.
- Instanciar VMs con Nova.
- Crear redes entre ellas con Neutron.
- Hacer ping a través de la consola de las VMs.
- Monitorización con Ceilometer (Gnochi).
- Control de recursos (Slicing).

La herramienta es utilizada también en la siguiente publicación "**The Net2Plan-OpenStack Project: IT Resource Manager for Metropolitan SDN/NFV Ecosystems**"[98], para más demostraciones de casos de uso.

5.1 Testbed

El banco de pruebas (Fig. 45) de nuestro caso de uso se compone de un portátil personal que ejecuta dos instancias de Net2Plan-OpenStack (nuestro plugin) y Net2Plan-Carrier, una VM de OSM[9] y dos mini PC que emulan los nodos de red metropolitana con una instancia VIM[10] OpenStack en cada uno de ellos. También se utiliza un mini PC que ejecuta el controlador ONOS[11] SDN y la red de transporte metropolitana emulada de Mininet[12] /Linc-OE[13] , IP sobre WDM[14] y dos switches autoconfigurados (uno dedicado para el plano de control y el otro dedicado al plano de datos).

Aunque en esta demostración el componente principal es nuestro plugin de Net2Plan para controlar OpenStack, también se hará una breve presentación del resto de componentes de nuestro banco de pruebas.

[9] OSM: software de gestión y orquestación (MANO) de código abierto alineada con los modelos de información del ETSI NFV. [10]

[10] VIM: El administrador de infraestructura virtualizada (Virtualized Infrastructure Manager, VIM) en una implementación de Virtualización de Funciones de Red (NFV) gestiona los recursos de hardware y software que el proveedor de servicios utiliza para crear cadenas de servicios y ofrecer servicios de red a los clientes.

[11] ONOS: (Open Network Operating System) es el principal controlador de SDN de código abierto para la construcción de soluciones SDN/NFV de próxima generación. [11]

[12] Mininet: proporciona un banco de pruebas virtual y un entorno de desarrollo para redes definidas por software (SDN).

[13] Linc-OE: emulador de la capa WDM para el banco de pruebas virtual.

[14] WDM: (Wavelength Division Multiplexing) la multiplexación por división de longitud de onda es una tecnología que permite transmitir varias señales independientes sobre una sola fibra óptica mediante portadoras ópticas de diferente longitud de onda, usando luz procedente de un láser.

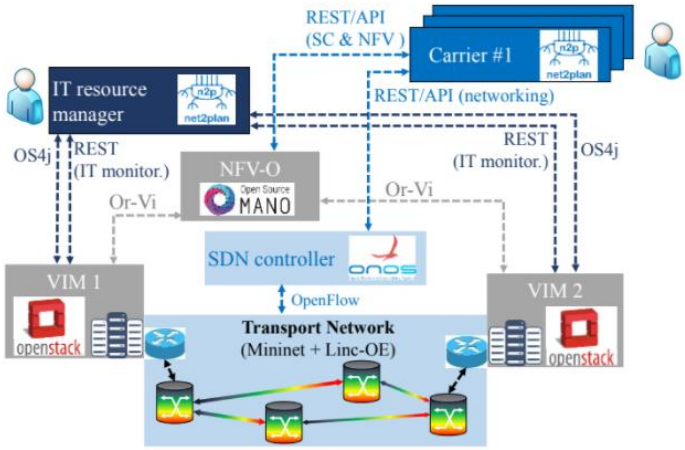

Figura 45. Testbed del caso de uso.

El plugin de Net2Plan-Carrier[12] emula el comportamiento de un operador de red que solicita el despliegue de una red a través de la interfaz gráfica de Net2Plan. Realiza el cómputo de una SC que representa una evolución del algoritmo *Path Computation Element* (PCE) ajustado para las asignaciones de la cadena de servicios, donde el camino está restringido para atravesar una secuencia de VNFs. La combinación de cálculo de ruta y decisión de colocación de VNF se implementa como un algoritmo en ese plugin de Net2Plan. Además, la implementación actual considera la emulación IP-over-WDM de la red de transporte.

El NFV-Orchestrator (NFV-O) está representado por OSM (versión 3) en una máquina virtual donde se realiza la gestión y orquestación de la infraestructura de virtualización que despliega VNF aprovechando las VIM.

Las VIM, implementadas con OpenStack (Queens), se encargan de instanciar y alojar las VM de los VNF. Con el sistema operativo RHEL e instaladas según el procedimiento descrito en el *Capítulo 2, en la sección 2.3.1 RDO, Packstack.*

Host	OpenStack	Tipo de nodo	IP
VIM 1	1	ALL-IN-ONE[15]	192.168.0.101
VIM 2	2	ALL-IN-ONE	192.168.0.102

Figura 46. Tabla configuración VIMs.

El control SDN de la red metropolitana es realizado por ONOS (1.14) a través de OpenFlow tanto al IP emulado en la red de Mininet y la capa WDM emulada por Linc-OE. El transporte emulado es proporcionado por la combinación de Mininet y Linc-OE de manera que el paquete realista fluye a través de una red IP de tamaño metropolitano sobre una capa WDM emulada se establecen entre los VNF instanciados en aparatos de hardware de consumo.

Una vez sabemos cuál el testbed de nuestro caso de uso y como están desplegados nuestros OpenStacks, nos disponemos a utilizar nuestra herramienta para realizar el siguiente caso de uso, donde nos conectaremos a los despliegues para instanciar una serie de máquinas virtuales, obtener sus consolas y comprobar que existe conectividad entre ellas. Además, por un lado, utilizaremos la pestaña *Ceilometer* de los OpenStack X para comprobar el uso de recursos

[15] All-in-one: Instalación de un nodo de OpenStack con todos los elementos CORE necesarios para un funcionamiento básico.

propios de las máquinas virtuales y, por otro lado, utilizaremos la pestaña Slicing de la herramienta para gestionar los recursos de un despliegue. [13]

5.2 Parte I. Creación de red y VMs

El primer paso sería añadir los despliegues a la herramienta, para ello podemos utilizar cualquiera de las funciones descritas en la sección *4.3 Acceso a un despliegue OpenStack* del *Capítulo 4 Manual de usuario*. Tras introducir las credenciales necesarias, obtendremos una vista similar a la Fig. 47, donde se encuentra ventana de diseño, en la que como podemos observar como en la subpestaña *Limits* de la pestaña *Slicing* se encuentran los datos de generales de cada OpenStack y además disponemos de una pestaña *OpenStack X* con la que podremos acceder a sus elementos principales. El *TopologyPanel* vacío, ya que en la ventana de diseño de red esta seleccionada la pestaña de *Slicing*.

Seleccionamos por ejemplo la pestaña *OpenStack X*, el *Topology Panel* representará los elementos de red disponibles en cada uno de los proyectos y cada proyecto está representado por un color en los enlaces, que se podrá ver la asociación color-proyecto en la subpestaña *Projects* de la pestaña *Keystone* (Fig. 48).

Una vez conectada la herramienta, comienza nuestro caso de uso. Nos disponemos a crear una red en uno de los despliegues. Para ello, nos dirigimos a la subpestaña *Networks* de la pestaña *Neutron* y hacemos clic derecho. Entre las opciones que nos aparecen, elegimos *Add network* (Fig. 49).

Esta opción, muestra una ventana con un formulario (Fig. 50) para rellenar con los diferentes tipos de redes que se pueden crear en un despliegue OpenStack dependiendo de la configuración de Neutron.

En nuestro caso, la vamos a crear una red en el proyecto admin[16], con el nombre my_network, de tipo VXLAN[17], que no será externa y cuyo proveedor de red será el 13.

Al hacer clic en *Enter*, si la red se ha creado correctamente, se actualizarán la ventana de diseño y de topología, para mostrar la nueva red (Fig. 51).

Debemos asegurarnos de que la columna *Admin State* este activada (Fig. 52). Además, podemos comprobar como todavía no tiene asociada ninguna subred.

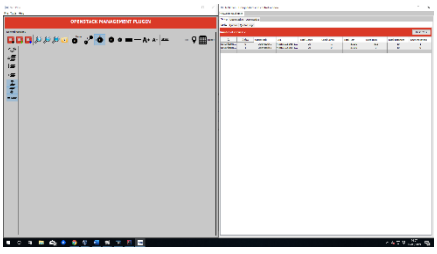

Figura 47. Pestaña Slicing con los dos despliegues conectados.

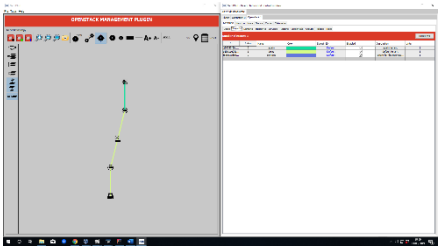

Figura 48. Pestaña OpenStack X correspondiente con uno de los despliegues.

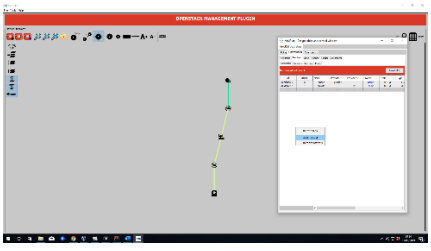

Figura 49. Opciones clic derecho en la tabla Networks.

[16] Proyecto admin: es uno de los proyectos por defecto que se crean en un despliegue OpenStack.
[17] VXLAN: una configuración que permite OpenStack para la creación de redes con VLANs virtuales.

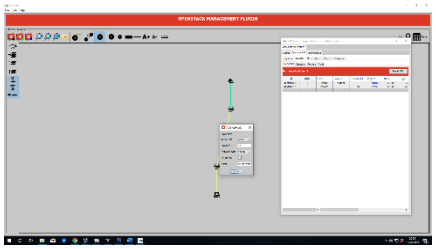

Figura 50. Formulario creación de red.

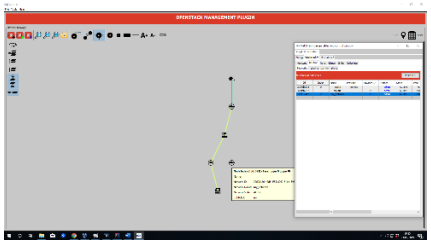

Figura 51. Actualización de tabla y topología tras crear la red.

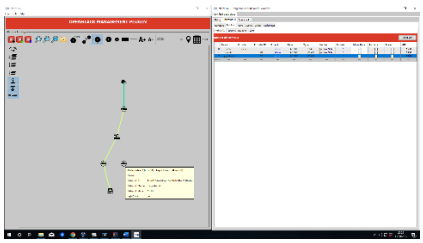

Figura 52. Modificar Admin State haciendo clic en la casilla de check.

El siguiente paso sería añadir una subred, por lo que nos situamos en la subpestaña *Subnets* y hacemos clic derecho, seleccionando la opción *Add subnet* (Fig. 53).

Al igual que en la opción *Add network*, nos aparecerá una ventana con un formulario para rellenar con los parámetros necesarios para crear una subred (Fig. 54).

Nuestra subred se llamará my_subnet, pertenecerá a red my_network, al proyecto admin, será de versión IPv4 y su CIDR[18] será 192.168.0.0/24 (IP de red y mascará).

Al pulsar *Enter*, se actualizan las ventanas para que aparezca nuestra subred (Fig. 55). Debemos activar DHCP[19].

Además, debemos añadirle un DNS[20]. Para ello, seleccionamos nuestra subred y hacemos clic derecho encima. De las opciones que salen, seleccionamos *Add subnet´s DNS* (Fig. 56).

Saldrá una ventana con un cuadro donde introducir la IP del servidor DNS que queremos utilizar, en nuestro caso el de Google 8.8.8.8 (Fig. 57).

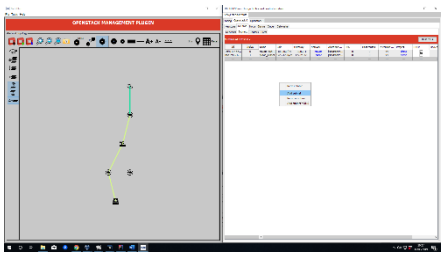

Figura 53. Seleccionar función clic derecho para añadir subred.

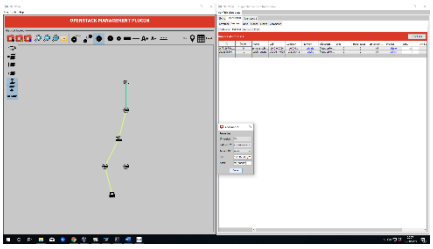

Figura 54. Formulario creación de subred.

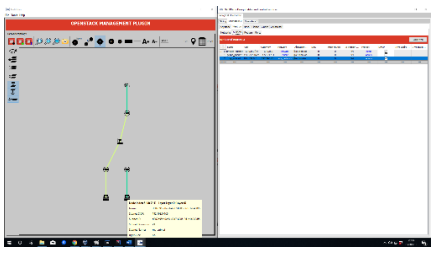

Figura 55. Actualización de topología y tablas con la creación de la subred.

[18] CIDR: es un estándar de red para la interpretación de direcciones IP que usa la técnica VLSM (variable length subnet mask), para hacer posible la asignación de prefijos de longitud arbitraria.
[19] DHCP: El protocolo de configuración dinámica de host es un protocolo de red de tipo cliente/servidor mediante el cual un servidor DHCP asigna dinámicamente una dirección IP y otros parámetros de configuración de red a cada dispositivo en una red para que puedan comunicarse con otras redes IP.
[20] DNS: (Domain Name System), y además de apuntar los dominios al servidor correspondiente, nos servirá para traducir la dirección real, su IP, en el nombre del dominio.

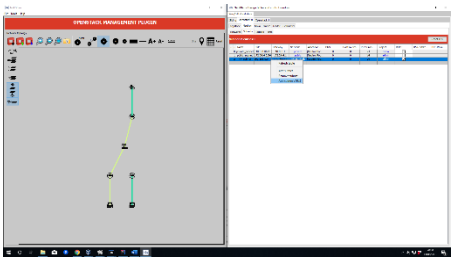

Figura 56. Selección de función de clic derecho para añadir un DNS a la subred.

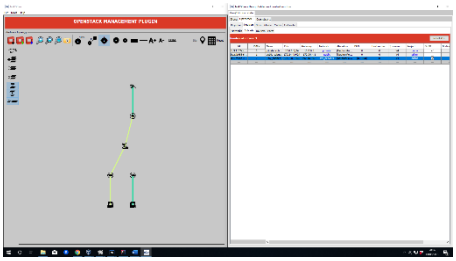

Figura 57. DNS añadido.

Una vez creada la red con su correspondiente subred, necesitamos un router. Para crear un router, nos dirigimos a la subpestaña *Routers*, hacemos clic derecho, y seleccionamos la opción *Add router* (Fig. 58).

Al igual que en los casos anteriores, esta función muestra un formulario para rellenar con los parámetros adecuados (Fig. 59).

El router se creará para el proyecto admin, con el nombre my_router, y para la red public[21], para que tener acceso al exterior con la red externa por defecto que se crea en un despliegue OpenStack.

Al pulsar *Enter*, se actualizan las ventanas para mostrar el nuevo router (Fig. 60).

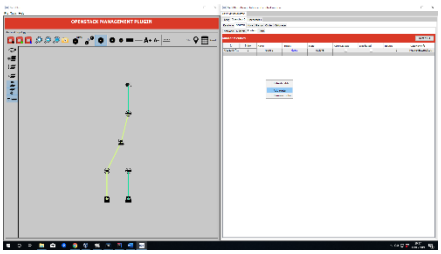

Figura 58. Añadir router con clic derecho en su tabla.

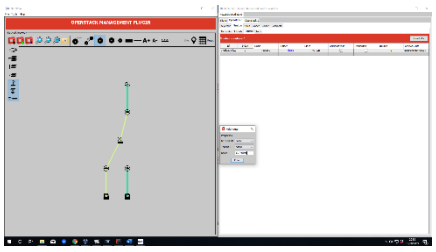

Figura 59. Formulario creación router.

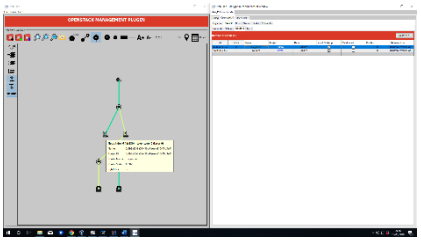

Figura 60. Actualización de topología y tablas tras la creación del router.

Ahora, debemos añadir a nuestro router un puerto que este asociado a nuestra red, para ello debemos situarnos en la pestaña *Ports*, y seleccionar la opción de clic derecho *Add port* (Fig. 61). En la ventana que aparece con el formulario debemos seleccionar la subred a la que queremos que pertenezca el puerto y el router al que queremos añadir el puerto (Fig. 62). El campo *Name* no es obligatorio, lo dejamos en blanco.

Una vez pulsamos *Enter* vemos como se crea el puerto y en la ventana de topología ya existe el enlace entre el router y la red, aunque su estado (State) sea DOWN[22] (Fig. 63).

[21] Red public: por defecto en la instalación de OpenStack la red pública con acceso a Internet.
[22] DOWN: estado de la VM que indica que esta apagada.

Puede deberse a diferentes factores, pero es probable que la herramienta haya recuperado los datos demasiado rápido. Para refrescar los datos, solo tenemos que seleccionar la opción de clic derecho *Refresh Table* de cualquiera de las pestañas (Fig. 64).

Podemos observar que ya se encuentra ACTIVE[23] y que todo está correctamente (Fig. 65).

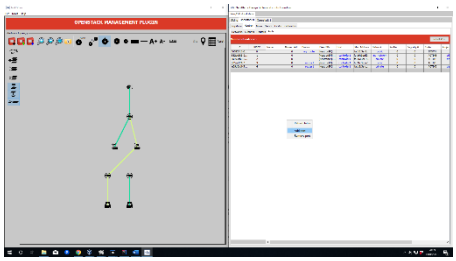

Figura 61. Selección de la función de clic derecho para añadir puerto.

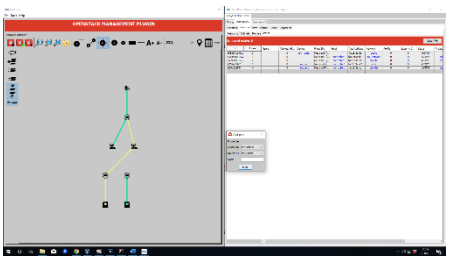

Figura 62. Formulario creación de puerto.

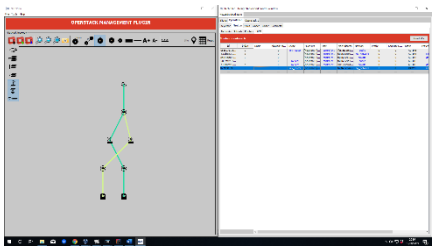

Figura 63. Comprobación del estado del puerto.

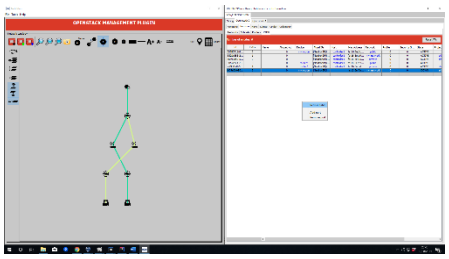

Figura 64. Refrescar tablas y topología.

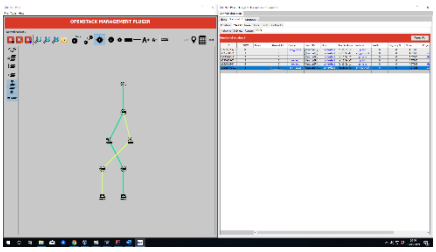

Figura 65. Verificación estado ACTIVE del puerto.

Ya hemos creado la red a la que se conectarán nuestras máquinas virtuales. A continuación, nos disponemos a crear las máquinas virtuales.

En primer lugar, vamos a subir una imagen de sistema operativo que nos permita realizar una prueba de concepto. Para ello, vamos a la subpestaña *Images* de la pestaña *Glance* y seleccionamos la opción de clic derecho *Create image* (Fig. 66). Abrirá una ventana de selección de archivo, debemos buscar nuestra imagen, seleccionarla y pulsar *Open* (Fig. 67). En nuestro caso utilizaremos una imagen CirrOS[24].

Al pulsar *Open*, nuestra imagen se subirá al despliegue OpenStack y esto tardará en función del tamaño de nuestra imagen. Cuando se suba, aparecerá una nueva entrada en la tabla con el nombre del archivo seleccionado (Fig. 68).

[23] ACTIVE: estado de la VM que indica que esta encendida.

[24] CirrOS: es una distribución mínima de Linux que fue diseñada para ser usada como una imagen de prueba en nubes como OpenStack Compute.

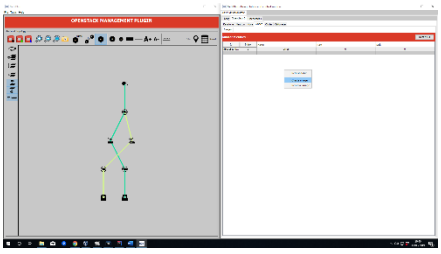

Figura 66. Selección de función clic derecho para subir imágenes.

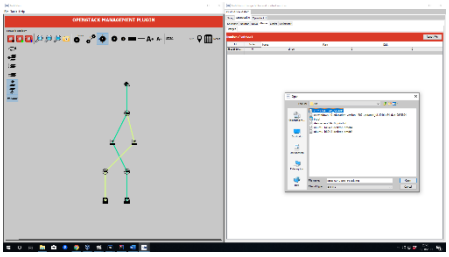

Figura 67. Selección de imagen en el buscador de archivos.

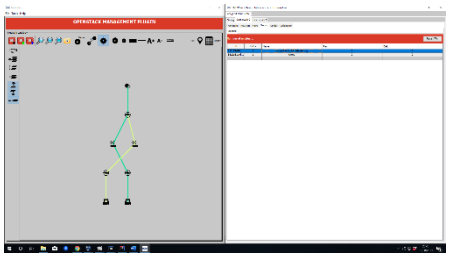

Figura 68. Actualización de tablas tras subir imagen.

Ya tenemos nuestra imagen, lo que nos permite instanciar nuestras máquinas virtuales. Nos dirigimos a la subpestaña *Servers* de la pestaña *Nova*. Hacemos clic derecho y seleccionamos la opción *Add server* (Fig. 69).

En la ventana resultante, se muestran las propiedades o parámetros para la creación de una máquina virtual (Fig. 70). En nuestro caso, crearemos las máquinas virtuales my_vm_1 y my_vm_2, que pertenecerán a la red my_network, con un flavor m1.tiny, un grupo de seguridad default y que utilizará la imagen que previamente hemos subido.

Al hacer *Enter*, se actualizará las ventanas, sin embargo, el proceso de instanciación es un proceso costoso, y dependerá de los recursos del equipo donde este desplegado el OpenStack (Fig. 71).

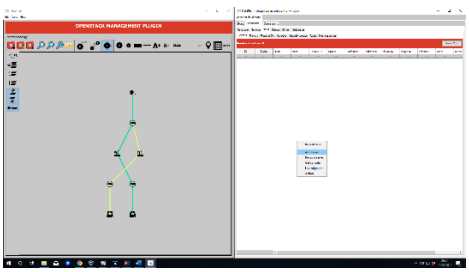

Figura 69. Selección de función clic derecho para añadir una VM (server).

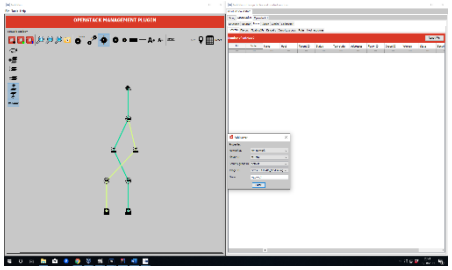

Figura 70. Formulario creación de una VM.

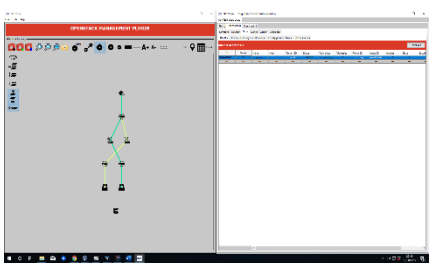

Figura 71. Actualización de topología y tablas tras la creación de la VM.

Una vez se hayan instanciado y se encuentren ACTIVE, podremos comprobar tanto en las tablas como en el panel de la topología (Fig. 72), que las maquinas pertenecen a nuestra subred y que tienen una IP.

Para asegurarnos de que tanto la instanciación como la red están correctamente podemos realizar una simple comprobación como realizar un PING entre las maquinas. Para ello, seleccionamos cada una de las máquinas y hacemos clic derecho, seleccionando la opción *Get console* (Fig. 73).

Las imágenes CirrOs, vienen con un usuario y contraseña por defecto, en este caso, “cirros” y “gocubsgo” respectivamente. Hacemos login, y comprobamos con el comando “ifconfig” que las interfaces disponen las IPs correspondientes (Fig. 74).

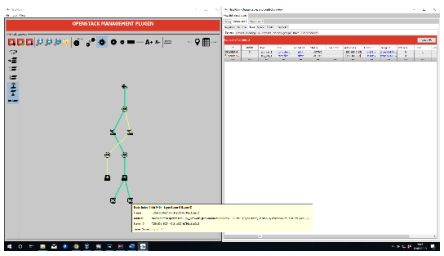

Figura 73. Tooltip con información de la VM añadida.

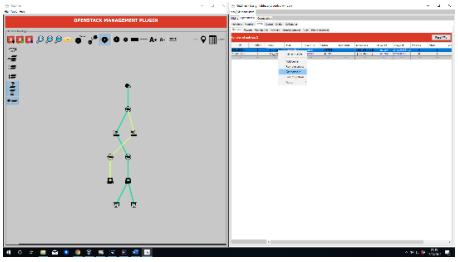

Figura 74. Función clic derecho para obtener la consola de la VM.

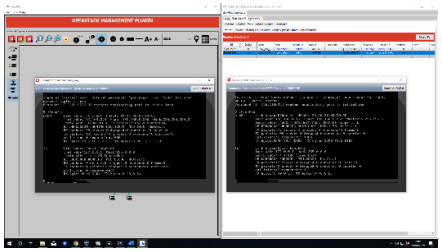

Figura 72. Comando ifconfig en la consola de las dos VMs.

Una vez comprobado, realizamos PING y vemos que efectivamente existe conectividad (Fig. 75).

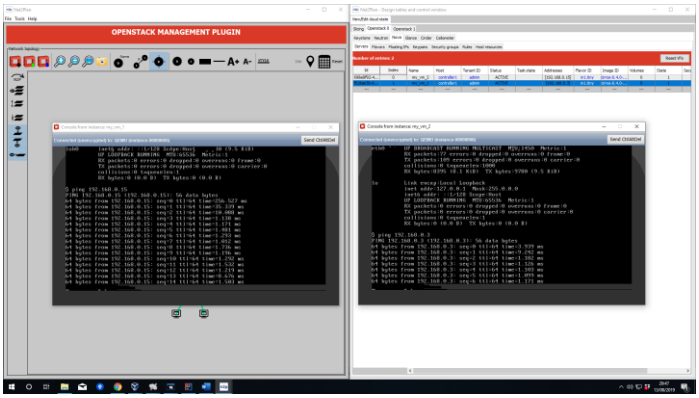

Figura 75. PING entre las VMs con éxito.

5.3 Parte II. Monitorización

La generación de paquetes a través de la red nos permite comprobar ciertas mediciones que la herramienta es capaz de mostrarnos. Para ello debemos irnos la subpestaña *Resources* de la pestaña *Ceilometer* que nos muestra las diferentes fuentes de información disponibles.

Al hacer clic derecho sobre una fuente de información y seleccionar la opción *Show análisis window*, nos aparecerá la ventana de análisis de la fuente con los diferentes parámetros a observar y las gráficas correspondientes (Fig. 76).

Por ejemplo, podríamos seleccionar la fuente que hace referencia la interfaz de red de nuestra. Entre los diferentes parámetros que pueden observar elegimos network.incoming.bytes (Fig. 77) y podemos ver como la gráfica muestra un nivel creciente, ya que el PING sigue funcionando.

Otro ejemplo podría ser, analizar el uso de CPU de una de las máquinas virtuales (Fig. 78).

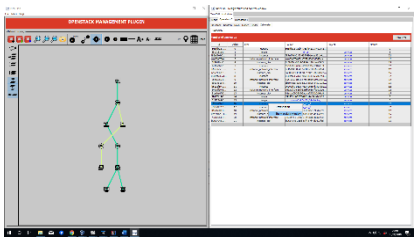

Figura 76. Función clic derecho para mostrar la ventana de análisis y monitorización.

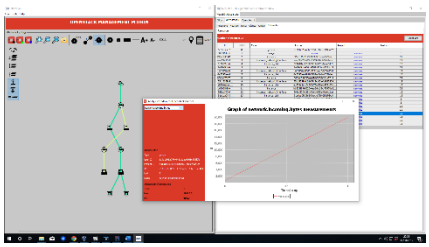

Figura 77. Ejemplo de análisis del parámetro network.incoming.bytes.

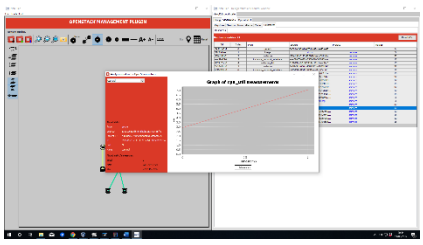

Figura 78. Ejemplo de análisis del parámetro CPU.

5.4 Parte III. Slicing

Este caso de uso también nos permite ver funcionalidades de la pestaña *Slicing*. En la subpestaña *Quota Usage* podríamos ver como existe un uso de los recursos del OpenStack X en el proyecto admin (Fig. 79).

Otro ejemplo como modificar los limites de los recursos del OpenStack X para que no se puedan instanciar más maquinas virtuales de las que ya existen. Como hemos creado 2 maquinas virtuales podemos cambiar el limite de 10 a 2. Para ello, seleccionamos el OpenStack X y hacemos clic derecho. De las opciones elegimos Adjust quotas manual (Fig. 80).

En el formulario resultante cambiamos los recursos Instances de 10 a 2 y pulsamos *Enter* (Fig. 81). Como podemos comprobar en la tabla, se han cambiado el número de instancias permitidas (Fig. 82).

Pero para comprobar si esto es verdad, intentamos instanciar otra maquina con la misma configuración que las anteriores (Fig. 83).

El resultado de pulsar *Enter* esta vez no es el mismo, nos aparece un cuadro de dialogo que nos informa de que se ha superado el límite establecido y que no es posible realizar la instanciación (Fig. 84).

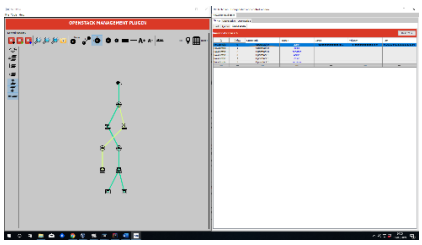

Figura 79. Subpestaña Quota Usage.

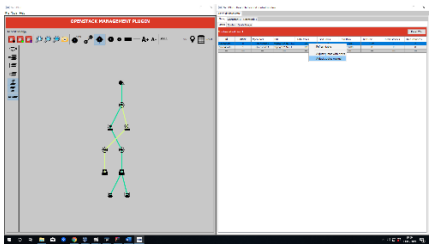

Figura 80. Selección función para ajustar cuotas de los proyectos de un OpenStack manualmente.

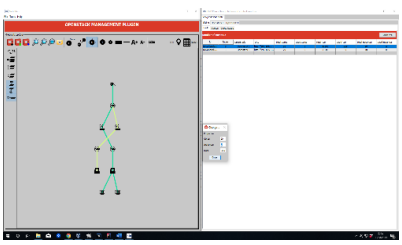

Figura 81. Formulario para modificar cuotas.

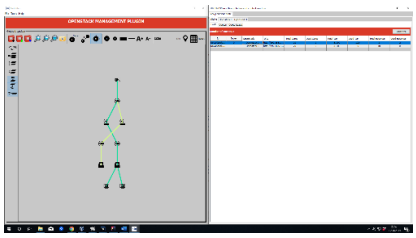

Figura 82. Cuotas modificadas correctamente.

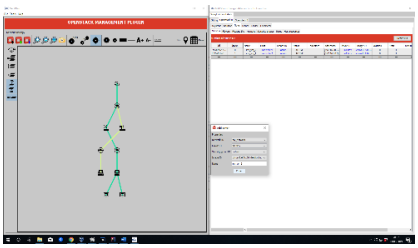

Figura 84. Formulario añadir VM.

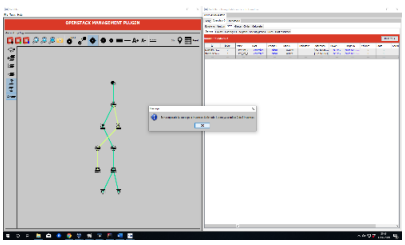

Figura 83. Error al crear una nueva instancia ya que se superan el límite.

Capítulo 6. Conclusiones y líneas futuras

Como conclusiones, podemos decir que los objetivos del proyecto se han cumplido correctamente, ya que para el desarrollo de la herramienta Net2Plan-OpenStack nos ha hecho falta investigar OpenStack y sus componentes, con las relaciones que existen entre ellos.

Por otro lado, este plugin es un software útil para la gestión de uno o múltiples despliegues de OpenStack a través de su interfaz intuitiva, permite el control de sus principales componentes (Keystone, Nova, Neutron …) ofreciendo al usuario que la maneja una serie de funcionalidades que van desde el control de los recursos disponibles en el despliegue/s hasta la instanciación de nuevas máquinas virtuales y/o control de estas a través de una terminal.

Además, nos presenta una representación aproximada de la estructura de nuestra nube respecto a las relaciones de red que existen entre los diferentes elementos que podemos encontrar en OpenStack (router, vm, redes …).

Se han realizado diversas pruebas para la verificación de su correcto funcionamiento, pero una funcionalidad destacable es la monitorización de los elementos de OpenStack a través del componente Ceilometer (Gnocchi) permite al controlador del sistema saber el estado y consumo de los elementos de la red para evitar posibles fallos.

Y finalmente se ha presentado una memoria con la información necesaria para exponer el trabajo realizado, con las partes claves del desarrollo que permitirán continuar su expansión.

6.1 Líneas futuras

Dada la importancia de OpenStack dentro del paradigma de la virtualización de la función de red (NFV), que permite la sustitución de hardware de red especializado con funciones de red virtualizadas (VNF) desplegadas de manera flexible en aparatos de hardware con grandes recursos, la herramienta Net2Plan-OpenStack ofrece las siguientes líneas futuras:

- Diseño de algoritmos que permitan desplegar las VNFs de una cadena de servicio (SC) de la manera óptima en los administradores de infraestructura virtuales (VIM) en base a sus recursos.
- Diseño de reportes que permitan conocer todo el estado de un despliegue de OpenStack a modo de informe.
- Nuevas funcionalidades que permita regular de manera automática el consumo excesivo de los recursos de un VIM con la monitorización y migración de las VM que contienen las VNFs.
- Ampliación de los componentes de OpenStack que la herramienta puede controlar e implementar nuevas funcionalidades para los componentes que ya existen.

Capítulo 7. Bibliografía

[1] OpenStack. https://www.openstack.org/ [1 Sep 2020]

[2] Net2Plan. http://www.net2plan.com/ [1 Sep 2020]

[3] RDO. http://www.rdoproject.org/ [1 Sep 2020]

[4] Red Hat Enterprise Linux. https://www.redhat.com/es/ [1 Sep 2020]

[5] OpenStack 4 Java. http://www.openstack4j.com/ [1 Sep 2020]

[6] Dan Radez. (2016). OpenStack Essentials, Second Edition. Birmingham: Packt Publishing.

[7] Gnocchi. http://www.gnocchi.xyz/ [1 Sep 2020]

[8] Net2Plan – OpenStack plugin. https://github.com/girtel/Net2Plan-OpenStack [1 Sep 2020]

[9] M. Garrich, et al., "The Net2Plan-OpenStack Project: IT Resource Manager for Metropolitan SDN/NFV Ecosystems," OFC, 2019.

[10] Open source mano. https://osm.etsi.org/ [1 Sep 2020]

[11] Open Network Operating System (ONOS) https://onosproject.org/ [1 Sep 2020]

[12] M. Garrich, et al., "Open-source Network Optimization Software in the Open SDN/NFV Transport Ecosystem," JLT, 2018.

[13] Selvaraj, Vinoth Kumar. (2017). OpenStack Bootcamp: Build and Operate a Private Cloud Environment Effectively. Birmingham: Packt Publishing.