UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA DE SISTEMAS DE TELECOMUNICACIÓN

Curso Académico 2018/2019

Trabajo Fin de Grado

DESPLIEGUE DE NODOS NFV UTILIZANDO TACKER SOBRE OPENSTACK

Autor : Daniel Crespo Beltrán

Tutor : Dr. Pedro De Las Heras Quirós

Trabajo Fin de Grado/Máster

Despliegue de Nodos NFV Utilizando Tacker sobre OpenStack

Autor : Daniel Crespo Beltrán

Tutor : Dr. Pedro de las Heras Quirós

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2019, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2019

Dedicado a mi familia

II

Resumen

El crecimiento en los últimos años de las redes de ordenadores, telefonía y datos ha supuesto un gran avance en las comunicaciones. A su vez, ese crecimiento tan grande ha implicado un incremento en la complejidad y coste de satisfacer esas necesidades de cómputo y red. Así surgió la virtualización de funciones de redes como solución a ésta necesidad.

Éste proyecto introduce los conceptos realacionados a la tecnología NFV y todo lo que la rodea, presenta una solución de orquestación y administración, y deja la puerta abierta a otros proyectos o investigaciones descubrir en profundidad lo que ésta tecnología puede brindar junto a otras.

Summary

The evolution in the last years of computer networks, telephony and data has meant a great advance in communications. At the same time, such large growth has led to an increase in the complexity and cost of meeting those computing and network needs.

This project introduces the concepts related to NFV technology and everything around it, presents an orchestration and management solution, and leaves the door open to other projects or researches to discover in depth what this technology can provide along with others.

Índice general

Índice de figuras

Capítulo 1

Introducción

Se ha diseñado una arquitectura de SDN (Red Definida por Software) que permita hacer una demostración de despliegue de diversos nodos NFV (Virtualización de Funciones de Redes) para introducir el proyecto de OpenStack Tacker y algunas de sus características principales.

1.1. NFV

Network Function Virtualization, en español, Virtualización de Redes es un concepto que utiliza las tecnologías de virtualización para virtualizar todo tipo de funciones de redes, que puedan conectarse o escalar para crear servicios. De ésta manera, se sustituyen firewalls, routers, switches o balanceadores de cargas por máquinas virtuales corriendo en un entorno de virtualización estándar (de software libre como OpenStack, o propietario como VMware). Así el tiempo, dificultad y costes de expansión se ven reducidos.
Ésto permite a las empresas expandir sus redes de ordenadores o comunicaciones al no requerir hardware específico o propietario en su arquitectura.

1.2. Estructura de la memoria

La memoria se divide en los siguientes capítulos:

- Introducción: Explica brevemente el proyecto, la estructura de la memoria y los objetivos que se han perseguido en su desarrollo.

- Estado del arte: Descripción a modo de introducción de las diversas tecnologías utilizadas o implicadas en el proyecto. Se explica en mayor profundidad la tecnología NFV y la iniciativa ETSI MANO.

- Diseño e implementación: Detalla la instalación de todos los componentes y configuraciones de la infraestructura para la realización de las pruebas de funcionalidad. Finalmente se realizan diferentes despliegues de nodos NFV para mostrar algunas virtudes y capacidades de Tacker.

- Conclusiones: Retrospección de todas las decisiones y pasos realizados para la realización del proyecto. Se comprueba como *checklist* los objetivos alcanzados, conocimientos aplicados y lecciones aprendidas.

1.3. Objetivos

El proyecto consiste en hacer una introducción y demostración general para mostrar las virtudes de NFV con Tacker y las tecnologías implicadas. Se pretende entonces, enseñar desde la infraestructura física hasta la virtualizada, el control de dicha infraestructura y cómo gestionar recursos para utilizarlos y realizar funciones de red.

- **Diseño de la arquitectura y dimensionamiento**
  El dimensionamiento y diseño debe ir acorde a unos requisitos, pero que permita la demostración de funcionalidades avanzadas.

- **Configuración e instalación**
  Configurar todo el entorno e infraestructura que compone el proyecto y conseguir realizar una instalación satisfactoria.

- **Despliegue y ciclo de vida de NFV**
  Explotación de Tacker y otros servicios para utilizar recursos con funciones de red.

Capítulo 2

Estado del arte

2.1. NFV

Network Function Virtualization, en español, Virtualización de Redes es un concepto que utiliza las tecnologías de virtualización para virtualizar todo tipo de funciones de redes, que puedan conectarse o escalar para crear servicios. De ésta manera, se sustituyen firewalls, routers, switches o balanceadores de cargas por máquinas virtuales corriendo en un entorno de virtualización estándar (de software libre como OpenStack o Xen, o propietario como VMware). Así el tiempo, dificultad y costes de expansión se ven reducidos.

La **ETSI** (*European Telecommunication Standards Institute*) es un grupo de estandarización creado en 1988 que elabora estándares para la información y para las tecnologías de la información. El Grupo de Especificaciones para la Industria de (NFV ETSI ISG NFV), es el encargado de desarrollar las especificaciones y arquitecturas de diversas funciones dentro de las redes de comunicación. Éste grupo fue formado en sus orígenes por AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica, y Verizon; al que más tarde se unirían otros operadores de redes, fabricantes de equipos y otros provedores.
Desde su formación en 2013, han realizado más de 100 publicaciones especificando estándares para las diferentes partes de una arquitectura NFV.

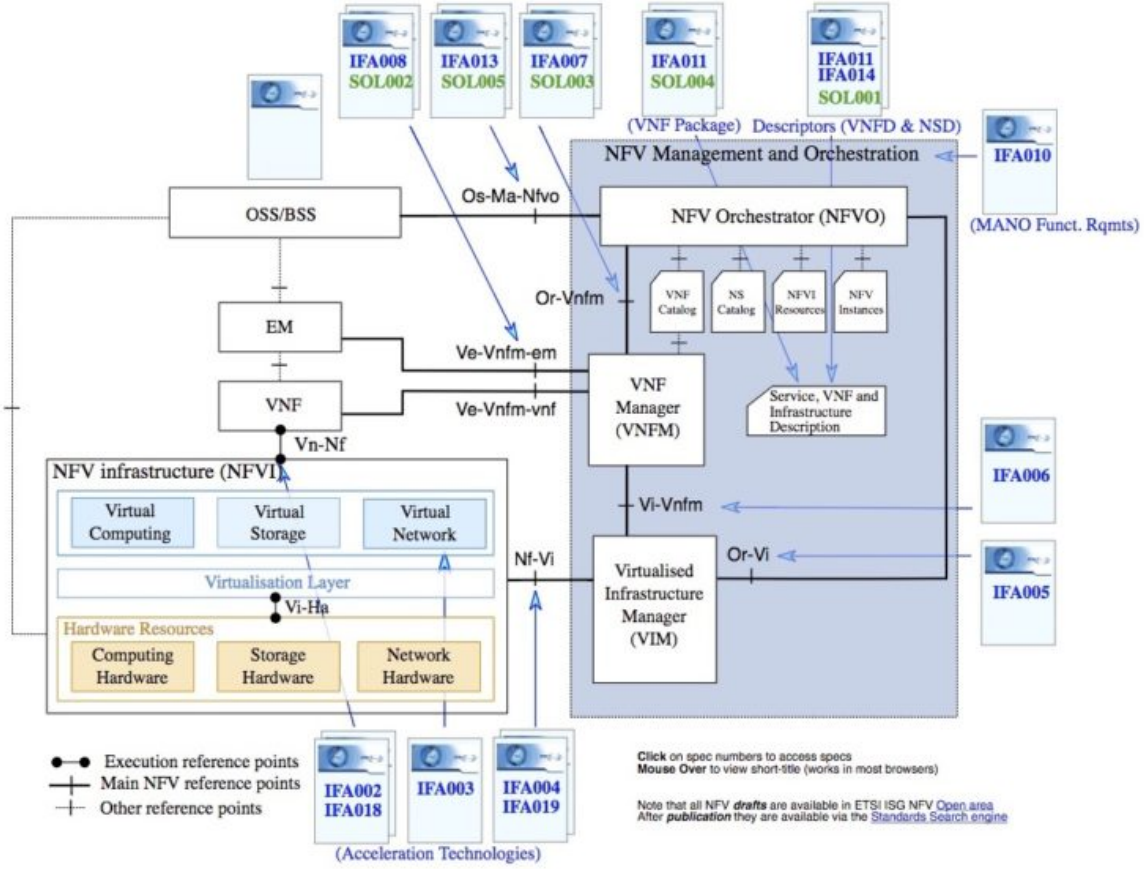

Figura 2.1: Estándares de ETSI ISG para NFV [1]

2.2. ETSI MANO

ETSI MANO (Management and Orchestration) es una iniciativa del grupo ETSI para desarrollar un stack de software alineado con la arquitectura NFV planteada por este mismo grupo. Como se puede observar a la derecha de la figura 2.1, los componentes claves son el Orquestador NFV, el Manager NFV y el VIM. [2]

La **orquestación de recursos** cumple la función de asegurar que el cómputo, redes y almacenamiento están disponibles para los servicios o funciones de red. El orquestador se encarga de las autotizaciones, despliegues y conexión de las funciones de red, interactuando con las APIs subyacentes de la infraestructura sobre la que reside la arquitectura, así se proporciona otra capa de abstracción en la que no solo desconocemos el hardware sobre el que se ejecuta la arquitectura, si no que también podemos obviar cuál es el software que administra todos esos recursos. En éste punto, debe haber un equilibrio o consciencia de que todos éstos nuevos servicios de red,

deben coexistir y operar con otras funciones como sistemas de soporte a operaciones y soporte a negocio (OSS/BSS), además de otros sistemas tradicionales.

Toda la escalabilidad, agilidad y rapidez que nos ofrece convertir una red de comunicaciones física en una virtual, crea nuevos desafíos que el **Manager NFV** ayuda a solventar. Es el componente que que permite la operación e interconexión entre los componentes NFV, y a su vez con la infraestructura virtualizada. También es el responsable de la gestión del ciclo de vida de los VNF. El mercado de VNFM está lleno de soluciones privativas de empresas de telecomunicaciones como Nokia, Cisco y Huawei, pero la mayoría de empresas buscan una solución de software libre para implementar, que permita su libertad para la adaptación a su entorno concreto. Aquí, surge **Tacker** como NFVM para OpenStack, siguiendo los estándares de ETSI MANO. Sin embargo, el proyecto mas extendido y conocido es **Open Source MANO (OSM)**[1], ya que cuenta con una gran interoperabilidad y compatibildiad e integración con múltiples plataformas VIM.

2.3. OpenStack

OpenStack es un sistema cloud para contolar grandes cantidades de recursos de cómputo, red y almacenamiento contenidos en un centro de procesamiento de datos para proporcionar infraestructura como servicio (IaaS[2]). Es Open Source, y se organiza por diferentes proyectos de la comunidad, en el que cada uno de ellos se ocupa de desarrollar, añadir funcionalidades, y mantener un servicio[3].

[1] `https://osm.etsi.org/`

[2] IaaS: Infraestructure as a Service

[3] `https://www.openstack.org/software/project-navigator/openstack-components#openstack-services`

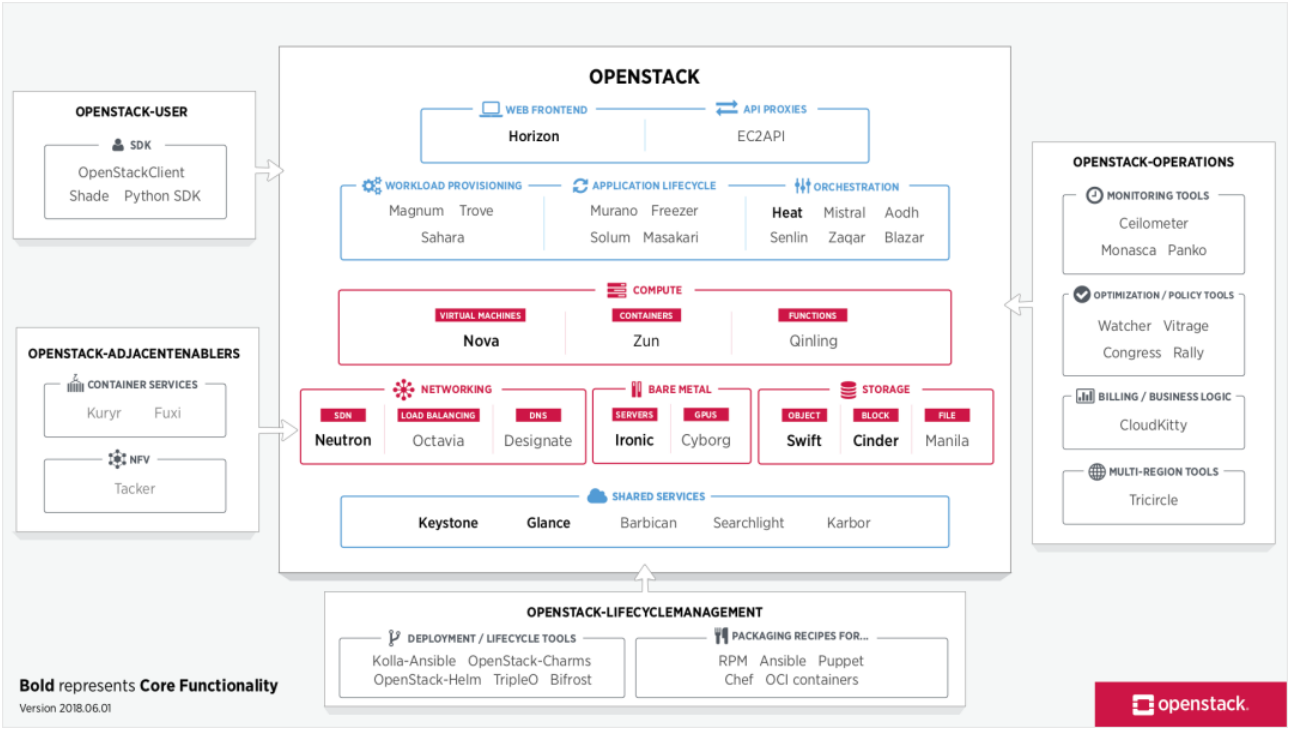

Figura 2.2: Diagrama de los proyectos de OpenStack [3]

2.4. Tacker

Tacker[4] es un proyecto oficial de OpenStack. Se encarga del despliegue y orquestación de cada nodo NFV dentro de una plataforma de infraestructura. Está compuesto por un *Manager* para crear elementos NFV, un *Orchestrator* para administrarlos y operar con ellos, y un *Catalog* para definir NFV. Tacker se basa en la estructura ETSI MANO y utiliza plantillas TOSCA en su catálogo para la especificar elementos y servicios.

[4] `https://wiki.openstack.org/wiki/Tacker`

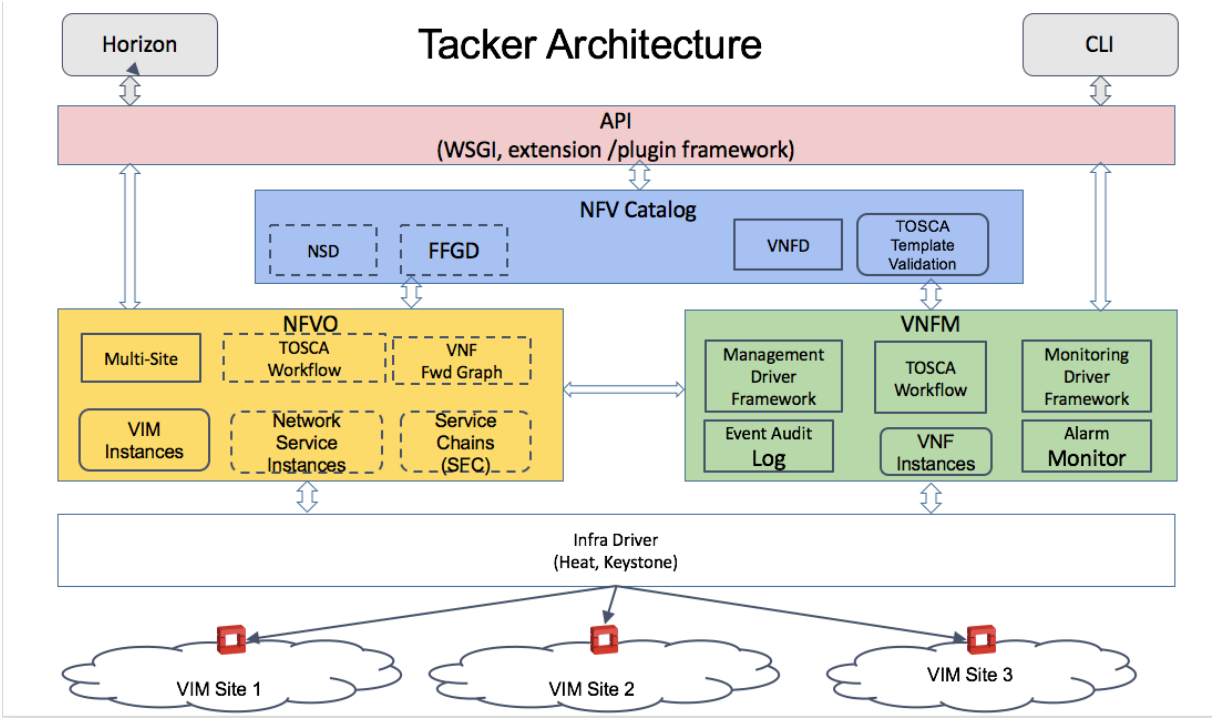

Figura 2.3: Arquitectura de Tacker [4]

2.5. Kolla

Kolla[5] es otro de éstos proyectos, y proporciona una serie de herramientas para despliegue y operación de OpenStack sobre contenedores. Utiliza la tecnología Docker para desplegar cada uno de los servicios de OpenStack dentro de un contenedor, y Ansible para automatizar y configurar esos despliegues.

Se puede usar para crear arquitecturas de OpenStack de gran tamaño, con multitud de configuraciones diferentes, ya que permite una personalización detallada de despliegues de servicios en nodos, y todo ello preparado para entornos de producción con alta disponibilidad.

2.6. Docker

Docker es un proyecto de software libre desarrollado en Go para desplegar y ejecutar aplicaciones dentro de contenedores. Éste proyecto se compone de dos partes: *Docker Engine* y *Docker Hub*.

[5] `https://wiki.openstack.org/wiki/Kolla`

Docker engine es la tecnología que permite aislar una capa del kernel de linux y utilizar recursos del sistema para que los contenedores se ejecuten dentro de la máquina (virtual o física). En éste punto es importante explicar qué es un contenedor: un contenedor es una unidad de software que contiene o empaqueta código y todas las dependencias necesarias para ejecutar una aplicación. De ésta manera, se consigue pasar de una máquina virtual y un sistema operativo al uso, a una capa de abstracción superior que solo necesita unos mínimos recursos del sistema para ejecutarse; consiguiendo rapidez de despliegue y escalado.

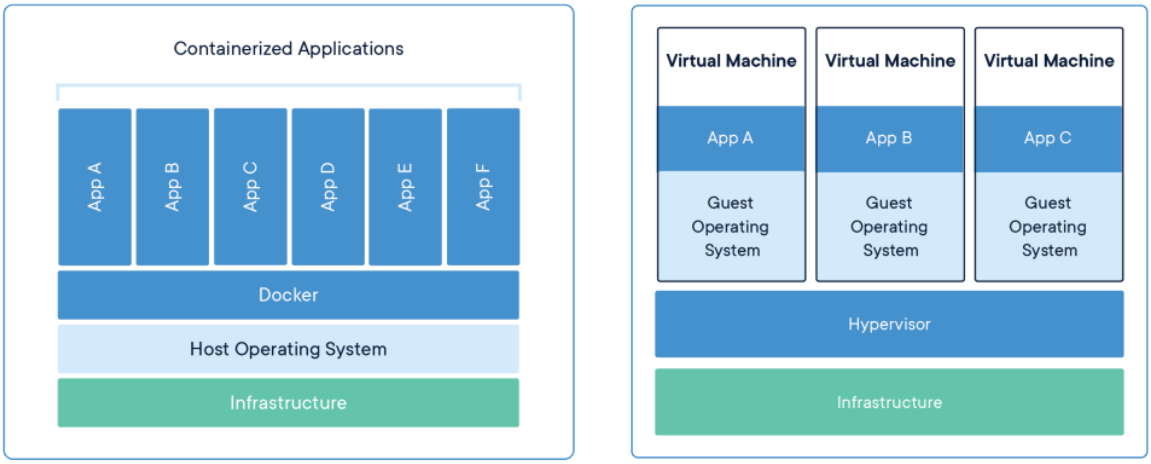

Figura 2.4: Comparativa de contenedor y máquina virtual [5]

El código de aplicaciones y librerías de dependencias que hemos nombrado, se encuentran ya empaquetadas en contenedores, y con éstos contenedores ya "cerrados"se ha creado un repositorio público llamado **Docker Hub**. Parecido a Git, podemos hacer un **pull** a Docker Hub y descargar a nuestro registro local una imágen de Docker de una aplicación. Será de Docker Hub de donde obtengamos las imágenes de contenedor de Tacker y resto de componentes de OpenStack.

2.7. Ansible

Ansible es una tecnología que permite la administración y configuración de máquinas, y permite también automatización de despliegues de infraestructura y aplicaciones.

Se basa en conexiones ssh[6] hacia las máquinas (aunque también es compatible con Windows usando WinRM) y ejecuta instrucciones de Python en la máquina. Por lo tanto, Ansible no necesita agentes ni clientes, tan solo que las máquinas remotas tengan instalado Python y que tengan compartidas claves ssh o establezcan algún otro tipo de relación de confianza.
Utiliza **yaml** para escribir playbooks, que son conjuntos de instrucciones y tareas. Éste es un ejemplo de un Playbook de Ansible con un Task:

```
---
- name: Hello World Playbook
  hosts: desarrollo
  tasks:
  - name: Print Hello World!
        shell: echo Hello World Ansible
```

Ansible utiliza módulos para realizar las diferentes acciones, tan solo hay que escribir el nombre del módulo seguido de dos puntos y las acciones concretas del módulo. Tiene módulos para praćticamente todos los comandos equivalentes de Linux, y si no existiera, se podría usar el módulo *shell*, o desarrollar el módulo a medida.
Otro aspecto destacable de Asible, es el inventario. Se trata de un fichero que contiene el hostname o IP de los hosts o nodos sobre los que se van a ejecutar los playbook. Los nodos se pueden agrupar con grupos, tipos, funciones, etc. Un ejemplo de un fichero de inventario ser:

```
[desarrollo]
nodo2.domain.com
nodo4.domain.com
[produccion]
nodo7.es.com ansible_port=2233
nodo9.es.com ansible_port=2233
```

[6] ssh: Secure Shell. Protocolo para acceso remoto en máquinas Linux.

Capítulo 3

Diseño e implementación

El proyecto se ha pensado y diseñado desde el pricipio como introducción a NFV con Tacker para futuros proyectos o formaciones académicas, por ésto la facilidad de despliegue, administración y posibles desarrollos eran el pricipal objetivo. En éste aspecto, la instalación con Kolla cumplía cumplía éstos requisitos, al ser cuestión de minutos desplegar una infraestructura OpenStack, desinstalar y redesplegar si fuera necesario.

3.1. Arquitectura general

Puesto que el proyecto no va a requerir grandes cantidades de cómputo, los requisitos hardware son bajos. Kolla permite desplegar OpenStack all-in-one (todos los servicios corriendo en una misma máquina) como arquitectura mínima, pero se ha implementado en dos nodos para mejorar la comprensión de la arquitectura y solución de éste proyecto.

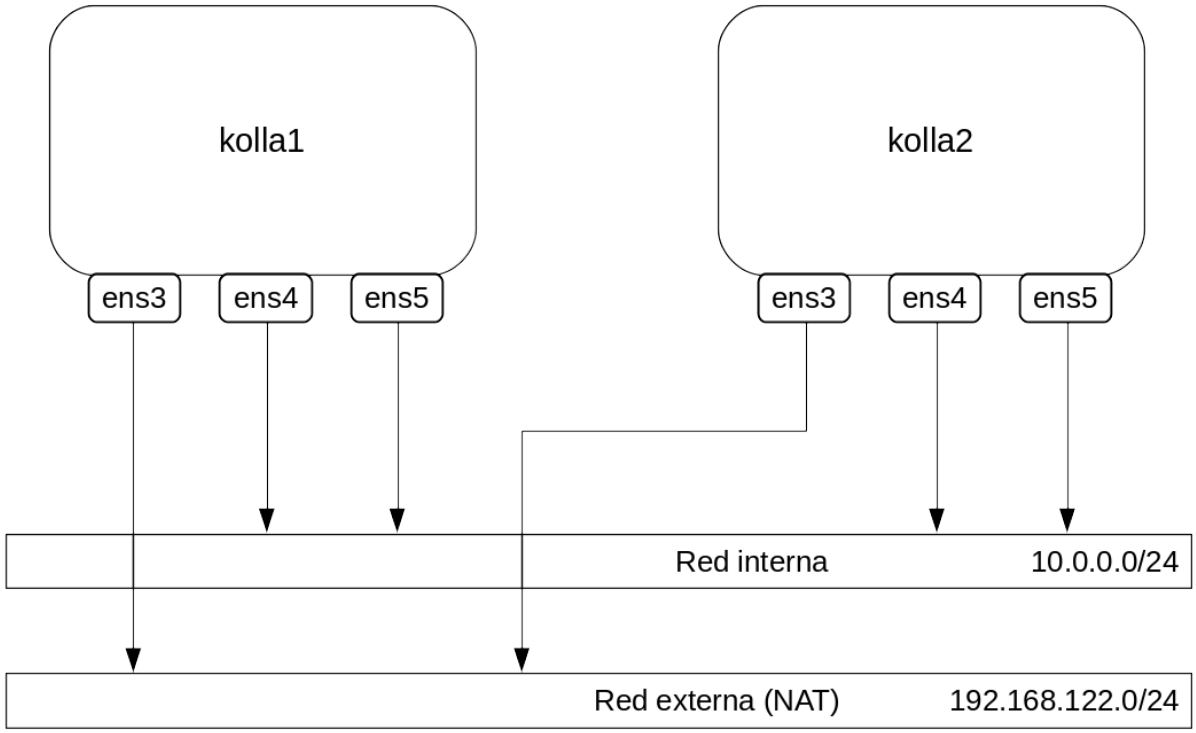

Figura 3.1: Arquitectura de los nodos del proyecto

3.1.1. Nodos

Como host, disponemos de un ordenador de sobremesa, con las siguientes especificaciones:

CPU	Intel i7 7700 (8 cores)
Memoria	16 GB 2400 MHz
Almacenamiento	500 GB

Cuadro 3.1: Especificaciones nodo host

Por su similitud con Red Hat Enterprise Linux (muy usado en entornos empresariales y servidores); se ha elegido como sistema operativo **Centos 7**. Se trata de una distribución de Linux basada en RHEL, muy extendida y estable. Mantenida y compilada por la comunidad, desde 2014 es patrocinada por Red Hat.

	Nodo1	Nodo2
Cores virtuales	2 cores	2 cores
Memoria	4 GB	4 GB
Disco	60 GB	60 GB
NIC	3	3

Cuadro 3.2: Especificaciones de nodos virtualizados

3.2. Configuración

3.2.1. Virtualización

Antes de empezar a instalar los paquetes de virtualización y un manager de máquinas virtuales en nuestro host, debemos asegurar que la virtualización está habilitada en la CPU y BIOS:

```
$ grep --color -E 'vmx|svm' /proc/cpuinfo
```

En la salida de éste comando, debemos observar vmx para procesadores Intel o svm para procesadores AMD. Si no se obtiene alguna de éstas salidas, se deberá configurar la BIOS/UEFI para habilitarlo. Es recomendable consultar el manual de la placa base o las especificaciones del procesador.

Los paquetes de virtulización que necesitaremos son:

```
$ yum install kvm libvirt
```

La herramienta que nos facilitará la creación y administración de éstas máquinas virtuales, sin tener que entrar en profundidad a conocer el comando *virsh* es **virt-manager**. Ofrece interfaz de usuario para administrar KVM en nuestro host, crear y administrar las máquinas virtuales, las redes, el almacenamiento y todo lo relacionado con el entorno virtualizado.

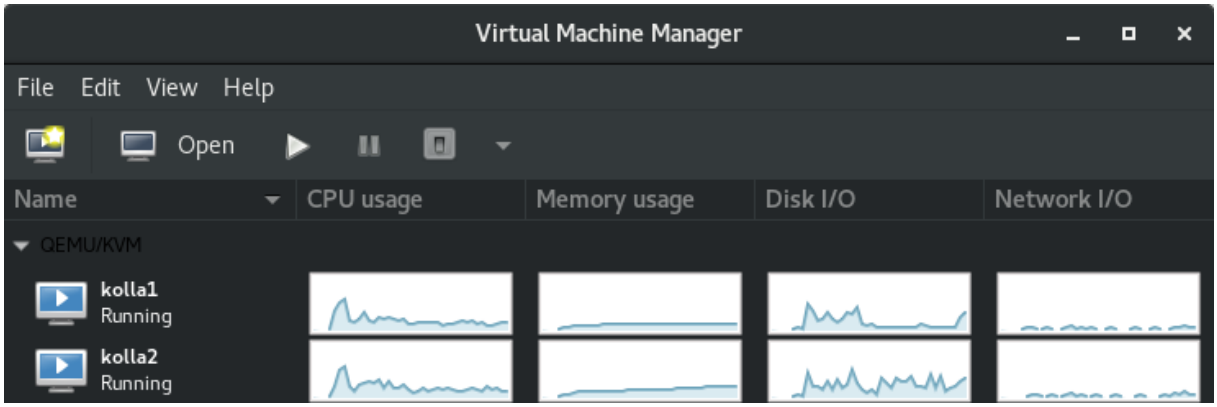

Figura 3.2: Pantalla de administración Virtual Machine Manager

Al tratarse de un paquete incluido en la mayor parte de las distribuciones de Linux más conocidas, se instala con el gestor de paquetes:

```
$ yum install virt-manager
```

Dada la arquitectura del proyecto, dispondremos máquinas virtuales, corriendo contenedores Docker, que a su vez ejecutarán instancias (vm de OpenStack). Para no tener probleamas, hay que comprobar si la virtualización anidada está habilitada [6]:

```
$ cat /sys/module/kvm_intel/parameters/nested
```

La salida de dicho comando nos indicará Y/N en caso de tener activada o desactivada dicha opción.

En caso de obtener N, se deberá activar de la siguiente manera.[1].

Llegados a éste punto ya se pueden crear las máquinas virtuales con las especificaciones descritas arriba. Siendo punto crítico y la única peculiaridad que tienen las máquinas sobre las que se desplegará OpenStack, es que tienen 3 interfaces; 2 en una red privada y otra con acceso a Internet por NAT[2].

[1] `https://docs.fedoraproject.org/en-US/quick-docs/using-nested-virtualization-in-kvm/index.html`

[2] NAT: Network Adress Translation

3.2.2. Preparación de los nodos

Al igual que en el host, en los nodos, se ha instalado también Centos 7 Minimal Server para tener mayor control de los paquetes instalados y entender en profundidad el funcionamiento. Al tratarse de un despliegue para pruebas y desarrollo, que contendrá pocas instancias a la vez, la instalación del sistema operativo puede hacerse por defecto, sin requerir de espacio en disco en concreto para los puntos de montaje[3].

Antes de empezar a instalar cualquier paquete, siempre es bueno actualizar todo el sistema operativo para corregir bugs, solucionar posibles incompatibilidades y por seguridad.

```
$ yum update -y
```

Si al hacer el upgrade, se instalan nuevos paquetes del kernel, se debe reiniciar los nodos.
A continuación, se instala pip[4] y se actualiza:

```
$ yum install epel-release
$ yum install python pip
$ pip install -U pip
```

Después unas dependencias para el correcto funcionamiento de Ansible y python, el propio paquete de Ansible y actualizar con pip:

```
$ yum install python-devel libffi-devel openssl-devel
$ yum install libselinux-python gcc
$ yum install ansible
$ pip install -U ansible
```

Con todo ésto, ya se dispondrá de la base del sistema operativo, paquetes y dependencias. Ahora tan solo queda la configuración de red y ssh entre las máquinas.

Ejemplos de configuración de las interfaces de red para el nodo kolla2 (en ambos nodos, la configuración de redes y nombres de interfaces es la misma:
Interfaz red NAT:

```
[root@kolla2]# cat /etc/sysconfig/network-scripts/ifcfg-ens3
```

[3]Para instalaciones más complejas con mayor número de servicios (o almacenamiento en bloque), se recomienda usar LVM (*Logical Volume Manager*) para la gestión de discos.

[4]pip (Pip Install Packages) es un gestor para instalar y administrar paquetes de Python.

```
TYPE="Ethernet"
BOOTPROTO="none"
DEFROUTE="yes"
NAME="ens3"
UUID="efa2ea9d-6db2-4d4a-973c-453eab7c7db0"
DEVICE="ens3"
ONBOOT="yes"
IPADDR="192.168.122.200"
PREFIX="24"
GATEWAY="192.168.122.1"
DNS1="8.8.8.8"
```

Interfaz 1 red privada:

```
[root@kolla2]# cat /etc/sysconfig/network-scripts/ifcfg-ens4
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=no
PEERDNS=yes
PEERROUTES=yes
IPV4_FAILURE_FATAL=no
NAME=ens4
UUID=d5f0f01a-92ed-46d4-8c99-cf0d1d6366ff
DEVICE=ens4
ONBOOT=yes
```

Interfaz 2 red privada:

```
[root@kolla2]# cat /etc/sysconfig/network-scripts/ifcfg-ens5
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
NAME=ens5
```

```
UUID=3923c6f7-97b8-4931-90a9-fd2818e07232
DEVICE=ens5
ONBOOT=yes
IPADDR=10.0.0.30
PREFIX=24
GATEWAY=10.0.0.1
DNS1=8.8.8.8
```

Para tener claro qué fichero de interfaz configurar como NAT o externo, hay que prestar atención a la dirección MAC que KVM ha dado a la interfaz. Desde Virtual Manager podemos conocer la dirección MAC, para después desde el *guest* ejecutar **ip addr show** y conocer el nombre de la interfaz que el sistema operativo a asignado a dicha MAC.

A continuación, se debe editar el fichero */etc/hosts* para incluir el hostname y la IP privada del resto de nodos y él mismo:

```
[root@kolla2 network-scripts]# cat /etc/hosts
127.0.0.1       localhost
::1             localhost localhost.localdomain
10.0.0.20       kolla1
10.0.0.30       kolla2
```

Tan sólo recordar, que el fichero de *hosts* contiene el nombre e IP de las máquinas que el localhost podrá resolver sin necesidad de DNS. Es de utilidad para la configuración de Kolla y Ansible, además de necesario para los servicios de OpenStack.

Por último se genera una clave ssh y se comparte con los otros nodos. Ésto se debe realizar para cada uno de los nodos, para que todos tengan acceso sin contraseña al resto y así facilitar otras tareas de administración o debugging cuando haya que lanzar comandos de un nodo a otro.

```
$ ssh-keygen
$ ssh-copy-id <nodo>
```

3.3. Instalación de OpenStack con Kolla

Llegados aquí, ya disponemos de las máquinas virtuales operativas y con la configuración requerida por la arquitectura. Antes de seguir, se puede realizar un *snapshot*[5] a las máquinas y tener una copia de seguridad con las máquinas limpias y configuradas, para que después podamos reinstalar OpenStack o tocar cualquier configuración adicional sin problemas.

Se clonan los repositorios de kolla y kolla-anisble de Github [7][8]. Se puede especificar el branch con **-b** <**nombre** >, para el caso del proyecto se usa master (que no hace falta especificar nada):

```
$ git clone https://github.com/openstack/kolla
$ git clone https://github.com/openstack/kolla-ansible
```

En el mismo directorio, se usa pip para instalar los requerimientos de kolla y kolla-ansible:

```
$ pip install -r kolla/requirements.txt
$ pip install -r kolla-ansible/requirements.txt
```

Dentro del directorio que se ha creado al clonar el repositorio de git, figuran unos ficheros de configuración por defecto. Para realizar una copia de ellos, y moverlo a la ubicación en la que kolla espera encontrarlos, se deben copiar a */etc/kolla* para editarlos. Al lanzar el ansible de Kolla, tomará esa configuración contenida en */etc/kolla/globals.yml* y */etc/kolla/passwords.yml*. Copiamos también los ficheros de inventario (all-in-one y multinode) al directorio actual.

```
$ mkdir -p /etc/kolla
$ cp -r kolla-ansible/etc/kolla/* /etc/kolla
$ cp kolla-ansible/ansible/inventory/* .
```

3.3.1. Ficheros de configuración

globals.yml

El archivo */etc/kolla/globals.yml* almacena la configuración de redes y servicios que van a ser desplegados al lanzar la instalación con Ansible. Éste archivo, tal como viene al hacer el clone de github, contiene parámetros y valores por defecto (comentados con #). Para cambiar

[5] snapshot: Copia de seguridad o 'instantánea' del sistema en un momento determinado

ése valor, se descomenta la línea y se asigna el nuevo valor deseado.
Ahora pasamos a describir algunas variables y explicar su valor en el proyecto.

kolla_internal_vip_address y **network_interface**

La variable kolla_internal_vip_address debe ser una IP sin usar que esté dentro de la subred privada (10.0.0.0 en nuestro caso). De ésta manera las API[6] de todos los servicios se comunicarán por ésta subred mediante una la IP flotante que indiquemos. Con variable network_interface indicamos a kolla en que interfaz debe configurar la VIP[7]. Adicionalmente, todas las vxlan y tuneles así como el tráfico red de almacenamiento, también correrán en ésta red e interfaz. En la arquitectura del proyecto, sus valores quedan de la siguiente manera:

```
kolla_internal_vip_address: "10.0.0.100"
network_interface: "ens5"
```

Aquí puede surgir el problema que las interfaces de cada nodo se llamen de maneras diferentes. Ésto se soluciona en el fichero de inventario de ansbile. Otra solución sería crear un bond para la interfaz en cada nodo, y declarar el nombre del bond en network_interface.

Para escenarios más grandes y complejos, kolla permite configurar en detalle las redes internas y externas, y el tráfico de los servicios:

```
kolla_external_vip_address: "{{kolla_internal_vip_address}}"
kolla_external_vip_interface: "{{network_interface}}"
api_interface: "{{network_interface}}"
storage_interface: "{{network_interface}}"
cluster_interface: "{{network_interface}}"
tunnel_interface: "{{network_interface}}"
dns_interface: "{{network_interface}}"
```

neutron_external_interface

Con ésta variable, indicamos a kolla que interfaz debe usar neutron como puerto externo. Ésta interfaz no debe tener configurada ninguna IP. Al tratarse de puerto externo para OpenStack, aunque no tenga ninguna dirección configurada con el networkmanager o en el fichero de

[6] API: application programming interface
[7] VIP: virtual IP address

configuración, ésta debe ser la interfaz que pertenezca a la red NAT de nuestro hipervisor. En el escenario del proyecto queda de la siguiente manera (ver sección 3.2.2):

```
neutron_external_interface: "ens4"
```

neutron_plugin_agent Aquí se configura cuál va a ser el plugin que va a controlar los agentes de red de neutron. Las posibles opciones son: openvswitch, linuxbridge, vmware_nsxv, vmware_nsxv3, vmware_dvs y opendaylight.

```
neutron_plugin_agent: openvswitch
```

enable_neutron_provider_networks Ésta opción permite habilitar la creación de redes tipo *provider network* en OpenStack. Ésto hace posible conectar instancias de cómputo directamente a las redes físicas (que están "por debajo" de OpenStack). Ésto es de utilidad, ya que se desplegrán VNF con interfaces en redes internas, pero será necesario acceder a éstas instancias mediante la interfaz de red de la red externa (red NAT), pero con la limitación de no usar floating ip.

```
enable_neutron_provider_networks: yes
```

openstack_release La versión de OpenStack que se va a desplegar. Indica a kolla que versión de imágenes de Docker Hub[8] deberá descargar. Es importante que la versión elegida sea la misma que la de kolla y kolla-ansible.

```
openstack_release: rocky
```

Servicios de OpenStack

La sección de configuración de servicios de OpenStack, permite seleccionar que servicios se van a desplegar. Los servicios básicos necesarios como keystone, nova, glance, neutron, etc; están descomentados el el fichero y habilitados por defecto. A continuación se muestran los servicios necesarios para el proyecto y otros que servirán de utilidad:

- tacker: Manager y orquestador de NFV.
- neutron_sfc: Habilita que neutron tenga la capacidad de realizar SFC. **Service function chaining** es una funcionalidad de SDN para crear una cadena de servicios de red.

[8]Docker Hub `https://hub.docker.com`

- mistral: Mistral es el proyecto de servicios de flujos de trabajo. Permite planificar tareas de cualquier tipo como cron, despliegues o migraciones; dependiendo de métricas, valores de monitorización o valores horarios establecidos.

- redis: Instala y permite utilizar base de datos en memoria.

- barbican: Barbican es el proyecto de gestión y almacenamiento de secretos como contraseñas o claves.

- heat: Heat es el proyecto para crear servicios y aplicaciones cloud a partir de plantillas. Para el caso de estudio del proyecto, se utilizan plantillas TOSCA[9].

- horizon_mistral y horizon_tacker. Son add-on al dashboard Horizon que se instala por defecto. Facilita mucho la visualización de plantillas y descriptores que se van definiendo. Éstos últimos son meramente ilustrativos, ya que la mayoría del trabajo se realizará vía cli.

La manera de indicar a kolla qué servicios se desean desplegar o no es:

```
enable_tacker: "yes"
enable_ceph: "no"
```

Con la herramienta de monitorización Weave Scope [9], podemos tener vista de alto nivel de los servicios desplegados y la manera en la que se comunican, puesto que se trata de la visualización de Docker.

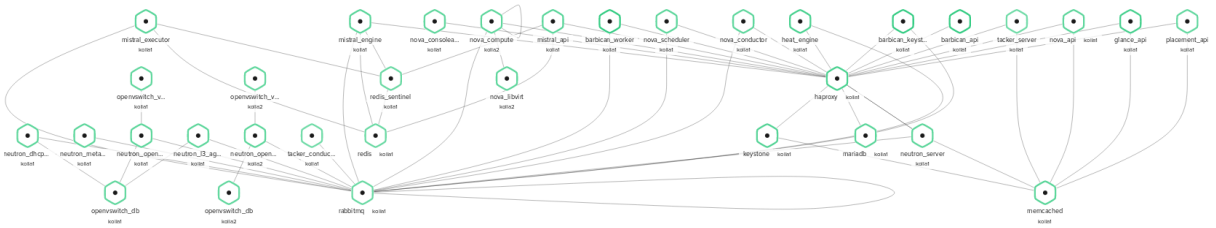

Figura 3.3: Vista de los servicios de OpenStack y sus conexiones

[9]TOSCA: Topology and Orchestration Specification for Cloud Applications

Inventario de hosts

Como hemos explicado antes, kolla realiza una instalación mediante Ansible. Para ejecutarse necesita un *playbook*, donde figuran las instrucciones que se van a realizar; y el fichero (o inventario) de hosts. En el inventario es donde se declaran las máquinas sobre las que se van a ejecutar acciones, agrupado por grupos. Así, distinguiendo por grupos de hosts, a su vez se pueden realiazr unas tareas u otras, dependiendo del grupo al que pertenezcan.

Kolla proporciona un fichero **all-in-one**, que declara que todo el despligue de todos los contenedores de servicios de OpenStack se van a realizar sobre la máquina donde se ejecute (localhost). Es muy sencillo para pruebas simples y desarrollos, pero requiere que se el despliegue se realice en una máquina con unos requisitos mayores de CPU y memoria.

El otro fichero que se proporciona es **multinode**. Éste permite declarar las máquinas dependiendo del diseño elegido y agrupar por cómputo, red, control, storage, etc. Para la arquitectura del proyecto, éstos son algunos apartados del fichero (el nodo donde se encuentra multinode es kolla1, que es el nodo desde donde se hará el despliegue, por lo tanto en el fichero figura como localhost):

```
[control]
localhost       ansible_connection=local
[network]
localhost       ansible_connection=local
[compute]
kolla2
[storage]
localhost       ansible_connection=local
[deployment]
localhost       ansible_connection=local
```

Hay que resaltar también que éste fichero, permite una configuración mucho más fina de los nodos, llegando a una granularidad prácticamente de qué servicio se desea que corra cada nodo. Para éste apartado, la herramienta anterior de monitorización, permite una visualización mas

granulada, a nivel de hosts, mostrando el número de contenedores que se ejecutan en cada uno; además de obtener también el consumo de recursos de cada nodo y cada servicio.

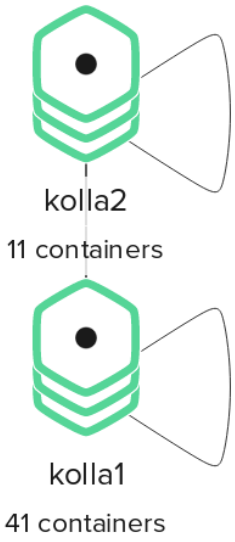

Figura 3.4: Contenedores y nodos de la arquitectura

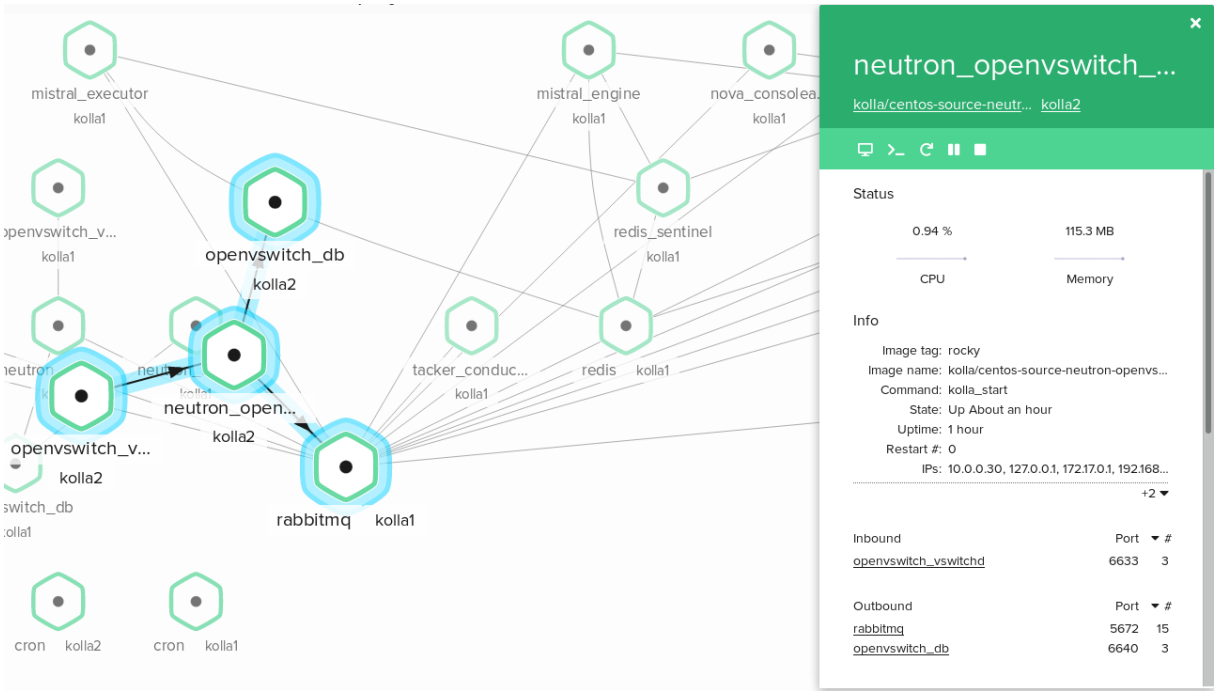

Figura 3.5: Ejemplo de la vista de Weave Scope de un servicio

Passwords

El otro fichero de configuración necesario para la instalación es */etc/kolla/passwords.yml* Contiene las contraseñas que van a ser usadas por kolla en la instalación y gestionadas posteriormente por Keystone. Por defecto no contiene ninguna contraseña, para generarlas, usamos el comando:

```
$ ./kolla-ansible/tools/generate_passwords.py
```

Después, por comodidad podemos cambiar cualquiera de dichas contraseñas. Es conveniente modificar **keystone_admin**, ya que será usada a menudo (accediendo a Horizon, por ejemplo).

3.3.2. Lanzando la instalación

Ahora tenemos todo el entorno preparado y configurado, pero antes de lanzar el playbook de instalación, se deben hacer unas últimas comprobaciones de conectividad y checks que proporcionan ansible y kolla.

```
$ ansible -i multinode all -m ping
```

Nos debe devolver una salida SUCCESS en ambos nodos. Ésto es tan solo para comprobar que hay conectividad entre nodos y que las claves ssh están bien compartidas y almacenadas en *.ssh/known_hosts* del home del usuario root.

Hasta aquí la configuración e instalaciones que hemos realizado era para que kolla-ansible funcionase, y el escenario cumpliese los requisitos de la arquitectura. A partir de aquí, se delega el resto de instalación de paquetes, dependencias e imágenes de Docker a kolla-ansible.

Para completar el despliegue, nos dirigimos al directorio al que hicimos git clone (el home del usuario para nuestro caso):

```
$ cd ~/kolla-ansible/tools
```

Y ejecutamos el playbook **bootstrap-servers**, que instalará Docker versión de la comunidad con el gestor de paquetes correspondiente a la distribución Linux que se haya elegido (recordemos Centos y yum para el caso del proyecto):

```
$ ./kolla-ansible -i \
 ../ansible/inventory/multinode bootstrap-servers
```

Éste playbook, también se asegurará que Docker está habilitado y ejecutándose, comprobará reglas de firewall del sistema operativo, y se asegurará que el resto de dependencias ya hayan sido cumplidas en todos los nodos.

En éste punto por fin disponemos del entorno configurado al 100 % listo para instalar OpenStack con todos los servicios que hemos declarado en globals.yml. Aún así podrían surgir incompatibilidades de servicios de OpenStack o alguna dependencia (entre servicios que se van a desplegar) que por desconocimiento o descuido al configurar nos hayamos saltado.

Así el playbook **prechecks**, nos indicará de manera rápida si la instalación va a ser satisfactoria o no (hay que decir que aún siendo muy rápida, el despliegue de un OpenStack básico con kolla, puede durar 20-30 minutos):

```
$ ./kolla-ansible -i \
 ../ansible/inventory/multinode prechecks
```

Por fin, podemos instalar:

```
$ ./kolla-ansible -i
 ../ansible/inventory/multinode deploy
```

El proceso de instalación en los nodos depende mucho de la velocidad de conexión a internet, porque tiene que descargar todas las imágenes de cada servicio (docker pull), el número de servicios desplegados y las características de los host en cuanto a disco (si es flash), memoria y cpu. En todo momento, ansbile indica cúal es la tarea que se está realizando, pero simultáneamente, se puede abrir ventanas de línea de comandos adicionales de cada host, y observar cómo se van descargando las imágenes de Docker y desplegando los contenedores:

```
$ watch -n1 docker images
$ watch -n1 docker ps
```

Una vez que la instalación acabe, se debe dar un par de minutos a que los contenedores acaben de arrancar y que cada servicio esté funcionando y con comunicación al resto. Mientras, se puede generar el fichero de credenciales de OpenStack de la siguiente manera:

```
$ ./kolla-ansible -i \
 ../ansible/inventory/multinode post-deploy
```

Ésto generará el fichero */etc/kolla/admin-openrc.sh*, que contiene variables como usuario y proyecto admin, la URL del endpoint de autenticación en OpenStack y la contraseña de admin; que se cargarán como variables de entorno.

Concretamente para el proyecto y uso de **tacker** se necesita instalar el cliente o cli que complementa a los comandos de OpenStack:

```
$ sudo pip install python-tackerclient
```

3.3.3. Primeros pasos en OpenStack

El primer paso, y para hacerlo vistoso, podemos acceder al dashboard de Openstack, *horizon*. En un navegador web, escribimos la dirección IP del nodo controlador: **http://10.0.0.20** Nos dirigirá a la página de login:

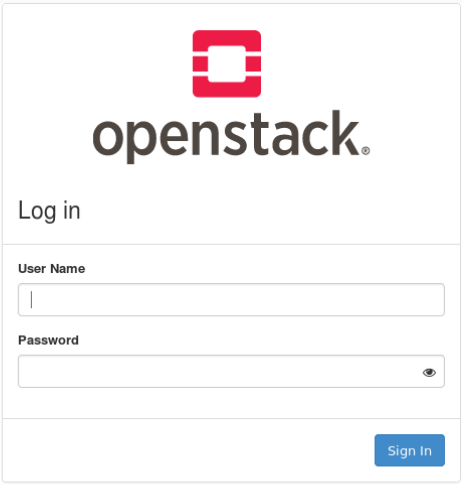

Figura 3.6: Pantalla login OpenStack

El usuario es admin, y para obtener la contraseña, en el terminal del nodo controlador, debemos escribir:

```
$ cat /etc/kolla/passwords.yml | grep keystone_admin
```

Para hacer login en la consola o cli[10], accedemos al terminal de la máquina desde donde realizamos el despliegue, y cargamos las variables de entrono que se encuentran en el fichero /etc/kolla/admin-openrc.sh:

```
$ source /etc/kolla/admin-openrc.sh
```

El fichero admin-openrc.sh se puede mover a cualquier otro directorio para que sea más accesible, o a otra máquina que tenga conectividad con en el entorno de OpenStack.
Una vez cargadas las variables de entorno en la sesión del terminal, ya es posible ejecutar cualquier comando de OpenStack. Para conocerlos u obtener ayuda, se puede consultar el manual[11].

Hemos comentado anteriormente que kolla proporciona OpenStack sobre contenedores preparado para puesta en producción, pero por su facilidad de despliegue y rapidez, es también muy usado (al igual que devstack[12]) para desarrollo y pruebas de CI[13]. Por ésta razón, el OpenStack recién desplegado debe ser también configurado de una manera rápida y ágil. Ésto significa crear redes, proyectos, imágenes, sabores, etc; que suele ser un proceso laborioso teniendo que lanzar multitud de comandos. Así, kolla también proporciona un script de post-instalación llamado **init-runonce**, ubicado en */root/kolla-ansible/tools*. El uso de éste script es muy sencillo, ya que solo requiere la customización de tres variables para hacerlas coincidir con el escenario desplegado, dependiendo de la configuración de redes de los nodos:

```
EXT_NET_CIDR='10.0.0.0/24'
EXT_NET_RANGE='start=10.0.0.150,end=10.0.0.199'
EXT_NET_GATEWAY='10.0.0.1'
```

Una vez cambiado ésto, ejecutamos el script y esperamos a que finalice (debería tardar tan solo 2-3 minutos).
Las redes que crea el script por defecto son una red pública(public1) y otra privada(demo-net). Adicionalmente y ya pensando en futuros requerimientos, se ha añadido al script init-runonce, la creación de otra red privada(demo-net2) y su correspondiente subred:

```
openstack network create \\
```

[10] cli: command line interface

[11] https://docs.openstack.org/python-openstackclient/pike/cli/command-list.html

[12] https://docs.openstack.org/devstack/latest/

[13] CI: continuous integration

```
--provider-network-type vxlan demo-net2
openstack subnet create \\
--subnet-range 10.0.20.0/24 --network demo-net2 \\
--gateway 10.0.20.1 --dns-nameserver 8.8.8.8 demo-subnet2
```

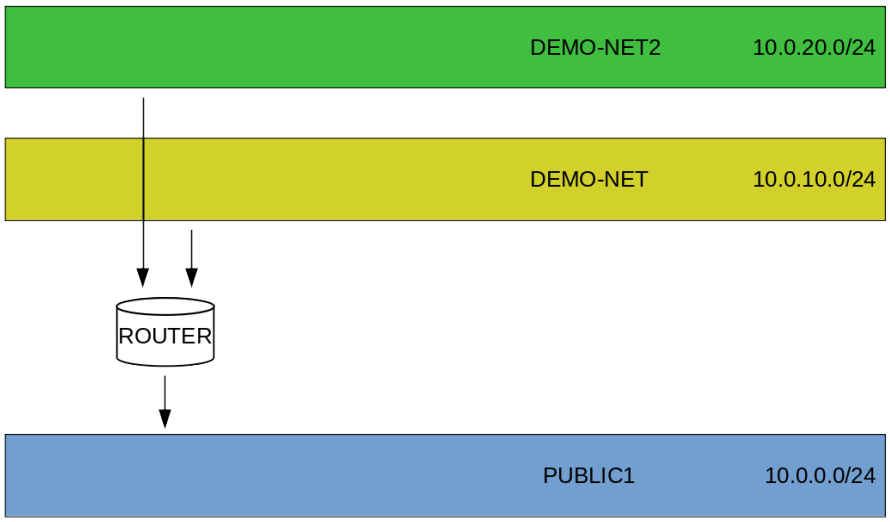

Figura 3.7: Redes creadas dentro de OpenStack

3.4. Tacker

Recordando y observando la arquitectura de Tacker (ver figura 3.5), los componentes básicos son el orquestador, el manager y el controlador de infraestructura.

El **controlador de infraestructura**, mediante keystone(servicio de identidades) y heat(servicio de orquestación y despliegue de máquinas basado en plantillas) proporciona el control de la plataforma hipervisor sobre la que se desplegarán los nodos y funciones de red.

De ésta manera, proporcionaremos a Tacker el "acceso" a OpenStack para que actúe a la vez como soporte para Tacker e infraestructura sobre la que Tacker desplegará NFV. Así se define el VIM que se usará en el proyecto:

```
$ cat kolla-vim.yaml
```

```
auth_url: http://10.0.0.100:35357/v3
username: admin
password: kHYui1jHMpKtcm8blbG6tT0syz1VgSiBMwY7GOkP
project_name: admin
project_domain_name: Default
user_domain_name: Default
```

Siendo **auth_url** el endpoint de keystone, y los credenciales y proyecto del usuario admin.
Para crear el VIM, darle nombre y que sea usado por defecto, utilizamos el comando de OpenStack (complementado con el cliente de Tacker que instalamos):

```
$ openstack vim register --config-file ./kolla-vim.yaml \
  --is-default kolla-vim
```

En éste punto, podemos entender como desde un mismo servidor de Tacker, se puede administrar múltiples entornos VIM, centralizando el control de varias infraestructuras OpenStack. Ésto permite a las compañías el aislamiento del plano de control y tener un NFV distribuido. La distribución de NFV puede ser geográfica, por tipos, por funciones, etc.

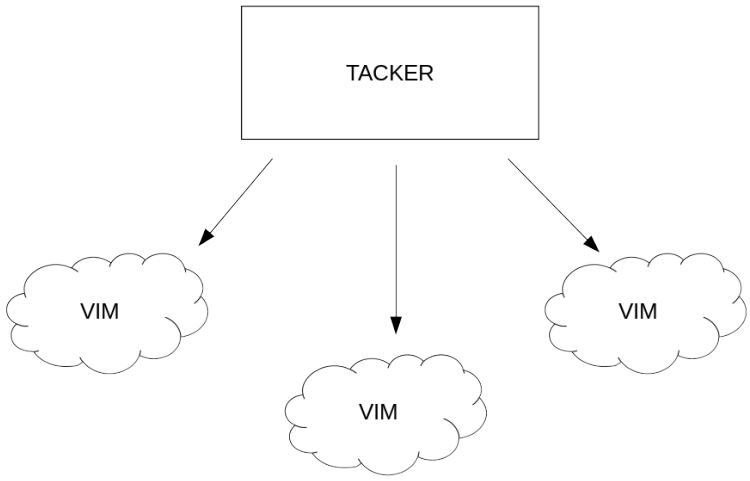

Figura 3.8: Tacker centralizado controlando múltiples sites

El siguiente componente que vamos a estudiar es el **catálogo**. El catálogo es un almacén o repositorio de descriptores tipo VNF definidos en plantillas TOSCA. Éstas plantillas se declaran en archivos *yaml* y luego se convierten en descriptores, pasando primero por un proceso de validación.

El siguiente yaml es un ejemplo muy sencillo de plantilla TOSCA, definiendo una máquina virtual con *ip forward* activado [10]:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
description: Simple VM
metadata:
  template_name: sample-vnfd-ipforward
topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
        image: cirros
        flavor: m1.tiny
        availability_zone: nova
        user_data_format: RAW
        user_data: |
          #!/bin/sh
          echo 1 > /proc/sys/net/ipv4/ip_forward
    CP11:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1
```

```
    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: demo-net
        vendor: Tacker
}
```

Para añadirlo al catálogo, usamos el comando de OpenStack:

```
$ openstack vnf descriptor create --vnfd-file \
  ./sample-vnfd-ipforward.yaml sample-vnfd-ipforward
```

vnf descriptor create es un comando específico de Tacker, y lo proporciona la cli cuando instalamos python-tackerclient. Todos los comandos específicos de Tacker para despliegue y administración de toda la arquitectura NFV son lanzados desde la cli de manera similar. Recordemos que en la instalación, también se habilitó el complemento de Horizon de Tacker, pero como siempre, la línea de comandos ofrece más posibilidades y es más completa.
Para crear el nodo VNF, usamos el comando *create*, le indicamos el nombre del descriptor que usar, y damos nombre al nuevo VNF:

```
$ openstack vnf create --vnfd-id sample-vnfd-ipforward \
  kolla-sample-vnf
```

Podemos verificar la creación de la instancia; bien usando **nova**, para corroborar que la instancia se ha creado; o usando **tacker**, para comprobar que el nodo VNF existe y tacker lo tiene en su base de datos:

```
$ openstack server list
$ openstack vnf list
$ openstack vnf show <id>
```

Ahora vamos a plantear un caso de uso mas aproximado a la realidad: despliegue de un nodo NFV con una imagen de **OpenWrt**[14] al que se le inyectan reglas de firewall por ssh [11]:
El primer paso es descargar una imagen de OpenWrt y cargarla en OpenStack con Glance:

[14]OpenWrt es un sistema operativo Linux embebido en dispositivos de red. URL: `https://archive.openwrt.org/chaos_calmer/15.05.1/x86/kvm_guest/openwrt-15.05.1-x86-kvm_guest-combined-ext4.img.gz`

```
$ wget <url> -O openwrt-x86-kvm_guest-combined-ext4.img.gz
$ gunzip openwrt-x86-kvm_guest-combined-ext4.img.gz
$ openstack image create OpenWRT --disk-format qcow2 \
--container-format bare \
--file openwrt-x86-kvm_guest-combined-ext4.img \
--public
```

El archivo yaml que define el descriptor es el siguiente [12]:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
description: OpenWRT firewall
metadata:
  template_name: OpenWRT
topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 512 MB
      properties:
        image: OpenWRT
        config: |
          param0: key1
          param1: key2
        mgmt_driver: openwrt
    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        order: 0
```

```
      management: true
    requirements:
      - virtualLink:
          node: VL1
      - virtualBinding:
          node: VDU1
  CP2:
    type: tosca.nodes.nfv.CP.Tacker
    properties:
      order: 1
    requirements:
      - virtualLink:
          node: VL2
      - virtualBinding:
          node: VDU1
  CP3:
    type: tosca.nodes.nfv.CP.Tacker
    properties:
      order: 2
    requirements:
      - virtualLink:
          node: VL3
      - virtualBinding:
          node: VDU1
  VL1:
     type: tosca.nodes.nfv.VL
     properties:
       network_name: public1
       vendor: Tacker
  VL2:
    type: tosca.nodes.nfv.VL
```

```
      properties:
        network_name: demo-net
        vendor: Tacker
    VL3:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: demo-net2
        vendor: Tacker firewall
```

En la plantilla, se especifica cuantas interfaces va a tener el nodo y a que redes van a estar conectadas, los recursos virtuales de cpu y memoria que va a disponer, la imagen desde la que se va a desplegar, y lo más importante para su uso: *mgmt_driver: openwrt*. El driver de OpenWrt se encuentra instalado en Tacker, por lo que al especificar ésta línea, se puede añadir reglas de firewall en éste caso a la instancia VNF en el momento de despliegue, o modificarlas de la misma manera más tarde. A continuación, se especifica la configuración de firewall en formato yaml [13]:

```
vdus:
  VDU1:
    config:
      firewall: |
        package firewall
        config defaults
            option syn_flood '1'
            option input 'ACCEPT'
            option output 'ACCEPT'
            option forward 'REJECT'
        config zone
            option name 'lan'
            list network 'lan'
            option input 'ACCEPT'
            option output 'ACCEPT'
```

```
        option forward 'ACCEPT'
    config zone
        option name 'wan'
        list network 'wan'
        option input 'REJECT'
        option output 'ACCEPT'
        option forward 'REJECT'
        option masq '1'
        option mtu_fix '1'
    config forwarding
        option src 'lan'
        option dest 'wan'
    config rule
        option name 'Allow-ssh'
        option src 'wan'
        option proto 'tcp'
        option dest_port '22'
        option family 'ipv4'
        option target 'ACCEPT'
    config rule
        option name 'Allow-ping'
        option src 'wan'
        option proto 'icmp'
        option icmp_type 'echo-request'
        option family 'ipv4'
        option target 'ACCEPT'
```

Como se puede observar, éste fichero es muy simple, y tan solo especifica reglas como permitir ping y ssh.

Los siguientes pasos son crear el descriptor y crear el VNF indicando a Tacker cuál es el fichero de configuración que debe usar para el firewall.

```
$ openstack vnf descriptor create \
--vnfd-file vnfd-openwrt.yaml vnfd-openwrt
$ openstack vnf create --vnfd-name vnfd-openwrt \
--config-file config-openwrt-firewall.yaml firewall
```

Es importante observar a que interfaz se ha asignado como gestión (*management: true*), en nuestro caso CP1 que se conectará a la subnet public1. Aquí cobra sentido lo que se explicó anteriormente sobre las redes de OpenStack, ya que dicha red es del tipo provider network y es compartida con los hosts. Así el nodo controller tendrá conectividad con el firewall y podrá hacer ssh para inyectar las reglas al driver.

Transcurrido aproximadamente un minuto, la instancia ya se ha desplegado y estará operativa. Para comprobar que las reglas del firewall presentes son las correctas, listamos los vnf para obtener la IP, accedemos por ssh como root(el usuario es root, y no tiene contraseña). Las configuración se encuentra en el fichero /etc/config/firewall.

```
$ openstack vnf list
$ ssh root@<ip del firewall>
$ cat /etc/config/firewall
```

Capítulo 4

Conclusiones

4.1. Consecución de objetivos

Ahora repasaremos los objetivos propuesto al comienzo del proyecto y analizaremos de qué manera se han alcanzado.

- **Diseño de la arquitectura y dimensionamiento**

  El diseño de la arquitectura ha dado unos resultados positivos, habiendo elegido correctamente los servicios instalados. Durante el desarrollo del proyecto y posteriormente, no se ha echado en falta ninguna funcionalidad que se no se hubiera desplegado; pero por otra parte todos los servicios elegidos han sido necesarios y han jugado cierto papel en el transcurso de los despliegues de nodos NFV o en su ciclo de vida.

  Por otra parte, el dimensionamiento ha resultado ser adecuado, ya que no se han experimentado bajadas de rendimiento en el sistema ni falta de recursos durante la creación de nodos NFV. También podemos concluir que no se ha sobredimesionado la infraestrctura, ya que los recursos elegidos son hasta cierto punto asequibles y al alcance de cualquier laboratorio.

- **Configuración e instalación**

  La instalación y configuración de los sistemas operativos e hipervisores se puede considerar trivial con alguna excepción muy puntual. La preparación y configuración de OpenStack y sus servicios para adaptarlo al entorno diseñado presentó alguna complicación en el apatado de creación de redes. Éste punto siempre es muy crítico en instalaciones de

OpenStack, y un error aquí supone tener que desinstalar, configurar otra vez e instalar de nuevo. La complicación que se encontró fue configurar las redes internas y externas de Neutron para que las instancias creadas tuviesen conectividad con el exterior. Una vez solucionado ésta parte y teniendo claras las redes que se han dado a Neutron, la instalación y posteriores pruebas de conectividad fueron satisfactorias.

- **Despliegue y ciclo de vida de NFV**
  Se ha conseguido desplegar múltiples instancias como funciones de red y realizar gestiones y configuraciones posteriores mediante la línea de comandos de OpenStack y Tacker, además de introducir configuraciones por ssh automáticamente.

4.2. Lecciones aprendidas

En el desarrollo de todo el proyecto, he aprendido y entendido en profundidad diferentes conceptos sobre redes de ordenadores y virtualización. No tan relacionado con el proyecto, pero no menos importante es el conocimiento de cómo funciona el desarollo de proyectos de la comunidad, su estructuración y como cooperan entre ellos.

4.3. Trabajos futuros

Éste proyecto ha planteado la puesta en funcionamiento y manejo de una infraestructura de virtualización de funciones de redes que pueda ser administrada mediante Tacker. Los posibles trabajos futuros que se plantean a la finalización de éste proyecto pueden ser un estudio de otras funcionalidades de Tacker:

- Manejo del ciclo de vida de NFV.
- VNF Forwarding Graph (VNFFG) y Network Services Descriptor (NSD).
- Network Service Chaining.

Bibliografía

[1] Imagen Estándares de ETSI ISG para NFV
https://www.etsi.org/technologies/nfv

[2] Definición Open Source MANO
https://www.etsi.org/technologies/nfv/open-source-mano

[3] Imagen Proyectos de OpenStack
https://www.openstack.org/software/

[4] Imagen Arquitectura de Tacker
https://wiki.openstack.org/wiki/Tacker

[5] Imagen Máquina Virtual VS Contenedor
https://www.docker.com/resources/what-container

[6] Comprobar y habilitar virtualización anidada
https://docs.fedoraproject.org/en-US/quick-docs/
using-nested-virtualization-in-kvm/index.html

[7] Repositorio de GitHub de Kolla
https://github.com/openstack/kolla

[8] Repositorio de GitHub de Kolla-Ansible
https://github.com/openstack/kolla-ansible

[9] Página Oficial y Descarga de Weave Scope
https://www.weave.works/docs/scope/latest/introducing/

[10] Plantilla TOSCA con ip forward
`https://github.com/openstack/kolla-ansible/blob/master/contrib/demos/tacker/deploy-tacker-demo`

[11] Despliegue nodo NFV con OpenWRT
`https://docs.openstack.org/tacker/latest/install/deploy_openwrt.html`

[12] Descriptor de despliegue de firewall
`https://github.com/openstack/tacker/blob/master/samples/tosca-templates/vnfd/tosca-vnfd-openwrt.yaml`

[13] Configuración de firewall
`https://github.com/openstack/tacker/blob/master/samples/tosca-templates/vnfd/tosca-config-openwrt-firewall.yaml`