Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DISEÑO E IMPLEMENTACIÓN DE UNA PROPUESTA DE ARQUITECTURA DE COMPUTACIÓN EN LA NUBE BASADA EN OPENSTACK KILO PARA ENTORNOS DE BIG DATA

TRABAJO FIN DE MÁSTER

Mercedes González González

2015

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UNA PROPUESTA DE ARQUITECTURA DE COMPUTACIÓN EN LA NUBE BASADA EN OPENSTACK KILO PARA ENTORNOS DE BIG DATA

Autor
Mercedes González González

Director
Joaquín Salvachúa Rodríguez

Departamento de Ingeniería de Sistemas Telemáticos

2015

Resumen

OpenStack constituye una solución de Computación en la Nube libre y de código abierto para el despliegue de nubes públicas, híbridas y privadas que proporcionan Infraestructura como Servicio (IaaS). Está compuesto por un conjunto de proyectos y componentes que interactúan entre sí para permitirles a los usuarios el auto-aprovisionamiento bajo demanda de recursos virtuales de computación, red y almacenamiento. El presente Trabajo de Fin de Master se enfoca en un estudio de los distintos componentes y servicios de OpenStack, particularmente, los correspondientes a la última versión de esta solución lanzada en Mayo de 2015 bajo el nombre de Kilo y se propone como objetivo el diseño e implementación de una arquitectura multi-nodo que, además de proveer los servicios básicos de aprovisionamiento de recursos de Nube, incluya soporte para el procesamiento de Big Data. Para facilitar el despliegue de la arquitectura propuesta y como parte de este trabajo, se programaron scripts en Python que automatizan el despliegue de la misma.

Abstract

OpenStack is an open source Cloud Computing solution to build public, hibrid and private clouds that provide Infrastructure-as-a-Service. It is composed by several projects and components that interact amongst each other to allow user to self-provision compute, network and storage virtual resources on demand. This Master Thesis focuses on the study of the distinct components and services that integrate with OpenStack, particularly, those included in the last release launched in May, 2015 under the name of Kilo and the main goal of this work is to design and implement a multi-node cloud architecture, that besides providing the basic cloud resources, manages to support Big Data processing as a service to cloud users. To ease the deployment of the proposed cloud architecture and as a result of this work, several Python scripts were programmed in order to automate the deployment tasks.

Índice general

Índice de figuras

Índice de tablas

Siglas

CAGR: Compound Annual Growth Rate

IT: Information Technology

NIST: National Institute of Standards and Technology

SaaS: Software as a Service

PaaS: Platform as a Service

IaaS: Infrastructure as a Service

IDC: International Data Corporation

AWS: Amazon Web Services

VDI: Virtual Desktop Infrastructure

MVC: Modelo-Vista-Controlador

API: Application Programming Interface

REST: Representational State Transfer

KVM: Kernel-based Virtual Machine

UEC: Ubuntu Enterprise Cloud

VMDK: Virtual Machine Disk

VDI: Virtual Disk Image

VHD: Virtual Hard Disk

OVF: Open Virtualization Format

DMTF: Distributed Management Task Force

CPU: Central Processing Unit

LXC: Linux Containers

OS: Operating System

VLAN: Virtual Local Area Network

DNS: Domain Name Server

DHCP: Dynamic Host Configuration Protocol

NAT: Network Address Translation

SNAT: Source Network Address Translation

DNAT: Destination Network Address Translation

VXLAN: Virtual Extensible LAN

GRE: Generic Routing Encapsulation

LAMP: Linux, Apache, Mysql, PHP

HOT: Heat Orchestration Template

URL: Uniform Resource Locator

UI: User Interface

GUI: Graphical User Interface

DAL: Data Access Layer

NFV: Network Function Virtualization

SSH: Secure SHell

Introducción

La Computación en la Nube se ha convertido en los últimos años en una de las tecnologías más populares y desplegadas a escala mundial. Louis Columbus, en su reporte publicado en Forbes, titulado "*Roundup Of Cloud Computing Forecasts And Market Estimates, 2015*" plantea que se espera que el gasto en infraestructura y plataforma de Computación en la Nube crezca a una tasa compuesta anual (Compound Annual Growth Rate - CAGR) de 30% desde 2013 hasta el 2018 en comparación con un CAGR de 5% para la empresa global de IT. [1]

Por otro lado, de acuerdo con los resultados obtenidos en el estudio "*2015 State of the Network Study – Technology Adoption Trends & Their Impact on the Network*", en el que participaron 230 ejecutivos y profesionales del sector IT y que fue publicado en enero de 2015 por IDG Enterprise, empresa consultora de marketing, las cuatro principales iniciativas en las que los ejecutivos del sector IT están planeando enfocar a sus organizaciones a lo largo del presente año son Seguridad (36%), Computación en la Nube (31%), Dispositivos Móviles (28%) y Data Management Analytics (27%). Lo anterior puede verse reflejado en la Figura 1. [2]

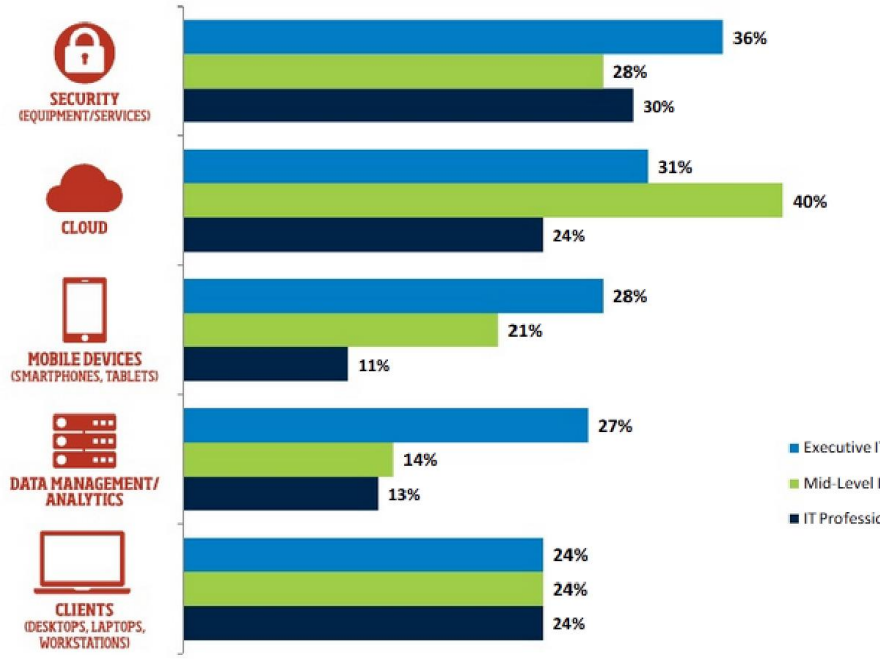


Figura 1. Principales áreas de inversión IT

Según The National Institute of Standards and Technology (NIST), el modelo de Computación en la Nube se compone de cinco características esenciales: *auto-servicio bajo demanda* mediante el cual los clientes se auto abastecen de manera unilateral de los recursos de cómputo, almacenamiento y red que necesitan; *acceso a los recursos a través*

de una red de banda ancha mediante mecanismos estandarizados que promueven el uso de plataformas heterogéneas que oscilan desde dispositivos ligeros como móviles y tablets hasta otros más complejos como ordenadores portátiles y estaciones de trabajo; *agrupación de recursos en lo que se conoce por su término en inglés de "resource pooling"* para dar servicio de manera simultánea a múltiples clientes pertenecientes a distintas organizaciones a los cuales se les asignan y reajustan diferentes recursos físicos y virtuales de manera dinámica y de acuerdo a la demanda y sin que ellos tengan conocimiento o control, por lo general, de la ubicación exacta de los recursos proporcionados, aunque en principio pudieran especificar dicha ubicación utilizando un nivel de abstracción superior tal como país, estado o centro de datos; *aprovisionamiento y liberación elásticos de las capacidades y recursos asignados* para escalar rápidamente hacia afuera y hacia el interior en consonancia con la demanda; y finalmente *monitorización, medición, control y reporte del uso de los recursos* con fines de optimización y para garantizarles al cliente y al proveedor del servicio una total transparencia. Generalmente la medición y facturación de los servicios de Nube siguen el modelo de pago por uso en el que el cliente solo paga por los recursos y servicios consumidos. [3]

Son múltiples las ventajas que se obtienen a partir del empleo de este modelo de computación distribuida. Entre ellas se pueden mencionar el ahorro en costes, en personal, en hardware y en licencias de software, la disminución del tiempo requerido para desarrollar y desplegar un producto o servicio conocido por el término en inglés "*time to market*", el incremento de la disponibilidad y escalabilidad dada la capacidad de adaptarse a las fluctuaciones de la demanda, la posibilidad de crear nuevas fuentes de ingreso y la independencia de la ubicación geográfica para el acceso a los recursos.

Es por ello que ha habido una tendencia en los últimos años por partes de las empresas a migrar sus cargas de trabajo hacia la Nube y se espera que para 2018, más del 60% de las mismas tendrán como mínimo la mitad de sus infraestructuras montadas en plataformas basadas en Nube. [4]

La computación en la Nube presenta tres modelos de servicios. Ellos son Software como Servicio (SaaS), Plataforma como Servicio (PaaS) e Infraestructura como Servicio (IaaS). Cada uno ofrece distintos recursos a los clientes y están ordenados en orden descendente de abstracción y en orden ascendente del control que el cliente tiene sobre los recursos y servicios asignados.

El proveedor de servicios de Nube posee una infraestructura compuesta por dos capas: una capa física y una capa de abstracción. La capa física incluye los recursos de hardware necesarios para brindar los servicios de Nube a los clientes y abarca los recursos de computación, almacenamiento y red. La capa de abstracción hace

referencia al software que es desplegado encima de la capa física y que es empleado para implementar y gestionar la Nube.

En el modelo de Software como Servicio (SaaS), la capacidad brindada a los usuarios consiste en la posibilidad de acceder, mediante sus dispositivos de cliente y a través de una interfaz ligera como lo es un navegador web, a aplicaciones que se ejecutan en la infraestructura de Nube del proveedor y que por lo tanto no deberán ser instaladas para poder utilizarlas. El cliente no gestiona o controla la infraestructura de Nube subyacente que incluye a la red, los servidores, los sistemas operativos, los dispositivos de almacenamiento e incluso las capacidades individuales de las aplicaciones, con excepción de limitados parámetros de configuración específicos [3]. De acuerdo a las predicciones realizadas por la compañía estadounidense de asesoría, análisis y estudio de mercado International Data Corporation (IDC), para 2018, un 27.8% del mercado de las aplicaciones empresariales a nivel mundial estará basado en el modelo SaaS, generando $50.8 billones en ingresos. Igualmente Cisco predice que para este año, el 59% del total de la carga de trabajo en la Nube se deberá a aplicaciones SaaS, en comparación con un 41% detectado en 2013. [1]

La capacidad de desplegar en la infraestructura de Nube aplicaciones adquiridas o creadas utilizando lenguajes de programación, librerías, servicios y herramientas soportadas por el proveedor, consiste la principal característica del modelo de Plataforma como Servicio (PaaS). En el modelo PaaS, el cliente sigue sin tener control sobre la infraestructura de Nube subyacente, sin embargo controla las aplicaciones desplegadas por él y tiene permiso de modificar determinados parámetros de configuración del entorno de Nube que aloja a estas aplicaciones. [3]

El tercer modelo de servicio es Infraestructura como Servicio (IaaS), el cual ofrece recursos de computación, procesamiento, almacenamiento y red sobre los cuales el cliente puede ser capaz de desplegar y ejecutar software arbitrario que abarca desde distintas versiones de sistemas operativos hasta aplicaciones de cliente. Aun cuando se carece de control sobre la infraestructura de Nube subyacente del proveedor, este modelo le ofrece al cliente mayores libertades en cuanto a la selección de su propio sistema operativo, soluciones de almacenamiento, aplicaciones y un control limitado sobre la selección de componentes de red tales como cortafuegos personales [3]. Se espera que para el año 2016, más de un 80% de las empresas a escala global utilicen IaaS. [1]

Los cuatro modelos de despliegue de Nube son Nube Privada, Nube Comunitaria, Nube Pública y Nube Híbrida. En enero de 2015, RightScale llevó a cabo su cuarta encuesta anual para conocer el estado del arte y las tendencias en la Computación en la Nube. En esta encuesta participaron 930 profesionales del sector IT procedentes de

empresas de distinta escala y de diferentes sectores de la industria. Los resultados arrojaron que el 82% de las empresas implementan una estrategia de Nube Híbrida, en comparación con un 74% en 2014, por lo que se nota una mayor adopción y popularidad de este modelo de despliegue. Por otro lado, la Nube Pública es líder en cuanto a tasa de adopción con un 88% de empresas suscritas a algún plan de algún proveedor de Nube Pública, mientras que el 63% de dichas empresas expresan disponer de una Nube Privada, siendo la Nube Privada líder en cuanto a carga de trabajo, es decir, 13% de estas empresas ejecutan más de 1000 máquinas virtuales en la Nube Pública frente a un 22% que ejecutan más de 1000 máquinas virtuales en la Nube Privada.[5]

En el caso de la Nube Privada, los recursos proporcionados son para uso exclusivo de una organización que puede abarcar múltiples usuarios, mientras que la Nube Comunitaria ofrece recursos y servicios a un conjunto de organizaciones que comparte intereses y responsabilidades. La Nube Privada puede ser operada y gestionada por la organización propietaria, mientras que en el caso de la Nube Comunitaria, una o varias de las organizaciones implicadas pueden encargarse de su gestión y operación. Igualmente ambos modelos de despliegue pueden ser gestionados y operados por terceros. Los recursos de la capa física de la infraestructura de Nube pueden encontrarse dentro de las instalaciones de la organización propietaria en el caso de la Nube Privada o dentro de las instalaciones de una o varias de las organizaciones propietarias en el caso de la Nube Comunitaria. En este caso se dice que los recursos físicos están "on-premise". De manera similar los recursos físicos pueden encontrarse "off-premise", es decir, externalizados y ubicados en las instalaciones de terceros.

La Nube Privada requiere de una inversión inicial considerable para la adquisición de los recursos de computación, almacenamiento y red que conforman la infraestructura física que da soporte a los servicios de nube. Es por ello que no constituye la solución más atractiva y recomendable para las pequeñas y medianas empresas, especialmente las startups, que por lo general cuentan con un reducido capital inicial y necesitan enfocar sus esfuerzos en el desarrollo de sus metas de negocio y no abrumarse con la carga de trabajo que conlleva gestionar una Nube Privada. Sin embargo, para aquellas grandes empresas que valoran sumamente la privacidad y seguridad de su información, la Nube Privada constituye el enfoque a seguir.

La Nube Pública ofrece servicios y recursos al público en general y los recursos de hardware sobre los que se sustenta yacen en las instalaciones del proveedor de servicios. Según la encuesta de RightScale, Amazon Web Services (AWS) continúa siendo el principal proveedor de Nube Pública con un 57% de suscripción por parte de

las empresas frente a un 12% de Microsoft Azure que ha ido creciendo en suscriptores en este último año colocándose en la segunda posición como lo indica la Figura 2.[5]

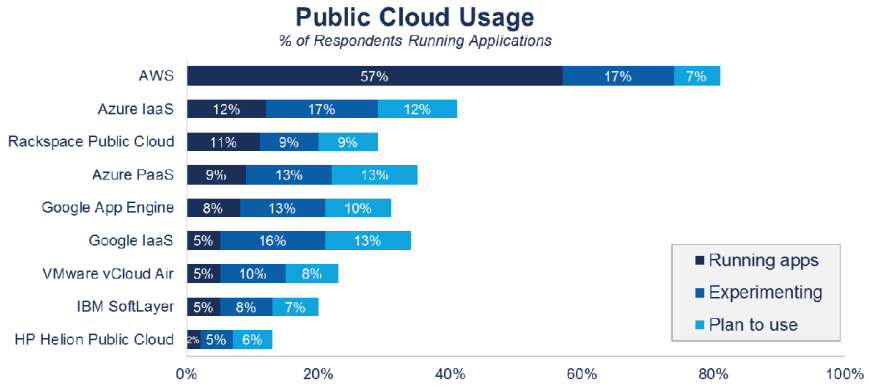


Figura 2. Tendencias en el uso de la Nube Pública

La Nube Híbrida se presenta como una composición de dos o más infraestructuras de nube distintas (privada, comunitaria o pública) que permanecen como entidades independientes pero que están conectadas mediante tecnologías estandarizadas o privadas que habilitan la portabilidad de las aplicaciones y estimulan el rendimiento a partir de facilitar el balanceo de carga entre nubes.

Mientras que la popularidad y adopción de la Nube Híbrida mantiene una pendiente creciente en los últimos años de acuerdo a los resultados de la encuesta de RightScale, pequeños han sido los cambios en la tasa de adopción de la Nube Privada entre 2014 y 2015. Sin embargo, según un reporte de Ovum, se prevé un mayor crecimiento en las inversiones en la Nube Privada en vistas al 2018. [6]

VMWare continúa siendo el líder en la Nube Privada. Un 33% de los participantes en la encuesta declaró utilizar VMware vSphere/vCenter frente a un 13% que utiliza VMware vCloud Suite. Por otro lado, el 13% de los participantes utiliza OpenStack. Esta solución de software libre y de código abierto, distribuida bajo los términos de la licencia Apache y gestionada por la Fundación OpenStack a la cual contribuyen empresas como IBM, HP, Intel, Canonical, Rackspace, Red Hat Inc., y SUSE, por mencionar algunas, continúa generando altos niveles de interés con un 30% de los participantes en la encuesta de RightScale evaluándola y planificando usarla.[5]

OpenStack es una solución de computación en la nube que permite la construcción de nubes públicas, híbridas y privadas que proporcionen Infraestructura como Servicio (IaaS). Consiste en un conjunto de proyectos que pueden instalarse y configurarse para que funcionen de manera independiente o para que se interrelacionen. Estos proyectos incluyen Servicio de Identidad (Keystone), Servicio de Imágenes (Glance), Servicio de Computación (Nova), Servicio de Red (Neutron), Servicio de Almacenamiento en

Bloques (Cinder), Almacenamiento de Objetos (Swift), Procesamiento de Datos (Sahara), Telemetría (Ceilometer), Orquestación (Heat) y Base de Datos (Trove). Debido a que la comunidad de desarrollo es sumamente activa y cuenta con un gran respaldo de financiamiento, en los últimos años han surgido nuevos proyectos que ofrecen la posibilidad de integrar a OpenStack con las últimas tecnologías de virtualización, red y almacenamiento que han tenido éxito en el sector IT, todo ello sobre la base de integrarse y reutilizar las funcionalidades brindadas por los proyectos principales de OpenStack mencionados anteriormente.

Existen distintos modos de despliegue de OpenStack que atienden a los fines con los cuales se vaya a utilizar la nube que se pretende construir con esta plataforma y que van desde despliegues sobre un único servidor físico o incluso máquina virtual en entornos de desarrollo, hasta despliegues de producción que como mínimo requieren dos servidores físicos y que pueden complejizarse y crecer hacia arquitecturas que ofrecen alta disponibilidad. Al mismo tiempo cada uno de los distintos componentes o proyectos que integran a OpenStack, ofrecen distintas alternativas de selección y configuración de las tecnologías subyacentes con las que se integran.

Entre los motivos que llevan a las grandes empresas actualmente a desplegar sus infraestructuras de Nube Privada se encuentra la necesidad de contar con una plataforma segura para el procesamiento de sus datos, siendo estos datos altamente sensibles para estas empresas y por lo tanto estas prefieren invertir en la Nube Privada y de esta forma evitar correr el riego de que la seguridad y privacidad de su información se vea comprometida en el entorno de la Nube Pública.

En la presentación realizada por Michael Coté, director técnico de marketing en Pivotal, y conocida como "Cloud State of the Union 2015", este hace referencia a los resultados de un estudio publicado por la empresa 451 Research bajo el nombre "Voice of the Enterprise: Cloud Computing" y en el que como se observa en la Figura 3, la mayor parte de los participantes manifestaron que consideraban la Nube Privada como el modelo de despliegue primario durante los próximos dos años para ejecutar distintas cargas de trabajo que incluyen: aplicaciones colaborativas, Infraestructura de Escritorio Virtual (VDI) o escritorio alojado, desarrollo de aplicaciones, comercio electrónico y sitios web, aplicaciones empresariales, procesamiento de Big Data y aplicaciones específicas de industria, entre otras.[7]

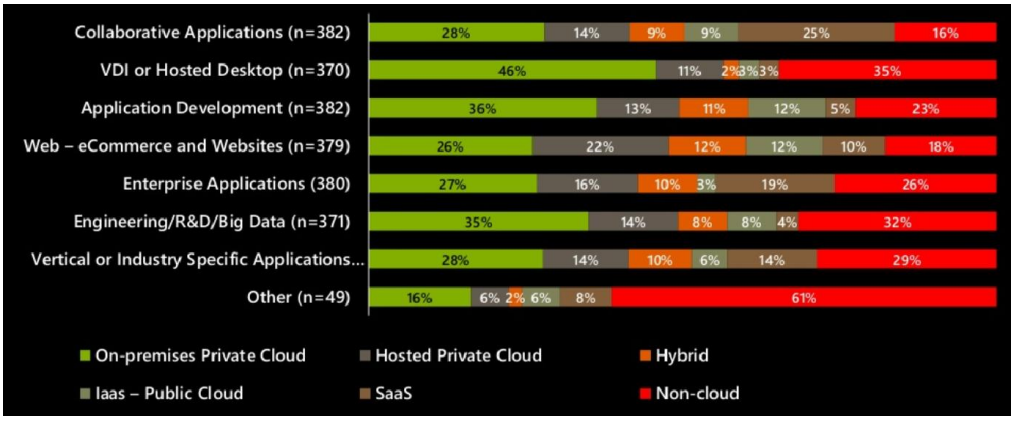


Figura 3. Tendencias en la distribución de las cargas de trabajo en los Modelos de Despliegue de Nube

Big Data es un término utilizado para describir grandes colecciones de datos, también conocidos como "datasets" que pueden estar estructurados, semi-estructurados o no estructurados y que crecen en tamaño y a una velocidad que resulta complejo gestionarlos empleando soluciones tradicionales de gestores de bases de datos y herramientas estadísticas.

Los principales retos que presenta el procesamiento de Big Data se describen mejor mediante las "tres Vs": Volumen, Variedad y Velocidad. Volumen implica que la cantidad de datos a almacenar es inmensa oscilando desde decenas de terabytes o petabytes hasta incluso exabytes en el caso de algunas compañías como Facebook, Twitter, Amazon, Google y Yahoo. Variedad se refiere a que los datos pueden encontrarse en diferentes formatos, es decir, estructurados, semi-estructurados o no estructurados. Los datos no estructurados suelen almacenarse en textos, imágenes, videos, audios, documentos y metadatos los cuales no tienen prácticamente estructura. Igualmente estos datos pueden clasificarse como semi-estructurados, como es el caso de los ficheros de bitácoras, o estructurados y estrictamente ordenados en filas y columnas, donde cada columna tiene asignado un tipo de dato. Finalmente Velocidad hace alusión al hecho de que los datos generados por una empresa, por sus clientes o que de cierta manera tienen relación con la misma, solamente tienen valor durante una determinada ventana de tiempo, después de la cual son inútiles, por lo que en la medida que crezca el volumen de los datos relacionados con la empresa, mayor será la velocidad requerida para procesar estos datos con el fin de sacar algún provecho de ellos. [8]

Apache Hadoop es un framework libre de código abierto escrito en Java para almacenar y procesar grandes volúmenes de datos de manera distribuida en clústeres de servidores con hardware básico. Hadoop ha sido optimizado para manipular

cantidades masivas de datos estructurados, semi-estructurados y no estructurados, los cuales pueden ser procesados de manera más eficiente y rápida a través del paralelismo. Este procesamiento masivo de datos en paralelo se ejecuta con un alto rendimiento, y la fiabilidad de la información y la recuperación ante fallos se garantizan a partir de emplear técnicas de replicación de datos a lo largo de múltiples nodos, de manera que si un nodo queda fuera de servicio, los datos pueden procesarse a partir de una réplica almacenada en otro nodo operativo. [8]

OpenStack, recientemente en su versión Havana de octubre de 2013, lanzó un nuevo proyecto llamado Sahara que tiene como objetivo dotar a los clientes de medios sencillos para desplegar un clúster Hadoop en OpenStack para procesamiento de Big Data, especificando varios parámetros como la versión de Hadoop, la topología del clúster, detalles del hardware de los nodos que integran el clúster, entre otros parámetros de configuración. El proyecto Sahara continúa desarrollándose y sus últimas actualizaciones están incluidas en la última versión de OpenStack llamada Kilo, lanzada en mayo de 2015 y que constituyó la versión seleccionada para la realización de este trabajo.

Motivación

Con el fin de integrarse al flujo de investigación y desarrollo en el entorno de la Computación en la Nube, en la ETSIT-UPM se ha venido trabajando en el desarrollo de extensiones para el componente de Gestión de Identidad de OpenStack (Keystone), en la modificación del componente de Gestión Web Horizon basándose en el empleo de frameworks que siguen el patrón Modelo-Vista-Controlador (MVC) del lado del cliente, así como en otras líneas vinculadas con la iniciativa europea FI-WARE y se ha planteado la necesidad de desplegar una maqueta de nube privada basada en OpenStack sobre la cual se puedan probar estas innovaciones y al mismo tiempo adoptar las últimas tecnologías en el despliegue de OpenStack que incluyen su integración con Docker para la gestión de contenedores virtuales que utilizan virtualización ligera a nivel de sistema operativo y el despliegue de clústeres para procesamiento de Big Data utilizando las tecnologías de Apache Hadoop.

Todo ello constituye un escenario de prueba que servirá de antesala a la futura migración del Centro de Datos de la ETSIT-UPM hacia OpenStack. Actualmente, el Centro de Datos de la ETSIT-UPM utiliza una versión antigua de VMware vSphere/vCenter y no tiene posibilidades de migrar hacia la versión actual de esta solución de VMware dado los altos costes de la licencia. Una migración hacia OpenStack garantizaría poder brindar los servicios desplegados en estos momentos, eliminar la dependencia de una solución comercial con altos costes de licencia y la posibilidad de implementar nuevos servicios gracias a la constante evolución de esta

herramienta de Nube que cuenta con una comunidad organizada, activa y en constante desarrollo que promueve la creación de nuevos proyectos con el fin de satisfacer la demanda de nuevos servicios.

Objetivos

Los objetivos de este Trabajo de Fin de Máster se desglosan a continuación:

- ✓ Diseñar e implementar una propuesta de arquitectura de Computación en la Nube basada en OpenStack Kilo para entornos de Big Data.
- ✓ Automatizar la instalación y despliegue de la arquitectura de Nube propuesta.

Para el cumplimiento de estos objetivos se fijaron las siguientes tareas de investigación:

- ✓ Estudiar los distintos proyectos que componen a OpenStack, profundizando por cada uno de ellos en su arquitectura interna, las funcionalidades que ofrece, las tecnologías sobre las que se sustenta, las distintas opciones de configuración que soporta, y como cada proyecto interacciona con el resto de los proyectos de OpenStack para ofrecer y solicitar servicios.

- ✓ Realizar un estudio del estado del arte de las distintas arquitecturas de Nube basada en OpenStack Kilo para entornos de producción especificando las distintas configuraciones de infraestructura de controladores, nodos de red, computación y almacenamiento.

- ✓ Diseñar e implementar una propuesta de arquitectura de Nube para entornos de procesamiento de Big Data en el centro de datos de la ETSIT-UPM que constituya una maqueta de pruebas.

- ✓ Programar los scripts Python encargados de automatizar la instalación de la arquitectura de Nube propuesta y ejecutar estos scripts en máquinas virtuales que representen cada nodo de la arquitectura para probar que en efecto se puede reproducir de manera automática el despliegue de dicha arquitectura.

Estructura del documento

El presente trabajo contiene una descripción de los distintos proyectos que componen a OpenStack, incluye un estudio del estado del arte de las distintas tecnologías de virtualización de contenedores y procesamiento de Big Data especificando cómo estas tecnologías se integran dentro del entorno de proyectos de esta solución de Nube, presenta y describe el diseño e implementación de la arquitectura de Nube propuesta para entornos de Big Data y finalmente procede a la realización de pruebas para verificar el correcto funcionamiento de la arquitectura. Este trabajo finaliza con unas conclusiones y recomendaciones con vistas a trabajos futuros.

Para cubrir estos temas se estructuró el trabajo en tres capítulos:

- ✓ **Capítulo 1:** *Arquitectura lógica y principales proyectos de OpenStack*. Presentación de OpenStack como herramienta IaaS para el despliegue de infraestructuras de Nube privada, híbrida y pública. Descripción de los proyectos básicos que componen la arquitectura lógica de OpenStack, especificando las distintas opciones de configuración y componentes de software con los que se integra cada proyecto y cómo estos proyectos colaboran entre sí para garantizar el funcionamiento de la Nube como un todo.
- ✓ **Capítulo 2:** *Tecnologías de virtualización de contenedores y procesamiento de Big Data en OpenStack*. Descripción de las principales tecnologías de virtualización de contenedores así como de las distintas herramientas de orquestación disponibles y en desarrollo actualmente para integrar el uso de contenedores con OpenStack. Presentación y análisis de las distintas tecnologías para incluir procesamiento de datos (Big Data) como un servicio brindado por OpenStack.
- ✓ **Capítulo 3:** *Diseño e Implementación de una arquitectura de Nube basada en OpenStack para entornos de Big Data*. Descripción de la arquitectura propuesta detallando las decisiones de configuración y componentes de software seleccionados para cada proyecto. Presentación de los script Python programados para reproducir de manera automática dicha arquitectura y explicación de cómo deben ejecutarse. Presentación de las distintas pruebas diseñadas con el fin de verificar el correcto funcionamiento de la arquitectura de Nube propuesta y análisis de los resultados obtenidos.

Capítulo 1. Arquitectura lógica y principales proyectos de OpenStack

1.1 Introducción

OpenStack constituye una plataforma de Computación en la Nube basada en software libre de código abierto integrada por un conjunto de servicios interrelacionados que ofrecen una Interfaz de Programación de Aplicaciones (API) que facilita su integración con la plataforma y que de manera conjunta gestionan y controlan los recursos de computación, almacenamiento y red que forman parte de la infraestructura de Nube del centro de datos. La Tabla 1 y la Tabla 2 describen brevemente los principales proyectos que conforman la arquitectura lógica de OpenStack.[9]

Tabla 1. Proyectos principales de OpenStack

Servicio	Proyecto	Descripción
Identidad	Keystone	Le proporciona un servicio de autenticación y autorización a otros proyectos de OpenStack. Provee un catálogo con los puntos de acceso o "endpoints" de cada uno de estos proyectos.
Computación	Nova	Gestiona el ciclo de vida de las máquinas virtuales incluyendo su creación, la selección del nodo de computación donde se ejecutarán y el desmantelamiento de las mismas bajo demanda.
Red	Neutron	Proporciona Conectividad de Red como Servicio a otros proyectos de OpenStack como por ejemplo Nova, el cual utiliza la API de Neutron para solicitar la conexión de las máquinas virtuales a un segmento de red determinado. Permite la creación de redes, subredes, enrutadores, cortafuegos, balanceadores de carga y redes privadas virtuales. Cuenta con una arquitectura modular que se integra con múltiples tecnologías de proveedores de red externos.
Imagen	Glance	Almacena y gestiona imágenes de los discos de las máquinas virtuales. Nova hace uso de estas imágenes durante el proceso de creación de las máquinas virtuales.
Interfaz de Gestión Web	Horizon	Portal de auto-servicio web que permite a administradores y clientes interactuar con los servicios subyacentes de OpenStack.

Tabla 2. Proyectos opcionales de OpenStack

Servicio	Proyecto	Descripción
Almacenamiento de Objetos	Swift	Almacena y gestiona datos arbitrarios no estructurados en forma de objetos los cuales son accedidos mediante una API RESTful basada en HTTP. Es altamente tolerante a fallos a partir de la implementación de mecanismos de replicación de datos y arquitectura escalable. Su implementación no consiste en un servidor de ficheros compartidos con directorios montados, sino que escribe objetos y ficheros en múltiples dispositivos, garantizando que los datos estén replicados a lo largo del clúster de servidores.
Almacenamiento de Bloques	Cinder	Proporciona almacenamiento permanente a las máquinas virtuales en ejecución. Presenta una arquitectura modular con una extensa variedad de drivers que permiten la creación de volúmenes de almacenamiento en bloques los cuales son acoplados a las máquinas virtuales.
Telemetría	Ceilometer	Supervisa y mide los distintos servicios y el consume de los recursos en OpenStack con fines de facturación, benchmarking, escalabilidad y reporte de estadísticas.
Orquestación	Heat	Orquesta múltiples aplicaciones de Nube compuestas a partir del empleo del formato de plantilla nativa "HOT" o del formato de plantilla AWS CloudFormation, a través de ambas: una API REST nativa de OpenStack y una API Query compatible con CloudFormation
Base de Datos	Trove	Proporciona Bases de Datos como Servicio de manera escalable y fiable. Las bases de datos ofertadas son tanto relacionales como no relacionales.
Procesamiento de Datos	Sahara	Proporciona las funcionalidades necesarias para desplegar y escalar un clúster Hadoop sobre OpenStack a partir de especificar parámetros de configuración del clúster tales como la versión de Apache Hadoop, la topología del clúster y los detalles de hardware de los nodos que lo conforman.

La Figura 4 muestra la manera en la que se interrelacionan estos distintos proyectos.[9]

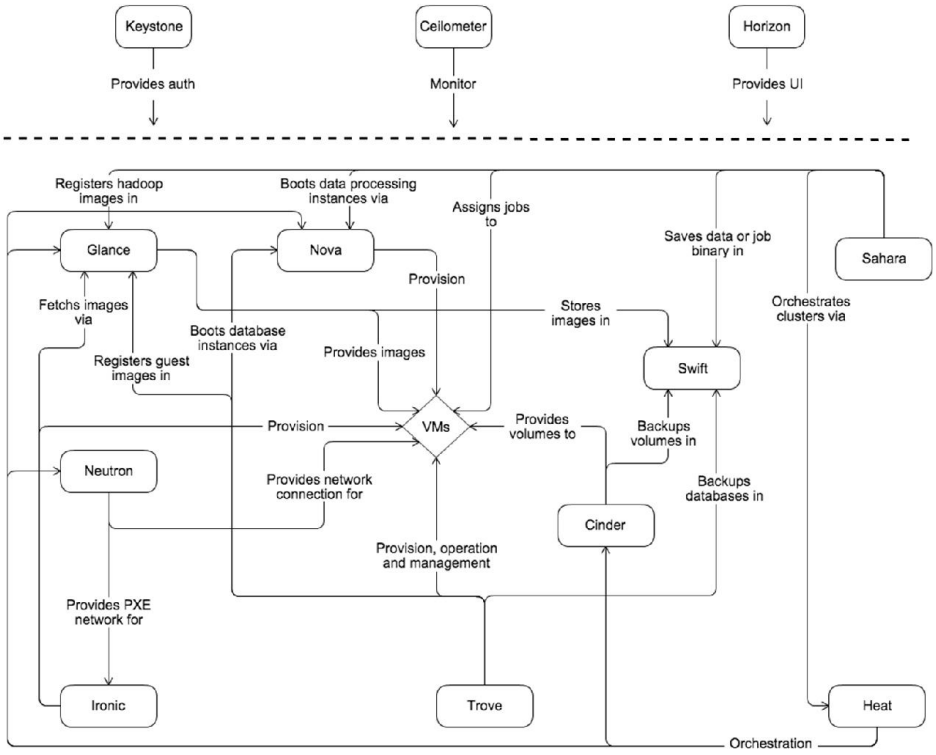


Figura 4. Arquitectura Conceptual de OpenStack: Interrelación de Proyectos

Como puede percibirse en la Figura 4, los distintos servicios independientes que conforman a OpenStack, se autentican a través de un servicio común llamado Keystone. Los servicios se comunican entre ellos mediante APIs públicas, excepto en el caso de comandos de administración que requieren de permisos privilegiados y para los cuales no se puede emplear la API del servicio.

Los usuarios pueden acceder a estos servicios mediante Horizon, que proporciona una interfaz de gestión web, mediante clientes de línea de comandos o utilizando la API pública del servicio ya sea mediante plugins del navegador o utilizando el programa "curl". En todos los casos todos estos métodos de acceso lanzan llamadas API REST a los distintos servicios de OpenStack.

En la Figura 4 se destaca cómo Nova se encarga de aprovisionar las máquinas virtuales, cómo Glance proporciona la imagen a partir de la cual se levanta el sistema operativo con las aplicaciones que se ejecutarán en la instancia, cómo Neutron garantiza la conectividad de dichas instancias a un segmento de red y Cinder acopla un volumen para el almacenamiento permanente de los datos. Por otro lado, Swift se encarga de almacenar en forma de objetos los volúmenes de almacenamiento en bloques creados con Cinder así como las imágenes de Glance. Ceilometer, por su parte, se encarga de medir y contabilizar el uso de los recursos de Nube.

Internamente y como puede observarse en la Figura 5 [9], los servicios de OpenStack están compuestos por varios procesos. Todos los servicios tienen como mínimo un proceso API, que escucha las solicitudes API, realiza un procesamiento inicial y luego las pasa al resto de los procesos internos del servicio. Como puede verse y a diferencia del servicio de Identidad, el trabajo pesado es llevado a cabo por el resto de los distintos procesos que componen al servicio.

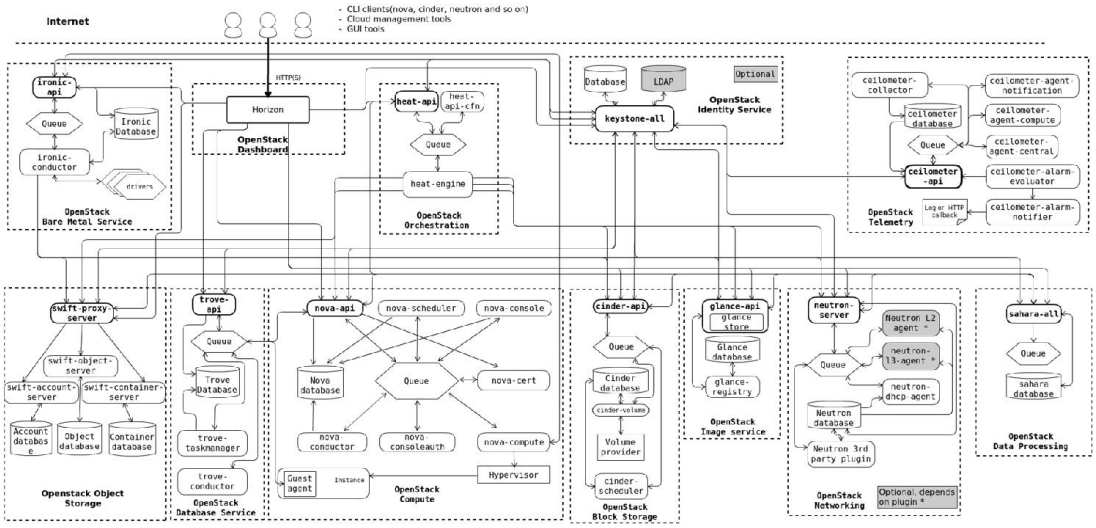


Figura 5. Arquitectura Lógica de OpenStack: Componentes de los Servicios

OpenStack se presenta como una solución simple de implementar, masivamente escalable y rica en funcionalidades. Actualmente es empleado por numerosos proveedores de servicios de Nube Pública, Híbrida y Privada y su tasa de aceptación ha presentado una curva ascendente en los últimos años. En los próximos apartados se describen los distintos proyectos de OpenStack particularizando en sus componentes y sus flujos de trabajo. OpenStack lanza dos versiones por año, la versión actual de esta plataforma de nube es Kilo y es la utilizada para el desarrollo de este trabajo.

1.2 Servicio de Identidad

El Servicio de Identidad de OpenStack llamado Keystone constituye el sistema de gestión de identidad por defecto que se encarga de realizar las funciones relacionadas con la autenticación y autorización de usuarios. [10]

En Keystone, los usuarios no son más que la representación digital de una persona, sistema o servicio que utiliza la nube de servicios de OpenStack. Los usuarios se autentican ante Keystone utilizando sus credenciales que pueden ser: nombre de usuario y contraseña, nombre de usuario y llave API o un token de autenticación que el propio servicio de Identidad le ha proporcionado al usuario. [10, 11]

Otro término que es necesario dominar para trabajar con el Servicio de Identidad es "tenant". Un "tenant" es un contenedor utilizado para agrupar y aislar recursos. Los tenants igualmente agrupan y aíslan objetos de identidad. En dependencia del operador del servicio, un tenant pude mapearse contra un cliente, una cuenta, una organización o un proyecto. Keystone implementa un control de acceso basado en roles. Un usuario puede ser asignado a un tenant y se le asignará un rol dentro de ese tenant. Cada usuario puede ser asignado a varios tenants y tener diferentes roles en cada tenant, igualmente un usuario puede tener varios roles en un mismo tenant. [10, 11]

Un rol no es más que un conjunto de permisos y privilegios para realizar un conjunto de operaciones específicas. Por ejemplo, un usuario puede tener asignando un rol en un tenant que le permite exclusivamente crear volúmenes de almacenamiento en bloques en OpenStack Cinder o almacenar datos en forma de objetos en OpenStack Swift, mientras que en otro tenant puede tener asignado un rol que le permita crear y destruir máquinas virtuales en OpenStack Compute, o incluso tener un rol de administración. Es por ello que cuando un usuario se registra en Keystone, tiene que especificar, además de sus credenciales, el tenant con el cual se quiere registrar. [10, 11]

Para que un usuario pueda acceder a un servicio de OpenStack, por ejemplo, listar las máquinas virtuales en ejecución, iniciar una nueva máquina virtual, crear una red, un enrutador o un volumen de almacenamiento en bloque para conectarlo a una máquina virtual, este usuario deberá registrarse ante Keystone. Si el usuario se autentica exitosamente, recibirá un token temporal que le valdrá para acceder al resto de los servicios de OpenStack durante un determinado período de tiempo después del cual este usuario deberá volver a autenticarse.

Cuando Keystone le asigna un token de autenticación a un usuario, este token trae incluido una lista con los roles de ese usuario en el tenant con el que este se registró en el Servicio de Identidad. Cuando un usuario solicita un servicio mediante una llamada API REST, el usuario le envía este token al servicio de OpenStack. Entonces, este servicio contacta con Keystone para verificar la validez del token proporcionado por el usuario y en caso de ser válido procede a determinar cómo interpretar el conjunto de roles de este usuario dentro del tenant especificado, e igualmente determina a qué operaciones o recursos cada rol concede acceso.

Entre los componentes de Keystone se encuentra una base de datos (en la mayoría de los despliegues se emplea MySQL o MariaDB) que mantiene un registro de usuarios, tenants o proyectos, roles, servicios y API endpoints o puntos de accesos a estos servicios. Las principales funciones de Keystone engloban el seguimiento de los usuarios y sus permisos así como el aprovisionamiento de un catálogo con los servicios

disponibles y sus puntos de acceso o API endpoints. En Keystone, los servicios se agrupan por regiones. Un punto de acceso a un servicio o API Endpoint incluye entre otros parámetros, la dirección IP y puerto en el que escucha el servicio y la región del mismo. Keystone provee un punto único de integración para las políticas de OpenStack. [10]

La Figura 6 ilustra el flujo e intercambio de mensajes propios del proceso de autenticación y autorización en Keystone.[10]

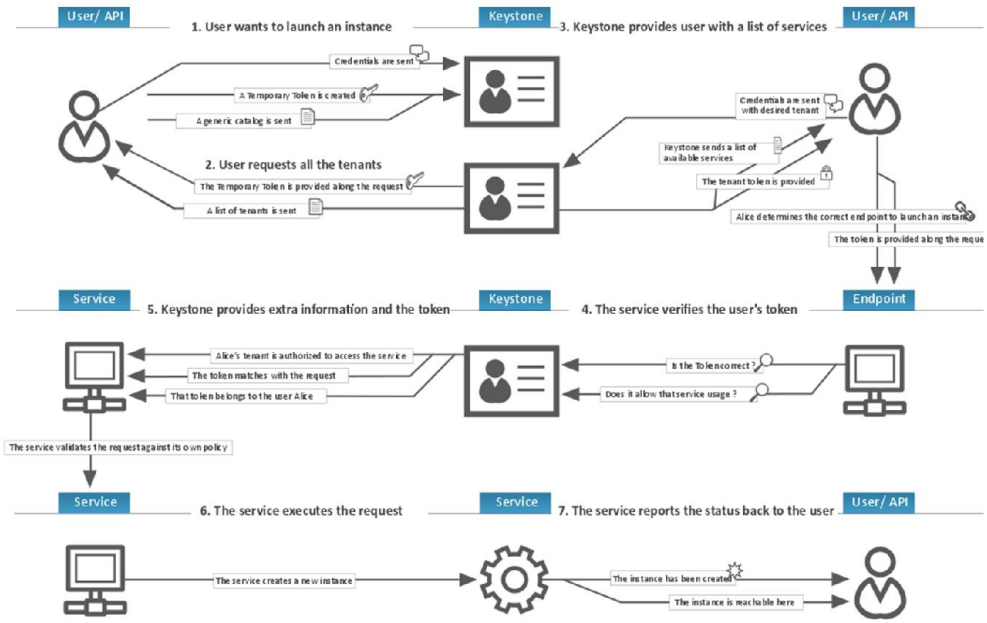


Figura 6. Proceso de Autenticación y Autorización en Keystone

Como se observa en la Figura 6, este proceso consta de varias fases:

i. El usuario envía sus credenciales a Keystone, este crea un token temporal y envía un catálogo genérico.
ii. El usuario solicita una lista de tenants o proyectos a los cuales puede acceder.
iii. Keystone responde con una lista de tenants o proyectos a los que este usuario tiene acceso y le envía el token temporal.
iv. El usuario selecciona un tenant y envía sus credenciales dentro de este tenant.
v. Keystone responde con la lista de servicios disponibles al usuario dentro de este tenant y le envía el token para este tenant.
vi. El usuario determina el endpoint correcto para acceder al servicio y le envía el token del tenant que recibió de Keystone.
vii. El servicio contacta a Keystone para validar el token de tenant recibido y si el usuario está autorizado para acceder a este servicio.

viii. Keystone notifica al servicio que el usuario está autorizado para solicitar ese servicio y que el token es válido.

ix. El servicio pasa a aplicar sus políticas internas y posteriormente ejecuta la solicitud del usuario.

En la Figura 7 se ilustra un ejemplo más concreto que consiste en la solicitud por parte del usuario de crear una máquina virtual y como en este proceso intervienen varios servicios: Keystone que se encarga de autenticar al usuario, Nova que se encarga de crear la máquina virtual como tal, Glance que proporciona la imagen desde la cual se arranca el sistema operativo de la instancia y Neutron que conecta la interfaz de red virtual de la máquina virtual a alguna de las redes virtuales del tenant al que pertenece el usuario que realiza la solicitud.[11]

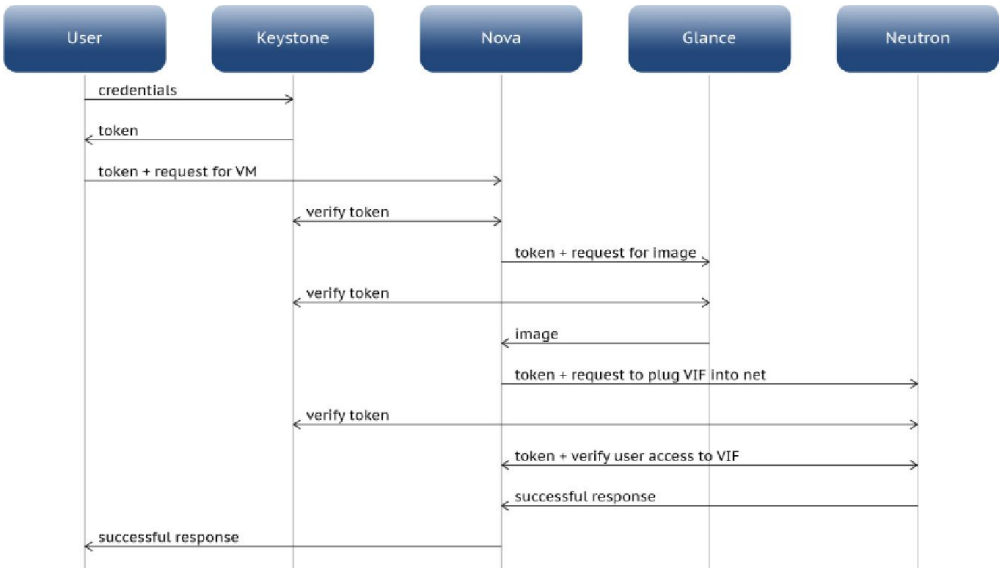


Figura 7. Proceso de creación de una máquina virtual

En el Anexo 1 se incluyen los comandos empleados para crear usuarios, tenants o proyectos, roles, servicios y API endpoints, así como los requeridos para asignarle un rol a un usuario dentro de un determinado tenant.

1.3 Servicio de Imágenes

El Servicio de Imágenes de OpenStack llamado Glance permite a los usuarios registrar, almacenar y gestionar imágenes de máquinas virtuales que posteriormente el Servicio de Computación (Nova) solicita para poder crear dichas instancias. Glance ofrece una API REST que permite encuestar al servicio para obtener metadatos de las imágenes almacenadas, e incluso obtener la propia imagen. [12]

Una imagen de una máquina virtual consiste en un fichero que contiene la imagen de un disco duro con un sistema operativo pre-instalado y que sirve como plantilla para generar el contenido del disco duro de la instancia que se desea crear con Nova. De

esta manera, cuando se crea una máquina virtual y esta se arranca, la máquina virtual levanta con el sistema operativo que detecta en su disco duro y que no es más que el especificado por la imagen de Glance. [13]

Una imagen puede ser empleada para levantar varias máquinas virtuales. Cada máquina virtual se ejecutará a partir de una copia de la imagen, de manera que cualquier cambio que se produzca en la máquina virtual no afectará a la imagen base. Igualmente se pueden crear snapshots de la máquina virtual en ejecución con el fin de crear una imagen basada en el estado actual del disco de la máquina virtual. [9]

Las imágenes de las máquinas virtuales almacenadas en Glance tienen un parámetro denominado formato de disco (disk-format) que puede configurarse de acuerdo a las siguientes opciones [13]:

- ✓ Raw: es el formato más simple y es soportado de manera nativa por los hipervisores KVM y Xen. Puede considerarse como el equivalente binario de un fichero de dispositivo de bloque.
- ✓ qcow2 (QEMU copy-on-write versión 2): este formato es comúnmente utilizado con el hipervisor KVM. Las imágenes que usan este formato tiene un tamaño más pequeño que las que utilizan el formato Raw. Además este formato permite crear snapshots.
- ✓ ami/aki/ari: formato inicialmente soportado por Amazon EC2 compuesto por tres ficheros:
  - ami (Amazon Machine Image): imagen de máquina virtual en formato raw.
  - aki (Amazon Kernel Image): un fichero kernel que el hipervisor carga inicialmente para arrancar la imagen. Para una máquina Linux, este sería el fichero "vmlinuz"
  - ari (Amazon Ramdisk Image): un fichero ramdisk opcional montado en tiempo de arranque de la máquina virtual. Para una máquina Linux este sería el fichero "initrd".
- ✓ UEC tarball (Ubuntu Enterprise Cloud): un comprimido construido con "gzip" o "tar" que contiene ficheros ami, aki y ari.
- ✓ VMDK (Virtual Machine Disk): formato de imagen utilizado por el hipervisor ESXi de VMware.
- ✓ VDI (Virtual Disk Image): formato de imagen utilizado por Oracle VirtualBox. Ninguno de los hipervisores disponibles en OpenStack soporta este formato por lo que la imagen debe ser convertida a otro formato antes de poder utilizarse para arrancar una máquina virtual.
- ✓ VHD (Virtual Hard Disk) y VHDX: formato utilizado por Microsoft Hyper-V.

- ✓ OVF (Open Virtualization Format): formato de empaquetado definido por el grupo de estandarización Distributed Management Task Force (DMTF) y contiene uno o varios ficheros de imágenes, un fichero XML de metadatos y otros ficheros adicionales.
- ✓ ISO: formato de imagen formateado con el sistema de ficheros de solo lectura ISO 9660 usado para CDs y DVDs. No suelen ser el primer formato de imagen en el que se piensa, sin embargo, es capaz de contener un sistema de ficheros con un sistema operativo instalado que permite arrancar una máquina virtual.
- ✓ Docker: formato de imagen utilizado para crear contenedores Docker.

De manera similar, las imágenes de Glance tienen otro parámetro llamado formato de contenedor (container-format) que indica si la imagen está almacenada en un formato de fichero que también contiene metadatos acerca de la máquina virtual. El Servicio de Imágenes y otros servicios de OpenStack actualmente no soportan este parámetro por lo tanto todas las máquinas virtuales almacenadas en Glance tendrán asignado a este parámetro el valor "bare" que implica que la imagen no contiene un envoltorio de metadatos. Otros valores posibles aunque no soportados son "ovf", "aki", "ari" y "ami". [13]

Para poder instalar Glance se debe tener previamente instalado Keystone. Los primeros pasos de instalación de Glance consisten en registrar el servicio en Keystone y crearle su punto de acceso o API endpoint.

Las imágenes en Glance pueden almacenarse en una variedad de ubicaciones. La opción por defecto es almacenarlas en el sistema de ficheros local del servidor físico donde está instalado el Servicio de Imágenes, particularmente en el directorio `/var/lib/glance/images/`.[10]

El resto de las soluciones, aunque añaden mayor complejidad en cuanto a la configuración, garantizan una mayor escalabilidad y tolerancia a fallos. Estas aparecen listadas a continuación [13]:

- ✓ Servicio de Almacenamiento de Objetos (Swift): la imagen se almacena en forma de objeto y Swift se encarga de proporcionar una alta disponibilidad y tolerancia a fallos a partir de la replicación de la información.
- ✓ Servicio de Almacenamiento de Bloques (Cinder)
- ✓ VMware con su sistema ESX/ESXi o vCenter Server
- ✓ Amazon S3 Simple Storage Service
- ✓ HTTP: Glance puede leer la imagen de una máquina virtual disponible en Internet utilizando HTTP. Este sistema de almacenamiento de imagen es de solo lectura.

- ✓ RADOS block device (RBD): almacena las imágenes de las máquinas virtuales en un clúster de almacenamiento Ceph utilizando la interfaz RBD de Ceph. Ceph es un sistema de almacenamiento distribuido altamente escalable, fiable y con un excelente rendimiento.
- ✓ Sheepdog: sistema distribuido de almacenamiento para QEMU/KVM
- ✓ GridFS: Almacena las imágenes utilizando MongoDB

Como se muestra en la Figura 8, el Servicio de Imágenes de OpenStack incluye los siguientes componentes [11]:

- ✓ Glance API (glance-api): gestiona la comunicación con el backend de almacenamiento de las imágenes y con el resto del entorno de servicios de OpenStack. Le permite a los clientes registrar nuevas imágenes y obtener información acerca de las mismas. Sin embargo su principal función es proporcionar la imagen necesaria al Servicio de Computación (Nova)
- ✓ Glance Registry (glance-registry): se encarga de mantener la información necesaria para gestionar las imágenes en forma de metadatos almacenados en una base de datos. Los metadatos incluyen parámetros como tamaño y tipo de imagen. Constituye un componente de servicio interno no accesible para el resto de los usuarios.
- ✓ Base de Datos: almacena metadatos acerca de las imágenes. El gestor de base de datos más frecuente utilizado es MySQL o MariaDB.
- ✓ Backend de Almacenamiento: repositorio donde se almacenan las imágenes

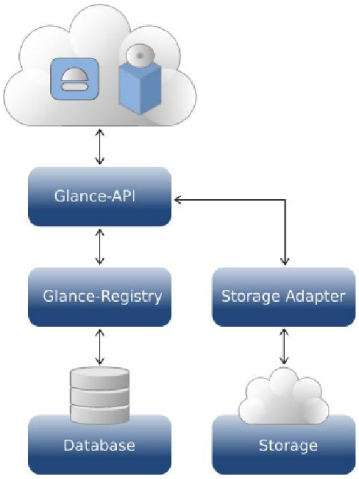


Figura 8. Arquitectura Interna de Glance

Los metadatos de una imagen pueden tener dos finalidades: ayudar a determinar la naturaleza de la imagen e influir en la decisión que el componente planificador del Servicio de Computación toma acerca de en qué nodo de computación se debe ejecutar la máquina virtual que se desea crear. Para que el planificador de Nova, conocido

como nova-scheduler, considere los metadatos de la imagen y en base a ellos seleccione el nodo de computación que cumpla con los requisitos, es decir, que satisfaga los valores de los metadatos, es necesario que el filtro ImageProperties esté habilitado en el fichero de configuración del Servicio de Computación, aunque cabe resaltar que este filtro del planificador está habilitado por defecto. [11, 13]

Entre los parámetros de metadatos asociados a una imagen se encuentran [14]:

- ✓ architecture: especifica la arquitectura del CPU que debe ser soportada por el hipervisor. Los valores posibles son `i686, x86_64, arm,` y `ppc64`.
- ✓ hypervisor_type: especifica el tipo de hipervisor. Los valores posibles son `KVM, Xen, qemu, lxc, UML, VMware, HyperV,` y `PowerVM`.
- ✓ hypervisor_version_requires: versión del hipervisor requerido para la imagen. Actualmente este parámetro solamente está soportado por Xen
- ✓ vm_mode: define la interfaz binaria de aplicación (ABI) del hipervisor requerida para la imagen. Los posibles valores son `hvm` para virtualización completa en el caso de QEMU y KVM, `xen` para paravirtualización con Xen, `uml` para Linux en Modo Usuario paravirtualizado y `exe` para ejecutables en LXC (Linux Containers).

Las imágenes en Glance se crean utilizando el siguiente comando [15]:

```
glance image-create [--id <IMAGE_ID>] [--name <NAME>] [--store <STORE>]
                    [--disk-format <DISK_FORMAT>]
                    [--container-format <CONTAINER_FORMAT>]
                    [--owner <TENANT_ID>] [--size <SIZE>]
                    [--min-disk <DISK_GB>] [--min-ram <DISK_RAM>]
                    [--location <IMAGE_URL>] [--file <FILE>]
                    [--checksum <CHECKSUM>] [--copy-from <IMAGE_URL>]
                    [--is-public {True,False}]
                    [--is-protected {True,False}]
                    [--property <key=value>] [--human-readable]
                    [--progress]
```

Para modificar una imagen luego de que esta ha sido creada se emplea el siguiente comando [15]:

```
glance image-update [--name <NAME>] [--disk-format <DISK_FORMAT>]
                    [--container-format <CONTAINER_FORMAT>]
                    [--owner <TENANT_ID>] [--size <SIZE>]
                    [--min-disk <DISK_GB>] [--min-ram <DISK_RAM>]
                    [--location <IMAGE_URL>] [--file <FILE>]
                    [--checksum <CHECKSUM>] [--copy-from <IMAGE_URL>]
                    [--is-public {True,False}]
                    [--is-protected {True,False}]
                    [--property <key=value>] [--purge-props]
                    [--human-readable] [--progress]
                    <IMAGE>
```

Los metadatos descritos anteriormente se le pasan a los dos comandos anteriores después del argumento –-property en forma de par llave, valor.

1.4 Servicio de Computación

El Servicio de Computación llamado Nova constituye el principal servicio de OpenStack. Es el encargado de crear, gestionar y eliminar las máquinas virtuales. Para ello interactúa con otros servicios como es el caso del Servicio de Identidad (Keystone) para verificar la identidad y los permisos de los usuarios, con el Servicio de Imágenes (Glance) para obtener las imágenes con las cuales inicializar las máquina virtuales, con el Servicio de Red (Neutron) para proporcionarles conectividad de red a las instancias y opcionalmente con el Servicio de Almacenamiento en Bloques (Cinder) para conectar volúmenes de almacenamiento en bloque a las máquinas virtuales con el propósito de garantizarle un almacenamiento permanente de los datos.

1.4.1 Arquitectura de Nova: Componentes Internos e Hipervisores

Nova API es uno de los componentes del Servicio de Computación y está compuesto por el servicio nova-api y el servicio nova-api-metadata. El servicio nova-api acepta y responde a las llamadas API realizadas por usuarios que desean acceder al Servicio de Computación. Para ello soporta tres tipos de APIs: OpenStack Compute API, Amazon EC2 API y una API especial llamada Admin API para usuarios privilegiados que realizan tareas administrativas. Este servicio de nova-api se encarga de hacer cumplir las políticas definidas para Nova e iniciar la mayoría de las tareas de orquestación. Una arquitectura de despliegue multi-nodo de OpenStack incluye, en su versión minimalista, un Nodo Controlador y uno o varios Nodos de Computación. Los servicios nova-api y nova-api-metadata se instalan en el Nodo Controlador.[10]

Otro componente de Nova es el servicio nova-compute. Este componente se instala en cada uno de los Nodos de Computación de un despliegue OpenStack. Es precisamente este servicio el encargado de crear, reiniciar y terminar las instancias de máquinas virtuales, conectar y desconectar volúmenes de almacenamiento en bloque a estas instancias y proporcionar acceso a la consola de las mismas, entre otras funcionalidades. Nova incluye drivers para un determinado conjunto de hipervisores soportados. El servicio nova-compute hace uso de estos drivers para comunicarse con el hipervisor instalado en el Nodo de Computación donde se va a crear la instancia o donde se encuentra la instancia que se va a modificar. El driver por su parte se comunica con el hipervisor a través de la API del hipervisor. [9]

KVM[1] (Kernel-based Virtual Machine) es uno de los hipervisores soportados por Nova. El driver utilizado para controlar el hipervisor es libvirt[2]. KVM constituye una solución de virtualización completa (full virtualization) para servidores Linux con arquitectura de procesador x86 que contienen extensiones de virtualización (Intel-VT o AMD-V)

[1] http://www.linux-kvm.org/page/Main_Page
[2] http://libvirt.org/

para aceleración de hardware. Este hipervisor permite la ejecución de múltiples máquinas virtuales que ejecutan una versión no modificada de la imagen de una distribución Linux o Windows de manera que cada máquina virtual tiene su propio hardware virtual. Si el CPU del nodo de computación no soporta las extensiones de virtualización Intel-VT o AMD-V, entonces la opción de hipervisor a utilizar es QEMU que también implementa virtualización completa pero es mucho menos eficiente, por lo que generalmente se emplea en entornos de desarrollo.[14]

Para entender cómo las extensiones de virtualización del CPU pueden acelerar el rendimiento es necesario conocer cómo funciona el CPU de una máquina virtual. En hardware real, el sistema operativo (OS) traduce los programas de alto nivel en instrucciones que pueden ejecutarse en el CPU físico. En una máquina virtual sucede lo mismo, sin embargo, la diferencia principal radica en que el CPU virtual de la instancia está emulado, es decir, virtualizado por el hipervisor. Por lo tanto, el software del hipervisor atrapa estas instrucciones de alto nivel, las emula en software y devuelve el resultado que normalmente devolvería el CPU físico. Este proceso implica una sobrecarga que va en detrimento del rendimiento y para minimizar esta sobrecarga los procesadores modernos incluyen las extensiones de virtualización que permiten que una porción del CPU físico pueda ser directamente mapeado con el CPU virtual de la instancia, y que de esta forma las instrucciones destinadas al CPU virtual se ejecuten directamente en esa porción del CPU físico.[16]

Nova también soporta otros hipervisores como Xen[3], XenServer[4] y Xen Cloud Platform (XCP)[5], así como a Microsoft Hyper-V[6] y VMware vSphere[7]. El driver que Nova utiliza para Xen se llama XenAPI. Nova crea instancias Linux y Windows basadas en imágenes VMware a través de una conexión con un servidor vCenter o con un host donde esté instalado el hipervisor ESXi. El driver utilizado para VMware se llama VMwareAPI.[10]

El Servicio de Computación (Nova) utiliza el driver libvirt para comunicarse con LXC[8], que constituye una tecnología de virtualización ligera a nivel de sistema operativo que permite crear contenedores Linux que se ejecutan como una especie de máquina virtual ligera y que por lo tanto contribuyen a mejorar el rendimiento.

Los contenedores aíslan y encapsulan los flujos de trabajo de las aplicaciones con respecto al sistema operativo del servidor anfitrión. Un contenedor puede considerarse

[3] http://www.xenproject.org/
[4] http://xenserver.org/
[5] http://www-archive.xenproject.org/products/cloudxen.html
[6] http://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx
[7] https://www.vmware.com/products/vsphere
[8] https://linuxcontainers.org/

como un sistema operativo dentro del propio sistema operativo del servidor anfitrión y para todos los efectos se comporta como una máquina virtual. El kernel de Linux permite esta emulación y el proyecto LXC proporciona plantillas para distintas distribuciones de sistemas operativos de contenedores así como aplicaciones de entorno usuario que se encargan de la gestión del ciclo de vida del contenedor.[17]

LXC no emula la capa de hardware para los contenedores, sino que hace uso de novedosas tecnologías habilitadas en el kernel de Linux como son grupos de control (cgroups) y espacios de nombre (namespaces), las cuales permiten crear entornos de virtualización ligera con velocidades próximas a soluciones bare-metal. Debido a que no se virtualiza el almacenamiento, un contenedor no tiene que tener en cuenta el sistema de almacenamiento y sistema de ficheros subyacente y puede operar encima de cualquier versión de estos.[17]

Esto cambia profundamente la forma en la que se virtualizan los flujos de trabajo y las aplicaciones, dado que los contenedores son más rápidos, más portables y pueden escalar de manera más eficiente que las soluciones basadas en virtualización de hardware, a excepción de casos donde los flujos de trabajo o aplicaciones requieren un sistema operativo diferente de Linux o una versión específica del kernel de Linux.[18]

Docker[9] constituye otra solución de virtualización ligera a nivel de sistema operativo que ofrece una manera simple y eficiente de desplegar contenedores Linux que ofrecen servicios a partir de aplicaciones débilmente acopladas. Estos contenedores Docker incluyen sus propios componentes de software así como sus dependencias. Su principal ventaja es su capacidad para proporcionar el mecanismo necesario para automatizar el despliegue seguro y repetible de un entorno de aplicaciones. La primera versión de Docker fue lanzada en marzo de 2013 y estaba basada en el proyecto LXC, aunque actualmente Docker ha desechado a LXC como su entorno de ejecución por defecto en virtud de su propia librería llamada libcontainer, implementada en Go y que le proporciona acceso directo a las APIs de los contenedores Linux.[19]

Como puede observarse en la Figura 9, a partir de la versión 0.9 de Docker, este es capaz de manipular espacios de nombre (namespaces), grupos de control (cgroups), perfiles apparmor, interfaces de red, reglas de filtrado de cortafuegos, todo ello de manera consistente y predecible sin depender de LXC o ningún otro paquete, lo que potencia la estabilidad y por ese motivo se decidió que libcontainer fuera el entorno de ejecución por defecto de Docker, quedando el uso de LXC como una opción. [20]

[9] https://www.docker.com/

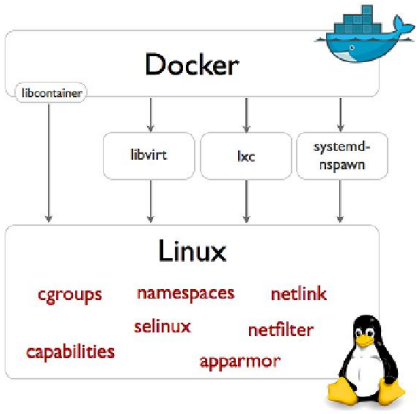


Figura 9. Entornos de ejecución de Docker

Otro componente del Servicio de Computación es el planificador o nova-scheduler, el cual se instala en el Nodo Controlador de la arquitectura de OpenStack. El planificador es el encargado de tomar las solicitudes de creación de máquinas virtuales de la cola interna de Nova y determinar en qué nodo de computación la máquina virtual debe ejecutarse. Para ello aplica una serie de filtros y solo considera como candidatos aquellos nodos de computación que pasen estos filtros y luego los ordena de acuerdo a sus pesos ponderados que están determinados por los recursos de hardware con los que cuenten estos nodos y finalmente le da mayor prioridad en la selección al que mayor peso tenga. [14]

Los filtros que por defecto vienen habilitados en el fichero de configuración del servicio (/etc/nova/nova.conf) son [14]:

- ✓ RetryFilter: determina si el nodo ha sido encuestado anteriormente para solicitar la creación y ejecución de la instancia utilizando sus recursos. Esto evita volver a encuestar a un nodo que ya ha sido previamente encuestado y no ha podido satisfacer la solicitud.
- ✓ AvailabilityZoneFilter: selecciona a los nodos que se encuentren en una zona de disponibilidad especificada.
- ✓ RamFilter: selecciona aquellos nodos que posean los recursos de memoria RAM disponibles para asignárselos a la nueva máquina virtual.
- ✓ ComputeFilter: selecciona los nodos que estén operacionales y habilitados
- ✓ ComputeCapabilitiesFilter: selecciona los nodos cuyas capacidades de computación satisfagan los parámetros adicionales especificados para la instancia.
- ✓ ImagePropertiesFilter: selecciona los nodos que satisfagan metadatos especificados en la imagen con la que se construye la instancia y que incluyen tipo de arquitectura, tipo y versión de hipervisor y modo de la máquina virtual.

- ✓ ServerGroupAntiAffinityFilter: selecciona los nodos que no pertenecen a un determinado grupo de afinidad.
- ✓ ServerGroupAffinityFilter: selecciona los nodos que se encuentran en un mismo grupo de afinidad

El componente del Servicio de Computación que se encarga de funcionar como mediador entre el servicio nova-compute y la base de datos recibe el nombre de nova-conductor. Este evita que el servicio nova-compute acceda de manera directa a la base de datos Nova y escala de manera horizontal. Este componente es instalado en el Nodo Controlador de la arquitectura de OpenStack.[10]

Existen muchos más componentes de Nova encargados del resto de las funciones de este complejo servicio y todos ellos se instalan junto a nova-api, nova-scheduler y nova-conductor, en el Nodo Controlador.

1.4.2 Servicio de Red de Nova (nova-network)

Nova posee un componente llamado nova-network, que aunque en la mayoría de los grandes despliegues de producción es remplazado por el proyecto de red de OpenStack llamado Neutron, constituyó por mucho tiempo la única solución para proporcionar conectividad de red a las instancias creadas con nova-compute hasta que OpenStack lanzara a Neutron en su versión Folsom de septiembre de 2012.

El servicio nova-network le permite a los proveedores de servicios crear redes capa 2 a partir de utilizar la tecnología de Linux Bridge (brctl). Estos Linux Bridges pueden ser configurados de manera estática en modelos planos (flat) o pueden configurarse de manera dinámica en modelos basados en VLANs. Nova-network igualmente se encarga de la asignación y gestión de direcciones IP a los tenants o proyectos y almacena toda la información en una base de datos SQL, que en la mayoría de los casos es una base de datos MySQL o MariaDB.[21]

Nova-network también se encarga de configurar un proceso dnsmasq que trabaja en calidad de servidor DNS y DHCP para dar servicio a las máquinas virtuales. Finalmente, hace uso de Linux Iptables para configurar reglas de filtrado de tráfico y llevar a cabo traducción de direcciones (NAT). A partir de la utilización de Linux Iptables, nova-network crea grupos de seguridad que filtran el tráfico entrante a la máquina virtual y configura reglas de traducción de direcciones que le permiten a la máquina virtual acceder y ser accedida desde Internet.[21]

Nova-network soporta tres modelos de red básicos: Flat, Flat DHCP y basado en VLANs.

Modelo Flat

En el modelo Flat mostrado en la Figura 10, todas las máquinas virtuales que se ejecutan en cada nodo de computación, ya sea si estas pertenecen a un mismo tenant o a múltiples tenants, están conectadas directamente al mismo bridge, usualmente un Linux Bridge, por lo tanto todo el tráfico de las máquinas virtuales es directamente puenteado a través de la interfaz de red física del Nodo de Computación hacia la red de trasporte física o VLAN física. Las direcciones IP y el enrutador por defecto son parámetros de configuración proporcionados a las instancias por un servidor DHCP externo que no forma parte de ningún componente de OpenStack, por lo que las máquinas virtuales tienen direcciones IP que pertenecen a una red de trasporte física o VLAN física existente en el centro de datos. Por otro lado, para que las máquinas virtuales puedan acceder y ser accedidas desde Internet, un enrutador externo aplica reglas de traducción de direcciones al tráfico entrante y saliente.[21]

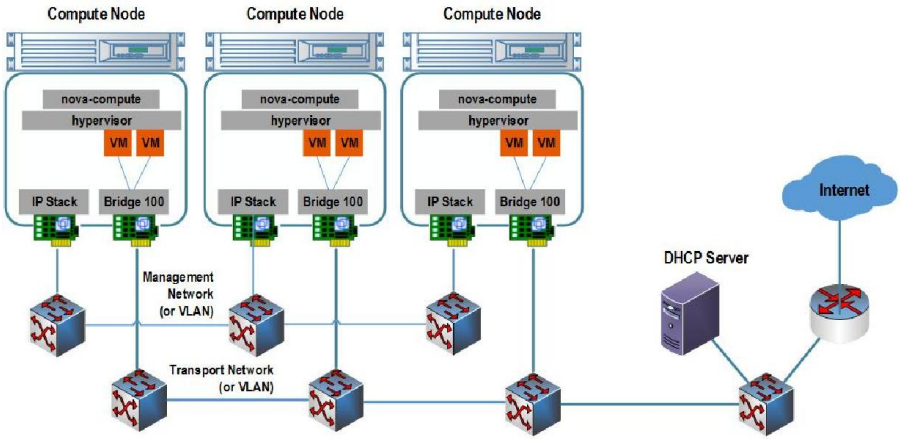


Figura 10. Arquitectura del modelo Flat de nova-network

Como puede constatarse, el modelo Flat de nova-network no incluye soporte para entornos multi-tenant más allá de los grupos de seguridad implementados con Linux Iptables.

Modelo Flat DHCP

El modelo Flat DHCP mostrado en la Figura 11 es muy similar al modelo Flat en el sentido que en ambos modelos, las máquinas virtuales están conectadas a un mismo Linux Bridge y todo el tráfico es directamente puenteado hacia la red física o VLAN a través de la interfaz de red física del Nodo de Computación. No obstante, una de las diferencias con el modelo Flat es que en el modelo Flat DHCP, se cuenta con un proceso llamado dnsmasq que se ejecuta en uno o en múltiples Nodos de Computación y que funciona como un servidor DHCP, por lo tanto es responsable de dinámicamente asignar la ruta por defecto y las direcciones IP del rango de la red física o VLAN a las máquinas virtuales conectadas a los Linux Bridges correspondientes en los Nodos de Computación.[21]

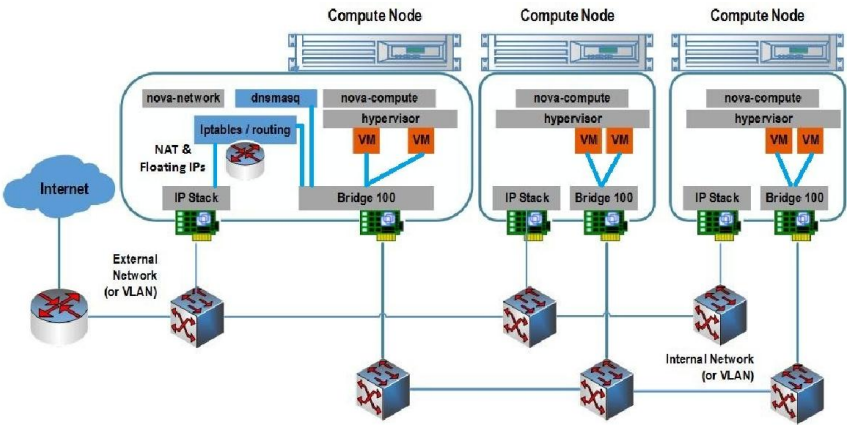


Figura 11. Arquitectura del modelo Flat DHCP de nova-network

Otra diferencia consiste en que el modelo DHCP, además de ofrecer capacidades de filtrado de tráfico y traducción de direcciones para que las instancias puedan acceder y sean accedidas desde Internet, proporciona capacidades de enrutamiento. Por ejemplo, si una máquina virtual quiere acceder a un servidor web en Internet, el tráfico de solicitud tiene que atravesar el Linux Bridge que conecta a la máquina virtual con la interfaz de red física del Nodo de Computación, posteriormente este tráfico es puenteado hacia la red física o VLAN y llega al Nodo de Computación donde se está ejecutando el servicio nova-network, siendo este servicio el encargado del enrutamiento el tráfico hacia Internet, pero antes de llevar a cabo el enrutamiento de la solicitud web, nova-network realiza un SNAT (Source Address Translation) al tráfico, es decir, una traducción de direcciones fuente, de manera que los paquetes tengan como dirección IP fuente, la dirección IP pública correspondiente a la interfaz de red del Nodo de Computación donde se ejecuta la instancia y que está conectada a la red externa. En el caso de que la instancia deba ser accedida desde Internet pues en ella se ejecuta algún servicio que se necesita que sea público, entonces nova-network lleva a cabo un DNAT (Destination Address Translation) o traducción de direcciones destino, es decir, los paquetes de solicitud procedentes del exterior que llegan a nova-network tienen como dirección IP destino una dirección IP pública que es la correspondiente a la interfaz de red del Nodo de Computación donde se ejecuta la instancia en la red externa, y nova-network traduce esta dirección IP pública a la dirección IP privada del Nodo de Computación correspondiente a la interfaz que este nodo tiene en la red interna.[21]

Al igual que en el modelo Flat, el modelo Flat DHCP no incluye soporte para entornos multi-tenant ya que todas las máquinas virtuales están conectadas a la misma red plana. La única solución para lograr el aislamiento entre tenants es a partir de utilizar Iptables para configurar grupos de seguridad, que no son más que un conjunto de reglas de filtrado aplicadas a las máquinas virtuales.

Modelo basado en VLANs

A diferencia de los dos modelos anteriores, el modelo basado en VLANs brinda soporte para entornos multi-tenant a partir de asignar una VLAN independiente a cada tenant. De esta manera y como se observa en la Figura 12, cada tenant tiene su propia red que es mapeada directamente con un Linux Bridge virtual en el Nodo de Computación y con una VLAN física que debe existir y estar pre-configurada en la infraestructura física del centro de datos.[21]

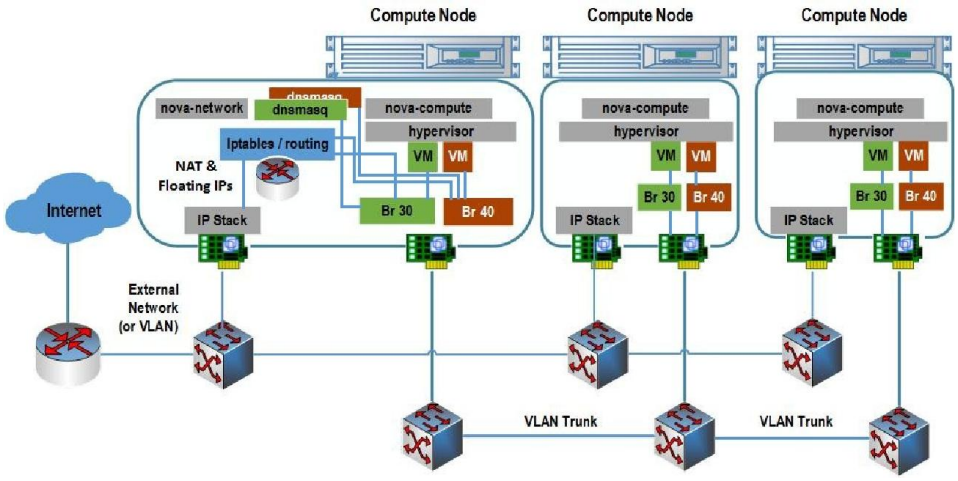


Figura 12. Arquitectura del modelo basado en VLANs de nova-network

De esta manera, el tráfico de las máquinas virtuales es puenteado hacia la red física a través de un Linux Brigde configurado con una VLAN por proyecto. Las máquinas virtuales de un tenant están aisladas de las de otro tenant, a pesar de que se ejecuten sobre el mismo Nodo de Computación. Sin embargo, este alcance se queda corto en términos de escalabilidad debido al simple hecho de que el estándar IEEE 802.11q define el identificador de VLAN como un campo de 12 bits, por lo tanto solo pueden haber 4094 VLANs, ya que los valores de 0 y FFF están reservados.[21]

Igualmente el modelo basado en VLANs proporciona mecanismos internos para el enrutamiento y filtrado del tráfico entrante y saliente de las máquinas virtuales, así como para implementar los distintos tipos de traducción de direcciones (SNAT y DNAT). En el modelo basado en VLANs, cada tenant tiene asociado un proceso dnsmasq que funciona como servidor DHCP encargado de asignar las direcciones IP de manera dinámica a las máquinas virtuales del tenant. Estas direcciones IP corresponden a las de una VLAN pre-configurada en el centro de datos.[21]

Otras dos limitantes del modelo basado en VLANs constituyen la poca flexibilidad que ofrece a los tenants dado que estos no pueden escoger el rango de direcciones IP de sus redes debido a la posibilidad de que el rango escogido por un tenant se solape con el escogido por otro tenant y desafortunadamente no existe una solución implementada

para resolver este problema en nova-network, y finalmente la incapacidad de que existan dos o más redes por tenant, estando el número de redes que un tenant puede tener restringido a uno. En otras palabras, los tenants no tienen absolutamente ningún control sobre la configuración de la capa de red virtual.[21]

Después de haber analizado los tres modelos, cabe destacar que a pesar de las limitaciones aparejadas al uso de nova-network, este componente constituye la solución de conectividad de red para máquinas virtuales escogida actualmente en no pocos despliegues de OpenStack, aun cuando existe un proyecto independiente llamado Neutron que surgió con el fin de solucionar las deficiencias de nova-network y que por lo general constituye la solución recomendada para la capa de red de una infraestructura de Nube OpenStack.

1.5 Servicio de Red

El proyecto de Red de OpenStack llamado Neutron es un servicio independiente que puede ser instalado de manera aparte al resto de los servicios de esta plataforma de Nube. Neutron comprende un conjunto de componentes que pueden desplegarse en múltiples nodos con el fin de obtener altos niveles de fiabilidad y redundancia. Igualmente todos estos componentes pueden instalarse en un solo nodo. Una arquitectura de despliegue multi-nodo de OpenStack concebida para escenarios de producción, además de incluir el Nodo Controlador y uno o varios Nodos de Computación, incluye un Nodo de Red. Los componentes del servicio Neutron se distribuyen entre los tres tipos de nodos en escenarios de producción.

La última versión de Neutron incluida en OpenStack Kilo incluye numerosos servicios de red presentes actualmente en la mayoría de los centros de datos. Estas funcionalidades incluyen conmutación y enrutamiento de paquetes, balanceo de carga, filtrado de paquetes y traducción de direcciones, redes privadas virtuales, entre otras. Estas tecnologías puede proporcionarse ya sea mediante soluciones de código abierto o comerciales integradas con OpenStack.

En Neutron, los conmutadores virtuales no son más que aplicaciones de software que conectan las interfaces virtuales de las máquinas virtuales a nivel de la capa de enlace del modelo OSI. Entre las múltiples plataformas de conmutación soportadas por Neutron se encuentran Linux Bridging y Open vSwitch (OVS), siendo esta última una solución basada en conmutadores virtuales que soportan interfaces estandarizadas y protocolos tales como OpenFlow, NetFlow, SPAN, RSPAN, LACP y IEEE 802.1q. Además de incluir soporte para la configuración de VLANs, Neutron incluye soporte para desplegar redes overlay en software utilizando protocolos de tunelización como Virtual Extensible LAN (VXLAN) y Generic Routing Encapsulation (GRE).

Las capacidades de enrutamiento y traducción de direcciones proporcionadas por Neutron son implementadas a partir de utilizar Linux Iptables y Espacios de Nombre de Red o network namespaces. Neutron le permite a cada tenant gestionar sus propios recursos de computación y red, es decir, los usuarios pertenecientes a un tenant determinado pueden crear sus propias subredes, crear enrutadores que interconecten dichas subredes y conectar sus máquinas virtuales a dichas subredes, configurar las tablas de rutas de dichos enrutadores, crear cortafuegos configurados con reglas de filtrado para aplicar políticas comunes que filtren el tráfico entrante y saliente de las máquinas virtuales de este tenant y configurar balanceadores de carga para balancear las solicitudes de un servicio entre un conjunto de máquinas virtuales que lo ofrecen. Para aislar los recursos en este tipo de entorno multi-tenant, Neutron utiliza los espacios de nombre de red o network namespaces.

Un espacio de nombre de red o network namespace es una replicación lógica del stack de red que incluye sus propios dispositivos e interfaces de red, sus propias tablas de rutas, su propio proceso Iptables con sus propias reglas para filtrado de tráfico y traducción de direcciones. Por lo tanto, cada vez que un tenant crea una red con su correspondiente servidor DHCP, un enrutador o un balanceador de carga, cada uno de estos elementos de red se representan como network namespaces. En pocas palabras, los network namespaces constituyen la solución empleada por Neutron para proporcionar de manera aislada servicios DHCP y de enrutamiento de tráfico a múltiples tenants, de forma tal que estos puedan crear sus propias subredes las cuales pueden solaparse con la de otros tenants en cuanto a espacio de direccionamiento IP, pero como cada subred pertenece a un network namespace diferente están aisladas una de la otra y no habrá conflictos.

Existen múltiples convenciones para denotar los network namespaces. Para servidores DHCP, los network namespaces siguen el siguiente formato: qdhcp-<network UUID> que contiene un servicio DHCP que proporciona las direcciones IP a las instancias que pertenezcan a la subred asociada al mismo. Este qdhcp namespace tiene una interfaz conectada a un conmutador virtual ubicado dentro del Nodo de Red y este conmutador virtual está conectado al resto de las máquinas virtuales que pertenecen a la misma capa 2 o subred y que se encuentran ejecutándose en los Nodos de Computación. Para los enrutadores, el formato del namespaces es qrouter-<router UUID> y representa a un enrutador que encamina el tráfico entre las diferentes subredes que tiene conectadas y entre estas subredes e Internet.

Los grupos de seguridad y los cortafuegos son los dos métodos que el Servicio de Red de OpenStack emplea para proporcionar seguridad a las instancias y a las redes. Al igual que en nova-network, un grupo de seguridad consisten en un conjunto de reglas

de filtrado de tráfico implementadas con Linux Iptables y configuradas en el Nodo de Computación donde se encuentran las instancias cuyo tráfico entrante o saliente se desea filtrar. El segundo método que consiste en Cortafuegos como Servicio o Firewall-as-a-Service (FWaaS) propone una estrategia diferente en la que la seguridad es implementada en el enrutador que se ejecuta en el Nodo de Red en vez de implementarse en el Nodo de Computación. Por lo tanto, todas las reglas de filtrado que controlan el tráfico entrante y saliente de las instancias conectadas a las redes de un tenant determinado, son configuradas en el enrutador de dicho tenant el cual se ejecuta en el Nodo de Red como un proceso aislado en su propio network namespace.

Neutron soporta dos servicios adicionales que son Balanceo de Carga como Servicio o Load-Balancing-as-a-Service (LBaaS) y Red Privada Virtual como Servicio o VPN-as-a-Service (VPNaaS). El primero de estos servicios permite a los usuarios distribuir las solicitudes de los clientes entre un conjunto de instancias de servidores que ofrecen un servicio determinado a estos clientes. Para ello Neutron incluye un plugin que hace uso del HAProxy como implementación software de un balanceador de carga. El segundo de estos servicios ofrece la posibilidad de extender la red privada de los clientes sobre Internet o cualquier otra red pública a partir de un conjunto de APIs que permiten que estos creen túneles VPN basados en IPSec que conectan de manera segura sus redes con endpoints remotos.

1.5.1 Arquitectura de Neutron

Los components del Servicio de Red (Neutron) pueden resumirse en un servidor Neutron accesible mediante una API e instalado en el Nodo Controlador, un plugin de conmutación instalado en el Nodo Controlador y en el Nodo de Red, el correspondiente agente para dicho plugin instalado en el Nodo de Red y en cada Nodo de Computación, así como un agente de capa 3, un agente DHCP y un agente de metadatos instalados en el Nodo de Red.

El servicio neutron-server expone una API a los usuarios y se encarga de realizar un procesamiento inicial de las solicitudes de los usuarios y posteriormente pasarlas al plugin y agentes de red para que estos se encarguen de realizar un procesamiento adicional.

Con anterioridad a Neutron, la implementación de red original de OpenStack (nova-network) proporcionaba funcionalidades de red básica a través del empleo de Linux Bridges, VLANs y Linux Iptables. El Servicio de Red (Neutron) introdujo soporte para integrarse con plugins de terceros que extienden las funcionalidades de red a las cuales los clientes pueden tener acceso y mejoran la implementación de la API de Neutron. Las empresas fabricantes de estos plugins ofrecen una implementación personalizada del back-end de la API de Neutron. Un plugin puede utilizar una variedad de

tecnologías para implementar la lógica para manipular las solicitudes a la API de Neutron. Algunos plugins utilizan soluciones básicas como Linux VLANs e Iptables mientras que otros plugins se basan en el uso de tecnologías más sofisticadas como tunelización L2-in-L3 o el protocolo OpenFlow para ofrecer funcionalidades similares.

Los plugins pueden tener propiedades diferentes para cada conjunto de requerimientos de hardware, funcionalidades, requerimientos de rendimiento, escalabilidad y herramientas del operador. Debido a que Neutron soporta un gran número de plugins, el administrador de nube deberá determinar la tecnología de red apropiada para desplegar entre las disponibles en la versión Kilo de Neutron. Estas incluyen Big Switch (Floodlight REST Proxy), Brocade, Cisco, Cloudbase Hyper-V, IBM SDN-VE, Linux Bridge, Midonet, ML2 (Modular Layer 2), NEC OpenFlow, Open vSwitch, OpenFlow Agent, PLUMgrid y VMware NSx.

El plugin escogido por el administrador para desplegar el Servicio de Red de OpenStack es cargado en tiempo de ejecución por el servicio de Neutron (neutron-service). Este plugin procesa las llamadas API y almacena los datos de la configuración lógica del modelo de red resultante en una base de datos dedicada a este servicio que suele ser MySQL o MariaDB. Esta información almacenada en la base de datos de Neutron es leída por los distintos agentes, por ejemplo en caso de emplear el plugin ML2 y el agente de Open vSwitch, es precisamente este último agente el encargado de a partir de la información leída, comunicarse con la instancia Open vSwitch ejecutándose en los Nodos de Computación y configurar los flujos en los switches virtuales Open vSwitch de manera que la red quede configurada según el modelo lógico de red.

Además del plugin ML2, otros dos plugins muy utilizados son los plugins de Linux Bridge y Open vSwitch. Estos proporcionan conectividad a nivel de capa 2 a las instancias, así como otros recursos de red a través de VLANs y tecnologías de red overlay como GRE y VXLAN. Ambos proporciona una infraestructura de conmutación a nivel de capa 2, pero cada uno lo hace de manera única. Ambos plugins son considerados monolíticos, lo que significa que no pueden emplearse de manera simultánea con ningún otro plugin. Los plugins de Linux Bridge y Open vSwitch han quedado en desuso en favor del plugin Modular Layer 2 (ML2) que permite trabajar con múltiples agentes, Linux Bridge u Open vSwitch, de manera simultánea.

El plugin Modular Layer 2 (ML2) es un framework que permite a Neutron utilizar una amplia gama de tecnologías de red a nivel de capa 2 típicamente desplegadas en grandes centros de datos a escala global. Actualmente trabaja con los agentes L2 openvswitch, linuxbrige y hyperv, y se pretende que remplace a los plugins monolíticos asociados con estos agentes L2. El framework ML2 igualmente simplifica

las tareas de incluir soporte para nuevas tecnologías de red a nivel de capa 2 pues requiere mucho menos esfuerzo que el que conllevaría añadir soporte para un nuevo plugin monolítico.

El framework ml2 está compuesto por drivers que implementan de manera separada un conjunto extensible de tipos de red y mecanismos de red para acceder a redes de estos tipos. Muchos mecanismos pueden emplearse de manera simultánea para acceder a puertos diferentes de la misma red virtual. Estos mecanismos pueden utilizar agentes L2 via RPC y/o mecanismos basados en drivers para interactuar con dispositivos externos o controladores.

Como se muestra en la Figura 13, el framework ML2 contiene dos tipos de drivers, Type Drivers y Mechanism Drivers. Los Type Drivers mantienen toda la información relativa al estado de red específico para cada tipo de red en particular, realizan la validación del proveedor de red y asignan recursos de red a los tenants. Los Mechanism Drivers son responsables de tomar la información directamente proporcionada por los Type Drivers y garantizan que esta es apropiadamente aplicada según los mecanismos de red específico que han sido habilitados. La interfaz de los Mechanism Driver soporta actualmente la creación, actualización y eliminación de redes y puertos. Los Type Drivers disponibles actualmente son: Local, Flat, VLAN, GRE y VXLAN mientras que los Mechanism Drivers disponibles son: Arista, Cisco Nexus, Hyper-V, L2 Population, LinuxBridge, Open vSwitch y Tail-F NCS.

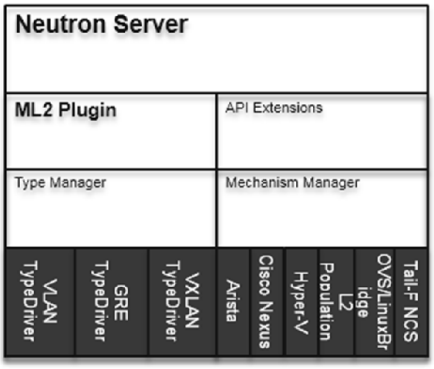


Figura 13. Arquitectura del ML2 Plugin

El Servicio de Red de OpenStack incluye una extensión que proporciona a los usuarios la habilidad de dinámicamente crear y configurar enrutadores virtuales a través de llamadas a la API de Neutron. Cada enrutador virtual interconecta los segmentos de redes virtuales L2 internas de un determinado tenant y se le asigna una dirección IP flotante que se corresponde con una dirección IP pública a la interfaz de este enrutador hacia una red virtual externa creada previamente en Neutron y cuyo rango IP se corresponde con un rango IP accesible desde Internet. De esta manera, cuando una

instancia, conectada a una de las redes virtuales internas del tenant, desea acceder a Internet y no tiene una dirección IP flotante asignada directamente, el enrutador realizará un SNAT al tráfico procedente de esta instancia utilizando para ello la dirección IP flotante de su interfaz conectada a la red virtual externa. El agente neutrón-l3-agent utiliza el stack IP de Linux así como Linux Iptables para llevar a cabo el enrutamiento a nivel de capa 3 y la traducción de direcciones. Con el fin de soportar múltiples enrutadores con direcciones IP con alta probabilidad de sobrelaparse, el agente neutron-l3-agent hace uso de los network namespaces para garantizar el aislamiento de los contextos de encaminamiento. Este agente es el mismo que se instala con independencia del plugin seleccionado y es instalado en el Nodo de Red de un despliegue OpenStack.

El servicio neutron-dhcp-agent es responsible de iniciar y controlar el proceso dnsmasq para cada subred de cada tenant que requiere el servicio de DHCP. Este servicio igualmente inicializa el proceso neutron-ns-metadata-proxy como parte del sistema de metadatos que es utilizado independientemente del plugin seleccionado. Por defecto, Neutron utiliza dnsmasq, una implementación libre y ligera de los servicios DNS y DHCP.

El Servicio de Computación (Nova) emplea un servicio de metadatos especial para permitir que las máquinas virtuales puedan obtener datos específicos acerca de ellas mismas. Esto incluye información como el hostname, llaves públicas, entre otros datos. Las instancias acceden al servicio de metadatos a través de HTTP durante la fase de arranque.

El agente neutron-metadata-agent proporciona un proxy a través de los namespaces del DHCP o del enrutador que permite el acceso al servicio nova-api-metadata que se ejecuta en el Nodo Controlador. Cada red tiene asignado su correspondiente proceso neutron-ns-metadata-proxy que es creado por el agente de metadatos. Con Neutron, el proceso de proporcionar metadatos a las instancias varía y está basado en el uso de enrutadores.

La Tabla 3 resume los distintos componentes del Servicio de Red (Neutron) en el caso de que se emplee el plugin ML2 con el agente L2 Open vSwitch y especifica dónde deben instalarse en el caso de un despliegue multi-nodo de OpenStack en entorno de producción.

Tabla 3. Componentes de Neutron utilizando el plugin ML2 y el agente L2 Open vSwitch

Nodo	Componentes
Controlador	neutron-service, neutron-plugin-ml2
De Red	neutron-plugin-ml2, neutron-plugin-openvswitch-agent, neutron-l3-agent neutron-dhcp-agent, neutron-metadata-agent
De Computación	neutron-plugin-ml2 neutron-plugin-openvswitch-agent

1.6 Servicio de Almacenamiento en Bloques

El Servicio de Almacenamiento en Bloques de OpenStack llamado Cinder es muy similar en cuanto a funcionalidades ofrecidas al servicio Amazon EC2 Elastic Block Storage (EBS). Cinder gestiona la creación y eliminación de volúmenes de almacenamiento en bloque, así como la asignación de estos a las máquinas virtuales creadas con el Servicio de Computación (Nova) y servidores físicos. Estos volúmenes de almacenamiento en bloque pueden gestionarse desde la Interfaz Web de OpenStack llamada Horizon, la cual permite a los administradores y usuarios satisfacer de manera sencilla sus necesidades de almacenamiento.

Además de utilizar tecnologías básicas de almacenamiento para servidores Linux, Cinder integra soporte para numerosas plataformas de almacenamiento del mercado incluyendo a Ceph, NetApp, Nexenta, SolidFire y Zadara. Además Cinder proporciona un servicio de gestión de snapshots con potentes funcionalidades para garantizar el respaldo de los datos almacenados en los volúmenes de almacenamiento en bloque. Los snapshots pueden ser restaurados o usados para crear nuevos volúmenes de almacenamiento en bloque.

Los volúmenes son dispositivos de bloques que se crean de forma independiente de la instancia y pueden asociarse y desasociarse de ella cuando se precise. Los volúmenes se crean en el nodo de almacenamiento (por defecto con LVM) y se conectan a la instancia que se desee mediante algún protocolo de redes de almacenamiento (iSCSI es el más utilizado). Cinder incluye controladores para gran cantidad de dispositivos de almacenamiento del mercado. Cuando Nova termina una instancia, todo el almacenamiento local asociado a ella se borra, pero no los volúmenes, por lo que estos reciben el nombre de almacenamiento permanente. Se puede pensar en los volúmenes como discos externos que se conectan o desconectan de las instancias, aunque se trate de un mecanismo completamente diferente.

La arquitectura interna de Cinder puede observarse en la Figura 14. La API de Cinder permite la manipulación de volúmenes, tipos de volúmenes y snapshots. Esta API llamada cinder-api acepta los pedidos y los enruta al componente cinder-volume para

ser procesados. El componente cinder-volume reacciona leyendo o escribiendo a la base de datos que mantiene el estado, interactúa con otros procesos como el cinder-scheduler a través de la cola de mensajes y directamente actúa sobre el almacenamiento proveyendo hardware o software. Por su parte, cinder-scheduler selecciona el nodo para el almacenamiento por bloques óptimo para la creación de un volumen sobre él. La cola de mensajes enruta información entre los procesos de Cinder, mientras que una base de datos almacena el estado de los volúmenes.

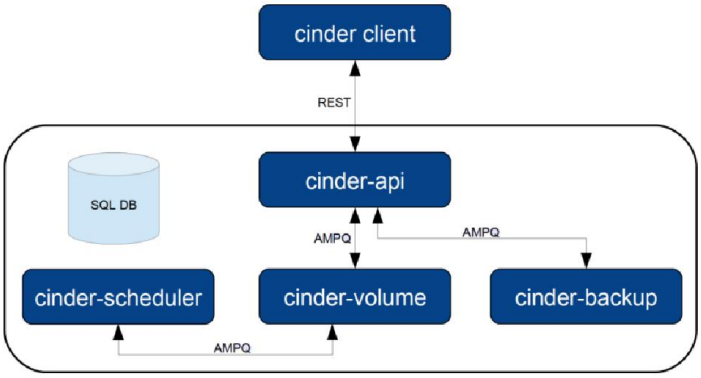


Figura 14. Arquitectura interna de Cinder

Por defecto, la API del Servicio de Almacenamiento en Bloques se ejecuta como un solo proceso. Esto limita el número de solicitudes API que el Servicio de Almacenamiento en Bloque puede atender en un momento determinado. En un entorno de producción, el throughput de la API del Servicio de Almacenamiento en Bloque debe incrementarse a partir de permitir que el servicio API se ejecute como múltiples procesos, tantos como los recursos de hardware de la máquina donde se ejecuta lo soporte.

El Servicio de Almacenamiento en Bloque debe instalarse en escenarios de despliegue de OpenStack que requieran persistencia de datos y tengan altos requerimientos en cuanto a rendimiento, tal es el caso del almacenamiento en bases de datos, sistemas de ficheros extensibles, o en el caso de que sea necesario proporcionar servidores con acceso a dispositivos de almacenamiento a nivel de bloque y en formato "raw".

1.7 Servicio de Almacenamiento de Objetos

El Servicio de Almacenamiento de Objetos de OpenStack recibe el nombre de Swift. Swift es un sistema de almacenamiento de objetos ampliamente utilizado y popular proporcionado bajo la licencia libre de código abierto de Apache 2. Está considerado como un sistema de almacenamiento de objetos con soporte multi-tenant, altamente escalable y resistente a fallos, especialmente diseñado para almacenar grandes cantidades de datos no estructurados a bajos costes. Swift permite almacenar ficheros, videos, resultados de procesamiento analítico de datos, contenido web, respaldos de información, imágenes de máquinas virtuales, snapshots de imágenes virtuales, así

como cualquier otro dato no estructurado, todo ello a gran escala con una alta disponibilidad y 12 nueves de durabilidad.

Swift puede ser configurado como mínimo con dos nodos de almacenamiento equipados con cierta cantidad de discos duros y posteriormente escalar gradualmente pero de manera sencilla hacia miles de nodos de almacenamiento con cientos de Petabytes de almacenamiento distribuido en regiones geográficamente distantes. Swift está diseñando desde sus raíces para escalar horizontalmente sin ningún punto de fallo.

Swift permite escalar de manera linear sobre la base de cuántos datos se necesita almacenar y cuántos usuarios necesitan recibir servicio. Un clúster Swift puede escalar desde unos cuantos pocos nodos con algunos discos duros hasta miles de servidores de almacenamiento con docenas, incluso cientos de Petabytes de almacenamiento. El rendimiento no se degrada en la medida que el clúster crece en cuanto a uso y aumento del número de solicitudes. Swift está diseñando desde sus raíces para escalar horizontalmente sin ningún punto de fallo, es decir, no existe un punto central de control. Cuando se necesita escalar, el sistema incrementa sus capacidades según se requiera, por ejemplo, a partir de añadir nodos de almacenamiento en el caso de que sea necesario almacenar más información, o a partir de añadir nodos proxies en el caso de que aumenten el número de solicitudes de parte de los clientes, o a partir de incrementar las capacidades de red suponiendo de que se detecte un cuello de botella.

Swift es utilizado a escala global en empresas de variado tamaño, proveedores de servicio y organizaciones vinculadas a la docencia o a la investigación. Actualmente las principales nubes de almacenamiento de objetos están implementadas en Swift, tal es el caso de Rackspace Cloud File, HP Cloud, IBM Softlayer e incontables clústeres privados de almacenamiento de objetos. Swift fue originalmente diseñado como el motor que respaldaba a RackSpace Cloud Files en 2010 y posteriormente, su código pasó a ser libre y a estructurarse como proyecto dentro de OpenStack. Con cientos de compañías y miles de desarrolladores contribuyendo y participando en este proyecto, el uso de Swift incrementa rápidamente su tasa de adopción.

Swift no es un sistema de ficheros tradicional o un dispositivo de almacenamiento en bloque “raw”. En cambio, permite almacenar, obtener y eliminar objetos así como los metadatos asociados a estos objetos a través de llamadas RESTful HTTP a la API de Swift. Todos los objetos almacenados en Swift tienen una URL única que los identifica de manera univoca y cada objeto tiene asociado un conjunto de metadatos. Los desarrolladores pueden escribir directamente una llamada que invoque la API de Swift o utilizar una de las múltiples librerías que existen para los lenguajes de programación populares como Java, Python, Ruby y C#.

El Servicio de Almacenamiento de Objetos se basa en una arquitectura distribuida sin ningún punto central de control lo que le permite alcanzar altos niveles de escalabilidad, redundancia y rendimiento. Los objetos son almacenados en múltiples discos duros físicos, y es el software de replicación de Swift el responsable de garantizar la replicación e integridad de estos datos en el clúster de almacenamiento. Cada objeto almacenado es replicado tres vedes en zonas tan únicas como sea posible. Una zona puede consistir en un subconjunto de discos duros ubicados en un nodo de almacenamiento, puede abarcar todos los discos duros de un nodo de almacenamiento, un conjunto de nodos de almacenamiento dispuestos en un rack de telecomunicaciones o un conjunto de racks que contienen nodos de almacenamiento. Si un nodo falla, Swift se encarga de replicar el contenido almacenado en los discos duros que formaban parte de este nodo hacia otros nodos que estén activos. Debido a que Swift utiliza software para garantizar la replicación de los datos y distribuirlos en diferentes dispositivos, es posible utilizar discos duros y servidores de bajos precios considerados "commodity hardware" y no es necesario invertir en equipamiento costoso que implemente RAID a nivel de hardware ya que las funcionalidades de tolerancia a fallos y distribución de datos que ofrece RAID son implementadas por Swift.

La arquitectura distribuida de Swift garantiza una elevada durabilidad de la información almacenada. La durabilidad de los datos se alcanza debido a que los objetos almacenados siempre estarán disponibles y se garantizará la integridad de la información almacenada en cada objeto. Para garantizar que un objeto está disponible de manera persistente, Swift realiza copias de cada objeto y distribuye estas copias a lo largo del clúster ubicando estas copias de manera que se encuentren lo menos cercanas las unas de las otras con el fin de garantizar altos niveles de tolerancia a fallos. Por defecto, Swift realiza tres copias de cada objeto. Para verificar que todos los datos aún siguen estando en buen estado y no se ha violado ningún principio de integridad de los mismos, se ejecuta un proceso de auditoría. De igual manera los replicadores de Swift se cercioran que siempre haya el número correcto de copias disponibles por cada objeto en el clúster. En el caso de que un disco duro falle, las copias de los objetos que se encontraban almacenadas en él se perderán, sin embargo por cada objeto cuya copia se perdió debido al fallo del disco duro, existen dos copias adicionales distribuidas en otros discos duros, e incluso, en otros nodos del clúster y una de estas copias será empleada para generar la copia perdida. Una vez que se replique la información perdida, esta será distribuida nuevamente a lo largo del clúster, garantizando que las tres copias de cada objeto se ubiquen de la manera lo menos cercana posible para mejorar la tolerancia a fallos. De esta manera se garantiza que siempre haya tres copias de cada objeto.

Swift puede distribuir datos a lo largo de múltiples centros de datos con altos niveles de latencia entre ellos. La distribución puede tener lugar debido a múltiples razones. Una de ellas es proporcionar alta disponibilidad de los datos a partir de permitir que estos puedan ser accedidos desde distintas regiones. Otra razón puede estar relacionada a la necesidad de disponer de una región designada como sitio de recuperación ante desastres. Swift permite esto a partir de permitirle a los operadores definir regiones y zonas dentro del clúster de almacenamiento. Las regiones generalmente especifican fronteras geográficas, como por ejemplo centros de datos ubicados en ciudades diferentes.

Las zonas son porciones de las regiones que definen puntos de fallos para un grupo de máquinas y de esta manera se logra aislar las fronteras de fallos. A un nivel más pequeño, una zona puede estar compuesta por un único disco duro o por un grupo de discos duros. Si hubiese cinco servidores de almacenamiento de objetos, cada servidor pudiera conformar una zona. Para grandes despliegues, una zona estará compuesta por un rack de servidores de almacenamiento de objetos o por múltiples racks. El objetivo de definir zonas es permitir que el clúster pueda rebasar fallos significativos en servidores de almacenamiento de objetos o discos duros sin perder todas las réplicas que se tienen de un objeto.

Como se dijo anteriormente, todo objeto en Swift tiene por defecto tres copias. Swift ubicará cada copia según el principio "tan-único-como-sea-posible" para garantizar altos índices de disponibilidad y durabilidad de la información. Este principio se basa en que a la hora de seleccionar una ubicación para cada réplica del objeto, Swift selecciona un servidor de almacenamiento en una zona donde no exista una copia del objeto en vez de seleccionar un servidor de almacenamiento ubicado en una zona que ya contiene una copia almacenada del objeto. En caso de que no se cuente con esta zona, tratará entonces dentro de la zona que ya posee una réplica del objeto, seleccionar un rack que no contenga ningún servidor de almacenamiento con una réplica almacenada del objeto. Si no logra encontrar este rack, entonces dentro del rack buscará un servidor de almacenamiento que no contenga una copia del objeto, y si por último no lo encuentra entonces pasará a buscar dentro del servidor, un disco duro que no contenga una copia del objeto.

Swift fue concebido para distribuir las solicitudes a lo largo de múltiples servidores de almacenamiento a partir de utilizar servidores proxy. De esta manera, Swift puede tomar ventaja de las capacidades de los servidores disponibles para manipular múltiples solicitudes de manera simultánea. Esto incrementa la concurrencia en el sistema y el throughput total disponible.

La habilidad de Swift de utilizar "commodity hardware", es decir, hardware barato implica que no existe ningún tipo de lock-in con ningún fabricante de hardware de almacenamiento en particular. Como resultado de esto, los despliegues pueden continuamente beneficiarse de la reducción de los precios del hardware e incrementar las capacidades de almacenamiento. Swift también permite que los datos sean movidos de un dispositivo a otro para resolver restricciones relacionadas con la latencia y las tasas I/O.

1.7.1 Arquitectura y componentes de Swift

Los componentes del Servicio de Almacenamiento de Objetos (Swift) que permiten garantizar una alta disponibilidad, durabilidad y concurrencia son servidores proxies, anillos, particiones, cuentas, contenedores, objetos, zonas y regiones.

Los servidores proxies son la cara pública del Servicio de Almacenamiento de Objetos y se encargan de manipular todas las solicitudes entrantes realizadas por los usuarios a través de la API de Swift. Una vez que un servidor proxy recibe una solicitud, este determina el nodo de almacenamiento donde está almacenado el objeto solicitado basándose en la URL del objeto proporcionada por el cliente y que identifica de manera única al objeto e indica su ubicación dentro del clúster. Los servidores proxies también coordinan las respuestas, se encargan de la manipulación de los fallos y de coordinar las estampillas de tiempo.

Los servidores proxies utilizan una arquitectura en la que no se comparte nada y pueden escalar tanto como se necesite en base a la demanda. Un mínimo de dos servidores proxies deben ser desplegados para garantizar redundancia como se muestra en la Figura 15. En este despliegue, si un servidor proxy falla, el otro toma el control.

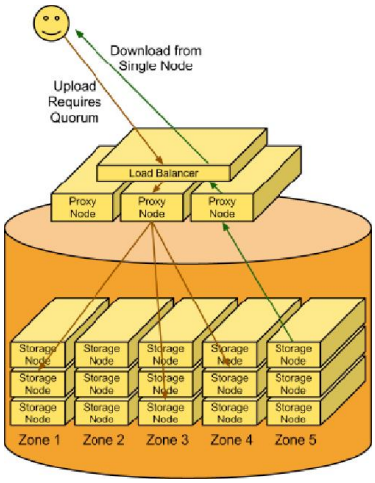


Figura 15. Arquitectura y componentes de Swift

En Swift, cada cuenta o contenedor constituye una base de datos SQLite individual que se encuentra distribuida a lo largo del clúster. Una cuenta es una base de datos SQLite que contiene una lista de los contenedores que pertenecen a esa cuenta. Un contenedor, por su parte, es otra base de datos SQLite que contiene un conjunto de objetos. Para mantener un registro de las ubicaciones de los objetos en el clúster, cada cuenta en el sistema contiene referencias a todos sus contenedores y cada contenedor contiene referencias a todos sus objetos.

Como se muestra en la Figura 16, una partición es una colección de datos almacenados que incluye cuentas, contenedores y objetos. Las particiones constituyen el núcleo del sistema de replicación. Esto significa que por defecto, tres copias de cada partición son creadas y distribuidas a lo largo de un clúster Swift. Los replicadores del sistema y las subidas y descargas de objetos trabajan sobre el concepto de partición. En la medida que el sistema escala, su comportamiento continúa siendo predecible debido a que el número de particiones permanece fijo y siempre es una potencia de 2. En otras palabras, una partición es justamente un directorio en un disco duro con su correspondiente tabla hash de lo que esta contiene. De cada partición se realizan tres réplicas y estas son asignadas a tres discos duros en distintas zonas.

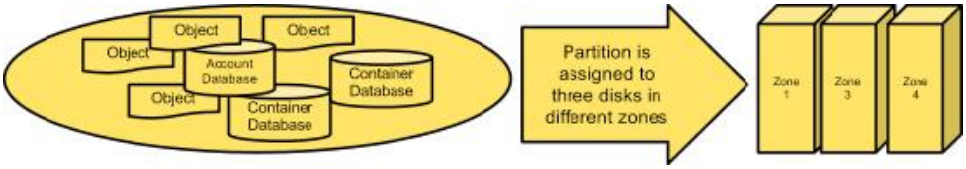


Figura 16. Particiones en Swift

Con el fin de garantizar que siempre haya tres copias de cada partición, los replicadores continuamente examinan cada partición. Para cada partición, el replicador la compara contra las otras dos copias de la misma almacenadas en otras zonas para determinar si existen diferencias. Si los hashes correspondientes al contenido de las réplicas de una misma partición difieren, esto significa que hubo un cambio en una de las copias y es necesario generar estos cambios al resto de las otras dos copias y realizar esto de manera consistente es responsabilidad del replicador.

Un anillo representa un mapeo entre los nombres de las entidades almacenadas en un disco y sus correspondientes ubicaciones físicas. Existen anillos independientes para cuentas, contenedores y objetos. Cuando otros componentes necesitan realizar una operación sobre un objeto, contenedor o cuenta, estos componentes tendrán que interactuar con el anillo apropiado para determinar la ubicación del elemento que desean acceder dentro del clúster. El anillo es utilizado por el servidor proxy y múltiples procesos que se ejecutan en el background como es el caso de los replicadores para determinar la ubicación de objetos, contenedores y cuentas.

Cada partición en un anillo es replicada por defecto tres veces a lo largo del clúster y las ubicaciones de las réplicas de cada partición son almacenadas en la tabla de mapeo mantenida por el anillo.

Las particiones de un anillo son divididas de manera equitativa entre todos los dispositivos que forman parte del clúster. Cuando es necesario mover las particiones, el anillo se asegura que el número de particiones movidas de manera simultánea sea mínimo y que solo una réplica de una partición sea movida en un instante determinado. A los discos duros se le asignan pesos de acuerdo a su capacidad de almacenamiento con el objetivo de balancear la distribución de particiones a lo largo de los discos duros que conforman el clúster. Esto es sumamente útil cuando se cuenta con discos duros diferentes.

La manera en la que Swift determina donde almacenar un nuevo objeto, es decir, en qué disco duro almacenarlo, o de igual forma, dónde ir a buscar un objeto que ya ha sido previamente almacenado es a través de una serie de pasos. Primero, se calcula un hash md5 de la URL del objeto. Posteriormente los primeros n bits del hash son tomados y a partir de estos n bits se determina el número de la partición en la que está almacenada el objeto o donde se debe almacenar el objeto. La forma de calcular este número n es n = log2(Número de particiones), de ahí que el número de particiones deba ser un número fijo y potencia de 2. Una vez que se conoce el número de la partición, la parte restante del procedimiento se reduce a determinar donde reside la partición y es aquí donde el anillo toma el control. Una representación gráfica de lo anterior puede verse en la Figura 17.

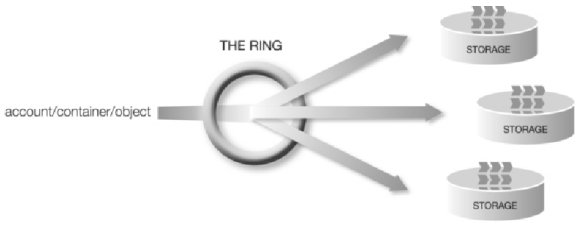


Figura 17. Anillo de Swift

Como se dijo previamente, los anillos contienen un mapeo entre las particiones y la ubicación física donde se almacenas dichas particiones. Formalmente hablando, un anillo es una estructura de datos que contienen tres piezas de información importantes: el número de bits a tomar de un hash md5 con el fin de calcular el número de la partición, una lista de dispositivos de almacenamiento que forman parte del clúster especificando para cada uno su peso, nodo y zona en la que se encuentra, y finalmente y más importante una lista de listas que contiene por cada entrada el número de la

partición y una lista de las posibles ubicaciones físicas donde están almacenadas las tres réplicas de dicha partición. Esta lista de listas es llamada tabla _replica2part2dev.

La table _replica2part2dev es una estructura de datos en forma de tabla bidimensional, de manera que para cada coordenada (número de réplica, número de partición), la tabla indica el dispositivo físico donde se ha almacenado la copia particular de la partición en donde se debe almacenar el objeto o en donde se encuentra almacenado el objeto.

La lista de dispositivos contiene toda la información que Swift necesita para localizar un dispositivo determinado. Esta contiene principalmente la dirección IP del nodo de almacenamiento en el que está ubicado el dispositivo, el puerto TCP que usa y el nombre físico del dispositivo en el nodo de almacenamiento. Por ejemplo de acuerdo a la Figura 18, si el número de la partición es 2, entonces la primera réplica de dicha partición está almacenada en el dispositivo 1 y como se observa en la Figura 19, el dispositivo está contenido en la zona 1 de la región 2 y tiene un peso de 1.

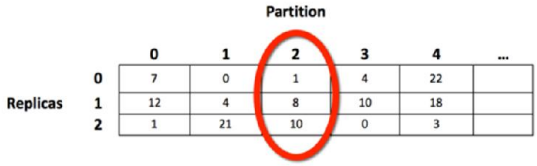


Figura 18. Tabla _replica2part2dev de un anillo Swift

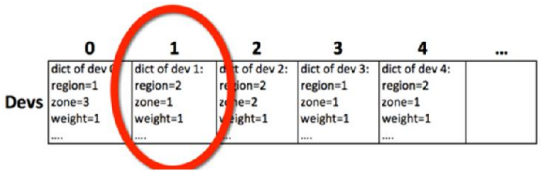


Figura 19. Lista de dispositivos de un anillo Swift

1.8 Interfaz Web de Gestión

El Servicio Web de OpenStack llamado OpenStack Dashboard o Horizon constituye el componente que proporciona el panel de control web de OpenStack y ha sido desarrollado en Django. No incluye todas las funcionalidades de las APIs de cada componente, pero si los métodos más utilizados.

Este panel de control de OpenStack (dashboard) proporciona al administrador y a los usuarios, una interfaz gráfica para acceder, provisionar y automatizar los recursos basado en la Nube. Además es fácil incorporar y presentar productos y servicios para terceros, como son la facturación, la monitorización y las herramientas de gestión adicionales.

Entre las funcionalidades ofrecidas por Horizon se encuentran las siguientes[9]:

- ✓ Permite a los administradores y usuarios controlar su "nivel de computación", almacenamiento y recursos de red.
- ✓ Como administrador, el panel de control ofrece una vista global del tamaño y estado de la nube. Se pueden crear usuarios y proyectos, asignar usuarios a proyectos y configurar limites en los recursos de esos proyectos.
- ✓ El panel de control proporciona a los usuarios un portal "self-service" para provisionar sus propios recursos dentro de los límites establecidos por el administrador.

Capítulo 2. Tecnologías de virtualización de contenedores y procesamiento de Big Data en OpenStack

2.1 Introducción

Hasta hace pocos años, las tecnologías de virtualización predominantes se basaban en el uso de hipervisores. Actualmente estas tecnologías se continúan utilizando, sin embargo es innegable el rotundo éxito y creciente adopción que han tenido las soluciones de virtualización ligera a nivel de sistema operativo o como también se les conoce, las soluciones de virtualización de contenedores. Es precisamente debido a las múltiples ventajas que ofrecen estas tecnologías que han surgido distintas iniciativas para integrarlas con las soluciones de Computación en la Nube existente, precisamente, con OpenStack.

De igual manera se han venido desarrollando y optimizando variadas técnicas y herramientas para el procesamiento de grandes cantidades de datos o como comúnmente se le conoce "Big Data". OpenStack ha incluido un proyecto llamado Sahara que permite agilizar la configuración de un clúster Hadoop para el procesamiento de datos y resulta sumamente interesante como estos clústeres pueden construirse, en vez de con máquinas virtuales basadas en tecnologías de virtualización completa, con contenedores.

En este capítulo se detallan estas tecnologías así como las distintas iniciativas para integrarlas con OpenStack.

2.2 Tecnologías de virtualización ligera a nivel de sistema operativo

Las tecnologías de virtualización basadas en hipervisores se basan en la emulación de hardware. El hipervisor se encarga de emular el hardware de cada máquina virtual, el kernel arranca sobre este hardware emulado y encima de este kernel se ejecutan las aplicaciones, programas y servicios. Dado que un hipervisor o mecanismo de virtualización completa emula el hardware para cada instancia, es posible ejecutar cualquier sistema operativo encima de otro, Windows sobre Linux, o viceversa. Tanto el sistema operativo huésped como el sistema operativo anfitrión se ejecutan con su propio kernel y la comunicación del sistema operativo huésped con el hardware físico se realiza a través de una capa de abstracción del hipervisor como se ilustra en la Figura 20.[22]

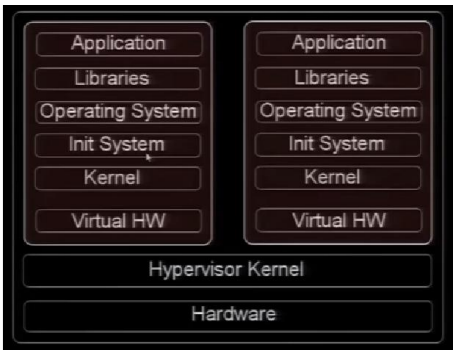


Figura 20. Tecnologías de virtualización completa basada en hipervisores

Este tipo de virtualización proporciona altos niveles de aislamiento y seguridad debido a que toda la comunicación entre la máquina virtual en la que reside el sistema operativo huésped con la máquina física en la que reside el sistema operativo anfitrión se realiza a través del hipervisor. Sin embargo, esta solución de virtualización es usualmente lenta e incurre en una sobrecarga significativa que afecta considerablemente el rendimiento a causa de la emulación del hardware. Para reducir esta sobrecarga se introdujo otra solución de virtualización llamada virtualización a nivel de sistema operativo o virtualización de contenedores que permite ejecutar múltiples instancias aisladas en espacio de usuario sobre el mismo kernel. Esto puede verse claramente en la Figura 21.[22]

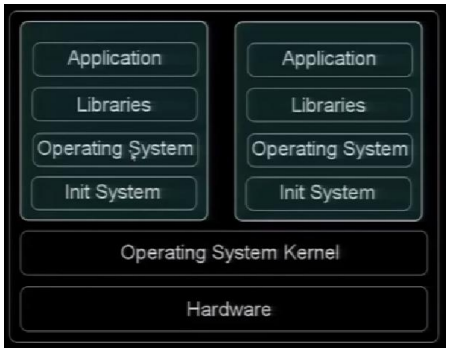


Figura 21. Tecnología de virtualización ligera a nivel de sistema operativo

Utilizando contenedores se pueden empacar muchas más aplicaciones en un servidor físico que si se utilizasen máquinas virtuales. Las máquinas virtuales consumen muchos recursos del sistema. Cada máquina virtual ejecuta no solo una copia completa del sistema operativo, sino una copia virtual de todo el hardware sobre el cual el sistema operativo se ejecuta, lo que implica un alto consumo de memoria RAM y ciclos de CPU. En contraste, todo lo que un contenedor necesita es un sistema operativo, librerías, programas y recursos para ejecutar un servicio en particular. Lo que esto significa es que en la práctica se puede tener dos o tres veces más aplicaciones desplegadas en contenedores en un servidor físico que lo que se podría tener

utilizando máquinas virtuales. Además a partir de usar contenedores se pueden desplegar entornos portables y consistentes para el desarrollo, prueba y despliegue de productos de software.[22]

Cada contenedor consiste en un entorno virtual ligero que agrupa y aísla un conjunto de procesos y recursos como memoria RAM, CPU, espacio de almacenamiento en disco, entre otros, del host anfitrión y del resto de los contenedores. El aislamiento garantiza que cualquier proceso dentro del contenedor no pueda ver ni acceder a ningún proceso o recurso fuera del contenedor.

CGroups o grupos de control y Namespaces o espacios de nombres, constituyen las principales tecnologías incluidas en el kernel de Linux para orquestar y gestionar contenedores. Estas tecnologías son empleadas actualmente por numerosas soluciones de virtualización ligera a nivel de sistema operativo basadas en contenedores tales como OpenVZ, LXC, Rocket y Docker.

CGroups controla los recursos asignados a un grupo de procesos. Estos recursos son CPU, memoria RAM, I/O, y ancho de banda de red. CGroups es utilizado para fijar un límite de los recursos del nodo anfitrión al que un determinado grupo de procesos puede acceder y es de esta manera como se les asignan y restringen los recursos a los contenedores.[22]

Los Namespaces son la tecnología utilizada para garantizar el aislamiento dentro de los contenedores. Cada Namespace separa y aisla un conjunto de recursos de manera tal que estos son solo visibles para los procesos que se ejecutan dentro del Namespace. El tipo de Namespace más conocido es el Namespace de red. Una vez que un dispositivo sea asignado a un Namespace de red, este dispositivo solo puede ser accedido por los procesos que se ejecutan dentro de ese Namespace de red. Existen seis tipos de Namespace dentro del kernel de Linux: Network, UTS (hostname), Mount, IPC, Process ID, User. [22]

Los contenedores ofrecen varias ventajas con respecto a las máquinas virtuales. El hecho de compartir el mismo kernel con el sistema anfitrión les proporciona la ventaja de ser muy rápidos debido a que la sobrecarga introducida a causa de la virtualización es prácticamente nula si se compara con la sobrecarga en escenarios con máquinas virtuales creadas a partir de utilizar tecnologías de virtualización completa. Además, el tiempo de arranque de un contenedor está comprendido en el orden de los milisegundos mientras que el tiempo de arranque de una máquina virtual está en el orden de los segundos. Por lo tanto puede decirse que los contenedores son miles de veces más rápidos que las máquinas virtuales. Por otro lado, el tamaño de una máquina virtual está en el orden de los Gigabytes ya que esta incluye su propio kernel

con su propio sistema operativo, mientras que el tamaño de un contenedor está en el rango de los Megabytes. [18]

Por todo ello, los contenedores pueden migrarse y escalarse a velocidades superiores que las que se pueden alcanzar con máquinas virtuales. Escalar un contener de manera horizontal implica quitarle o asignarle más recursos. Sus tamaños reducidos favorecen su transferencia de un nodo a otro en caso de que sea necesario migrarlos.[18]

Sin embargo, compartir el kernel también tiene desventajas en el caso de los contenedores. Una de ellas consiste en que el tipo de contenedores que puede instalarse en el nodo anfitrión debe soportar el kernel del nodo anfitrión, es decir, no se puede instalar un contenedor con Windows en un nodo anfitrión que tiene instalado Linux o viceversa. [23]

Otra desventaja está relacionada con el aislamiento y la seguridad. El aislamiento entre los contenedores y el nodo anfitrión no es tan fuerte como el aislamiento que se logra entre máquinas virtuales en las soluciones de virtualización completa basadas en hipervisores. Esto se debe a que los contenedores comparten el mismo kernel, y han habido casos donde un proceso en el contenedor ha conseguido infiltrarse en el espacio del kernel del nodo anfitrión y llevar a cabo operaciones maliciosas que han comprometido al nodo anfitrión y al resto de los contenedores.[23]

Para entender mejor como la seguridad se puede ver comprometida en el caso de los contenedores, tomemos el ejemplo de Docker. Docker utiliza libcontainers como tecnología de contenedores. Libcontainers accede a cinco namespaces: Process, Network, Mount, Hostname y Shared Memory para trabajar con Linux. Sin embargo existen muchos otros subsistemas del kernel de Linux fuera del contenedor. Esto incluye a todos los dispositivos, SELinux, cgroups y todos los ficheros montados debajo de /sys. Por lo que si un usuario o aplicación tiene permisos de superusuario dentro del contenedor, el sistema operativo subyacente pudiera, en teoría, verse comprometido.[23]

No obstante, hay formas de proteger a los contenedores Docker así como a los contenedores creados con otras tecnologías de virtualización ligera. Por ejemplo, se puede montar el sistema de ficheros debajo de /sys asignando permisos de solo lectura, se puede forzar a los procesos ejecutándose en el contenedor a escribir solamente en sistemas de ficheros específicos para contenedores y configurar el namespace de red de manera que solo se conecte con una subred privada específica. Sin embargo ninguna de estas medidas de seguridad vienen implementadas por defecto en las soluciones basadas en contenedores y toma tiempo y esfuerzo asegurar estos contenedores.[23]

La regla básica a seguir para gestionar contenedores es tratarlos de la misma manera que se tratan a las aplicaciones de servidor: eliminar los permisos tan pronto como sea posible, ejecutar los servicios como usuarios sin permisos de administración (non-root) cuando sea posible, tratar al root dentro de los contenedores como si fuese root fuera del contenedor.

Otra amenaza de seguridad ligada al uso de contenedores es la tendencia a distribuir aplicaciones empaquetadas en contenedores y el hecho de que estos contenedores, además de contener la aplicación de interés, pueden portar algún malware que en caso de no aplicar las medidas de seguridad requeridas, pudiera infectar el servidor físico sobre el cual se despliegua el contenedor.

Los dos principales casos de uso en los que son empleados los contenedores son como Contenedores de Sistema Operativo o como Contenedores de Aplicaciones. Existen otros casos de uso en los que los contenedores son configurados para funcionar como enrutadores sin embargo esta no es la aplicación más frecuente.

Los Contenedores de Sistema Operativo pueden considerarse como máquinas virtuales ligeras donde pueden instalarse, configurarse y ejecutarse múltiples aplicaciones, librerías, frameworks, etcétera. Como en cualquier máquina virtual, cualquier aplicación ejecutándose dentro de un contenedor solo tendrá acceso a los recursos asignados a este contenedor.[24]

Los Contenedores de Sistema Operativo son útiles cuando se desea ejecutar una flota de distribuciones idénticas o diferentes pero que comparten el mismo kernel como se observa en la Figura 22. En la mayoría de los casos, los contenedores se crean a partir de plantillas o imágenes que determinan la estructura y el contenido del contenedor permitiendo de esta manera crear contenedores que tienen el mismo entorno y las mismas versiones de paquetes instalados e igualmente configurados.[24]

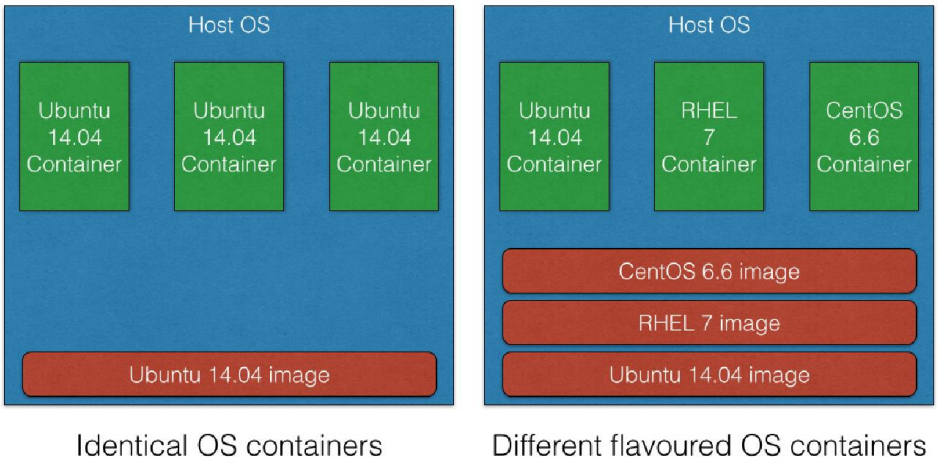


Figura 22. Contenedores de Sistema Operativo

Mientras que los Contenedores de Sistema Operativo están pensados para ejecutar múltiples procesos y servicios, los Contenedores de Aplicaciones están diseñados para empaquetar y ejecutar un único servicio o aplicación. Tecnologías de contenedores tales como Docker y Rocket[10] son ejemplos fehacientes de ello. Por ejemplo, cuando se crea e inicializa un contenedor Docker, este ejecuta un único proceso. Docker responde a la filosofía de crear contenedores por aplicación. Esto es muy diferente de los contenedores de Sistema Operativo tradicionales donde se pueden instalar y ejecutar múltiples aplicaciones y servicios a la vez en un contenedor.[24]

La arquitectura interna de un contenedor Docker está compuesta por múltiples capas de software como puede observarse en la Figura 23.[24]

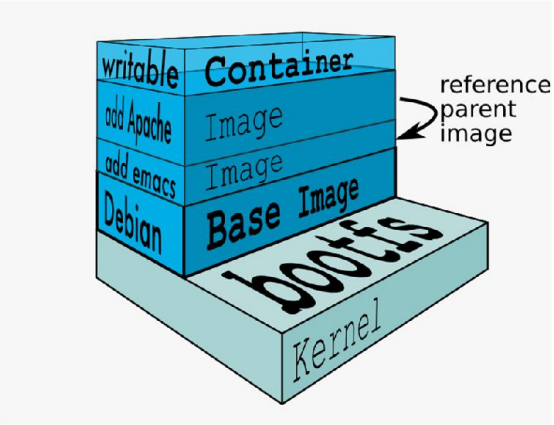


Figura 23. Arquitectura interna por capas de Docker

Cualquier comando RUN especificado en el fichero Dockerfile crea una nueva capa en el contenedor Docker. Al final, cuando se ejecuta el contenedor, Docker se encarga de combinar estas capas de software. Esta estructura de capas ayuda a reducir la duplicación y a incrementar la reutilización de componentes y es especialmente útil cuando se desean construir múltiples contenedores, ya que se puede comenzar a partir de una imagen base que es común para todas las aplicaciones y comenzar a añadir capas que son específicas para poder ejecutar la aplicación que se desea que corra en el contenedor. Otra ventaja de la estructura en capas es la capacidad de revertir los cambios pues se puede retornar a capas viejas y desechar las nuevas capas añadidas sin que esto implique grandes sobrecargas o detrimento del rendimiento.[24]

La idea detrás de los Contenedores de Aplicaciones es poder crear diferentes contenedores para cada componente de un sistema que se desea desplegar. Esto es especialmente útil cuando se desea desplegar un sistema multi-componente distribuido utilizando una arquitectura micro-servicio. El equipo de desarrollo tiene la libertad de empaquetar sus propias aplicaciones haciendo uso de contenedores únicos

[10] http://rocket.readthedocs.org/en/latest/

y desplegables. El equipo de operaciones, por su parte, puede desplegar cada contenedor en el sistema operativo de su elección y escalar las diferentes aplicaciones tanto de manera horizontal como vertical. El estado final es un sistema compuesto por múltiples aplicaciones y servicios, cada uno ejecutándose en un contenedor y comunicándose con los otros contenedores donde se encuentran el resto de las aplicaciones y servicios del sistema a través de APIs y protocolos que cada contenedor soporta.

Con el fin de comprender lo que significa ejecutar contenedores de aplicaciones utilizando Docker, consideremos el despliegue de una arquitectura de tres capas para desarrollo web que incluye una capa de datos con PostgreSQL, una capa de aplicación con Node.js y una capa de balanceo de carga con Nginx. En un despliegue tradicional, se instalaría la base de datos, la aplicación Node.js y Nginx en una misma máquina virtual. Sin embargo, si se utiliza Docker para desplegar esta arquitectura, entonces es necesario construir una imagen de contenedor Docker para cada capa y posteriormente se procede a desplegar estas imágenes de manera independiente, creando contenedores que varían en tamaño y capacidad de acuerdo a la demanda de recursos de cada capa. Este despliegue utilizando contenedores Docker aparece ilustrado en la Figura 24.[24]

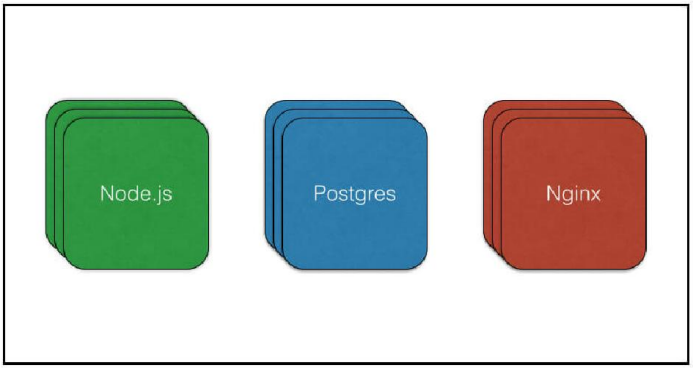


Figura 24. Arquitectura 3-capas para desarrollo web con contenedores Docker

No obstante, hay que tener en mente que a pesar que dividir el despliegue de un sistema en componentes funcionales discretos, esto implica una mayor complejidad a la hora de gestionar las distintas partes.

En general, si se desea empaquetar y distribuir aplicaciones en forma de componentes, las soluciones de contenedores de aplicaciones como Docker son la mejor opción a utilizar. Sin embargo, si se desea un sistema operativo en el cual instalar múltiples aplicaciones, bases de datos y librerías, se recomienda utilizar contenedores de sistemas operativos.

2.3 Docker y su integración con OpenStack

Existe una variedad de herramientas que pueden emplearse para integrar Docker en OpenStack. Entre ellas se encuentran un plugin para el proyecto Heat, un driver para el proyecto Nova (nova-docker), el proyecto Magnum que propone ofrecer Contenedores como Servicio, así como el proyecto Murano integrado con Kubernetes[11].

2.3.1 Plugin de Heat para Docker

Heat constituye un motor de orquestación que permite desplegar múltiples aplicaciones de Nube compuestas a partir de plantillas. Las plantillas Heat contienen una descripción del sistema que se desea desplegar. Heat posee un plugin para Docker, que le permite comunicarse de manera directa con la API de Docker desde una plantilla Heat como se observa en la Figura 25. [25]

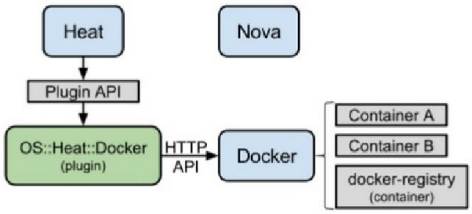


Figura 25. Orquestación de Docker en OpenStack utilizando Heat

Mediante el uso de plantillas, como es el caso de las plantillas con formato HOT, Heat puede unificar los recursos Nova y Docker y desplegar contenedores encima de máquinas virtuales o instancias bare-metal como aparece descrito en la Figura 26.[26]

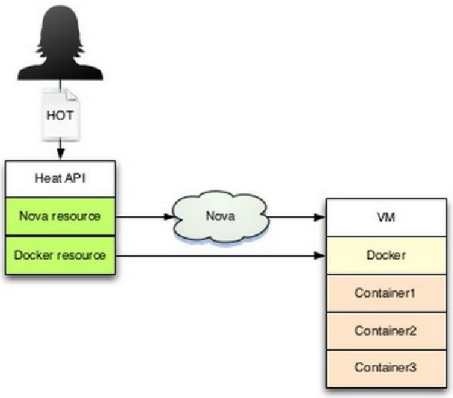


Figura 26. Gestion de recursos Docker y Nova en Heat

En la plantilla, pueden declararse uno o múltiples contenedores que se ejecutarán encima de la máquina virtual o instancia bare-metal. Como puede observarse en la Figura 27, para añadir más contenedores solo basta con añadir más secciones my_docker_container. Heat permite que los contenedores Docker se conecten (linking

[11] http://kubernetes.io/

containers), funcionalidad no disponible si se utiliza la próxima solución de orquestación de Docker en OpenStack y que consiste en utilizar un driver de Nova llamado nova-docker.[26]

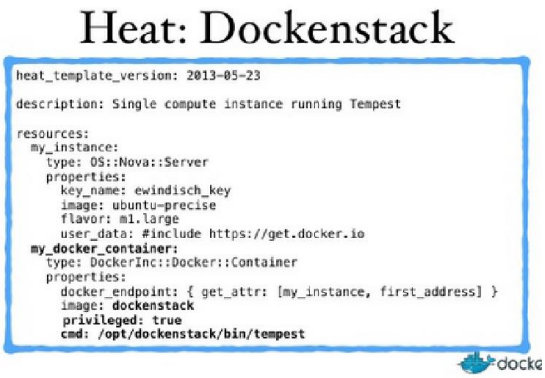


Figura 27. Ejemplo de plantilla Heat para crear contenedores Docker en OpenStack

2.3.2 Driver de Nova para Docker (nova-docker)

Cuando OpenStack lanzó su versión Havana, el Servicio de Computación Nova incluyó un driver para Docker llamado nova-docker. Nova permite desplegar contenedores Docker en vez de máquinas virtuales utilizando la misma API de Nova y haciendo uso de este driver, por lo que Nova trata a Docker como una especie de hipervisor con el cual se comunica mediante este driver. Al mismo tiempo se sigue teniendo acceso a la API de Docker para gestionar dichos contenedores. Lo anterior puede observarse en la siguiente Figura 28.[26]

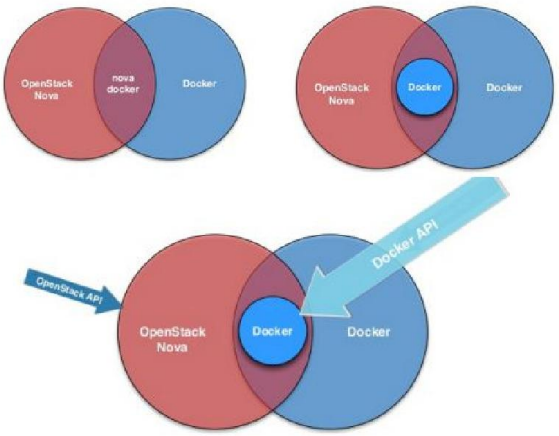


Figura 28. Integración de Nova con Docker en OpenStack

Como puede apreciarse en la Figura 29, el driver de Nova para Docker (nova-docker) tiene embebido un cliente ligero HTTP que se comunica con la interfaz Rest API interna de Docker mediante un socket de Unix. Este driver utiliza la interfaz HTTP API para controlar los contenedores y obtener información acerca de ellos.

El driver obtiene las imágenes para los contenedores del Servicio de Imágenes de OpenStack (Glance) y las carga en el sistema de ficheros de Docker. Las imágenes son cargadas en Glance a partir de exportarlas utilizando el comando "docker save".[26]

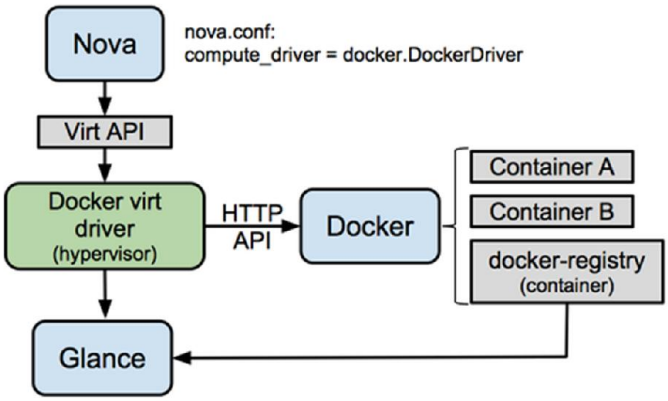


Figura 29. Arquitectura del driver de Nova para Docker

El driver nova-docker permite múltiples operaciones sobre los contenedores tales como: crear un contenedor, terminarlo, reinicializarlo, acceder a su consola serie, crear un snapshot del contenedor, soporte para almacenar imágenes con las cuales crear contenedores en Glance, soporte para proporcionar conectividad de red a los contenedores mediante Neutron, así como detener y reanudar contenedores.

Sin embargo, Nova no soporta todas las funcionalidades incluidas en Docker como solución de virtualización independiente, es decir, Nova no soporta la conexión entre las redes de los contenedores Docker (linking containers), la creación de volúmenes Docker, la compartición de volúmenes Docker entre los contenedores, pasarle variables de entorno a los contenedores y especificarle su directorio de trabajo, entre otras limitantes. Nova funciona como una capa de abstracción para máquinas virtuales, no como una capa de abstracción para procesos.

Por su parte Docker igualmente presenta limitantes que impiden su compatibilidad completa con todas las funcionalidades de Nova. Entre estas limitantes figuran el hecho de que Docker no soporta montar dispositivos, la migración en caliente, levantar un contenedor a partir de un dispositivo de almacenamiento en bloque y no incluye soporte nativo para Glance.

Es por ello que el driver nova-docker ofrece un subconjunto de las funcionalidades de Nova y un subconjunto de las funcionalidades de Docker, resultando esto suficiente incluso para poder ejecutar Docker dentro de un contenedor Docker creado con este driver como se ilustra en la Figura 30, lo que se conoce como Docker-en-Docker y mantener la mayor parte de los beneficios de altas prestaciones de Docker. Kubernetes,

Heat, Mesos, Magnum y CloudFoundry son algunas de las herramientas actualmente disponibles para orquestar y utilizar contenedores. [26]

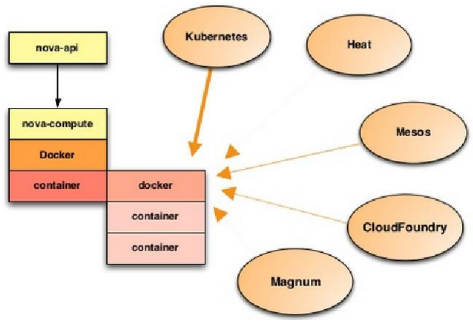


Figura 30. Docker-en-Docker con el driver nova-docker

La ventaja de integrar a Docker con OpenStack mediante el uso de este driver es que Nova permite configurar una arquitectura híbrida en la que se puedan desplegar contenedores Docker, máquinas virtuales utilizando algunos de los hipervisores antes mencionados como KVM, Xen, Hyper-V, entre otros, así como instancias bare-metal utilizando el proyecto Ironic. Esto se puede observar claramente en la Figura 31.[26]

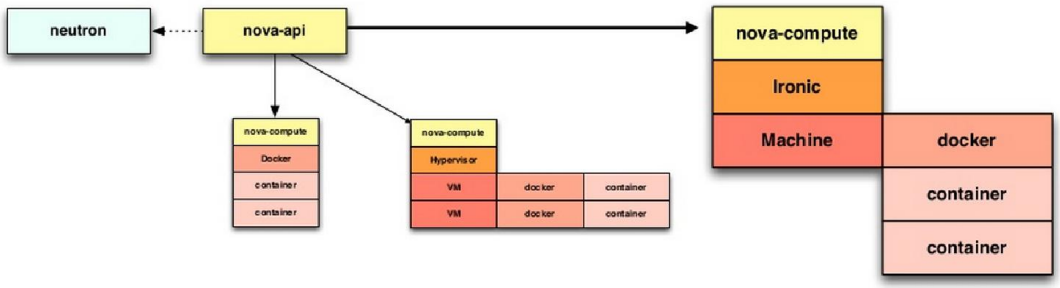


Figura 31. Arquitectura híbrida de Nova

En la versión IceHouse de OpenStack, se decidió retirar este driver del árbol principal de desarrollo de Nova y así ha permanecido en las versiones posteriores de Juno y Kilo. No obstante, esto no ha impedido que el driver continúe desarrollándose y llegue a su madurez y se espera que sea incorporado nuevamente en futuras versiones de OpenStack. Tampoco constituye esto un impedimento para instalar el driver y ponerlo a funcionar junto con Nova y Docker. Actualmente el driver se encuentra disponible en Github en el repositorio stackforge/nova-docker, donde aparecen los pasos a seguir para instalarlo.

2.3.3 Proyecto Magnum (Container-as-a-Service)

Magnum es un servicio de OpenStack con soporte para entornos multi-tenant, accesible mediante una API y desarrollado por el OpenStack Container Team con el fin de hacer que motores de orquestación de contenedores tales como Docker y

Kubernetes se integren con OpenStack como recursos de primera clase y ofrezcan un servicio de contenedores. Magnum no es ninguna tecnología de contenedores nueva, sino un proyecto recientemente lanzado en enero de 2015 que propone la filosofía de ofrecer Contenedores como Servicio a partir de utilizar el proyecto OpenStack Heat para orquestar imágenes minimalistas de sistemas operativos tales como Fedora Atomic, CoreOS o Ubuntu Snappy que contengan Docker y Kubernetes instalados y ejecutar dichas imágenes tanto en máquinas virtuales Nova como en instancias Ironic bare-metal que forman parte de un clúster de nodos. La Figura 32 ilustra la arquitectura interna de Magnum y como este se integra con el resto de proyectos de OpenStack.[27]

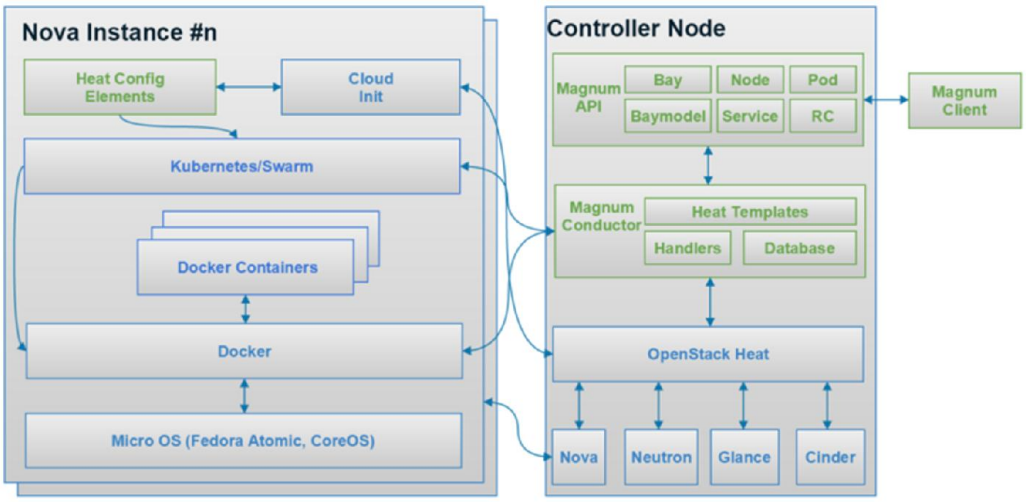


Figura 32. Arquitectura interna de OpenStack Magnum

Ninguna de las tecnologías disponibles para proporcionar contenedores en OpenStack (el driver libvirt para contenedores LXC, el driver nova-docker para proporcionar contenedores Docker a través de Nova y el plugin de Heat para desplegar contenedores Docker en máquinas virtuales) soportan funcionalidades de planificación y orquestación que permitan controlar lo que realmente pasa en los procesos que se ejecutan dentro de los contenedores. Estas funciones de planificación y orquestación sobrepasan los propósitos para los cuales la API de Nova fue diseñada y construida. Heat dio un paso de avance en este aspecto pero no lo suficientemente grande como para garantizar estas funcionalidades pues en Heat no hay ningún concepto de clúster de nodos que ejecuten contenedores.[28]

Existe un solapamiento ligero entre los requisitos en cuanto a funcionalidades para gestionar máquinas virtuales con Nova y los requisitos para gestionar contenedores. La realidad es que los contenedores presentan un ciclo de vida diferente y presuponen el soporte de ciertas funcionalidades que no son compatibles con las ofrecidas por la API de Nova. Estas diferencias surgen debido a que Nova es un proyecto para gestionar máquinas virtuales y no procesos ejecutándose en contenedores. Lo anterior puede

verse claramente en la Figura 33. Por lo tanto, la propuesta de Magnum es ofrecer una API para la gestión específica de contenedores que soporte las operaciones que se esperan poder realizar sobre los mismos y que ahora no son ofrecidas por la API de Nova.[28]

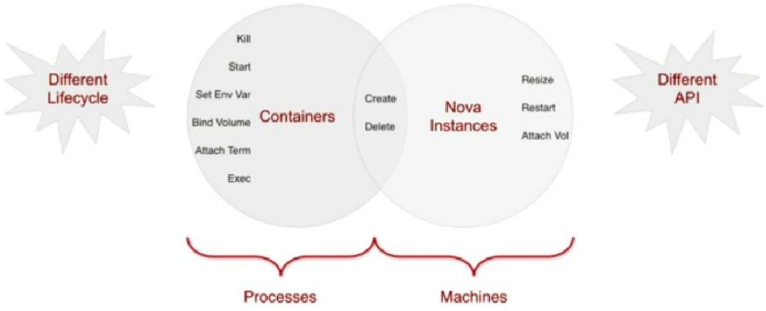


Figura 33. Diferencias en cuanto a requerimientos funcionales para la gestión de contenedores y máquinas virtuales

Como se muestra en la Figura 34, Magnum se presenta como la combinación de múltiples tecnologías entre las que figuran OpenStack, Kubernetes, Flannel y Docker Swarm.

Figura 34. Integración de tecnologías en Magnum

La API de Magnum está compuesta por los siguientes recursos[28]:

- ✓ Node/Nodo: instancia bare-metal proporcionada por el proyecto Ironic o máquina virtual proporcionada por el proyecto Nova donde se planifica la ejecución de trabajos.
- ✓ Bay/Bahía: colección o clúster de nodos.
- ✓ Bay Model/ Modelo de Bahía: objeto que almacena información en formato de plantillas acerca de una Bay y que es utilizada para crear nuevas Bays de manera consistente.
- ✓ Pod/Cápsula: colección de contenedores ejecutándose en una máquina física o virtual y que trabajan juntos de manera muy cercana comunicándose entre sí.
- ✓ Service/Servicio: abstracción que define un conjunto lógico de pods y una política mediante la cual acceder a ellos.
- ✓ Replication Controller/Controlador de Replicación: abstracción para gestionar un grupo de Pods y garantizar que un determinado número de Pods se estén ejecutando apropiadamente.

✓ Container/Contenedor: un contenedor Docker.

Magnum permite escoger el motor de orquestación de contenedores a utilizar. Las opciones actuales son Kubernetes y Docker Swarm. Ambos son soluciones de código abierto y cuentan con una comunidad colaborativa de desarrolladores. Como resultado se pueden tener dos tipos de Bays: un cluster o Bay Kubernetes o un cluster o Bay Docker Swarm. En un clúster o Bay Docker Swarm se cuenta con nodos y contenedores. En un clúster o Bay Kubernetes se cuenta con nodos, pods, contenedores, servicios y controladores de replicación. Esto se puede observar en la Figura 35.[28]

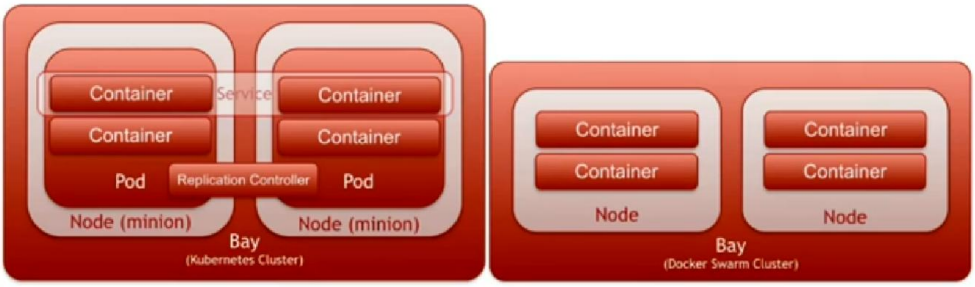


Figura 35. Recursos de Magnum

Magnum se diferencia del resto de los sistemas de orquestación de contenedores por el hecho de que este se integra con los servicios de infraestructura de OpenStack y evita re-implementar soluciones a problemas que ya han sido proporcionadas por estos servicios. Entre los servicios con los cuales Magnum se integra se encuentran Keystone (Identidad), Heat (Orquestación), Nova (Computación), Glance (Imagen), Cinder (Almacenamiento en Bloques) y Neutron (Red). Actualmente los contenedores proporcionados por Magnum se ejecutan en nodos que no son más que instancias de máquinas virtuales Nova. No obstante se espera que en un futuro cercano, Magnum permita que los nodos puedan ser también instancias bare-metal proporcionadas por Ironic o incluso contenedores dentro de los cuales se puedan ejecutar contenedores y Bays dado que las sobrecargas debidas a la anidación de contenedores (nesting containers) no tiene las mismas penalizaciones en cuanto a rendimiento que la anidación en el caso de la virtualización completa (nested virtualization).[28]

OpenStack es un sistema de software multi-tenant, lo que significa que es capaz de proporcionar una vista de acceso controlado a cada tenant o proyecto de manera individual, de manera que este solo pueda ver y acceder a los recursos de Nube que ha creado o que le han sido asignados. Hasta el lanzamiento de Magnum, todos los sistemas de gestión de contenedores existentes no incluían soporte para entornos multi-tenant, lo que significa que cualquier tenant con acceso al sistema tiene acceso

absoluto a todos los recursos del sistema incluso a aquellos que no han sido creados por él o que no les han sido asignados. Magnum incluye soporte para entornos multi-clientes a partir de reutilizar funcionalidades incluidas en OpenStack para estos fines y las extiende desde las capas inferiores hasta las superiores garantizando que cada capa en el sistema está apropiadamente aislada. Esto significa que Magnum se puede utilizar para proporcionar contenedores en una nube pública construida con OpenStack con los mismos niveles de aislamiento que las máquinas virtuales le ofrecen a los diferentes tenants.

2.3.4 Proyecto Murano

El proyecto Murano perteneciente a OpenStack ofrece un catálogo de aplicaciones que le permite a los desarrolladores y administradores publicar variadas aplicaciones listas para ser desplegadas en entornos de nube en un catálogo web que agrupa dichas aplicaciones por categorías. Este catálogo de aplicaciones ofrecido por Murano puede ser utilizado por los usuarios de la nube, incluyendo a los más expertos, para seleccionar las aplicaciones y servicios que necesitan desplegar y de esta manera confiar en las funcionalidades subyacentes de Murano para desplegar entornos confiables que incluyan estas aplicaciones y servicios sin tener que lidiar con los trabajos de configuración de las aplicaciones y servicios y el aprovisionamiento de recursos virtuales de computación, almacenamiento y red sobre los que estas aplicaciones y servicios se ejecutarán.[29]

El objetivo principal es ofrecer una interfaz de usuario (UI) o API que permita componer y desplegar entornos compuestos a un nivel de abstracción de aplicación y posteriormente brindar la posibilidad de gestionar el ciclo de vida de este entorno.

Murano contiene una interfaz de usuario muy simpl, integrada al dashboard web standard de OpenStack, es decir, integrada al proyecto Horizon. La interfaz de usuario con su catálogo de aplicaciones le permite a los usuarios encontrar las aplicaciones que necesitan desplegar a partir de realizar búsquedas por categorías de aplicaciones, etiquetas u otros atributos. Por ejemplo, Apache Tomcat o un Java Servlet Container pueden buscarse a partir de las etiquetas: application service, java o servlet.[29]

Una vez que el usuario encuentra la aplicación o servicio que desea, es capaz de desplegarla inmediatamente o añadirla a un entorno Murano a partir de definir una aplicación multi-capa como es el caso de un stack LAMP para desarrollo web. En el caso de que no se desee usar la interfaz de usuario de Horizon, existe la opción de utilizar el servicio de una API REST HTTP que permite crear y utilizar entornos Linux y Windows, definir servicios dentro de ellos e iniciar el despliegue. Cualquier servicio ofrecido por terceros puede utilizar Murano para automáticamente desplegar y dinámicamente actualizar servicios bajo demanda.

Una vez seleccionado los servicios y configurado sus propiedades, solo basta presionar un botón y Murano automáticamente se encargará del resto del trabajo que incluye las siguientes tareas[29]:

- ✓ Aprovisionamiento de máquinas virtuales en OpenStack, garantizar la conectividad de red de dichas instancias y la asignación de volúmenes de almacenamiento permanente a dichas instancias.
- ✓ Instalación de distribuciones de sistemas operativos Linux o Windows en dichas instancias.
- ✓ Despliegue de los servicios requeridos.
- ✓ Realizar todas las configuraciones y cambios requeridos para que los servicios y aplicaciones sean accesibles por usuarios que ordenaron su despliegue.

Murano se encarga de gestionar las aplicaciones mientras que Heat se ocupa de orquestar la infraestructura. De hecho, Murano utiliza una capa de orquestación de infraestructura basada en Heat para aprovisionar los recursos subyacentes para poder desplegar las aplicaciones y servicios. Estos recursos subyacentes incluyen máquinas virtuales Nova, conectividad de red para dichas instancias a partir de solicitar los servicios de Neutron y volúmenes de almacenamiento en bloque proporcionados por Cinder. Con la versión 0.6 de Murano incluida en la versión Juno de OpenStack, es posible subir aplicaciones escritas en plantillas con formato HOT (Heat Orchestration Template) al catálogo de aplicaciones de Murano. Este formato es soportado por la herramienta CLI de Murano, la cual automáticamente genera un paquete de aplicaciones Murano apropiado a partir de la plantilla HOT proporcionada por el usuario.[27, 30]

Murano está compuesto de los siguientes repositorios de código fuente[30]:

- ✓ murano: es el repositorio principal. Contiene código para el servidor API de Murano, el motor Murano y MuranoPL.
- ✓ murano-agent: agente que se ejecuta en máquinas virtuales huéspedes y ejecuta un plan de despliegue.
- ✓ murano-dashboard: interfaz de usuario de Murano implementada como un plugin para OpenStack Dashboard (Horizon)
- ✓ python-muranoclient: librería cliente y CLI cliente para Murano

Murano utiliza otros servicios de OpenStack con el fin de evitar re-implementar las funcionalidades existentes. Se encarga de interactuar con estos servicios a partir de utilizar sus correspondientes API REST a través de sus clientes Python. Los servicios externas más utilizados son Keystone y Heat.

Heat es utilizado para orquestrar la infraestructura de recursos tales como máquinas virtuales, volúmenes y redes que interconectan a las máquinas virtuales. Sobre la base de la definición de las aplicaciones y servicios a desplegar, Murano crea automáticamente las plantillas para Heat. Todas las operaciones remotas a ejecutar en las máquinas virtuales de los usuarios se realizan a través de la cola AMQP y son ejecutadas por el murano-agent. Ejemplo de dichas operaciones son la instalación y configuración de software en el servidor. No obstante cabe notar en la Figura 36 que la comunicación puede ser fácilmente configurada en una instancia separada de AMQP para asegurar que la infraestructura y las máquinas virtuales están aislados. La arquitectura y principales componentes de Murano pueden observarse en la Figura 36.[30]

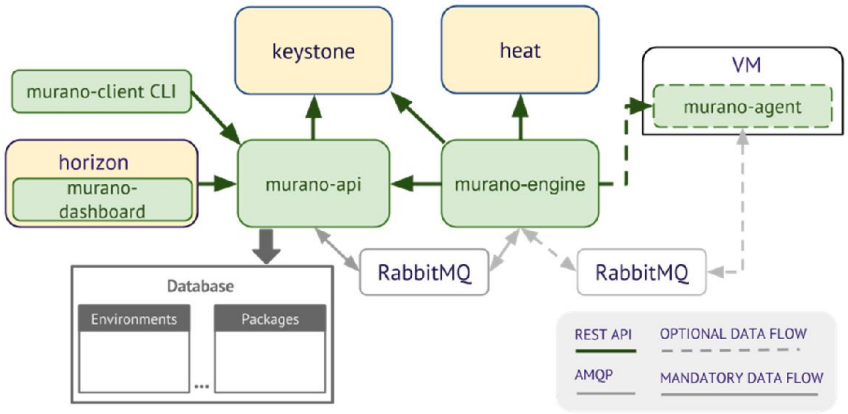


Figura 36. Arquitectura y componentes de Murano

Murano y su integración con Kubernetes y Docker

Murano proporciona un paquete de aplicación para Kubernetes el cual ofrece una capa de abstracción para Kubernetes y Pods. Los desarrolladores pueden empaquetar sus aplicaciones para usarlas como normalmente lo harían, de manera sencilla añadiéndolas a un clúster Kubernetes.

El paquete Kubernetes ofrecido por Murano soporta las mismas funcionalidades que se esperan encontrar en un despliegue típico de Kubernetes, tales como la creación de Pods que implementan contenedores Docker, la monitorización de la disponibilidad y la carga de los Pods, y la posibilidad de escalar los Pods sobre la base de la configuración de Kubernetes. Igualmente coordina la conectividad entre los Pods y la infraestructura subyacente.

Murano gestiona y orquesta la infraestructura subyacente que consiste en recursos de OpenStack. Se ocupa de configurar la red virtual para Kubernetes y los Pods, y utiliza el Servicio de Orquestación de OpenStack (Heat) con el fin de aprovisionar los recursos

que Kubernetes necesita, como por ejemplo máquinas virtuales Nova, conectividad de red a partir de conectar las interfaces de red virtuales de dichas máquinas a redes virtuales creadas y configuradas con Neutron, la configuración de grupos de seguridad y la configuración de los routers igualmente a partir de solicitar estos servicios a Neutron, y la creación y asignación de volúmenes de almacenamiento Cinder a las instancias Nova.

En Kubernetes, los contenedores son agrupados en Pods y Kubernetes escala los Pods dentro de un clúster a partir de crear e inicializar contenedores en nodos Kubernetes. En el caso de que se cuente con múltiples nodos, Kubernetes distribuye los contenedores entre los nodos.

Cuando la aplicación crece hasta tal punto que el propio clúster Kubernetes necesita escalar, el sistema necesita recibir ayuda desde el exterior, es decir, se requiere de un sistema externo que añada recursos. En este caso ese sistema externo es Murano, que utiliza el Servicio de Telemetría de OpenStack llamado Ceilometer para detectar cuándo es necesario agregar recursos adicionales. Murano añade un nuevo nodo al cluster Kubernetes utilizando la función "add node" y Kubernetes se encarga de distribuir la carga. Igualmente Murano puede inicializar aplicaciones de escalabilidad dentro de un clúster en caso de ser necesario.

2.4 Procesamiento de Big Data en OpenStack

El proyecto OpenStack Sahara se propone proporcionarle a los usuarios medios simples para aprovisionar clústeres Hadoop a partir de especificar múltiples parámetros tales como versión de Hadoop, topología del clúster, detalles del hardware de los nodos que forman parte del clúster, entre otros. Una vez que el usuario proporciona todos estos parámetros, Sahara despliega el clúster en unos minutos. Igualmente Sahara proporciona los medios para escalar clústeres existentes y desplegados a partir de agregar o eliminar nodos bajo demanda. Apache Hadoop constituye un estándar industrial ampliamente adoptado como implementación de MapReduce. El objetivo de Sahara es permitirles a los usuarios desplegar y gestionar de manera sencilla clústeres Hadoop en OpenStack así como Amazon viene haciéndolo desde hace ya algunos años a través del servicio Amazon Elastic MapReduce (EMR).[31]

Sahara abarca los siguientes casos de uso:

- ✓ rápido aprovisionamiento de clústeres Hadoop en OpenStack
- ✓ utilización de capacidad de computación no utilizada para nubes IaaS de propósito general

- ✓ "Analytics as a Service" o Analítica como Servicio para flujos de trabajo de analítica de datos ad-hoc o en ráfagas, de manera similar a AWS EMR.

Las principales características de Sahara son[31]:

- ✓ Está diseñado como un componente de OpenStack
- ✓ Es gestionable a través de una API REST con UI disponible como parte del proyecto OpenStack Dashboard (Horizon).
- ✓ Soporta diferentes distribuciones de Hadoop:
  - Sistema modular basado en plugins que permite la integración con múltiples motores de instalación de Hadoop.
  - Integración con herramientas de gestión específicas de distintos proveedores comerciales tales como Apache Ambari o Cloudera Management Console.
- ✓ Incluye plantillas predefinidas con configuraciones de despliegue de clústeres Hadoop y ofrece la posibilidad de modificar los parámetros de dichas plantillas.

Como se observa en la Figura 37, el proyecto Sahara se comunica con los siguientes componentes de OpenStack[31]:

- ✓ Horizon: proporciona la GUI con las funcionalidades requeridas para que los usuarios puedan acceder a todos los componentes de Sahara.
- ✓ Keystone: autentica a los usuarios y proporciona los tokens de seguridad que son utilizados para acceder al resto de los servicios de OpenStack, por lo tanto limita las habilidades de los usuarios en Sahara de manera que sean compatibles con los permisos que se le han otorgado en OpenStack Keystone.
- ✓ Nova: es utilizado para aprovisionar máquinas virtuales o contenedores para el clúster Hadoop.
- ✓ Heat: Sahara puede ser configurado para utilizar Heat como motor de orquestación de manera que Heat se encargue de orquestar el aprovisionamiento de los servicios y recursos de computación, almacenamiento y red requeridos en el clúster Hadoop.
- ✓ Glance: almacena las imágenes de las máquinas virtuales o contenedores que forman parte del clúster Hadoop. Cada imagen contiene una distribución de sistema operativo y algunas traen también incluido una versión de Hadoop instalada.
- ✓ Swift: utilizado para el almacenamiento de los datos que serán procesados por los Hadoop Jobs y para almacenar los resultados del procesamiento.
- ✓ Cinder: usado para crear volúmenes de almacenamiento en bloque que serán conectado a las instancias.
- ✓ Neutron: utilizado para proporcionar conectividad de red a los nodos del clúster.
- ✓ Ceilometer: utilizado para coleccionar datos acerca del uso del clúster con propósitos de medición, monitoreo y benchmarking.

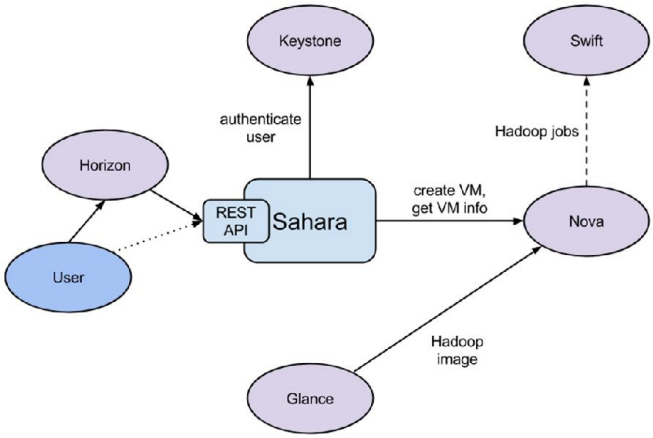


Figura 37. Interacción de Sahara con otros proyectos de OpenStack

Sahara proporciona dos niveles de abstracción para la API y la UI basados en el caso de uso en cuestión para el cual se emplee: aprovisionamiento de clústeres Hadoop o Servicio de Análisis de datos.

Para el aprovisionamiento de un clúster Hadoop el flujo de trabajo genérico incluye los siguientes pasos:

- ✓ Seleccionar la versión de Hadoop
- ✓ Seleccionar la imagen base con o sin Hadoop pre-instalado:
  - Para imágenes base sin una versión de Hadoop pre-instalada, Sahara soporta motores de despliegues integrados con herramientas comerciales de proveedores que pueden añadirse a Sahara mediante su mecanismo modular para desplegar Hadoop sobre los nodos del clúster una vez que este se haya creado.
- ✓ Definir la configuración del clúster, incluyendo tamaño y topología del mismo y especificando los diferentes tipos de parámetros de Hadoop. Para facilitar la configuración de dichos parámetros, Sahara proporciona un mecanismo de plantillas configurables que una vez creadas pueden almacenarse para futuros despliegues o sencillamente modificar aquellos parámetros que difieran para un despliegue de clúster Hadoop específico.
- ✓ Aprovisionar el clúster: Sahara se encargará de ordenar la creación de máquinas virtuales o contenedores y de configurar Hadoop, así como de añadir o eliminar nodos del clúster y de terminar el clúster cuando este ya no sea necesario.

Para el servicio de análisis de datos, el flujo de trabajo genérico incluye los siguientes pasos:

- ✓ Seleccionar una de las versiones predefinidas de Hadoop
- ✓ Configurar el trabajo:
  - Escoger el tipo de trabajo: pig, hive, jar-file, etcétera
  - Proporcionar la fuente del script que contiene el trabajo o la ubicación del fichero jar
  - Seleccionar la ubicación de los datos de entrada y de salida (inicialmente solo incluye soporte para Swift)
  - Seleccionar la ubicación para los logs.
- ✓ Especificar el límite para el tamaño del clúster
- ✓ Ejecutar el trabajo:
  - Todo el aprovisionamiento del clúster y la ejecución del trabajo transcurre de manera trasparente para el usuario
  - El clúster se eliminará automáticamente una vez se termine de ejecutar el trabajo
- ✓ Obtener los resultados del procesamiento (por ejemplo, desde Swift).

Para aprovisionar clústeres Hadoop mediante Sahara, los usuarios operan sobre tres tipos de entidades: "Node Group Templates", "Cluster Templates" y "Clusters".

Un Node Group Template es una plantilla que describe a un grupo de nodos dentro de un clúster. Esta contiene una lista de procesos Hadoop que deben ser desplegados en cada instancia de un mismo grupo. Igualmente un Node Group Template debe proporcionar configuración a nivel de nodos para estos procesos. Este tipo de plantillas encapsula parámetros de hardware conocido en el argot de OpenStack como "flavor" para cada máquina virtual o contenedor que representa un nodo del clúster, así como parámetros de configuración para los procesos Hadoop que se ejecutan en los nodos.

Un Cluster template es otra plantilla que está diseñada con el fin de unificar múltiples Node Group Templates y conformar un Cluster. Un Cluster Template define qué Node Groups serán incluidos y cuántas instancias serán creadas en cada uno. Algunos de los parámetros de configuraciones de Hadoop no pueden ser aplicados a un solo nodo, pero sí sobre un Cluster, por lo tanto los usuarios pueden especificar este tipo de configuración en un Cluster Template. Sahara permite que los usuarios especifiquen los procesos que deben ser añadidos a un grupo anti-affinity dentro de un Cluster Template. Si un proceso está incluido en un grupo anti-affinity, esto significa que las máquinas virtuales/contenedores que conforman a los nodos Hadoop donde este

proceso será desplegado deben ser planificadas y creadas en diferentes Nodos de Computación Nova.

Entre los principales parámetros de configuración de un Cluster se encuentra la imagen que será utilizada para crear los nodos Hadoop que conformarán el clúster. Esta imagen puede tener una versión pre-instalada de Hadoop o puede emplearse un motor de despliegues como Apache Ambari o Cloudera Management Console que se encargará de instalar Hadoop en los nodos una vez que se haya creado el clúster. Los usuarios pueden escoger un Cluster Template pre-configurado para inicializar el clúster.

Sahara incluye un conjunto de restricciones para las topologías de los clústeres Hadoop. Los procesos JobTracker y NameNode pueden ejecutarse en una misma máquina virtual/contenedor o en dos instancias separadas. Igualmente el clúster puede contener nodos "worker" de diferente tipo. Los nodos "worker" pueden ejecutarse tanto como TaskTracker y DataNode, o solamente como uno de estos dos. Sahara permitirá a los usuarios crear clústeres Hadoop con cualquiera de estas combinaciones de opciones, sin embargo no permitirá crear topologías que no se ajusten a estas opciones, por ejemplo, no permitirá crear un conjunto de nodos "worker" que funcionen como DataNodes pero sin ningún nodo que funcione como NameNode.

Cada clúster pertenece a algún tenant. Los usuarios solo tendrán acceso a los objetos localizados en los tenants a los que estos pertenecen. Los usuarios solo pueden editar y/o eliminar los objetos que ellos han creado dentro de un tenant determinado, por lo tanto un usuario no podrá editar ni eliminar un clúster Hadoop creado para un tenant al cual ese usuario no pertenece. Todo ello se logra gracias a la integración de Sahara con Keystone. Naturalmente los usuarios con permiso de administración tendrán acceso completo a cada objeto en OpenStack. De esta manera se lo logra que Sahara cumpla con las políticas de acceso generales de OpenStack.

El servicio Swift es el servicio de OpenStack para almacenamiento de datos en forma de objetos análogo al servicio Amazon S3. Es natural esperar una integración entre Sahara y Swift. Esto se logra a partir de un parche incluido en Sahara que incluye una implementación de sistema de ficheros para Swift de manera que desde Hadoop se pueda cargar y guardar datos en Swift. Este sistema de ficheros recibe el nombre de HADOOP-8545, permite listar puntos de acceso para un objeto, cuenta o contenedor, y hace posible la integración de Swift con software que se basa en información de localidad de los datos para evitar la sobrecarga de la red.

Los path Swift son expresados en Hadoop de acuerdo con la siguiente plantilla: `swift://${container}.${provider}/${object}.` Un ejemplo sería `swift://integration.sahara/temp`, donde `integration.sahara` es el nombre DNS del servicio Swift que contiene al contenedor con el objeto que se desea obtener y `temp` es el nombre del objeto.

La arquitectura de Sahara aparece descrita en la Figura 38 y consta de los siguientes componentes[31]:

- ✓ Componente Auth: responsable de la autenticación y autorización de los clientes y para ello se comunica con Keystone.
- ✓ DAL (Data Access Layer): Capa de Acceso de Datos, persiste los modelos internos en bases de datos.
- ✓ Provisioning Engine: componente responsable de la comunicación con Nova, Heat, Cinder y Glance.
- ✓ Vendor Plugins: meganismo basado en plugins responsable de configurar y desplegar Hadoop en máquinas virtuales/contenedores. Soluciones de gestión existentes tales como Apache Ambari y Cloudera Management Console pueden ser utilizada para estos fines.
- ✓ EDP (Elastic Data Processing): responsable de planificar y gestionar trabajos Hadoop (Hadoop Jobs) en clúster aprovisionados por Sahara.
- ✓ REST API: expone las funcionalidades de Sahara a través de REST.
- ✓ Python Sahara Client: de manera similar a otros componentes de OpenStack, Sahara tiene su propio cliente Python.
- ✓ Sahara pages: GUI para que Sahara sea localizable en OpenStack Horizon.

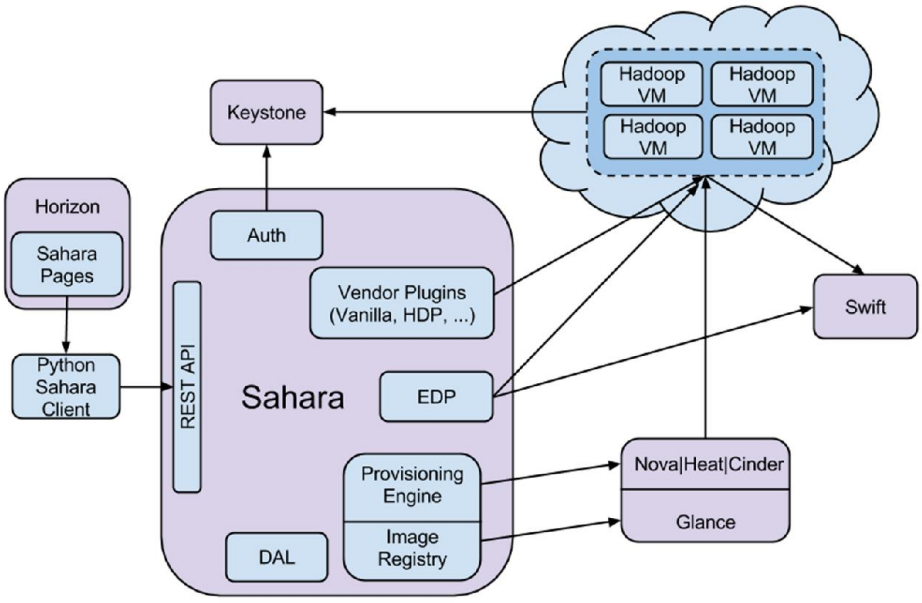


Figura 38. Arquitectura interna de OpenStack Sahara

Un clúster desplegado con Sahara consiste de múltiples Node Groups. Los Node Groups varían en dependencia de su rol, parámetros y número de máquinas virtuales/contenedores que lo integran. La Figura 39 ilustra un ejemplo de clúster Hadoop consistente de tres Node Groups cada uno con diferentes roles o conjunto de procesos ejecutándose.[31]

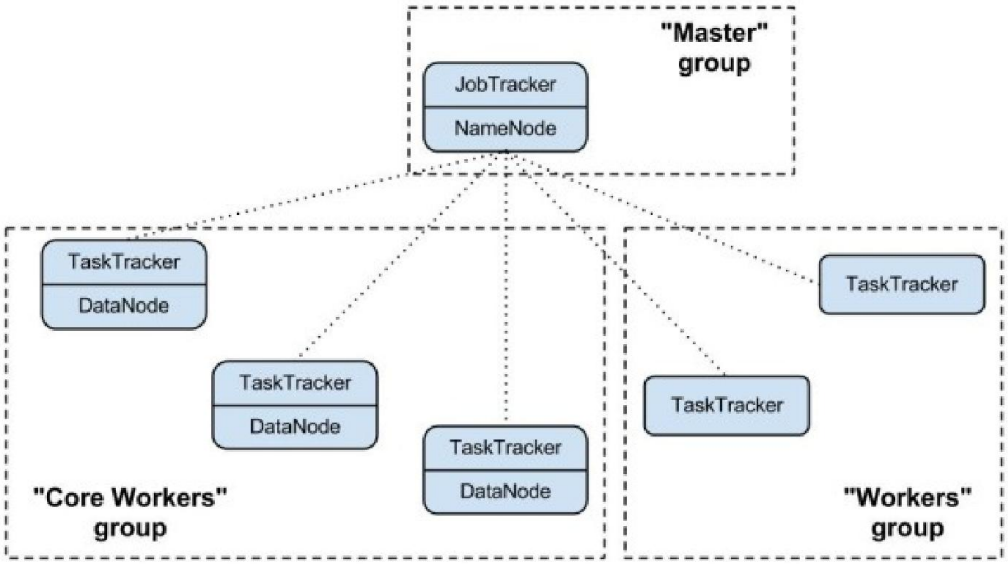


Figura 39. Ejemplo de clúster Hadoop (Map Reduce) desplegado con Sahara

El plugin seleccionado es uno de los componentes responsables de aprovisionar un clúster Hadoop. Generalmente cada plugin es capaz de aprovisionar una distribución específica de Hadoop. Igualmente el plugin pude instalar herramientas adicionales de gestión y monitorización para el clúster. Debido a que los parámetros de Hadoop varían en dependencia de la distribución y versión de Hadoop, las plantillas son específicas para el plugin y para la versión de Hadoop. Una plantilla no puede ser usada para crear un cluster donde las versiones de Hadoop y del plugin son diferentes a las de la plantilla cuando esta fue creada.

Entre los plugins disponibles se encuentran:

- ✓ Vanilla Plugin
- ✓ Hortonworks Data Platform Plugin
- ✓ Spark Plugin
- ✓ Cloudera Plugin
- ✓ MapR Distribution Plugin

Apache Spark es un sistema de computación para la creación de clústeres rápidos y de propósito general. Proporciona APIs de alto nivel para Java, Scala, Python y R, así como un motor optimizado que soporta la ejecución de grafos. Igualmente soporta un amplio conjunto de herramientas de alto nivel incluyendo Spark SQL para SQL y procesamiento de datos estructurado, MLlib para machine learning, GraphX para procesamiento de grafos y Spark Streaming.

Apache Spark es el principal candidato para suceder a MapReduce. Al igual que MapReduce es un motor de propósito general, sin embargo está diseñado para ejecutar muchos más flujos de trabajo y para hacerlo de una manera más rápida y eficiente.

El Spark plugin para Sahara proporciona una vía de desplegar clústeres Apache Spark en OpenStack con tan solo un click y de manera simple y repetible. Actualmente Spark es instalado en modo independiente (standalone mode) sin soporte YARN o Mesos. Para el aprovisionamiento del clúster se deben usar imágenes preparadas. El Spark plugin ha sido desarrollado y probado con imágenes generadas utilizando sahara-image-elements: https://github.com/openstack/sahara-image-elements. Este repositorio es clonado en la máquina que se utilice para construir la imagen y se utiliza el comando tox -e venv -- sahara-image-create para generar una imagen en formato qcow2.

Para seleccionar el plugin se utiliza:

#tox -e venv -- sahara-image-create -p [vanilla|spark|hdp|cloudera|storm|mapr]

Para seleccionar la versión de Hadoop se utiliza:

#tox -e venv -- sahara-image-create -v [1|2|plain]

Para seleccionar el sistema operativo se utiliza:

#tox -e venv -- sahara-image-create -i [ubuntu|fedora|centos]

El formato qcow2 puede cambiarse a vmdk para VMware, o a tar y luego cargarse en Docker para crear una imagen Docker. Con este comando se puede construir una imagen que tiene como sistema operativo a Ubuntu y trae instalado Cloudera CDH5 HDFS y Apache Spark.

OpenStack Sahara requiere que las imágenes que se emplean para crear los contenedores o máquinas virtuales que funcionan como nodos del clúster Hadoop sean cargadas en el Sahara Image Registry. Los requerimientos que Sahara impone a cada imagen dependen del plugin que se va a utilizar y de la versión de Hadoop. Algunos plugin solo requieren una imagen con un sistema operativo instalado y posteriormente

se encargan de instalar Hadoop en la máquina virtual o contenedor. Otros plugins requieren que las imágenes contengan una versión pre-instalada de Hadoop.

El Spark plugin requiere que la imagen sea etiquetada con las etiquetas: “spark” y “<Spark version>” por ejemplo “1.0.0”, cuando sea cargada en el Sahara Image Registry. Igualmente se debe especificar el nombre de usuario del cloud-user por defecto usado en la imagen. Una desventaja que pudiera presentar el Spark plugin es que no soporta la integración con Swift.

El cluster Spark es desplegado utilizando los script disponibles en la distribución de Spark, los cuales permiten iniciar todos los servicios (master y slaves), detener todos los servicios, entre otras funcionalidades. Por tal motivo y a diferencia de los demonios CDH HDFS, Spark no es desplegado como un servicio standard de Ubuntu y si las máquinas virtuales o contenedores son reinicializados, Spark no será reiniciado, sino que habrá que hacerlo de manera manual.

Spark necesita unos pocos parámetros para trabajar y tiene configuraciones por defecto bastante sensatas. Si se necesitase, estas pueden ser modificadas cuando se crea el Cluster Template.

Docker es una tecnología emergente que se ha vuelto muy popular en el mercado en poco tiempo debido a la flexibilidad que ofrece su arquitectura para el despliegue de aplicaciones. OpenStack por su parte, también es otra tecnología de nube popular y se ha vuelto más estable y ha incorporado nuevas funcionalidades en sus últimas distribuciones. Sería muy conveniente integrar Docker con Sahara de manera que Sahara utilizara Heat para orquestar la creación de nodos del clúster Hadoop que son aprovisionados por el driver nova-docker de Nova y de esta manera lograr una mejor utilización de los recursos que la que se obtendría con hipervisores como KVM, Xen, Hyper-V o VMWare ESXi. El rendimiento que se puede lograr a partir de desplegar clústeres Hadoop donde los nodos son contenedores es muy superior al que se podría alcanzar con máquinas virtuales. No obstante, a pesar de todas las ventajas que traería consigo esta solución, cabe destacar que el driver nova-docker aún no tiene implementada la funcionalidad de conectar volúmenes de almacenamiento permanente Cinder a los contenedores Docker y esto pude constituir una limitante que no presentan las máquinas virtuales proporcionadas por las distintas versiones de hipervisores.[32]

La arquitectura lógica de los servicios de OpenStack que intervendrían en un desliegue de Sahara utilizando al Spark plugin y al driver nova-docker se muestra en la Figura 40.[32]

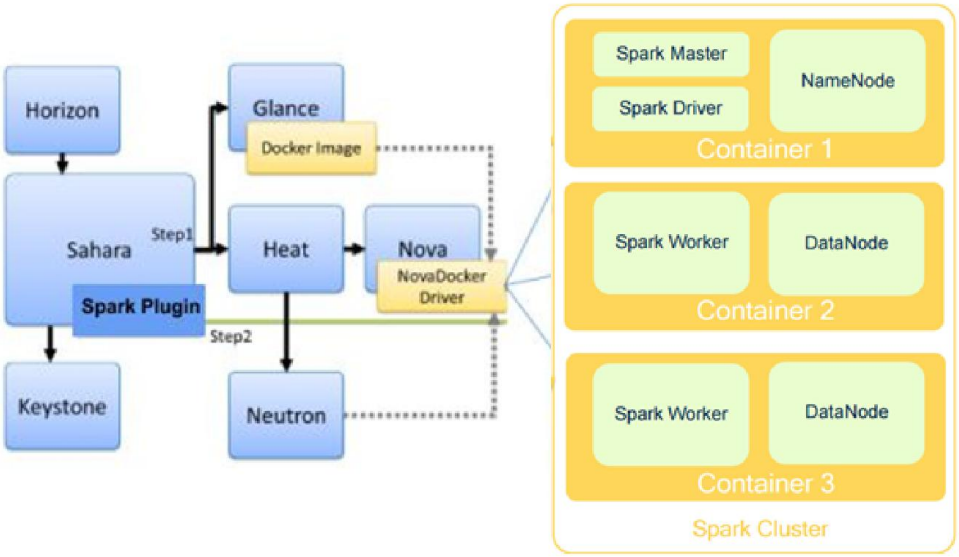


Figura 40. OpenStack Sahara con Spark Plugin y Nova-Docker

Capítulo 3. Diseño e Implementación de una arquitectura de Nube basada en OpenStack para entornos de Big Data

3.1 Introducción

En este capítulo se explican los principales criterios tenidos en cuenta a la hora de proponer una arquitectura de despliegue multi-nodo de OpenStack Kilo para procesamiento de Big Data, se expone cómo se llevó a cabo la selección del hardware para la infraestructura física que da soporte a los servicios de Nube, se describe la arquitectura, sus componentes, la interrelación de los mismos y cómo llevar a cabo el despliegue de cada servicio en los servidores físicos correspondiente a partir de ejecutar scripts Python que automatizan las tareas de instalación y configuración de los servicios de OpenStack antes mencionados y que fueron programados como parte de la realización de este Trabajo de Fin de Máster y finalmente se muestran capturas de pantalla que muestran los resultados de pruebas hechas para verificar el correcto funcionamiento de la arquitectura propuesta.

3.2 Criterios de selección de la arquitectura de despliegue

Un despliegue de OpenStack puede ser de propósito general, enfocado a los recursos de computación, enfocado a los recursos de red o enfocado en los recursos de almacenamiento. Igualmente se pude desplegar una arquitectura multi-nodo simple que cuenta con un único Nodo Controlador o una arquitectura orientada a la alta disponibilidad basada en la redundancia de recursos de cómputo, almacenamiento y red y que se caracteriza por utilizar balanceadores de carga para distribuir las solicitudes de los usuarios entre varios Nodos Controladores.

Un despliegue OpenStack enfocado a los recursos de computación está especialmente pensado para soportar casos de uso con flujos de trabajo que demanden una utilización intensiva del procesador y de la memoria RAM, pero que no impongan requerimientos exigentes relacionados con la capacidad de los recursos de red y de almacenamiento. Entre estos casos de uso se encuentra la Computación de Altas Prestaciones o High Performance Computing (HPC), el Análisis de Big Data utilizando Apache Hadoop o cualquier otra solución de almacenamiento y procesamiento distribuido de datos, el modelo de Plataforma como Servicio (PaaS), el Procesamiento de señales para Virtualizaciones de Funciones de Red o Network Function Virtualization (NFV), así como entornos de Integración Continua y Despliegue Continuo (CI/CD).

Debido a que el caso de uso para el cual se pretende diseñar e implementar la arquitectura de Nube en este Trabajo de Fin de Máster es el análisis de Big Data, se

decidió diseñar la arquitectura de manera tal que esta estuviese enfocada en los recursos de computación.

Una arquitectura de nube enfocada a los recursos de computación debe garantizar que los recursos de hardware de los nodos servidores que forman parte de la infraestructura, especialmente el procesador y la memoria RAM, sean lo suficientemente potentes para poder dar los servicios que demandarán los usuarios, en este caso, poder brindar el servicio de creación de clústeres Hadoop para procesamiento de Big Data. Los recursos de red y de almacenamiento no serán sometidos a cargas intensivas de uso por lo que la inversión de capital debe estar principalmente destinada a la selección de potentes recursos de cómputo.

Existen múltiples arquitecturas de referencia para despliegues multi-nodo de OpenStack. La arquitectura de Nube propuesta en este trabajo se basa en incluir un Nodo Controlador, un Nodo de Red, varios Nodos de Computación y varios Nodos de Almacenamiento, además de utilizar al Servicio de Red (Neutron) para el aprovisionamiento de recursos de red. La arquitectura propuesta se basa en arquitecturas de referencia multi-nodo para despliegues iniciales de Nubes OpenStack en entornos de producción. Estas arquitecturas de referencias consultadas están descritas en la documentación oficial de OpenStack y en la documentación de Mirantis OpenStack.

Este trabajo pretende proponer una arquitectura multi-nodo simple, es decir, integrada por un solo Nodo Controlador, que incluya soporte multi-tenant, que sea tolerante a fallos a partir de incluir redundancia de recursos de cómputo, es decir, múltiples Nodos de Computación, y que pueda escalar en un futuro cercano a una configuración de alta disponibilidad caracterizada por múltiples Nodos Controladores, múltiples Nodos de Red y nuevos Nodos de Computación añadidos a los ya existentes, todo ello con el objetivo de poder hacer frente al incremento de la demanda de recursos por parte de los tenants o a la incorporación de nuevos tenants a los que se les desea dar la posibilidad de desplegar sus escenarios virtuales en la Nube.

No obstante es importante destacar que en entornos de desarrollo, las funciones del controlador, de red y de computación de los nodos pueden unificarse en un único servidor físico o máquina virtual. Igualmente existen soluciones de despliegue para entornos de producción que sugieren combinar el Nodo Controlador y el Nodo de Red en un único servidor físico e incluyen igualmente uno o múltiples Nodos de Computación. Sin embargo, si las máquinas virtuales o contenedores de los tenants, conectadas a redes virtuales creadas y gestionadas con OpenStack Neutron, incrementan el volumen de tráfico que envían o reciben desde Internet debido, por ejemplo, a que necesitan acceder a un servicio Swift público para obtener o almacenar

objetos que pudiesen ser datasets a procesar por clústeres Hadoop creados con OpenStack Sahara, entonces en vez de integrar las funciones del nodo controlador y del nodo de red en un mismo servidor físico, se recomienda separarlas en dos servidores físicos independientes. Esto se debe a que a partir de utilizar servidores físicos independientes para cada uno de estos tipos de nodos, es posible evitar la congestión potencial del procesador que resultaría si en el mismo servidor físico se llevasen a cabo el encaminamiento intensivo de paquetes y las funciones de filtrado de tráfico típicas del nodo de red, además de la sobrecarga que imponen los servicios de OpenStack que se ejecutan en el Nodo Controlador. Es por ello que en la arquitectura propuesta para desplegar se decidió separar las funciones del Nodo Controlador y del Nodo de Red en dos servidores físicos independientes.

En la Figura 41 se muestra una arquitectura mínima compuesta por tres nodos y que constituye la arquitectura multi-nodo recomendada en la documentación oficial de Openstack para hacer un despliegue inicial de un escenario de producción con altos niveles de escalabilidad, redundancia y disponibilidad. En el Nodo Controlador de la arquitectura se ejecutan servicios básicos como Keystone, Glance, Horizon, y componentes de gestión de los servicios Nova y Neutron. Igualmente, en el Nodo Controlador, se instalan servicios de soporte tales como un gestor de bases de datos SQL para crear y gestionar las bases de datos de cada servicio que lo requiera y permite escoger entre MySQL o MariaDB, un agente intermediario de mensajes o "message broker" que puede ser RabbitMQ o Qpid y que permite el intercambio de mensajería entre los distintos servicios de OpenStack, además del servidor de tiempo, usualmente implementado con el Network Time Protocol (NTP). Opcionalmente se pueden instalar en el Nodo Controlador otros servicios de OpenStack, tales como los componentes de gestión de Cinder y Swift para almacenamiento en bloques y de objetos respectivamente, el Servicio de Orquestación Heat, el Servicio de Telemetría Ceilometer, el Servicio de Bases de Datos Trove y el Servicio de Procesamiento de Datos Sahara.

En el Nodo de Red se instalan los plugins y agentes del Servicio de Red (Neutron) encargados de brindar funcionalidades de red tales como la creación de redes, subredes y enrutadores virtuales para los tenants. Estos componentes de Neutron son los responsables de brindar los servicios de conmutación, enrutamiento, filtrado de tráfico, asignación dinámica de direcciones IP a las instancias de los tenants en dependencia de la subred a la que estén conectadas y conectividad a Internet a partir de implementar reglas de traducción de direcciones. Los componentes de gestión de Neutron instalados en el Nodo Controlador transferirán las solicitudes de los clientes a los componentes del Nodo de Red una vez que hayan hecho un procesamiento inicial de esta solicitud. Una vez que los componentes del Nodo De Red hayan terminado de

procesar la solicitud, notificarán el resultado a los componentes de gestión de Neutron del Nodo Controlador para de esta manera notificar al cliente.

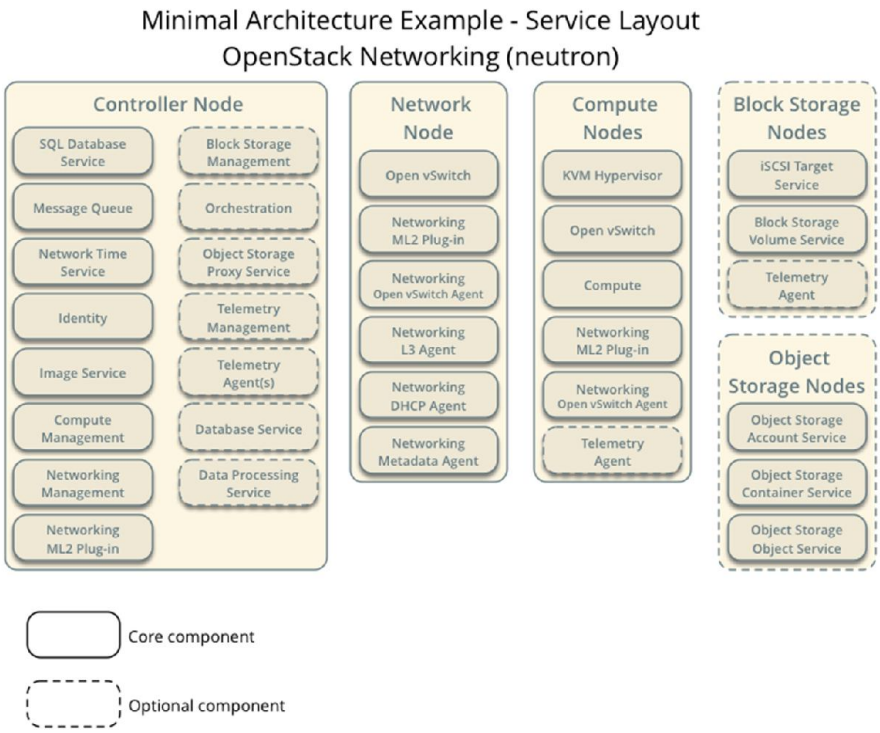


Figura 41. Arquitectura multi-nodo mínima de OpenStack para entornos de producción

En los Nodos de Computación, se instala un hipervisor o solución de virtualización ligera a nivel de sistema operativo, así como componentes del Servicio de Computación Nova que permiten aprovisionar las máquinas virtuales o contenedores de los tenants. Además en estos nodos también se instalan instancias de los plugins y agentes de red de Neutron requeridos para proporcionar conectividad de red a las máquinas virtuales o contenedores, así como configurar grupos de seguridad para filtrar el tráfico entrante y saliente de dichas máquinas virtuales a partir de reglas de filtrado implementadas con Linux Iptables. Esta última opción de filtrado de paquetes a partir de la configuración de grupos de seguridad y la asignación de instancias a estos grupos de seguridad constituye una alternativa a la opción de filtrado de paquetes que se realiza en el Nodo de Red, conocida como Cortafuegos como Servicio y que se basa en configurar la reglas de filtrado en el enrutador de cada tenant. Opcionalmente en los Nodos de Computación se pueden instalar agentes del Servicio de Telemetría de OpenStack con el fin de colectar métricas usadas con fines de facturación, benchmarking y reportes de estadísticas en cuanto al uso de los recursos y que son útiles a la hora de determinar si es necesario expandir la arquitectura.

Los Nodos de Almacenamiento en Bloques (Cinder) y Almacenamiento de Objetos (Swift) son opcionales debido al carácter opcional de estos servicios en un despliegue

OpenStack. Los Nodos de Almacenamiento en Bloques contienen los discos duros sobre los cuales se crean los volúmenes que Cinder proporciona a las instancias de contenedores o máquinas virtuales Nova como solución de almacenamiento permanente. Igualmente los Nodos de Almacenamiento de Objetos contienen los discos duros utilizados para almacenar los objetos, contenedores y cuentas de Swift.

3.3 Selección de Hardware

Los requisitos de hardware mínimos recomendados para un despliegue multi-nodo de OpenStack pueden apreciarse en la Figura 42. Estos requisitos son los recomendados por la documentación oficial de OpenStack para un despliegue inicial en entornos de producción.

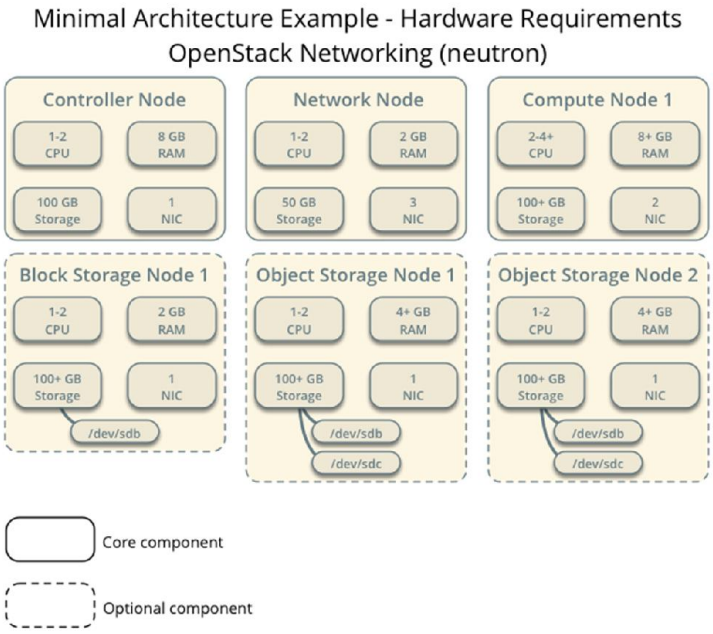


Figura 42. Requerimientos de Hardware mínimos para una arquitectura multi-nodo OpenStack

Sin embargo, si se conoce la cifra aproximada de máquinas virtuales o contenedores que se crearán y se ejecutarán de manera concurrente sobre la infraestructura de Nube, así como las características aproximadas en cuando a número de núcleos virtuales del procesador y capacidad medida en GigaBytes de memoria RAM de las máquinas virtuales o contenedores, entonces es necesario hacer una planificación incial de los recursos de manera tal que se pueda satisfacer esta demanda inicial y ser capaces además de aprovisionar recursos adicionales para cubrir futuras necesidades de expansión.

OpenStack permite sobredimensionar el CPU y la memoria RAM en los Nodos de Computación. Para ello, en el fichero de configuración del servicio Nova (/etc/nova/nova.conf) existen dos variables que permiten especificar las razones de sobredimensionamiento del CPU y de la memoria RAM respectivamente. La razón de dimensionamiento expresa la relación entre recursos virtuales disponibles y recursos físicos disponibles. En el fichero de configuración de Nova, el valor por defecto para la razón de sobredimensionamiento del CPU, especificada con la variable cpu_allocation_ratio es 16:1. Esto significa que el componente de planificación de Nova (nova-scheduler) permite asignar 16 núcleos virtuales por núcleo físico. Es decir, si un servidor físico que hace función de Nodo de Computación tiene 12 núcleos físicos, nova-scheduler verá 192 núcleos virtuales, por lo tanto, si se construyen máquinas virtuales de manera que cada una tenga dos núcleos virtuales, entonces Nova será capaz de desplegar 96 instancias. Por otro lado, el valor por defecto de la razón de sobredimensionamiento de la memoria RAM, especificada con la variable ram_allocation_ratio es 1.5:1. Esto implica que nova-scheduler planificará la creación de instancias en un Nodo de Computación mientras que la cantidad de memoria RAM total asociada a las instancias que se ejecutan sobre este servidor físico sea menor que 1.5 veces la cantidad de memoria RAM física disponible en dicho servidor. Por ejemplo, si un Nodo de Computación tiene 48 GB de memoria RAM, el planificador creará instancias en este nodo hasta que la suma de la RAM asociada con estas instancias alcance los 72 GB, lo que equivaldría a la posibilidad de crear 18 máquinas virtuales con 4 GB de RAM cada una.

La determinación de los valores que se le asignarán a estas variables durante la fase de diseño tiene un impacto directo sobre la planificación del hardware para los Nodos de Computación. Mientras mayor sean las razones de sobredimensionamiento para el CPU y la memoria RAM, mayor será la cantidad de instancias de máquinas virtuales o contenedores que se podrán ejecutar sobre estos Nodos de Computación, sin embargo peor será el rendimiento de estas instancias. Por lo tanto la selección de un valor para estas variables es una relación de compromiso entre rendimiento y cantidad de recursos de CPU y RAM que se necesitan por cada nodo, y que se traduce en monto del capital para invertir para adquirirlos.

Un ejemplo que puede ayudar a aclarar lo anteriormente expuesto es el caso en que se necesite construir una nube para dar soporte a instancias que como promedio se corresponden con el flavor m1.small de OpenStack, es decir, que poseen 1 CPU virtual, 20 GB de almacenamiento efímero y 2 GB de memoria RAM. Si los servidores físicos que funcionan como Nodos de Computación presentan dos socket de CPU, con 10 núcleos por socket, con soporte multi-threading habilitado y con una razón de sobredimensionamiento de CPU configurada con su valor por defecto de 16:1, entonces

el número de CPU virtuales disponibles para crear instancias es 640 (2 x 10 x 2 x 16) y como cada instancia de tipo m1.small requiere un CPU virtual, se podrían crear 640 instancias en cada nodo. Siguiendo el mismo razonamiento, para una razón de sobredimensionamiento de memoria RAM de 1.5:1, que es su valor por defecto, se necesitaría que cada Nodo de Computación tuviese 853 GB de RAM (640 x 2GB / 1.5). También es importante tener en cuanta cuando se está dimensionado la memoria RAM en los Nodos de Computación que una cantidad de memoria RAM adicional es necesaria para que el sistema operativo pueda ejecutarse, así como otro procesos fundamentales.

Para facilitar el dimensionamiento de los recursos de CPU y RAM de la arquitectura de Nube propuesta se utilizó la Calculadora BOM de Mirantis disponible en https://www.mirantis.com/openstack-services/bom-calculator/. Debido a que la arquitectura de Nube propuesta para OpenStack Kilo es una arquitectura multi-nodo inicial que funcionará como una especie de maqueta de pruebas, no se espera que la cantidad de máquinas virtuales que se desplieguen inicialmente sobrepasen las 100 instancias y las características de hardware promedio de dichas instancias corresponden a las de un flavor m1.small de Nova.

La Calculadora BOM de Mirantes está basada en ciertas consideraciones que se han tenido en cuenta a la hora de la selección del hardware para los servidores que funcionarán como Nodos de Computación. Entre ellas se encuentran las siguientes:

- ✓ Los Nodos de Computación necesitan una gran cantidad de memoria RAM para aprovisionar las máquinas virtuales o contenedores.
- ✓ Estos deben tener suficientes CPUs para ejecutar tantas máquinas virtuales o contenedores como las capacidades de memoria RAM lo permitan.
- ✓ Deben contar con una configuración de red que garantice una conexión robusta y resistente.

Otro factor tenido en cuenta por la Calculadora BOM de Mirantes a la hora de proponer las características de hardware de la infraestructura física es que tener slots vacíos en los servidores generalmente es una mala opción. Si se desea escalar la Nube, es mejor añadir nuevos servidores que reformar los servidores en producción. Por lo tanto con todas estas consideraciones en mente, se han aplicado las siguientes restricciones a todas las configuraciones de servidores:

- ✓ Un servidor debe tener como mínimo 16 GB de memoria RAM, al mismo tiempo, este necesita tener al menos el 60% de su capacidad de memoria máxima instalada.

- ✓ Un servidor no debe contener sockets de CPU vacíos, y debe tener al menos cuatro núcleos por CPU.
- ✓ Un servidor debe tener al menos dos interfaces 10G Ethernet para conectarse a la red donde se produce la comunicación interna de máquina virtual con máquina virtual y para acceder a la red donde se encuentran los nodos de almacenamiento.
- ✓ Un servidor debe tener al menos dos interfaces de red adicionales con no menos de 1G Ethernet cada una para la conexión con la red de gestión y la red pública.

La calculadora de Mirantis considera que la razón de sobredimensionamiento del CPU es 6:1 y que cada servidor tendrá 4 tarjetas de red por lo que se necesitarán cuatros puertos en el conmutador de red por cada servidor físico.

Para 100 máquinas virtuales que como promedio tengan 1 CPU virtual y 2 GB de memoria RAM, utilizando como proveedor de los nodos servidores a DELL y como proveedor del equipamiento de red a Cisco, la Calculadora BOM de Mirantis arrojó las siguientes características de los recursos de hardware para una arquitectura multi-nodo con un solo Nodo Controlador.

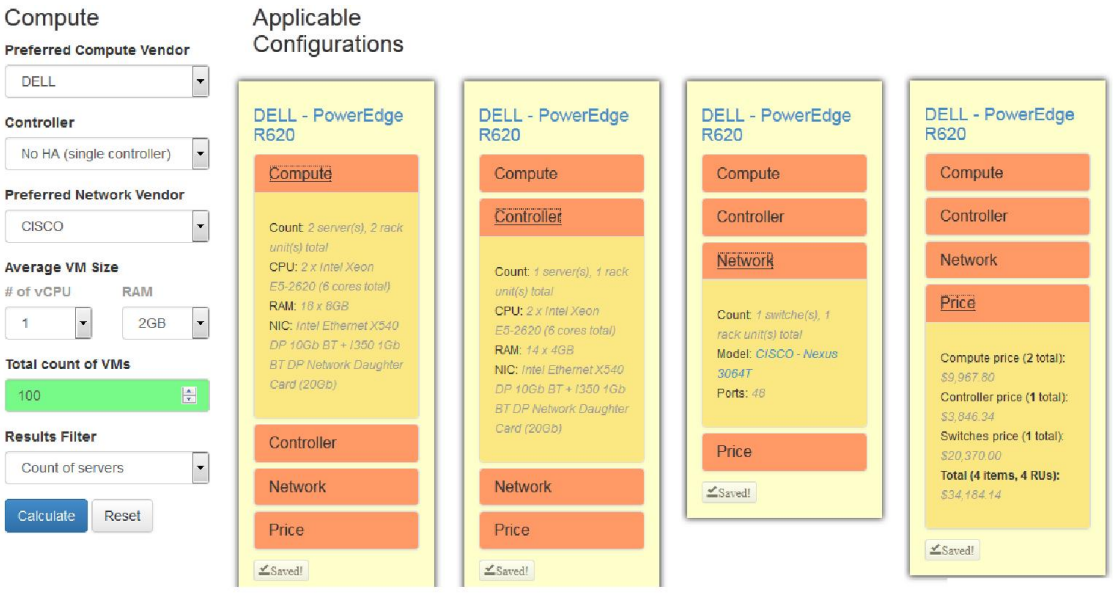


Figura 43. Configuración de recursos de hardware propuesta por la Calculadora BOM de Mirantis

Para la maqueta de pruebas que se desplegó en el centro de datos de la ETSIT UPM no se emplearon estos recursos de hardware sino otros de los que ya se disponía y que satisfacían los requerimientos de hardware mínimo que aparecen detallados en la documentación de OpenStack y que fueron presentados anteriormente. Sin embargo la selección de recursos de hardware ofrecida por la Calculadora de Mirantis constituye

la más adecuada para despliegues de producción que cuenten con el capital requerido para esta inversión.

3.4 Propuesta de arquitectura OpenStack Kilo multi-nodo

La arquitectura multi-nodo de OpenStack Kilo para procesamiento de Big Data propuesta aparece descrita en la Figura 44. Como puede verse esta arquitectura está basada en la arquitectura de referencia mínima multi-nodo propuesta en la documentación de OpenStack Kilo. Está compuesta por un Nodo Controlador, un Nodo de Red, tres Nodos de Computación, un Nodo de Almacenamiento en Bloques y dos Nodos de Almacenamiento de Objetos. En este despliegue pueden observarse tres tipos de redes, la Red de Gestión, la Red de Datos y la Red Externa.

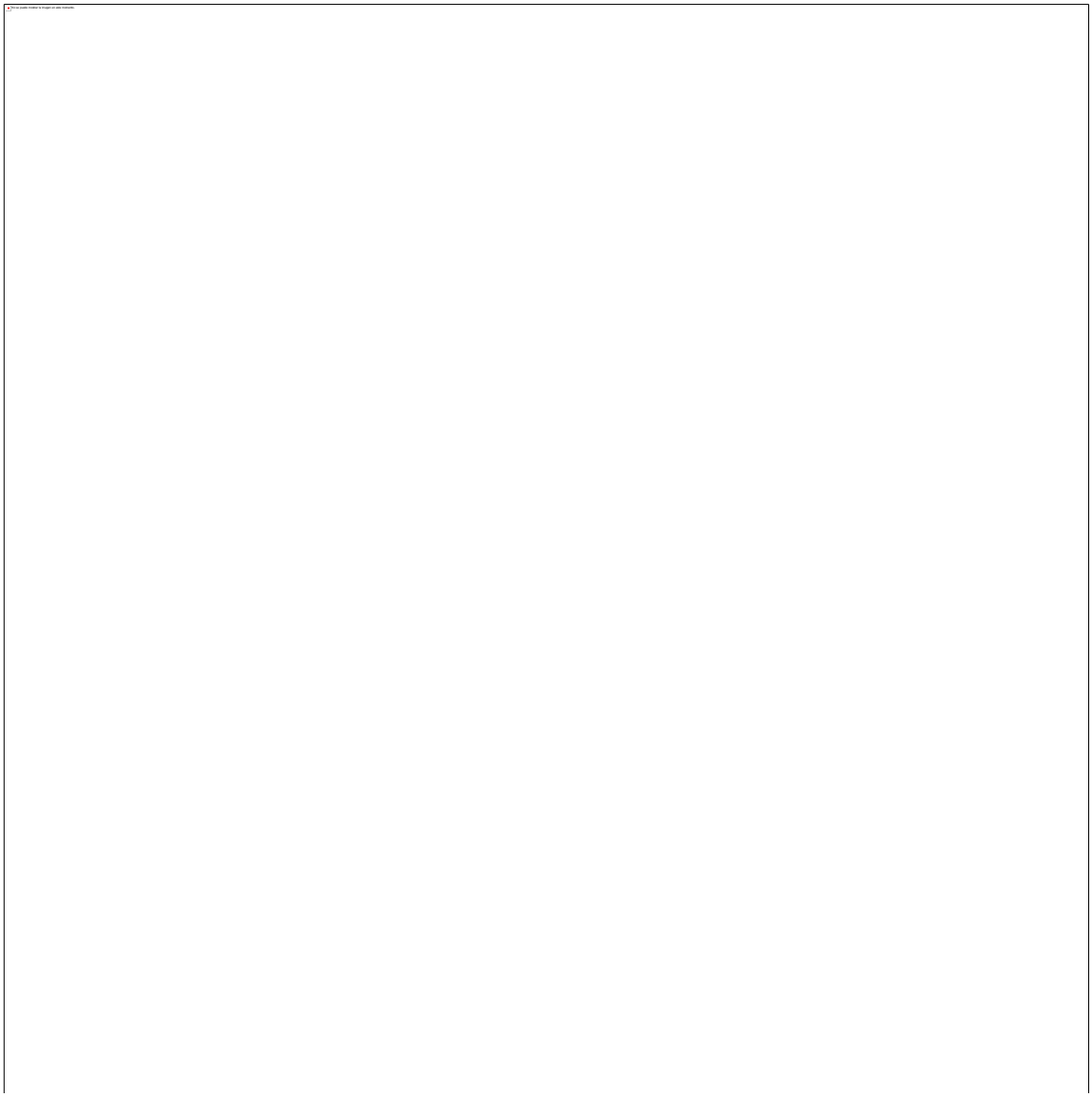


Figura 44. Arquitectura multi-nodo OpenStack Kilo propuesta

La arquitectura de referencia de OpenStack define cuatro tipos de redes: Red de Gestión, Red API, Red de Datos para tráfico de máquinas virtuales y Red Externa. Estos distintos tipos de redes no requieren interfaces de redes dedicadas en los nodos que conforman la arquitectura y el tráfico de cada red puede mezclarse a través de una única interfaz de red física. Sin embargo, para despliegues altamente seguros se recomienda interfaces dedicadas para cada tipo de tráfico.

La Red de Gestión es utilizada para que los componentes y servicios de OpenStack se comuniquen entre ellos y accedan a los servicios comunes tales como el servicio de base de datos y el servicio de mensajería. En esta arquitectura se decidió combinar la Red de Gestión con la Red API. La Red API expone las APIs de OpenStack a los tenants y servicios externos que deseen acceder a los servicios OpenStack. Cabe destacar que por motivos de seguridad, solo será accesible desde el exterior, la dirección IP del Nodo Controlador en la Red de Gestión, para los puertos correspondientes a cada servicio OpenStack que debe ser accedido desde el exterior. El rango de direcciones IP de la Red de Gestión es público, pero solo las solicitudes dirigidas a la dirección IP del Nodo Controlador y los puertos donde escuchan los servicios OpenStack autorizados serán encaminadas por el cortafuego de la ETSIT UPM, el resto será descartado.

Cada servicio de OpenStack posee tres API endpoints: uno en la red de administración, otro en la red interna y otro en la red externa. Una posible configuración para entornos de producción consiste en que cada uno de los API endpoints resida en una red diferente que sirve a un tipo determinado de usuarios. Cada API endpoint está compuesto por la dirección IP del nodo donde está instalado el servicio, en este caso el Nodo Controlador, en la red correspondiente y por el puerto donde escucha este servicio. Sin embargo otra configuración muy común y empleada en esta propuesta de arquitectura con el objetivo de simplificar el despliegue consiste en configurar los servicios de OpenStack de manera tal que para cada uno, sus tres API endpoints tengan la misma dirección IP que se corresponde con la dirección IP de la interfaz de red del Nodo Controlador en la Red de Gestión.

La Red de Datos VM Tunnel (GRE) es la que porta el tráfico entre las máquinas virtuales de los distintos tenants y entre estas y los componentes Neutron radicados en el Nodo de Red utilizando como mecanismo de traspote al protocolo de tunelización GRE. Neutron permite la creación de múltiples tipos de redes que van desde redes Flat, redes basadas en VLANs y redes overlay que utilizan protocolos de tunelización como GRE y VXLAN para encapsular el tráfico de máquinas virtuales entre Nodos de Computación y entre estos y el Nodo de Red. Para esta red en particular se utilizó GRE aunque podía haberse configurado igualmente VXLAN.

La Red de Datos VM VLAN igualmente porta tráfico entre las máquinas virtuales de los distintos tenants y entre esta y los componentes de Neutron, pero utiliza VLAN como mecanismo de trasporte.

En la Figura 44 se muestran las dos redes cada una utilizando un mecanismo de trasporte diferente, sin embargo en los scripts implementados en Python solo está automatizado la configuración para trabajar con túneles GRE y no con VLAN, en primer lugar debido a que las VLAN tiene como desventajas su reducida escalabilidad y en segundo lugar porque cada vez que se crea una red virtual mapeada con una VLAN, la VLAN debe exister y debe darse de alta en cada conmutador por donde deba pasar el tráfico y esto va en detrimento de la flexiblidad y rapidez del despliegue de la red virtual, es decir, una vez que el tenant la crea, debe esperar porque el administrador la configure y habilite en los conmutadores físicos. Con la tecnología de GRE, los tenants pueden crear sus subredes virtuales, que no existirán directamente en la infraestructura física del centro de datos, como constituye el caso de las subredes que se crean cuando el mecanismo de trasporte es VLAN. Una vez que el tenant crea su red, esta está inmediatamente disponible y con conectividad y los paquetes de las máquinas virtuales serán encapsulados en túneles GRE para ir desde un Nodo de Computación a otro o desde un Nodo de Computación a un Nodo de Red. Este mecanismo de encapsulamiento es el que permite que las direcciones IP de las redes virtuales creadas por los tenants no tengan que existir necesariamente en la infraestructura física del centro de datos. Una desventaja de utilizar GRE o VXLAN es que todo el tráfico es manipulado por el Nodo de Red, y esto incluye un único punto de fallo.

3.5 Automatización del despliegue y selección de tecnología de virtualización

El código de los scripts Pythons que fueron programados para automatizar el despliegue de los servicios OpenStack sobre la infraestructura de Nube anteriormente presentadas se encuentran disponibles en el siguiente repositorio de Github: https://github.com/mcygglez/OpenStackKiloInstallation.git

El despliegue de la arquitectura propuesta puede hacerse en un escenario de producción con servidores físicos que satisfagan los requerimientos mínimos de hardware presentados, aunque se reitera que se recomienda seguir las recomendaciones de la Calculadora BOM de Mirantis, o en máquinas virtuales VirtualBox a modo de realizar una prueba de despliegue y familiarizarse con la gestión de una Nube OpenStack. Durante la realización de este Trabajo de Fin de Máster, se instaló la arquitectura propuesta en servidores físicos en el centro de datos de la ETSIT-

UPM e igualmente se ejecutaron estos scripts para automatizar el despliegue y verificar su correcto funcionamiento en máquinas virtuales VirtualBox.

El primer paso consiste en clonar este repositorio GitHub en cada servidor físico o máquina virtual VirtualBox que forma parte de la infraestructura de Nube a partir de ejecutar el siguiente comando:

#git clone https://github.com/mcygglez/OpenStackKiloInstallation.git

En el fichero Readme.md del repositorio OpenStackKiloInstallation vienen los pasos a seguir para ejecutar los scripts en cada nodo y desplegar los servicios de OpenStack correspondientes en cada uno de ellos, no obstante se encuentran resumidos a continuación.

En el Nodo Controlador, ejecute los siguientes comandos en la consola luego de clonar el repositorio y colocarse dentro de la carpeta raíz de dicho repositorio.

#python enable_openstack_repository.py

#python controller_installer.py

El script Python enable_openstack_repository.py actualiza los repositorios de Ubuntu 14.04, que fue el sistema operativo que se seleccionó e instaló en cada servidor, y carga los repositorios de OpenStack Kilo. Este primer script se ejecutará en cada nodo de la infraestructura. El segundo script Python controller_installer.py instala en el Nodo Controlador todos los servicios y componentes de OpenStack que aparecen detallados en la Figura 44 de la sección anterior.

En el Nodo de Red, una vez clonado el repositorio y desde el directorio raíz del mismo se deben ejecutar los siguientes comandos:

#python enable_openstack_repository.py

#python network_installer.py

El script Python network_installer.py instala todos los programas, componentes y servicios OpenStack que aparecen descritos en la Figura 44 de la sección anterior, particularmente para el Nodo de Red.

En los Nodos de Computación se deben ejecutar los siguientes scripts Python posteriormente a la clonación del repositorio.

#python enable_openstack_repository.py

#python compute_installer.py [hypervisor_type]

El hypervisor_type es un parámetro que se le pasa al script para seleccionar la tecnología de virtualización que se desplegará en el Nodo de Computación. Las opciones posibles son: qemu, kvm, lxc y docker. Como se dispone de tres Nodos de Computación, en el Nodo de Computación 1 se instaló qemu/kvm, en el Nodo de Computación 2 se instaló lxc y en el Nodo de Computación 3 se instaló Docker y el driver nova-docker.

La automatización de la instalación de los Nodos de Almacenamiento de Bloque y de Objetos aún no está implementada. El despliegue de los Servicios Cinder, Swift, Heat, Sahara y Ceilometer se realizó de manera manual siguiendo las orientaciones de la guía de instalación oficial de OpenStack Kilo para Ubuntu 14.04 y respetando las opciones de configuración que la guía recomienta. Está guía está disponible en http://docs.openstack.org/kilo/install-guide/install/apt/content/.

Una vez desplegado OpenStack Kilo es momento para cargar imágenes de máquinas virtuales o contenedores en Glance, crear la red externa que proporciona a las instancias conectividad con Internet, así como las subredes internas para un determinado tenant en Neutron y finalmente crear máquinas virtuales y contenedores en Nova. Para realizar las acciones anteriores se creó una cuenta de administración admin y una cuenta para un tenant de prueba llamado demo. Estos tenant, los usuarios, roles, permisos y ficheros donde se encuentran las variables de entorno correspondiente para ejecutar comandos OpenStack como admin y como demo fueron generados por los scripts Python.

Para cargar las imágenes se ejecutaron los siguientes comandos en el terminal del Nodo de Computación luego de importar las variables de entorno del tenant admin que se encuentran en el fichero /root/admin-openrc.sh. Se descargaron las imágenes de Cirros, Ubuntu 14.04.2 y Ubuntu 15.04 e Hipache Docker.

#source admin-openrc.sh

#mkdir /tmp/images

#wget -P /tmp/images http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img

#glance image-create --name "cirros-0.3.4-x86_64" --file /tmp/images/cirros-0.3.4-x86_64-disk.img --disk-format qcow2 --container-format bare --is-public True --progress

#wget -P /tmp/images https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img

#glance image-create --name "Ubuntu 14.04.2 x64 LTS (Trusty Tahr)" --file /tmp/images/trusty-server-cloudimg-amd64-disk1.img --disk-format qcow2 --container-format bare --is-public True --progress

#wget -P /tmp/images https://cloud-images.ubuntu.com/vivid/current/vivid-server-cloudimg-amd64-disk1.img

#glance image-create --name "Ubuntu 15.04 x64 (Vivid Vervet)" --file /tmp/images/vivid-server-cloudimg-amd64-disk1.img --disk-format qcow2 --container-format bare --is-public True –progress

En el Nodo de Computación 3 donde está instalado Docker se ejecutaron los siguientes comandos:

source admin-openrc.sh

#docker pull hipache

#docker save hipache > /tmp/hipache.tar

#glance image-create --is-public True --container-format docker --disk-format raw --name hipache --progress --file /tmp/hipache.tar

Una vez cargadas las imágenes se procedió a crear la red externa.

#neutron net-create ext-net --router:external --provider:physical_network external --provider:network_type flat

#neutron subnet-create ext-net 138.4.20.0/24 --name ext-subnet --allocation-pool start=138.4.20.100,end=138.4.20.250 --disable-dhcp --gateway 138.4.20.1

Este rango de direcciones IP se corresponde con el rango de la Red Externa que aparece descrita en la Figura 44 de la sección anterior. La red externa en Neutron siempre se mapea contra una red física en la infraestructura del centro de dato que es accesible desde Internet ya que uno de sus principales fines es servir como fuente de direcciones IP flotantes que serán asociadas a las máquinas virtuales a la hora de realizar traducción de direcciones (SNAT y DNAT) y de esta manera lograr que dichas máquinas virtuales puedan acceder y sean accesibles desde Internet.

Posteriormente se cargaron las variables de entorno para el tenant demo y se procedió a crear subredes para este tenant, así como crear un enrutador para interconectar estas subredes entre ellas y con la red externa que proporciona conectividad con Internet.

#source demo-openrc.sh

#neutron net-create demo-net

#neutron subnet-create demo-net 172.16.1.0/24 --name demo-subnet --gateway 172.16.1.1

#neutron router-create demo-router

#neutron router-interface-add demo-router demo-subnet

#neutron router-gateway-set demo-router ext-net

Una vez creadas las redes y los enrutadores pertinentes para el tenant demo se procedió a crear una instancia de prueba para dicho tenant. Para ello primeramente se generó un par de llaves pública y privada con el comando ssh-keygen y se le indica a Nova la ruta hacia el fichero que contiene la llave pública. Estos pares de llaves pública y privada permiten acceder a las instancias de manera segura utilizando SSH

#source demo-openrc.sh

#ssh-keygen

#nova keypair-add --pub-key ~/.ssh/id_rsa.pub demo-key

Posteriormente se listaron los flavors disponibles en Nova para seleccionar uno de ellos para crear la instancia. Igualmente se listaron las imágenes disponibles y las subredes disponibles con el fin de obtener el identificador de la imagen y de la subred.

#nova flavor-list

#glance image-list

#neutron subnet-list

Finalmente se ejecuta el commando requerido para crear la instancia que por defecto se creará en el Nodo de Computación 1 donde está instalado qemu/kvm.

#nova boot --flavor m1.tiny --image cirros-0.3.4-x86_64 --nic net-id=5bf6508d-359d-42ac-8560-b92e24266485 --security-group default --key-name demo-key demo-instance1

Para verificar que la instancia se ha creado exitosamente y que se su tarjeta de red virtual ha sido conectada a la subred correspondiente asignándole una dirección IP de este rango se puede utilizar el siguiente comando:

#nova list

Esta instacia será asignada al grupo de seguridad por defecto al cual hay que habilidarle el tráfico ssh para que poder entrar de manera remota a la instancia. Igualmente si se quieren realizar pruebas de conectividad utilizando el comando ping, habrá que habilitar el protocolo ICMP. Esto se realiza a partir de ejecutar los siguientes comandos.

#nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0

#nova secgroup-add-rule default tcp 22 22 0.0.0.0/0

Para que la instancia sea accessible desde Internet, se debe crear una dirección IP flotante y se debe asignar dicha dirección IP flotante a la instancia. Esta dirección IP flotante pertenece al rango de direcciones IP de la red externa creada en Neutron anteriormente y que está directamente mapeada con la Red Externa, que constituye una red con direcciones IP públicas. Para ello se ejecutaron los siguientes comandos:

#neutron floatingip-create ext-net

#nova floating-ip-associate demo-instance1 138.4.20.101

Posteriormente se debe poder acceder de manera remota a la instancia utilizando SSH con el siguiente comando.

#ssh cirros@138.4.20.101

La configuración creada anteriormente debe modificarse ligeramente si se desea explotar al máximo las características del escenario. Por ejemplo si se desea instruir a Nova para que cuando se ordene la creación de una máquina virtual con qemu/kvm, Nova sepa que debe desplegar dicha instancia en el Nodo de Computación 1, o que cuando se ordene la creación de un contenedor lxc, Nova lo despliegue en el Nodo de Computación 2, o siguiendo la misma lógica, cuando se desee desplegar un contenedor Docker, Nova lo haga en el Nodo de Computación 3. Para ello es necesario llevar a cabo una configuración más compleja que hace uso de los filtros del componente nova-scheduler y de un concepto propio de Nova llamado Host Aggregates. Los detalles de la configuración aparecen descritos en la próxima sección.

3.6 Configuración de Nova Host Aggregates y ejecución de pruebas

Los Host aggregates consitutyen un mecanismo de Nova para particionar los nodos en una nube OpenStack basándose en características arbitrarias, por ejemplo el tipo de virtualización utilizada en el nodo. Los Host Aggregates no están expuestos de manera explícita a los usuarios, sino que los administradores son los encargados de mapear los flavors de Nova con los Host Aggregates. Los administradores logran hacer esto a partir de configurar metadatos en un Host Aggregate y asignar especificaciones

adicionales a un flavor de manera tal que haga juego, es decir, que coincidan los valores de estos parámetros adicionales del flavor con los metadatos del Host Aggregate. El encargado de encontrar esta correspondencia entre flavor y Host Aggregate es el componente planificador de Nova, es decir, el nova-scheduler. El nova-scheduler aplica un conjunto de filtros que le permiten seleccioanr, a partir de un flavor especificado por el tenant, un nodo que pertenezca a un Host Aggregate cuyos metadatos concuerden con los parámetros adicionales configurados en el flavor, y cuando encuentra este nodo, ordena la creación de la máquina virtual o contenedor sobre el mismo. Los metadatos de los Host Aggregates y los parámetros adicionales de los flavors se especifican en el formato llave-valor. Para que todo esto funcione, es necesario agregar a los filtros del nova-scheduler que por defecto vienen habilidados en el fichero de configuración de Nova (/etc/nova/nova.conf) el filtro AggregateInstanceExtraSpecsFilter.

Para que el planificador de Nova, nova-scheduler, filtre la solicitud de creación de la instancia atendiendo a las especificaciones adicionales del flavor y encuentre el Host Aggregate con los valores adecuados en los metadatos

Mediante los siguientes comandos se crearon tres Host Aggregates en Nova. Cada Host Aggregate contendrá nodos de computación con diferente tecnología de virtualización (qemu, lxc, docker). Estos comandos fueron ejecutados en el terminal del Nodo Controlador luego de cargar las variables de entorno correspondientes al tenant admin.

#nova aggregate-create qemuhyp nova

#nova aggregate-create lxchyp nova

#nova aggregate-create dockerhyp nova

Posteriormente se especificaron los metadados para cada Host Aggregate, en este caso solo uno que consiste en el tipo de virtualización (qemu, lxc, docker):

#nova aggregate-set-metadata 1 virt_type=qemu

#nova aggregate-set-metadata 2 virt_type=lxc

#nova aggregate-set-metadata 3 virt_type=docker

Luego se agregó cada nodo a un Host Aggregate

#nova aggregate-add-host 1 node1

#nova aggregate-add-host 2 node2

#nova aggregate-add-host 3 node3

Después se crearon los flavors especificando parámetros de hardware tales como memoria RAM en MB, capacidad de almacenamiento y número de CPUs virtuales. A estos flavor se le asignaron parámetros adicionales, en este caso solo uno, que coincide con el tipo de virtualización a utilizar (qemu, lxc, docker).

#nova flavor-create qemu.hyp 7 1536 10 1

#nova flavor-key qemu.hyp set aggregate_instance_extra_specs:virt_type=qemu

#nova flavor-create lxc.hyp 8 1536 10 1

#nova flavor-key lxc.hyp set aggregate_instance_extra_specs:virt_type=lxc

#nova flavor-create docker.hyp 9 1536 10 1

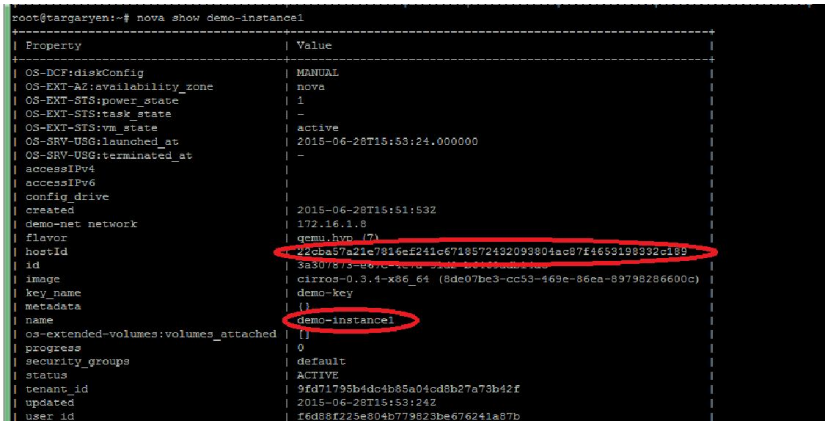
#nova flavor-key docker.hyp set aggregate_instance_extra_specs:virt_type=docker

Para verificar que la configuración anterior funciona adecuadamente se crearon tres instancias, una por cada flavor y se comprobó que para el flavor qemu.hyp, la instancia se creaba en el Nodo de Computación 1 en forma de máquina virtual, para el flavor lxc.hyp, la instancia se creaba en el Nodo de Computación 2 en forma de contenedor lxc, y finalmente para el flavor docker.hyp, la instancia se creaba en el Nodo de Computación 3 en forma de contenedor Docker. Todos estos comandos fueron lanzados en el terminal del Nodo Controlador luego de cargar las variables de entorno correspondiente al tenant demo.

Para crear una máquina virtual con QEMU en el Nodo de Computación 1 se utilizó el flavor qemu.hyp como aparece en el siguiente comando:

#nova boot --flavor qemu.hyp --image cirros-0.3.4-x86_64 --nic net-id=52c651cf-39fc-4f64-b8df-ca4379a6734b --security-group default --key-name demo-key demo-instance1

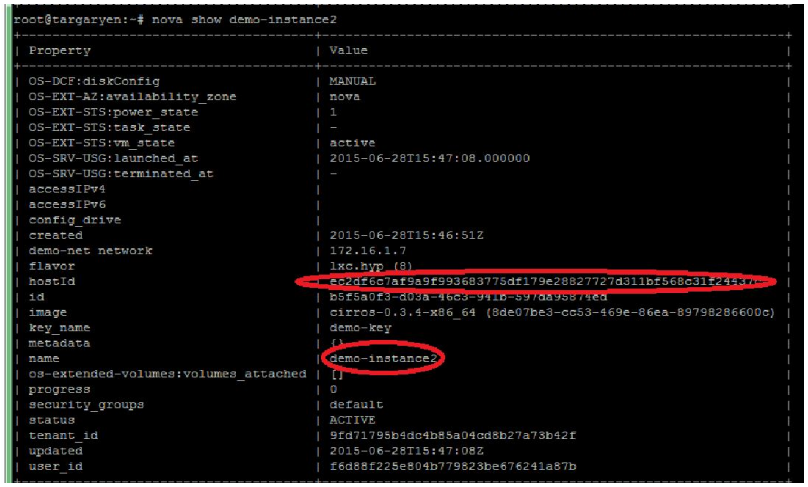
Para verificar que la instancia demo-instance1 se creó en el Nodo de Computación 1 se ejecutó el siguiente comando.



Para crear una máquina virtual con LXC en el Nodo de Computación 2 se utilizó el flavor lxc.hyp como aparece en el siguiente comando:

#nova boot --flavor lxc.hyp --image cirros-0.3.4-x86_64 --nic net-id=52c651cf-39fc-4f64-b8df-ca4379a6734b --security-group default --key-name demo-key demo-instance2

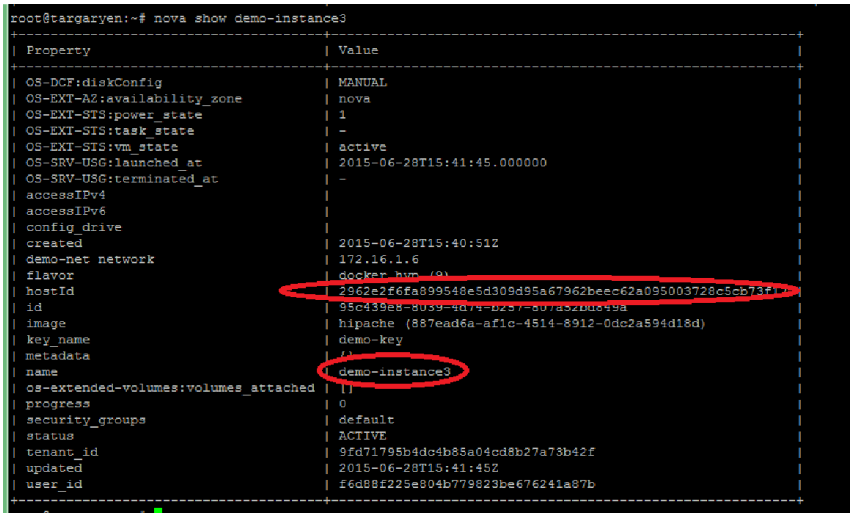
Para verificar que la instancia demo-instance2 se creó en el Nodo de Computación 2 se ejecutó el siguiente comando



Para crear una máquina virtual con Docker en el Nodo de Computación 2 se utilizó el flavor lxc.hyp como aparece en el siguiente comando:

nova boot --flavor docker.hyp --image cirros-0.3.4-x86_64 --nic net-id=52c651cf-39fc-4f64-b8df-ca4379a6734b --security-group default --key-name demo-key demo-instance3

Para verificar que la instancia demo-instance3 se creó en el Nodo de Computación 3 se ejecutó el siguiente comando:



Conclusiones y Trabajos Futuros

Con el presente Trabajo de Fin de Máster, se pudo profundizar y lograr un mejor entendimiento de los distintos componentes y servicios disponibles en OpenStack Kilo. La familiarización con las funcionalidades ofrecidas por cada servicio, el estudio de las características de sus arquitecturas internas, las distintas opciones de configuración que soportan, las diversas tecnologías sobre las que se sustentan y como cada proyecto interactúa con el resto de los proyectos de OpenStack constituyeron tareas claves que proporcionaron los conocimientos y habilidades requeridas para proponer una arquitectura de Nube multi-nodo para procesamiento de Big Data basada en esta solución de Infraestructura como Servicio.

Luego de exponer las distintas ventajas y desventajas de las soluciones de virtualización existentes y su integración con OpenStack, así como de las distintas soluciones ofrecidas por OpenStack para proporcionar servicios de red, se decidió desplegar, además de las tecnologías de virtualización completa, las tecnologías de virtualización ligeras como LXC y Docker, este último a partir de instalar el driver de Nova (nova-docker).

La Arquitectura de Nube multi-nodo propuesta se basó en la última distribución de OpenStack llamada Kilo. La infraestructura física subyacente presenta una estructura simple con un solo Nodo Controlador, un Nodo de Red, tres Nodos de Computación y tres Nodos de Almacenamiento, uno de Bloques y los dos últimos de Objetos. Los servicios de OpenStack instalados incluyeron los servicios básicos de Keystone (Identidad), Glance (Imágenes), Nova (Computación), Red (Neutron) y los opcionales Cinder (Almacenamiento en Bloques), Swift (Almacenamiento de Objetos), Heat (Orquestación) y Ceilometer (Telemetría).

La arquitectura propuesta incluyó tres Nodos de Computación donde se instalaron QEMU/KVM, LXC y Docker respectivamente. Por otro lado, para brindar los servicios de red de la Nube, se decidió utilizar Neutron en vez del componente de red de Nova llamado nova-network dada las limitaciones de este último en cuanto a escalabilidad y flexibilidad.

Para dar solución al caso de uso de procesamiento de Big Data en la Nube, se instaló el servicio OpenStack Sahara el cual se encarga de crear clústeres Hadoop. Sahara podrá crear clústeres de máquinas virtuales QEMU/KVM o clústeres de contenedores LXC o Docker, en dependencia del flavor que se seleccione para crear los nodos del clúster Hadoop. Esto es posible debido a que se crearon tres flavors que el planificador de Nova mapeará con tres Host Aggregates, de manera que cada Host Aggregate contiene Nodos de Computación con una tecnología de virtualización diferente (QEMU/KVM,

LXC, Docker). La posibilidad de crear nodos Docker como parte de un clúster Hadoop constituye una solución que máxima la eficiencia en el uso de los recursos.

Se recomienda a modo de continuidad de este proyecto, el continuo estudio de OpenStack para lograr despliegues de Nube cada vez más acorde a las necesidades de las empresas a partir de utilizar tecnologías en desarrollo novedosas y eficientes. OpenStack es un sistema en constante evolución, que se va reinventando para poder combinarse con múltiples servicios y tecnologías que van surgiendo, así como para la mejora de la seguridad y eficiencia de la nube.

Igualmente constituye una recomendación de este trabajo, estudiar las opciones de migración a despliegues de alta disponibilidad. El escenario propuesta constituye un despliegue inicial a modo de maqueta, sin embargo, para escenarios de producción de grandes centros de datos, es recomendable una solución basada en la redundancia de recursos y la alta disponibilidad.

Bibliografía

References

[1] (January, 2015). *Roundup Of Cloud Computing Forecasts and Market Estimates, 2015.
*. Available: http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/.

[2] (January 2015). *2015 State of the Network Study – Technology Adoption Trends & Their Impact on the Network*. Available: http://www.scribd.com/doc/253406492/Network-World-State-of-the-Network-2015.

[3] (September, 2011). *The NIST Definition of Cloud Computing, Recommendations of the National Institute of Standards and Technology*. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[4] (November, 2014). *Digital Business Rethinking Fundamentals*. Available: http://cbs2014.saugatucktechnology.com/images/Documents/Presentations/CBS14%20McNee%20Keynote%20-%20Cloud%20and%20Digital%20Business-12Nov2014.pdf.

[5] (February, 2015). *Right Scale Cloud Computing Trends: 2015 State of the Cloud Survey*. Available: http://assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf.

[6] (November, 2014). *The Role of Cloud in IT Modernisation: The DevOps Challenge*. Available: http://www.rackspace.co.uk/sites/default/files/UnlockedNov2014_TheRoleOfCloudInITModernisation_Ovum.pdf.

[7] (January, 2015). *Cloud State of the Union, 2015*. Available: http://www.slideshare.net/cote/cloud-state-of-the-union-2015-or-whats-up-with-cloud-5-years-later.

[8] D. DeRoos and I. Books24x7, *Hadoop for Dummies*. Hoboken, New Jersey: John Wiley & Sons, 2014.

[9] (June, 2015). *OpenStack Cloud Administration Guide (Kilo)*. Available: http://docs.openstack.org/admin-guide-cloud/admin-guide-cloud.pdf.

[10] (June, 2015). *OpenStack Installation Guide (Kilo)*. Available: http://docs.openstack.org/kilo/install-guide/install/apt/openstack-install-guide-apt-kilo.pdf.

[11] Tilmann Beitter and Sebastian Zielenski, *IaaS Mit OpenStack.* dpunkt, 2014.

[12] Frank J. Ohlhorst, "OpenStack: An Overview," *Network Computing - Online,* 2012.

[13] (June, 2015). *OpenStack Virtual Machine Image Guide (Kilo)*. Available: http://docs.openstack.org/image-guide/image-guide.pdf.

[14] (June, 2015). *OpenStack Configuration Reference (Kilo)*. Available: http://docs.openstack.org/kilo/config-reference/config-reference-kilo.pdf.

[15] (June, 2015). *OpenStack Command Line Interface Reference (Kilo)*. Available: http://docs.openstack.org/cli-reference/cli-reference.pdf.

[16] (2007). *Understanding Full Virtualization, Paravirtualization and Hardware Assist*. Available: http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf.

[17] (2014). *Linux Container and The Future Cloud*. Available: http://haifux.org/lectures/320/netLec8_final.pdf.

[18] (April, 2014). *Passive Benchmarking with docker LXC, KVM & OpenStack*. Available: http://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack.

[19] C. Anderson, "Docker," *IEEE Software,* vol. 32, pp. 102-c3, 2015.

[20] (March, 2014). *Docker drops LXC as default execution environment*. Available: http://www.infoq.com/news/2014/03/docker_0_9.

[21] (October, 2013). *OpenStack Networking Technical Deep Dive*. Available: http://www.slideshare.net/yfauser/open-stack-networking101part2techdeepdive?related=1.

[22] (2014). *The Future of Containers in Linux and OpenStack*. Available: https://www.openstack.org/summit/openstack-summit-atlanta-2014/session-videos/presentation/the-future-of-containers-in-linux-and-openstack.

[23] (April, 2015). ***Containers vs. virtual machines: How to tell which is the right choice for your enterprise***. Available: http://www.itworld.com/article/2915530/virtualization/containers-vs-virtual-machines-how-to-tell-which-is-the-right-choice-for-your-enterprise.html.

[24] (May, 2015). ***Operating System Containers vs. Application Containers***. Available: https://blog.risingstack.com/operating-system-containers-vs-application-containers/.

[25] (November, 2013). *Docker plugin for Heat*. Available: http://www.slideshare.net/dotCloud/open-stack-heat-docker-unconference-1.

[26] (November 2014). ***Orchestrating Docker with OpenStack***. Available: http://www.slideshare.net/ewindisch/docker-open-stack-paris.

[27] (May, 2015). *Magnum Wiki*. Available: https://wiki.openstack.org/wiki/Magnum.

[28] (May, 2015). ***Magnum - Containers-as-a-Service for OpenStack***. Available: https://www.openstack.org/summit/vancouver-2015/summit-videos/presentation/magnum-containers-as-a-service-for-openstack.

[29] (May, 2015). *Murano Wiki*. Available: https://wiki.openstack.org/wiki/Murano/ApplicationCatalog.

[30] (April, 2015). *Murano User Documentation*. Available: http://murano.readthedocs.org/en/stable-kilo/.

[31] (May, 2015). *Sahara Wiki*. Available: http://docs.openstack.org/developer/sahara/.

[32] (December, 2014). ***Experimenting with OpenStack* Sahara* on Docker* Containers***. Available: https://software.intel.com/en-us/blogs/2014/12/28/experimenting-with-openstack-sahara-on-docker-containers.