ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE UN PROTOTIPO PARA LA GENERACIÓN AUTOMÁTICA DE INFRAESTRUCTURA DE RED SDN A TRAVÉS DE UNA NUBE (IaaS)

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

ALVARO DAVID JARRÍN ARBOLEDA
alvarojarrin@hotmail.com

DIRECTOR: ING. DAVID MEJÍA, MSc.
david.mejia@epn.edu.ec

Quito, Julio 2015

DECLARACIÓN

Yo, Alvaro David Jarrín Arboleda, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Alvaro David Jarrín Arboleda

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Alvaro David Jarrín Arboleda, bajo mi supervisión.

_____

Ing. David Mejía, MSc.
DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

A Dios, que cuando menos lo esperamos nos coloca delante una aventura mágica que pone a prueba nuestra voluntad, nuestros “planes”, pero que nos permite seguir nuestros sueños.

A mi director de Proyecto de Titulación, Ing. David Mejía MSc, que con sus conocimientos guió y puso orden a este trabajo y particularmente, porque siempre estuvo dispuesto a colaborar para alcanzar el objetivo y de esta manera con mis ilusiones.

A mis padres por su amor, tiempo y apoyo incondicional a lo largo de mi vida, me enseñaron que en la vida lo importante es formarse como persona para lograr cualquier cosa. A mi hermano por todo lo vivido, su amor, cuidado y ser un modelo a seguir.

A Verónica Carrión, por su amor, cariño, apoyo y compañía durante toda mi carrera universitaria y en especial durante el desarrollo del Proyecto de Titulación en donde me brindó un apoyo incondicional.

A la hermosa familia de la que soy parte, a mis abuelitos, tíos y primos, porque ser parte de ustedes es mucho más que buena suerte, es el mejor regalo que se puede recibir en la vida.

A los maestros que han enriquecido mi vida y que me han llevado a descubrir cada día que no se termina de aprender.

A mis compañeros y amigos por las risas y los buenos y malos momentos compartidos.

Gracias a todos los que han tocado mi vida.

DEDICATORIA

A mis padres porque de su mano he llegado a este momento, por su infinito amor, por ser constantes en su apoyo y en su comprensión.

A mi ñaño por tantas y tantas vivencias, eres más que un hermano y significas mucho para mí.

A Verónica Carrión porque es cómplice de todos mis logros y mi compañera en quien puedo confiar en buenos y malos momentos.

Son el latido de mí ser.

CONTENIDO

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

CAPÍTULO IV

ÍNDICE DE FIGURAS

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

ÍNDICE DE TABLAS

CAPÍTULO I

CAPÍTULO II

ÍNDICE DE COMANDOS

CAPÍTULO II

CAPÍTULO III

ÍNDICE DE CÓDIGOS

CAPÍTULO III

PRESENTACIÓN

Hoy en día la conexión entre dispositivos es algo indispensable y necesario para el desarrollo de la comunicación, es por eso que se ha diseñado un prototipo que permite la creación de infraestructura SDN en la nube, conjugando estos dos conceptos que están tomando gran protagonismo en el campo de las redes.

El presente Proyecto de Titulación está compuesto por cuatro capítulos, desarrollados de la siguiente manera: el Capítulo I, presenta el marco teórico que establece una base conceptual acerca de lo que es la computación en la nube, el Capítulo II se refiere a la implementación de la nube en el Proyecto de Titulación, el Capítulo III corresponde al proceso de desarrollo de la aplicación web que permita automatizar el proceso de creación de SDN, finalmente en el Capítulo IV se presentan las conclusiones y recomendaciones derivadas de la realización del Proyecto.

En el marco teórico se define la computación en la nube y sus características generales con una breve descripción de la virtualización y su aplicación a fin de comprender la relación que existe entre virtualización y la computación en la nube. Dichos conceptos dan una referencia para la interpretación de los capítulos posteriores.

La implementación de la nube se describe en el Capítulo II, definiendo como antecedentes la descripción del entorno de desarrollo y los dispositivos utilizados, lo que permite explicar el procedimiento realizado para la implementación en la nube y el procedimiento para iniciar una instancia.

En el desarrollo de la aplicación web para automatizar el proceso de creación de SDN se da una breve explicación sobre Django que es el *framework* que se va a emplear para realizar la aplicación web y también una de sus principales características que es el uso del lenguaje de programación Python, además se describen las funcionalidades que presenta la aplicación web para el usuario facilitando que se familiarice con SDN. Finalmente, el conjunto de todos estos procesos culmina con la realización de pruebas para comprobar el correcto funcionamiento de la aplicación.

El Capítulo IV detalla las conclusiones y recomendaciones obtenidas en este Proyecto de Titulación.

En los anexos se describe la configuración de las interfaces de red que debe ser realizada en los servidores, los pasos a seguir para la instalación del software OpenStack en un ambiente multi-nodo conformado por tres nodos: nodo de red, nodo de cómputo y nodo controlador, los *scripts* desarrollados en el lenguaje Python para automatizar la creación de la infraestructura SDN, y los *scripts* necesarios para crear la aplicación web en Django.

RESUMEN

El presente Proyecto de Titulación inicia con una breve investigación sobre los conceptos básicos de computación en la nube y los temas relacionados que permiten comprender el funcionamiento de la nube, una vez realizada la familiarización con estos conceptos se investiga el software OpenStack mediante el cual se va a crear la nube, en donde se ha decidió implementar el ambiente multi-nodo por las características que presenta.

Posteriormente se describe cómo se realizó la implementación de la nube, para lo cual se inicia con un detalle de las características del ambiente en el que se implementó el Proyecto de Titulación, continuando con la explicación de las configuraciones necesarias y los requerimientos particulares que se debe tener en ambientes virtualizados, para finalizar con la comprobación del correcto funcionamiento del software mediante el uso de la nube para levantar una instancia.

A continuación se habla sobre la implementación de la aplicación web mediante la cual se va a automatizar la creación de la infraestructura SDN en la nube, para el desarrollo de esta aplicación se divide el proceso en dos etapas principales que son: *frontend* y *backend*, primero se presenta cómo se creó el *frontend* que es la parte con la que va a interactuar el y posteriormente se presenta el desarrollo del *backend* que corresponde a la lógica mediante la cual se realizan los procesos necesarios para el funcionamiento de la aplicación web, en la parte final se conecta el resultado de las dos etapas para verificar que la aplicación web funcione correctamente.

Finalmente se presentan las conclusiones que se obtuvieron al finalizar el Proyecto de Titulación y las recomendaciones que se deben tener en cuenta cuando se trabaja con temas relacionados a los presentados.

CAPÍTULO I

1. MARCO TEÓRICO

1.1 CARACTERÍSTICAS GENERALES DE COMPUTACIÓN EN LA NUBE

En este capítulo se presenta el marco teórico que establecerá una base conceptual acerca de lo que es la computación en la nube, y será una referencia para interpretar los capítulos siguientes que describen un proceso práctico a través del que se desarrollará el Proyecto. La computación en la nube propone un modelo en donde el usuario final accede a los servicios y recursos computacionales a través de una conexión a la red, siendo esta el Internet o la intranet, a diferencia de la realidad actual donde la mayoría de servicios y recursos se tienen en la computadora personal o de escritorio. Las altas prestaciones de las computadoras actuales no van a ser indispensables, ya que este modelo de computación lo que busca es centralizar los recursos y distribuirlos a los usuarios a través de una conexión de red.

En el presente capitulo se presentará la definición de computación en la nube, una breve idea de la virtualización y su aplicación para comprender la relación que existe entre virtualización y computación en la nube, y dar paso a los modelos de servicio disponibles, continuando con los modelos de implementación y las características esenciales con las que debe contar la computación en la nube, para, finalmente hablar sobre OpenStack, sus servicios, componentes y arquitectura.

1.1.1 DEFINICIÓN DE COMPUTACIÓN EN LA NUBE [1]

La computación en la nube (*Cloud Computing*) según la NIST (*National Institute of Standards and Technology*) es: “un modelo que permite el acceso a un conjunto de recursos computacionales compartidos (ej: redes, servidores, almacenamiento, aplicaciones y servicios) de una manera ubicua, conveniente y

bajo demanda, que puede ser provista de una forma rápida e implementada con una mínima configuración o interacción con el proveedor" [1].

1.1.2 VIRTUALIZACIÓN

A continuación se presentan dos conceptos de virtualización.

"La virtualización permite que múltiples sistemas operativos se ejecuten simultáneamente en un solo equipo; rompiendo la relación entre el hardware y un único sistema operativo. Cada sistema operativo invitado es dirigido por un monitor de máquina virtual (VMM), también conocido como un hipervisor[1]. Debido a que el sistema de virtualización se sitúa entre el invitado y el hardware, se puede controlar el uso por parte de los clientes de la CPU[2] (*Central Processing Unit)*, la memoria y el almacenamiento, permitiendo incluso a un sistema operativo invitado migrar de una máquina a otra" [3].

La virtualización es la "Combinación de hardware y software que permite a un recurso físico funcionar como múltiples recursos lógicos" [4].

En consecuencia se puede decir que la virtualización es la abstracción de los recursos físicos de computación, los cuales deben ser implementados en hardware, eliminando la relación de uno a uno que existía entre hardware y software, dando paso a nuevas alternativas para el desarrollo de la computación. La virtualización plantea un escenario que tiene un sistema operativo como anfitrión sobre el cual se pueden implementar uno o varios sistemas operativos de computación virtualizados.

El hipervisor es el encargado de administrar los recursos físicos como memoria, CPU y NIC[3] (N*etwork Interface Card*) para que estos puedan ser utilizados por los demás sistemas operativos de una manera adecuada.

---

[1] Hipervisor: Es un software que permite que los dispositivos físicos compartan sus recursos con máquinas virtuales que están siendo ejecutadas como invitados sobre dichos dispositivos [13].
[2] CPU: Se encarga de realizar las operaciones de cálculo y también de controlar el flujo de datos entre los diversos elementos que forman una computadora [23].
[3] NIC: Dispositivo que permite la conexión del computador con otros dispositivos computacionales a través de una red [24].

La computación en la nube permite compartir datos, información y servicios entre otros, pero la virtualización es la que facilita la implementación de la infraestructura como servicio, es por eso que está directamente relacionada con la computación en la nube. Se puede decir que la virtualización es un componente importante del paradigma de computación en la nube. En la Figura 1.1 se presenta la relación entre la virtualización y la computación en la nube.

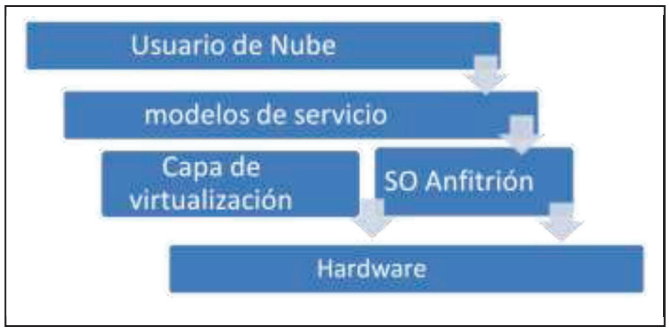

Figura 1.1. Modelo de virtualización en la nube [22]

Los principales modelos de virtualización son tres: para-virtualización, virtualización total y virtualización híbrida, la cual se basa en los dos modelos mencionados previamente [22].

a) Modelo Para-virtualizado

En este modelo se modifica el sistema operativo invitado para que esté preparado para la virtualización y no disminuya su rendimiento. Los *drivers*[4] que se emplean están modificados para obtener un mayor rendimiento en función de las características del entorno de virtualización.

Uno de los inconvenientes que presenta este modelo es que no cuenta con los permisos para la modificación de configuración en el sistema operativo anfitrión, ya que requiere determinados privilegios y no cuenta con ellos [9].

XenServer de Citrix es un hipervisor que utiliza este modelo, es una plataforma de código abierto que brinda un entorno para el usuario muy amigable.

[4] *Driver*: Programa que permite la correcta interacción entre *hardware* y *software.*

b) Modelo de Virtualización Total

Este modelo implementa el sistema operativo invitado sin realizar modificaciones sobre el mismo, usando *drivers* genéricos. El hipervisor se encarga de realizar las tareas necesarias para la correcta interactividad entre el sistema operativo y el hardware [9].

ESXi Server de VMware, es un hipervisor que utiliza el modelo mencionado, es una plataforma propietaria que permite un entorno completamente virtualizado.

c) Modelo híbrido

Se implementa el modelo para-virtualizado en un contenedor de máquinas virtuales que emplea el modelo de virtualización total, la idea es que mediante la para-virtualización se manejan las entradas y salidas necesarias para la virtualización y para la CPU se maneja una virtualización total, aprovechando así las mejores prestaciones de cada uno de los modelos indicados previamente, con la finalidad de presentar un modelo híbrido más eficiente.

KVM (*Kernel Based Virtualization Machine*) es una plataforma de virtualización de código abierto, está diseñada para que trabaje con el *kernel* de Linux, el cual se encarga de que el sistema operativo Linux funcione como hipervisor para una correcta virtualización, utilizando el modelo híbrido aprovechando las mejores características de los modelos mencionados.

1.1.3 MODELO DE COMPUTACIÓN EN LA NUBE

El modelo de computación en la nube está compuesto por cinco características esenciales, tres modelos de servicio y cuatro modelos de implementación, los cuales se van a presentar a continuación.

1.1.3.1 Características esenciales

A continuación se presentan las cinco características esenciales con las que se debe contar en el modelo de computación en la nube.

Autoservicios bajo demanda

Cuando el usuario requiere modificar recursos computacionales no es necesario una interacción con el proveedor de servicios, es un proceso que lo puede realizar el cliente en cualquier momento por sí solo.

Gran ancho de banda

Mediante la red se accede a los servicios que se ofrecen a través de la nube, razón por la cual se debe contar con un ancho de banda que garantice que los servicios entregados lleguen al cliente con parámetros aceptables para su uso.

Conjunto de recursos

El proveedor de servicios debe contar con un conjunto de recursos computacionales, para brindar servicios a diferentes clientes independientemente de la ubicación, los recursos computacionales son asignados y re-asignados a los diferentes clientes en función de las necesidades.

Rápida escalabilidad

Los recursos computacionales pueden ser incrementados en función de las necesidades del cliente de una manera rápida, ya que el proveedor de servicios debe estar preparado para brindar una mayor capacidad de recursos en corto tiempo.

Medición de los servicios

El modelo que propone la computación en la nube debe contar con control automático para el monitoreo y reportes sobre el uso de los recursos ofrecidos por el proveedor de servicios.

1.1.3.2 Modelos de servicio de computación en la nube

La computación en la nube pretende centralizar los recursos de computación y entregar como servicio lo que el cliente necesite a través de una conexión de red. Los modelos de servicios que ofrece la computación en la nube son tres, cada uno de ellos ofrece un servicio diferente al usuario y cumple una función específica. En la Figura 1.2 se muestran los diferentes modelos de servicio y la relación que existe entre ellos.

Figura 1.2. Modelos de servicio [14]

1.1.3.2.1 Modelo de servicio SaaS [2], [14].

SaaS (*Software as a Service*) busca eliminar la necesidad de instalar y ejecutar un determinado software en el computador personal o de escritorio, lo que se propone es garantizar el acceso a software a través de una conexión a la red. Este modelo de computación en la nube evita que el usuario tenga que actualizar el software e instalar parches, ya que esas actividades son competencia del proveedor de servicios.

Forrester define los niveles de madurez que detallan las diferentes opciones a través de las que se puede implementar SaaS [5], [2], los cuales se exponen a continuación:

Nivel 1. Proveedor de aplicaciones de servicio de manera manual

Es un modelo de entrega de software en donde cada cliente tiene su versión personalizada de la aplicación y cuenta con su propia instancia de la aplicación en los servidores del proveedor de SaaS. En la Figura 1.3 se muestra una

representación gráfica del nivel 1, este tipo de SaaS está orientado a medianas empresas porque permite implementar aplicaciones específicas sin preocuparse de la infraestructura, implementación y mantenimiento.

Figura 1.3. Nivel 1 de madurez

Nivel 2. Proveedor de aplicaciones de servicio industrial

El proveedor del software ofrece una instancia para cada cliente. En comparación con el primer nivel donde cada cliente tiene su propia versión, en este nivel, todas las instancias utilizan la misma versión de la aplicación y el proveedor cumple con las necesidades de los clientes mediante las opciones de configuración. En la Figura 1.4 se muestra una representación gráfica del nivel 2.

Figura 1.4. Nivel 2 de madurez

Nivel 3. Una sola aplicación

El proveedor a través de una única instancia da servicio a todos los clientes, en algunos casos ofreciendo la posibilidad de que cada uno configure su aplicación en base a sus necesidades. Las políticas de autorización y seguridad permiten que cada cliente mantenga sus datos separados de otros clientes y, desde el

punto de vista del usuario, no hay indicios de que la instancia esté siendo compartida entre varios clientes. Dado que varios clientes comparten una instancia, los datos de cada cliente son separados de manera lógica. En la Figura 1.5 se muestra una representación gráfica del nivel 3.

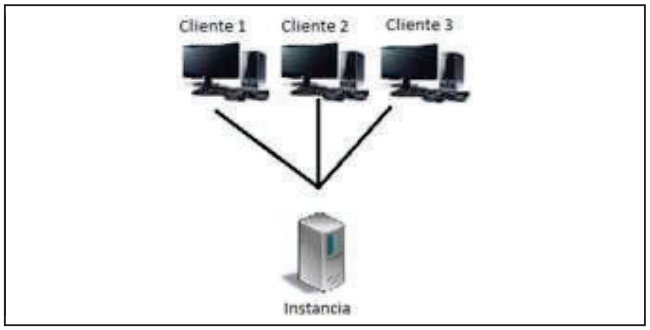

Figura 1.5. Nivel 3 de madurez

Nivel 4. Módulo del Negocio

El proveedor da servicio a distintos clientes a través de varias instancias de nivel 3 balanceando la carga de cada instancia. Se puede dar servicio a varios clientes desde máquinas distintas, ofreciendo este nivel un alto grado de escalabilidad, ya que el número de servidores puede ser aumentado o disminuido para satisfacer la demanda, sin necesidad de rediseñar la aplicación. El proveedor permite que el cliente agregue servicios según sus requerimientos con la finalidad de que pueda mover todo un módulo de su negocio y lo reciba como servicio. En la Figura 1.6 se muestra una representación gráfica del nivel 4.

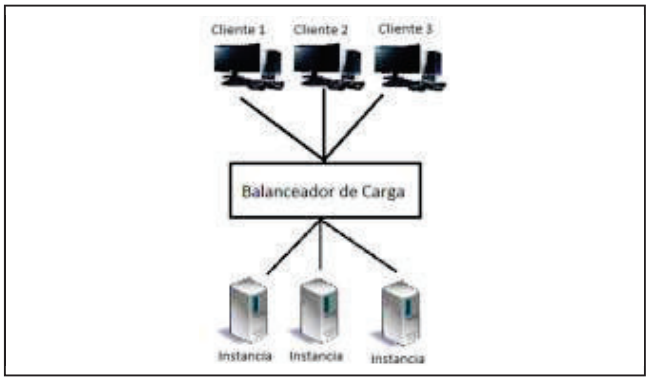

Figura 1.6. Nivel 4 de madurez

Nivel 5: Aplicación empresarial dinámica como servicio

La aplicación dinámica de negocios abarca un nuevo paradigma de desarrollo de aplicaciones basada en: "diseño para la gente, construir para el cambio". Un proveedor de SaaS avanzado de nivel 5 proporciona una plataforma de integración y aplicación integral bajo demanda, que consiste en agregar previamente aplicaciones de negocio o servicios empresariales, estas pueden ser aplicaciones específicas del usuario en varios niveles. La agilidad del proceso trae beneficios a todos, incluyendo a los clientes de grandes empresas. En la Figura 1.7 se presenta una gráfica del nivel de madurez 5.

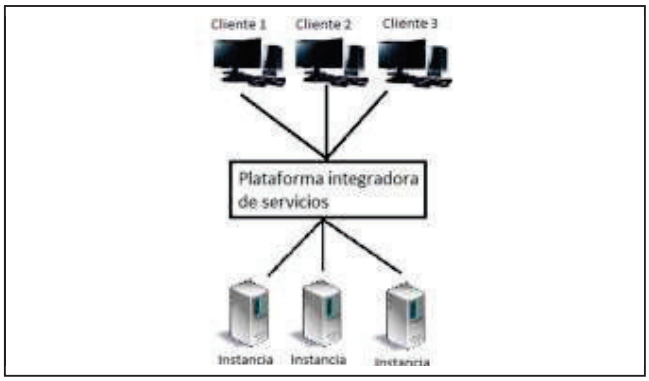

Figura 1.7. Nivel 5 de madurez

1.1.3.2.2 Modelo de servicio PaaS [2], [14]

PaaS (*Plataform as a Service*) es un grupo de servicios que abstrae la infraestructura de aplicaciones, los sistemas operativos, el middleware[5] y los detalles de configuración, y ofrece a los equipos de desarrolladores la capacidad de aprovisionar, desarrollar, diseñar, probar e implementar aplicaciones. PaaS facilita la implementación de aplicaciones mediante herramientas, recursos y automatización por demanda y autoservicio, además de un contenedor de tiempo de ejecución. Esto elimina la necesidad de un kit de instalación y los desarrolladores ya no deberán configurar servidores físicos o máquinas virtuales

[5]Middleware: Software que realiza la función de una capa de conversión o traducción. Soluciones de middleware personalizadas pueden ser desarrolladas para permitir la interacción de aplicaciones con distintas plataformas o aplicaciones de distintos fabricantes [15].

(VM) y esperar que funcionen, ni copiar archivos de un ambiente a otro a medida que avanza el ciclo de vida de la aplicación.

PaaS agiliza la administración del ciclo de vida, desde el diseño de la aplicación hasta su eliminación al finalizar su ciclo, automatizando los pasos y la funcionalidad asociados con cada etapa. PaaS también puede simplificar la actualización de versiones, los parches y otras actividades de mantenimiento. Sin duda PaaS trae beneficios si se los sabe aprovechar, a continuación se mencionarán varios de ellos:

- Facilidad en el despliegue de las aplicaciones del cliente, reduciendo costos y tiempo, se evitaría la compra de hardware, software y la capacitación requerida para administrarlos, los clientes necesitarán solicitar al proveedor los recursos indispensables.

- Flexibilidad para que los clientes accedan a través de la red a todas las herramientas necesarias para crear y entregar servicios y aplicaciones web, cuenten con control total sobre lo instalado en sus plataformas y las adapten perfectamente a sus necesidades concretas.

- Permite la colaboración entre equipos situados en lugares distintos, se necesita únicamente una conexión a Internet y un navegador web, el grupo de trabajo puede estar disperso y aun así colaborar en el desarrollo de la misma aplicación.

La Figura 1.8 presenta un diagrama en el que se muestra ciertas características que tiene PaaS en cada una de las etapas por las que debe pasar una aplicación [2].

Para lanzar una aplicación desarrollada para PaaS se debe considerar que PaaS impulsa la aplicación hacia la nube desde una interfaz de línea de comando o directamente desde un ambiente de desarrollo integrado (IDE)[6], usando un plug-in. Después de analizar la aplicación, PaaS la aloja en el contenedor de tiempo de

[6] IDE (*Integrated development environment):* Es un software que presenta un conjunto de herramientas específicas para el desarrollo de aplicaciones.

ejecución que coincide con sus requerimientos de recursos. Para finalmente instanciar una o múltiples copias en la nube o varias nubes, para ambientes que podrían requerir que se los aísle de otros de la empresa. En cada uno de estos casos, el desarrollador puede seguir utilizando herramientas comunes y mejores prácticas, pero cuenta con un ambiente seguro, por separado [6].

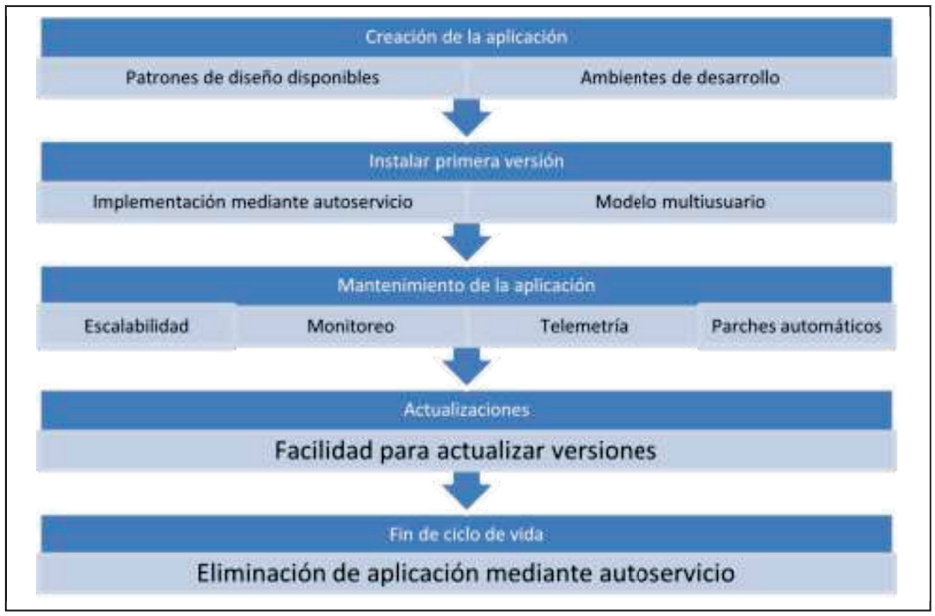

Figura 1.8. Etapas de una aplicación PaaS [26]

El modelo de servicio PaaS está orientado a un ambiente de desarrollo, ya que ofrece plataformas y herramientas para implementar nuevas aplicaciones de software principalmente, en donde se cuenta con los recursos de computación necesarios para un adecuado procesamiento de la información, para la realización de pruebas e implementación de las aplicaciones de software. Este servicio garantiza que todos tengan acceso a los recursos adecuados para el desarrollo de software, ya que no es necesario adquirir la infraestructura, lo que se requiere en este modelo es garantizar una adecuada conexión a la red.

1.1.3.2.3 Modelo de servicio IaaS [2], [14]

IaaS (*Infraestructure as a Service*) proporciona acceso a recursos computacionales de manera remota, a través de una conexión a la red, que suele ser Internet. En el caso de IaaS, los recursos de computación ofrecidos consisten

en hardware virtualizado o, en otras palabras, infraestructura de procesamiento. IaaS abarca aspectos como servidores, conexiones de red, balanceadores de carga y servicios de red entre otros. Físicamente, los recursos de hardware disponibles están alojados en grandes centros de datos, de cuyo mantenimiento se encarga el proveedor del servicio. El cliente, por su parte, obtiene acceso a los recursos de computación para construir con ellos su propia plataforma computacional.

El modelo IaaS permite a los clientes crear soluciones informáticas económicas y fáciles de ampliar, en las cuales toda la complejidad y el costo asociados a la administración del hardware se dejan en manos del proveedor del servicio. Si la escala o el volumen de actividad del negocio del cliente fluctúan, o si la empresa tiene previsto crecer, puede recurrir a los recursos computacionales entregados a través de la red en el momento y de la manera en que lo necesite, en lugar de tener que adquirir, instalar e integrar hardware por su cuenta.

A continuación se mencionarán algunas de las ventajas que presenta una implementación basada en el modelo de Infraestructura como Servicio [14]:

- Escalabilidad; los recursos están disponibles de la manera y en el momento en que el cliente los necesita, por lo que desaparecen los tiempos de espera a la hora de ampliar la capacidad, permitiendo que los recursos sean utilizados de una manera eficiente.

- Inversión inicial reducida, el hardware físico sobre el que funciona el servicio IaaS es configurado y mantenido por el proveedor del servicio, lo que evita que el cliente tenga que dedicar tiempo y dinero a realizar esa instalación.

- Incluye un modelo de tarifación, similar al de los suministros públicos como luz o agua, el servicio es accesible bajo demanda, y el cliente solo paga por los recursos utilizados.

- Independiente de la ubicación, por lo general, se puede acceder al servicio desde cualquier lugar, siempre y cuando se disponga de una conexión a Internet y las políticas de la empresa proveedora lo permitan.

- Redundancia, si falla un servidor o un conmutador, el servicio global no se verá afectado, gracias a la gran cantidad restante de recursos de hardware y configuraciones redundantes. En muchos servicios, incluso la caída de un centro de datos entero, no afecta en absoluto al funcionamiento del servicio IaaS.

Hay varios casos de uso de la infraestructura como servicio [14], así:

Infraestructura corporativa, las redes internas de una empresa, las nubes privadas y las redes locales virtuales, que utilizan recursos de red y de servidores agrupados pueden almacenar sus datos y ejecutar las aplicaciones que necesiten para su funcionamiento diario.

Las empresas en crecimiento pueden ampliar su infraestructura a medida que aumenten su volumen de actividad, mientras que las nubes privadas (accesibles solo para la propia empresa) permiten proteger el almacenamiento y transferencia de los datos delicados que algunas empresas necesitan manejar.

Hosting cloud, alojamiento de sitios web en servidores virtuales que funcionan sobre recursos físicos. Una web alojada en una plataforma *cloud*, por ejemplo, puede beneficiarse de la redundancia y la escalabilidad para afrontar cualquier pico inesperado de tráfico en su web.

Virtual Data Centers (VDC), una red virtualizada de servidores virtuales interconectados que puede utilizarse para ofrecer funcionalidades avanzadas alojadas en un entorno *cloud*, para implementar la infraestructura informática de la empresa, o para integrar todas esas operaciones dentro de una implementación *cloud* pública o privada [7].

La Tabla 1.1 resume las principales aplicaciones que ofrece cada modelo de servicio y presenta ejemplos de proveedores de cada uno de los modelos.

MODELOS DE SERVICIOS SUS APLICACIONES Y EJEMPLOS		
SaaS	***PaaS***	***IaaS***
Software as a Service	**Plataform as a Service**	**Infrastructure as a Service**
Aplicaciones • Herramientas de Producción. • Aplicaciones Gubernamentales. • Colaboración • Planeamiento empresarial.	**Aplicaciones** • Desarrollo de aplicaciones • Servicios de Seguridad • Administración de base de datos.	**Aplicaciones** • Servidores • Redes de Computación • Almacenamiento • Administración
Ejemplos • SalesForce.com • Oracle • IBM • Google Apps • NetSuite	**Ejemplos** • Microsoft Azure • Amazon EC2 • Google App Engine	**Ejemplos** • GoGrid • Flexiscale • Joyent

Tabla 1.1. Modelos de servicios, sus aplicaciones y ejemplos

1.1.3.3 Modelos de implementación [8], [1]

En la Nube se pueden implementar varios modelos de acuerdo a las necesidades, estos modelos se diferencian básicamente en la ubicación de los servicios, ya que éstos pueden ser de acceso público a través del Internet o de acceso privado que significa que están limitados a una red privada.

Otros aspectos que se deben considerar para seleccionar el modelo de implementación de la nube son el costo de inversión, confidencialidad de la información y servicios a ser implementados en la nube. En la Figura 1.9 se presentan los diferentes modelos de implementación y a continuación una explicación de cada uno de ellos.

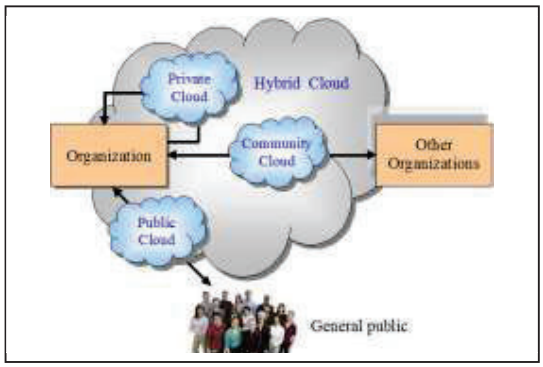

Figura 1.9. Modelos de implementación de nube [34]

1.1.3.3.1 Nube pública

Una nube se denomina pública cuando la infraestructura y los recursos lógicos son propiedad del proveedor de servicios, son administrados por este y forman parte del entorno que se encuentra disponible para el público en general, los usuarios únicamente contratan los servicios que necesitan, a los cuales se puede tener acceso mediante Internet, los proveedores de servicios elaboran aplicaciones y soluciones disponibles al público. El modelo usualmente es "pago por uso" esto quiere decir que se paga únicamente los servicios consumidos.

Reduce significativamente los costos iniciales de desarrollo de estructura y acceso inmediato a los servicios contratados, ofreciendo arquitectura estandarizada y garantizando un alto desempeño y seguridad para los usuarios.

1.1.3.3.2 Nube privada

Es aquel modelo en el cual la infraestructura se gestiona únicamente por una organización. La administración de aplicaciones y servicios suele estar a cargo de la misma organización o de un tercero y la infraestructura asociada puede estar dentro de la organización o fuera de ella. El objetivo de una nube privada es que una organización se beneficie de estos servicios. Las nubes privadas por lo general son implementadas por grandes empresas debido al alto grado de conocimientos y compromiso en virtualizar todo el ambiente de negocios.

La nube privada utiliza infraestructura propia del usuario, que garantiza el control total sobre las aplicaciones implementadas. Este modelo tiene ventajas como la simplificación de la administración de las aplicaciones, control de acceso a aplicaciones y la reducción de costos de los derechos de las licencias necesarias.

1.1.3.3.3 Nube comunitaria

Es aquel modelo donde la infraestructura es compartida por diversas organizaciones y su principal objetivo es soportar a una comunidad específica que posea un conjunto de preocupaciones similares. La infraestructura implementada y los servicios que ofrece son gestionados por las organizaciones o bien por un tercero, la ubicación de la infraestructura puede estar en las instalaciones de una de las organizaciones que son parte de la comunidad o fuera de ellas.

El modelo de nube comunitaria está orientado a organizaciones que emplean recursos computacionales afines, los cuales se pueden compartir con el propósito de reducir costos en la implementación de la infraestructura o con la finalidad de compartir y tener acceso a la información de cada una de las organizaciones para un trabajo conjunto.

1.1.3.3.4 Nube híbrida

Este modelo está formado por la combinación entre nubes públicas y privadas, la nube híbrida tiene la ventaja de mantener los dos modelos a disposición de la organización, permitiendo que sean usados de acuerdo a las necesidades y recursos, con este modelo las organizaciones pueden manejar diferentes arquitecturas en el área de tecnología de la información, pueden contar con servidores en la nube privada y utilizar aplicaciones en la nube pública o bien pueden contratar servidores físicos en centros de datos y utilizar servicios de nubes públicas y privadas.

Este modelo es una buena opción para organizaciones que necesitan aumentar la capacidad de sus servidores esporádicamente, y también puede garantizar niveles de tolerancia a fallas a través de recursos locales y servicios remotos, ya que las nubes implementadas se mantienen como entidades separadas pero están unidas por tecnologías estandarizadas o propietarias, que permiten la portabilidad de datos y aplicaciones. Razón por la cual, la computación en la nube híbrida exige un alto estándar de interoperabilidad para su implementación.

1.2. OPENSTACK

Es un proyecto iniciado en el año 2010 por la empresa Rackspace Cloud y por la agencia espacial norteamericana NASA. Las cuales decidieron crear una comunidad abierta en donde todos son bienvenidos a participar, cuenta con colaboración de desarrolladores, pioneros en computación en la nube y grandes empresas en el campo de la computación, basándose en los principios de código abierto, diseño abierto y libre desarrollo, actualmente OpenStack se maneja bajo los términos de la licencia Apache con el objetivo de producir una plataforma

ubicua que cumpla con los requerimientos tanto de nubes públicas como privadas sin importar la dimensión de las mismas.

El proyecto busca brindar soluciones para todo tipo de nubes ofreciendo una gran escalabilidad, una amplia gama de prestaciones y una comunidad dispuesta a colaborar.

OpenStack es un software que controla un conjunto de recursos de: cómputo, almacenamiento y redes, los cuales están interrelacionados mediante diferentes API (*Application Programming Interface*) con una interfaz en donde se puede administrar cada uno de los recursos para ofrecer una solución completa en la nube. En la Figura 1.10 se muestra de una manera gráfica la idea general de OpenStack.

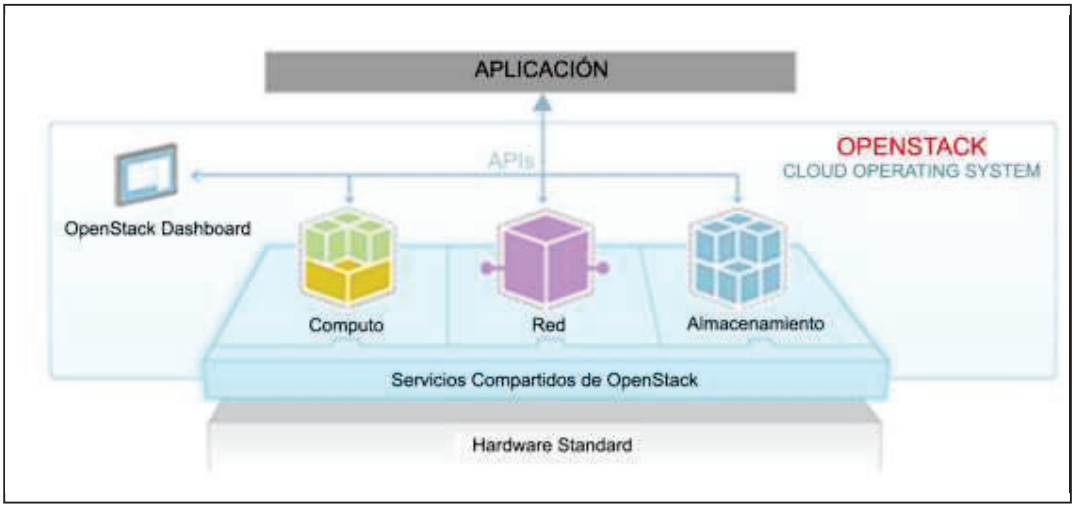

Figura 1.10. Visión General OpenStack [35]

1.2.1. SERVICIOS DE OPENSTACK

A continuación se mencionarán los servicios que maneja OpenStack, es importante tener claro que los que se emplean para la creación de una nube dependen de los requerimientos y los servicios que va a brindar la nube.

Los servicios que presenta OpenStack pueden variar de una versión a otra, dado que se van incorporando nuevas prestaciones al software OpenStack. Para este Proyecto de Titulación se empleó la versión Juno que fue lanzada en octubre del 2014 y ofrece los siguientes servicios:

1.2.1.1 Servicios Básicos

Compute (Nova)

Maneja el ciclo de vida de las instancias en el ambiente OpenStack, este servicio es el que se encarga de cargar, asignar los recursos a cada instancia y además es el organizador, responsable de manejar los procesos en las diferentes instancias [10].

Dashboard (Horizon)

Es un portal web que permite interactuar con los servicios de OpenStack, por ejemplo asignar direcciones IP, configurar los controles de acceso, cargar imágenes y lanzar las instancias [10]. Este portal web permite una administración de Openstack con ayudas gráficas tanto para el proveedor de servicios como para el usuario.

Networking (Neutron)

Es el que permite la conexión de red como servicio para los demás servicios que presenta OpenStack, cuenta con un API para que los usuarios puedan definir las redes y los dispositivos que van a estar conectados a la misma [10]. Permite la creación y administración de la red que interconecta las instancias y los dispositivos virtuales de conectividad.

1.2.1.2 Servicios de almacenamiento

Object storage (Swift)

*Almacena y recupera objetos no estructurados a través de una API HTTP (*Hypertext Transfer Protocol)[7] *basada en RESTful*[8]*. Es altamente tolerable a fallas mediante datos replicados y permite escalabilidad [10].*

Block Storage (Cinder)

---

[7] HTTP (*Hypertext Transfer Protocol):* Es un protocolo de capa, aplicación que permite la transferencia de información a través de WWW (*World Wide Web).*

[8] RESTful: Es un servicio que emplea la arquitectura REST *(Representational State Transfer*), la cual está diseñada para el desarrollo de aplicaciones web, maneja el modelo cliente-servidor, y por lo general emplea el protocolo HTTP para realizar las llamadas entre máquinas. Garantizando simplicidad al momento de realizar CRUD (*Create/Read/Update/Delete*) [17].

Ofrece almacenamiento en bloques para las instancias que están en ejecución y cuenta con un *driver* para facilitar el manejo y la creación del almacenamiento en bloques [10].

1.2.1.3 Servicios compartidos

Image Service (Glance)

Almacena y maneja las imágenes de los discos de las VM. OpenStack Compute utiliza Glance durante la instanciación de las VM.

Identity service (Keystone)

Presta el servicio de autenticación y autorización para los demás servicios de OpenStack, además tiene un catálogo de los *endpoints*[9] para todos los servicios de OpenStack [10]. Este servicio es el que maneja los perfiles de los servicios para garantizar que tengan los recursos requeridos para el correcto funcionamiento.

Telemetry (Ceilometer)

Monitorea y maneja las métricas de OpenStack para la facturación, escalabilidad y estadísticas de los servicios.

1.2.1.4 Servicios de niveles superiores

Orchestration (Heat)

Orchestration es un término relacionado a computación en la nube y hace referencia al proceso que se encarga de automatizar las tareas de los servicios y su adecuada interacción para un correcto desempeño de la nube. En OpenStack este servicio permite utilizar diferentes aplicaciones para la nube utilizando el formato HOT[10] (*Heat Orchestration Template*), o el formato nativo AWS[11] (*Amazon Web Services*) cloudfFormation y se puede acceder a través de una API

[9] *Endpoint*: Dirección que puede ser accedida a través de la red para acceder a un servicio, generalmente es una dirección URL (*Uniform Resource Locator)* [10].
[10] HOT: Es un nuevo formato de plantilla para el servicio de Orchestration diseñado para remplazar el formato nativo CFN (*CloudFormation-compatible*) [18].
[11] AWS: Es una plataforma en la nube que provee un grupo de servicios diseñados para trabajar en conjunto con la finalidad de diseñar aplicaciones sofisticadas entregadas a través del Internet.

mediante formato REST nativo de OpenStack o con una API Query que es el formato de AWS cloudFormation.

Database Service (Trove)

Ofrece una base de datos como servicio, escalable y segura, está diseñado para motores de base de datos relacionales y no relacionales.

En la Tabla 1.2 se presenta un resumen de los servicios con su respectivo nombre y función en OpenStack.

OPENSTACK			
SERVICIOS		**NOMBRE**	**DESCRIPCION**
BÁSICOS	Compute	Nova	Administra y hospeda los sistemas de computación en la nube.
	Dashboard	Horizon	Es un portal web que permite administrar e interactuar con los servicios de OpenStack.
	Networking	Neutron	Permite la conexión de red como servicio para los demás servicios que presenta OpenStack.
ALMACENAMIENTO	Objetos	Swift	Almacena y recupera objetos no estructurados a través de una API HTTP basada en RESTful.
	En bloque	Cinder	Ofrece almacenamiento en bloque para las instancias que están en ejecución.
COMPARTIDOS	Identidad	Keystone	Presta el servicio de autenticación y autorización para los demás servicios de OpenStack.
	Imágenes	Glance	Almacena y maneja las imágenes de los discos de las VM
	Telemetría	Ceilometer	Monitorea y maneja las métricas de OpenStack.
NIVELES SUPERIORES	Orchestration	Heat	Permite utilizar diferentes aplicaciones para la nube utilizando ciertos formatos.
	Database	Trove	Ofrece una base de datos como servicio escalable y seguro.

Tabla 1.2. Resumen de los servicios de OpenStack

1.2.2 ARQUITECTURA DE OPENSTACK

Se va a presentar dos arquitecturas del software OpenStack, la arquitectura conceptual, la cual presenta los patrones de diseño y la arquitectura lógica, la cual pretende mostrar los principios técnicos que maneja OpenStack.

1.2.2.1 Arquitectura Conceptual [11]

La arquitectura conceptual busca aislar las tecnologías de implementación finales. Es decir, se trata de la estructura y comportamiento de los componentes del sistema, enfatizando en los patrones de diseño [11].

La Figura 1.11 es un diagrama que representa cómo interactúan los diferentes servicios al momento de lanzar una instancia. El diagrama ofrece una idea de la arquitectura conceptual de un ambiente OpenStack común.

A continuación se presenta una descripción del diagrama para detallar cómo interactúan los servicios cuando se tiene una instancia funcionando, es necesario mencionar que la conexión entre los diferentes servicios se realiza a través de *endpoints.*

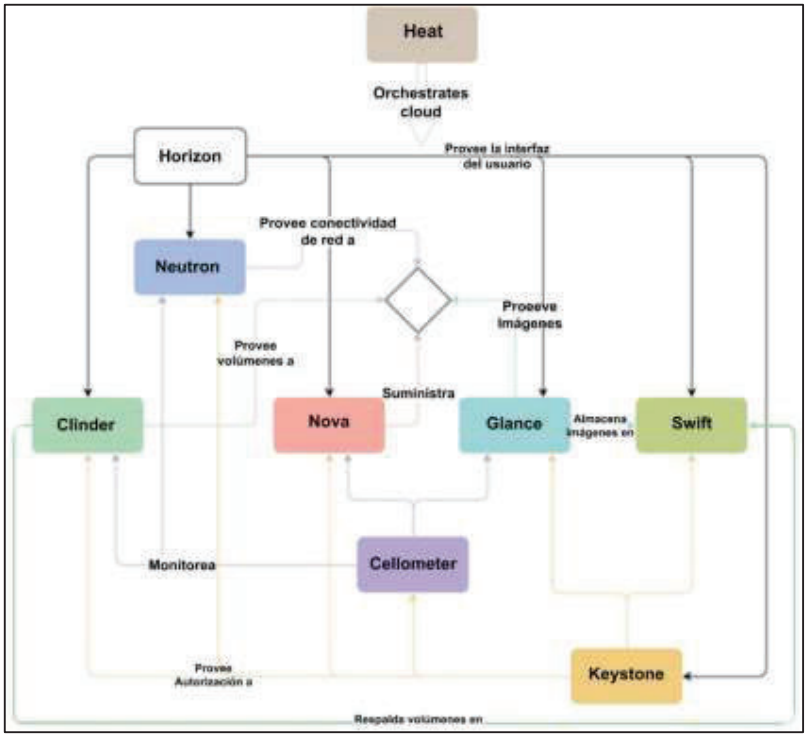

Figura 1.11. Arquitectura conceptual OpenStack [10]

Comenzando desde la parte superior se tiene Heat que es un servicio que se lo puede incorporar si se van a emplear aplicaciones de computación en la nube, Horizon es el servicio relacionado con Cinder, Neutron, Nova, Glance, Swift y Keystone, presenta una interfaz gráfica que muestra el estado actual de la nube y además permite administrarla, por lo tanto se deben tener los permisos asignados para interactuar y modificarlos de ser necesario.

Neutron es el encargado de brindar conectividad y servicios de red entre todos los servicios y la instancia, está conectado directamente con Keystone que garantiza los permisos necesarios para establecer los servicios de red.

El servicio Nova está conectado con la instancia, ya que se encarga de proveer los recursos computacionales y se conecta con Keystone para obtener todos los permisos requeridos.

El servicio Glance es el que provee la imagen a la instancia, la cual se obtiene del servicio de almacenamiento Swift donde se almacenan las imágenes y también los respaldos de los volúmenes que son manejados por Cinder.

El servicio Cinder es el encargado de proveer los volúmenes de almacenamiento a las instancias que se encuentran en ejecución.

Celiometer tiene conexión con los servicios Nova, Glance y Cinder para obtener las métricas y obtener estadísticas que pueden ser utilizadas para conocer y analizar el rendimiento de OpenStack.

Finalmente, Keystone es un servicio fundamental porque es el encargado de asignar los permisos y manejar los perfiles para la correcta administración y funcionamiento de la nube.

1.2.2.2 Arquitectura lógica [12]

Es el conjunto de conceptos y principios técnicos que permiten la operación lógica de un sistema.

En [10] se presenta un diagrama de la arquitectura lógica del software OpenStack. Este diagrama cuenta con un nivel de complejidad mayor, ya que se busca

detallar los procesos que se realizan con cada uno de los servicios para el correcto funcionamiento de la nube.

Cada servicio que presenta OpenStack tiene un conjunto de componentes, a continuación se describirá los componentes de los principales servicios para conocer la funcionalidad de cada uno de ellos.

1.2.2.2.1 Compute Service (Nova)

Servicio que permite manejar y administrar los servicios de computación en la nube. Está conformado por los siguientes componentes:

1.2.2.2.1.1 API (Aplication Programming Interface)

Conformado por `Nova-api` Service y `Nova-api-metadata` Service:

- `Nova-api` Service:

  Acepta y responde las llamadas del usuario final a la API de Compute. El servicio soporta los siguientes API de OpenStack compute, API de Amazon EC2 y API especial para administración orientada para usuarios con permisos para realizar cambios administrativos. Nova-api service se encarga de manejar ciertas políticas y la mayoría de servicios de coordinación y automatización como por ejemplo el ejecutar una instancia.

- `Nova-api-metadata` Service:

  Acepta las peticiones de *metadata*[12] enviadas por las instancias. Esta API se la emplea generalmente cuando se implementa un entorno multi-nodo que emplea nova-network.

1.2.2.2.1.2 Compute Core

[12] *Metadata*: Información estructurada que describe, explica y localiza los datos. Lo que hace más fácil el manejo y uso de los recursos de información, generalmente se lo conoce como datos de los datos o información de la información [19].

Conformado por `Nova-compute` Service, `Nova-Scheduler` Service, `Nova-conductor` Module, `Nova-cert` Module.

- `Nova-compute` Service*:*

  Es un *daemon*[13] que se encarga de crear y terminar la instancia de las máquinas virtuales a través de la API de un hipervisor, el procesamiento es complejo, el *daemon* se encarga de recibir comandos de una cola y ejecuta una serie de comandos del sistema.

- `Nova-Scheduler` Service*:*

  Recibe los requerimientos de las máquinas virtuales instanciadas que vienen desde la cola y determina en qué servidor de cómputo van a ser procesadas.

- `Nova-conductor` Module*:*

  Es el mediador en la interacción entre el servicio `nova-compute` y la base de datos, elimina el acceso directo de `nova-compute` a la base de datos de la nube [12].

- `Nova-cert` Module*:*

  Es un servidor ejecutado como *daemon* que se encarga de generar los certificados X509[14], los cuales son necesarios cuando se ocupa la API EC2.

1.2.2.2.1.3 Redes entre máquinas virtuales

- `Nova-network` Worker Daemon*:*

  Similar al trabajo que realiza el servicio `Nova-compute`, recibe las peticiones que se encuentran en cola y maneja la red entre máquinas virtuales. Realiza

[13] *Daemon*: Es un software que se ejecuta en segundo plano y realiza actividades periódicas de tal manera, que su presencia pasa desapercibida para el usuario [20].

[14] X509: Son certificados digitales estandarizados por UIT-T que permite comprobar la identidad de un usuario empleando la llave pública de infraestructura PKI (*Public Key Infrestructure*). Estos certificados son utilizados cuando se establece una comunicación que maneja información codificada, garantizando confidencialidad e integridad [15].

tareas como la configuración de las interfaces puente y maneja las reglas de IPtables[15].

1.2.2.2.1.4 Interfaz de consola

- `Nova-consoleauth` Daemon*:*

  Autoriza los *tokens* que son entregados a los usuarios por los proxy de consola, este servicio debe estar ejecutándose para que el servidor proxy funcione.

- `Nova-novncproxy` Daemon*:*

  Provee un proxy que permite el acceso a las instancias que estén ejecutándose a través de una conexión VNC[16] (*Virtual Network Computing*).

- `Nova-spicehtml5proxy` Daemon:

  Provee un proxy que permite el acceso a las instancias que estén ejecutándose a través de una conexión SPICE[17] (*Simple Protocol for Independent Computing Environments*).

- `Nova xvpvncproxy` Daemon:

  Provee un proxy que permite el acceso a las instancias que estén ejecutándose a través de una conexión VNC. Soporta clientes de OpenStack escritos en Java.

- `Nova-cert` Daemon*:*

  Genera certificados X509.

---

[15] IPtables: Es un firewall instalado por defecto en la mayoría de distribuciones oficiales de Linux, por defecto la configuración del firewall admite todo el tráfico [42].
[16] VNC: Software de código abierto que permite la compartición de escritorios de una manera remota [18].
[17] SPICE: Es un protocolo que emplea el modelo cliente-servidor para establecer conexión con escritorios de manera remota, SPICE es parte de un proyecto que busca mejorar la experiencia en el manejo de escritorios remotos tratando de disminuir el CPU (*Compute Processing Unit*) y GPU (*Graphic Processing Unit*) empleado [21].

1.2.2.2.1.5 Otros Componentes

- The Queue:

  Central por donde pasan todos los mensajes entre *daemons*, generalmente se lo implementa con RabbitMQ[18], pero también puede ser implementado como cola de mensajes AMQP[19] (*Advanced Message Queuing Protocol*), tales como ApacheQpid[20] o ZeroMQ[21].

- Base de datos SQL:

  Almacena los estados construidos y en ejecución de la infraestructura de la nube entre los que se encuentran: proyectos, instancias en uso, redes disponibles y tipos de instancias disponibles.

En la Tabla 1.3 se presenta un resumen de los componentes que forman parte del servicio Nova.

COMPUTE SERVICE (NOVA)			
COMPONENTE		**TIPO**	**DESCRIPCION**
API (APPLICATION PROGRAMMING INTERFACE)	nova-api	Servicio	Acepta y responde las llamadas del usuario final a la API de Compute.
	nova-api-metadata	Servicio	Acepta las peticiones de *metadata* solicitadas por las instancias.
COMPUTE CORE	nova-compute	Servicio	Se encarga de crear y terminar la instancia de las máquinas virtuales a través de la API de un hipervisor.
	nova-scheduler	Servicio	Recibe los requerimientos de las instancias que están en cola y determina donde van a ser procesadas.
	nova-conductor	módulo	Mediador en la interacción entre el servicio nova-compute y la base de datos.
	nova-cert	módulo	Servidor ejecutado como *daemon* que se encarga de generar los certificados X509.

[18] RabbitMQ: Es un sistema de colas eficiente y de fácil implementación que maneja el tráfico de mensajes [38].
[19] AMPQ: Es un estándar libre para pasar mensajes empresariales entre aplicaciones u organizaciones [39].
[20] ApacheQpid: Es un sistema de mensajería que permite realizar herramientas de mensajería que emplean AMPQ y soporta una variedad de lenguajes [40].
[21] ZeroMQ: Es un sistema de mensajería que permite desarrollar un medio de comunicaciones entre aplicaciones [41].

REDES PARA VM	nova-network	*Daemon*	Recibe las peticiones que se encuentran en cola y maneja la red entre VM.
INTERFAZ DE CONSOLA	nova-consoleauth	*Daemon*	Permite utilizar diferentes aplicaciones para la nube utilizando ciertos formatos.
	nova-novncproxy	*Daemon*	Provee un proxy para acceder instancias a través de una conexión VNC para los clientes escritos en Java.
	nova-spicehtml5proxy	*Daemon*	Provee un proxy para acceder instancias a través de una conexión SPICE.
	nova-xvpvncproxy	*Daemon*	Provee un proxy para acceder instancias a través de una conexión VNC.
	nova-cert	*Daemon*	Genera certificados X509.

Tabla 1.3. Resumen de componentes del servicio Nova

1.2.2.2.2 Almacenamiento de objetos (Swift)

Se encarga del almacenamiento de objetos, es altamente escalable y puede manejar un gran número de datos no estructurados a través de una API RESTful HTTP. Cuenta con los siguientes componentes:

- `Swift-proxy-server:`

  Servidor proxy que acepta las peticiones de la API Object Storage y las peticiones HTTP para cargar archivos, modificar *metadata* y crear contenedores. También brinda un listado de archivos o contenedores a los exploradores web.

- `Swift-account-server:`

  Administra las cuentas que emplean almacenamiento de objetos.

- `Swift-container-server:`

  Administra el mapeo de contenedores o carpetas, con almacenamiento de objetos.

- `Swift-objetc-server:`

  Maneja los objetos existentes tales como los archivos que se encuentran en los nodos de almacenamiento.

En la Tabla 1.4 se presenta un resumen de los componentes que forman parte del servicio Swift.

Almacenamiento de objetos (Swift)		
COMPONENTE	**TIPO**	**DESCRIPCION**
swift-proxy-server	Servicio	Servidor proxy que acepta las peticiones de la API de Object Storage.
swift-account-server	Servicio	Administra las cuentas que emplean almacenamiento de objetos.
swift-container-server.	Servicio	Administra el mapeo de contenedores o carpetas con almacenamiento de objetos.
swift-objetc-server	Servicio	Maneja los objetos que se encuentran en los nodos de almacenamiento.

Tabla 1.4 Resumen del servicio Swift

1.2.2.2.3 Almacenamiento en bloque (Cinder)

El servicio de almacenamiento en bloque ofrece el almacenamiento persistente a las máquinas virtuales y provee infraestructura para la administración de volúmenes e interacción con el servicio de cómputo para brindar volúmenes de almacenamiento para las instancias. A continuación se menciona los componentes de este servicio.

- `Cinder-api:`

  Recibe peticiones que vienen desde la API, y las direcciona a cinder-volumen para tomar acción.

- `Cinder-volume:`

  Interactúa directamente con los servicios y procesos de almacenamiento en bloque, esta interacción se la realiza a través de una cola de mensajes. El servicio cinder-volume realiza la lectura y escritura de las peticiones realizadas al almacenamiento de objetos con la finalidad de mantener el estado.

- `Cinder-scheduler Daemon:`

  Selecciona el nodo proveedor de almacenamiento óptimo en donde crear el volumen.

- `Cinder-backup Daemon:`

  Servicio que permite respaldar volúmenes para ofrecer un respaldo al proveedor de almacenamiento.

- Cola de mensajes:

  Direcciona información entre los procesos y el almacenamiento en bloque.

La Tabla 1.5 presenta un resumen de los componentes de Cinder.

Almacenamiento en bloque (Cinder)		
COMPONENTE	**TIPO**	**DESCRIPCIÓN**
cinder-api	Servicio	Recibe peticiones que vienen desde la API de Cinder
cinder-volume	Servicio	Realiza la lectura y escritura de las peticiones realizadas al almacenamiento de objetos
cinder-scheduler	Daemon	Selecciona el nodo proveedor de almacenamiento óptimo en donde crear el volumen.
cinder-backup	Daemon	Crea respaldos para el proveedor de almacenamiento.

Tabla 1.5. Resumen de componentes del servicio Cinder

1.2.2.2.4 OpenStack Networking (Neutron)

Permite crear y adjuntar a la red dispositivos que cuentan con interfaces y que son gestionados por otros servicios de OpenStack.

- `Neutron-server:`

  Acepta y direcciona las peticiones que vienen desde la API al correcto *plug-in* para que tome acción.

- *Plug-in de red y agentes OpenStack:*

  Se encargan de conectar y desconectar puertos, crear redes y subredes y proporciona el direccionamiento IP. Los agente generalmente implementados son de L3 (*layer* 3), DHCP (*Dynamic Host IP addressing*) y el agente que maneja el *plug-in.*

- Cola de mensajes:

  Permite el direccionamiento de la información entre neutron-server y los agentes.

En la Tabla 1.6 se muestra los componentes que forman parte del servicio Neutron.

Redes (Neutron)		
COMPONENTE	**TIPO**	**DESCRIPCIÓN**
neutron-server	Servicio	Acepta y direcciona las peticiones que vienen desde el API.
plug-in neutrón	plug-in	Se encarga de conectar y desconectar puertos, crear redes y subredes y proporcionar el direccionamiento IP

Tabla 1.6. Resumen de componentes del servicio Neutron

1.2.2.2.5 Servicio de Identidad (Keystone)

Se va a presentar de una manera más detallada el funcionamiento de Keystone, ya que este servicio interactúa con todos los servicios, coordina, otorga los permisos y presenta un catálogo de los servicios con sus respectivos *endpoints* para acceder a las API.

En el servicio Keystone se utilizan términos que tienen un significado específico, es por eso que se va a listar los términos con su respectivo significado.

- Usuario:

  Representación digital de una persona, sistema o servicio que utiliza los servicios de nube ofrecidos por OpenStack.

- Credenciales:

  Conjunto de datos que definen la identidad del usuario.

- Autenticación:
  Proceso mediante el cual se confirma la identidad de un usuario

- *Token:*
  Una cadena de texto alfa numérica empleada para acceder a los recursos de OpenStack y API.

- *Tenant:*

Contenedor que permite agrupar o aislar recursos, un *tenant* puede ser asociado a un cliente, cuenta, organización o proyecto.

- Rol:
  Presencia que tiene un conjunto de permisos y privilegios otorgados para realizar un conjunto de tareas específicas.

En la Figura 1.12 se muestra un diagrama en donde se puede observar el proceso que realiza el servicio de identidad Keystone para otorgar los permisos y comprobar la identidad de los usuarios. En el diagrama se presenta el caso en el que un usuario busca lanzar una instancia, este proceso consta de 7 pasos que son necesarios para cumplir esta tarea.

Paso 1: El usuario desea lanzar una instancia

Se envía las credenciales a Keystone, se crea un *token* temporal y una lista de los servicios disponibles, esta lista de servicios se conoce como catálogo.

Paso 2: Usuario solicita los *Tenants*

Keystone envía un *token* temporal junto con una lista de *tenants.*

Paso 3: *Keystone* entrega al usuario una lista de servicios

El usuario envía sus credenciales junto con los *tenants* que desea utilizar, Keystone retorna una lista con los servicios disponibles y le entrega el *token* para los *tenants* solicitados. Con esta información el usuario determina el *endpoint* para lanzar la instancia y envía el *token* ha dicho *endpoint*.

Paso 4: El servicio verifica los *token* del usuario

Se envía los *token* a Keystone para confirmar que son válidos y autorizar el acceso al servicio.

Paso 5: Keystone envía información adicional y el *token*

Keystone envía al servicio correspondiente, información del usuario y confirma que tiene autorización para usar el servicio y que el *token* coincide con la petición realizada.

Paso 6: El servicio ejecuta la petición

El servicio crea una nueva instancia.

Paso 7: El servicio reporta los resultados al usuario

El servicio le envía al usuario información que indica que se ha creado con éxito la instancia y le indica que puede acceder.

1.2.2.2.6 Servicio de imágenes (Glance)

El servicio de imágenes es una parte fundamental para la implementación de IaaS, dado que es el encargado de entregar las imágenes o discos para la creación de infraestructura en la nube. A continuación se presentan los componentes que forman parte de este servicio.

- `Glance-api`
  Acepta las peticiones que vienen desde la API para explorar recuperar y almacenar imágenes.

- *`Glance-registry`*
  Almacena, procesa y recupera *metadata* de las imágenes.

Con la arquitectura conceptual y lógica presentada, junto con la descripción de los componentes que forman parte de OpenStack y su función, se puede tener una idea más clara de cómo funciona e interactúan los componentes entre sí. OpenStack es una herramienta muy poderosa con altas prestaciones para desarrollar una nube según las necesidades del usuario, además cuenta con la documentación necesaria para desarrollar y comprender su funcionamiento, lo que permite centrar los esfuerzos en desarrollar una plataforma adecuada a los requerimientos.

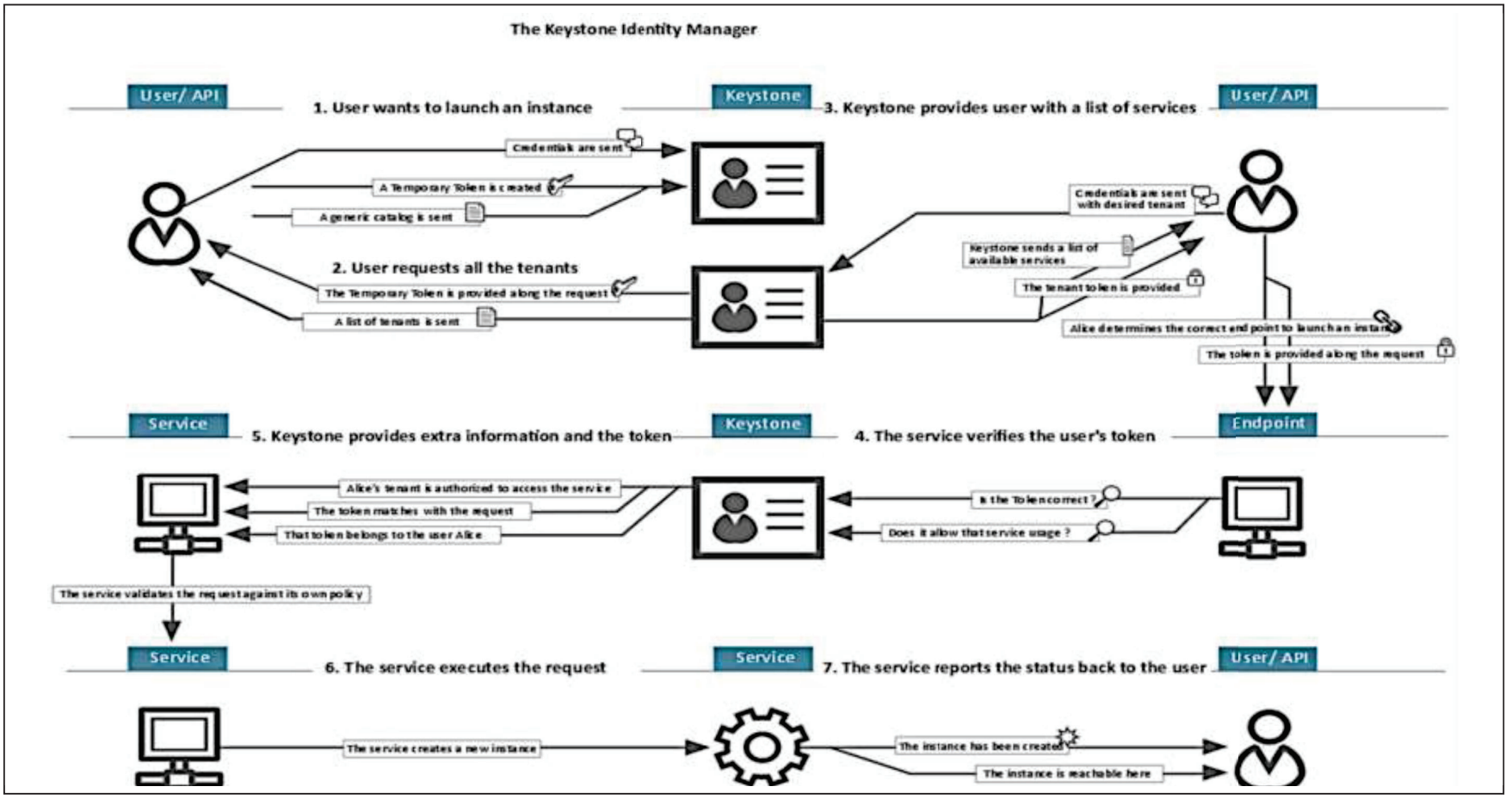

Figura 1.12. Proceso del servicio Keystone al lanzar una instancia [12]

CAPÍTULO II

2. IMPLEMENTACIÓN DE LA NUBE

En este capítulo se presenta la implementación de la nube para el Proyecto de Titulación, se inicia con la descripción del entorno de desarrollo y los dispositivos utilizados, a continuación se explica el procedimiento realizado para la implementación y se mencionan los pasos para iniciar una instancia.

2.1. ENTORNO DE DESARROLLO

El software OpenStack cuenta con distintas alternativas para su implementación, en función de las necesidades del usuario y su aplicación. La alternativa seleccionada para el desarrollo del Proyecto es el modelo multi-nodo con los nodos controlador, cómputo y red, ya que este modelo cuenta con las prestaciones necesarias para realizar el despliegue de la nube. Cada uno de los nodos cumple una función específica y cuenta con un conjunto de servicios y componentes que deben interactuar entre sí para obtener el resultado deseado.

El nodo controlador es el cerebro de este modelo considerando que aloja servicios como: el servidor del sistema de mensajes, la base de datos, el servicio de identidad y el almacenamiento de imágenes, los mismos que tienen relación directa con los servicios de los otros nodos.

El nodo de red es el encargado de manejar los servicios de red y que todos los servicios de OpenStack y sus respectivos componentes tengan conexión a ella, en este nodo se instala neutron-server, *plug-ins*, agentes y el sistema de cola de mensajes, que permite dar servicio como DHCP, manejar los servicios de capa 3 y otros agentes que permiten interactuar con *plug-ins* de fabricantes específicos.

El nodo de cómputo aloja y administra las instancias, para esto se debe instalar varios componentes que permiten la interacción entre este nodo y los servicios de

autenticación, imágenes, el servidor de imágenes, la interfaz gráfica y la base de datos.

En la Tabla 2.1 se listan los servicios de OpenStack que alojan cada uno de los nodos. En el modelo multi-nodo, este modelo cuenta con tres nodos que van a ser implementados en tres servidores independientes.

Para el desarrollo del Proyecto se cuenta con un servidor con las características que se muestran en la Tabla 2.2.

ENTORNO MULTI-NODO	
NODO	**COMPONETE**
CONTROLADOR	*Identity Service*
	Network Time Protocol (NTP)
	Image Service
	Servicio de base de datos SQL
	Cola de mensajes
	Dashboard
	Administración de red
	Administración de cómputo
	Plug-in de red ML2
CÓMPUTO	Hipervisor KVM
	Open vSwitch
	Servicio de cómputo
	Agente de red Open vSwitch
	Plug-in de red ML2
RED	Open vSwitch
	Agente DCHP
	Agente de Metadatos de red
	Plug-in de red ML2
	Agente de red Open vSwitch
	Agente de red L3

Tabla 2.1. Ambiente de desarrollo Multi-nodo [10]

En el servidor se encuentra instalado el hipervisor ESXi desarrollado por VMware, instalado directamente sobre el hardware y que maneja un *kernel*[22] diseñado por

[22] *Kernel*: Programa que es el núcleo del sistema operativo, es el encargado de controlar y manejar todo lo que ocurre en el sistema operativo [53].

VMware, lo que le permite al hipervisor tener un control absoluto sobre los recursos computacionales y emplear así el modelo de virtualización total.

SERVIDOR	
Marca	Hewlett-Packard (HP)
Modelo	ProLiant DL360e Gen8 E5-2403v2
Procesador	Intel® Xeon® E5-2403 v2 (4 core, 1.8 GHz, 10MB, 80W)
RAM	8 GB
Capacidad Disco Duro	1TB

Tabla 2.2. Características del servidor empleado [54]

Sobre el entorno virtualizado se van a crear tres máquinas virtuales, las cuales van ser los nodos para implementar OpenStack. La nube estará conformada por tres nodos: controlador, cómputo y red. La documentación de OpenStack recomienda trabajar sobre el sistema operativo Ubuntu o Red Hat, para este Proyecto se ha seleccionado el sistema operativo Ubuntu, considerando que tanto el software como la distribución son libres.

Los recursos de hardware disponibles se asignaron según las siguientes consideraciones, el nodo controlador es el cerebro del ambiente multi-nodo y maneja varios servicios que deben estar en continua interacción con los demás nodos, por lo que necesita una buena cantidad de memoria RAM, es por eso que se asigna 3GB de los 8GB disponibles, en cuanto al almacenamiento se debe considerar que en el controlador se encuentra el servicio de imágenes, lo cual requiere un espacio considerable.

Para el nodo de cómputo se debe considerar que hospedará y administrará los sistemas de cómputo, lo que implica que las prestaciones de hardware de este nodo van a disponer las instancias en la nube, por lo tanto para este nodo se asignó 3GB de los 5GB disponibles y finalmente para el nodo de red se debe considerar que las interfaces de red trabajen correctamente y que el procesamiento sea aceptable para que los recursos de red en la nube funcionen

adecuadamente, es por eso que se ha asignado los 2GB restantes siendo este el nodo con menor memoria RAM. La capacidad de almacenamiento de los nodos fue asignada en función de los recursos disponibles en el servidor, el nodo de red cuenta con un disco duro de mayor capacidad, para aprovechar los recursos restantes disponibles en el servidor. En la Tabla 2.3 se presentan las características y la función de cada una de las máquinas virtuales creadas.

El número de interfaces de red en cada uno de los nodos está definido en función de la arquitectura multi-nodo de OpenStack a ser utilizada. En el nodo controlador se maneja una interfaz de red que va a estar conectada a la red de administración, en el nodo de cómputo se necesita dos interfaces de red, ya que este nodo debe estar conectado a la red de administración y a la red para túneles y el nodo de red debe contar con tres interfaces de red para tener conexión a la red de administración, red de túneles y red externa.

CARACTERÍSTICAS DE NODOS		
NODO	**TIPO**	**DESCRIPCIÓN**
CONTROLADOR	Sistema Operativo	Ubuntu 14.04 LTS
	RAM	3GB
	Capacidad Disco Duro	150 GB
	Número de Interfaces de red	1
CÓMPUTO	Sistema Operativo	Ubuntu 14.04 LTS
	RAM	3 GB
	Capacidad Disco Duro	150 GB
	Número de Interfaces de red	2
RED	Sistema Operativo	Ubuntu 14.04 LTS
	RAM	2GB
	Capacidad Disco Duro	160 GB
	Número de Interfaces de red	3

Tabla 2.3. Características de Nodos

Para un entorno multi-nodo implementado con la versión Juno se recomienda que los nodos cuenten con los requerimientos de hardware presentados en la Tabla 2.4.

REQUERIMIENTOS DE HARDWARE DE OPENSTACK		
NODO	**TIPO**	**DESCRIPCIÓN**
CONTROLADOR	RAM	8 GB
	Capacidad Disco Duro	100 GB
	Número de Interfaces de red	1
CÓMPUTO	RAM	8+ GB
	Capacidad Disco Duro	100+ GB
	Número de Interfaces de red	2
RED	RAM	2 GB
	Capacidad Disco Duro	50+ GB
	Número de Interfaces de red	3

Tabla 2.4. Requerimientos de hardware para ambiente multi-nodo [10]

2.2 CONFIGURACIÓN E INSTALACIÓN

En cada uno de los nodos se debe realizar la configuración e instalación del Software OpenStack, usando la versión Juno, para lo cual se recomienda seguir la documentación oficial de OpenStack disponible en [10].

Para el correcto funcionamiento del software OpenStack es necesaria la instalación de diferentes servicios, los cuales cumplen una determinada función.

2.2.1 CONFIGURACIÓN DE RED

Lo primero que se debe configurar son las interfaces de red de cada uno de los nodos. Es necesario establecer tres redes, una red para administración, una red para túneles y otra red externa para tener comunicación con el mundo exterior y acceso al Internet. En la Figura 2.1 se muestra el diagrama de red y las interfaces de red que tiene cada uno de los nodos con sus respectivas direcciones IP.

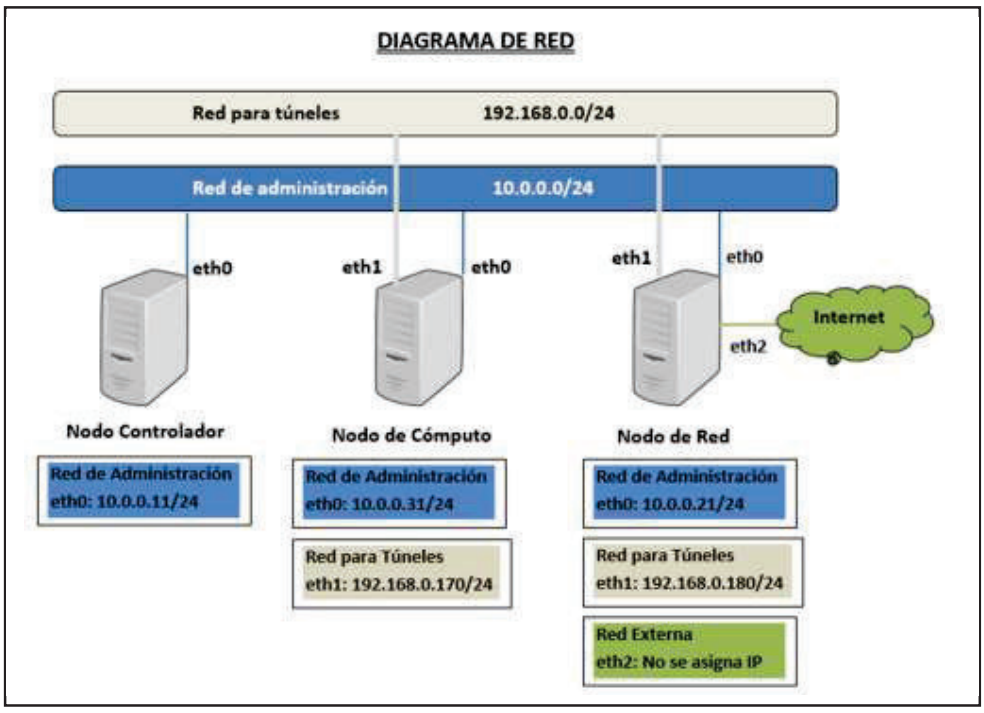

Figura 2.1. Diagrama de Red de la Nube

Red de administración

Es la red que está diseñada para que los componentes de OpenStack se comuniquen a través de ella, esta red no debería ser visible desde el exterior por motivos de seguridad.

Red para túneles

Es la red que transporta el tráfico generado por las instancias, las cuales se encuentran en el nodo de cómputo y el de red, y permite que tengan conectividad con los demás elementos de la nube.

Red externa

Está diseñada para que las instancias tengan acceso a Internet a través de ella.

Las direcciones IP asignadas en cada nodo deben ser estáticas, lo cual evitará tener posibles problemas o errores durante la instalación, además es necesario desactivar todas las actualizaciones automáticas que puedan modificar la configuración especificada.

En el nodo de red es importante la configuración de la interfaz de red para la conexión con la red externa. Esta interfaz debe ser configurada en modo IP no numerada con configuración manual, lo cual permite que la interfaz sea configurada por programas externos, en este caso OpenStack. Para conocer los detalles de cómo configurar la interfaz en modo IP no numerada se recomienda ver el Anexo A página A-1.

Una vez establecidas las direcciones IP de cada nodo, se debe proceder a configurar la resolución de nombres, para que los diferentes nodos puedan establecer comunicación entre ellos haciendo uso de un nombre, en lugar de una dirección IP, lo cual simplificará el proceso de configuración. Para comprobar que la configuración ha sido exitosa se debe realizar pruebas de conectividad. Primero es necesario comprobar que exista conectividad con Internet, para lo cual se puede usar el Comando 2.1. Luego debe asegurarse que exista conectividad y resolución de nombres con los nodos, para lo cual se puede emplear el Comando 2.2.

```
# ping -c 5 www.google.com
```

Comando 2.1. Prueba de conectividad a Internet

```
 # ping -c 5 network
 # ping -c 5 controller
 # ping -c 5 compute
```

Comando 2.2. Prueba de conectividad entre nodos

Si se obtiene una respuesta exitosa en cada uno de los nodos, la configuración de red es adecuada y se puede proceder. Un ejemplo del resultado esperado en las pruebas se presenta en el Comando 2.3.

```
# ping -c 5 controller
PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263
ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202
ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203
ms
```

Comando 2.3. Resultado Prueba de la conectividad

En el Anexo A, se presenta con mayor detalle los pasos recomendados para la configuración de red y la resolución de nombres.

2.2.2 CONFIGURACIÓN DE NETWORK TIME PROTOCOL (NTP)

El protocolo NTP permite que los nodos estén sincronizados para que los componentes trabajen adecuadamente entre sí. Se debe configurar el protocolo NTP de la siguiente forma: el nodo controlador se sincronizará con servidores NTP externos y los demás nodos lo harán con el nodo controlador.

Si los nodos no se encuentran sincronizados, los servicios que se encuentran en diferentes nodos pueden no interactuar adecuadamente, produciendo un error debido a la diferencia horaria en los nodos.

2.2.3 INSTALACIÓN DEL SOFTWARE OPENSTACK

Para la descarga de los paquetes se recomienda usar la última versión para aprovechar sus prestaciones, en este Proyecto de Titulación se usa la versión Juno. Como ya se ha mencionado, el software OpenStack es un conjunto de componentes, los cuales deben ser instalados en determinado orden y en diferentes nodos. Para garantizar que el sistema operativo de cada uno de los servidores soporte la última versión del Software OpenStack se debe instalar el paquete `ubuntu-cloud-keyring` en cada nodo, el cual se encarga de instalar cualquier parche y comprobar que el sistema operativo sea apto.

Como pre-requisito se deben instalar herramientas que permitan la correcta interacción entre los servicio de OpenStack, estas herramientas son la base de datos y el servicio de mensajes.

2.2.3.1 Base de Datos

La base de datos que va a ser instalada es MariaDB, una base de datos compatible con SQL y de software libre.

La base de datos debe ser instalada en el nodo controlador y en cada uno de los nodos se debe instalar un cliente, la base de datos se encarga de almacenar información que se maneja en cada uno de los servicios, es importante configurar la dirección IP mediante la cual la base de datos recibe las peticiones y debe ser la dirección de la red de administración, que para este Proyecto corresponde a la dirección IP 10.0.0.11, dirección de la red de administración del nodo controlador.

2.2.3.2 Servidor de Mensajes

OpenStack emplea un servicio de mensajería para intercambiar información del estado de los servicios y coordinar las operaciones entre ellos, el servicio de mensajería que se empleará es RabbitMQ, el cual trabaja con toda la distribución de OpenStack.

El servidor de mensajes se instalará en el nodo controlador. Considerando que es un entorno de desarrollo, se recomienda utilizar la cuenta de usuario `guest` que viene por defecto, a la cual se debe cambiar la contraseña por motivos de seguridad.

2.2.3.3 Instalación de los servicios OpenStack

Una vez instalada tanto la base de datos como el servicio de mensajería, se puede iniciar la instalación y configuración de los servicios de OpenStack.

2.2.3.3.1 Servicio de identidad (Keystone)

Keystone es el servicio encargado de llevar un “registro” de OpenStack, se deben registrar los usuarios, los servicios y la ubicación de cada servicio (*endpoint*). Cuando un usuario desea interactuar con el software, lo primero que debe hacer es enviar una petición al servicio Keystone, el cual verifica si está registrado y que permisos tiene, para responder con un catálogo donde se encuentran todos los servicios y su ubicación, este es un servicio fundamental en OpenStack y se encuentra instalado en el nodo controlador, es el primer servicio en ser instalado

porque todos los servicios que van a ser instalados desde este punto en adelante se deben registrar en el catálogo que maneja Keystone, al igual que los usuarios.

Nodo Controlador

En el nodo controlador, primero debe instalarse Keystone, servicio que se maneja mediante *tokens* para autenticar a un usuario y otorgarle los permisos correspondientes. Todos los *token* generados durante la operación de OpenStack expiran después de cierto tiempo debido a que tienen un ciclo de vida definido. Es necesario indicar que los *tokens*, por defecto, son almacenados indefinidamente en la base de datos, lo cual incrementa el tamaño de la base de datos considerablemente y puede influir en el rendimiento del servicio, por lo que para evitar esto, se recomienda crear una actividad periódica con la ayuda de `cron`[23] para descartar los *tokens* expirados cada hora. En el Anexo B, se presenta en detalle el comando para configurar la actividad periódica.

La configuración en el servicio de Keystone debe iniciar con la creación de proyectos (*tenant*), rol y usuarios. Se van a establecer dos proyectos, roles y usuarios, estos son `admin`, el cual cuenta con permisos y privilegios de administración para acceder a todos los servicios y modificarlos y `demo` que cuenta con los privilegios de un usuario normal. También se crea un proyecto (*tenant*) para los servicios del software OpenStack que se llamará `service`, en el que se registrarán los servicios disponibles y su respectivo *endpoint*.

Para la creación inicial de proyectos, roles, usuarios, y el registro de *endpoints* es necesaria la creación de un *token* que autentifique al usuario actual, el mismo que en la base de datos no tiene registrada ninguna información, razón por la cual se debe generar un *token* aleatorio y utilizarlo para realizar esta tarea. Este proceso se muestra en el Anexo B.

Con los proyectos, usuarios y roles creados, se debe establecer una relación entre ellos para que los permisos puedan ser asignados en función de los tres

[23] Cron: comando del sistema operativo Unix que permite programar la ejecución periódica de determinados comandos [55].

parámetros. Los usuarios pueden ser miembros de uno o más proyectos y el rol define qué acciones puede realizar cada usuario.

Si el servicio Keystone se ha instalado y configurado adecuadamente, se puede comprobar su correcto funcionamiento mediante los comandos que se presentarán a continuación. A manera de ejemplo se muestra su respectiva respuesta, cabe indicar que los identificadores van a ser distintos dado que son generados aleatoriamente.

En el Comando 2.4 se presenta el comando requerido para listar los usuarios registrados en el servicio Keystone, así como su resultado. Se puede apreciar que se emplea como *tenant* y como usuario la cuenta `admin`. En el comando en mención, debe cambiarse ADMIN_PASS por la contraseña del usuario `admin` del servicio Keystone.

En el Comando 2.5 se presenta tanto el comando requerido para comprobar que con un usuario normal, no es posible ejecutar el listado de usuarios, así como su respuesta. Para la ejecución del mencionado comando, se empleó como *tenant* y cuenta de usuario a `demo`. El comando falla puesto que no se permite la ejecución de comandos exclusivos para el usuario administrador a través del CLI. En el comando debe reemplazarse DEMO_PASS por la contraseña de `demo` del servicio Keystone.

```
$ keystone --os-tenant-name admin --os-username admin --os-
                    password ADMIN_PASS \
  --os-auth-url http://controller:35357/v2.0 user-list
+----------------------------------+---------+---------+---------------------------+
|                id                |   name  | enabled|          email            |
+----------------------------------+---------+---------+---------------------------+
| ea8c352d253443118041c9c8b8416040 | admin   |  True  | admin@example.com |
| 7004dfa0dda84d63aef81cf7f100af01 |  demo   |  True  | demo@example.com  |
+----------------------------------+---------+---------+---------------------------+
```

Comando 2.4. Listar usuarios como usuario y *tenant* `admin`

```
$ keystone --os-tenant-name demo --os-username demo --os- password
DEMO_PASS \
--os-auth-url http://controller:35357/v2.0 user-list

You are not authorized to perform the requested action, admin_required.
(HTTP 403)
```

Comando 2.5. Listar usuarios como usuario y *tenant* `demo`

Durante la instalación del servicio Keystone se emplean una combinación de variables y comandos que permiten interactuar con el Servicio de Identidad a través del cliente Keystone. Para optimizar el proceso es posible crear *scripts* soportados por OpenStack llamados archivos OpenRC[24]. Los *scripts* que van a ser creados realizan un proceso de automatización de la autenticación de los usuarios `demo` y `admin` en el servicio Keystone, ya que para la instalación y administración de los servicios, los usuarios deben autenticarse. Los *scripts* mencionados se incluyen en el Anexo B, página B-11.

2.2.3.3.2 Servicio de Imágenes (Glance)

El Servicio de Imágenes (Glance) permite a los usuarios registrar, recuperar y explorar imágenes de máquinas virtuales, acepta peticiones de imágenes, discos y metadatos de las imágenes generadas por la API.

Nodo Controlador

Para la instalación del servicio de imágenes se debe iniciar con la creación de una base de datos que tenga el mismo nombre del servicio (Glance), después se deben referenciar las credenciales del usuario `admin` para tener los permisos necesarios para registrar el servicio Glance y su ubicación en Keystone y de esta manera pueda ser agregado al catálogo de los servicios disponibles. Con estos cambios realizados se deben modificar los archivos de configuración para registrar la cadena de conexión a la base de datos, URL del nodo controlador para que interactúen con los demás servicios instalados en este nodo.

Este proceso de configuración debe ser realizado en los dos servicios que conforman Glance: `glance-registry` y `glance-api`.

La instalación y configuración del Servicio de Imágenes (Glance) se presenta con mayor detalle en el Anexo B en la página B-11.

[24] OpenRC: Permite configurar las credenciales de inicio de sesión con las herramientas de la línea de comandos de OpenStack [43].

2.2.3.3.3 Servicio de Cómputo (Nova)

El Servicio de Cómputo de OpenStack permite alojar y administrar los sistemas de computación, y es una parte esencial para manejar IaaS. Se debe realizar la configuración e instalación de paquetes en el nodo controlador y en el nodo de cómputo.

Nodo Controlador

El servicio de cómputo cuenta con un nodo específico para manejar y administrar los recursos de cómputo, sin embargo es necesario realizar ciertas actividades en el nodo controlador. Dentro de estas actividades se puede mencionar a la siguiente: crear la base de datos, registrar el servicio y el *endpoint* de Nova en Keystone e instalar y configurar los servicios `nova-api`, `nova-cert`, `nova-consoleauth`, `nova-scheduler`, `nova-conductor` y `nova-novncproxy`. Estos servicios se instalan en el nodo controlador, y en estos servicios se deben configurar los parámetros de la base de datos, el servicio Keystone y el sistema de mensajes.

Nodo de Cómputo

En el nodo de cómputo se debe instalar y configurar el servicio `nova-compute`. Previo a su instalación, se recomienda verificar si el hardware sobre el que está instalado el nodo de cómputo permite aceleración de hardware para que las instancias tengan un mejor desempeño. Para verificar si el nodo de cómputo soporta aceleración de hardware, se emplea el Comando 2.6, el cual permite verificar la información del CPU. En este comando, se buscan dos posibles banderas: *vmx* o *svm* para Intel[25] o AMD[26] respectivamente, las cuales indican si el procesador soporta la tecnología de virtualización.

[25] Intel: Compañía que diseña y fabrica dispositivos esenciales en el área de la computación, muy conocida por sus procesadores.
[26] AMD (*Advanced Micro Device):* Compañía que diseña y fabrica dispositivos para computación, es muy conocida por el procesador que fabrica.

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

Comando 2.6 Aceleración de hardware

Este comando retorna un valor, si el valor es uno o mayor, el nodo soporta aceleración de hardware; si el valor que retorna es cero no soporta aceleración de hardware. En el ambiente de desarrollo se obtuvo un valor de cero, lo que implica que no soporta aceleración y por lo tanto se escogió configurar el uso de QEMU[27], lo cual se indica en el Anexo B.

2.2.3.3.4 Servicio de Red (Neutron)

El Servicio de Red Neutron administra y provee los recursos de red para las instancias, presta las funcionalidades básicas de red y brinda la posibilidad de utilizar *plug-ins* para una variedad de productos que soportan redes virtuales.

Nodo Controlador

El servicio de red tiene un nodo específico, pero es necesario instalar ciertos paquetes en el nodo controlador y realizar configuraciones como: la creación de la base de datos, el registro del servicio y el *endpoint* en Keystone. Por tanto, se debe instalar los paquetes `neutrón-server`, `neutrón-plugin-ml2` y `python-neutronclient` en el controlador. Los paquetes instalados permiten que el nodo controlador y los servicios instalados en él, interactúen con los servicios de red y los servicios de los otros nodos.

Al instalar el nodo de red se deben realizar ciertos cambios en la configuración del servicio de cómputo en el nodo controlador, ya que la configuración por defecto es con *Legacy networking* que es un servicio de red con prestaciones básicas que ofrece el servicio Nova, se debe cambiar esta configuración para que utilice Neutron como servicio de red.

[27] QEMU: Es un simulador genérico y de código abierto que también permite virtualizar, cuando se lo usa para virtualizar tiene un desempeño muy similar al equipo nativo dado que ejecuta el código del SO invitado directamente sobre el CPU del anfitrión.

Nodo de Red

En el nodo de red se deben instalar los paquetes `neutron-plugin-ml2`, `neutron-plugin-openvswitch-agent`, `neutron-l3-agent` y `neutron-dhcp-agent`, los cuales permiten al nodo de red manejar los principales servicios tales como: asignación de direcciones IP mediante DHCP, enrutamiento de capa 3, realizar NAT[28] (*Network Address Translation*) para que las instancias tengan acceso a Internet, creación de infraestructura de red.

El agente OVS (Open vSwitch) es el encargado de manejar el tráfico de capa 2 junto con el *plug-in* `ml2`, que permite que varios servicios de capa dos como VLAN o GRE trabajen simultáneamente.

El nodo de red utiliza una red de túneles para establecer comunicación entre el nodo de cómputo y el nodo red, por la cual se transfiere el tráfico generado desde o hacia las instancias, desde el Internet y los servicios instalados en ese nodo, este túnel se establece mediante el protocolo GRE (*Generic Routing Encapsulation*), por lo que, durante la configuración del nodo de red es importante tener en cuenta que el protocolo GRE para túneles incluye un encabezado que incrementa la sobrecarga y disminuye el espacio para los datos en los paquetes.

Las instancias desconocen la existencia de esta red virtual y por tanto tratarán de enviar paquetes utilizando el tamaño máximo de la trama Ethernet, es decir 1500 bytes como MTU[29] (*Maximum Transmission Unit*). Las redes IP contienen el mecanismo PMTUD (*path MTU discovery*) para detectar el tamaño de la MTU extremo a extremo y ajustar el tamaño del paquete en función de eso. Algunos sistemas operativos y redes bloquean o no soportan PMTUD causando una degradación en el desempeño y fallas de conectividad, para prevenir esto se

[28] NAT: Proceso mediante el cual se modifica la dirección de origen o de destino de un paquete en la cabecera del paquete IP mientras el paquete está en tránsito [52].
[29] MTU: Unidad máxima de transmisión, término que generalmente se emplea en capa 2 para hacer referencia al tamaño máximo de la trama que puede ser transmitida sin pérdida de datos [49].

deben habilitar las tramas jumbo[30] en la red física, permitiendo tramas de hasta 9000 bytes, con lo que se evitará los problemas de sobrecarga generada por el protocolo GRE. El inconveniente de emplear las tramas jumbo es que se debe tener acceso a la administración de la red y a los dispositivos de red.

Otra alternativa, considerando todos estos inconvenientes, consiste en prevenir los problemas por el tamaño del MTU, disminuyendo el valor del MTU en cada una de las instancias, en función de la sobrecarga de GRE. Se debe mencionar que para determinar el tamaño apropiado se debe realizar varios experimentos, tomando como como referencia el valor indicado en la documentación de OpenStack, en donde se recomienda establecer un MTU de 1454 bytes.

Cabe indicar que se puede configurar el servicio DHCP para asignar direcciones IP y para establecer el MTU con el que deben trabajar las instancias, para realizar esta configuración se debe seguir el procedimiento que se encuentra detallado en el Anexo B.

Nodo de Cómputo

En el nodo de cómputo es necesario realizar ciertas configuraciones para que trabaje con el nodo de red y además se debe instalar los paquetes `neutron-plugin-ml2` y `neutron-plugin-openvswitch-agent`, paquetes que son necesarios dado que el nodo de cómputo es el que maneja la conectividad y los grupos de seguridad de las instancias.

Creación de la red inicial

Con todos los paquetes indicados en párrafos anteriores instalados y que OpenStack requiere para la implementación de redes virtuales, es posible crear la infraestructura de red virtual necesaria para establecer la conexión entre las instancias y el Internet o la red externa, en la Figura 2.2 se representa el flujo de tráfico desde las instancias al Internet y la conexión con los servicios de red.

[30] Tramas Jumbo: Es una trama que tiene un MTU mayor a 1500, el valor de datos que puede tener una trama está definido por el fabricante del hardware, es por eso que no se puede dar un valor exacto para las tramas Jumbo [49].

El flujo va a ser analizado con mayor detalle ya que permite comprender cómo funciona la red interna y qué función cumple cada elemento para que las instancias accedan al Internet.

El nodo de cómputo es el encargado de manejar los recursos computacionales y las instancias que se crean, en la parte de red debe manejar los grupos de seguridad y debe proveer la conectividad a las instancias creadas. Se asume que se dispone de una primera instancia cuyo nombre es `demo-instance1`, y que el tráfico de esta instancia va directamente a un filtro (un filtro no es más que un grupo de seguridad) donde se encuentra definido un conjunto de reglas mediante el cual se maneja el tráfico IP, estas reglas pueden ser modificadas según las necesidades del usuario. Una vez que el tráfico pasa el filtro entra a la zona de Open vSwitch, que es la que se encarga de crear la infraestructura de red virtual necesaria, en este caso se crea un `br-int` (*bridge*[31] de integración) que cumple la función de integrar la diferentes instancias que se encuentran en el mismo proyecto (*tenant*) y agrega o retira las etiquetas de VLAN[32] (Virtual Local Area *Network*), etiquetas que permiten dividir el tráfico entre las distintas redes.

Por otro lado, un `br-tun` (*bridge* de túneles), se encarga de encapsular el tráfico que viene etiquetado con un ID de VLAN desde `br-int` mediante el protocolo GRE para transportarlo a través del túnel establecido entre el nodo de cómputo y el nodo de red. En el lado del nodo de red el `br-tun` realiza el proceso contrario, desencapsula los paquetes GRE y envía al `br-int` para que etiquete o retire las etiquetas de VLAN y proceda a determinar a qué red debe ir la información.

En el nodo de red la zona de Open vSwitch es más amplia, cuenta con un `br-tun`, el cual ya se conoce su función, un `br-ex` (*external bridge*), que se encarga de conectar la red externa con el `br-int`. El `br-int` es el que maneja las dos redes principales del nodo de cómputo. La primera red se denomina `demo-net`, la cual debe crearse con el objetivo de que todas las instancias creadas se conecten a esta red, convirtiéndose en la LAN de las instancias; la asignación de

[31] *Bridge*: Software o Hardware que permite la conexión de dos o más segmentos de red [51].
[32] VLAN: Es un grupo de estaciones con determinados requerimientos en común que tiene conexión entre sí, independientemente de la ubicación física [50].

direcciones IP en esta red es gestionada por el servicio DHCP y un *gateway* para tener conexión con la red externa y salir al mundo exterior. La segunda red se denomina `ext-net`, y corresponde a la red externa que permite que las instancias tengan acceso al Internet mediante dos mecanismos de NAT: SNAT[33] (*Source Network Address Translation*) y DNAT[34] (*Destination Network Address Translation*). Para la conexión de dos redes es necesario un router (`demo-router`), en este caso entre la red interna y la red externa.

Cuando una instancia es creada se le asigna una dirección IP de manera automática, la cual tiene acceso al Internet a través de NAT, otra alternativa de tener acceso al Internet desde una instancia es asignar direcciones IP flotantes, que consiste en un *pool* de direcciones preestablecido durante la instalación que tiene acceso al internet.

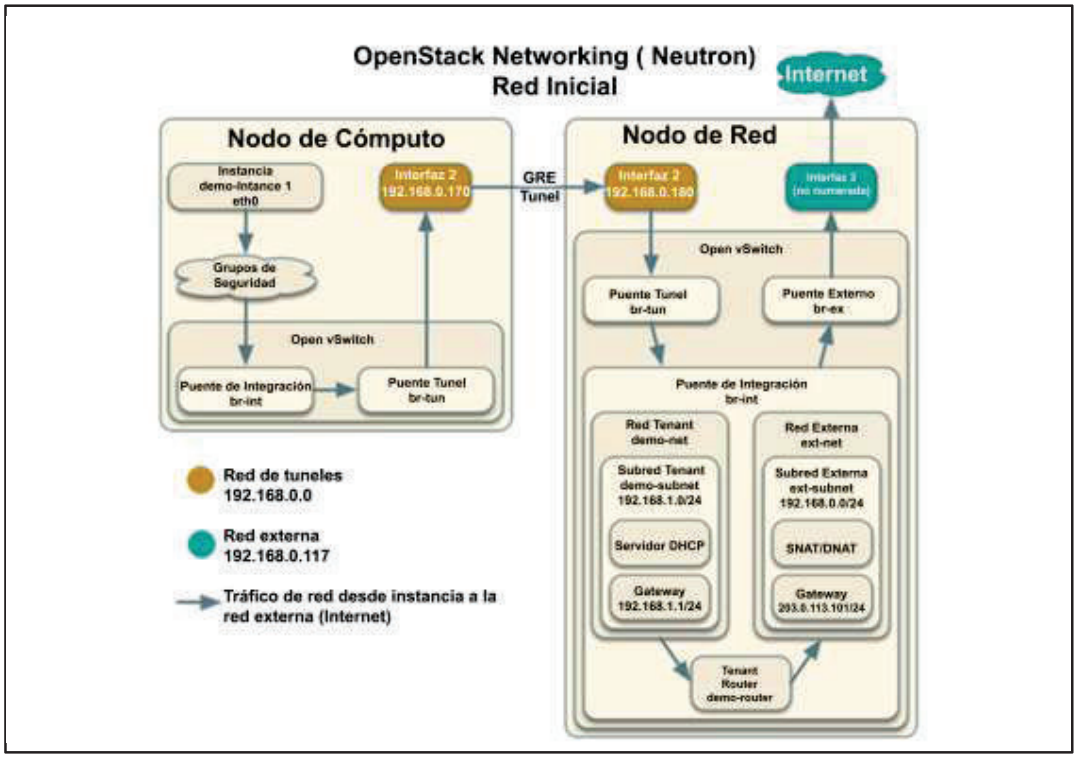

Figura 2.2. Red inicial de OpenStack

[33] SNAT: El *router* encargado del proceso de NAT modifica la dirección del que envía el paquete IP, generalmente se utiliza para que dispositivos con direcciones privadas se conecten con el Internet [52].
[34] DNAT: El *router* encargado del proceso de NAT modifica la dirección de destino del paquete IP [52].

El *tenant* `admin` es propietario de la red externa dado que esta red solo debe ser modificada por el administrador, se podría hacer una similitud al decir que la red `ext-net` es la WAN de la nube, ya que esta es la que se encarga de brindar acceso al internet y conexión al exterior. Por otro lado el *tenant* demo es propietario de la red `demo-net` que es como la LAN, a la cual deben estar conectadas las instancias creadas por los usuarios. Para la conexión de la red LAN con la WAN se debe utilizar un router que en este caso es el *tenant* `demo-router`.

Las redes `demo-net` y `ext-net` tienen creadas las subredes `demo-subnet` y `ext-subnet` respectivamente, con el objetivo de segmentar la red permitiendo que las instancias puedan estar conectadas a diferentes segmentos de red tanto en la red interna como externa.

Lo mencionado permite comprender con mayor claridad el funcionamiento de los componentes de red que maneja OpenStack, esta red inicial establece la conectividad de las instancias a los servicios de red y al Internet. El proceso completo de creación de la red inicial se presenta en el Anexo B en la página B-51.

En la Figura 2.3 se muestra el resultado que presenta OpenStack en su interfaz web, donde se puede observar que las dos redes han sido creadas con los parámetros mencionados.

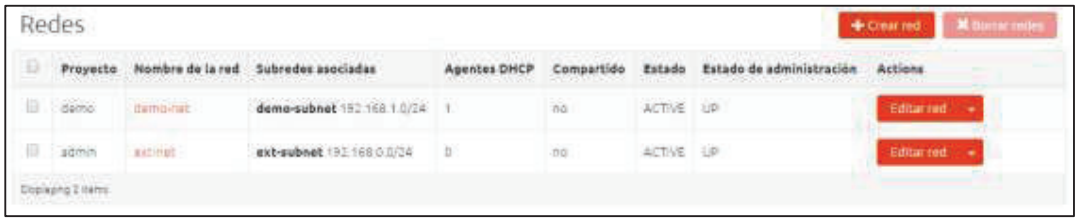

Figura 2.3. Red inicial creada

2.2.3.3.5 Servicio de Dashboard (Horizon)

El Servicio de *Dashboard* (Horizon) consta de una interfaz web que permite a los administradores y usuarios manejar e interactuar con los recursos y servicios de OpenStack. La instalación de este servicio se la realiza en el nodo controlador,

para esto se deben instalar los paquetes `openstack-dashboard`, `apache2`, `libapache2-mod-wsgi`, `memcached` y `python-memcache`.

Es un servicio opcional que permite administrar los recursos de OpenStack mediante una interfaz gráfica, este servicio puede ser de mucha utilidad cuando se va a ofrecer el servicio de *cloud computing* para que los usuarios que no tienen un conocimiento de OpenStack puedan administrar sus recursos mediante una interfaz gráfica.

2.2.4 LANZAR UNA INSTANCIA EN OPENSTACK

Para lanzar una instancia lo primero que se recomienda hacer es reemplazar el sistema de autenticación con usuario y contraseña por autenticación basada en llave pública, para esto se debe crear una llave pública utilizando `ssh-keygen`[35] y agregar la llave generada al entorno OpenStack.

Posteriormente para iniciar la instancia se debe utilizar un comando donde se requiere especificar varios parámetros tales como: *flavor*[36], nombre de la imagen que se va a cargar, red a la cual se va a asociar la instancia, y el grupo de seguridad al que va a estar asociado. Para lanzar una instancia con los parámetros indicados se puede emplear el Comando 2.7. El procedimiento para lanzar una instancia se puede encontrar en el Anexo B, en la página B-55.

```
$ nova boot --flavor m1.medium --image SDN-x86_64 --nic
  net- id=DEMO_NET_ID \
  --security-group default --key-name demo-key SDN-
  Package
```

Comando 2.7 Inicialización de la instancia

Una vez que se ha ejecutado el Comando 2.7 se debe esperar cierto tiempo mientras se comprueba que los parámetros sean correctos y se inicialice la

[35] `ssh-keygen`: comando del sistema operativo Unix que permite la creación, administración y almacenamiento de llaves [56].
[36] *flavor*: Conjunto de recursos de hardware identificado por un nombre en el entorno OpenStack, los cuales pueden ser asignados a una instancia.

imagen en la nube, una vez que ha terminado ese proceso se puede observar que la imagen esta iniciada mediante la interfaz web de OpenStack. En la Figura 2.4 se presenta una captura de pantalla de la interfaz web en la que se aprecia que la instancia creada está ejecutándose correctamente.

Figura 2.4. Instancia en la nube

CAPÍTULO III

3. APLICACIÓN WEB

En este capítulo se describirá el desarrollo de una aplicación web que permite automatizar el proceso de creación de Redes Definidas por Software mediante la nube previamente implementada, para lo cual se inicia con una breve explicación sobre Django que es el *framework* usado para el desarrollo de la aplicación web, se mencionará de manera simple sus principales características, continuando con las funcionalidades que ofrece la aplicación web para que el usuario se familiarice con la tecnología de SDN, y se presenta el proceso de desarrollo y los lenguajes empleados, para finalizar con la realización de pruebas para comprobar el correcto funcionamiento de la aplicación.

3.1. DJANGO

Django fue creado por un grupo de desarrolladores de Kansas, Estados Unidos, nació en 2003 cuando los programadores web del periódico Lawrence Journal-World, Adrian Holovaty y Simon Willison, empezaron a utilizar el lenguaje Python para desarrollar aplicaciones. Los requerimientos de manejo de información dinámica en páginas web de los diarios y la actualización de sus noticias en breves intervalos de tiempo, les llevó a crear un *framework* para el desarrollo de aplicaciones web en corto tiempo. En 2005 cuando habían perfeccionado dicho *framework* que era usado para el desarrollo de la mayoría de los sitios web y tenía una gran acogida, decidieron lanzarlo como software de código abierto y le asignaron el nombre de Django en honor al guitarrista de Jazz Django Reinhardt.

La historia del *framework* Django es importante porque presenta dos características esenciales: está diseñado principalmente para entornos que manejan contenidos dinámicos mediante el uso de herramientas de mucha utilidad para modificar el contenido de manera rápida, por ejemplo para páginas que manejan noticias o artículos de venta en las que la información de la base de

datos está cambiando constantemente. Aunque es una herramienta muy poderosa para el desarrollo de aplicaciones web en general. Otra característica es que Django fue desarrollado en un entorno de producción en donde se propuso una solución para los problemas cotidianos de un programador de aplicaciones web lo que trae muchas ventajas porque simplifica procesos que pueden ser tediosos y complicados [48].

3.1.1 PATRÓN DE DISEÑO

El patrón de diseño de software permite que el código utilizado pueda ser reusado con mayor facilidad, reduce la complejidad al momento de modificar el código y simplifica el proceso de mantenimiento de software.

Como antecedente, se realizará una breve descripción del patrón de diseño MVC (*Model-View-Controller*) que es muy utilizado para el desarrollo de aplicaciones web, con la finalidad de comprender sus conceptos y principios, para luego relacionarlos con los conceptos del patrón de diseño utilizado por Django conocido como MTV (*Model-Template-View*) que es muy similar al modelo MVC pero con ciertas diferencias.

3.1.1.1 Patrón de diseño MVC [49]

El patrón de diseño MVC propone dividir una aplicación en tres módulos claramente identificados y con sus funcionalidades bien definidas: Modelo, Vista y Controlador. El Modelo es un módulo conformado por componentes que se encargan de realizar el procesamiento requerido, la Vista es la encargada de presentar al usuario los aspectos que se obtienen del modelo y el Controlador es el que se encarga de enviar mensajes al modelo y proveer la interfaz entre el modelo, la vista asociada y los dispositivos usados para interactuar con la interfaz del usuario (por ejemplo: mouse, teclado, etc.).

En la relación entre los objetos que se muestra en la Figura 3.1 se puede observar que tanto la vista como el controlador dependen del modelo, pero el modelo no depende de ninguno de los dos, esta es la base de la separación que

presenta el patrón de diseño MVC entre la lógica de la interfaz del usuario y la lógica de procesamiento de la aplicación.

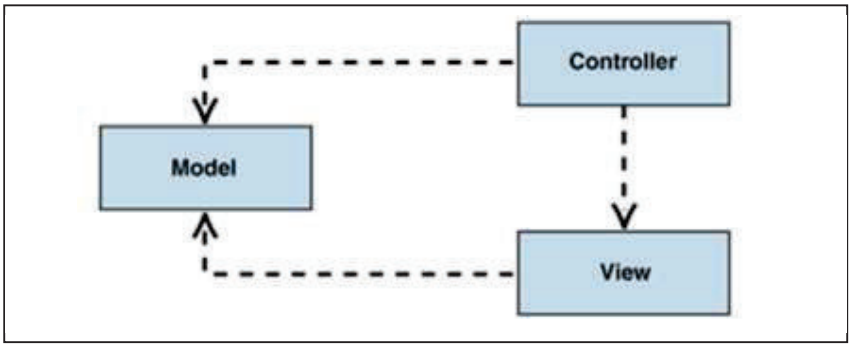

Figura 3.1. Relación entre los objetos del patrón de diseño MVC [68]

3.1.1.1.1 Modelo [66], [68]

Es una o más clases que manejan el comportamiento y los datos del dominio de la aplicación (*application domain*)[37], se encarga de realizar el procesamiento de la información según lo especificado por el programador, convirtiéndose en la estructura central de la aplicación, además responde a las peticiones de información sobre su estado y responde a instrucciones de cambio de estado.

3.1.1.1.2 Vista [66], [68]

Maneja la presentación de la información, solicita información al modelo y la muestra al usuario. La vista es la que se encarga de toda la parte gráfica. Maneja las ventanas, las crea, modifica y es en donde se define la relación que existe entre las diferentes ventanas y componentes gráficos.

3.1.1.1.3 Controlador [66], [68]

El controlador funciona como una interfaz entre los modelos y las vistas asociadas, se encarga de monitorear las señales de control que ejecutan los dispositivos de entrada tales como: teclado y mouse, por lo tanto el controlador se encarga de realizar el correcto procesamiento para informar al modelo o la vista del cambio solicitado.

[37] *Application domain*: Es un mecanismo para dividir los procesos en una aplicación y que no interfieran entre sí para brindar seguridad y fiabilidad [67].

3.1.1.2 Patrón de diseño MTV [69]

Django utiliza un patrón de diseño muy similar a MVC, lo que hace Django es dividir las funcionalidades de una manera distinta y asigna otros nombres. La función del controlador es realizada por Django, el cual se encarga de las peticiones, respuestas y enrutamiento adecuado de los datos. Las vistas están escritas por el programador para recolectar la información que va a ser presentada en las plantillas (*templates*), las plantillas y las vistas forman la capa de presentación.

3.1.1.2.1 Modelo

Es la fuente de información de donde se obtiene los datos, el modelo es el encargado de definir el comportamiento y los campos de la información almacenada.

3.1.1.2.2 Template

En Django se separa los datos de la presentación de información al usuario, es por eso que la vista selecciona que información va a ser presentada y la delega a una plantilla que es en donde se define el contenido estático de la plantilla HTML y como va a ser insertado el contenido dinámico.

3.1.1.2.3 View

Es una función de Python que recibe una petición web y retorna una respuesta web, es donde se debe especificar la lógica necesaria para que la respuesta enviada a la petición sea lo especificado por el programador, por lo general las respuestas son contenidos HTML, redirección del URL, un error, o una imagen entre otros.

El patrón de diseño MVC y MTV tienen el mismo fin, separar la aplicación en tres módulos con características y funcionalidades muy claras, la idea es separar la presentación de la información al usuario con la parte donde se procesan las peticiones y se selecciona la información.

Esto hace que el diseño de la aplicación sea más amigable porque permite diseñar cada una de los componentes de manera independiente, la solución de problemas es más simple porque se puede localizar con mayor facilidad los errores, modificar y aumentar funcionalidades; no es tan complicado, porque se puede identificar de una manera más rápida en dónde se debe agregar el código para que cumpla dicha función.

3.1.2 LENGUAJE DE PROGRAMACIÓN DE DJANGO [71]

Django utiliza el lenguaje de programación Python, que tiene las siguientes características:

- Interpretado: El código escrito en Python es procesado en tiempo de ejecución por el intérprete, no necesita que sea compilado antes de ser ejecutado.
- Interactivo: Ofrece la posibilidad de interactuar con el intérprete y escribir un programa directamente desde el *prompt*[38] de Python.
- Orientado a objetos: Python soporta el paradigma de la programación orientada a objetos.
- Lenguaje para principiantes: Python es un lenguaje muy apropiado para los programadores que están iniciándose, ya que permite el desarrollo de aplicaciones tanto de baja como de alta complejidad.

3.2 DESARROLLO DE LA APLICACIÓN WEB

La aplicación web se desarrolla con el propósito que el usuario cree de manera automática la infraestructura SDN en la nube. El usuario debe seleccionar los parámetros topología, el número de host y el controlador, una vez que han sido seleccionados se crea la topología y se muestra una representación gráfica de la misma para que el usuario pueda relacionarse con una arquitectura SDN.

[38] *Prompt*: Es un símbolo de texto que representa que el sistema está esperando una entrada del usuario [60].

3.2.1 FUNCIONALIDADES DE LA APLICACIÓN WEB

La aplicación web está diseñada para que el usuario pueda conocer los principales conceptos sobre SDN y cómo interactúan los dispositivos entre sí, el usuario debe seleccionar qué topología desea implementar, esto quiere decir que tiene diferentes opciones para seleccionar la distribución de los componentes de red.

Con la topología definida puede seleccionar el número de *hosts* que desea incorporar y qué controlador va a utilizar, ya que los controladores pueden tener características diferentes. Con los componentes de red definidos se debe seleccionar la opción para generar la topología. La topología establece la forma en la que están conectados los switches y los *hosts*. La aplicación mediante una representación gráfica presentará la forma en la que están conectados todos los equipos, para que el usuario se relacione con topologías de red.

La aplicación web está diseñada para que el usuario pueda interactuar con los conceptos de SDN sin un conocimiento amplio sobre este tema, ya que el manejo de SDN, en la mayoría de los casos, requiere interactuar con terminales de comandos para generar la topología deseada. La interfaz gráfica pretende que el usuario pueda generar una topología SDN sin tener que manejar comandos y pueda visualizar el resultado mediante una representación gráfica. Se debe mencionar que la aplicación automatiza la creación de la topología en una máquina virtual, la cual se encuentra alojada en la nube.

La máquina virtual que se encuentra alojada en la nube, tiene el sistema operativo Xubuntu que es un sistema operativo con buenas prestaciones y bajos requerimientos de hardware, está basado en Ubuntu y presenta las herramientas necesarias para la implementación de la infraestructura SDN, como son: Mininet, que es el simulador en donde se va a crear los dispositivos SDN, switch virtual Open vSwitch y controladores POX, RYU y Pyretic.

3.2.1.1 Tipos de Topología

La topología de red es la distribución física y lógica de los dispositivos en una red de comunicación, la aplicación web presenta dos alternativas para que el usuario seleccione una de ellas.

3.2.1.1.1 Topología Tipo Árbol

En esta topología se puede indicar que el nombre describe cómo se conectan los dispositivos, cuenta con una "raíz" un switch del cual salen "ramas", la idea en esta topología es que cada rama que sale de la raíz se sigue expandiendo, generando nuevas ramas hasta llegar al usuario final. Para la creación de la topología árbol se debe especificar dos parámetros que son *depth* y *fanout*, el parámetro *depth* define cuántos niveles debe haber desde el switch principal hasta el usuario final y el parámetro *fanout* define cuántos dispositivos se conectan a cada una de las ramas. En esta topología el número de *hosts* está definido por los parámetros *depth* y *fanout*. Para que el concepto de topología árbol quede claro se van a presentar dos ejemplos de esta topología creadas usando la aplicación web.

En el primer ejemplo, presentado en la Figura 3.2 se ha seleccionado *depth* = 2 y *fanout* = 3. Se puede ver que existen dos niveles desde los *hosts* hasta el switch principal y a cada dispositivo de red se conectan 3 dispositivos, también se puede ver el controlador en la parte superior, el número de *hosts* creados en este ejemplo es 9.

En el segundo ejemplo, el cual se presenta en la Figura 3.3, se ha seleccionado *depth* = 3 y *fanout* = 2. Se puede observar que se tienen tres niveles desde los *hosts* hasta el switch principal y a cada dispositivo de red se conectan 2 dispositivos, lo que quiere decir que se va a tener un total de 8 *hosts*.

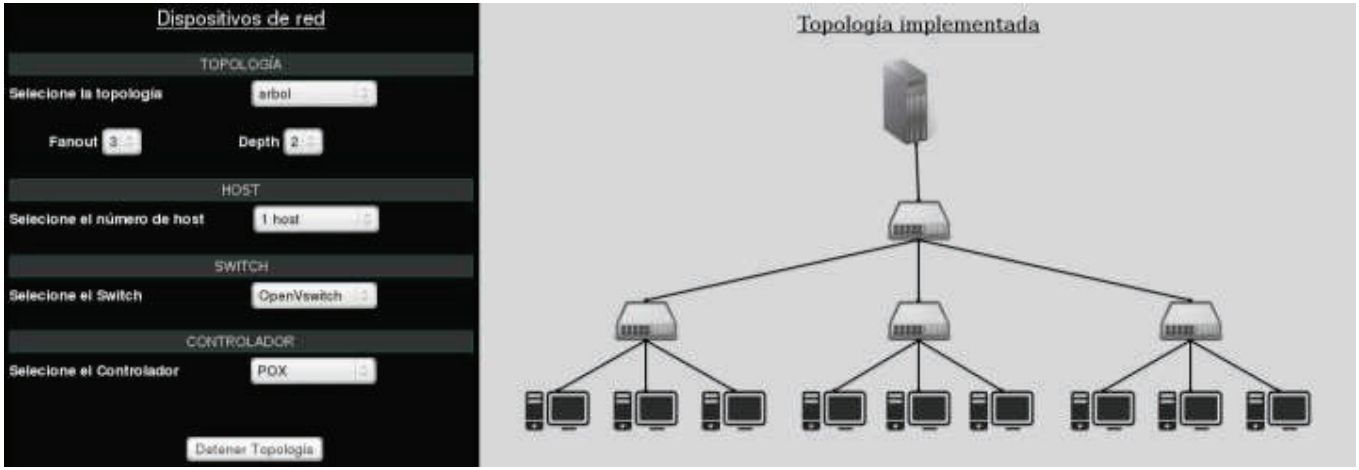

Figura 3.2. Topología tipo árbol con parámetros *fanout*=3 y *depth*=2

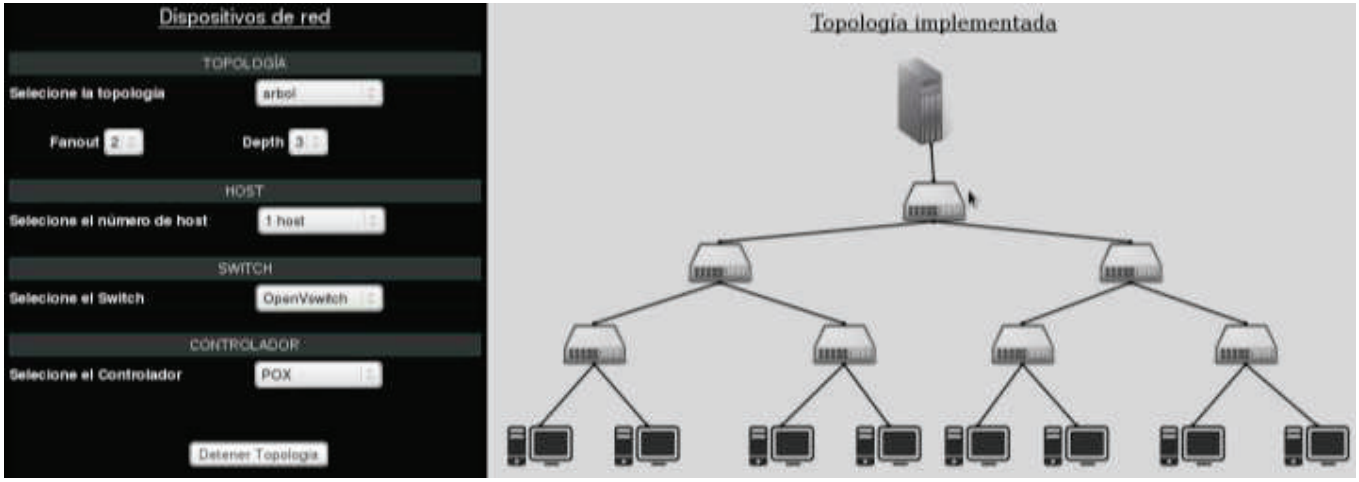

Figura 3.3. Topología tipo árbol con parámetros *fanout*=2 y *depth*=3

3.2.1.1.2 Topología simple

La topología simple es una topología en donde se tiene un solo nivel, los *hosts* están conectados directamente al switch. En este caso se debe seleccionar un valor en el parámetro para definir el número de *hosts*. Se presentan dos ejemplos que fueron creados en la aplicación para ilustrar la topología simple.

En el primer ejemplo se han seleccionado 4 *hosts*, lo cual se puede ver en la Figura 3.4, en donde hayun único switch con los 4 *hosts* conectados a él, y en la parte superior se puede observar el controlador que está conectado al switch.

En el segundo ejemplo se han seleccionado nueve *hosts*, los cuales están conectados directamente al switch, el cual está conectado al controlador. La topología indicada se presenta en la Figura 3.5.

3.2.1.2 Número de hosts

Parámetro que permite seleccionar el número de *hosts* que se van a conectar al switch en la topología simple, esta opción se deshabilita cuando el usuario selecciona la topología tipo árbol, considerando que el número de host está definido por los parámetros *depth* y *fanout.* En la Figura 3.6 se muestra lo mencionado.

Figura 3.4 Parámetro *hosts* en topología simple vs topología árbol

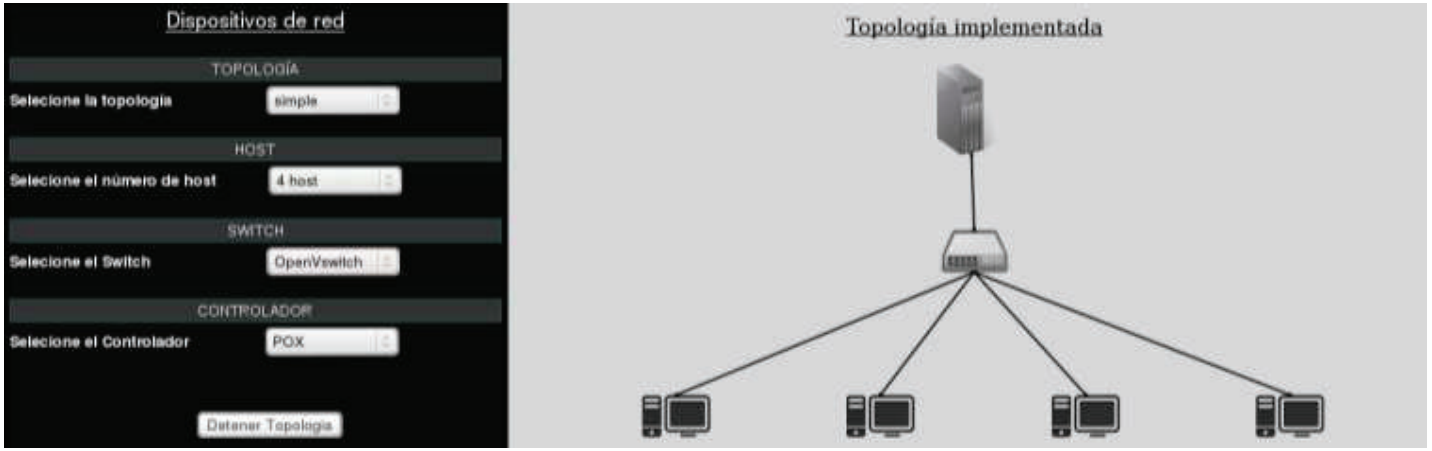

Figura 3.5. Topología simple con 4 *hosts*

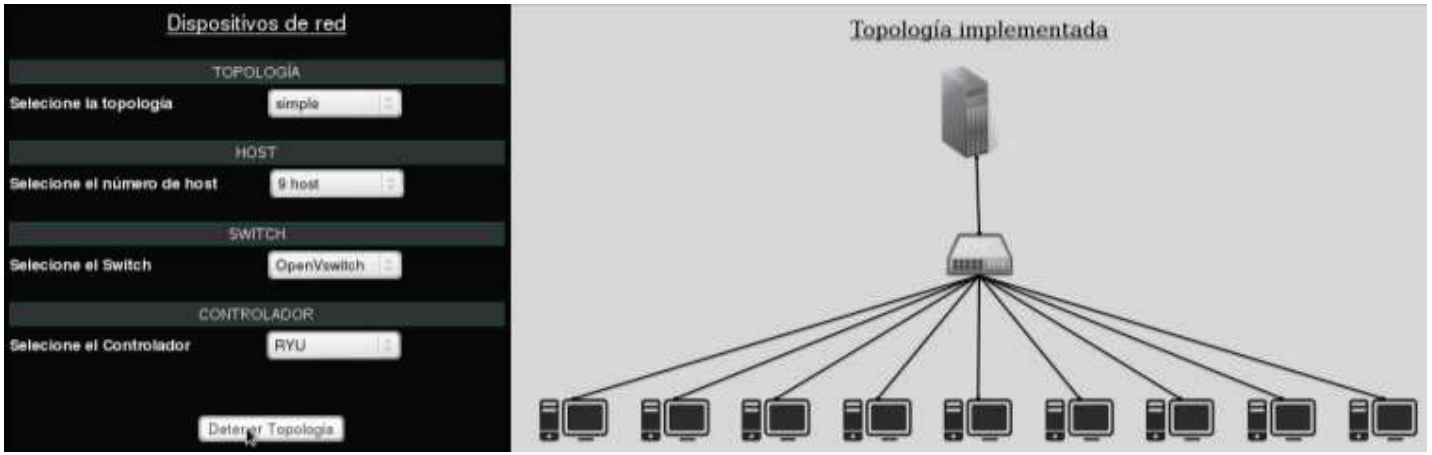

Figura 3.6. Topología simple con 9 *hosts*

3.2.1.3 Switch

En la alternativa switch se ha seleccionado Open vSwitch como única opción, ya que tiene una gran acogida en el área de SDN y ofrece altas prestaciones de red. Open vSwitch es un switch virtual multicapa diseñado para ambientes de producción y se maneja bajo la licencia Apache 2.0.

3.2.1.4 Controlador

Tiene una visión general de la red considerando que es el encargado de establecer las políticas del tráfico que ingrese y salga de ella. Una parte interesante que ofrece SDN es que las políticas de tráfico en la red pueden ser programadas por el usuario para definir el comportamiento y funcionalidades que va a tener, lo que permite independizarse del fabricante. Se presentan tres alternativas de controladores para que el usuario conozca las diferentes opciones que hay en el mercado y deben ser seleccionadas en función de los requerimientos del usuario tales como: lenguaje de programación, compatibilidad con dispositivos específicos, entre otros.

3.2.1.4.1 POX

Es un *framework* para el desarrollo rápido de controladores de redes definidas por software basado en el lenguaje de programación Python, POX es la nueva versión del controlador NOX[39] y presenta funcionalidades para facilitar la programación de controladores para SDN.

3.2.1.4.2 RYU

Es un *framework* basado en componentes (*component-based)*[40] para el desarrollo de redes definidas por software, RYU ofrece componentes con las API bien

[39] NOX: Plataforma para construir aplicaciones de control de red que ha sido desarrollada desde los inicios de SDN.
[40] *Component based:* Es una metodología para desarrollo de software que se basa en el uso de código creado previamente que se lo llama componente, la idea es unir los componentes para crear el software deseado haciendo más eficiente el proceso de creación de software [63].

definidas que simplifican el proceso de creación de controladores SDN. RYU es un controlador basado en el lenguaje de programación Python y soporta las versiones 1.0, 1.2, 1.3, 1.4 del protocolo OpenFlow[41], lo cual es importante dado que es el protocolo más utilizado para establecer la conexión entre el controlador y las capas de envío de SDN. [62]

3.2.1.4.3 Pyretic

Es un miembro de la familia del lenguaje de programación para SDN Frenetic[42], Pyretic permite a los programadores desarrollar aplicaciones modulares proveyendo abstracciones que son de mucha utilidad. Pyretic presenta dos funcionalidades principales, es un lenguaje embebido en Python amigable con el programador y es un sistema de ejecución que se encuentra en los switches para implementar los programas escritos en Pyretic.

3.2.1.5 Representación gráfica de la topología implementada

El propósito de la aplicación web es que los usuarios se familiaricen con los conceptos básicos de SDN, el usuario puede seleccionar los dispositivos SDN, mostrar el resultado de una manera gráfica permite que el usuario sepa que es lo que está haciendo y como se conectan los dispositivos entre sí.

En la Figura 3.7 se presenta la representación gráfica que brinda como resultado la aplicación web al escoger una topología tipo árbol y, en la Figura 3.8 se presenta la representación gráfica que despliega la aplicación web al escoger la topología simple.

3.2.2 IMPLEMENTACIÓN DE LA APLICACIÓN WEB

La implementación de la aplicación web está definida en dos etapas, la primera corresponde al desarrollo del *frontend* que es lo que el usuario ve y le permite

[41] OpenFlow: Es una interfaz de código abierto para controlar de manera remota las tablas de envió en los switch, *routers* y *access point*, permite a los investigadores ejecutar protocolos experimentales en el área de las redes.
[42] Frenetic: Es un proyecto que busca crear un lenguaje para la programación de redes.

interactuar con la aplicación web. La segunda es el desarrollo del *backend* que es donde está toda la lógica de la aplicación web.

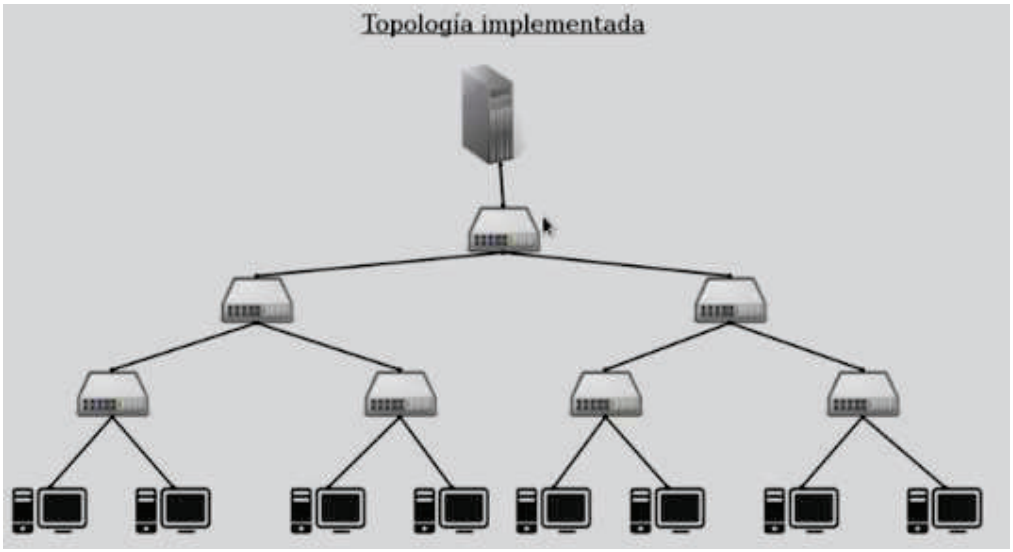

Figura 3.7. Topología Tipo Árbol

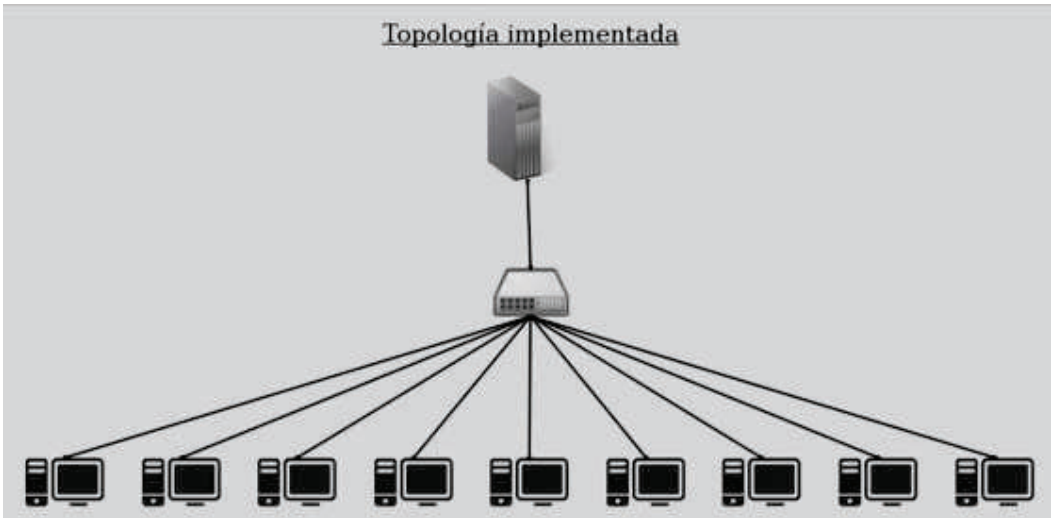

Figura 3.8. Topología Simple

3.2.2.1 Desarrollo del Frontend

Se inició el desarrollo con el diseño visual y la distribución que van a tener los componentes, la distribución realizada se muestra en la Figura 3.9.

Con el diseño en bloques se busca delimitar los diferentes componentes que presenta la aplicación, cada uno de los bloques realiza una función diferente y

tiene un propósito específico, es por eso que cada uno de los bloques debe estar bien delimitado.

Bloque 1:

Está diseñado para ser el primero en captar la atención del usuario y presentar el título, el cual describe la función de la aplicación y una breve explicación sobre SDN. Como fondo del contenedor correspondiente al título se usa un color fuerte que contrasta con el fondo del bloque 1 para que a primera vista se destaque.

Bloque 2:

Presenta al usuario los diferentes dispositivos que pueden ser implementados en la red SDN, aquí se cargarán las diferentes opciones mediante un `dropdown list`[43], para que el usuario tenga conocimiento de las diferentes opciones y seleccione una alternativa en cada opción para formar la topología deseada.

Bloque 3:

En este bloque se va a presentar la topología implementada con las alternativas y los dispositivos que seleccionó el usuario.

3.2.2.1.1 Lenguajes para el desarrollo de la aplicación web

HTML (*HyperText Markup Lenguage*)

Para elaborar plantillas y presentar la información al usuario se va a emplear el lenguaje HTML que es un lenguaje de programación muy utilizado en WWW (*World Wide Web*) donde se puede obtener una cantidad muy extensa de información que se encuentra en las páginas web mediante texto, imágenes y diferentes herramientas visuales.

[43] `dropdown list`*:* Es un control grafico que le permite al usuario seleccionar una sola alternativa de la lista de opciones presentada.

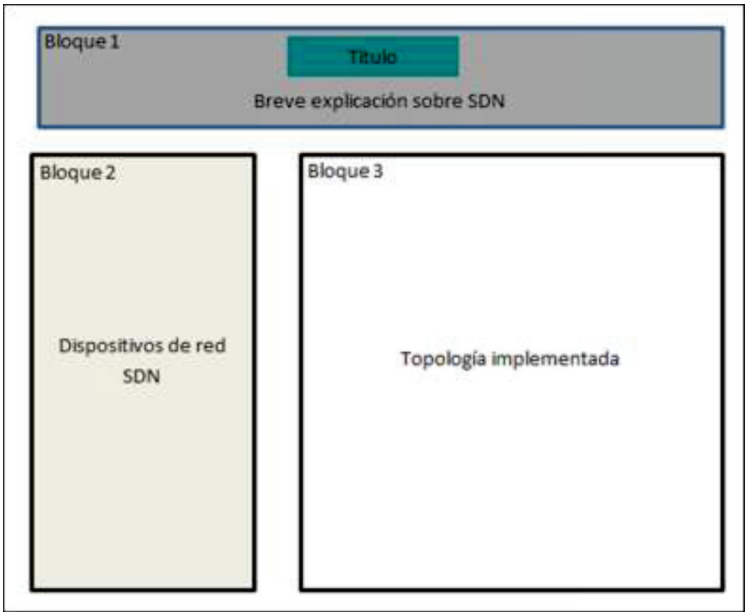

Figura 3.9. Distribución visual de los componentes

HTML se caracteriza por ser un lenguaje basado en etiquetas, las cuales se encargan de manejar el contenido y la estructura con las que será presentada la información.

CSS (*Cascading Style Sheets*)

"CSS es un lenguaje creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML[44]. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas" [61].

JavaScript

Es un lenguaje de *scripting* mediante el cual se determina el funcionamiento que va a tener cada uno de los objetos definidos mediante el documento HTML, este lenguaje permite agregar funcionalidad al contenido especificado en HTML.

[44] XHTML (*Extensible Hypertext Markup Lenguage*): es un lenguaje basado en hipertexto basado en la arquitectura XML para brindar mayor flexibilidad y utilidades que HTML. XML (*Extensible Markup Lenguage*) es un lenguaje que permite definir la estructura de documentos electrónicos.

3.2.2.1.2 Codificación de la aplicación web

Para la programación de la aplicación web se va a utilizar los tres lenguajes mencionados, cada uno de ellos cumple una función en el proceso de diseño e implementación de la aplicación web, la función de HTML es definir el contenido, CSS el estilo y JavaScript el comportamiento de la aplicación web.

Con las herramientas indicadas y una vista general de la distribución de los componentes, se inicia la implementación con la ayuda del lenguaje HTML, donde se establecen tres contenedores los cuales se ubican según la distribución presentada, dentro de cada uno de estos se definen diferentes secciones en donde se ubica cada elemento con la finalidad de manipular cada uno de ellos de una manera más organizada. Para dividir cada uno de los bloques y secciones se utiliza la etiqueta `div`[45], esta etiqueta permite añadir un identificador único al bloque o elemento, con el objetivo de ubicarlo con mayor facilidad al momento de editar su estilo o funcionalidad.

En el Código 3.1 se presenta el contenedor con todo el bloque superior y los elementos que forman parte de él, los cuales son el título de la aplicación, el sello de la Escuela Politécnica Nacional y una descripción de SDN, que indica además, cuál fue el patrón de diseño usado para el desarrollo de la aplicación web.

Con el lenguaje HTML se definen los contenidos de la aplicación y con la ayuda del lenguaje CSS se le da estilo al contenido determinado en el documento HTML, en la Figura 3.10 se presenta el contenido del bloque 1 (bloque superior) sin estilo y en la Figura 3.11 se ve el contenido HTML con estilo. Con CSS se puede dividir el contenido del diseño, lo cual facilita un desarrollo más estructurado de la aplicación. Este concepto se emplea para toda la aplicación web.

[45] div: Es una etiqueta para un contenedor que encapsula otros elementos de la página y divide el documento HTML en secciones

```
26  <!--Se crea el Encabezado-->
27  <div id='parte_superior'>
28
29      <div id='encabezado'>
30      PROTOTIPO PARA LA IMPLEMENTACIÓN DE REDES SDN EN LA NUBE
31      </div>
32
33      <!--Logo Escuela Politecnica Nacional -->
34      <div id="logo_epn">
35          <img id='imagen_epn' src='/root/Desktop/GUI_TESIS/Imagenes/epn-logo2.png' vspace="10" />
36      </div>
37
38      <!--Descripcion que resume los dispositivos de una red SDN -->
39      <div id='descripcion'>
40          Una red SDN tiene tres dispositivos esenciales, Host, Switch y Controlador. En donde el Controlador es el encargado
            de establecer las reglas para el trafico en la red, el switch tiene una conexión a través del protocolo openflow
            con el controlador para conocer cual es la acción que debe realizar cuando llega un determinado paquete y
            finalmente se tiene el host en donde el usuario final realiza sus tareas.
41      </div>
42  </div>
```

Código 3.1. Segmento HTML para el bloque 1

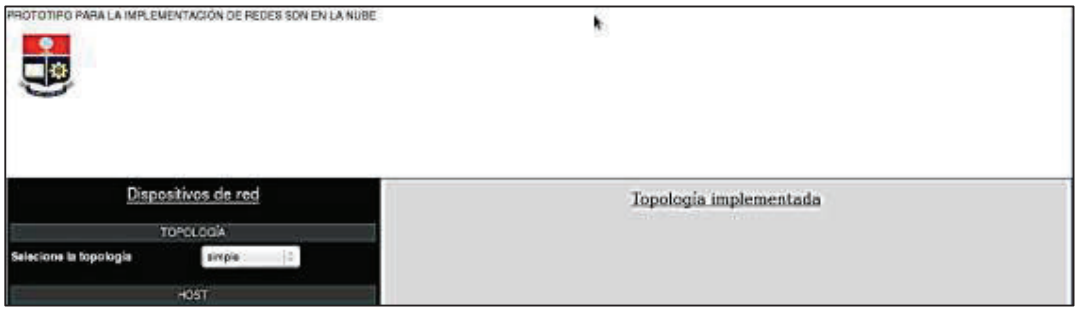

Figura 3.10. Bloque superior sin estilo CSS

Figura 3.11. Bloque superior con estilo CSS

Para dar formato a un determinado elemento se lo debe referenciar y dar valor a los diferentes atributos que tiene el elemento, en el Código 3.2 se presenta un rápido ejemplo de cómo modificar el estilo de un elemento, en este caso se establecen los siguientes parámetros: se justifica el texto, color de la fuente blanco, familia de fuente Arial, posición y márgenes del elemento.

```css
#descripcion
{

text-align: justify;
color: white;
margin-left: 20px;
margin-right: 20px;
margin-bottom: 20px;
font-family: arial;
margin-top: 20px;

}
```

Código 3.2. Ejemplo de estilo en lenguaje CSS

Para escoger las opciones se ha decidido utilizar un `dropdown list`, ya que permite listar todas las opciones disponibles y seleccionar solo una de ellas, para la configuración de este componente se debe especificar las opciones a mostrar, como se presenta en el Código 3.3.

```html
<!--Dropdown que permite seleccionar el numero de host -->
<div id='num_host'>
    <label>
        Selecione el número de host
    </label>
    <select id='select_2'>
        <option value='1'> 1 host </option>
        <option value='2'> 2 host </option>
        <option value='3'> 3 host </option>
        <option value='4'> 4 host </option>
        <option value='5'> 5 host </option>
        <option value='6'> 6 host </option>
        <option value='7'> 7 host </option>
        <option value='8'> 8 host </option>
        <option value='9'> 9 host </option>
    </select>

</div>
```

Código 3.3. Segmento de código HTML con un dropdown list

El lenguaje JavaScript se utilizó para definir el comportamiento que va a tener la aplicación web, esto quiere decir que se determinó qué acción se va a realizar cuando se presione un botón, o cómo se va a comportar la aplicación cuando se seleccione una determinada topología.

Las principal acción que se realiza con el lenguaje JavaScript es presentar la topología implementada por el usuario, para esto se utiliza la sentencia `switch`, entre otras, para definir una acción en cada una de las posibles elecciones del usuario, en el Código 3.4 se presenta la sección con la sentencia `switch` para la topología simple, en donde el usuario debe establecer el número de *hosts*.

```
364  if (valor == "simple") {
365
366
367      switch(document.getElementById('select_2').value) {
368      case "1":
369          $('#btn_detener_Topo').show();
370          $('#img_controller').show();
371          $('#img_switch').show();
372          $('#img_host_1').show();
373          $('#img_host_1').css('margin-left',"46%");
374
375          //Se dibuja las lineas entre dispositivos
376          jsPlumb.ready(function() {
377
378                  dibujarLinea("img_controller","img_switch");
379                  dibujarLinea("img_switch","img_host_1")
380
381          });
382          jsPlumb.repaintEverything();
383
384          break;
```

Código 3.4. Sentencia switch empleada en la topología simple

En el caso de la topología tipo árbol se tienen que evaluar dos parámetros para presentar la topología, para esto se utilizó la sentencia `switch` junto con la sentencia `if`. En el Código 3.5 se muestra un segmento donde se puede observar lo indicado.

Con estos dos ejemplos se describe cómo se utilizó el lenguaje JavaScript para controlar la funcionalidad de la aplicación.

Para la presentación de la topología se definieron un conjunto de imágenes de los dispositivos de red según la topología seleccionada. Es necesario indicar que se debe establecer la ubicación de las imágenes para que se coloquen en el lugar correcto, para esto se debe manipular los márgenes y los valores que definen la posición con la ayuda del lenguaje JavaScript.

Para establecer la conexión entre dispositivos de red se empleó la herramienta `jsPlumb`, la cual permite dibujar las líneas de interconexión entre los diferentes dispositivos. Para dibujar una línea se debe definir dos puntos, el ancho de la línea, color y el tamaño de los puntos de inicio y fin. Cuando se presiona el botón que finaliza la topología se deben eliminar todos los puntos de referencia para que en la próxima topología no se tomen como referencia los puntos anteriores, ya que las imágenes se van a encontrar en diferentes posiciones.. Para la creación de las líneas se definió una función llamada `dibujarLinea` con la ayuda de JavaScript, en donde se llama a las funciones de la librería `jsPlumb`, esta función acepta como parámetro un punto inicial y un punto final, en el Código 3.6 se presenta la función mencionada.

```
634     switch(document.getElementById('select_6').value) {
635     case "1":
636         if(document.getElementById('select_5').value == "1")
637         {
638             $("#btn_detener_Topo").show();
639             $("#img_controller").show();
640             $("#img_switch").show();
641             $("#img_host_1").show();
642             $('#img_host_1').css('margin-left',"46%");
643
644             //Dibujar lineas entre dispositivos
645             jsPlumb.ready(function() {
646             jsPlumb.reset();
647
648             dibujarLinea("img_controller","img_switch");
649             dibujarLinea("img_switch","img_host_1");
650             });
651         jsPlumb.repaintEverything();
652
653         }
654
655         else if(document.getElementById("select_5").value == "2")
656         {
657
658         $('#btn_detener_Topo').show();
659         $('#img_controller').show();
660         $('#img_switch").show();
661         $('#img_host_1").show();
662         $('#img_host_2").show();
663         $('#img_host_1').css('margin-left','26%');
664         $('#img_host_2').css('margin-left','31%');
```

Código 3.5. Sentencias `if` y `switch` en la topología Tipo Árbol

```
//Funcion para dibujar lineas entre dispositivos-->

function dibujarLinea(inicio, fin)
{
    jsPlumb.connect({

        source:inicio,
        target:fin,

        anchors:["BottomCenter","TopCenter"],
        paintStyle:{lineWidth:2, strokeStyle:"black"},
        connector:'Straight',
        deleteEndpointsOnDetach:true,
        endpointStyle:{radius:2}
    });

}
```

Código 3.6. Función para dibujar líneas

Para la presentación de la topología se debe mostrar las imágenes correspondientes y dibujar la línea entre ellos, en el Código 3.7 se puede ver

como se presentan las imágenes y se dibujan las líneas para la topología simple con un *host*.

```
if (valor == "simple") {


    switch(document.getElementById("select_2").value) {
    case "1":
        $("#btn_detener_Topo").show();
        $("#img_controller").show();
        $("#img_switch").show();
        $("#h1").show();
        $('#h1').css('margin-left',"46%");

        //Se dibuja las lineas entre dispositivos
        jsPlumb.ready(function() {

                dibujarLinea("img_controller","img_switch");
                dibujarLinea("img_switch","h1")

        });
        jsPlumb.repaintEverything();

        break;
```

Código 3.7. Mostrar topología simple con un *host*

3.2.2.2 Desarrollo de Backend

3.2.2.2.1 Creación del proyecto y aplicación en Django

Para el desarrollo del *Backend* se debe iniciar con la creación de un espacio de nombre en donde se va a alojar el proyecto y la aplicación que va a ser creada, una vez que se ha definido la carpeta (espacio de nombres) en donde se va a trabajar se debe crear el proyecto, para esto de se debe ejecutar el Comando 3.1 en donde se debe cambiar NOMBRE por el nombre que se desea asignar al proyecto.

```
#django-admin.py start project NOMBRE
```

Comando 3.1. Creación del proyecto de Django

Cuando se crea el proyecto de Django, se crea una carpeta con el mismo nombre del proyecto en donde se encuentran los archivos: `__init__.py`,

`settings.py`, `urls.py`, `wsgi.py` y fuera de esta carpeta se crea el archivo `manage.py`. A continuación se detalla cuál es la función de cada uno de estos archivos creados por Django.

- `__init__.py`: Es un *script* en blanco que indica que el directorio en el que se encuentra debe ser considerado un paquete Python.
- `settings.py`: *Script* en donde se encuentra la configuración del proyecto Django.
- `urls.py`: Es un *script* en donde se define las aplicaciones del proyecto, en este caso se re direcciona a la aplicación web porque solo se tiene una aplicación, en el Código 3.8 se puede observar esta redirección.

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'interfaz_1.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),

    url(r'^', include('gui_sdn.urls')),  #Redireccion a urls.py de la aplicacion sdn
)
```

Código 3.8. Archivo `urls.py` del proyecto

- `wsgi.py`: *Script* que permite acceder a servidores web desarrollados bajo los parámetros WSGI[46] (*Web Server Gateway Interface*).
- `manage.py`: permite realizar tareas de administración, como ejecutar el servidor, migrar la base de datos, entre otros.

Una vez que se conoce qué función cumple cada uno de los archivos generados automáticamente por Django, se va a listar las carpetas y los archivos que fueron creados en esta carpeta para este Proyecto de Titulación.

- `static`: Carpeta en donde se almacena todas las herramientas que permiten modificar el comportamiento del contenido definido en HTML. Se

[46] WSGI: Especificación que describe cómo un servidor web se comunica con la aplicación web.

encuentra la librería de JavaScript, las imágenes que se utilizan y el archivo CSS en donde se define el estilo.

- `template`: Carpeta en donde se almacena la plantilla HTML utilizada.
- `Controller.py`: *Script* en donde se especifican las funciones y el procedimiento para la creación de la infraestructura SDN.

Posteriormente debe crearse la aplicación, para lo cual se emplea el Comando 3.2, en donde se debe reemplazar NOMBRE por el nombre que se le quiere dar a la aplicación web.

```
# python manage.py startapp NOMBRE
```

Comando 3.2. Creación de la aplicación web en Django

Cuando se crea la aplicación web Django genera varios archivos, algunos de estos archivos como `__init__.py` cumplen las mismas funciones que ya fueron mencionadas anteriormente, además genera los siguientes archivos:

- `model.py`: *Script* que maneja los datos almacenados, su comportamiento y mapea la información a la base de datos.
- `test.py`: *Script* que permite establecer funciones para realizar pruebas sobre el código que se está desarrollando.
- `admin.py`: *Script* que permite activar tareas administrativas.
- `Views.py`: *Script* que cumple un papel importante porque es el encargado de recibir la petición y devolver la respuesta correcta, en este *script* se especifica el procedimiento necesario para devolver la respuesta deseada.

3.2.2.2.2 Programación de la aplicación web

Para que la aplicación web realice el proceso de creación de infraestructura SDN en la Nube, se debe especificar cómo se realizarán estos procesos, para lo cual se emplean dos *scripts* principalmente, estos son `views.py` en la aplicación web y `controller.py`.

controller.py

Script en donde se define los procedimientos y comandos que deben ser ejecutados para automatizar la creación de infraestructura en la nube.

En este *script* se define un conjunto de funciones que permite la creación de la infraestructura según los parámetros recibidos. En el Código 3.9 se presenta la función para iniciar el controlador POX.

```
def crear_pox():
    os.chdir('/home/ubuntu/pox')
    subprocess.call('python pox.py log.level --DEBUG forwarding.tutorial_l2_hub &',shell=True)
```

Código 3.9. Función para iniciar el controlador POX

Para iniciar el controlador RYU se define la función `crear_ryu`, la cual se muestra en el Código 3.10.

```
def crear_ryu():
    os.chdir('/home/ubuntu/ryu')
    subprocess.call('./bin/ryu-manager* --verbose ryu/app/simple_switch_13.py &',shell=True)
```

Código 3.10. Función para iniciar el controlador RYU

Para iniciar el controlador Pyretic se utiliza la función que se muestra en el Código 3.11.

```
def crear_pyretic():
    os.chdir('/home/ubuntu/pyretic')
        subprocess.call('python  pyretic.py -m p0 pyretic.modules.mac_learner &',shell=True)
```

Código 3.11. Función para iniciar el controlador Pyretic

Además, se dispone de una función para cada topología, cada una de estas funciones especifica qué topología se implementará, el switch y el controlador remoto a usarse mediante Mininet. En el Código 3.12 se presenta la función para la creación de la topología simple en el simulador Mininet, esta función recibe un parámetro que es el número de *hosts*.

```
def topologia_simple(host):

    net = Mininet( topo=SingleSwitchTopo( host), controller=partial(RemoteController, ip='127.0.0.1', port=6633) )
    net.start()
    resultado=net.pingAll()
    net.stop()
    subprocess.call('sudo mn -c',shell=True)
    subprocess.call('sudo fuser -k 6633/tcp',shell=True)
    return resultado
```

Código 3.12. Función para crear topología simple

En el Código 3.13 se presenta la función para la creación de la infraestructura SDN, esta función recibe dos parámetros, que son *depth* y *fanout*.

```
def topologia_arbol(_depth,_fanout):
    treeTopo=TreeTopo(depth=_depth,fanout=_fanout)
    net= Mininet(topo=treeTopo, controller=partial(RemoteController, ip='127.0.0.1', port=6633))
    net.start()
        resultado=net.pingAll()
        net.stop()
        subprocess.call('sudo mn -c',shell=True)
    subprocess.call('sudo fuser -k 6633/tcp',shell=True)
        return resultado
```

Código 3.13. Función para crear topología árbol

Se requiere finalmente definir la función `main`, en la cual se analizarán los parámetros que se pasen mediante las sentencias `if` y `elif`, que permiten discriminar qué parámetros ha pasado el usuario. El primer parámetro es el controlador, el segundo es la topología; si es topología simple el tercer parámetro es el número de *hosts* y, si es topología árbol el tercer parámetro es *depth* y el cuarto parámetro es *fanout*. En base a estos parámetros y las sentencias de control mencionadas se inicia un subproceso mediante Mininet en el cual se implementa la topología que el usuario selecciona. En el código 3.14 se presenta la sección de la función `main` en donde se verifica si es el controlador POX y qué topología se va a crear.

```
if __name__=='__main__':

    if len(sys.argv)>0:

        if sys.argv[1]=='pox':

            crear_pox()
            time.sleep(15)
            if sys.argv[2]=='simple':
                if sys.argv[3]:
                    print '\n'
                    print 'Topologia simple '+'Host: '+sys.argv[3]+'\n'
                    print 'Prueba de conectividad POX'+'\n'
                    print '% de paquetes perdidos: ' + str(topologia_simple(int(sys.argv[3])))
            elif sys.argv[2]=='arbol':
                if sys.argv[3] and sys.argv[4]:
                    print '\n'
                                    print 'Topologia arbol '+'\n'
                    print 'Depth: '+sys.argv[3]+ ' Fanout: '+sys.argv[4]+'\n'
                                    print 'Prueba de conectividad POX'+'\n'
                                    print '% de paquetes perdidos: '+ str(topologia_arbol(int(sys.argv[3]),int(sys.argv[4])) )
```

Código 3.14. Función `Main` parte 1

En el código 3.15 se presenta la sección de la función `main` para inicializar el controlador RYU y determinar qué topología ha seleccionado el usuario, para

determinar qué usuario y topología se debe implementar, se utilizan los parámetros que se pasan cuando se llama al *script*.

```
elif sys.argv[1]=='ryu_topo':
                time.sleep(15)
                if sys.argv[2]=='simple':
                        if sys.argv[3]:
            print '\n'
                                print 'Topologia simple '+'Host: '+sys.argv[3]+'\n'
                                print 'Prueba de conectividad RYU'+'\n'
                                print '% de paquetes perdidos: ' + str(topologia_simple(int(sys.argv[3])))

                elif sys.argv[2]=='arbol':
                        if sys.argv[3] and sys.argv[4]:
            print '\n'
                                print 'Topologia arbol '+'\n'
                                print 'Depth: '+sys.argv[3]+ ' Fanout: ' +sys.argv[4]+'\n'
                                print 'Prueba de conectividad RYU'+'\n'
                                print '% de paquetes perdidos: '+ str(topologia_arbol(int(sys.argv[3]),int(sys.argv[4])) )

elif sys.argv[1]=='ryu_controller':
    crear_ryu()
```

Código 3.15. Función `Main` parte 2

En el código 3.16 se presenta la sección de la función `main` en donde se inicializa el controlador Pyretic y se determina qué topología es la que el usuario desea implementar.

```
elif sys.argv[1]=='pyretic_topo':

    time.sleep(15)
                if sys.argv[2]=='simple':
                        if sys.argv[3]:
            print '\n'
                                print 'Topologia simple '+'Host: '+sys.argv[3]+'\n'
                                print 'Prueba de conectividad PYRETIC'+'\n'
                                print '% de paquetes perdidos: ' + str(topologia_simple(int(sys.argv[3])))

                elif sys.argv[2]=='arbol':
                        if sys.argv[3] and sys.argv[4]:
                                print '\n'
                                print 'Topologia arbol '+'\n'
                                print 'Depth: '+sys.argv[3]+ ' Fanout: '+ sys.argv[4]+'\n'
                                print 'Prueba de conectividad PYRETIC'+'\n'
                                print '% de paquetes perdidos: '+ str(topologia_arbol(int(sys.argv[3]),int(sys.argv[4])) )
```

Código 3.16. Función `Main` parte 3

En la función `main` se debe especificar un caso para cada uno de los controladores, dado que estos deben ser inicializados con diferentes permisos de usuario, caso contrario se generan errores al inicializarlos.

views.py

En el *script* `views.py` es en donde se debe generar la respuesta que va a ser enviada a la petición que llegó al servidor. Para que el resultado deseado sea enviado al usuario se realizó los siguientes cambios.

Lo primero que se realiza es almacenar las variables que vienen en la petición, en el Código 3.17 se muestra cómo se almacenan estas variables.

```
def topologia(request):
    lineas_respuesta=[]
    #verificar es una peticion ajax
    if request.is_ajax():
        if request.method=='POST':

            #almacena las variables del diccionario de la peticion post

            _userName=request.POST.get('nombre','')
            _topologia=request.POST.get('topologia','')
            _hosts=request.POST.get('hosts','')
            _switchh=request.POST.get('switchh','')
            _controlador=request.POST.get('controlador','')
            _fanout=request.POST.get('fanout','')
            _depth=request.POST.get('deep','')
```

Código 3.17. Almacenar las variables que vienen en la petición

Con las variables que envió el cliente se puede llamar a las funciones que realizan la actividad que el cliente ha solicitado, esto se lo realiza mediante el análisis de las variables con las sentencias `if` y `elif`. En el Código 3.18 se presenta el código empleado cuando se ha solicitado iniciar el controlador POX.

```
#iniciar controlador pox
if _controlador=='pox':
    if _topologia=='simple':

        subprocess.call('sudo python controller.py pox simple '+_hosts+' > resultados.txt',shell=True)
    elif _topologia=='fan_out':
        subprocess.call('sudo python controller.py pox arbol '+_depth+' '+_fanout+' > resultados.txt',shell=True)
```

Código 3.18. Segmento de código para determinar el controlador y la topología

Este procedimiento se repite para los dos controladores restantes, en donde primero se determina el controlador y después la topología para implementar lo que el usuario ha solicitado.

3.2.3 CREACIÓN DE LA INFRAESTRUCTURA DE RED

Para la creación de la infraestructura de red se cargará una imagen en la nube, la imagen tiene instalado el simulador de SDN Mininet, y los controladores POX, RYU y PYRETIC. La infraestructura a ser implementada se crea mediante el software Mininet, en el cual se seleccionará el controlador de entre los tres

disponibles y permite realizar pruebas de conectividad para verificar su funcionamiento, y le permitirá al usuario usar la infraestructura levantada.

3.2.3.1 Mininet

Mininet ofrece la alternativa de crear la infraestructura SDN de una manera virtual, lo cual permite la creación de los dispositivos de red, el switch y el controlador en un solo computador. Mininet es un software que permite implementar redes SDN.

3.2.3.2 Creación de la topología

Para la creación de la infraestructura se debe ejecutar un comando en el simulador Mininet para especificar la topología que se va a implementar y los parámetros que requiere cada una de ellas, el controlador que va a ser utilizado y el switch.

En el Comando 3.3 se presenta el comando para la creación de una red SDN con topología simple con 4 *hosts*, en el que cada nodo de red tiene preestablecida una dirección MAC, con un controlador remoto y el switch Open vSwitch. El parámetro `--mac` indica que se asigne de manera aleatoria las direcciones MAC.

```
$ sudo mn --topo single,4 --mac --controller remote   \
  --switch ovsk
```

Comando 3.3. Creación de topología simple en Mininet

Con el Comando 3.4 se realiza la creación de una red con topología tipo árbol con parámetros *depth*=2, *fanout*=2. Se define la asignación de direcciones MAC de manera aleatoria, se selecciona el switch Open vSwitch y se define el controlador remoto.

```
$ sudo mn --topo single,4 --mac --controller remote   \
  --switch ovsk
```

Comando 3.4. Creación de topología tipo árbol en Mininet

3.2.3.3 Selección del controlador

Creada la topología, se debe especificar el controlador remoto, el switch espera conectarse a un controlador que se encuentre en *localhost* y en el puerto 6633. Para iniciar el controlador se debe hacer un cambio de directorio a la carpeta en donde se encuentra el controlador e iniciarlo, en el Comando 3.5 se especifica el procedimiento para iniciar el controlador POX.

```
$ cd /home/ubuntu/pox && ./pox.py log.level –DEBUG \
  forwarding.tutorial_l2_hub
```

Comando 3.5. Iniciar el controlador POX

En el caso de Ryu se realiza un proceso similar al descrito anteriormente, lo cual se muestra en el Comando 3.6.

```
$ cd /home/ubuntu/ryu && ./bin/ryu-manager –verbose \
  ryu/app/simple_switch_13.py
```

Comando 3.6. Inicializar RYU

Para la inicialización del controlador Pyretic se debe ejecutar el Comando 3.7.

```
$ cd /home/ubuntu/ pyretic && pyretic.py -m p0
  pyretic.modules.mac_learner
```

Comando 3.7. Inicializar Pyretic

3.2.3.4 Pruebas de conectividad

Una vez que se ha creado la topología y se ha inicializado el controlador correspondiente se pueden realizar pruebas de conectividad empleando el Comando 3.8. En este comando se deben seleccionar los *hosts* entre los cuales se desea realizar la prueba de conectividad, en este caso se va a seleccionar el host 1 (h1) y el host 2 (h2) que son dispositivos creados por Mininet.

```
mn > h1 ping h2
```

Comando 3.8. Prueba de Conectividad entre *hosts*

3.3 PRUEBAS DE FUNCIONAMIENTO

Las pruebas de funcionamiento consisten en verificar que todo el ambiente desarrollado esté funcionando de forma adecuada. Las pruebas fueron divididas en tres etapas, la primera consiste en verificar el correcto funcionamiento de la nube, la segunda verifica que se pueda crear la infraestructura SDN en la nube mediante comandos y la tercera confirma que se puede crear la infraestructura SDN en la nube mediante la aplicación web de una manera automática.

3.3.1 FUNCIONAMIENTO DE LA NUBE

Para confirmar el funcionamiento de la nube, inicialmente se debe comprobar que los componentes de los diferentes servicios se encuentren trabajando y habilitados, luego se puede revisar cuáles son las imágenes que están cargadas y las instancias con las que se cuenta en la nube, finalmente es necesario verificar el estado de los recursos que tiene la nube.

Para ratificar los resultados de las pruebas se va a presentar la información obtenida mediante línea de comando y mediante la interfaz gráfica.

3.3.1.1 Servicio de Cómputo

Para esta prueba se listan los componentes del servicio de cómputo, se puede ver que todos los servicios se encuentran trabajando, lo que permite afirmar que el servicio de cómputo ha sido configurado correctamente. En la Figura 3.12 se presenta el resultado obtenido mediante CLI con el Comando 3.9 y, en la Figura 3.13 el resultado obtenido mediante la interfaz web de Horizon.

```
$ nova service-list
```

Comando 3.9. Comando para listar los servicios de cómputo

```
controller@controller:~$ nova service-list
+----+------------------+------------+----------+---------+-------+----------------------------+----
-------------+
| Id | Binary           | Host       | Zone     | Status  | State | Updated_at                 | Dis
abled Reason |
+----+------------------+------------+----------+---------+-------+----------------------------+----
-------------+
| 1  | nova-cert        | controller | internal | enabled | up    | 2015-06-29T17:53:26.000000 | -
             |
| 2  | nova-consoleauth | controller | internal | enabled | up    | 2015-06-29T17:53:26.000000 | -
             |
| 3  | nova-scheduler   | controller | internal | enabled | up    | 2015-06-29T17:53:26.000000 | -
             |
| 4  | nova-conductor   | controller | internal | enabled | up    | 2015-06-29T17:53:26.000000 | -
             |
| 5  | nova-compute     | compute    | nova     | enabled | up    | 2015-06-29T17:53:25.000000 | Non
e            |
+----+------------------+------------+----------+---------+-------+----------------------------+----
-------------+
```

Figura 3.12. Componentes del servicio Nova obtenido mediante línea de comandos

Figura 3.13. Componentes del servicio nova obtenido mediante Horizon

Una vez que se ha verificado que los componentes están trabajando adecuadamente se presenta la lista de imágenes que han sido cargadas en la nube, la imagen que se está utilizando se denomina SDN-x86_64, en donde se tienen las herramientas necesarias para crear la infraestructura SDN. En la Figura 3.14 se presenta el resultado del Comando 3.10 ingresado mediante línea de comandos y en la Figura 3.15 el resultado obtenido mediante la interfaz web.

```
$ glance image-list
```

Comando 3.10. Listar imágenes cargadas en la nube

```
controller@controller:~$ glance image-list
+--------------------------------------+---------------------+-------------+------------------+-----
-------+--------+
| ID                                   | Name                | Disk Format | Container Format | Size
       | Status |
+--------------------------------------+---------------------+-------------+------------------+-----
-------+--------+
| 35475002-7c85-4fbb-aedd-8f616ac178a3 | cirros-0.3.3-x86_64 | qcow2       | bare             | 1320
0896   | active |
| 97bc4673-d8be-4ce1-ae1b-45dc39d5f2b1 | Minnet-x86_64       | vmdk        | bare             | 2913
796096 | active |
| 29f356b5-c542-4efd-b1ea-f3d276ae96b8 | SDN-x86_64          | vmdk        | bare             | 7500
660736 | active |
+--------------------------------------+---------------------+-------------+------------------+-----
-------+--------+
```

Figura 3.14. Lista de imágenes en la nube obtenida mediante línea de comandos

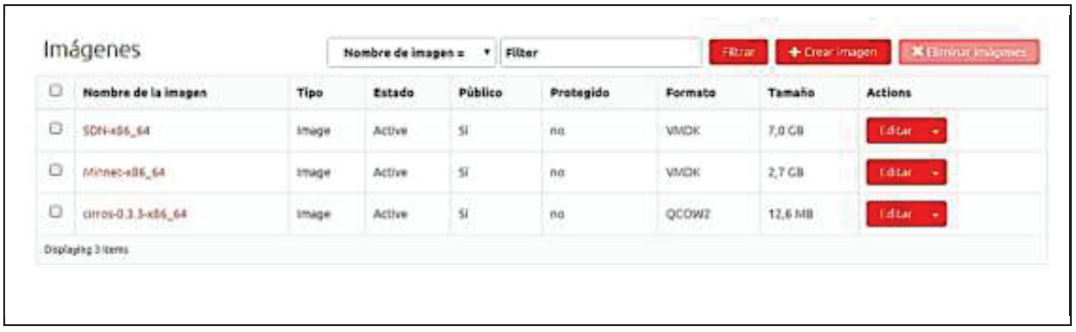

Figura 3.15. Lista de imágenes en la nube obtenida en Horizon

Para aprovechar los recursos disponibles en la nube se modificó el *flavor* `m1.medium`. En la Figura 3.16 se presentan los diferentes *flavors* disponibles y se puede observar que `m1.medium` cuenta con un ID distinto por los cambios realizados. El cambio que se realizó fue un incremento la memoria RAM para aprovechar el máximo valor disponible.

Figura 3.16. Lista de *flavors*

3.3.1.2 Servicio de Red Neutron

Para verificar el funcionamiento del servicio de red se van a listar los componentes en los cuales se puede verificar el estado y comprobar que se encuentran activos, demostrando que el servicio de red está correctamente configurado y funcionando. En la Figura 3.17 se presenta el resultado del Comando 3.11 ejecutado en la línea de comandos, el cual lista los componentes de Neutron, se puede ver que todo ellos se encuentran activos y funcionando, y en la Figura 3.18 se presenta los componentes del servicio Neutron obtenido mediante la interfaz web.

```
controller@controller:~$ neutron agent-list
+--------------------------------------+--------------------+---------+-------+----------------+----
-----------------------+
| id                                   | agent_type         | host    | alive | admin_state_up | bin
ary                    |
+--------------------------------------+--------------------+---------+-------+----------------+----
-----------------------+
| 0c47437a-9889-4cf5-960b-06d1e05be5fd | Metadata agent     | network | :-)   | True           | neu
tron-metadata-agent    |
| a33b9bd1-a383-4976-9ec9-740dc8abe1b6 | Open vSwitch agent | compute | :-)   | True           | neu
tron-openvswitch-agent |
| c9129f0a-baab-4630-9c3c-3df68eacf58d | L3 agent           | network | :-)   | True           | neu
tron-l3-agent          |
| cdafa206-22e6-4b03-a8bb-d8e0acab6bfd | Open vSwitch agent | network | :-)   | True           | neu
tron-openvswitch-agent |
| e4bf90fb-1f10-4ed4-852f-65f7f09a2d12 | DHCP agent         | network | :-)   | True           | neu
tron-dhcp-agent        |
+--------------------------------------+--------------------+---------+-------+----------------+----
-----------------------+
```

Figura 3.17. Listado de los componentes de Neutron obtenidos mediante línea de comandos

Figura 3.18. Listado de los componentes de Neutron obtenidos mediante Horizon

Comprobado el funcionamiento del servicio de red se pueden listar las redes que se han creado como parte de la red inicial, mediante línea de comandos se

enumeran las redes disponibles con el Comando 3.11 y sus resultados se muestran en la Figura 3.19. En la Figura 3.20 se exponen las redes disponibles que presenta la interfaz gráfica.

```
$ nova net-list
```

Comando 3.11. Listar las redes disponibles

```
controller@controller:~$ nova net-list
+--------------------------------------+----------+------+
| ID                                   | Label    | CIDR |
+--------------------------------------+----------+------+
| 59b0f727-dd03-4bae-a674-0edeee5022e2 | demo-net | None |
| d434063c-a9d8-486b-9e26-cadf257c4d94 | ext-net  | None |
+--------------------------------------+----------+------+
```

Figura 3.19. Lista de redes creadas para las instancias obtenidas mediante línea de comandos

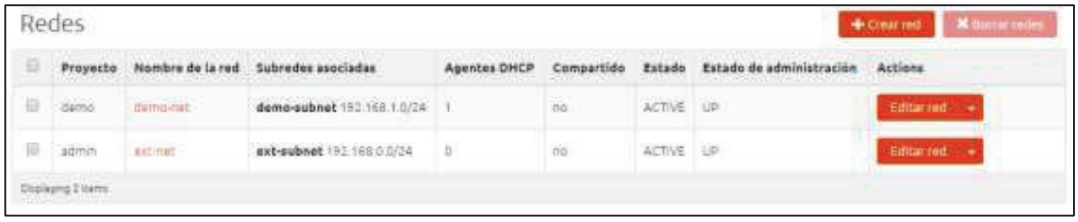

Figura 3.20. Lista de redes creadas obtenida en Horizon

En la Figura 3.21 se presenta el diagrama de red generado en Keystone para verificar la red inicial, en la topología de red se tiene la red externa `ext-net` (azul) y la red `demo-net` (amarillo) que es la red interna a la que se conectan las instancias creadas, entre estas dos redes se tiene un *router* que permite el tráfico entre ellas.

Figura 3.21. Diagrama de red

3.3.1.3 Servicio Dashboard Horizon

La Figura 3.13, Figura 3.15, Figura 3.18 y Figura 3.21 presentadas previamente demuestran que el servicio Horizon está funcionando correctamente, ya que se puede interactuar con el software OpenStack a través de la interfaz web. En la Figura 3.22 se muestra el inicio de sesión para acceder al servicio Horizon. Al iniciar sesión se indica el usuario y por ende, Horizon establece los permisos y privilegios que va a tener dicho usuario.

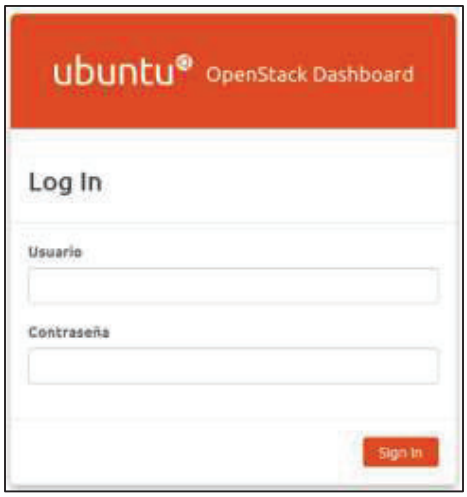

Figura 3.22. Pantalla de inicio de sesión de Horizon

3.3.1.4 Otros Servicios

Los servicios que no se han mencionado son aquellos transparentes para el usuario porque se encargan de los procesos necesarios para el funcionamiento de OpenStack. El Servicio de Identidad Keystone es transparente para el usuario porque es el que se encarga de asignar permisos, definir a qué recurso tiene acceso y mantener informados a los demás servicios la ubicación de los servicios registrados. La base de datos es muy importante para un adecuado funcionamiento, se ha comprobado que está instalada acertadamente porque OpenStack está funcionando. En la Figura 3.23 se presentan los servicios instalados y configurados para el correcto funcionamiento de OpenStack.

Figura 3.23. Otros Servicios Instalados y Configurados

3.3.1.4 Recursos computacionales de la nube

La Figura 3.24 es una representación gráfica de la utilización de los recursos computacionales disponibles en la nube, como se mencionó previamente se realizaron las configuraciones necesarias para aprovechar los recursos al máximo. Se está haciendo uso del único procesador virtual disponible y se está aprovechando la máxima capacidad de la memoria RAM.

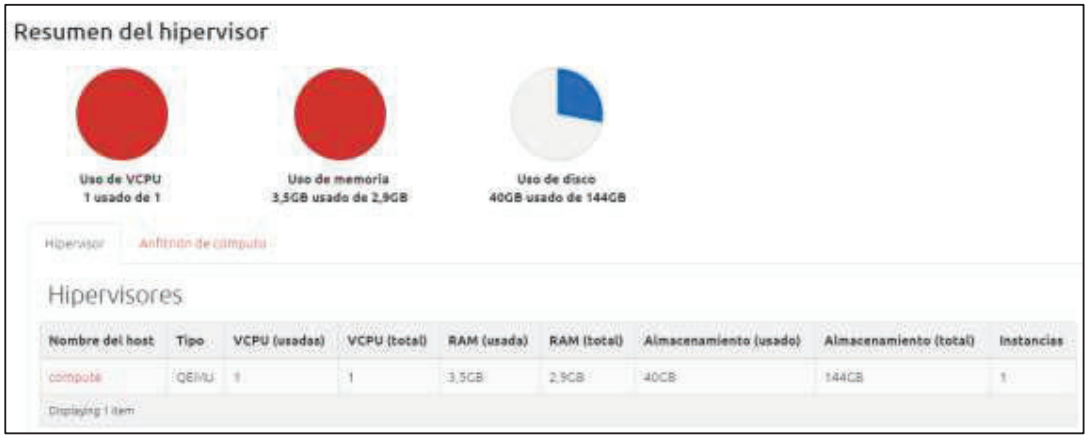

Figura 3.24. Recursos computacionales de la nube

3.3.2 INICIO DE LA INSTANCIA

Una vez que se ha verificado que la nube está funcionando adecuadamente, se inicia la instancia sobre la que se implementará la infraestructura SDN.

Para iniciar la instancia se cargó en la nube la imagen que está disponible en [73], la imagen está pre-configurada con el simulador Mininet, el controlador POX, controlador RYU y el controlador PYRETIC, son todas las herramientas necesarias para implementar la infraestructura SDN.

Con la imagen SDN se va a iniciar la instancia, para lo cual se necesita definir los parámetros computacionales, se selecciona el *flavor* `m1.medium`, la red a la que se va a conectar, que es `demo-net` y el grupo de seguridad al que pertenece, con todos estos parámetros se inicia la instancia. En la Figura 3.25 se tiene un resumen de Horizon y en la Figura 3.26 se observa el resultado mediante la línea de comando, se puede ver que la información es la misma y que la instancia está activa.

Figura 3.25. Información de la instancia en Horizon

```
controller@controller:~$ nova list
+--------------------------------------+-------------+--------+------------+-------------+----------
------------+
| ID                                   | Name        | Status | Task State | Power State | Networks
            |
+--------------------------------------+-------------+--------+------------+-------------+----------
------------+
| fc8ec616-7c72-4f21-ab22-1e5d7e8bf6a3 | SDN-Package | ACTIVE | -          | Running     | demo-net=
192.168.1.7 |
+--------------------------------------+-------------+--------+------------+-------------+----------
------------+
```

Figura 3.26. Información de la instancia mediante línea de comandos

Para acceder a la instancia se va a utilizar el acceso remoto VNC (*Virtual Network Computing*), para lo cual se debe generar un *token* en el servicio Keyston para Nova, como se muestra en la Figura 3.27.

```
controller@controller:~$ nova get-vnc-console SDN-Package novnc
+-------+-----------------------------------------------------------------------------------+
| Type  | Url                                                                               |
+-------+-----------------------------------------------------------------------------------+
| novnc | http://10.0.0.11:6080/vnc_auto.html?token=94e8f1ef-a809-4b0e-9926-faffb6821958 |
+-------+-----------------------------------------------------------------------------------+
```

Figura 3.27. Generación de *token* para acceder a la instancia mediante VNC

Se debe copiar el URL en el cual se encuentra el *token* generado en cualquier navegador web para poder acceder a la instancia. En la Figura 3.28 se muestra el acceso a la instancia mediante el explorador web usando el *token* obtenido.

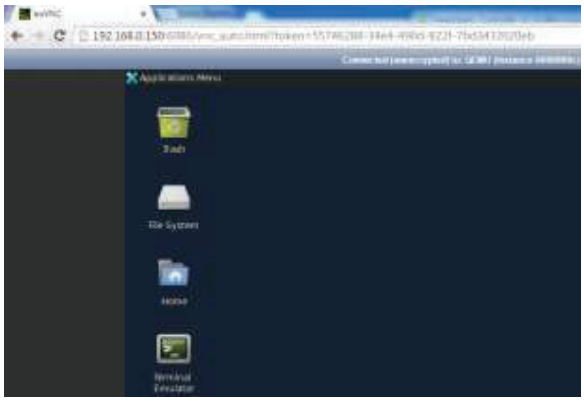

Figura 3.28. Acceso a la instancia en la nube

En la instancia se verifica que las herramientas para la creación de software estén funcionando adecuadamente, para lo cual se crea una topología simple en Mininet con 4 dispositivos y se especifica que va a ser utilizado un controlador remoto. En la Figura 3.29 se muestra el comando y los resultados de la creación de la infraestructura SDN indicada.

```
ubuntu@sdnhubvm:~[17:57]$ sudo mn --topo single,4 --mac --controller remote --switch ovsk
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Figura 3.29. Comando y resultados de la creación de la topología de red

Se inicia el controlador POX como se muestra en la Figura 3.30, se puede apreciar que en la salida de este comando, se indica que el switch se ha conectado, lo que indica que la infraestructura está lista para ser utilizada.

```
ubuntu@sdnhubvm:~/pox[18:03] (eel)$ cd /home/ubuntu/pox/ && ./pox.py log.level -
DEBUG forwarding.tutorial_l2_hub
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

Figura 3.30. Inicialización del controlador

Al realizar una prueba de conectividad para verificar que la infraestructura funciona correctamente, mediante el comando ping, se obtuvo un resultado exitoso, como se muestra en la Figura 3.31.

```
mininet> h1 ping -c 4 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=209 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=67.0 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=74.5 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=72.0 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 67.066/105.771/209.422/59.904 ms
```

Figura 3.31. Prueba de conectividad

En la Figura 3.32 se presenta la inicialización del controlador RYU, con la topología simple de 4 host en Mininet y en la Figura 3.33 se presenta la prueba de conectividad con 0% de paquetes perdidos.

```
ubuntu@sdnhubvm:~/ryu/bin[20:48] (master)$ cd /home/ubuntu/ryu/ && ./bin/ryu-man
ager --verbose ryu/app/simple_switch_13.py
loading app ryu/app/simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
```

Figura 3.32. Inicialización del controlador RYU

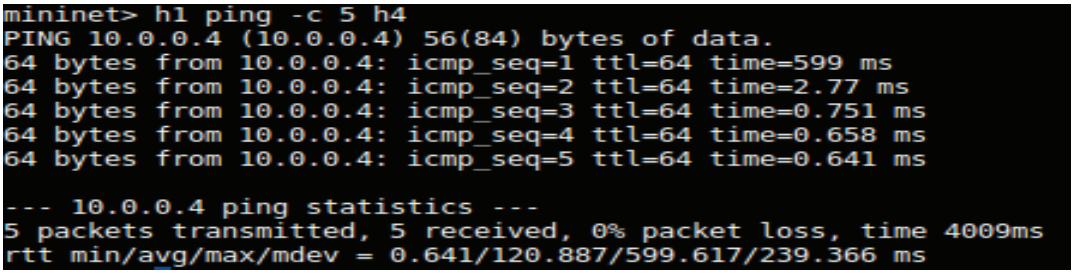

Figura 3.33. Prueba de conectividad con el controlador RYU

En la Figura 3.34 se presenta la inicialización del controlador Pyretic y en la Figura 3.35 se presenta el resultado de las pruebas de conectividad.

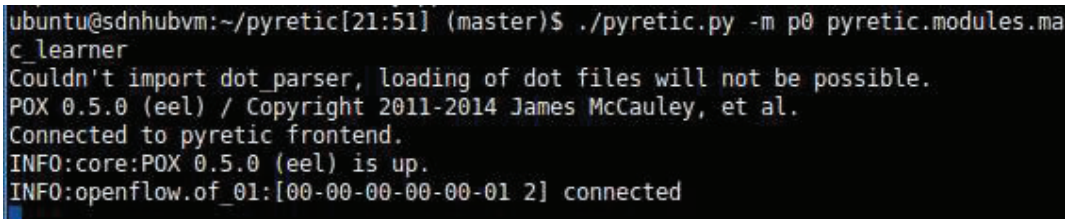

Figura 3.34. Se inicializa el controlador Pyretic

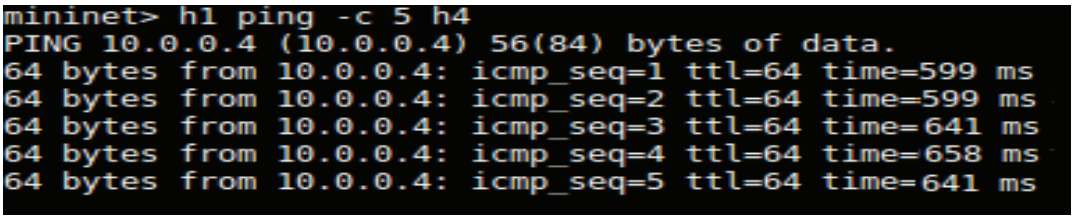

Figura 3.35. Resultado de las pruebas de conectividad

Las pruebas se repitieron varias ocasiones con el fin de verificar el funcionamiento de las diferentes topologías y los tres controladores que se desea implementar, mostrando un resultado exitoso, lo que indica que las herramientas para la creación de la infraestructura de red funcionan correctamente.

3.3.3 FUNCIONAMIENTO DE TODO EL PROTOTIPO

Para el adecuado funcionamiento del prototipo se debe verificar que la nube, la instancia en la nube y la aplicación web interactúen para crear la infraestructura de red.

Primero se verifica que el servidor web que está alojado en la instancia en la nube se encuentre levantado y permita acceder a la aplicación, en la Figura 3.36 se muestra que el servidor está levantado y se presentan los resultados de la primera petición en el servidor, y en la Figura 3.37 se presenta el resultado a la petición realizada por el cliente.

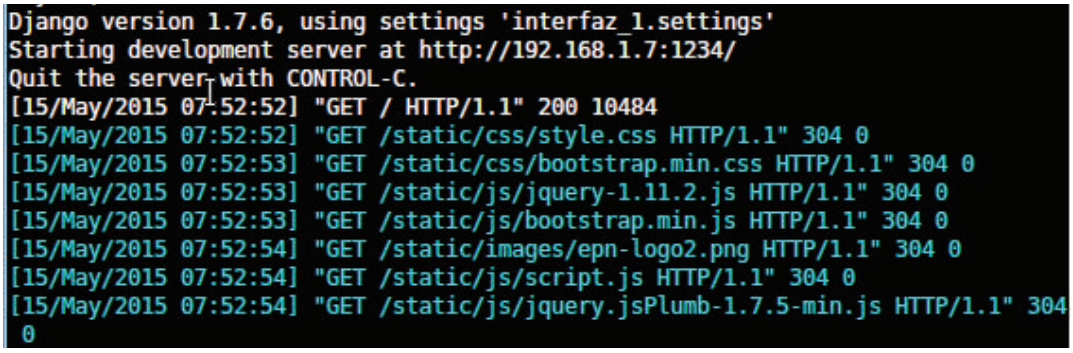

Figura 3.36. Resultado de la petición inicial en el servidor

Figura 3.37. Vista inicial de la aplicación web

Como prueba se iniciará una topología simple con tres *hosts* y el controlador Pyretic, en la Figura 3.38 se presenta el resultado que se muestra en el servidor cuando se crea el controlador y el resultado de la prueba de conectividad obtenido en el servidor. En la Figura 3.39 se presenta la topología junto con la respuesta obtenida en el lado del cliente; en la Figura 3.40 se presenta el resultado de la prueba de conectividad mostrado al cliente.

```
controller.py
pyretic_topo
simple
3
argumw
Unable to contact the remote controller at 127.0.0.1:6633
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.5.0 (eel) is up.
*** Ping: testing ping reachability
h1 -> INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
X X
h2 -> h1 h3
h3 -> h1 h2
*** Results: 33% dropped (4/6 received)
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
6633/tcp:            6186
```

Figura 3.38. Inicialización del controlador prueba de conectividad en el servidor

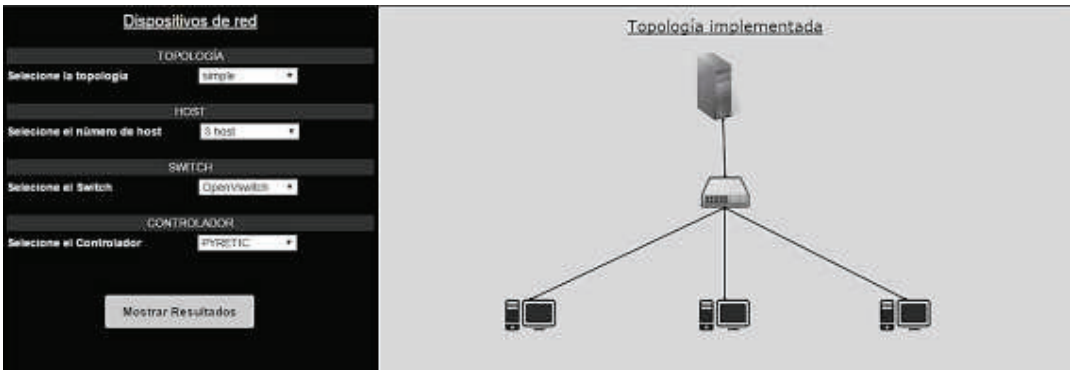

Figura 3.39. Resultado de la topología simple con Pyretic

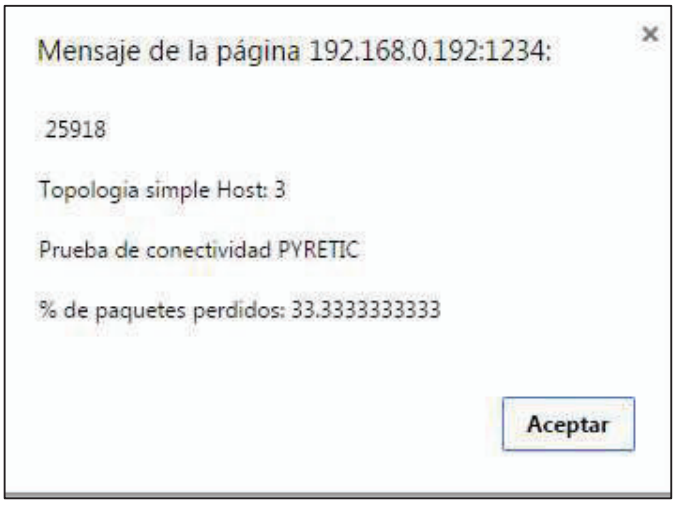

Figura 3.40. Resultados de prueba de conectividad

Para una segunda prueba, se crea una topología árbol con los parámetros *depth*=2 y *fanout*=2 y el controlador RYU, en la Figura 3.41 se presentan los resultados que se obtiene en el servidor cuando el controlador se ha inicializado, en la Figura 3.42 se presenta el resultado de la prueba de conectividad que muestra en el servidor, en la Figura 3.43 se presenta el resultado de la topología implementada y en la Figura 3.44 el resultado de la prueba de conectividad presentada en el lado del cliente.

```
ubuntu@sdnhubvm:~/prototipo/interfaz_1[17:22]$ sudo python controller.py ryu_top
o arbloading app ryu/app/simple_switch_13.py
olloading app ryu.controller.ofp_handler
instantiating app ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPHello
 2 2
connected socket:<eventlet.greenio.GreenSocket object at 0x7f0f44847350> address
:('127.0.0.1', 48015)
```

Figura 3.41. Inicialización del controlador RYU

```
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet in 1 b6:76:a8:c3:83:5b 76:36:e3:76:a8:a8 1
h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura 3.42. Resultado de la prueba de conectividad

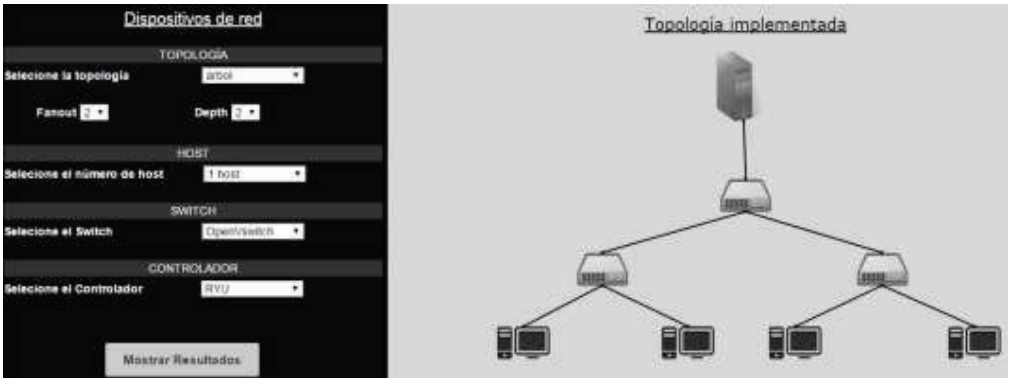

Figura 3.43. Resultado de la topología implementada

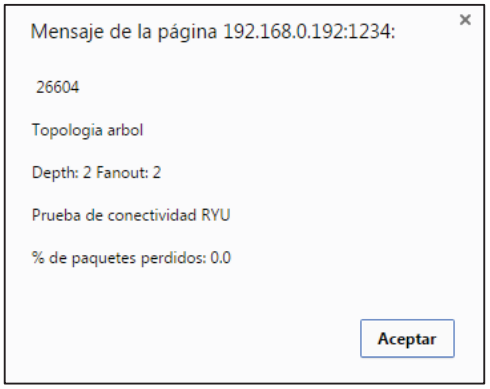

Figura 3.44. Resultado de la prueba de conectividad

La topología más grande que permite el prototipo considerando que está desarrollado para familiarizarse con los conceptos de SDN es la topología árbol con parámetros *dept*=3 y *fanout*=3, las diferentes alternativas que presenta la aplicación web son suficientes para conocer las diferentes topologías y cómo se conectan los dispositivos entre sí, permitiendo que el usuario se familiarice con la infraestructura SDN, además se consideró los recursos de hardware que se tiene disponibles para ofrecer un tiempo de espera aceptable para la creación de la topología, con esta topología el tiempo de espera es de 52 minutos, por lo tanto se decidió poner esta topología como la máxima, la aplicación permite la creación de esta topología pero el usuario debe esperar un tiempo de 5 minutos aproximadamente hasta que ésta se levante.

La prueba con la topología más grande que permite el prototipo, es la topología tipo árbol con los parámetros *depth*=3 y *fanout*=3 en donde se crean 27 *hosts*. En la Figura 3.45 se muestran los resultados que presenta el servidor cuando se

inicializa el controlador. En la Figura 3.46 se indica el resultado que se muestra en el servidor cuando el controlador establece la conexión con los 13 switches que se crean en la topología. En la Figura 3.47 se presenta el mensaje en el servidor cuando se realiza la prueba de conectividad y, en la Figura 3.48 aparece el resultado de la topología en el lado cliente.

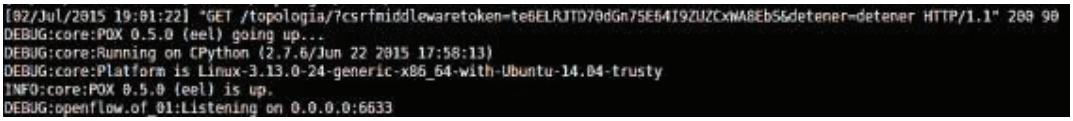

Figura 3.45. Mensaje de inicialización del controlador

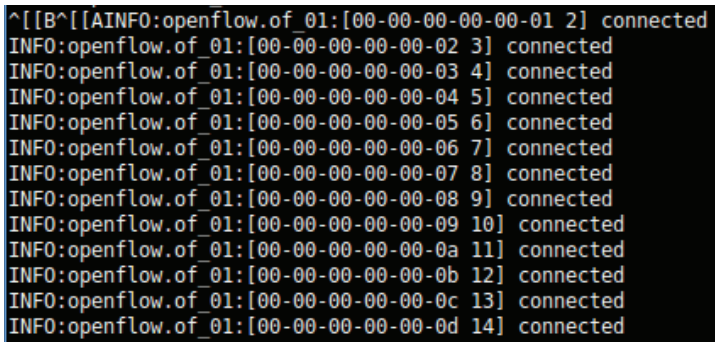

Figura 3.46. Mensaje en el servidor que se ha establecido conexión entre el switch y el controlador

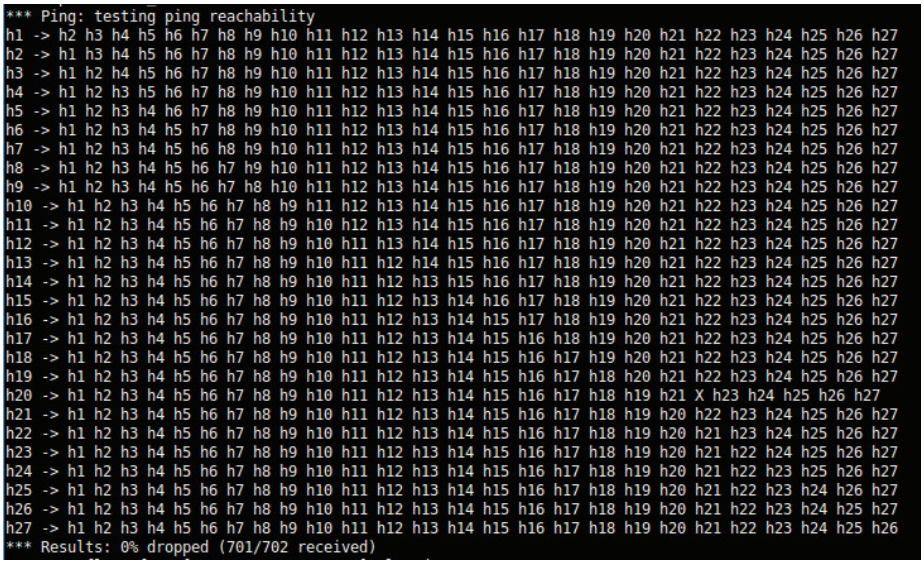

Figura 3.47. Mensaje en el servidor de la prueba de conectividad

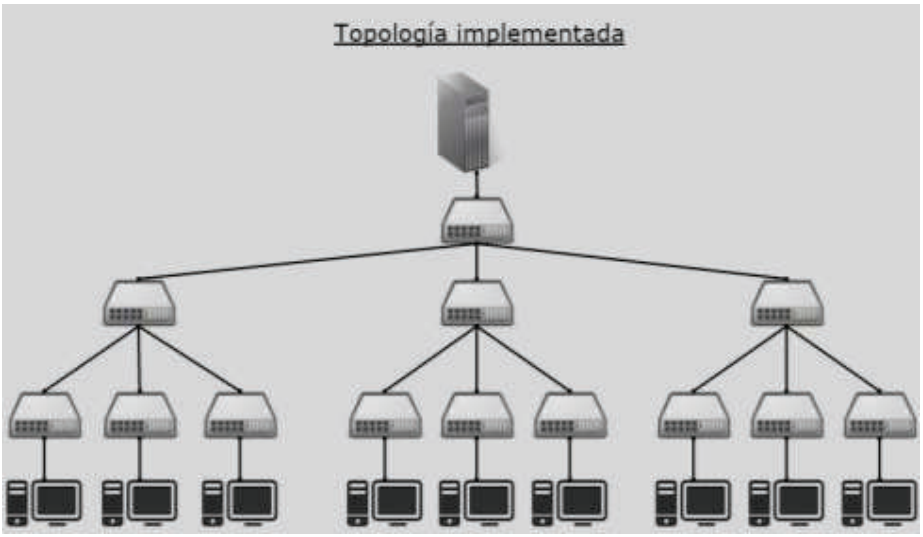

Figura 3.48. Resultado en el lado del cliente de la aplicación en la topología tipo árbol 3x3

Para documentar las pruebas se seleccionó diferentes topologías como se muestra en las Figuras 3.39, 3.43 y 3.48 con cada uno de los controladore; las pruebas presentaron resultados exitosos porque la nube, la instancia y la aplicación web están interactuando correctamente, lo que permite demostrar que el prototipo está funcionando según lo esperado. La inicialización del controlador se está realizando adecuadamente, pero cada uno presenta diferentes tiempos de inicialización, según la cantidad de recursos computacionales empleados. El controlador POX y RYU funciona bien con los recursos disponibles, al realizar la prueba de conectividad presentan un 0% de paquetes perdidos como se muestra en la Figura 3.44 y la Figura 3.47, mientras que el controlador Pyretic presenta un porcentaje de pérdidas de 33.33% como se demuestra en la Figura 3.40. Por su configuración por defecto, la cual establece que todos los paquetes, sean conocidos o desconocidos, vayan al controlador. Considerando los recursos disponibles de hardware este proceso toma mucho tiempo y ciertos paquetes se pierden. Con las pruebas realizadas se confirma que la infraestructura de red está funcionando correctamente.

CAPÍTULO IV

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Al finalizar el Proyecto de Titulación se cuenta con la implementación de una aplicación web que permite la creación de infraestructura SDN en la nube, para la cual se implementó una nube mediante el software OpenStack con un modelo multinodo. Se instaló una imagen con las herramientas necesarias y los controladores POX, RYU y Pyretic para la creación de Infraestructura SDN mediante una aplicación web que permite al usuario la creación automática de esta infraestructura.

- Se pudo determinar que la virtualización es una herramienta muy poderosa y de mucha utilidad en la actualidad. Para la creación de la nube mediante el modelo multinodo con nodo controlador, nodo de red y nodo de cómputo son necesarios tres servidores independientes, los cuales fueron creados como máquinas virtuales en un solo servidor físico, demostrando que se puede implementar infraestructura para virtualización sobre infraestructura virtualizada aprovechando los recursos.

- Los recursos de hardware disponibles para la implementación del presente Proyecto de Titulación no fueron suficientes. Para que el ambiente multi-nodo tenga el resultado esperado se necesitaba mayor capacidad de memoria RAM y de procesadores, lo que ocasionó una degradación en el desempeño de la instancia que se encuentra en la nube y en consecuencia, en la aplicación web; esto influyó en el tiempo de espera, porque la creación de los dispositivos SDN en la nube, superaba los cinco minutos. La instancia requiere mayor procesamiento dado que las acciones y peticiones cotidianas tales como abrir una carpeta o cambiarse de directorio toman un intervalo de tiempo considerable, mayor a diez segundos, y no digamos, las tareas que

requieren más procesamiento como crear la infraestructura SDN en el simulador Mininet y los controladores.

- Se comprobó la versatilidad que presenta Mininet, puesto que ofrece opciones pre configuradas para la implementación de infraestructura SDN, que pueden ser utilizadas por personas que están iniciándose en este campo. Los usuarios con mayores conocimientos tienen las herramientas para programar sus propias topologías, establecer reglas personalizadas en los controladores y configurar la red según sus requerimientos. Mininet es un simulador que puede ser usado con varios propósitos como investigación, desarrollo, pruebas, convirtiéndose en una herramienta de uso común en el campo de SDN.

- Se constata que la interfaz gráfica es una herramienta que permite al usuario interactuar con el software de una manera más sencilla, realizando ciertos procesos de manera transparente para el usuario, lo que implica que un usuario con mínimos conocimientos puede aprovechar las funcionalidades del software.

- Al implementar los dispositivos SDN en la nube se comprobó que es posible ofrecer infraestructura de red como servicio, ya que los dispositivos SDN son implementados sobre servidores o dispositivos de cómputo virtualizados que tienen acceso al Internet. Esto quiere decir que se puede establecer comunicación entre dos o más dispositivos y definir las políticas de tráfico de la red sin necesidad de dispositivos de red físicos.

- Se concluye que el software libre brinda libertad para usar el conocimiento, facilita el aprendizaje y proporciona herramientas ya probadas que permiten que el proceso de desarrollo sea más eficiente. Al ser un software que no tiene costo, los usuarios tienen la ventaja de trabajar en igualdad de condiciones y no depender de un presupuesto para definir el alcance de su investigación. La posibilidad de acceder a comunidades que comparten

conocimientos ayuda a encontrar respuestas a problemas e inquietudes de manera rápida y precisa.

- Durante la etapa de pruebas, el controlador Pyretic mostró un porcentaje mayor al 33% de pérdida de paquetes, el tiempo de respuesta del controlador es alto, considerando que por defecto la configuración indica que todos los paquetes deben ir al controlador para definir la acción a tomar, retardo que se lo atribuye a las limitaciones de hardware debido a que el controlador toma demasiado tiempo en determinar la acción que se debe tomar con dicho paquete causando el descarte del mismo.

- Durante la etapa de pruebas se pudo verificar que la implementación de la topología tipo árbol con los parámetros *depth*=3, *fanout*=3 y controlador POX tomó 52 minutos, pero en las pruebas de conectividad se obtuvo un 0% de pérdida de paquetes, lo que demostró que el prototipo funciona correctamente pero el tiempo que se toma en procesar los requerimientos y realizar las pruebas de conectividad es muy alto considerando que este procedimiento toma de 3 a 4 minutos.

- La creación de infraestructura SDN en la nube es la evolución de la tecnología en conectividad de red, porque se ofrecer dispositivos de red basados en software, los cuales permiten definir las políticas de tráfico mediante programación y la infraestructura SDN se encuentra virtualizada en su totalidad.

4.2 RECOMENDACIONES

- Al finalizar una topología se recomienda verificar que tanto el controlador como Mininet no dejen procesos ejecutándose, ya que estos generan errores al crear una nueva topología y lanzar un nuevo controlador porque los sockets que emplean se encuentra abiertos.

- Para desarrollar los scripts de creación de infraestructura automática se recomienda verificar con qué privilegios deben ser iniciados cada uno de los

controladores, ya que esto puede causar errores con diferentes mensajes dado que el usuario con el que se inició no tiene los permisos adecuados.

- Cuando se instala el software OpenStack en un entorno virtualizado se debe verificar que la tarjeta de red de la máquina física donde se encuentra el nodo de red esté en modo promiscuo para tener acceso al Internet desde las instancias.

- Se recomienda utilizar un entorno virtualizado cuando no se dispone del número suficiente de servidores para implementar el software OpenStack o cuando se tiene un dispositivo de hardware con altas prestaciones.

- Para familiarizarse con los principales conceptos de SDN, se recomienda el uso de Mininet, considerando la variedad de herramientas que ofrece.

- Cuando se usa software libre se tiene como fuente de información la documentación oficial que generalmente es muy completa y para la solución de errores se recomienda acceder a las comunidades creadas por los usuarios.

- Cuando se utiliza un entorno virtualizado se recomienda aprovechar las ventajas que éste presenta, por ejemplo, obtener imágenes instantáneas de los servidores sobre los que se está trabajando para evitar pérdidas del trabajo realizado, debido a una mala configuración o cambio.

- Para el desarrollo del contenido estático en el *frontend* de la aplicación web se utiliza el lenguaje HTML y se recomienda asignar un nombre afín con el componente para que su modificación, asignación de estilo o funcionalidad se realice de una manera más organizada.

REFERENCIAS BIBLIOGRÁFICAS

CAPÍTULO I

[1] The NIST Definition of Cloud Computnig, Septiembre 2011, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, (Consultado el 6 de Abril 2015)

[2] Cloud Computing Retos y Oportunidades, Mayo 2012, http://www.ontsi.red.es/ontsi/sites/default/files/1-_estudio_cloud_computing_retos_y_oportunidades_vdef.pdf (Consultado el 6 de Abril 2015)

[3] Red Hat What is Virtualization?, n.d, http://www.redhat.com/f/pdf/virtualization/gunner_virtual_paper2.pdf (Consultado el 7 de Abril del 2015)

[4] Virtualización de Servidores conceptos básicos, n.d, http://www.gonzalonazareno.org/cloud/material/IntroVirtualizacion.pdf (Consultado el 7 de Abril del 2015)

[5] SaaS Maturity Model according to Forrester, Agosto 2008, http://blogs.msdn.com/b/architectsrule/archive/2008/08/18/saas-maturity-model-according-to-forrester.aspx, (Consultado el 7 de Abril del 2015)

[6] Plataforma como servicio. Cómo aumentar la adopción de la nube ofreciendo a los desarrolladores la clave para un desarrollo que considere la nube, Agosto 2013, http://dialogoti.intel.com/sites/default/files/documents/10110442_overcoming barriers_whitepaper_v2f_dwc.pdf, (Consultado el 8 de Abril del 2015)

[7] ¿Qué es IaaS?, n.d, http://www.interoute.es/what-iaas, (Consultado el 8 de Abril del 2015)

[8] ¿Qué ES CLOUD COMPUTING? Recomendaciones para usuario, n.d, http://www.internetsano.gob.ar/archivos/Cloud_computing_usuarios.pdf, (Consultado el 8 de Abril del 2015)

[9] Evaluation of Different Hypervisors Performance in the Private Cloud with SIGAR Framework, Enero 2014, http://thesai.org/Downloads/Volume5No2/Paper_10-Evaluation_of_Different_Hypervisors_Performance_in_the_Private_Cloud_with_SIGAR_Framework.pdf, (Consultado el 9 de Abril del 2015)

[10] OPENSTACK INSTALLATION GUIDE FOR UBUNTU 14.04 JUNO, Octubre 2014, http://docs.openstack.org/juno/install-guide/install/apt/content/ch_overview.html, (Consultado 10 de Abril del 2015)

[11] Arquitectura conceptual y tecnologías asociadas, Marzo 2014 https://code.google.com/p/uda/wiki/Arquitectura, (Consultado 13 Abril del 2015)

[12] OPENSTACK CLOUD ADMINISTRATOR GUIDE, n.d, http://docs.openstack.org/admin-guide-cloud/content/compute-service.html, (Consultado el 13 de Abril del 2015)

[13] Understanding the Virtualization Market, Agosto 2102, http://www.datacenterknowledge.com/archives/2012/08/01/hypervisor-101-a-look-hypervisor-market/, (Consultado el 14 de Abril del 2015)

[14] Understanding the cloud computing stack SaaS, PaaS, IaaS, Rackspace hosting, Octubre 2013, http://www.rackspace.com/knowledge_center/whitepaper/understanding-the-cloud-computing-stack-saas-paas-iaas, (Consultado el 14 de Abril del 2015)

[15] Middleware,n.d, http://www.pcmag.com/encyclopedia/term/47013/middleware, (Consultado 15 de Abril del 2015)

[16] Certificados X.509 y la autoridad de certificación, n.d, http://www.zeroshell.net/es/x509details/, (Consultado el 15 de Abril del 2015)

[17] RESTful Web Services: A Tutorial, Septiembre 2014, http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069, (Consultado el 15 de Abril del 2015)

[18] VNC, n.d, http://www.pcmag.com/encyclopedia/term/57380/vnc, (Consultado el 15 de Abril del 2015)

[19] Understanding Metadata, n.d, http://www.niso.org/publications/press/UnderstandingMetadata.pdf, (Consultado el 17 de Abril del 2015)

[20] How To Write a UNIX Daemon, n.d, http://cjh.polyplex.org/software/daemon.pdf, (Consultado el 17 de Abril del 2015)

[21] Spice for Newbies, Enero 2009, http://www.spice-space.org/docs/spice_for_newbies.pdf, , (Consultado el 17 de Abril del 2015)

[22] Understanding Full Virtualization, Paravirtualization and Hardware Assist, n.d, http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf, (Consultado el 17 de Abril del 2015)

[23] CPU, Enero 2002, http://www.econ.uba.ar/www/departamentos/sistemas/plan97/tecn_informa c/briano/seoane/tp/2002_1/UnidadCentralProceso.htm, (Consultado el 17 de Abril del 2015)

[24] Introducción a la Informática, n.d, http://publiespe.espe.edu.ec/librosvirtuales/informatica-basica/informatica-basica/informatica-basica01.pdf, (Consultado el 20 de Abril del 2015)

[25] Virtualization Technology in Cloud Computing Environment, Marzo 2013 http://www.ijetae.com/files/Volume3Issue3/IJETAE_0313_131.pdf, (Consultado el 20 de Abril del 2015)

[26] Extensión de la nube privada empresarial de Intel con plataforma como servicio. IT de Intel, Junio 2012, http://www.intel.com/content/www/us/en/it-management/intel-it-best-practices/extending-intels-enterprise-private-cloud-with-platform-as-a-service.html, (Consultado el 20 de Abril del 2015)

[27] VMware ESX and VMware ESXi, Junio 2015, http://www.vmware.com/files/pdf/VMware-ESX-and-VMware-ESXi-DS-EN.pdf, (Consultado el 21 de Abril del 2015)

[28] ProLiant Servers, n.d http://www8.hp.com/us/en/products/proliant-servers/product-detail.html?oid=6399894&jumpid=reg_r1002_usen_c-001_title_r0001#!tab=specs, (Consultado el 21 de Abril del 2015)

[29] Secure Database, n.d https://www.digitalocean.com/community/tutorials/how-to-secure-mysql-and-mariadb-databases-in-a-linux-vps, (Consultado el 21 de Abril del 2015)

[30] OpneStack Notificarions, n.d, http://docs.openstack.org/developer/glance/notifications.html, (Consultado el 22 de Abril del 2015)

[31] Flat Network, n.d, https://developer.rackspace.com/blog/neutron-networking-simple-flat-network/, (Consultado el 22 de Abril del 2015)

[32] n.d ,http://man7.org/linux/man-pages/man8/ip-netns.8.html, (Consultado el 23 de Abril del 2015)

[33] http://en.linuxreviews.org/Defunct_process, (Consultado el 23 de Abril del 2015)

[34] http://www.jmir.org/article/viewFile/1867/1/10659, (Consultado el 4 de Mayo del 2015)

[35] https://www.openstack.org/assets/welcome-guide/OpenStackWelcomeGuide.pdf, (Consultado el 4 de Mayo del 2015)

[36] http://en.linuxreviews.org/Defunct_process, (Consultado el 4 de Mayo del 2015)

CAPÍTULO II

[37] ML2, n.d, https://wiki.openstack.org/wiki/Neutron/ML2, (Consultado el 5 de Mayo del 2015)

[38] http://www.manning.com/videla/, (Consultado el 5 de Mayo del 2015)

[39] AMQP, n,d, https://www.amqp.org/about/what, (Consultado el 6 de Mayo del 2015)

[40] QPID, n.d, https://qpid.apache.org/, (Consultado el 6 de Mayo del 2015)

[41] 0MQ, n.d, http://zeromq.org/intro:read-the-manual, (Consultado el 6 de Mayo del 2015)

[42] IptablesHowTo, n.d, https://help.ubuntu.com/community/IptablesHowTo, (Consultado el 6 de Mayo del 2015)

[43] OpenRC, n.d, http://docs.openstack.org/developer/devstack/openrc.html, (Consultado el 7 de Mayo del 2015)

[44] Framework, n.d, http://techterms.com/definition/framework, (Consultado el 7 de Mayo del 2015)

[45] Open vSwitch, n.d, http://openvswitch.org/, (Consultado el 7 de Mayo del 2015)

[46] Generic Receive Offload, n.d, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/6.0_Release_Notes/networking.html, (Consultado el 7 de Mayo del 2015)

[47] NAT, n.d, http://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/26704-nat-faq-00.html, (Consultado el 8 de Mayo del 2015)

[48] Memecached, n,d, http://memcached.org/, (Consultado el 8 de Mayo del 2015)

[49] Jumbo Frames, n.d, http://www.ethernetalliance.org/wp-content/uploads/2011/10/EA-Ethernet-Jumbo-Frames-v0-1.pdf, (Consultado el 8 de Mayo del 2015)

[50] VLAN, n.d, http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12- 2SX/configuration/guide/book/vlans.html, (Consultado el 11 de Mayo del 2015)

[51] Network Bridge, n.d, http://windows.microsoft.com/is-is/windows/create-network-bridge#1TC=windows-7, (Consultado el 11 de Mayo del 2015)

[52] NAT, n.d, http://docs.openstack.org/networking-guide/intro_network_address_translation.html, (Consultado el 11 de Mayo del 2015)

[53] Kernel, n.d, http://www.linfo.org/kernel.html, (Consultado el 12 de Mayo del 2015)

[54] Server Specification, n.d, http://www8.hp.com/us/en/products/proliant-servers/product-detail.html?oid=6399894&jumpid=reg_r1002_usen_c-001_title_r0001#!tab=specs, (Consultado el 12 de Mayo del 2015)

[55] http://kvz.io/blog/2007/07/29/schedule-tasks-on-linux-using-crontab/, (Consultado el 12 de Mayo del 2015)

[56] SSH Keygen, n.d, http://linux.die.net/man/1/ssh-keygen, (Consultado el 12 de Mayo del 2015)

CAPÍTULO III

[57] Cross Site Request, Octubre 2015, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF), (Consultado el 15 de Mayo del 2015)

[58] Clickjacking, n.d, https://www.owasp.org/index.php/Clickjacking, (Consultado el 15 de Mayo del 2015)

[59] Python, n.d, http://www.tutorialspoint.com/python/python_overview.htm, (Consultado el 15 de Mayo del 2015)

[60] Prompt, n.d, http://www.computerhope.com/jargon/p/prompt.htm, (Consultado el 15 de Mayo del 2015)

[61] CSS, n.d, https://librosweb.es/libro/css/capitulo_1.html, (Consultado el 16 de Mayo del 2015)

[62] RYU, n.d, http://osrg.github.io/ryu/, (Consultado el 16 de Mayo del 2015)

[63] Desarrollo de Software basado en Componentes, n.d, https://msdn.microsoft.com/es-es/library/bb972268.aspx, (Consultado el 16 de Mayo del 2015)

[64] http://www.puyb.net/download/djangobook/res.pdf, (Consultado el 16 de Mayo del 2015)

[65] MVC, n.d, http://www.academia.edu/5217432/El_patr%C3%B3n_de_dise%C3%B1o_Modelo-Vista-Controlador_MVC_y_su_implementaci%C3%B3n_en_Java_Swing, (Consultado el 17 de Mayo del 2015)

[66] MVC, n.d, http://www.create.ucsb.edu/~stp/PostScript/mvc.pdf, (Consultado el 17 de Mayo del 2015)

[67] Application Domains, n.d, https://msdn.microsoft.com/en-us/library/2bh4z9hs%28v=vs.110%29.aspx, (Consultado el 17 de Mayo del 2015)

[68] Model-View-Controller, n.d, https://msdn.microsoft.com/en-us/library/ff649643.aspx, (Consultado el 22 de Mayo del 2015)

[69] Django FAQ: General, n.d, https://docs.djangoproject.com/en/1.4/faq/general/, (Consultado el 22 de Mayo del 2015)

[70] Django Overview, n.d, https://www.djangoproject.com/start/overview/, (Consultado el 22 de Mayo del 2015)

[71] Inyección de código SQL, n.d, https://technet.microsoft.com/es-es/library/ms161953(v=sql.105).aspx, (Consultado el 22 de Mayo del 2015), (Consultado el 22 de Mayo del 2015)

[72] Cross-site Scripting, Abril 2014, https://www.owasp.org/index.php/Cross-site_Scripting_(XSS), (Consultado el 22 de Mayo del 2015)

[73] Imagen Máquina Virtual, http://sdnhub.org/tutorials/sdn-tutorial-vm/, (Consultado el 22 de Junio del 2015)

ANEXOS

Los anexos se encuentran en el CD adjunto.